МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Розробка програмного забезпечення»

Група: 4РП-07

Дипломний проект

здобувача освіти денної форми навчання
РП.07.17.000.ДП

ПАВЛЮКА ОЛЕКСІЯ ПАВЛОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітньо-професійна програма: «**Розробка програмного забезпечення»**

Група: **4РП-07**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка шахової гри на програмному рушії Unity

Проектний матеріал складається з пояснювальної записки на 79 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах)

Дипломник _____ (Павлюк О.П.)

Керівник _____ (Джабраілов Д.В.)

Консультанти:

з економічного розділу _____ (Іванченков В.С.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «17» 06 2024 р. Протокол ЕК № 1

Оцінка ЕК 5 (відмінно) / 98

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *121 «Інженерія програмного забезпечення»*

Освітньо-професійна програма *«Розробка програмного забезпечення»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"15" 09 2024 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Павлюку Олексію Павловичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка шахової гри на програмному рушії Unity*

затверджена наказом по коледжу від "02" 11 2023 р. № 244-А2-ОД

2. Термін здачі закінченого проекту 10.06.2024

3. Вихідні данні до проекту *Використання програмного рушія Unity; Використання мови програмування C# та бібліотек Unity для розробки гри; Використання принципів ООП у проектуванні та розробці ігрових проектів; Мають бути використані класичні правила шахів; Гра повинна реалізовувати ігровий процес для двох гравців за одним ігровим місцем; Гра повинна мати мінімальний інтерфейс.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Аналіз існуючих ігрових рішень; Огляд та обрання правил шахових ігор для проекту; Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом; Формування концепту ігрового процесу шахової гри; Проектування основних елементів ігрового процесу та принципи їх роботи; Реалізація основних елементів ігрового процесу; Тестування працездатності ігрових елементів.

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

Існуючі шахові ігри, алгоритми та їх призначення; Огляд шахових правил та причини обрання класичних шахів; Особливості програмного рушія Unity та причини його вибору для розробки проекту; Особливості ігрових механік розроблюємого проекту; Огляд основних елементів ігрового процесу; Принцип роботи основних елементів ігрового процесу; Етапи реалізації основних елементів ігрового процесу; Хід тестування гри; Скріншоти гри.

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Джабраілов Д.В.*		
Економічний розділ	*Іванченков В.С.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник *Джабраілов Д.В.* _____
(підпис)

Завдання прийняв до виконання *Павлюк О.П.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Вступ. Постановка мети та задач проектування*	*20.05.2024*	Виконав
2	*Аналіз існуючих ігрових рішень, шахових правил*	*23.05.2024*	Виконав
3	*Аналітичний огляд та вибір програмного рушія*	*25.05.2024*	Виконав
4	*Формування концепту ігрового процесу гри*	*28.05.2024*	Виконав
5	*Проектування основних елементів ігрового процесу*	*30.05.2024*	Виконав
6	*Проектування графічної частини гри*	*01.06.2024*	Виконав
7	*Реалізація графічної частини гри*	*03.06.2024*	Виконав
8	*Реалізація основних елементів ігрового процесу*	*05.06.2024*	Виконав
9	*Імплементація та відлагодження ігровіх елементів*	*07.06.2024*	Виконав
10	*Тестування працездатності елементів гри*	*09.06.2024*	Виконав
11	*Виправлення виявлених помилок*	*10.06.2024*	Виконав
12	*Аналіз результатів, підготовка слайдів презентації*	*11.06.2024*	Виконав
13	*Економічні розрахунки та питання з охорони праці*	*12.06.2024*	Виконав
14	*Підготовка графічної частини проекту*	*13.06.2024*	Виконав
15	*Підготовка проекту до захисту та тестування ПП*	*14.06.2024*	Виконав

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 07. 17 000. 00 ДП	Дипломний проект		
А4			РП 07. 17 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	РП 07. 17 000. 00 ДП		
Розробив		Павлюк О.П.		10.06	Розробка шахової гри на програмному рушії Unity	Літ.	Аркуш 4 / Аркушів 79
Перевірив		Джабраілов Д.В.		10.06			
Н. контр.		Петрашова В.І.		10.06		ВСП "ОТФК ОНТУ" гр.4РП-07	
Затверд.		Кривченко Ю.В.		10.06			

ЗМІСТ

ВСТУП

Шахи досить давня гра, яка ще в часи шостого століття нашої ери. Навіть з таких старих часів люди грали в шахи. Ця гра не тільки дає змогу провести час, а ще й дозволяє розвинутися в інтелектуальному плані. Для деяких це просто привід зустрітися з друзями та просто провести час. Також слід зазначити, що для деяких людей це не просто дозвілля, це повноцінна професійна діяльність, яка визначає людей на турнірах, або у тренерах, або просто як професійних гравців.

Темою цього дипломного проекту є «Розробка шахової гри на програмному рушії Unity». Шахи були обрані через те, що вони достатньо цікаві для розробки та дають можливість провести час за розвитком як для людини, яка будить розробляти проект так і для гравця, який буде просто грати у нього. Платформа Unity була обрана як програмний рушій, через те, що вона є найбільш простою та універсальною. Вона дає змогу створити проект як і для персональних комп'ютерів, так і для мобільних пристроїв, які значно полегшують гру і дають змогу грати в неї тоді, коли люди мають доступ по свого телефона. Гра повинна бути простою для виконання на більшості девайсів. Вона також повинна в повній мірі давати можливість грати в шахи навіть якщо, можливості грати реальними дошкою та фігурами немає.

В подальшому проект може бути розвинений такими доповненнями як доданням комп'ютерів з різними видами складності. В майбутньому, якщо проект набуде популярності можна буде додати гру через мережу з іншими гравцями і на основі цього зробити рейтинг гравців, які показали найкращі результати. Можна буде додати різні шахові варіації, такі як: шахи навпаки, шахи на трьох або чотирьох і інші.

Розробка проекту на Unity дає змогу набути нові навички та закріпити вже освоєні. Розробка на Unity є широко популярною і досвід у проектах та потенційне портфоліо дає змогу більш швидко розвиватися в цьому напрямку. Саме тому, я вважаю що тема дипломного проекту є актуальною і корисною.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз існуючих ігрових рішень

1.1.1 Огляд програми Chess.com

Розглянемо готові рішення з ринку шахових ігор. Я розгляну лише три шахові гри, але їх існує набагато більше. Почнемо з першої і найбільш популярної шахової гри – Chess.com. Chess.com стала не просто грою, а найбільшою платформою для гри, вивчення або просто слідкуванням за шахами. Ця платформа має велику кількість гравців з різних куточків світу, так само як і можливість тренуватися різними методами, як показано на рисунку 1.1:

Рисунок 1.1. Скріншот можливостей навчання на платформі Chess.com

Також є можливість слідкувати за різними подіями не виходячи з сайту. Ці і інші можливості платформи показано на рисунку 1.2. Ще є мобільний додаток, що говорить про кросплатформену підтримку цієї платформи. Chess.com має можливість аналізувати партії за допомогою їх шахової програми для аналізу – Stockfish 3.0. Chess.com дає змогу грати в різні види шахів, такі як: швидкі шахи – на партію дається лише 10 хвилин, бліц – на партію дається лише 3 хвилини, шахи на чотирьох – в одній грі є чотири ігроки. Мульти-дошкові шахи – в грі є 2 дошки, які пересікаються в середині по вертикалі. Також є можливості грати в шахи за різними правилами. Все це робить Chess.com не просто грою в шахи, а

найбільшою і найпопулярнішою платформою з шахів.

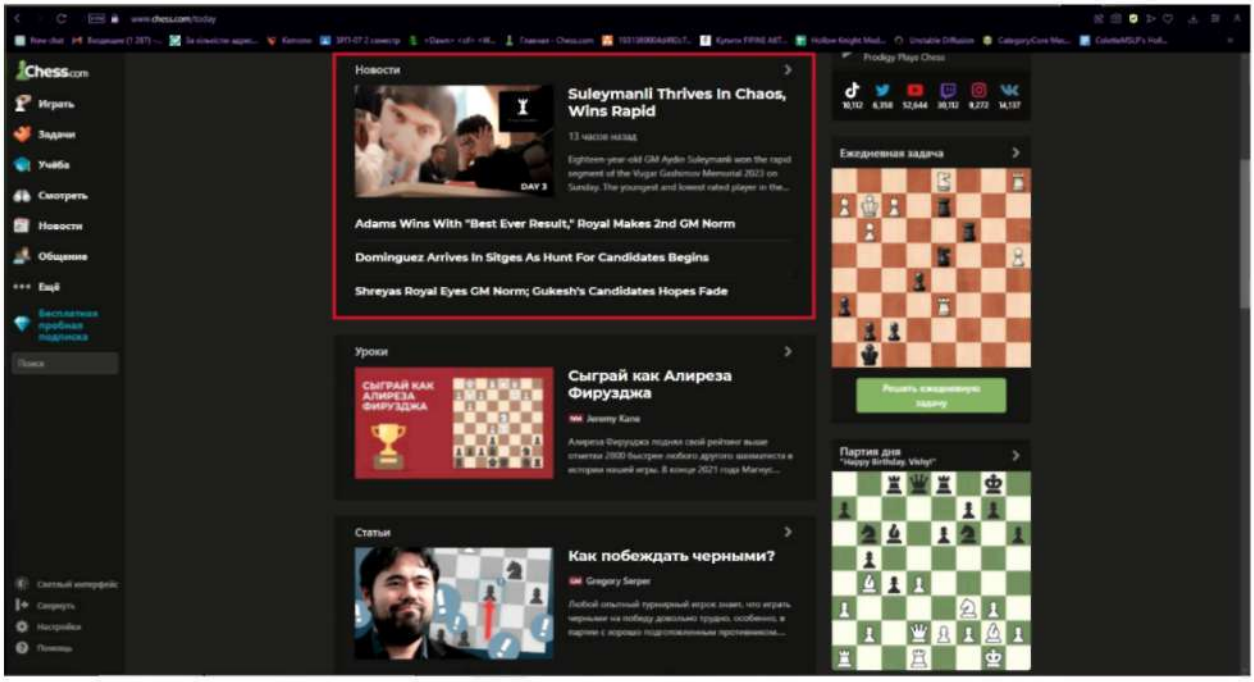


Рисунок 1.2. Скріншот головної сторінки сайту

1.1.2 Огляд програми Play Magnus

Наступна програма для аналізу – Play Magnus. Це мобільний додаток, який спеціально розроблений для гри в шахи на телефоні. Він спеціалізується конкретно на грі, але дає змогу вирішувати задачі та тренувати різні аспекти гри.

Особливістю гри є те, що в ній рівні складності визначають приблизний вік опонента, який є чемпіоном світу Магнусом Карлсеном в різному віці, як показано на рисунку 1.3:

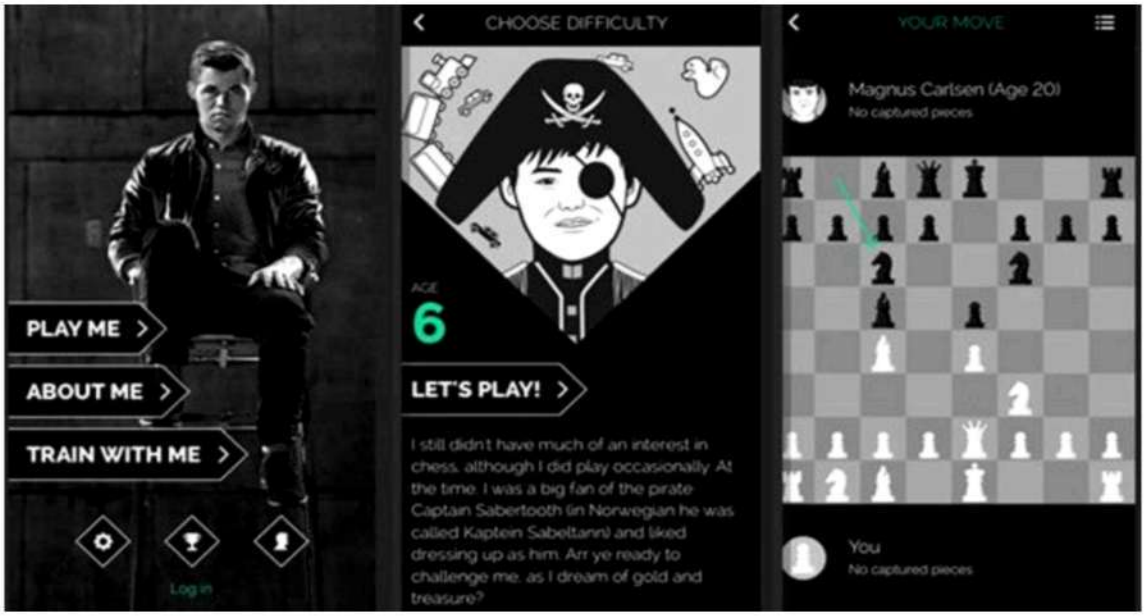


Рисунок 1.3. Скріншоти з гри Play Magnus

1.1.3 Огляд програми Lichess

Lichess — це популярна безкоштовна онлайн-платформа для гри в шахи створена французьким програмістом Тібо Дюплесси. Від моменту свого запуску у 2010 році Lichess швидко стала однією з найпопулярніших шахових платформ завдяки своєму простому інтерфейсу, широкому функціоналу та відкритому вихідному коду.

Так саме як і на платформі Chess.com на Lichess можна грати в різні варіації шахів, гра проти комп'ютерів та інших людей, аналіз партій та деякі соціальні функції. Але основною і відмінною функцією є можливість грати велику кількість годин та навіть днів. Проводилися турніри, які були між командами гравців, які не могли комунікувати між собою і робили 1 хід за одну годину.

Трохи застарілий дизайн та відсутність деяких функцій, що показано на рисунку 1.4, які є у конкурентних платформ є фактором, може відлякати нових користувачів від цієї платформи.

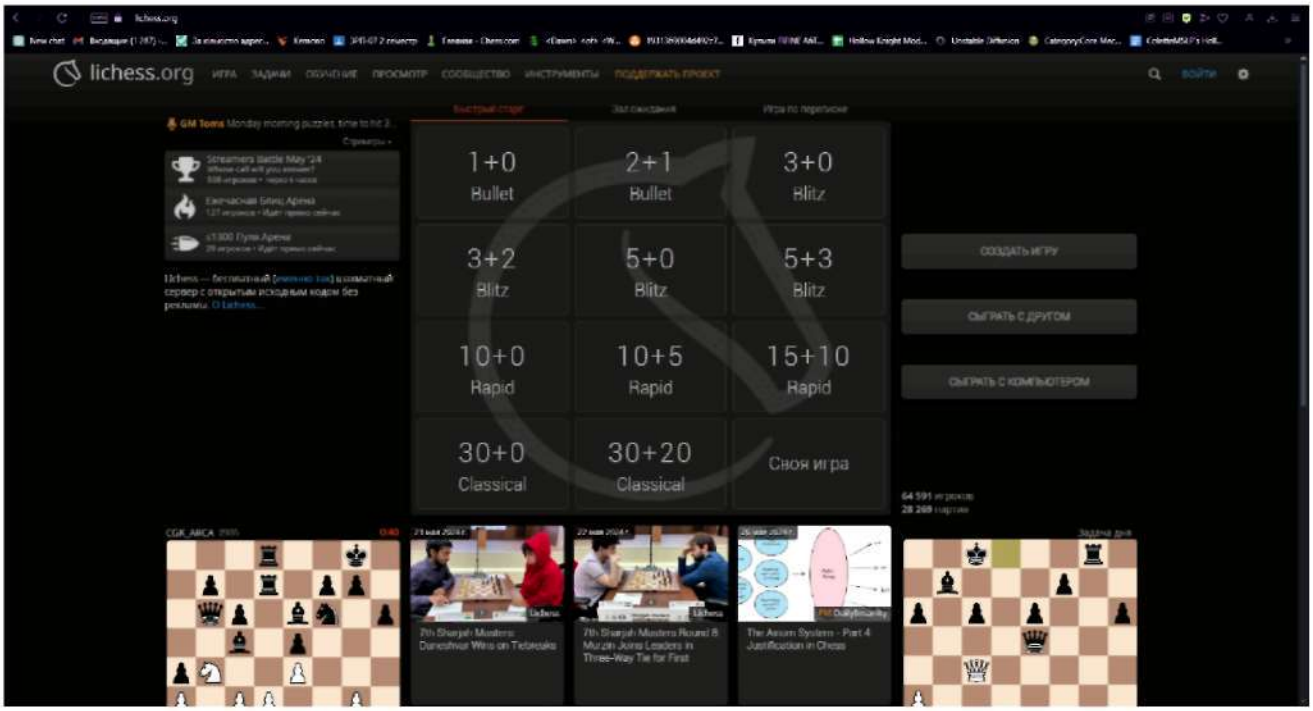


Рисунок 1.4. Скріншот головної сторінки сайту Lichess

Як можна побачити в прикладах в кожної шахової гри є свої переваги та недоліки, за допомогою цього дослідження, маємо змогу зробити висновки, які дадуть змогу використовувати лише кращі аспекти з цих програм, а саме: простий, але вдалий дизайн, кросплатформенність та можливість грати з мінімальними

вимогами.

1.2 Огляд та обрання правил шахових ігор для проекту

1.2.1 Огляд існуючих правил

Перейдемо до огляду та обрання правил для шахів. У шахах існує багато різновидів правил такі як: стандартні, шахи Фішера, анти-шахи, чотири гравці, тощо.

Розберемо кожні з наведених правил. Шахи на чотирьох гравців є широко розповсюдженою і більш-менш популярною грою. Ці шахи слідують стандартним шаховим правилам, але в них одночасно грають чотири гравці. Всі фігури всіх гравців розташовані на одній дошці, як показано на рисунку 1.5:

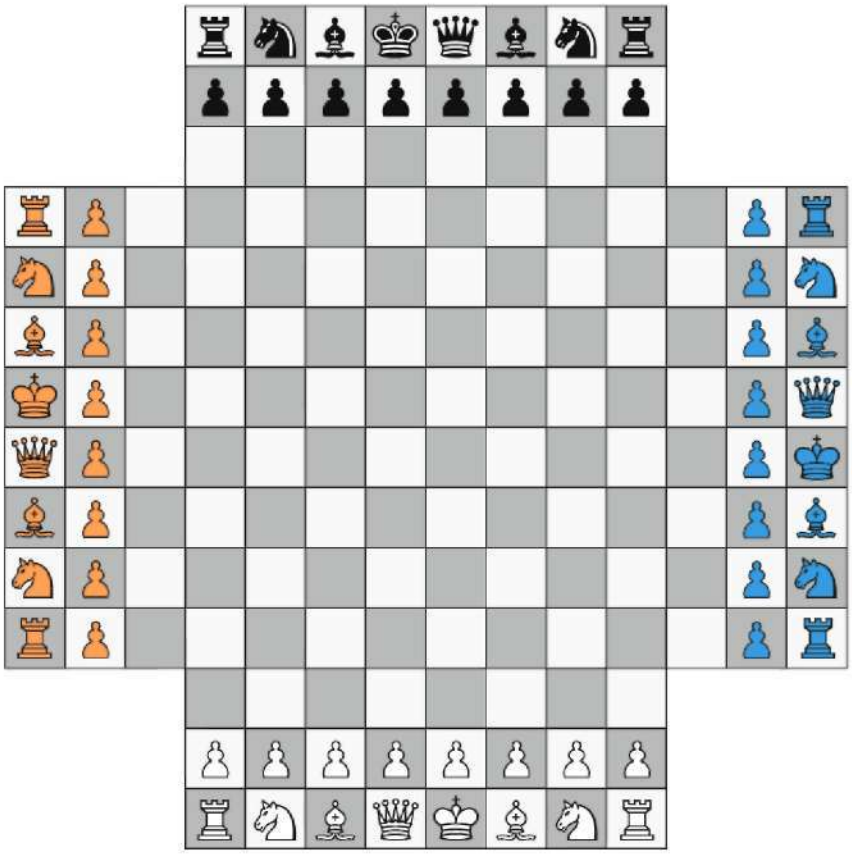


Рисунок 1.5. Розташування фігур у шахах на чотирьох

Отже шахи на чотирьох — це захоплюючий варіант традиційних шахів, який додає нові виміри стратегії та тактики. Завдяки участі чотирьох гравців, ця версія гри пропонує унікальні виклики, що вимагають не тільки знання класичних шахових правил, але й вміння взаємодіяти з іншими гравцями, передбачати їхні ходи та швидко адаптуватися до змін на дошці. Такий варіант шахів сприяє

розвитку соціальних навичок та покращує командну гру, роблячи процес гри ще більш динамічним і цікавим.

Наступним варіантом гри в шахів є анти-шахи. Анти-шахи також відомі як "шахи навпаки" або "шахи з униканням". Вони є варіантом шахів, де мета гри і правила руху фігур значно відрізняються від традиційних шахів. Ось основні правила та цілі гри:

- Головна мета гри в анти-шахи — позбутися всіх своїх фігур. Гравець, який першим позбувся всіх своїх фігур, виграє партію. Якщо один з гравців не може зробити жодного ходу і при цьому у нього залишилися фігури, він програє партію.
- Взяття є обов'язковим. Якщо гравець має можливість взяти фігуру суперника, він зобов'язаний це зробити. Якщо є декілька можливих варіантів взяття, гравець може обирати, яку фігуру взяти.
- Всі фігури рухаються так само, як у традиційних шахах, але з обов'язковим взяттям при можливості. Король не має особливого значення і може бути взятий як будь-яка інша фігура, а рокіровка не дозволяється.
- Коли пішак досягає останньої горизонталі, він перетворюється в будь-яку іншу фігуру за вибором гравця (звичайно в ферзя).

Анти-шахи кардинально змінюють традиційні стратегії шахової гри, оскільки ціль гри протилежна звичайним шахам — позбутися всіх своїх фігур замість захисту та нападу на короля суперника. Це впливає на такі аспекти гри як: позиційна гра, тактика взяття, захист фігур, рух пішаками та інші аспекти гри.

Шахи Фішера, також відомі як Шахи 960, були створені відомим шахістом Боббі Фішером у 1996 році. Цей варіант шахів має декілька ключових відмінностей від класичних шахів:

- Початкова розстановка фігур;

  В шахах Фішера початкова розстановка фігур на першій і восьмій горизонталях вибирається випадковим чином із 960 можливих варіантів. Є кілька обмежень: слони повинні стояти на клітинах різних кольорів, а

король повинен знаходитися між турами для можливості рокіровки.

- Рокіровка;

  Правила рокіровки аналогічні класичним шахам, але враховують випадкову початкову розстановку фігур. Рокіровка завершується таким чином, що після неї король і тура займають стандартні позиції, як і в класичних шахах.

  - Пішакова структура;

  Початкова розстановка пішаків не змінюється, вони розташовані на другій і сьомій горизонталях, як і в традиційних шахах.

- Інші правила гри;

  Всі інші правила гри (рух фігур, взяття, шах, мат тощо) залишаються такими ж, як і в класичних шахах.

Розглянемо переваги та недоліки цих правил. Почнемо з переваг: випадкова початкова розстановка фігур усуває перевагу, яку гравці можуть мати завдяки глибокому знанню дебютів у класичних шахах, ставлячи обох гравців в однакові умови на початку гри. Шахи Фішера стимулюють креативність і стратегічне мислення, оскільки гравцям доводиться розробляти нові плани і тактики, адаптуючись до незвичної початкової позиції. Випадкова розстановка фігур зменшує вплив заздалегідь підготовлених дебютів і аналізів, що робить гру більш чесною та динамічною. Оскільки гравцям доводиться адаптуватися до нових ситуацій, це сприяє покращенню їхнього загального розуміння шахової стратегії та тактики.

В цих шахах є і недоліки: випадкова початкова розстановка може бути важкою для початківців, оскільки вони не мають можливості використовувати відомі дебюти та можуть відчувати труднощі з оцінкою нових позицій. Шахи Фішера не такі популярні, як класичні шахи, що може обмежувати кількість доступних суперників та турнірів. Нові правила рокіровки можуть бути спочатку складними для розуміння та застосування, що може спричинити помилки у грі. Гравці не можуть підготуватися до партії так само глибоко, як у класичних шахах, що може бути як перевагою, так і недоліком залежно від рівня підготовки і стилю

гри гравця.

Розглянемо, де використовуються шахи Фішера. Головною проблемою цих шахів є складність для новачків; в них грають в основному гравці розвинутого рівня. Але крім гросмейстерів є ще один пласт гравців, які не мають проблем з складними шахами. Штучний інтелект також грає в шахи і вийшов на зовсім не досяжний для людей рівень. Тому для тестування і перевірки штучних інтелектів використовують саме шахи Фішера.

1.2.2 Аргументація вибору

Розглянувши основні шахові варіації, можна перейти до вибору правил саме для цього проекту. Першим критерієм є простота. За цим критерієм можна відкинути шахи Фішера. Шахи Фішера є складними для рядових гравців і тим паче, для новачків. Другим критерієм є простота інтерфейсу. Оскільки проект буде орієнтований на мобільні пристрої, буде тяжко розташувати всі фігури на дошці для чотирьох гравців, тому шахи на чотирьох теж можна відкинути. І останнім критерієм є розповсюдженість та зрозумілість, за цим критерієм виграють стандартні шахові правила. Вони більш відомі та розповсюдженні, тож чим більша кількість користувачів буде зацікавлена в саме таких правилах, тим більша кількість гравців буде спроможна грати в мою гру.

1.3 Аналітичний огляд програмних рушіїв з умовно безкоштовним доступом

Розвиток ігрової індустрії в останні десятиліття визначає необхідність використання сучасних технологій для розробки ігор. З поглибленням цифровізації та збільшенням кількості геймерів важливо знайти оптимальні і ефективні засоби для створення ігрових продуктів. Аналіз систем для розробки ігор стає актуальною задачею для розробників та бізнесу в цій галузі. Головною метою цієї частини дипломного проекту – є вивчення та аналіз сучасних систем для розробки ігор, визначення їх переваг та недоліків.

Перейдемо до огляду програмно-ігрових рушіїв. Вони є необхідними інструментами для розробки відеоігор, які надають зручність та ефективність у процесі програмування та відлагодження. Вони дають можливість

використовувати вже готовий двигун, який може прискорити розробку гри. Одною з таких IDE є Unity.

Unity підтримує розробку ігор для різних платформ, включаючи ПК, консолі та мобільні пристрої, і використовує мову програмування C#. Також програмні інженери можуть ефективно створювати логіку гри, взаємодію з об'єктами, обробляти фізику та реалізовувати алгоритми штучного інтелекту в самій Unity. Інтерфейс рушія Unity можна побачити на рисунку 1.6.

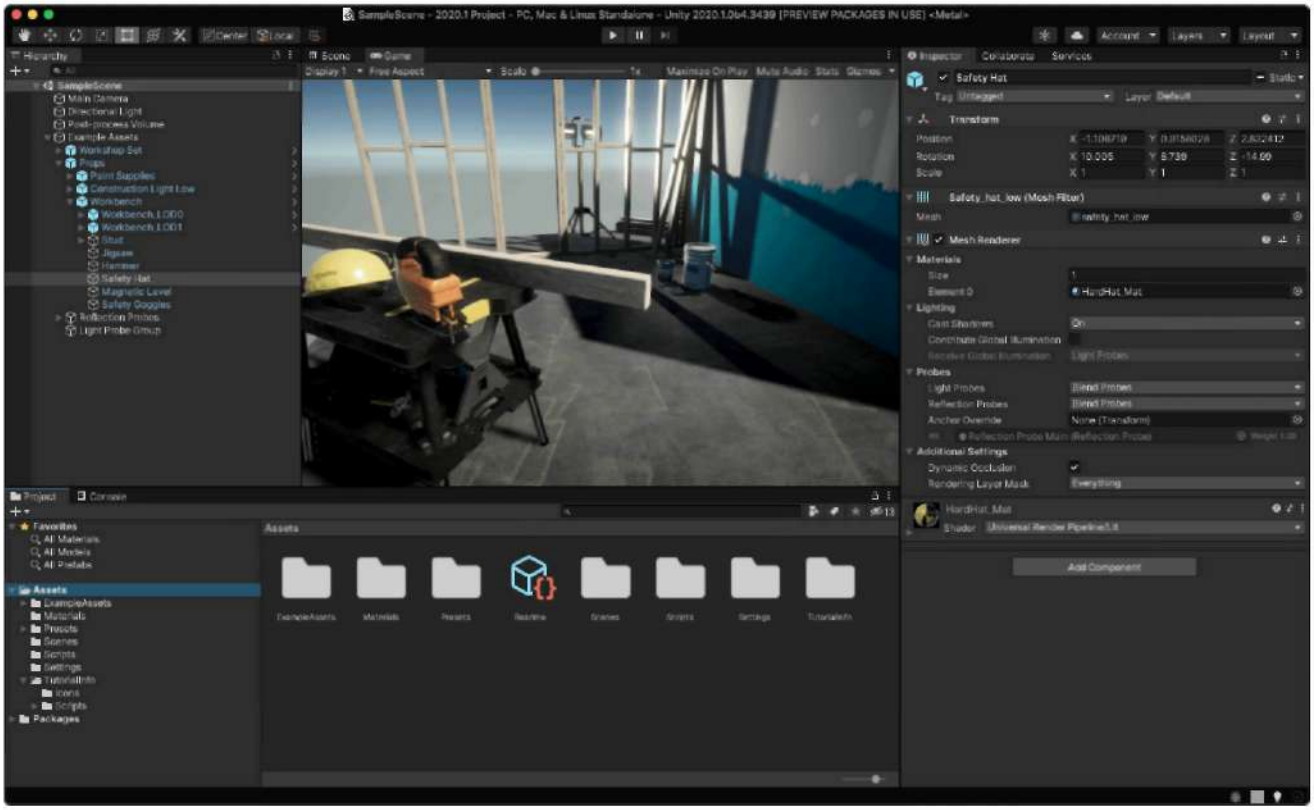


Рисунок 1.6. Інтерфейс програмного рушія Unity

Однак, високо використовуваним рушієм геймдеву також є і Unreal Engine. Unreal Engine також має свій власний двигун, який дозволяє розробникам створювати ігри на мові програмування C++ або використовувати візуальний інтерфейс Blueprint для швидкої інтеграції логіки гри, як показано на рисунку 1.7:

Інші популярні порграмно грові рушії включають Godot Engine, який має відкритий код та надає інтуїтивний інтерфейс розробки для 2D та 3D ігор. Також він має вбудовану систему скриптів (GDScript), що робить його привабливим для інді-розробників та освітніх проектів. Проте, через відносно невелику спільноту користувачів та обмежену кількість навчальних ресурсів, Godot поки що не здобув

широкої популярності.

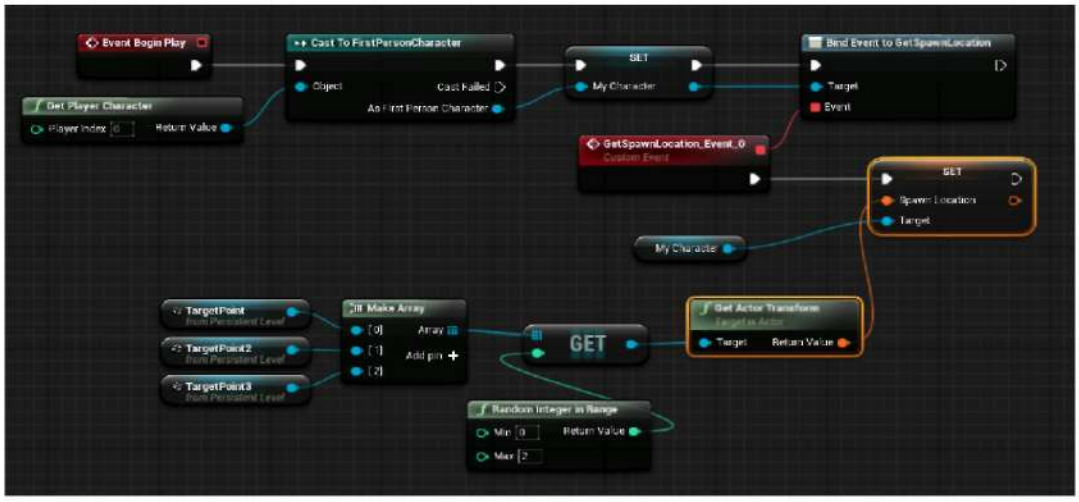


Рисунок 1.7. Використання інтерфейсу Blueprint в Unreal Engine.

Або ж GameMaker Studio, призначений для розробки 2D ігор з використанням власної мови програмування GML. Також є більш спеціалізовані ігрові рушії які використовуються для більш конкретних задач та жанрів ігор. Прикладом такого програмно ігрового рушія є RPG Maker. Він є примітивним але доречним інструментом у розробці невеликих ігор у жанрі jRPG. Його інтерфейс показаний на рисунку 1.8.

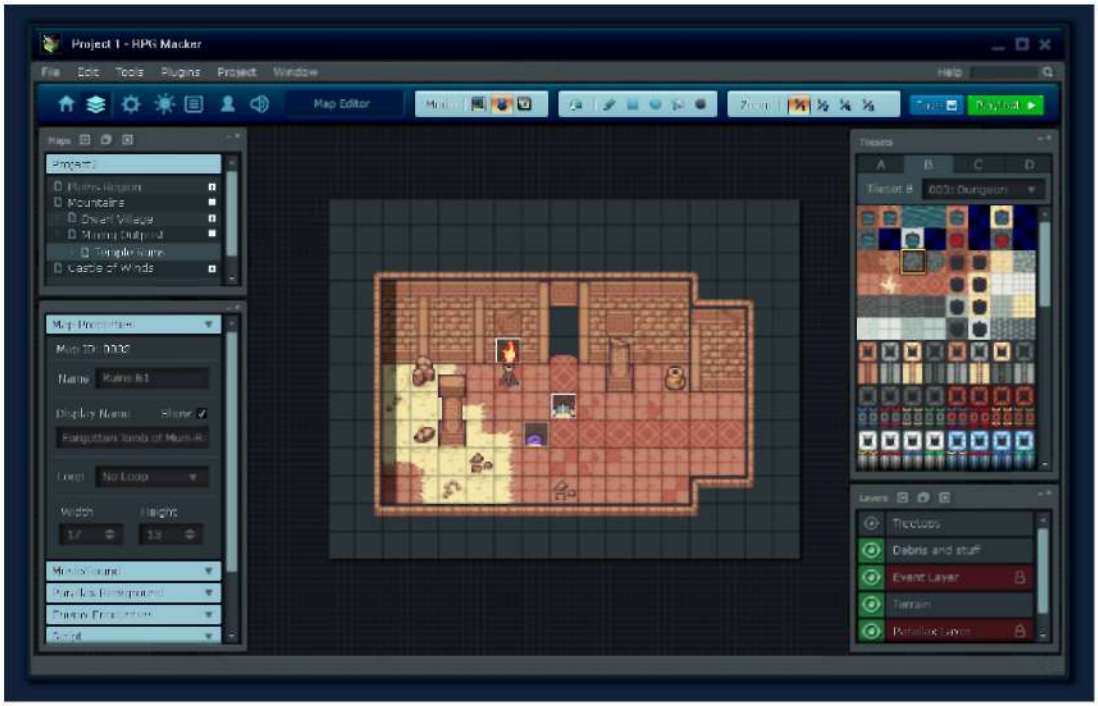


Рисунок 1.8. Інтерфейс програмно ігрового рушія RPG Maker

Ще один специфічний ігровий рушій — Defold. Він орієнтований на розробку 2D-ігор і пропонує легку, але потужну платформу для створення

мобільних та веб-ігор. Defold використовує мову програмування Lua, що забезпечує швидкість і простоту розробки. Особливістю цього рушія є його відмінна інтеграція з хмарними сервісами для зберігання проектів і спільної роботи над ними. Незважаючи на свої переваги, Defold залишається маловідомим через свою вузьку спеціалізацію та конкуренцію з іншими 2D-рушіями, такими як Unity і Construct.

Після огляду безкоштовних програмно ігрових рушіїв я обрав саме той, який підходить. Основними критеріями відбору є простота, велике ком'юніті, частота використання, універсальність та C# як основна мова програмування. За критеріями великих ком'юніті та частоти використання відпадають майже всі рушії, але ще залишаються Godot Engine, Unity та Unreal Engine. Хоча Godot більш простий і нішевий, але він не має такої універсальності як інші рушії, також для того щоб добитися кросплатформенності потрібно витратити набагато більше ресурсів і сил, що не є ідеалом. Залишилися лише Unity та Unreal Engine. Хоча це два дуже близькі рушії вибір падає саме на Unity, бо вона не потребує ніяких плагінів та додатків для підтримки C# як основної мови. Також саме Unity має найбільший спектр інструментів для забезпечення кросплатформенності.

1.4 Формування концепту ігрового процесу шахової гри

Цей етап включає визначення основних механік гри, розробку інтерфейсу користувача та планування додаткових функцій. Формування концепту ігрового процесу є основним етапом у розробці шахової гри. Цей концепт визначає загальні принципи, механіки та елементи, що забезпечать захоплюючий ігровий досвід для гравців.

Почнемо з огляду основних механік гри. Як описано в попередньому розділі я буду використовувати стандартні шахові правила, які включають в себе: стандартній рух фігур, рокіровку, взяття на проході, перетворення пішака, правила шаху та мату. Тепер, коли визначено з якими правилами будемо працювати, можна перейти до схематичного описання логіки деяких з них.

Першим і найбільш головним є правила перемоги, тобто правила шаху та мату. Спочатку розберемо, як буде перевірятися шах, бо саме через цю перевірку

буде змога перевірити гравця на можливий мат. Першим і головним є пряма атака на короля, але у випадку з конем, атака може бути і не прямою. Спочатку перевіряється, чи є фігура на одній лінії з королем, а потім – яка саме ця фігура, і в останню чергу перевіряється чи є шах від коня. Також потрібно зазначити, що для економії ресурсів комп'ютера потрібно зробити перевірку на наявність фігури свого кольору для блокування непотрібних розрахунків після знаходження фігури. Вигляд схеми логічної перевірки шаху показаний на рисунку 1.9.

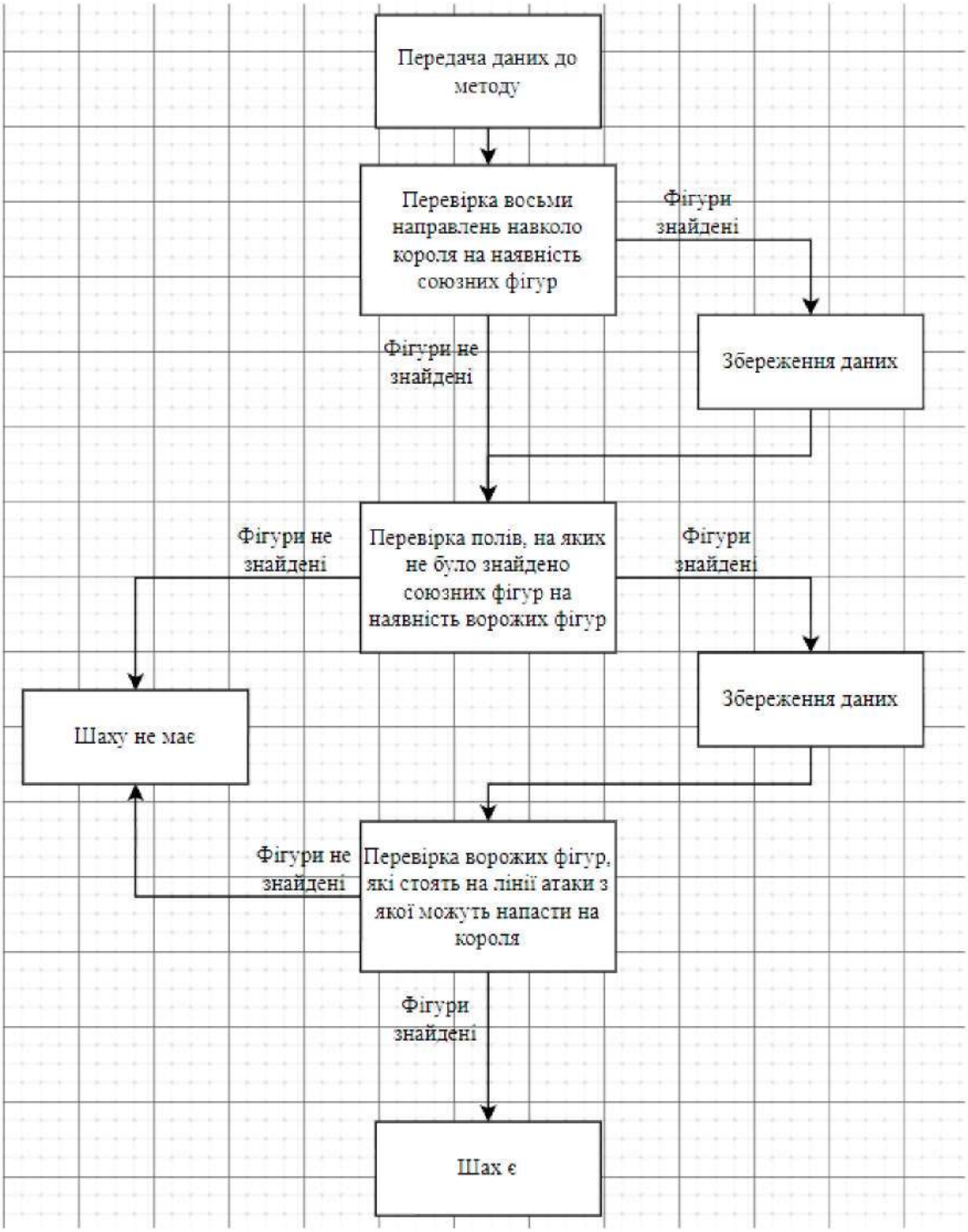


Рисунок 1.9. Схематичне зображення перевірки шаху

Після цієї серії перевірок можна додати лише одну і вона зможе сказати, чи поставлений мат королю. Ця перевірка звучить так: «чи покаже перевірка ще один шах після будь-якого пуху короля?» Тепер є логіка перемоги і можна перейти до інших правил.

Першим з них буде перевірка на можливість пішака стати королевою. Вона дуже проста, знаючи скільки полів на дошці можна перевірити, чи дійшов пішак до кінця дошки. Наступним правилом є рокіровка. Реалізація цього правила більш складна ніж перевірка чи дійшов пішак до кінця, але значно простіша за перевірку на шах чи мат. Першим фактором повинно бути чи ходив король або тура. Згідно шахових правил: «рокіровка може бути виконана якщо: король та тура з якою гравець має намір провести рокіровку не зробили жодного ходу». Саме тому першої перевіркою буде на робили ходи король та тури. Для оптимізації треба зазначити, що вигідніше перевірити короля першим, бо якщо одна з турів зробила хід можна провести рокіровку з іншою, а якщо король зробив хід, він не може зробити рокіровку з жодною з турів. Потім потрібно перевірити чи є між туром та королем інші фігури. І останнім перевіряємо чи є можливий шах на шляху, через який буде проходити рокіровка. А тепер останнє правило шахів – взяття на проході. Нам потрібно перевірити чи міг би пішак взяти пішака, якщо б ворожий пішак не скористався рухом на дві клітинки без попереднього ходу.

Коли приблизно розуміємо правила, які будемо використовувати, можна перейти до обрання того, як буде виглядати мій проект. Першим і найбільш головним є вирішення, чи буде проект на площині або ж він буде в 3D. Я вибрав проект у стилі 2D, тому що спрайти для дошки та фігур значно простіше зробити, ніж 3D моделі для гри в цьому ж стилі. Також завдяки вибору 2D моделі можна гарантувати, що проблем з грою на одному й тому самому девайсі не буде. 2D моделі дадуть можливість використовувати одні й ті ж самі ассети для різних платформ, бо велико деталізовані 3D моделі можуть сильно напружувати систему на якій буде встановлена гра. Гра, також планується для мобільних пристроїв, саме цьому було прийняте рішення використовувати 2D рушій.

Зараз про режим гри. Гра планується бути для двох гравців на одному

пристрої. Я обрав таку модель програми через те, що сьогодні зменшилася зацікавленість молоді збиратися групами для цікавого дозвілля, тому сподіваюся, що ігри, які дають можливість грати «лице в лице», будуть спонукати молодь частіше збиратися разом.

На даному етапі проектування та програмування гри я прийняв рішення, що це будуть 2D шахи для двох гравців. Основним пристрієм для гри буде мобільний телефон. Це будуть стандартні шахи, для більш широкої аудиторії гравців та більших можливостей для покращення та додання нового функціоналу.

1.5 Проектування основних елементів ігрового процесу та принципи їх роботи

1.5.1 Проектування руху по дошці

В цьому розділі спроектуємо основні елементи ігрового процесу шахової гри на платформі Unity. Це включає в себе визначення ключових компонентів, розробку їх взаємодії та забезпечення плавного ігрового досвіду. Основними елементами ігрового процесу є шахівниця, шахові фігури, правила гри, інтерфейс користувача (UI) та додаткові функції.

Почнемо з дошки, її положення та розстановці фігур на ній. Оскільки гра планується для мобільних пристроїв, дошка повинна бути достатньо простою на вертикально орієнтованою. Говорячи про те, як будуть розставлятися фігури на ній нам потрібно спочатку вирішити як буде проходити калькуляції позицій у грі при русі або початкових позиціях. У звичайних шахах це позначення кожної клітини від A1 до H8. Це можна реалізувати через двовимірну матрицю. Тож для руху будемо додавати значення до позиції на дошці тим самим змінюючи положення в матриці і на дошці.

Зараз поговоримо як буде реалізовуватися рух на дошці. Зазвичай фігури ходять лінійно відносно вертикалей та горизонталей або діагоналей. Але є фігури які ходять за своїми власними правилами. Це король, який ходить виключно навколо свого місця перебування. Це пішак, який ходить вперед до протилежного кінця поля. І це кінь, який ходить на дві клітинки і змінює направлення паралельно до першого руху на одну клітинку, створюючи паттерн руху, який нагадує букву

«Г». Оскільки більшість фігур ходить лінійно, зробимо логіку їх руху першою. Так як гра знаходиться у двовимірному просторі будемо постійно додавати або віднімати одиницю до потрібних координат. Наприклад для руху вверх потрібно додавати 1 для значення координати Y, а для руху в діагоналі потрібно додавати або віднімати від обох координат одразу. Наприклад для руху у ліву нижню діагональ потрібно відняти значення 1 від обох координат X та Y. Наступним буде рух пішака. Для цієї фігури є різниця в русі від кольору, тому що, для руху білих пішаків потрібно додавати 1 до X координати, а для чорного віднімати. Тож рух для пішаків це дві різні функції. Останнім фактором руху пішаків є їх атака. Вони не атакують за своїм стандартним рухом, тому потрібно додати окрему можливість атакувати. І останнім є рух коня. Кінь рухається на дві клітини в обидві вертикалі і горизонталі та потім на одну перпендикулярно першому ходу. Як виглядає математична схема руху королеви можна побачити на рисунку 1.10.

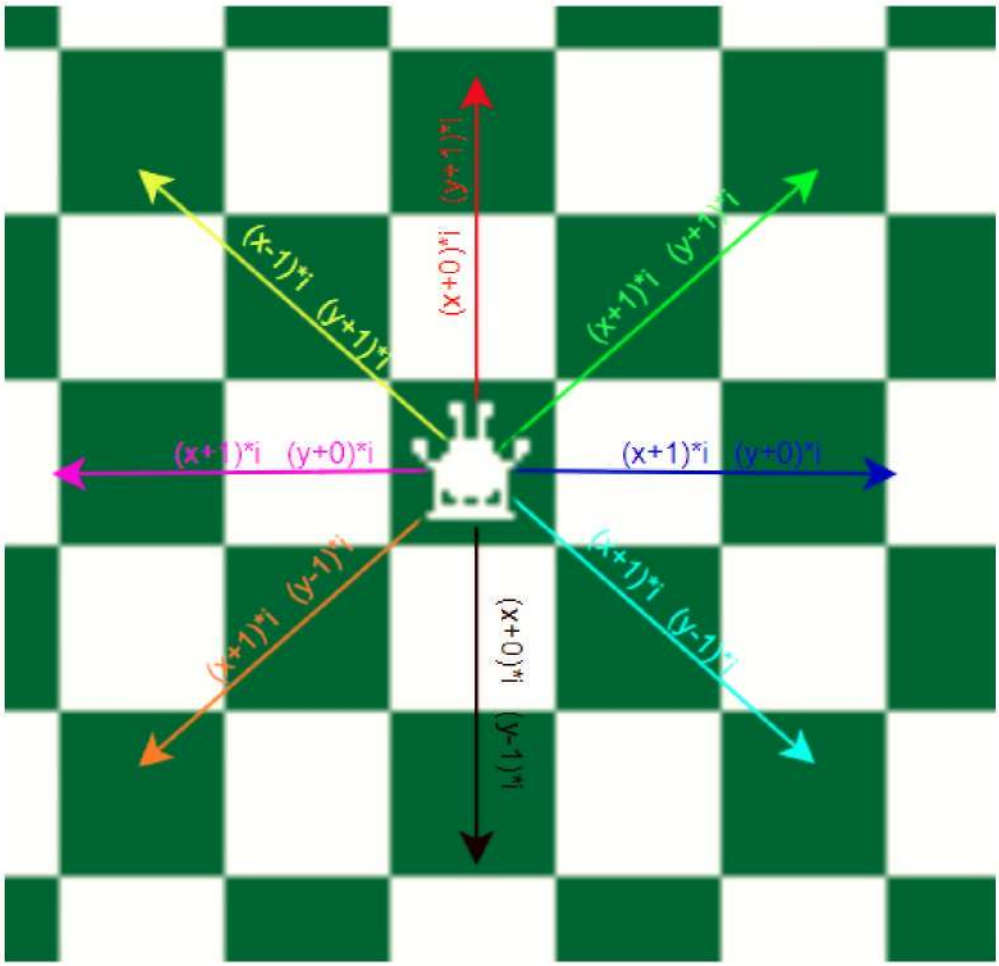


Рисунок 1.10. Схематичне зображення математичного руху королеви

1.5.2 Проектування інтерфейсу та підказок для гравця

Наступним елементам проектування є проектування інтерфейсу. Оскільки гра планується на мобільні пристрої, інтерфейс потрібен бути простим та не великим. Для спрощення інтерфейсу в цілях простоти розуміння та оптимізації запуск гри можна робити одразу ж після запуску програми, а при перемозі, можна зробити перезапуск гри при повторному натисканні на екран. Також потрібно розробити меню, яке буде демонструвати гравця, який переміг у грі. Для цього можна зробити змінну, яка буде перевіряти на надходження даних у вигляді строки, яка буде в себе включати який саме гравець виграв, а саме білий чи чорний. Взаємодію між двигуном Unity та цією змінною можна показати на рисунку 1.11.

Рисунок 1.11. Схема взаємодії даних в функції виводу переможця

Як можна побачити на рисунку 1.11, спочатку функція яка перевіряє гру на наявність мату відправляє данні двигуну Unity, який в свою чергу передає їх на обробку у скрипт, який розшифрувавши ці дані, розуміє хто переміг і передає їх у функцію яка запише їх у змінну і передасть їх двигуну Unity, який використовуючи дані в змінній виведе на екран хто саме переміг після виклику необхідної функції.

Також потрібно зробити деякі підказки, наприклад куди може ходити фігура та кого вона може атакувати. Тобто потрібно не тільки прорахувати траєкторію ходу а ще й показати її і міняти в залежності від атака це, чи просто хід. Основні елементи які треба виділити:

- При виборі фігури треба перевірити чи може вона робити хід і не буть порушені правила шахів.
- Після перевірки можливості ходу потрібно підсвітити ці самі ходи:

клітини, на які може переміститися обрана фігура, підсвічуються спеціальним кольором (наприклад, зеленим), клітини, на яких розташовані фігури суперника, які можуть бути взяті, підсвічуються іншим кольором (наприклад, червоним).

- Також потрібно дати можливість гравцю змінити фігуру, якою він планує ходити: якщо гравець натискає на іншу фігуру або порожню клітину, підсвічування зникає і система або вибирає нову фігуру, або скасовує вибір.

Для всіх цих елементів можна використовувати різні скрипти які будуть взаємодіяти між собою за допомогою методів об'єктно орієнтованого програмування. Це означає що в кожному скрипті буде розроблена окрема частина гри і взаємодія їх буде давати змогу грати в мою гру. В першому скрипті можна розташувати фігури перед початком гри та проводити основні дії, такі як: наступний хід, обчислення позицій та переможний алгоритм. В наступному скрипті можна розробити логіку фігур, а саме присвоєння їм спрайтів, логіки поведінки, перевірка їх положення та деякі нестандартні правила. І в останньому скрипті можна зробити всі функції інтерфейсу: підсвічування ходів, виклик меню переможця, тощо. Схему скриптів та коду що в них буде можна побачити на рисунку 1.12:

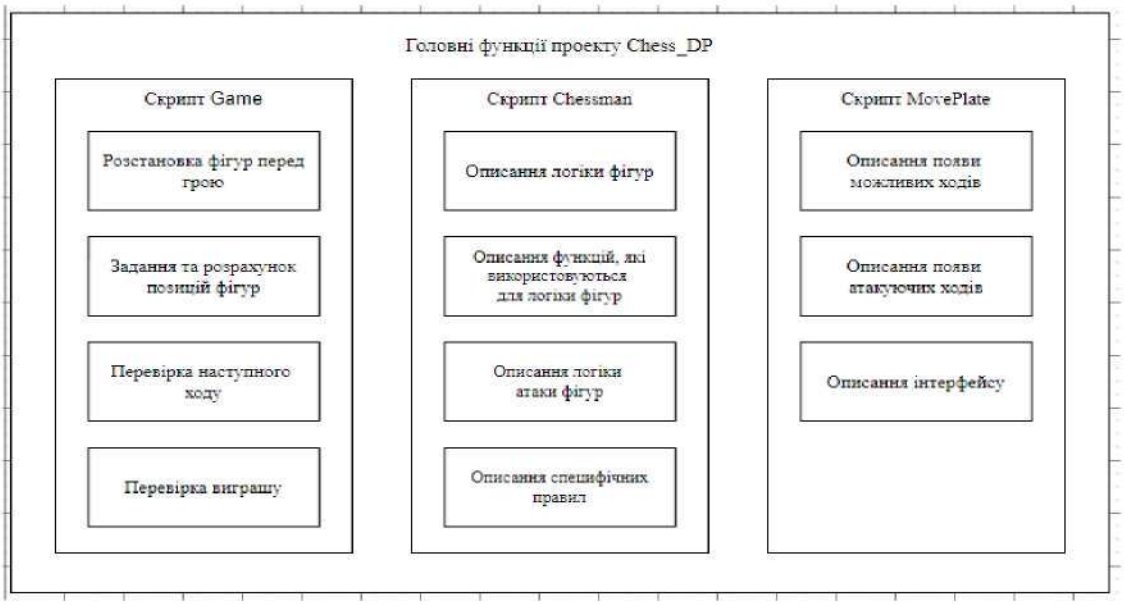


Рисунок 1.12. Схематичне зображення головних функцій скриптів

Наступне питання для проектування гри – це створення зображень, які будуть використовуватися для ігрових моделей. Для створення спрайтів буде використовуватися програма Adobe Photoshop. Також були досліджені різні методи створення спрайтів для 2D ігор на програмному рушії Unity. З досліджень видно, що для ігор такого типу краще всього використовувати формат файлів .png, в якому будуть зберігатися ігрові спрайти, бо в Unity є можливість змінити формат графіки моделей, якщо він відповідає критеріям:

- Формат файлу: .png або .jpg
- Створення моделей в програмах які підтримують як растрову графіку, так і векторну;
- Файли спрайтів повинні бути доданні до проекту саме як спрайти.

Коли вже зрозумілі основні параметри проекту, а саме: як будуть реалізовуватися деякі частини проекту, структура проекту, взаємодія з вбудованими функціями рушія Unity і робота з моделями для проекту можна перейти до етапу розробки самого проекту.

1.6 Реалізація основних елементів ігрового процесу

1.6.1 Вибір шаблону проекту

Для реалізації завдання проекту потрібно створити проект Unity. Я буду створювати 2D проект як було зазначено в попередніх розділах. Під час процесу створення проекту я обрав, який це саме проект, з усіх варіантів обираю між двома: Unity: 2D (Built-In Render Pipeline), Universal 2D. Розберемо кожен з них.

2D (Built-In Render Pipeline) - це традиційний шаблон для створення 2D-ігор у Unity, який використовує вбудований рендерний конвеєр (Built-In Render Pipeline). Цей шаблон забезпечує базові можливості для розробки 2D-ігор без додаткових налаштувань. Його основні особливості це вбудований рендерний конвеєр: використовує класичний рендерний конвеєр Unity, який добре підходить для більшості 2D-ігор. Простота використання: шаблон включає стандартні інструменти та компоненти для роботи з 2D-графікою, такі як Sprite Renderer, 2D Physics, Tilemap. Стабільність та сумісність: підтримка багатьох платформ і добре

перевірений у численних проектах. Ось деякі з його переваг: простота для новачків, лише найбільш популярні налаштування, підтримує найбільшу кількість функцій, які більше не доступні на більш нових версіях рушія. Але в нього також є і недоліки: обмежені можливості налаштування графіки, що дає менше опцій для налаштування рендерингу та оптимізації порівняно з новішими конвеєрами, та відсутність новітніх функцій.

Другий це – Universal 2D. Він – це шаблон для створення 2D-ігор на основі Universal Render Pipeline (URP), який надає покращені можливості рендерингу та більше інструментів для налаштування графіки. Основними особливостями цього шаблону це використовування URP, який пропонує більш сучасний та ефективний підхід до рендерингу. URP також підтримує різні ефекти, такі як пост-обробка, освітлення, шейдери та інші вдосконалення. Та останнє більше можливостей для налаштування матеріалів, освітлення та ефектів. Його переваги над Unity: 2D (Built-In Render Pipeline) це покращена продуктивність та оптимізація, підтримка передових графічних ефектів та технологій, більше інструментів для налаштування рендерингу та створення унікального візуального стилю. Ось його недоліки: складність налаштування, вимоги до навчання та найбільш головний, це можливі проблеми з сумісністю.

Оскільки шахова гра планується для мобільних пристроїв з можливістю розширення кількості платформ потрібно використовувати найбільш простий та сумісний шаблон. Також через відсутність необхідності використання складних рендерів та комплексних шаблонів та текстур вигідніше використовувати саме 2D (Built-In Render Pipeline). Процес створення проекту на необхідному шаблоні можна побачити на рисунку 1.13.

При створенні проекту Unity створює камеру та сцену, але, якщо стандартні налаштування камери нам підходять то налаштування сцени – ні. Нам потрібно змінити роздільна здатність сцени з 1980X1080 на 1080X1980, саме ця роздільна здатність є універсальною для більшості мобільних пристроїв що нам і необхідно. Зміна цих налаштувань показана на рисунку 1.14.

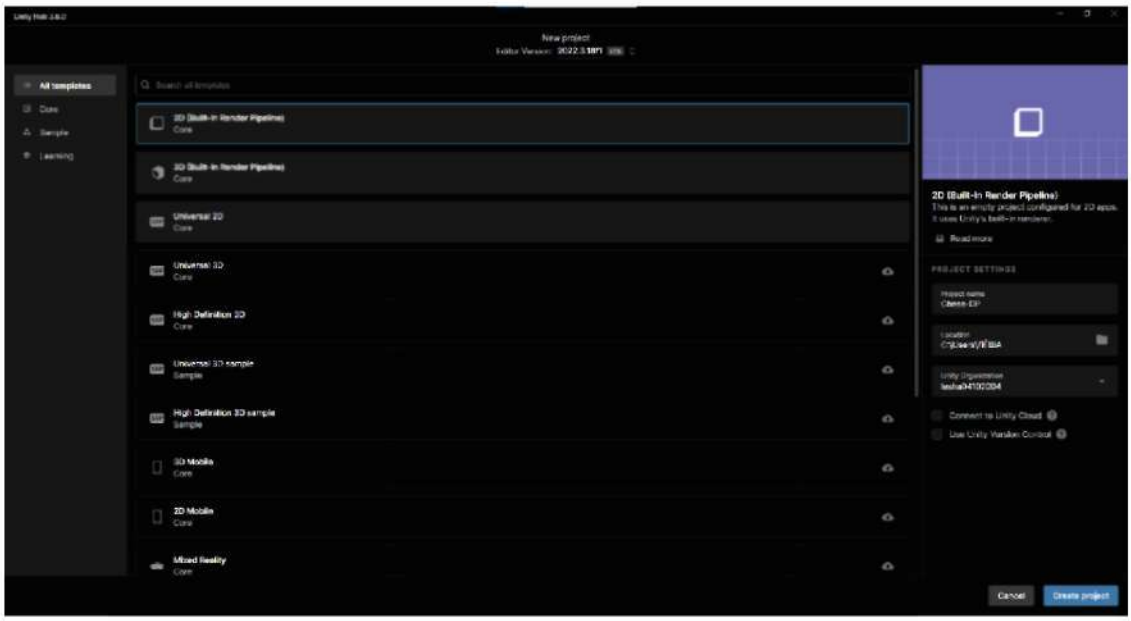


Рисунок 1.13. Створення проекту на шаблоні 2D (Built-In Render Pipeline)

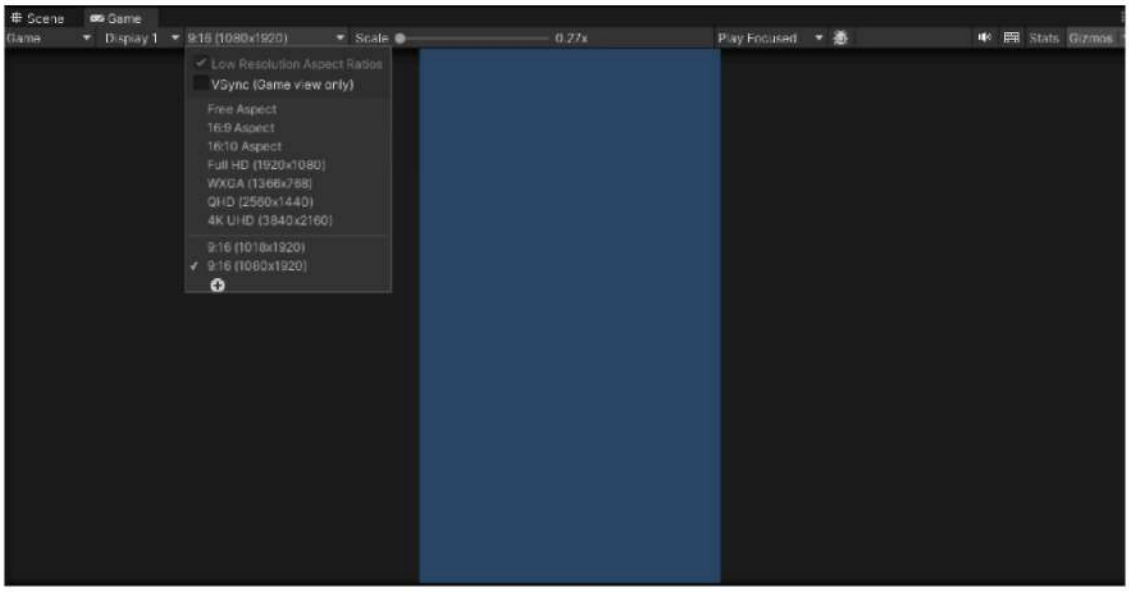


Рисунок 1.14. Зміна налаштувань сцени

Тепер перейдемо до створення спрайтів для дошки та фігур і додання їх до проекту. Для початку зробимо спрайти у програмі Adobe Photoshop. Всі спрайти мають бути на прозорому фоні та збережені у форматі .png та додані до проекту. Деякі з них можна побачити на рисунку 1.15.

Попередньо створивши папку, в яку ми будемо зберігати всі спрайти, додаю їх до проекту. Після цього змінюю графіку всіх спрайтів з растрової на векторну, завдяки влаштованій в Unity функції, а потім змінюю колір сцени з синього на чорний.

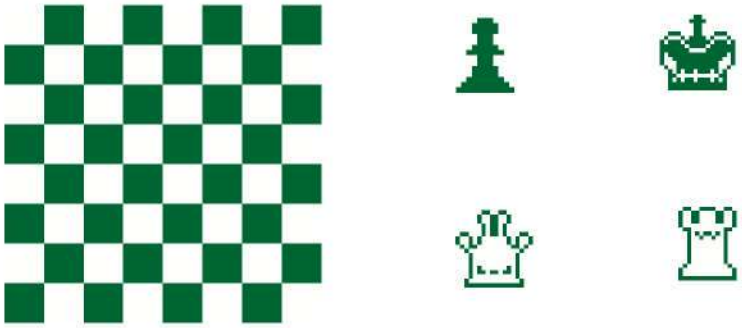


Рисунок 1.15. Створені спрайти дошки та деяких фігур

1.6.2 Створення методів для появи та вирівнювання фігур

Тепер створимо перший скрипт, в якому пропишемо функцію, яка буде розставляти фігури на ігровій дошці. Спочатку потрібно створити на сцені у параметрах ієрархії об'єкт за допомогою якого буде дублюватися і змінюватися його спрайт, що відповідатиме фігурі, яка повинна знаходитися на цій, обраній, позиції.

В коді потрібно створити об'єкт, через який будемо звертатися до об'єкта на сцені. Також потрібно створити двовимірний масив, який буде представляти собою умовну дошку. Відповідно до положень потрібно створити масив фігур для кожного гравця, в який буде записувати, які є фігури у гравця та через який можна буде дізнатися, які фігури в нього залишилися. Частина коду, яка буде створювати фігури на дошці показана на рисунку 1.16 та нижче у текстовому форматі:

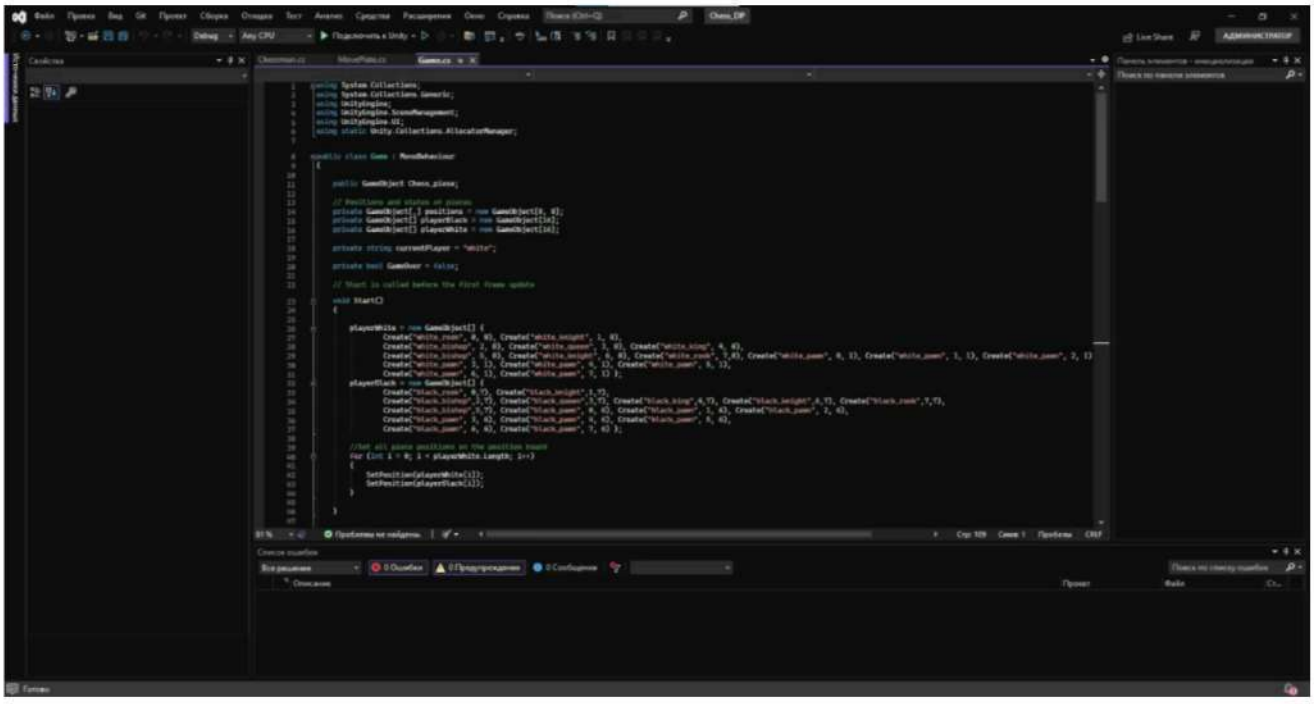


Рисунок 1.16. Код у редакторі Microsoft Visual Studio

```
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_knight",6,7), Create("black_rook",7,7),
Create("black_bishop",5,7), Create("black_pawn", 0, 6), Create("black_pawn", 1, 6),
Create("black_pawn", 2, 6), Create("black_pawn", 3, 6), Create("black_pawn", 4, 6),
Create("black_pawn", 5, 6), Create("black_pawn", 6, 6), Create("black_pawn", 7, 6) };
    //Поставити позиції всіх фігур на дошку
    for (int i = 0; i < playerWhite.Length; i++)
    {
        SetPosition(playerWhite[i]);
        SetPosition(playerBlack[i]);
    }
}
```

Ще не створений метод «Create», тому створимо його. Він буде копіювати вже існуючий об’єкт, наділяти його координатами і відправляти на дошку. Також потрібно створити методи, які будуть брати ставити координати для фігур, так названі «гетери» та «сетери».

Тепер перейдемо до наступної частини, потрібно не просто поставити фігури на позиції, а ще й змінити спрайти на необхідні та присвоїти фігури кожному гравцю. Також потрібно створити функцію, яка буде центрувати фігури на клітинках, бо зараз вони просто знаходяться рядом з їх передбаченою позицією. Ця функція буде множити координати на знайдені константи 0,66 та віднімати 2,3, які вирівнює фігури по клітинкам. Це можна побачити на прикладі коду та на рисунку 1.17.

```
public void SetCoords()
{
    float x = xBoard;
    float y = yBoard;
    x *= 0.66f;
    y *= 0.66f;
```

```
        x += -2.3f;
        y += -2.3f;
        this.transform.position = new Vector3(x, y, -1.0f);
    }
```

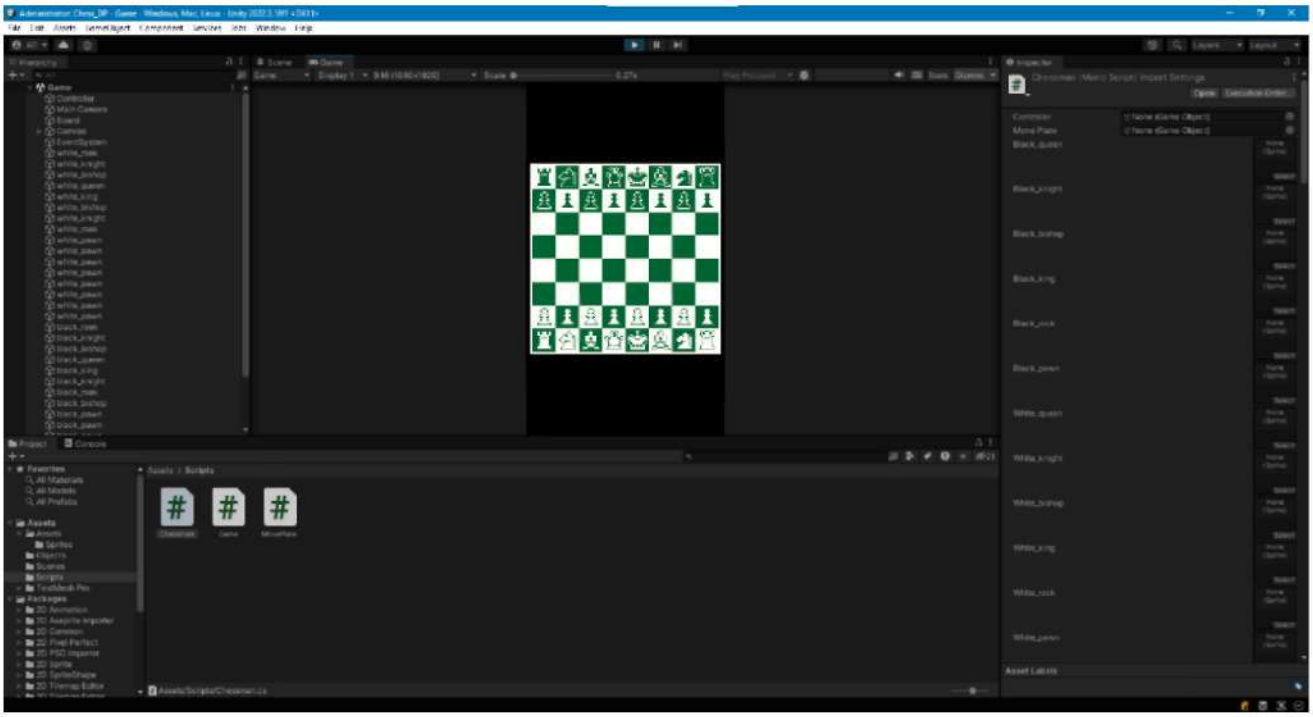


Рисунок 1.17. Розташування фігур після додання функція вирівнювання

1.6.3 Написання логіки фігур

Тепер можна спробувати прописати логіку руху фігур. Це і все інше зв'язане з логікою саме фігур буде розроблятися у наступному скрипті, тож почнемо. Для кожної фігури траєкторія руху та атаки унікальна тому щоб не робити для кожної фігури окремий метод руху, можна використати загальні методи та передати їх через функції «switch case». Оскільки зазначено імена фігур, можна звертатися до фігур саме за ними і перевіряти їх. Для більшості фігур можна використати метод який буде створювати рух по прямій лінії в залежності від координат, які будуть задані при виклику метода. Цей метод буде використовуватися для таких фігур як: королева, тура та офіцер. Ось код цього методу:

```
    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);
        }
    }
```

Створено метод, який приймає дві числові змінні, які будуть пояснювати куди буде направлена ця лінія руху. В циклі заплановано використовувати метод «MovePlate», який я запрограмую в майбутньому і за допомогою його буде показуватися, куди може ходити фігура. У функції «If» буде можливість перевірити, чи є на лінії руху ворожа фігура, що має параметр «player», який не дорівнює значенню параметру «player» самої фігури і в цьому випадку викликаємо інший, в майбутньому запрограмований метод «MovePlateAttackSpawn».

Наступним, потрібно створити метод, який буде описувати можливі ходи коня. Оскільки кінь не ходить по прямій він потребує окремого методу, який буде оброблювати його ходи. Ось його код:

```
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);
}
```

Використовуючи даний метод, що містить в собі ще не створену функцію «PointMovePlate», основною суттю якої буде створення підказки для гравця. Саме цей код виконує цю функцію:

```
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);
            }
        }
    }
```

Цей код описує вище надану функцію, отримуючи координати «x» та «y» і створюючи на цих координатах «MovePlate». Також потрібно створити метод для руху короля, він може бути схожий на метод руху коня, але буде використовувати різні з конем координати.

І останнім методом потрібно створити метод ходів пішака. Пішак найпростіша фігура, але з точки зору коду вона найбільш складна, оскільки пішак атакує на як стандартна фігура, паттерн його атаки відрізняється від паттерну його руху. Ось як виглядає код цього методу:

```
    public void PawnMovePlate(int x, int y)
    {
        Game sc = controller.GetComponent<Game>();
        if (sc.PositionOnBoard(x, y))
        {
            if (sc.GetPosition(x, y) == null)
            {
                MovePlateSpawn(x, y);
            }
        }
        // Діагональна перевірка атаки
        if (sc.PositionOnBoard(x + 1, y))
        {
            GameObject cp = sc.GetPosition(x + 1, y);
            if (cp != null && cp.GetComponent<Chessman>().player != player)
            {
                MovePlateAttackSpawn(x + 1, y);
            }
```

```
        }
        if (sc.PositionOnBoard(x - 1, y))
        {
            GameObject cp = sc.GetPosition(x - 1, y);
            if (cp != null && cp.GetComponent<Chessman>().player != player)
            {
                MovePlateAttackSpawn(x - 1, y);
            }
        }
    }
```

В цьому коді видно, що частина, яка виконує звичайний рух, дуже схожа на метод «PointMovePlate», але в методі ходу пішака, спостерігається ще дві перевірки: першу, яка перевіряє атакуючу можливість вліво і другу, яка перевіряє теж саме, але вправо. Коли вже є методи для обробки руху фігур, потрібно задати їх для цих самих фігур. Це будемо робити через функцію «switch case» в окремому методі «InitiateMovePlates» через перевірку імені фігури, тож розберемо цей код:

```
    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":
            if (yBoard == 6)
            {
                PawnMovePlate(xBoard, yBoard - 1);
            }
            break;
        case "white_pawn":
            if (yBoard == 1)
            {
                PawnMovePlate(xBoard, yBoard + 1);
            }
            break;
    }
}
```

Як можна побачити, в цьому коді використовуються вже створені методи і до них додаються необхідні координати. Як було показано вище, для того, щоб королева ходила в усіх напрямах, а саме: вертикаль, горизонталь та обидві діагоналі, просто надаються необхідні координати, які описують напрямок. Для коня, наприклад, теж просто вказується напрямок, в якому він спочатку робить хід на дві клітинки, а потім інший, в якому він робить хід лише на одну. Цей метод дозволяє роздати функціонал всім фігурам одночасно, без виклику окремих методів для кожної фігури.

1.6.4 Створення об'єктів для взаємодії з скриптами та підказок для гравців

Коли фігури вже можуть ходити, потрібно створити методи «MovePlateSpawn» та «AttackMovePlateSpawn», які будуть створювати візуальні зазначення, куди може піти фігура. Для початку потрібно створити спрайт цього

об’єкту і додати його у Unity, так само як і для попередніх спрайтів потрібно змінити растрову графіку на векторну, це показано на рисунку 1.18:

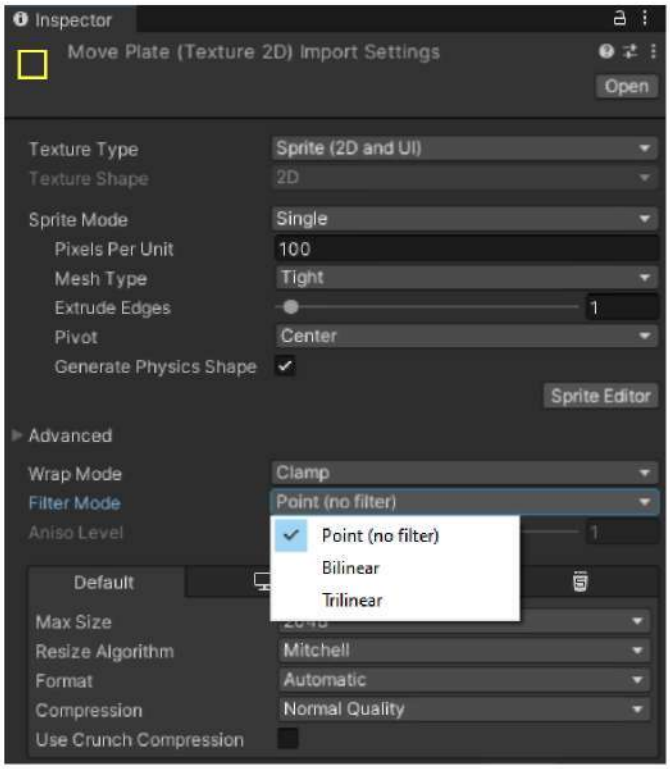


Рисунок 1.18. Зміна растрової графіки на векторну у Unity

Тепер потрібно створити об’єкт на основі цього спрайту і додати його до проекту у папку «Objects». Після створення об’єкту його потрібно налаштувати: змінити колір, закріпити за ним скрипт та додати параметр «Box Collider». Налаштування об’єкту виглядає як показано рисунку 1.19.

Тепер створимо методи «MovePlateSpawn» та «AttackMovePlateSpawn», які дозволять фігурам почати рухатися по дошці. Як і в методі, який створює фігури, потрібні змінні, до яких програма буде звертатися, після чого програма створить підказку для гравця, використовуючи координати, записані у відповідних змінних. Також потрібно перевести ці змінні до нашої системи, тобто вирівняти їх за допомогою раніше знайдених констант. Наступним потрібно створити об’єкт через функцію Unity «Instantiate». Ця функція приймає три параметри: ім’я об’єкту, його позицію у трьох координатах та кут повороту. І останнім потрібно прирівняти цей створений об’єкт до об’єкту в папці «Objects» та його координат. Теж саме потрібно зробити і у методі «AttackMovePlateSpawn», але єдина відмінність його в тому, що там додана змінна «attack», що буде відповідати за

можливість атаки через взаємодію з цим візуальним орієнтиром. Код цих методів показаний нижче.

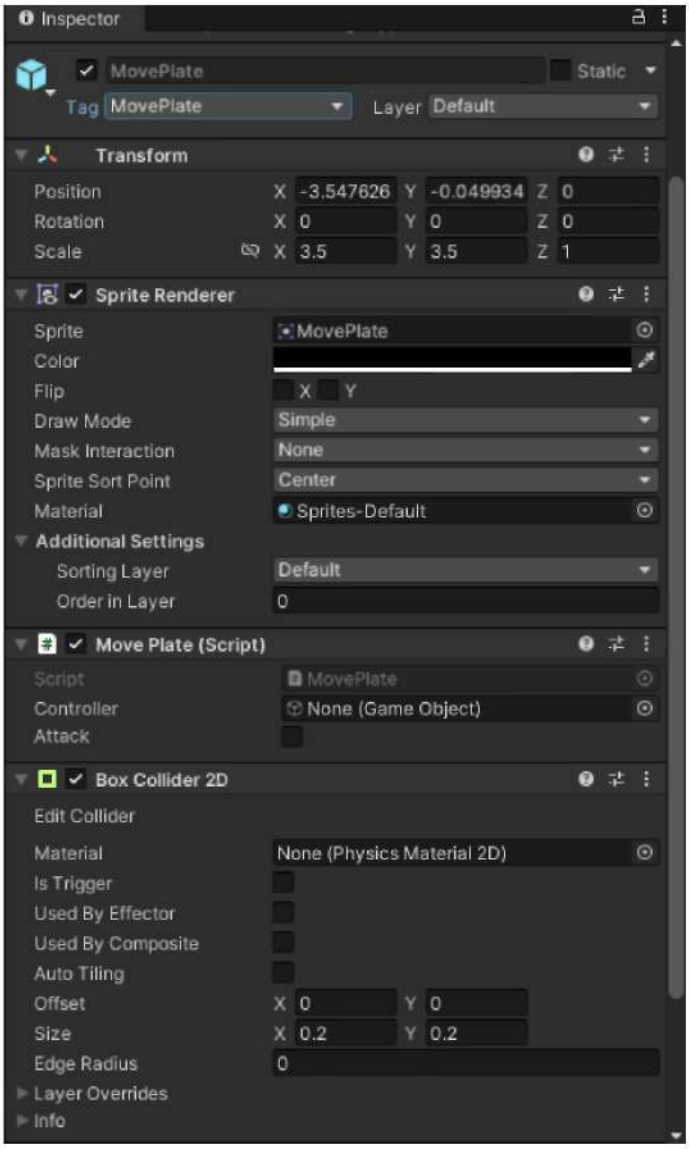


Рисунок 1.19. Змінені параметри об'єкту «MovePlate»

```
public void MovePlateSpawn(int matrixX, int matrixY)
{
    float x = matrixX;
    float y = matrixY;
    x *= 0.66f;
    y *= 0.66f;
    x += -2.3f;
    y += -2.3f;
    GameObject mp = Instantiate(movePlate, new Vector3(x, y, -3f),
Quaternion.identity);
    MovePlate mpScript = mp.GetComponent<MovePlate>();
    mpScript.SetReference(gameObject);
    mpScript.SetCoords(matrixX, matrixY);
```

```
    }
    public void MovePlateAttackSpawn(int matrixX, int matrixY)
    {
        float x = matrixX;
        float y = matrixY;
        x *= 0.66f;
        y *= 0.66f;
        x += -2.3f;
        y += -2.3f;
        GameObject mp = Instantiate(movePlate, new Vector3(x, y, -3f),
Quaternion.identity);
        MovePlate mpScript = mp.GetComponent<MovePlate>();
        mpScript.attack = true;
        mpScript.SetReference(gameObject);
        mpScript.SetCoords(matrixX, matrixY);
    }
```

Тепер фігури можуть ходити, але ще не має послідовності ходів, що означає, що кожен з гравців може робити будь-яку кількість ходів, тому це потрібно виправити. Для цього створимо метод(«геттер»), який буде повертати створену раніше змінну «currentPlayer». Це буде звичайна змінна типу string, яка буде приймати строку з кольором гравця, який має робити хід. Після цього створимо метод «NextTurn», який буде змінювати чергу хода гравця через функцію «If», яка буде перевіряти: якщо у змінній «currentPlayer» записаний білий гравець, то потрібно змінити строку на чорного гравця після ходу, бо якщо змінити його одразу ж, гра буде змінювати гравця який має ходити кожен фрейм, що зробить гру неможливою для гри і в іншому випадку потрібно зробити зміну строки на білого гравця, щоб у наступному русі він був гравцем, який ходить. Після всіх цих доповнень, можемо почати грати в гру. Це показано на рисунку 1.20.

1.6.5 Розробка специфічних правил

Зараз потрібно додати додаткові правила, які дадуть можливість поширити функціонал гри. Першим подібним правилом буде можливість пішака ходити на дві клітини, якщо цей пішак ще не зробив жодного ходу. Для цього додамо перевірку у ту частину коду, яка відповідає за рух кожної фігури. Я не буду змінювати метод ходу пішака, замість цього трохи зміню функцію «switch case», а саме кейси білого та чорного пішака.

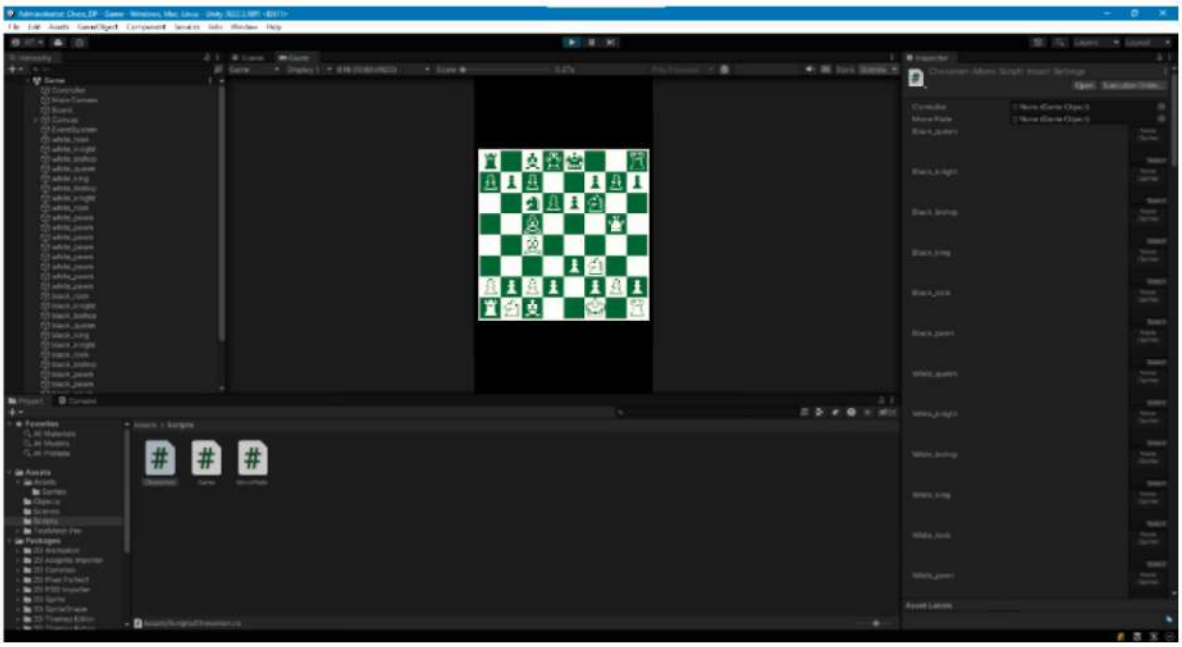


Рисунок 1.20. Вигляд гри на даному етапі

Наступним кроком потрібно додати перевірку на те, чи зробила фігура хід чи ще ні. Замість того, щоб змінювати методи кожної фігури, створювати змінну, яка буде за це відповідати та додавати її до кожної фігури можна зробити перевірку положення пішаків. Через те, що є дві різні функції для різних пішаків, чорного та білого, та те, що пішаки не можуть рухатися назад, можна зробити лише перевірку положення. Наприклад для короля та рокіровки, так зробити не можна було б, бо король міг би встати на місце його появи вже після попереднього ходу, тому цей метод буде використовуватися виключно для пішаків. Як виглядає змінений код для руху пішаків можна побачити нижче.

```
case "black_pawn":
    if (yBoard == 6)
    {
        PawnMovePlate(xBoard, yBoard - 1);
        PawnMovePlate(xBoard, yBoard - 2);
    }
    else
    {
        PawnMovePlate(xBoard, yBoard - 1);
    }
    break;
    case "white_pawn":
        if (yBoard == 1)
        {
```

```
            PawnMovePlate(xBoard, yBoard + 1);
            PawnMovePlate(xBoard, yBoard + 2);
        }
        else
        {
            PawnMovePlate(xBoard, yBoard + 1);
        }
        break;
```

Наступним елементом потрібно зробити функцію шаху, а на її основі функцію мату. В розділі формування концепту гри розглядалася ідея, а тепер потрібно лише реалізувати її. Для цього у скрипті створюємо змінну, яка буде відповідати за шах королю. Потім створюємо метод, який буде перевіряти, чи стоїть королю шах. Для цього записуємо координати короля і перевіряємо всі поля, з яких його можуть атакувати. Для цього можна використовувати ті ж самі методи ходів фігур, а саме це ходи королеви та коня, бо атака королеви це поєднання паттернів атак всіх фігур окрім коня, тому його потрібно додати окремо. Після цього створюємо дві перевірки, кожна з яких буде відповідати чорному та білому королю. Для прикладу я опишу перевірку шаху саме для білого короля. Спочатку перевіряємо його позицію з перетинанням атакуючих паттернів різних фігур і якщо ця перевірка виявила ворожу фігуру на полі з якого можна атакувати короля, то виконуються серія перевірок на те, яка саме фігура стоїть на атакуючому полі і якщо ця фігура може атакувати короля з цього поля в змінну «white_check» записуються дані про те, що білий король знаходиться під шахом. Схематичне зображення логіки шаху для білого короля показано на рисунку 1.21.

Ця ж сама логіка використовується для перевірки чорного короля для шаху, але перевірка на шах від пішака відрізняється в координатах, вони змінені на «x+1 y-1» та «x-1 y-1», бо білі пішаки ходять вверх дошки, тому перевіряти слід координати знизу короля. Це показано в коді нижче:

```
    if (sc.PositionOnBoard(x, y))
    {
        GameObject attack_place = sc.GetPosition(x, y);
        if (attack_place == x + 1, y - 1 || attack_place == x - 1, y - 1 && name ==
"black_pawn")
        {
```

```
            check = true;
        }
    }
```

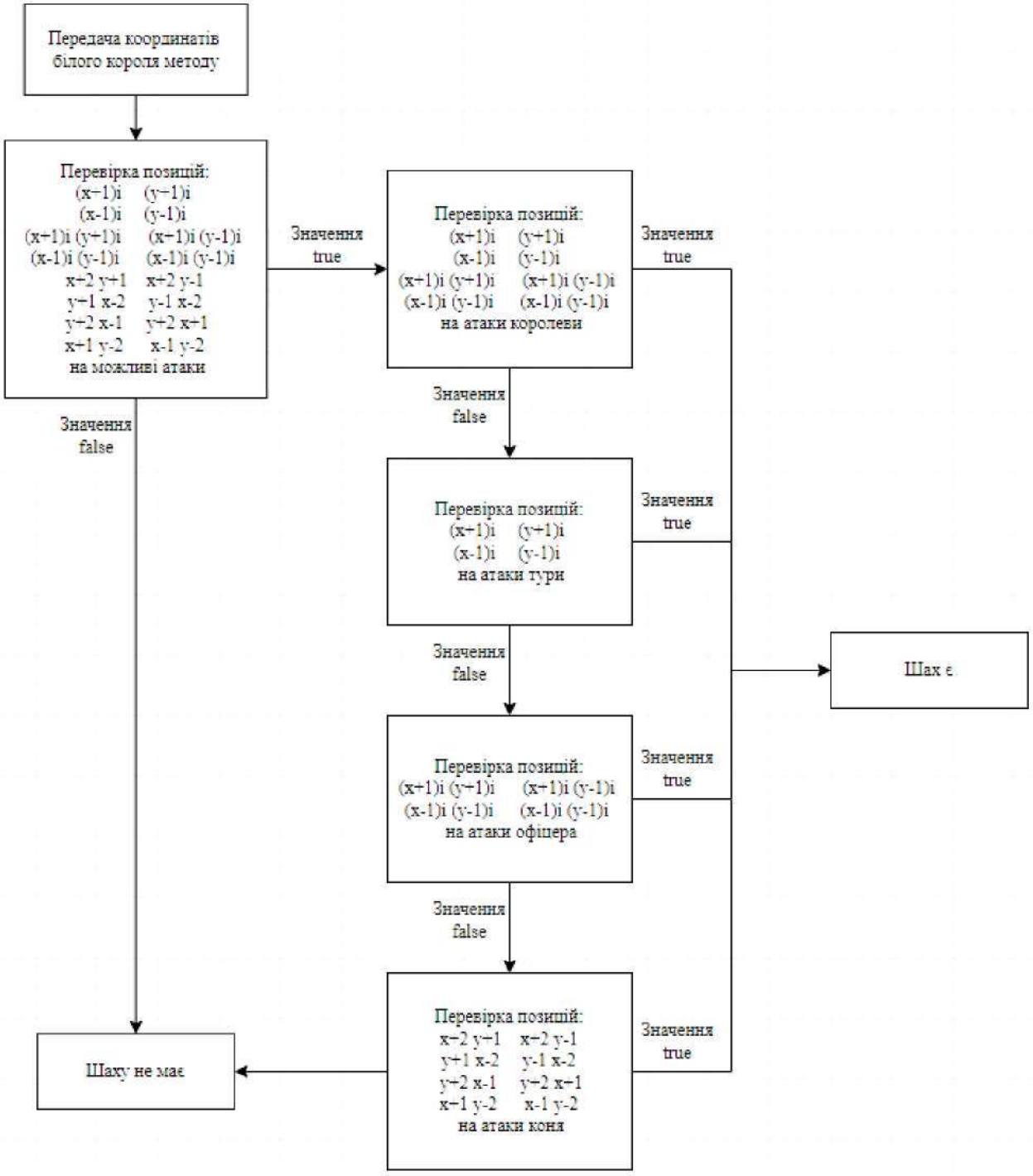


Рисунок 1.21. Схематичне зображення логіки методу для виявлення шаху для білого короля

Тепер потрібно звертатися до цього методу, щоб це зробити необхідно викликати його у методі, який є в самій бібліотеці Unity і викликається кожен фізичний фрейм. Це метод «Update». В ньому ми можемо викликати створений

метод з пошуку шаху і програма завжди буде бачити, коли одному з королів буде шах.

Тепер реалізуємо рокіровку, для цього вже маємо майже всі необхідні елементи. У програмі прописана функція шаху, яка буде перевіряти, чи можливий шах на шляху руху короля в процесі рокіровки. Тому потрібно додати змінну «isMoved» для короля та турів для того, щоб звертатися до неї і перевіряти її і розуміти робила фігура ходи до цього, чи ні. Після перевірки цієї змінної у короля, а потім у турів, можна перевірити чи є шах на шляху руху короля і якщо все добро дозволити провести рокіровку. Вигляд схеми, яка показує логіку рокіровки можна побачити на рисунку 1.22.

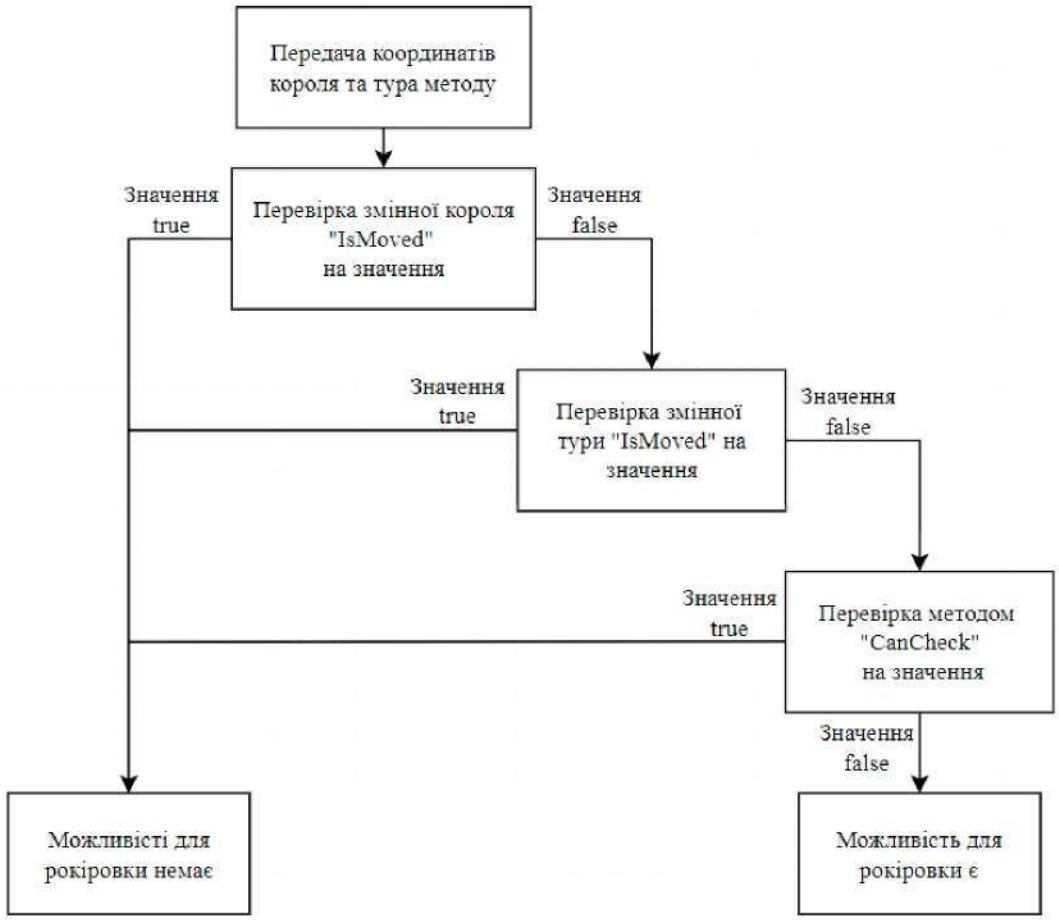


Рисунок 1.22. Схематичне зображення логіки рокіровки.

Останнім специфічним правилом буде перетворення пішака на королеву при досягненні протилежної частини дошки. Для цього знову будемо використовувати метод «Update», який буде перевіряти всі фігури на потрібній

лінії, у випадку з чорним пішаком – це перша лінія, у випадку ж з білим – це восьма. В програмі є матричне зображення дошки, тож просто перевіряємо назву фігури, яка знаходиться на потрібній лінії, і якщо це пішак, перетворимо його у королеву. Ось код який реалізує цю функцію:

```
void Update()
{
   switch (this.name)
   {
      case "black_pawn":
        if (yBoard == 0)
        {
           GameObject cp =
controller.GetComponent<Game>().GetPosition(xBoard, yBoard);
           cp.name = "black_queen";
           this.GetComponent<SpriteRenderer>().sprite = black_queen; player =
"black";
        }
        break;
        case "white_pawn":
           if (yBoard == 7)
           {
              GameObject cp =
controller.GetComponent<Game>().GetPosition(xBoard, yBoard);
              cp.name = "white_queen";
              this.GetComponent<SpriteRenderer>().sprite = white_queen; player
= "white";
        }
        break;
   }
}
```

Як видно в коді я використовую функцію «switch case», яка перевіряє назву фігури, і якщо це пішак, то перевіряє: чи знаходиться він на необхідній лінії для перетворення. У випадку з білим пішаком це значення сім, бо масив починається з нуля, тож остання лінія масиву – це сьома, а у випадку з чорним пішаком – вона нульова. І при виконанні всіх вище перерахованих умов данні про фігуру записуються у створену змінну і змінюються через цю ж змінну. Програмою змінюється ім'я цієї фігури, спрайт та переназначається її до гравця відповідного кольору.

1.6.6 Створення логіки перемоги і інтерфейсу для неї

Тепер створимо функцію перемоги. Для неї можна використати два метода: перший метод виграш при з'їданні короля і другий виграш через шах і мат. Для більшої зрозумілості порівняємо ці методи. Перший метод доволі простий і використовується в деякий шахових програмах для мобільних пристроїв. Через свою простоту і можливість зміни його на більш досконалі, деякі ігри використовують саме його. Другий метод – навпаки, більш складний та ресурсозатратний. Його використовують для більш досконалих систем та для сильних в плані працездатності пристроїв. Його мінусом є те, що кожен фрейм потрібно перевіряти чи є мат. Для перевірки шаху за стандартними, більш досконалими алгоритмами слабкі пристрої можуть використовувати більшість своїх ресурсів, а на прорахунок ще восьми таких позицій ресурсів може просто не вистачити. Через те, що пріоритетом є велика кількість потенційних гравців з мобільних пристроїв використовувати будемо саме перший метод виграшу гри.

Тепер перейдемо до реалізації виграшу. Це я зроблю у методі, який є стандартним методом Unity «OnMouseUp». Цей метод спрацьовує, коли гравець відпускає кнопку миші або перестає торкатися екрану. В ньому створюється змінна, до якої буде звертатися програма в подальшому. Тепер зробимо перевірку на з'їдання короля обох гравців. Для того, щоб не перевіряти це кожний фрейм, будемо перевіряти, чи є атака в принципі, і якщо є – робимо перевірку. Якщо перевірка дала позитивний результат, завертаємося до змінної, яка приймає рядок, в який записаний колір гравця і змінюємо її значення на необхідне. Код описаної функції поданий нижче:

```
controller = GameObject.FindGameObjectWithTag("GameController");
if (attack)
{
    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);
```

}

Після цього потрібно повністю зупинити гру і вивести якесь повідомлення. Для цього потрібно створити змінну, яка буде відповідати за кінець гри. Коли гра починається, вона буде дорівнювати false, а коли до змінної «playerWinner» надходить інформація про гравця переможця, буде змінюватися значення змінної «GameOver» з false на true.

Тепер потрібно створити спрайти з текстом про виграш гравця. Для цього я знову використовував програму для створювання зображень Adobe Photoshop. В ньому створюю три спрайти, які будуть з'являтися на екрані при перемозі. Ці спрайти показані на рисунку 1.23.

Рисунок 1.23. Створені спрайти для виклику у випадку перемоги

Спрайт, на якому написано «tap to restart» буде викликатися завжди, неважливо хто з гравців переміг, а інші спрайти будуть викликатися в залежності від гравця, який переміг, чорний чи білий.

Отримавши всі елементи, для написання коду, що буде реалізовувати виведення інформації на екран, можна перейти до написання цього методу. Ця функція буде викликатися, коли змінна «playerWinner» була змінена і після

виклику зупиняє гру. Також викликає функцію метода «SpriteRenderer» «enabled», яка дає змогу ввімкнути відображення спрайтів на сцені. Спочатку вмикається відображення спрайта «restart», а потім перевіряється, хто з гравців переміг, і від цього вмикається відповідний спрайт. Код який це реалізує наданий нижче:

```
public void Winner(string playerWinner)
{
   GameOver = true;
GameObject.FindGameObjectWithTag("RestartText").GetComponent<SpriteR
enderer>().enabled = true;
   if (playerWinner == "black")
   {
GameObject.FindGameObjectWithTag("BlackText").GetComponent<SpriteRe
nderer>().enabled = true;
   }
   if (playerWinner == "white")
   {
GameObject.FindGameObjectWithTag("WhiteText").GetComponent<SpriteR
enderer>().enabled = true;
   }
}
```

Також потрібно реалізувати метод, який буде перезапускати гру, якщо вона закінчилася. Для цього створимо перевірку в методі «Update», яка буде перевіряти дві речі: змінну «GameOver» на значення true та клік або тап по екрану від гравця. Після перевірки він змінить значення змінної «GameOver» з true на false та перезапускає сцену за допомогою вбудованого метода «LoadScene». Ось код, який реалізує дану задачу:

```
public void Update()
{
  if(GameOver == true && Input.GetMouseButtonDown(0))
  {
    GameOver = false;
    SceneManager.LoadScene("Game");
  }
}
```

На цьому етапі гра майже закінчена і в неї вже можна грати. Це показано на рисунку 1.24.

Рисунок 1.24. Готова шахова гра

1.7 Тестування працездатності ігрових елементів

У цьому розділі, я протестую елементи ігрового процесу, щоб точно зрозуміти правильність створеної гри, і якщо будуть знайдені недоліки – виправлю їх. Це включає в себе функціональне тестування, виявлення та виправлення помилок, а також оцінку продуктивності та користувацького досвіду.

Першим етапом є визначення основних цілей тестування, якими є: перевірка функціональності кожного елементу гри, забезпечення належної взаємодії між компонентами, виявлення та виправлення помилок.

Першою перевіркою буде розуміння, що всі елементи гри працюють як треба. Для цього виділимо елементи, які хочемо перевірити: дошка, фігури та інтерфейс. Для перевірки правильності функціоналу дошки можна спробувати запустити гру, почати її, програти кілька ходів, вимкнути її і знову запустити. Цей

тест дасть змогу побачити чи є залежність вимкнення гри та позицією на дошці. Очікуваним результатом цієї перевірки є повне скидання позиції гри до початкового. Цей тест показав, що код, який повинен знищувати данні про попередню партію працює правильно. Ще одним тестом є зміна роздільної здатності екрану і можливість гри для неї. Для цього змінюємо її на дві різні і запускаємо гру. Як показав тест, потрібно змінити частину кода та додати центрування та вирівнювання за границями. Як виглядає програма до виправлення можна побачити на рисунку 1.25, вигляд програми після виправлення багу можна побачити на рисунку 1.26.

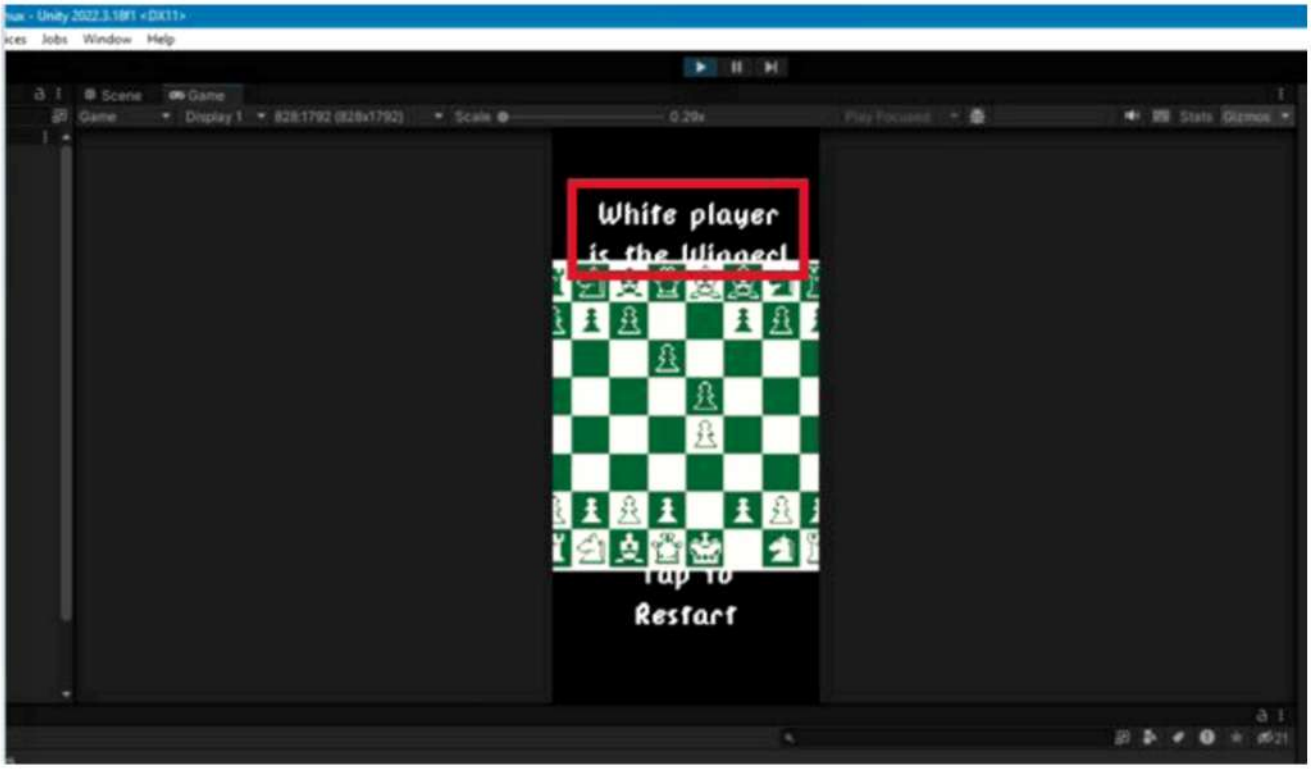


Рисунок 1.25. Програма до виправлення багу

Наступною частиною для тестування є фігури. Потрібно перевірити: чи правильно ходять всі фігури. Після всіх тестів було знайдено баг, який не давав ферзям ходити назад. Баг було виправлено після зміни у цій частині коду:

```
case "black_queen":
case "white_queen":
    LineMovePlate(1, 0);
    LineMovePlate(0, 1);
    LineMovePlate(1, 1);
    LineMovePlate(-1, 0);
//строка яка була змінена
```

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

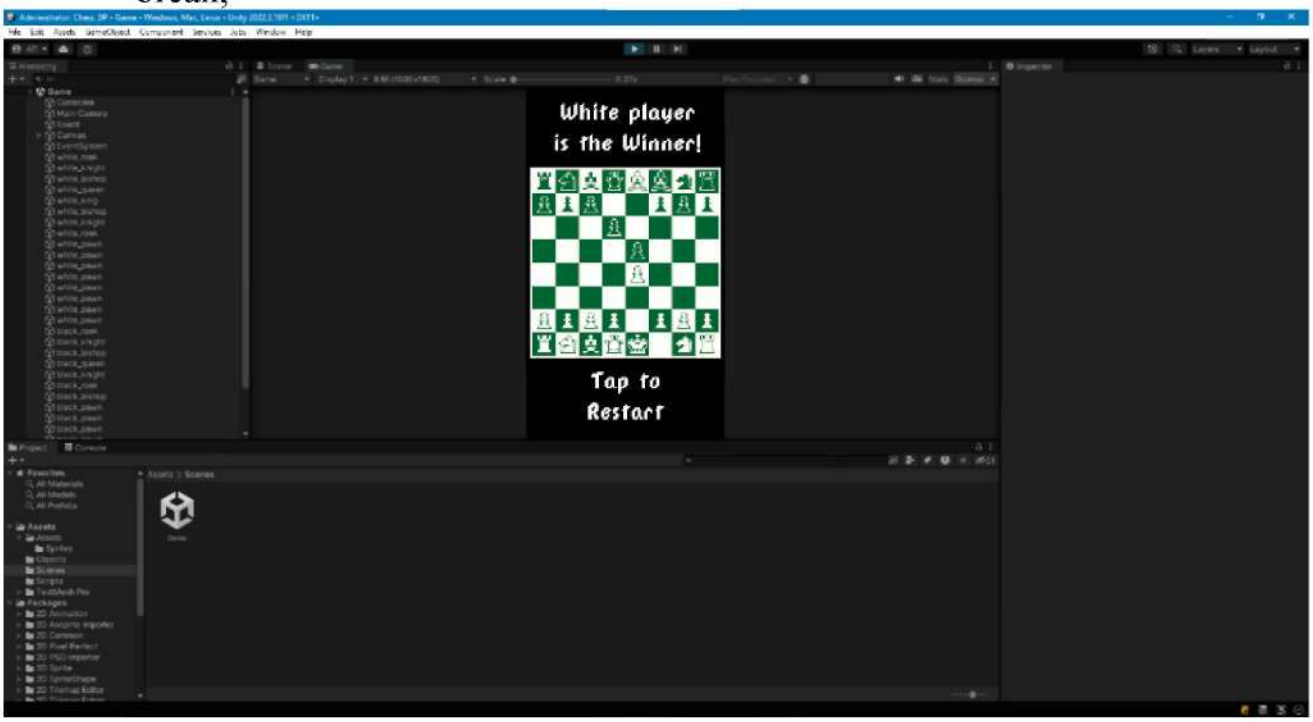


Рисунок 1.26. Програма після виправлення багу

Перед виправленням строка, відповідаюча за рух назад, робила теж саме, що і строка, яка відповідає за рух вперед.

Ще одним тестом, який був зроблений це тест перевірка, яка покаже: чи закінчується спавн клітин, які показують куди можна ходити за дошкою. Тест показав, що коли ви нажимаєте на фігуру, яка використовує метод руху «LineMovePlate» клітини з'являються, навіть коли вони можуть покинути сцену. Для виправлення цієї проблеми було проаналізовано скрипт «MovePlate» і після цього було виявлено, що помилка полягає в тому, що не було створено методу, який знищував би поля після їх виходу зі сцени. Для виправлення даної помилки було створено метод «DestroyMovePlates» в скрипті «Chessman», для подальшого виклику цього метода у попередньому скрипті. Метод знаходить всі об'єкти з тегом, який відповідає за всі підказки для гравця. Далі в ньому створений цикл який не дає клітинам з'являтися за сценою, як це показано на рисунку 1.27.

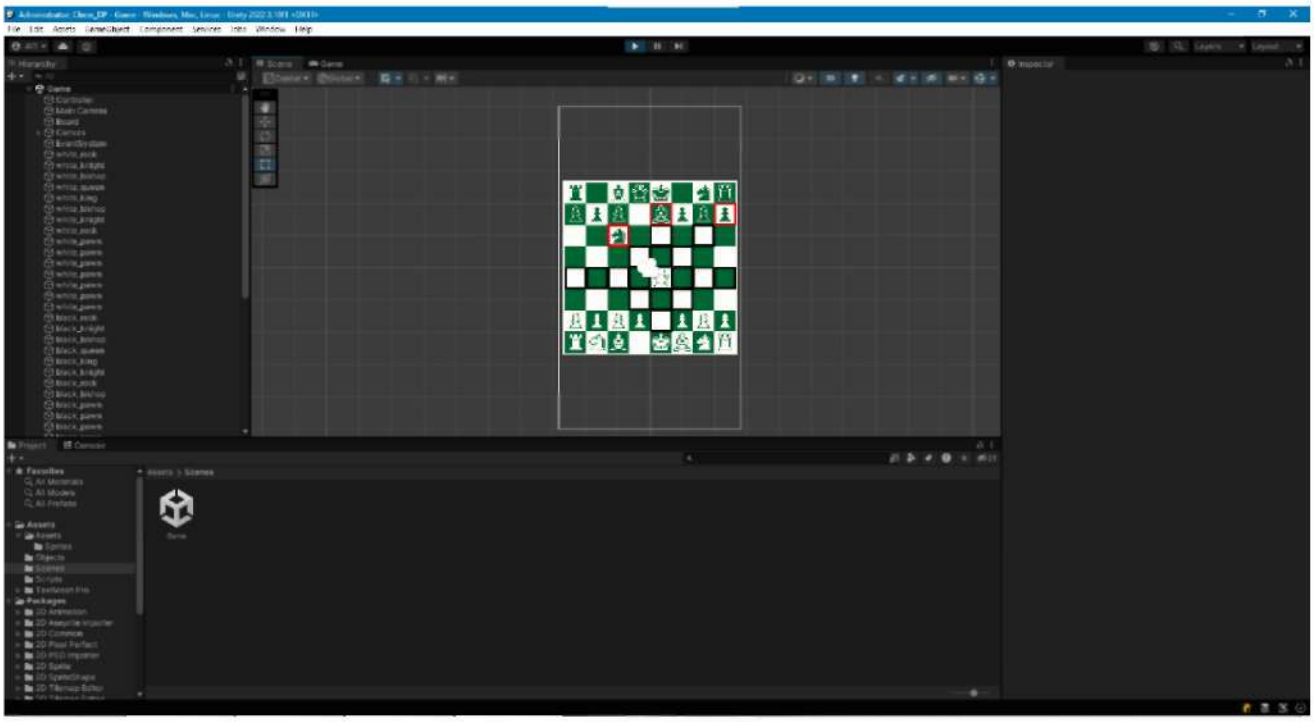


Рисунок 1.27. Кінцевість методу «LineMovePlate»

Після проведених тестів, будо виявлено та виправлено баги, які були в проекті, що показує важливість проводити тестування будь якого продукту перед закінченням його розробки.

Тепер коли проект закінчено, на рисунку 1.28 можна побачити, як виглядає схематичне зображення структури проекту.

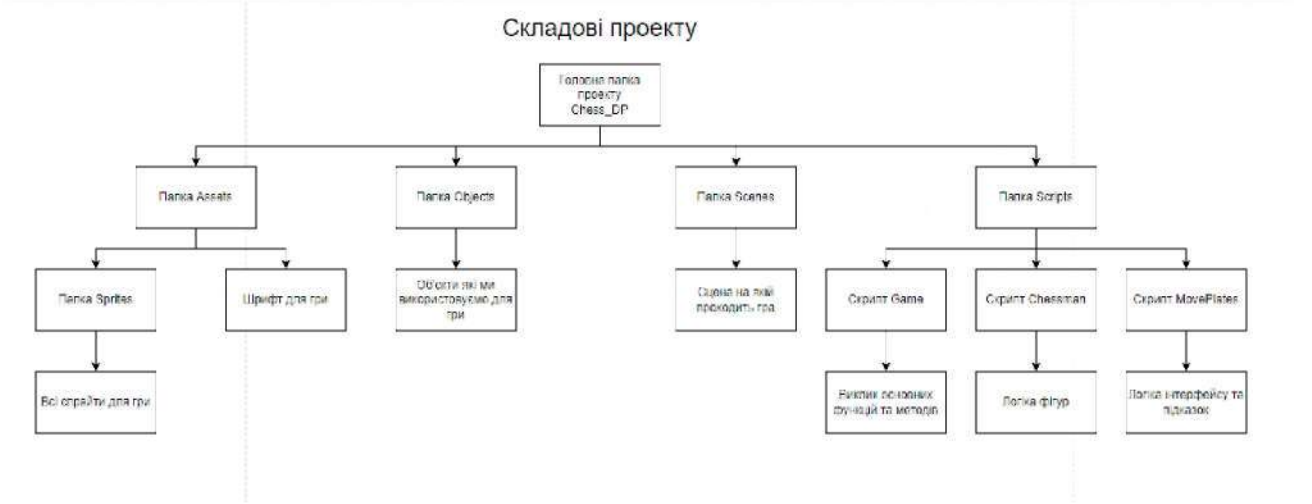


Рисунок 1.28. Схема проекту

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

В даному дипломному проекті розроблено та реалізовано початкову версію шахової гри на програмному рушії Unity. Було впроваджено основні шахові механіки, необхідні для повноцінного ігрового процесу. Виконано роботу над програмною частиною інтерфейсу та його взаємодією з ігровими елементами. Використання принципів об’єктно орієнтованого програмування дозволило створити гнучку архітектуру, яка забезпечує можливість подальшої модифікації гри, додавання нових елементів, механік або режимів.

Оцінка якості програмного продукту з точки зору користувача визначається необхідним на стадії функціонування розміром оперативної пам'яті ЕОТ, витратами машинного часу, пропускною спроможністю каналів передачі даних. Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога

Каталог аналогів

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Таблиця 2.1 Каталог аналогів програмного забезпечення

Найменування ПЗ	Обсяг функції ПП = V_0, ум. машинних команд
1. ПП СУБД	2500 – 9800
2. ПП загальної математики і ПП імітаційного моделювання	7800 – 8800
3. ПП оптимізаційних розрахунків	1300 – 4200

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.2.2

Таблиця.2.2 Трудомісткості на основі умовних машинних команд

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар = 229 х 0,7 = 160.30 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{ТЗ} = T^{a}p \times L_1 \times K_{Н} \qquad (2.1)$$

$$T_{ТП} = T^{a}p \times L_2 \times K_{Н} \qquad (2.2)$$

$$T_{РП} = T^{a}p \times L_3 \times K_{Н} \times K_{Т} \qquad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.2.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.3.);Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.4.).

Таблиця 2.3 Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПЗ	1,75 – 1,2
Б	ПЗ – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПЗ маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 2.5 Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПЗ типовими програмами, %	Значення $К_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання:

$$Т_{тз}=Та*L_1*К_н=160,3 *0,12*0,8= 15,38 \text{ (люд/годин)}$$

Трудомісткість розробки технічного проекту:

$$Т_{тп}=Та*L_2*К_н=160,3 *0,11*0,8 = 14,11 \text{ (люд/годин)}$$

Трудомісткість розробки робочого проекту:

$$Т_{рп}=Та*L_3*К_н*К_т=160,3 *0,61*0,8*0,8= 62,58 \text{ (люд/годин)}$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}=2$ (стр), розробка ТП $N_{тп}=16$(стр), розробка робочого проекту $N_{рп}=15$ (стр), пояснювальна записка відповідно $N_{пз}=51$ (стр) Розрахунок зведений у таблицю 2.6

Таблиця 2.6 Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Ртз}$=15,38	$T_{кк}=0,7*N_{тз}$= 0,7*2=1,4	$T_{нк}=0,15*N_{тз}$=0,15*2=0,30
2.Розробка ТП	$T_{Ртп}$=14,11	$T_{кк}=0,7*N_{тп}$=0,7*16=11,2	$T_{нк}=0,15*N_{тп}$=0,15*16=2,4
3.Розробка РП	$T_{Ррп}$=62,58	$T_{кк}=0,7*N_{рп}$=0,7*15=10,5	$T_{нк}=0,15*N_{рп}$=0,15*15=2,25
4.Розробка ПЗ	$T_{пз}=1,5**N_{пз}$=1,5*51 =76,5	$T_{кк}=0,7*N_{тз}$=0,7*51=35,7	$T_{нк}=0,15*N_{пз}$=0,15*51 =7,65
Усього, в т.ч.:	225,99		
- на розробку	$\Sigma T_р$=160,19		
- контроль керівника		$\Sigma T_{кк}$= 54,2	
- нормоконтроль			$\Sigma T_{нк}$=11,6

2.3 Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПЗ. Розрахунок основної заробітної плати виконавців приведений у таблиці 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2022» встановлено мінімальну заробітну плату у місячному розмірі з 1 квітня 2024 року - 8000 гривень; мінімальну погодинну тарифну ставку – 48 грн.

Таблиця 2.7 Розрахунок основної заробітної плати виконавців

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	160,19	48	7689,12
2.Контроль керівника	54,2	68,10	3691,02
3.Нормоконт-роль	11,6	68,10	789,96
Усього	-	-	ΣЗо= 12170,10

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 2.8

Таблиця 2.8 Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	80	3,50	280
Разом	-	-	-	$В_{м\,і}$=280,0
Транспортно– заготівельні витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$= 0,1х280.0=28,0
Усього				$В_м=В_{мі}+В_{тр\ з}$= 308,0

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 2.9.

Таблиця 2.9 Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	308,0	$В_м$ (див. табл. 2.8)
2. Основна заробітна плата	12170,10	$З_о$ (див. табл. 2.7)
3.Додаткова заробітна плата	1217,01	$Зд = 0{,}15 \times Зо$=12170,10х0,1
4.Відрахування до єдиного фонду соціального внеску	2945,16	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$=0,22х (12170,10+ 1217,01)
5. Накладні витрати	7302,06	$Внак. = 0{,}6 \times Зо$ =0,6 х 12170,10
6. Повна собівартість	23942,33	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 308,0+12170,10+1217,01+2945,16+ 7302,06

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$П=(С_п*Р)/100$= (23942,33*10)/100 = 2394,23грн (2.4)

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о = С_п+П$ =23942,33+2394,23= 26336,56грн (2.5)

Податок на додану вартість визначаємо по наступній формулі:

ПДВ = 0.2 * $Ц_о$ = 26336,56 *0,2 = 5267,31грн; (2.6)

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$Ц_р=Ц_о$+ПДВ = 26336,56 + 5267,31 = 31603,87 грн (2.7)

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

3.1 Вступ

Охорона праці є одним з найважливіших соціальних чинників і важливу роль відіграє на підприємстві оскільки, якими б важливими не були трудові здобутки, вони не можуть компенсувати людині втраченого здоров'я, а тим більше життя.

Розробка гри потребує дотримання певних умов праці програміста, включаючи підтримку оптимальної робочої пози та режиму праці й відпочинку, особливо під час роботи з комп'ютером, яка характеризується значною розумовою напругою, нервово-емоційним навантаженням, високою напруженістю зорової роботи та достатньо великим навантаженням на м'язи рук при роботі з клавіатурою та мишею.

Згідно з ч. 1 ст. 13 Закону України «Про охорону праці», роботодавець зобов'язаний створити на робочому місці умови праці відповідно до нормативно-правових актів, а також забезпечити дотримання вимог законодавства щодо прав працівників у галузі охорони праці. Дотримання норм охорони праці є спільним завданням як роботодавця, так і працівника. У вирішенні питань з охорони праці можна звернутися до відповідного законодавства України.

Даний розділ присвячено вивченню оптимальних умов праці програміста під час розробки шахової гри на Unity та забезпеченню дотримання норм охорони праці. Це є важливим аспектом успішної реалізації проекту, оскільки здоров'я та ефективність працівників безпосередньо впливають на якість кінцевого продукту.

3.2 Аналіз небезпечних та шкідливих чинників, що впливають на працівника

У розробці шахової гри на програмному рушії Unity можуть виникати небезпечні та шкідливі чинники, що впливають на здоров'я та ефективність працівників. Визначення та аналіз цих чинників є ключовими для забезпечення безпечних умов праці та мінімізації ризиків. Основні небезпечні та шкідливі фактори, які варто враховувати:

Неправильна організація робочого місця: неправильна конструкція та розташування елементів робочого місця можуть призвести до тривалого напруження м'язів спини, шиї та рук, спричиняючи ризик розвитку професійних захворювань, таких як остеохондроз або тунельний синдром.

Недостатнє освітлення: недостатнє або неправильно налаштоване освітлення може викликати напругу зору, що призводить до швидкої втоми очей, головного болю та зниження працездатності.

Тривала робота за комп'ютером: тривале сидіння за комп'ютером без перерв може призвести до загального фізичного виснаження, проблем із зором (комп'ютерний зоровий синдром), а також до порушень кровообігу.

Недостатня вентиляція: погана вентиляція у приміщенні може призвести до зниження якості повітря, що викликає головний біль, втому та зниження концентрації.

Температурний режим: невідповідний температурний режим (занадто висока або низька температура) може негативно впливати на комфорт та продуктивність працівників.

3. 3 Розробка заходів з охорони праці

3.3.1 Вимоги до приміщення

Вимоги до приміщень, де проводиться робота з відеодисплейними терміналами (ВДТ) та персональними електронно-обчислювальними машинами (ПЕОМ), визначаються нормативно-правовими актами, зокрема, ДСанПІН 3.3.2.007-98.

- Відповідність нормативним вимогам: приміщення для роботи з ВДТ та ПЕОМ мають відповідати вимогам ДСанПІН 3.3.2.007-98. Розміщення робочих місць у підвальних та цокольних приміщеннях заборонено.
- Площа та об'єм приміщення: на кожне робоче місце повинна припадати площа не менше 6,0 м², а об'єм приміщення – не менше 20,0 м³. Це забезпечує комфортні умови праці та допомагає знизити ризик професійних захворювань

- Побутові приміщення: при приміщеннях повинні бути обладнані побутові приміщення для відпочинку, які дозволяють працівникам ефективно відновлювати сили під час перерв.

3.3.2 Гігієнічні вимоги

Вологе прибирання: у приміщеннях слід щоденно проводити вологе прибирання для підтримання чистоти та запобігання розповсюдженню пилу та алергенів, що можуть негативно впливати на здоров'я працівників.

Аптечки першої допомоги: приміщення повинні бути оснащені аптечками першої медичної допомоги, які містять необхідні медикаменти та засоби для надання першої допомоги у випадку нещасних випадків.

3.3.3 Вимоги до кольорового оформлення

Використання кольорів: для стін і робочих поверхонь рекомендується використовувати мало насичені (основні) кольори. Це допомагає створити комфортну робочу атмосферу та знижує навантаження на зір.

Контрастність і акценти: для невеликих ділянок або поверхонь, що рідко потрапляють у поле зору, можна використовувати кольори середньої насиченості (допоміжні). Для маленьких площ поверхонь використовують насичені кольори (акценти) для функціонального фарбування.

Відблиски та матовість: поверхні обладнання у приміщеннях повинні бути матовими або напівматовими, щоб виключити відблиски світла в очі працюючих. Стіни повинні бути пофарбовані фарбами пастельних тонів, що запобігає створенню відблисків та знижує навантаження на зір.

3.3.4 Вентиляція та мікроклімат

Вентиляція: приміщення повинні бути оснащені ефективною вентиляційною системою для забезпечення постійного обміну повітря, що сприяє підтриманню здорового мікроклімату та знижує концентрацію шкідливих речовин у повітрі.

Температурний режим: температура в приміщенні повинна підтримуватись на рівні, комфортному для працівників, зазвичай у межах 18-22°C.

Відносна вологість: відносна вологість повітря у приміщеннях для роботи з ВДТ та ПЕОМ повинна підтримуватись на оптимальному рівні для забезпечення комфортних умов праці та запобігання виникненню негативних наслідків для здоров'я працівників. Оптимальний рівень відносної вологості повітря становить 40-60%.

3.3.5 Організація робочого місця користувача ПК

Організація робочого місця користувача ПК є важливим аспектом, що впливає на ефективність роботи, здоров'я та комфорт працівника. Правильно організоване робоче місце знижує ризик виникнення професійних захворювань та підвищує продуктивність.

- Розташування обладнання: висота та кут нахилу монітора повинні забезпечувати оптимальне положення для очей користувача, щоб уникнути надмірного напруження шиї та зору. Розташування клавіатури ти миші повинно бути таким, щоб уникнути зайвих рухів та забезпечити комфортне положення рук.
- Ергономіка робочого місця: висота, підтримка спини та підлокітники крісла мають значення для забезпечення правильної позиції тіла та уникнення м'язового напруження.

3.3.6 Електробезпека

Забезпечення електробезпеки на робочому місці є ключовим аспектом для запобігання нещасних випадків та забезпечення безпеки працівників.

- Використання електрообладнання: перед використанням слід перевірити електроінструменти та обладнання на наявність пошкоджень чи ознак зносу. Пошкоджене обладнання повинно бути негайно замінене або відремонтоване. Важливо використовувати заземлені розетки та обладнання для захисту від перенапруг та уникнення уражень струмом.
- Працівники повинні мати необхідні знання та навички з електробезпеки, а також дотримуватися правил безпеки при роботі з електричними приладами. Робота з електрообладнанням у вологому середовищі є

небезпечною. Приміщення, де використовується електрообладнання, повинні бути сухими.

- Робочі приміщення повинні бути обладнані вогнегасниками, а працівники повинні бути навчені їх використовувати в разі виникнення пожежі. Необхідно уникати зберігання запальних матеріалів поруч з електрообладнанням.

3.4 Пожежна безпека

Пожежна безпека охоплює можливість виникнення та поширення пожежі в будь-яких умовах та середовищах. Перебуваючи в зоні пожежі, людина стикається з ризиком впливу наступних небезпечних факторів: токсичні продукти згорання, вогонь, підвищена температура, дим, недостатність кисню, руйнування будівельних конструкцій, вибухи, паніка.

Усі працівники повинні бути навчені користуватись вогнегасниками та іншими першими засобами пожежогасіння, а також знати їх місце розташування. Первинні засоби пожежогасіння включають вогнегасники, пожежний інвентар та пожежний інструмент. Пожежні щити (стенди) мають бути встановлені на території об’єкта з розрахунком одного щита на площу 5000 м².

Ящики для піску повинні мати об'єм 0,5, 1,0 або 3,0 м³ та бути укомплектовані совковою лопатою. Вибір та кількість вогнегасників визначаються відповідно до типових норм, затверджених відповідними органами. Евакуаційні шляхи та виходи мають бути вільними та не захаращеними згідно з вимогами національних нормативних документів.

Евакуаційні шляхи та запасні виходи повинні маркуватися та дотримуватися норм відповідно до НАПБ А.01.001 2004, бути доступними, відкритими або ж закритими не на ключ, а на засов, який можуть відкрити у разі виникнення пожежі.

ВИСНОВКИ

Метою розробки цього проекту було створення шахової гри з використанням ігрового рушія Unity.

Для виконання цієї задачі були досліджені інші шахові проекти, такі як: Chess.com, Play Magnus та Lichess. На основі цього дослідження було виділено аспекти, які повинні бути у грі. За допомогою аналізу цих програм було позначено курс подальшої розробки.

Розробка велася в умовно безкоштовному рушії Unity, який в порівнянні з іншими виявився найкращим для даного проекту. Інтегроване середовище розробки в якому велося програмування гри було Microsoft Visual Studio 2023 з доданими бібліотеками для роботи з Unity.

Процес розробки почався формування концепту гри та проектування її модулів. В процесі формування концепту було вирішено, що основним пристрієм для гри буде телефон. Були описані деякі правила, які в подальшому були реалізовані вже в коді. В процесі проектування проекту було вирішено, що гра буде у форматі 2D, для більш широкого охвату гравців на мобільних пристроях. На цьому етапі також було що спрайти будуть створені в програмі Adobe Photoshop 2023. В подальшому почався процес розробки гри на основі попередніх досліджень та проектування елементів. Були створені схематичні функціональні блок-схеми, за якими було створені частки коду. В подальшому розробка велася згідно завдань які були поставлені на етапі проектування.

В результаті роботи було створено початкову версію шахової гри з використанням головних ігрових механік, які були зазначені на етапі огляду схожих проектів та методик об’єктно орієнтованого програмування яке може дати змогу додання функцій до гри.

Також потрібно зазначити, що проект має потенціал для подальшого розвитку, а саме: можливість додання комп’ютерів різної складності для гри, мультиплеєру та лідербордів. Підтримка такого проекту може стати наступним шагом для реалізації себе в галузі інформаційних технологій.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Ерік Фрімен, Елізабет Робсон, Берт Бейтс, Кеті Сієрра, Книга Head First. Патерни проектування, Київ, Фабула, 672 ст.
2. Коноваленко І.В., Програмування мовою C# 6.0, Тернопіль, ТНТУ, 2016 р., 227 ст.
3. Nuclino: [Website]. URL: https://www.nuclino.com/articles/game-design-document-template (viewed on: 04.06.2024).
4. Microsoft learn: [Website]. URL: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/identifier-names (viewed on: 04.06.2024).
5. Microsoft learn: [Website]. URL: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/covariance-contravariance/ (viewed on: 04.06.2024).
6. Unity Documentation: [Website]. URL: https://docs.unity3d.com/Manual/GameObjects.html (viewed on: 04.06.2024).
7. Unity Documentation: [Website]. URL: https://docs.unity3d.com/Manual/CrossPlatformConsiderations.html (viewed on: 04.06.2024).
8. Unity Documentation: [Website]. URL: https://docs.unity3d.com/Manual/Constraints.html (viewed on: 04.06.2024).
9. Docs.unity3d: [Website]. URL: https://docs.unity3d.com/Packages/com.unity.simulation.games@0.4/manual/index.html (viewed on: 04.06.2024).
10. Uiprep: [Website]. URL: https://www.uiprep.com/blog/the-best-way-to-document-ux-ui-design (viewed on: 04.06.2024).

ДОДАТОК А. Лістинг коду скриптів гри мовою C#

Скрипт Game.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using static Unity.Collections.AllocatorManager;
public class Game : MonoBehaviour
{
    public GameObject Chess_piese;
    // Позиції та положення фігур
    private GameObject[,] positions = new GameObject[8, 8];
    private GameObject[] playerBlack = new GameObject[16];
    private GameObject[] playerWhite = new GameObject[16];
    private string currentPlayer = "white";
    private bool GameOver = false;
    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_knight",6,7), Create("black_rook",7,7),
            Create("black_bishop",5,7), Create("black_pawn", 0, 6),
Create("black_pawn", 1, 6), Create("black_pawn", 2, 6),
            Create("black_pawn", 3, 6), Create("black_pawn", 4, 6),
Create("black_pawn", 5, 6),
            Create("black_pawn", 6, 6), Create("black_pawn", 7, 6) };
        //Проставлення позицій всіх фігур
        for (int i = 0; i < playerWhite.Length; i++)
        {
```

```
            SetPosition(playerWhite[i]);
            SetPosition(playerBlack[i]);
        }
    }

    public GameObject Create(string name, int x, int y)
    {
        GameObject obj = Instantiate(Chess_piese, new Vector3(0, 0, -1),
Quaternion.identity);
        Chessman cm = obj.GetComponent<Chessman>();
        cm.name = name;
        cm.SetXBoard(x);
        cm.SetYBoard(y);
        cm.Activate();
        return obj;
    }

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

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

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

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

    public string GetCurrentPlayer()
    {
        return currentPlayer;
```

```
        }

        public bool IsGameOver()
        {
            return GameOver;
        }

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

        public void Update()
        {
            if(GameOver == true && Input.GetMouseButtonDown(0))
            {
                GameOver = false;
                SceneManager.LoadScene("Game");
            }
        }

        public void Winner(string playerWinner)
        {
            GameOver = true;
    GameObject.FindGameObjectWithTag("RestartText").GetComponent<SpriteR
enderer>().enabled = true;
            if (playerWinner == "black")
            {
    GameObject.FindGameObjectWithTag("BlackText").GetComponent<SpriteRe
nderer>().enabled = true;
            }
            if (playerWinner == "white")
            {
    GameObject.FindGameObjectWithTag("WhiteText").GetComponent<SpriteRe
nderer>().enabled = true;
            }
```

```
    }
}
```

Скрипт Chessman.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Xml.Linq;
using UnityEngine;
public class Chessman : MonoBehaviour
{
    // Референси
    public GameObject controller;
    public GameObject movePlate;
    // Позиції
    private int xBoard = -1;
    private int yBoard = -1;
    // Стартовий гравець
    private string player;
    //Задання спрайтів
    public Sprite black_queen, black_knight, black_bishop, black_king,
black_rook, black_pawn;
    public Sprite white_queen, white_knight, white_bishop, white_king,
white_rook, white_pawn;
    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;
            }
        }

        public void SetCoords()
        {
            float x = xBoard;
            float y = yBoard;
            x *= 0.66f;
            y *= 0.66f;
            x += -2.3f;
            y += -2.3f;
            this.transform.position = new Vector3(x, y, -1.0f);
        }
        // Геммеры
        public int GetXBoard()
        {
            return xBoard;
        }
        public int GetYBoard()
        {
            return yBoard;
        }
        //Семмеры
        public void SetXBoard(int x)
        {
            xBoard = x;
        }
        public void SetYBoard(int y)
        {
            yBoard = y;
        }

        private void OnMouseUp()
```

```
        {
            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":
            if (yBoard == 6)
            {
                PawnMovePlate(xBoard, yBoard - 1);
                PawnMovePlate(xBoard, yBoard - 2);
            }
            else
            {
                PawnMovePlate(xBoard, yBoard - 1);
            }
            break;
        case "white_pawn":
            if (yBoard == 1)
            {
                PawnMovePlate(xBoard, yBoard + 1);
                PawnMovePlate(xBoard, yBoard + 2);
            }
            else
            {
                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()
    {
        PointMovePlate(xBoard, yBoard + 1);
        PointMovePlate(xBoard, yBoard - 1);
        PointMovePlate(xBoard - 1, yBoard - 1);
        PointMovePlate(xBoard - 1, yBoard - 0);
        PointMovePlate(xBoard - 1, yBoard + 1);
        PointMovePlate(xBoard + 1, yBoard - 1);
        PointMovePlate(xBoard + 1, yBoard - 0);
        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 void PawnMovePlate(int x, int y)
{
    Game sc = controller.GetComponent<Game>();

    if (sc.PositionOnBoard(x, y))
    {
        if (sc.GetPosition(x, y) == null)
        {
            MovePlateSpawn(x, y);
        }
    }
    // Перевірка діагоналей
    if (sc.PositionOnBoard(x + 1, y))
    {
        GameObject cp = sc.GetPosition(x + 1, y);
        if (cp != null && cp.GetComponent<Chessman>().player != player)
        {
            MovePlateAttackSpawn(x + 1, y);
        }
    }
    if (sc.PositionOnBoard(x - 1, y))
    {
        GameObject cp = sc.GetPosition(x - 1, y);
        if (cp != null && cp.GetComponent<Chessman>().player != player)
        {
            MovePlateAttackSpawn(x - 1, y);
        }
    }
}

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

```
            x *= 0.66f;
            y *= 0.66f;
            x += -2.3f;
            y += -2.3f;
            GameObject mp = Instantiate(movePlate, new Vector3(x, y, -3f),
Quaternion.identity);
            MovePlate mpScript = mp.GetComponent<MovePlate>();
            mpScript.SetReference(gameObject);
            mpScript.SetCoords(matrixX, matrixY);
        }

        public void MovePlateAttackSpawn(int matrixX, int matrixY)
        {
            float x = matrixX;
            float y = matrixY;
            x *= 0.66f;
            y *= 0.66f;
            x += -2.3f;
            y += -2.3f;
            GameObject mp = Instantiate(movePlate, new Vector3(x, y, -3f),
Quaternion.identity);
            MovePlate mpScript = mp.GetComponent<MovePlate>();
            mpScript.attack = true;
            mpScript.SetReference(gameObject);
            mpScript.SetCoords(matrixX, matrixY);
        }

        void Update()
        {
            switch (this.name)
            {
                case "black_pawn":
                    if (yBoard == 0)
                    {
                        GameObject cp =
controller.GetComponent<Game>().GetPosition(xBoard, yBoard);
                        cp.name = "black_queen";
                        this.GetComponent<SpriteRenderer>().sprite = black_queen;
player = "black";
                    }
                    break;
                case "white_pawn":
                    if (yBoard == 7)
```

```
                {
                    GameObject cp =
controller.GetComponent<Game>().GetPosition(xBoard, yBoard);
                    cp.name = "white_queen";
                    this.GetComponent<SpriteRenderer>().sprite = white_queen;
player = "white";
                }
                break;
            }
        }
    }
```

Скрипт MovePlate.cs

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;
    public class MovePlate : MonoBehaviour
    {
        public GameObject controller;
        GameObject reference = null;
        // позиції на дошці(не у сцені)
        int matrixX;
        int matrixY;
        // false: хід; true: атака;
        public bool attack = false;
        public void Start()
        {
            if (attack)
            {
                // зміна кольору
                gameObject.GetComponent<SpriteRenderer>().color = new
Color(1.0f, 0.0f, 0.0f, 1.0f);
            }
        }

        public void OnMouseUp()
        {
            controller = GameObject.FindGameObjectWithTag("GameController");
            if (attack)
            {
                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.GetCompone
nt<Chessman>().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 мультимедійної презентації

Розробка шахової гри на програмному рушії Unity

ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА ГРУПИ 4РП-07: ПАВЛЮКА ОЛЕКСІЯ ПАВЛОВИЧА

КЕРІВНИК: ДЖАБРАЇЛОВ Д.В.

Існуючі шахові ігри, алгоритми та їх призначення

Популярні шахові програми:

- Chess.com

Одна з найпопулярніших платформ для гри в шахи онлайн.

- Lichess

Безкоштовна платформа для гри в шахи з відкритим вихідним кодом.

- Play Magnus

Один з найпопулярніших мобільних додатків для гри в шахи з комп'ютером

Існуючі шахові ігри, алгоритми та їх призначення

Основними алгоритмами шахових програм є:

- Алгоритм Мінімакс

Оптимізує хід, мінімізуючи максимальні втрати.

- Алгоритм Альфа-Бета відсікання

Скорочує кількість перевірених вузлів у дереві рішень.

- Алгоритм Монте-Карло

Використовує випадкові симуляції для оцінки ефективності ходів.

Огляд цих алгоритмів дав змогу проаналізувати елементи гри і створити стратегію, за якою були створені алгоритми та методи для логіки фігур

Огляд шахових правил та причини обрання класичних шахів

Були розглянуті різні варіанти шахових правил та після їх огляду були вибрані саме класичні.

Це рішення зроблене на основі цих причин:

- Правила повинні бути простими та зрозумілими
- Шахи повинні підходити під девайс для гри
- Розповсюдженість і популярність

Особливості програмного рушія Unity та причини його вибору для розробки проекту

Unity є доволі простим та популярним програмним рушієм. Але при виборі рушія були такі вимоги:

- Простота в розробці
- Велике ком'юніті
- C# як основна мова програмування
- Можливість кроссплатформеної підтримки

За всіма цими критеріями була вибрана саме Unity. Вона дала змогу використовувати деякі вже створені методи та можливість скомпонувати гру для мобільних девайсів

Особливості ігрових механік розроблюємого проекту

Після аналізу схожих проектів були зроблені висновки, що до механік проекту:

Проект використовував стандартні шахові правила із деякими змінами. Основною зміною буде можливість з'їдання короля.

Були додані підказки з можливими ходами гравців.

Також були додані специфічні правила, такі як: перетворення в королеву та хід на дві клітини пішаком

Огляд основних елементів ігрового процесу

Для створення хоча і простої й початкової версії гри потрібно створити деякі елементи для гри перед їх імплементацією в саму гру.

Для гри були створені спрайти дошки та фігур, так само як і спрайти написів хто саме переміг.

Були знайдені різні ассети шрифтів та можливих варіацій логічних схем

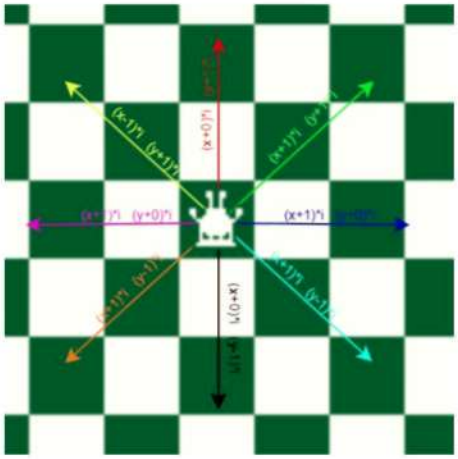
Принцип роботи основних елементів ігрового процесу

Розробка функції шаху був одним з найважливіших логічних елементів гри.

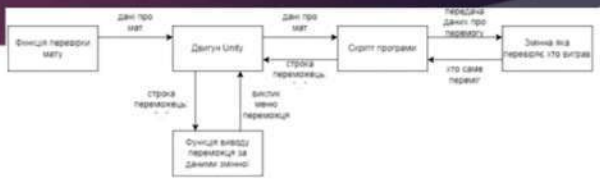
Була створена функції рокіровки.

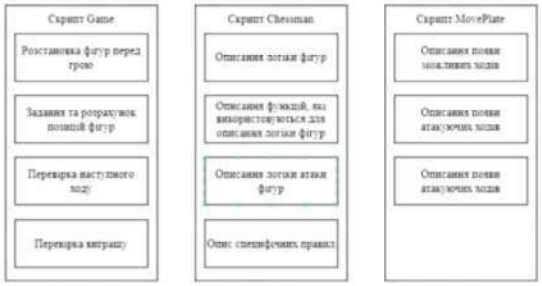
Ще одним важливим елементом проектування то програмування гри, було створити математичне зображення руху фігур.

Ще на початку проектування було зрозуміло, як буде проводитися розробка програми: які в ній будуть компоненти і як вони будуть взаємодіяти



Принцип роботи основних елементів ігрового процесу

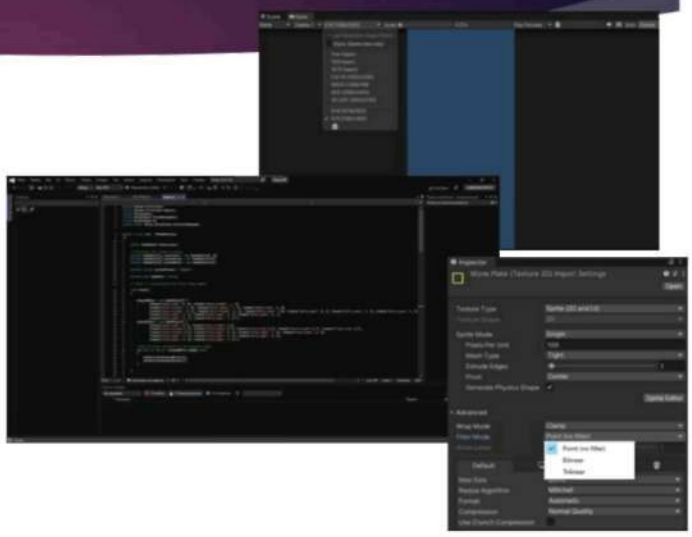




Етапи реалізації основних елементів ігрового процесу

Для реалізації задумки гри, були виконані наступні кроки:

- Створення проекту в Unity
- Додання та налаштування об'єктів
- Написання скриптів
- Прив'язка скриптів до об'єктів
- Реалізація базового інтерфейсу
- Тестування та виправлення знайдених багів

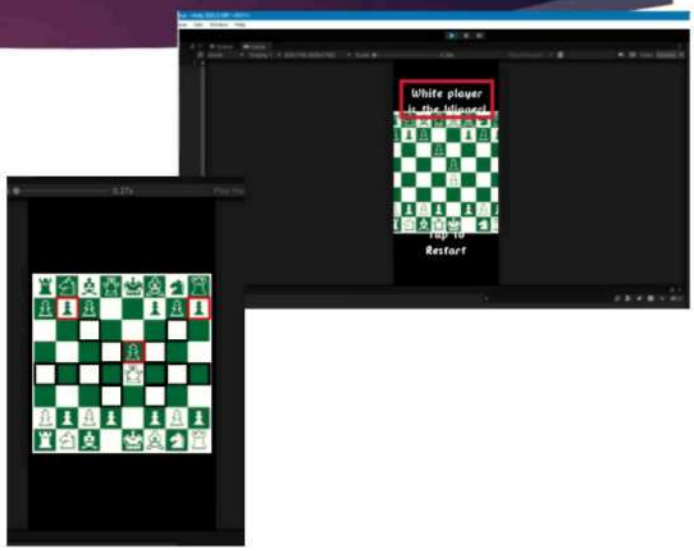


Хід тестування гри

На етапі тестування були знайдені деякі баги, які були виправленні після їх знахідки:

Було виправлено баг з розташуванням елементів на сцені при зміні роздільної здатності екрану

Було виправлено баг з неможливістю королеви ходити назад.



Скріншоти готової гри

Після розробки та тестування гра була завершена.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Павлюка Олексія Павловича

(прізвище, ім`я та по батькові)

Спеціальність: 121 "Інженерія програмного забезпечення"

Освітня програма: «Розробка програмного забезпечення»

Тема дипломного проекту: Розробка шахової гри на програмному рушії Unity

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 79 сторінок. У пояснювальній записці виконано опис предметної області, способи реалізації шахової гри для двох гравців. Проведено проектування та реалізація елементів шахової гри. Графічна частина складається з 12 слайдів мультимедійної презентації, які передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.

б) самостійність роботи над проектом: Протягом всього строку дипломного проектування та переддипломної практики здобувач освіти Павлюк О.П. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.

в) теоретична підготовка випускника (випускниці): Здобувач освіти Павлюк О.П. під час роботи над дипломним проектом вивчив достатню кількість літературних джерел та матеріалів за даною тематикою.

Вважаю, що теоретична підготовка дипломника добра і він готовий до захисту дипломного проекту

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проектування здобувач освіти Павлюк О.П. мав змогу самостійно приймати рішення з реалізації елементів шахової гри, та показав вміння організовано працювати над поставленим завданням, складати схеми та проводити розробку коду за допомогою актуальних для теми комп'ютерних програмних засобів.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Відмінно

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові керівника дипломного проекту

Джабраілов Дмитро Володимирович

Місце роботи і посада керівника дипломного проекту

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач комісії комп'ютерних технологій та програмної інженерії

Підпис

«10» 06 2024 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Павлюка Олексія Павловича

(прізвище, ім`я та по батькові)

Спеціальність 121 " Інженерія програмного забезпечення "

Освітня програма «Розробка програмного забезпечення»

Керівник дипломного проекту (роботи) Джабраілов Дмитро Володимирович

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) Розробка шахової гри на програмному рушії Unity

Обсяг розрахунково-пояснювальної записки 79 сторінок

Обсяг графічної (презентаційної) частини 12 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлений на рецензію дипломний проект повністю відповідає меті проектування та технічному завданню. Тематика дипломного проекту є актуальною для своєї галузі та присвячена питанням створення ігрових продуктів в цілому та розробці ігор на ігровому програмному рушії Unity, в цілому.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Дипломний проект складається зі вступу, трьох розділів, висновків, переліку використаних джерел. У технологічному розділі розглянуті питання проблематики розробки гри на ігровому програмному рушії Unity, сформовано вимоги до гри згідно до теми дипломного проекту та завданню, виконано проектування основних аспектів розробляємої гри. За допомогою відповідного програмного забезпечення реалізовані всі намічені зміни до ігрового процесу

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана акуратно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Детально описано процес виконання розробки гри;
2. Виконано проектування елементів гри із поясненнями на схемах та за допомогою коду;
3. Розроблено функціонуючу гру за допомогою відповідних інструментів розробки.

д) основні недоліки дипломного проекту (роботи)

1. У поточній версії гра має лише режим гри «людини з людиною»;
2. Бажано передбачити режим навчання у грі

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Відмінно

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові рецензента Василіу Євген Вікторович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, д.т.н., проф. кафедри КБ та ТЗІ

Підпис

«14» червня 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016337595

Дата перевірки:
09.06.2024 11:47:56 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2024 11:50:49 EEST

ID користувача:
100011688

Назва документа: 4РП-07_Павлюк

Кількість сторінок: 43 Кількість слів: 8600 Кількість символів: 57558 Розмір файлу: 2.56 MB ID файлу: 101613853

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

11.3%
Схожість

Найбільша схожість: 5.03% з Інтернет-джерелом (http://eir.zntu.edu.ua/bitstream/123456789/10309/1/BR_Schevtsov.pdf)

11.3% Джерела з Інтернету | 75 ..... Сторінка 45

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування | 10 сторінок

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНГО ДИПЛОМНОГО ПРОЕКТА
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Павлюк Олксій Павлович,
здобувач освіти гр. 4РП-07, та

Джабраілов Дмитро Володимирович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускного дипломного проекту фахового молодшого бакалавра на тему:

«Розробка шахової гри на програмному рушії Unity» (автор роботи – Павлюк О.П., керівник роботи – Джабраілов Д.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи, і даємо згоду на обробку персональних даних.

Виконавець _____ / Павлюк О.П./

Керівник _____ / Джабраілов Д.В./

«10» 06 2024 р.