Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Калінкіна Богдана Павловича*
(ПІБ)

академічної групи *122-20-3*
(шифр)

напряму підготовки *122 Комп'ютерні науки*
(код і назва напряму підготовки)

на тему: *Розробка комп'ютерної гри "Гомоку" на платформі Андроїд.*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Кабак Л.В*			
розділів:				
спеціальний	*доц. Кабак Л.В*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*доц. Каштан В.Ю.*			
Нормоконтролер	*доц. Гуліна І. Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-3* (група) *Калінкіна Богдана Павловича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка комп'ютерної гри "Гомоку" на платформі Андроїд*

затверджена наказом ректора НТУ «ДП» від *23 . 05 .2024 р.* № *470 -с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2024 р.***

Завдання видав _____ _____
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Калінкін Б.П.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 64 с., 15 рис., 0 табл., 3 дод., 17 джерел

Об'єкт розробки: комп'ютерна гра "Гомоку" для платформи Android.

Мета кваліфікаційної роботи: розробка та реалізація комп'ютерної гри "Гомоку" для платформи Android з метою надання користувачам можливості насолоджуватися грою на мобільних пристроях.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної області, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано платформу для розробки, виконано проектування і розробку програми, наведено опис алгоритму і структури функціонування підсистеми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної підсистеми, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Практичне значення роботи полягає у створенні мобільної гри "Гомоку", яка сприяє розвитку логічного та стратегічного мислення користувачів.

Актуальність мобільної гри "Гомоку" визначається великим попитом на подібні розробки, що дозволяють користувачам насолоджуватися грою на мобільних пристроях, забезпечуючи високий рівень залученості та інтерактивності.

Список ключових слів: ANDROID, ГРА, ҐОМОКУ, UNITY, C#, МІНІМАКС.

ABSTRACT

Explanatory note: 64 pages, 15 figures, 0 tables, 3 appendices, 17 sources

Object of development: a computer game "Gomoku" for the Android platform.

Purpose of the qualification work: development and implementation of the computer game "Gomoku" for the Android platform to provide users with the opportunity to enjoy the game on mobile devices.

The introduction examines the analysis and current state of the problem, specifies the purpose of the qualification work and its application field, provides justification for the relevance of the topic, and clarifies the task setting.

The first section analyzes the subject area, determines the relevance of the task and the purpose of the development, sets the task, and specifies the requirements for software implementation, technologies, and software tools.

The second section analyzes existing solutions, selects the platform for development, designs and develops the program, describes the algorithm and structure of subsystem functioning, defines input and output data, provides the characteristics of hardware parameters, and describes the application's invocation and loading, as well as its operation.

The economic section determines the labor intensity of the developed information subsystem, calculates the cost of work on the creation of the application, and estimates the time required for its development.

The practical significance of the work lies in the creation of the mobile game "Gomoku," which promotes the development of users' logical and strategic thinking.

The relevance of the mobile game "Gomoku" is determined by the high demand for such developments, allowing users to enjoy the game on mobile devices, providing a high level of engagement and interactivity.

Keywords: ANDROID, GAME, GOMOKU, UNITY, C#, MINIMAX.

ЗМІСТ

ВСТУП

В сучасному світі ігрова індустрія знаходиться на піку популярності, завойовуючи симпатії мільйонів гравців по всьому світу. Розробка комп'ютерних ігор стає не лише розважальною справою, але й важливим напрямом розвитку інформаційних технологій. Серед різноманіття ігор особливе місце займають логічні ігри, які сприяють розвитку мислення та стратегічного мислення гравців.

Однією з таких ігор є "Гомоку", яка має величезний потенціал для зацікавлення гравців будь-якого віку. Розробка комп'ютерної версії цієї гри на платформі Android стає актуальною завдяки швидкому розвитку мобільних технологій та зростанню популярності мобільних ігор серед користувачів.

Мета даної кваліфікаційної роботи полягає у розробці та реалізації комп'ютерної гри "Гомоку" для платформи Android з метою надати користувачам можливість насолоджуватися цією захоплюючою грою на своїх мобільних пристроях. Враховуючи популярність гри "Гомоку", а також зростання популярності мобільних платформ, розробка даної програми відповідає сучасним тенденціям розвитку ігрової індустрії та вимогам споживачів.

Актуальність теми обумовлена не лише загальним інтересом до комп'ютерних ігор, але й потребою у використанні сучасних технологій для розвитку ігрової індустрії. Розробка гри на платформі Android відкриває нові можливості для розваг та розвитку логічного мислення користувачів мобільних пристроїв.

Завданням є розробка програмного забезпечення для мобільних пристроїв на базі операційної системи Android, що відповідає вимогам сучасної ігрової індустрії та забезпечує якісне, захоплююче та доступне ігрове досвід користувачам.

Отже, розробка комп'ютерної гри "Гомоку" для платформи Android має значний практичний і науковий інтерес, що робить її актуальною для подальшого дослідження та розвитку в галузі геймдеву.

Беручи це все до уваги було сформовано тему кваліфікаційної роботи: «Розробка комп'ютерної гри "Гомоку" на платформі Андроїд

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості с предметної галузі

Гомоку, є однією з найстаріших та найпопулярніших логічних ігор. Вважається, що гра була створена в Китаї і їй не менше двох тисяч років. Близько VII століття вона була занесена до Японії, де взяла саме назву гри, яка походить від японського слова *ґомокунарабе*, що означає «п'ять штук в ряд»

Спочатку гра проходила на полі 19 × 19 без жодних обмежень, нині цей варіант гри відомий під назвою *Ґомоку без обмежень*. Пізніше додали правило заборони виграшу за допомогою «довгого ряду», і саме ця гра тепер має назву **Ґомоку** (рис.1.1).

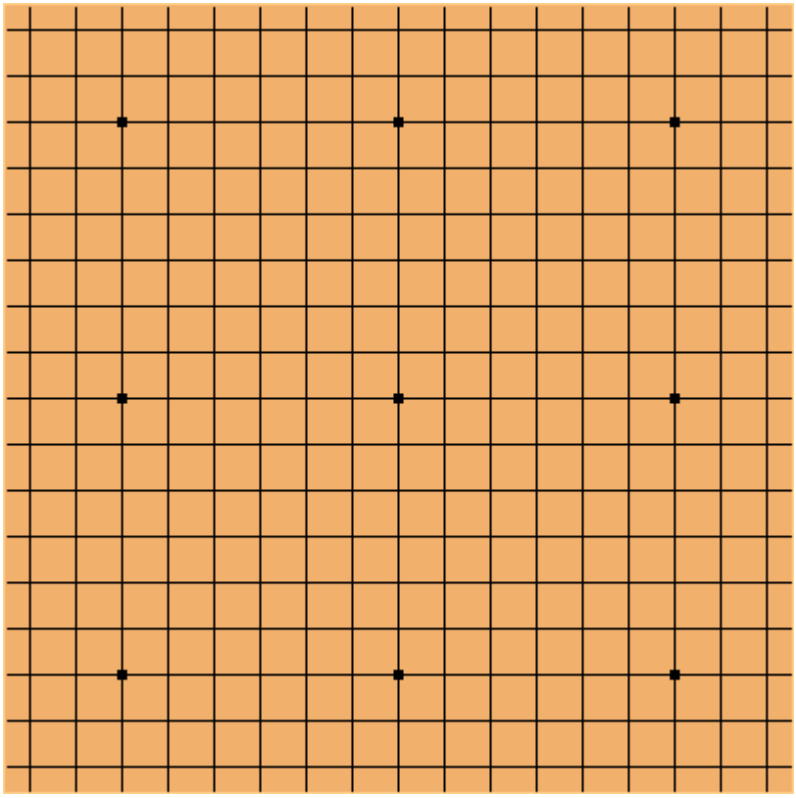

Рис 1.1. Класична дошка для гри в гомоку

Однак, в такому варіанті гравець, що робить свій хід першим (чорні камені), має беззаперечну перевагу, і, за оптимальної гри, завжди виграє, що нестрого довели деякі японські професіонали.

Пізніше, було обмежено розмір дошки до 15 х 15, що зараз є найбільш популярною варіації гри.

Однак, Розвиток гри *Гомоку* не стояв на місці і, згодом, отримала безліч варіацій, які варіюються за розміром дошки, правилами гри та стратегією. Однак, незалежно від варіації, ця гра завжди вимагає від гравців стратегічного мислення та аналізу.

Абстрактні стратегічні настільні ігри відіграють важливу роль як у суспільстві, так і в інформатиці. У них грають мільйони людей і вони використовуються в інформатиці з багатьох причин, включаючи освітні цілі та контрольні показники штучного інтелекту.

Протягом багатьох років програмісти застосовували штучний інтелект для гри в гомоку. У 1962 році Джозеф Вайзенбаум, учений, фахівець у галузі штучного інтелекту, опублікував статтю "Як зробити комп'ютер розумним", в якій описав стратегію для програми гомоку, яка може перемагати початківців. Проте ще не вирішені всі теоретичні значення позицій та відкриті правила, які використовуються професіоналами, тому розробка алгоритмів гомоку залишається викликом для вчених-програмістів навіть сьогодні.

Отже, ця гра знаходить широке застосування в галузі штучного інтелекту, ставлячи виклик алгоритмам пошуку та інтелектуальним стратегіям. Багато досліджень у цій галузі спрямовані на розвиток більш складних алгоритмів та нових стратегій гри для гравців.

Як ми вже зазначили, гра "Гомоку" відзначається своєю простотою правил, але водночас вимагає великого рівня стратегічного мислення та обчислювальної складності для створення ефективного штучного інтелекту.

У зв'язку з цим, вона стає ідеальним полем для випробування та вдосконалення різних алгоритмів штучного інтелекту. Основні алгоритми для розробки або тренування штучного інтелекту:

- Мінімаксний алгоритм (англ. *Minimax)* є одним з найбільш поширених підходів до розробки штучного інтелекту для гри "Гомоку". Він базується на ідеї максимізації виграшу для одного гравця та мінімізації виграшу для іншого. Алгоритм просувається рекурсивно через можливі комбінації ходів, оцінюючи потенційні наслідки кожного кроку для кожного гравця. (рис. 1.2.)

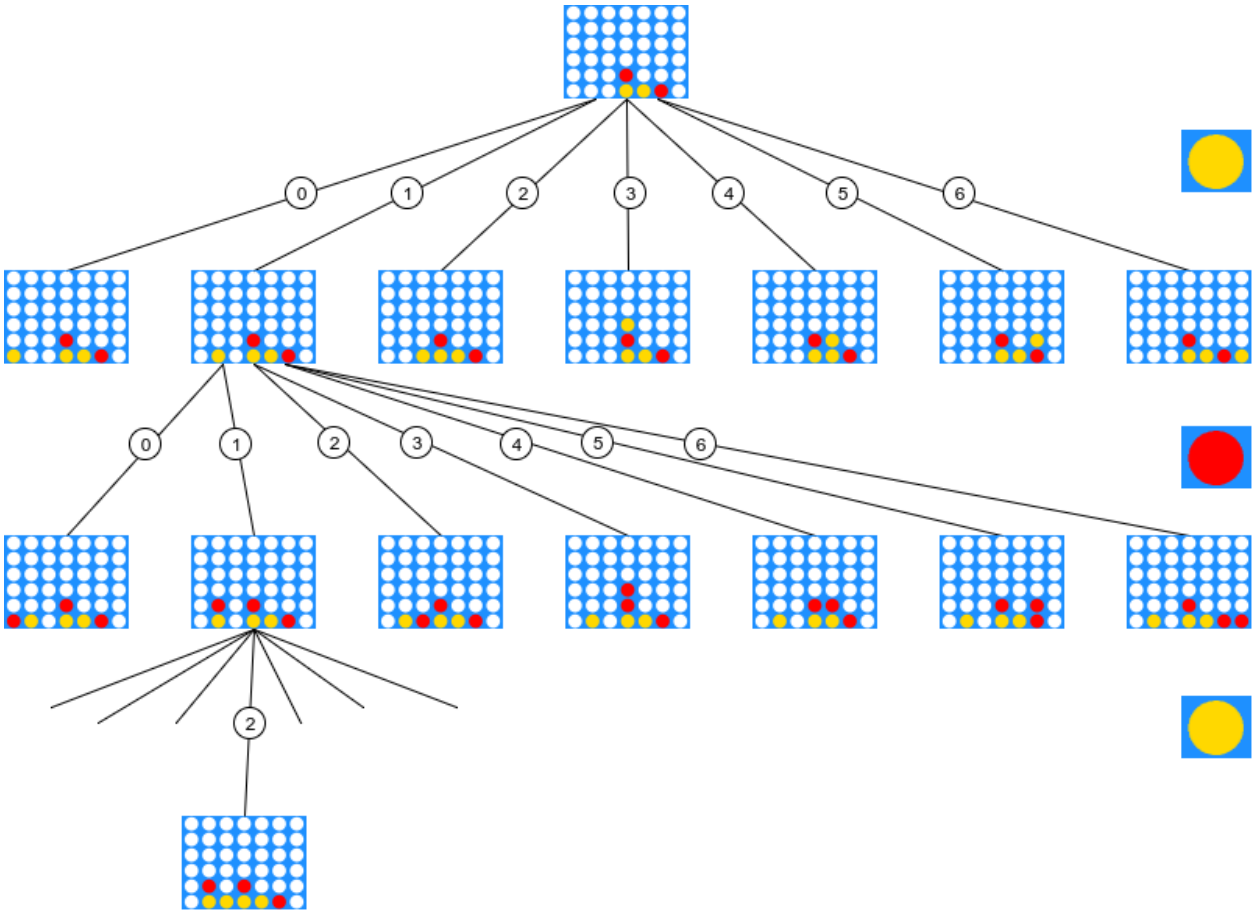

Рис 1.2. Логіка алгоритму мінімакс на прикладі Гомоку

У Мінімаксі фактично можна назвати двох гравців максимайзером і мінімайзером. Максимайзер завжди хоче отримати найвищий рахунок, тоді як мінімайзер намагається отримати найнижчий рахунок.Таким чином, у

кожному пошуку алгоритм відстоює сторону максимізації вигод поточного гравця. Кожний "найкращий хід", виведений з алгоритму, не є максимумом за поточної ситуації, але найкращим вибором після узгодження, враховуючи, що супротивник буде мінімізувати вигоди для поточного гравця.його формальне визначення – $v_i = max_{a_i} min_{a_{-i}} v_i(a_i, a_{-i})$, де i - це індекс поточного гравця, $-i$ позначає всіх інших гравців, крім гравця i, a_i - це дія, вчинена гравцем i, a_{-i} і v_i - це функція значення гравця i.

Характеристики прийняття рішень мінімаксу не є ймовірними, оскільки вихідні результати макро-походять для оцінки очікуваної вартості та корисності. Він не аналізує ймовірність кожного результату, а лише передбачає та обчислює кожен можливий сценарій. (рис. 1.3.)

```
1: function MINIMAX(position, depth, maximizingPlayer)
2:     if depth == 0 or game over in position then
3:         return static evaluation of position
4:     if maximizingPlayer then
5:         maxEval = −∞
6:         each child in position
7:         eval = Minimax(child, depth-1, false)
8:         maxEval = max(maxEval, eval)
9:         return maxEval
10:    else
11:        minEval = +∞
12:        each child in position
13:        eval = Minimax(child, depth-1, true)
14:        minEval = min(minEval, eval)
15:        return minEval
```

Рис 1.3. Псевдокод алгоритму мінімакс

Це рекурсивна функція, яка складається з трьох частин. Перша частина - це вихід знизу, умова, за якої досягнуто цільову глибину пошуку або гра закінчилася. Друга і третя частини вирішують максимальне значення поточного гравця і оптимальне рішення супротивника відповідно. У псевдокоді maximizingPlayer є булевою змінною. Коли поточна глибина є непарною (maximizingPlayer == true), що означає, що це хід першого гравця, вона буде оцінена на основі встановленого рівняння оцінки дошки. Навпаки, якщо це парна глибина (maximizingPlayer == false), тобто це хід другого

гравця, вона знайде вузол у поточній ситуації, який є найбільш невигідним для першого гравця. Фактично, легко помітити, що цей алгоритм все ще є вичерпним процесом розв'язання, оскільки, якщо алгоритм не робить жодних змін, він поверне остаточну відповідь після пошуку всіх можливостей.

- Алгоритм альфа-бета відсічення - це вдосконалений алгоритм, який може зменшити кількість вузлів у Мінімаксі. У алгоритмі Мінімакс ми говорили про те, як дерево гри шукає всі можливості для досягнення кінцевого результату, але в цьому процесі багато вузлів насправді не потрібно робити глибокий пошук. Відсічення альфа-бета припинить оцінювати хід, коли досягне вузла, який гірший, ніж раніше вивчений. Потім процес регулярно обчислює залишені вузли над вузлами, які наразі відкинуті. Коли ми застосовуємо стандартний алгоритм Мінімаксу, альфа-бета повертає той самий результат, що й Мінімакс, але відсічає деякі гілки, які не можуть вплинути на кінцеве рішення.(рис. 1.4.)

```
function MINIMAX(position, depth, alpha, deta, maximizingPlayer)
    if depth == 0 or game over in position then
        return static evaluation of position
    if maximizingPlayer then
        each child in position
        eval = Minimax(child, depth-1, false)
        maxEval = max(maxEval, eval)  alpha= max(alpha, eval)
        if beta ≤ alpha break
        return maxEval
    else
        each child in position
        eval = Minimax(child, depth-1, true)
        minEval = min(minEval, eval)  beta= max(beta, eval)
        if beta ≤ alpha break
        return minEval
```

Рис 1.4. Псевдокод алгоритму альфа-бета відсічення

У цьому алгоритмі вводяться дві значення, альфа і бета, які представляють максимальний рахунок поточного гравця, який потрібно перемістити, та мінімальний рахунок супротивника. Початкове значення для альфи - це від'ємна нескінченність, тоді як для бети - це позитивна нескінченність. Коли мінімальний рахунок супротивника менший за максимальний рахунок поточного гравця, тоді ми можемо зупинити процес, оскільки ці рахунки ніколи не з'являться у фактичній грі. У статті Nasa [1], де також було реалізовано Мінімакс та альфа-бета, і потім порівняно їх

експериментальні дані, було очевидно, що у їх результатах альфа-бета мав швидшу швидкість відповіді.(рис.1.5.)

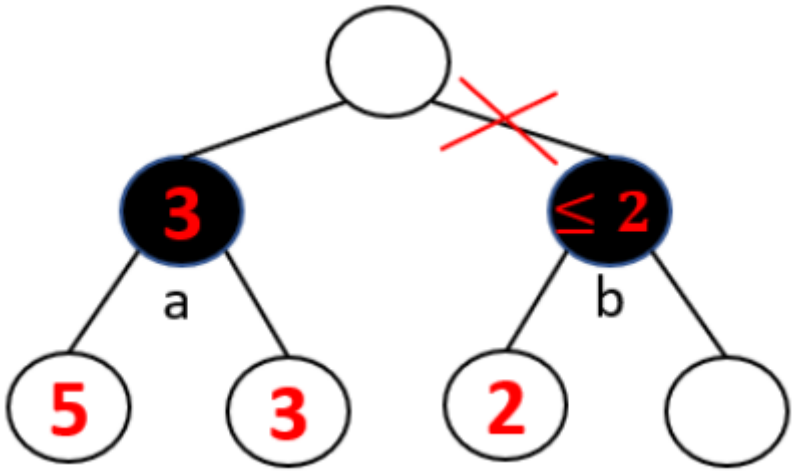

Рис 1.5. Відсічення альфа-бета

У рисунку1.5 ми наводимо простий приклад. За правилом пошуку першої глибини ми вже дослідили значення вузла a - 3, і значення вузла b менше або дорівнює 2, тоді це відповідає правилу, яке ми описали, що альфа, максимальне значення протилежного гравця, менше за значення бета. Тоді ми можемо повністю відкинути гілку вузла b.

Штучний інтелект - найгарячіша тема в галузі комп'ютерів, і людство зробило великі успіхи після 50 років досліджень і досліджень. У вирішенні настільної гри, в яку люди займаються тисячі років. У книзі, написаній Іаном Міллінгом [2], докладно описано та перераховано деякі ігри, які можна вирішити за допомогою простих алгоритмів. Штучний інтелект був активно розвивається і вивчається. Від "deep blue" в минулому столітті, який повністю перевершив людей у грі в шахи, до Alpha Go, розробленого компанією Google в 2016 році, який повністю перевершив рівень гри в Го людей, та зростання самокеруючих автомобілів, здається, що люди стають краще в штучному інтелекті. Проте дослідження базових алгоритмів ніколи не припинялося.

У початку минулого століття Варді [3] представив найпростіші алгоритми, такі як графік гри та Мінімакс, для вирішення деяких відносно простих настільних ігор, таких як найпростіші "хрестики-нулики" і шашки. Цей тип настільних ігор має очевидну характеристику, яка дозволяє використовувати швидкі обчислення комп'ютера, зберігання даних і обробку здатності для оцінки наступних ходів. Алгоритм альфа-бета може перелічити всі можливості та вибрати наступний крок через вбудований оцінювач шашок. На основі пропозиції про обрізання Дональда Е. Кнута [4] швидкість обчислення альфа-бета прискорюється, а непотрібна гілка пошуку спрощується, що значно прискорює обробку. На основі цього Мінімакс навіть має свої унікальні переваги і може бути ефективнішим, ніж альфа-бета, в тому сенсі, що Мінімакс ніколи не оцінює вузла, який альфа-бета може ігнорувати, Г.С. Стокман [5]. Варді [3] запропонував новий алгоритм Мінімакс. Різниця полягає у використанні нерівностей змінних для обробки умов обмежень та має стратегію області довіри, яка використовується зі структурою проблеми.

Для деяких настільних ігор, таких як шахи, алгоритм альфа-бета вже не є відповідним. Побудова функції оцінки на основі евристичних знань для неперервної позиції є складною та часоємкою задачею в таких проблемах. Дерево пошуку методом Монте-Карло *(анг. Monte Carlo Search Tree* - MCT) зробило можливим перевершення людьми комп'ютерами. Етаповий розвиток відбувся у 1997 році, коли штучний інтелект Deep Blue, розроблений компанією IBM, повністю переміг визнаних в світі відмінних шахістів. Алгоритм використовував MCT за роботою Філіппа Рольфшагена та Колтона [6]. Кулом [7] узагальнив три основні кроки в MCT: побудову ймовірності, реалізацію вибірки розподілу ймовірності та встановлення різноманітних оцінок. Гійом Шасло та Іштван Сзіта [8] запропонували використання каркасу MCT для запобігання експлуатації простору пошуку для передбачення найбільш обіцяних ігрових дій. Вони вважають, що MCT

може бути ефективно використаний в більшості настільних ігор та комп'ютерних іграх.

Проте для деяких складних настільних ігор навіть MCT не може бути симульований та оцінений. Тому Го вважався останньою бастионом людського інтелекту перед комп'ютерами. Однак поява нейронних мереж зламала тупік за роботою Купера та ін. [9]. Зокрема, з використанням високорівневих ігор, використовуються згорткові нейронні мережі та багатовимірні рекурентні мережі Ронгсіао [10]. Аналогічно до Гомоку, Чжентао Танг та ін. [11] поєднали дерево пошуку Монте-Карло та адаптивне динамічне програмування, яке може зміцнити навчання нейронної мережі. До фінального тесту Alpha Go, Сільвер Девід [12] опублікував офіційне повідомлення в Nature, оголошуючи, що Го буде вирішено новим способом, заявляючи, що мережі значень допоможуть оцінювати позицію на дошці, а мережі політики - вибирати ходи. Alpha Go перемогла Лі Седола через кілька місяців.

Хоча здається, що людство поступилося в настільних іграх. Проте дослідження алгоритмів ніколи не припинялося. Комбінуючи MCT і Deep Neural Network (DNN) навчання з підсиленням з використанням нейронних мереж, Сільвер Девід [12] запропонував систему, яка навчає машину без попередніх навичок людини. Це було досягнуто в Alpha Zero у 2017 році, розробленому Антоном [13]. Здається, що нейронна мережа з MCT змогла вирішити більшість настільних ігор, і ми сподіваємося отримати іншу перспективу для досягнення Гомоку. Основна відмінність між Гомоку та Го полягає в тому, що зміни в Го є більш складними та ненадійними, але Гомоку все ще є грою, яка потребує точного обчислення, і вибір кожного ходу повинен бути необхідним і потужним. Зрештою, вчені Гійом Шасло та Іштван Сзіта [8] використовували нейронні мережі для вирішення Гомоку.

Результати досліджень з використання різних алгоритмів для гри "Гомоку" можуть допомогти в розумінні їхньої ефективності та придатності

для цієї конкретної гри. Аналіз та порівняння різних підходів може виявити переваги та недоліки кожного з них, сприяючи подальшому розвитку штучного інтелекту в грі "Гомоку".

У моїй розробленій версії гри «Гомоку» був використаний мінімаксний алгоритм. Цей алгоритм базується на принципі максимізації виграшу для одного гравця та мінімізації для іншого, та він просувається рекурсивно через можливі комбінації ходів, оцінюючи потенційні наслідки кожного кроку для кожного гравця. Використання мінімаксного алгоритму дозволило досягти ефективності та стратегічності штучного інтелекту, що дозволяє йому приймати оптимальні рішення під час гри.

1.2 Призначення розробки та область застосування.

Цей проект орієнтований на широке коло користувачів, які мають інтерес до покращення своїх когнітивних здібностей, розваг та вивчення алгоритмів штучного інтелекту. Додаток не має обмежень за спеціалізацією або віковими категоріями і призначений для всіх, хто прагне розвивати свій розум та навички.

Цілі розробленого додатку включають:

1. Розваги: Гра Гомоку створює можливість для користувачів розслабитися, відпочити та насолодитися часом, проведеним за грою.

2. Розвиток когнітивних здібностей: Гра сприяє покращенню логічного мислення, аналітичних здібностей та стратегічного планування, що може бути корисним у різних аспектах життя, від розв'язання проблем до прийняття рішень.

3. Навчання: Гра Гомоку може бути використана як інструмент для навчання дітей основам логічного мислення та стратегічного мислення.

4. Розвиток моторики рук: Участь у грі сприяє покращенню моторики рук та координації рухів, що особливо корисно для молодших користувачів.

5. Змагання: Гра Гомоку може слугувати основою для змагань з друзями та членами родини, стимулюючи здоровий конкурентний дух та сприяючи взаємній взаємодії.

Цей проект спрямований як на створення інтерактивного середовища, що сприяє розвитку розумових та фізичних навичок користувачів, так і на вдосконалення мінімаксного алгоритму штучного інтелекту для підвищення рівня гри в Гомоку.

1.3 Підстава для розробки

Підставою для розробки кваліфікаційної роботи на тему “Розробка веб-додатку для покращення пам’яті, реакції та моторики рук” є наказ по Національному технічному університету “Дніпровська політехніка” від 23 . 05 .2024 р . № 470 -с

1.4 Постановка завдання

Метою даного проекту є розробка мобільного додатка на базі Android для гри в Гомоку з інтегрованим мінімаксним алгоритмом. Головною метою є створення цікавої та захоплюючої гри, яка сприятиме розвитку розумових навичок у користувачів та надасть можливість грати з друзями або з комп'ютером.

Для досягнення поставленої задачі необхідно вирішити основні завдання:

-Вивчення предметної області: Ознайомлення з правилами гри в Гомоку, аналіз існуючих додатків для гри в Гомоку та вивчення технологій, які будуть використані для розробки додатка.

-Проектування інтерфейсу: Розробка інтерактивного інтерфейсу для користувача, який буде зручним і привабливим.

- Реалізація гри: Створення логіки гри, включаючи можливість грати з іншим користувачем або з комп'ютером.

- Імплементація алгоритму для гри проти комп’ютера: Розробка алгоритму штучного інтелекту для автоматизованої гри проти комп'ютера, який забезпечить унікальний ігровий опит для гравця.

-Тестування та відладка: Проведення тестів для перевірки функціональності, ідентифікації та виправлення помилок.

В цілому, цей проект спрямований на створення захоплюючої гри, яка не лише забезпечить розвагу, а й сприятиме розвитку розумових навичок та стратегічного мислення у користувачів.

1.5 Вимоги до програми або програмного виробу.

1.5.1. Вимоги до функціональних характеристик.

-Можливість грати з іншим користувачем:

Додаток повинен забезпечувати можливість гри в Гомоку між двома користувачами на одному пристрої.

-Гра з комп'ютером:

Наявність режиму гри проти комп'ютера.

-Інтерактивний інтерфейс:

Інтуїтивний та зручний інтерфейс для користувача, який дозволить легко взаємодіяти з додатком та здійснювати потрібні дії.

-Сумісність з різними пристроями

Додаток повинен бути сумісний з різними пристроями на базі Android і забезпечувати оптимальну роботу на різних розширеннях екрану.

Ці вимоги забезпечать функціональність та зручність використання додатку для користувачів та гарантують його ефективну роботу на різних пристроях.

1.5.2 Вимоги до інформаційної безпеки

Оскільки додаток – є грою й виконується тільки на клієнтській стороні й не працює з конфіденційними даними, не підключений до інтернету, вимог до інформаційної безпеки немає.

1.5.3 Вимоги до складу та параметрів технічних засобів

-Операційна система: Android версії 10 та вище.

-Оперативна пам'ять: Мінімум 500 МБ оперативної пам'яті для оптимальної продуктивності.

-Вільне місце на диску: Рекомендується мати не менше 60 МБ вільного місця для інсталяції та збереження додатку та додаткових даних.

Ці вимоги забезпечать ефективну роботу програмного виробу на пристроях з Android та забезпечать зручне користування грою без зайвих перешкод або обмежень.

1.5.4 Вимоги до інформаційної та програмної сумісності

Для успішної розробки та функціонування створеного ігрового додатка було використано наступні технології та платформи:

Unity. Потужний двигун для розробки ігор, що забезпечує створення графіки, анімації та інтерактивності. Для відтворення графічних об'єктів, реалізації ігрової логіки та взаємодії був використаний Unity.

Мова програмування C# була використана для розробки функціональності гри, включаючи логіку гри, обробку подій та взаємодію об'єктів.

Вимоги до операційної системи та середовища розробки:

Операційна система: Для розробки програми необхідна операційна система, яка підтримує Unity та C#. Зазвичай Unity підтримує більшість популярних операційних систем, таких як Windows, macOS та Linux.

Unity Hub і Unity Editor: Для розробки було використано Unity Hub для управління версіями Unity та Unity Editor для створення та налаштування проекту.

Бібліотеки та фреймворки:

Для цього проекту не було використано сторонніх бібліотек або фреймворків.

Додаткові ресурси:

Для візуальних елементів гри були використані графічні ресурси, такі як спрайти символів та ігрових об'єктів.

Зазначений набір програмного забезпечення є достатнім для розробки та функціонування гри "Гомоку" в середовищі Unity, але хочу зазначити, що

для запуску гри "Гомоку" необхідний лише мобільний пристрій або симулятор з операційною системою Android. Гра була розроблена спеціально для платформи Android, і тому операційна система Android є обов'язковою вимогою.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Мета даної кваліфікаційної роботи полягає у розробці та реалізації комп'ютерної гри "Гомоку" для платформи Android з метою надати користувачам можливість насолоджуватися цією захоплюючою грою на своїх мобільних пристроях. Враховуючи популярність гри "Гомоку", а також зростання популярності мобільних платформ, розробка даної програми відповідає сучасним тенденціям розвитку ігрової індустрії та вимогам споживачів.

Гра "Гомоку" - це комбінаторна логічна гра, в якій гравці розміщують свої шашки на ігровому полі з метою створити безперервний ряд з п'яти або більше шашок свого кольору по горизонталі, вертикалі або діагоналі. Гравець, який перший досягне цієї мети, виграє гру.

2.2. Опис застосованих математичних методів

У моїй розробленій версії гри «Гомоку» був використаний мінімаксний алгоритм. Цей алгоритм базується на принципі максимізації виграшу для одного гравця та мінімізації для іншого, та він просувається рекурсивно через можливі комбінації ходів, оцінюючи потенційні наслідки кожного кроку для кожного гравця. Використання мінімаксного алгоритму дозволило досягти ефективності та стратегічності штучного інтелекту, що дозволяє йому приймати оптимальні рішення під час гри.

Часова складність мінімакса дорівнює O(b^m), а просторова складність – O(bm), де b – кількість допустимих ходів у кожній точці, а m – максимальна глибина дерева.

2.3. Опис використаних технологій та мов програмування

Під час розробки гри "Гомоку" було ретельно обрано програмні засоби та технології, що найкращим чином задовольняють вимоги завдання.

Unity було обрано як основне середовище розробки завдяки його потужним можливостям для створення ігор та додатків. Unity надає засоби для розробки графічного інтерфейсу, анімацій, програмування логіки гри, що робить його ідеальним вибором для цього завдання.

Unity - багатоплатформовий інструмент для розроблення відеоігор і застосунків, і рушій, на якому вони працюють. Створені за допомогою Unity програми працюють на настільних комп'ютерних системах, мобільних пристроях та гральних консолях у дво- та тривимірній графіці, та на пристроях віртуальної чи доповненої реальності. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL.

Робота з ресурсами

Редактор Unity має інтерфейс, що складається з різних вікон, які можна розташувати на свій розсуд. Завдяки цьому можна проводити налагодження гри чи застосунка прямо в редакторі. Головні вікна — це оглядач ресурсів проєкту, інспектор поточного об'єкта, вікно попереднього перегляду, оглядач сцени та оглядач ієрархії ресурсів.

Проєкт в Unity поділяється на сцени (рівні) — окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як об'єкти-моделі (ландшафт, персонажі,

предмети довкілля тощо), так і порожні ігрові об'єкти — ті, що не мають моделі, проте задають поведінку інших об'єктів (тригери подій, точки збереження прогресу тощо). Їх дозволяється розташовувати, обертати, масштабувати, застосовувати до них скрипти. В них є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов'язково наявний компонент Transform — він зберігає в собі координати місця розташування, повороту і розмірів по всіх трьох осях. У об'єктів з видимою геометрією також за умовчанням присутній компонент Mesh Renderer, що робить модель видимою. Різні моделі можуть об'єднуватися в набори (ассети) для швидкого доступу до них. Наприклад, моделі споруд на спільну тему.

Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

У 2D іграх Unity переважно використовує спрайти. В 3D іграх Unity здебільшого використовує тривимірні моделі (меші), на які накладаються текстури (зумовлюють вигляд поверхні об'єктів), матеріали (зумовлюють як поверхня реагуватиме на різні фактори) та шейдери (невеликі скрипти, за яким вираховується зміна кольору кожного пікселя згідно заданих параметрів, як-от розсіяння відбитого світла). В обох видах застосовуються системи часток для відображення субстанцій, таких як рідини чи дим.

Unity підтримує стиснення текстур, міпмапінг і різні налаштування роздільності екрана для кожної платформи; забезпечує бамп-мапінг, мапінг відображень, паралакс-мапінг, затінення навколишнього світла у екранному просторі, динамічні тіні за картами тіней, рендер у текстуру та повноекранні

ефекти обробки зображення, такі як зернистість, глибина чіткості, розмиття в русі, відблиски віртуальних лінз або ореол навколо джерел світла.

Рендеринг

Рендеринг зображення відбувається через віртуальну камеру огляду. В робочій області редактора ігрова сцена може розміщуватися як завгодно, а при рендерингу — так, як її видно з камери. В сцені може бути декілька камер, які рухаються за персонажем чи за вказаною траєкторією. Вигляд з камери подається в двовимірно чи тривимірно (в перспективі або ортографічно). Фон сцени, видимий через камеру, типово зображає небо, утворене скайбоксом, але може презентувати й інше довкілля.

Графічний рушій використовує DirectX (Windows), OpenGL (Mac, Windows, Linux), OpenGL ES (Android, iOS), та спеціальне власне API для Wii. Також підтримуються bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), динамічні тіні з використанням shadow maps, render-to-texture та повноекранні ефекти post-processing.

Unity підтримує файли 3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks та Allegorithmic Substance. В ігровий проєкт Unity можна імпортувати об'єкти цих програм та виконувати налаштовування за допомогою графічного інтерфейсу.

Для написання шейдерів використовується ShaderLab, що підтримує шейдерні програми написані на GLSL або Cg. Шейдер може включати декілька варіантів реалізації, що дозволяє Unity визначати найкращий варіант для конкретної відеокарти. Unity також має вбудовану підтримку фізичного рушія Nvidia PhysX (колишнього Ageia), підтримку симуляції одягу в системі реального часу на довільній та прив'язаній полігональній сітці (починаючи з Unity 3.0), підтримку системи ray casts та шарів зіткнення.

Скрипти

Скриптова система ігрового рушія зроблена на Mono — вільному відкритому проєкті з реалізації .NET Framework. Програмісти можуть використовувати UnityScript (власна скриптова мова, подібна до JavaScript та ECMAScript), C# або Boo (мова програмування, подібна до Python). Починаючи з версії 3.0, до Unity входить перероблена версія MonoDevelop для зневадження скриптів.

З виходом версії 5.2 у 2015 році передбачена вбудована можливість редагувати скрипти у середовищі Visual Studio.

Мову програмування C# використано для реалізації функціональності гри. Об'єктно-орієнтований підхід мови дозволяє організовувати код ефективно та структуровано.

Опис C#:

C# — об'єктно-орієнтована, орієнтована на компоненти мова програмування. C# надає мовні конструкції безпосередньої підтримки такої концепції роботи. Завдяки цьому C# підходить для створення та застосування програмних компонентів. З моменту створення мова C# збагатилася функціями для підтримки нових робочих навантажень та сучасними рекомендаціями щодо розробки ПЗ. В основному C# - об'єктно-орієнтована мова.

Ось лише кілька функцій мови C#, які дозволяють створювати надійні та стійкі програми:

-Складання сміття автоматично звільняє пам'ять, зайняту недосяжними об'єктами, що не використовуються.

- Типи, що допускають значення null, забезпечують захист від змінних, які посилаються на виділені об'єкти.

- Обробка винятків надає структурований та розширюваний підхід до виявлення помилок та відновлення після них.

-Лямбда-вираження підтримують прийоми функціонального програмування.

- Синтаксис LINQ створює загальний шаблон для роботи з даними будь-якого джерела.

- Підтримка мов для асинхронних операцій забезпечує синтаксис для створення розподілених систем.

- В C# є Єдина система типів.

Всі типи C#, включаючи типи-примітиви, такі як int та double, успадковують від одного кореневого типу об'єкта. Усі типи використовують загальний набір операцій, а значення будь-якого типу можна зберігати, передавати і обробляти так. Більше того, C# підтримує як визначені користувачами типи посилань, так і типи значень. C# дозволяє динамічно виділяти об'єкти та зберігати спрощені структури у стеку. C# підтримує універсальні методи та типи, що забезпечують підвищену безпеку типів та продуктивність.

C# надає ітератори, які дозволяють розробникам класів колекцій визначати варіанти поведінки для клієнтського коду.

C# підкреслює Керування версіями для забезпечення сумісності програм і бібліотек з часом. Питання управління версіями суттєво вплинули на такі аспекти розробки C#, як роздільні модифікатори virtual та override, правила дозволу навантаження методів та підтримка явного оголошення членів інтерфейсу.

Мова C# використовується для:

- Розробки десктопних програм (Windows Forms, WPF).

- Створення веб-застосунків (ASP.NET, Blazor).

- Написання мобільних додатків (Xamarin).

- Розробки ігор (Unity, MonoGame).

- Створення служб (Windows services).

- Розробки серверних додатків (ASP.NET Core).

- Реалізації різноманітних мікросервісів та API (ASP.NET Core Web API).

- Розробки кросплатформових додатків (Uno Platform).

- Використання у наукових дослідженнях, наприклад, у даних аналізі та обробці.

- Написання скриптів для автоматизації завдань на Windows (за допомогою .NET Framework або .NET Core).

Архітектура .NET

Програми C# виконуються в .NET, віртуальній системі виконання, що викликає загальномовне середовище виконання (CLR) та набір бібліотек класів. Середовище CLR – це реалізація загальномовної інфраструктури мови (CLI), що є міжнародним стандартом від корпорації Майкрософт. CLI є основою для створення середовищ виконання та розробки, у яких мови та бібліотеки прозоро працюють одна з одною.

Вихідний код, написаний мовою C# компілюється в проміжну мову (IL), що відповідає специфікаціям CLI. Код на мові IL та ресурси, у тому числі растрові зображення та рядки, зберігаються у збірці, зазвичай з розширенням .dll. Складання містить маніфест з інформацією про типи, версії, мові та регіональні параметри для цієї збірки.

Під час виконання програми C# збірка завантажується у середу CLR. Середовище CLR виконує JIT-компіляцію з коду мовою IL в інструкції машинної мови. Середовище CLR також виконує інші операції, наприклад,

автоматичне складання сміття, обробку винятків та управління ресурсами. Код, виконуваний середовищем CLR, іноді називають "керованим кодом". "Некерований код" компілюється на машинну мову, призначену для конкретної платформи. Забезпечення взаємодії між мовами є ключовою особливістю .NET. Код IL, створений компілятором C# відповідає специфікації загальних типів (CTS). Код IL, створений з коду C#, може взаємодіяти з кодом, створеним з .NET версій для мов F#, Visual Basic, C++. Існує більше 20 інших мов, сумісних із CTS. Одна збірка може містити кілька модулів, написаних різними мовами .NET, і всі типи можуть посилатися один на одного, якби вони були написані однією мовою.

Крім служб часу виконання .NET також включає розширені бібліотеки. Ці бібліотеки підтримують безліч різних робочих навантажень. Вони впорядковані за просторами імен, які надають різні корисні можливості: від операцій файлового введення та виведення до керування рядками та синтаксичного аналізу XML, від платформ веб-додатків до елементів керування Windows Forms. Зазвичай програма C# активно використовують бібліотеку класів .NET для вирішення типових завдань.

2.4. Опис структури системи та алгоритмів її функціонування

Графічні ігри, які не вимагають обробки особистих даних або системи нагород, можуть бути розроблені виключно з використанням клієнтських мов програмування.

Архітектура додатку базується на MVVM (Model-View-ViewModel), який є шаблоном проєктування для створення архітектури програмних додатків.

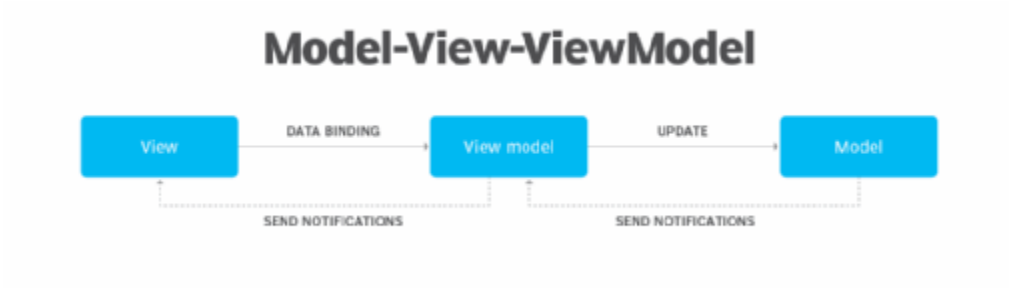

Рис. 2.1. Патерн MVVM

Логіка гри має структуру, зображену на UML діаграмі:

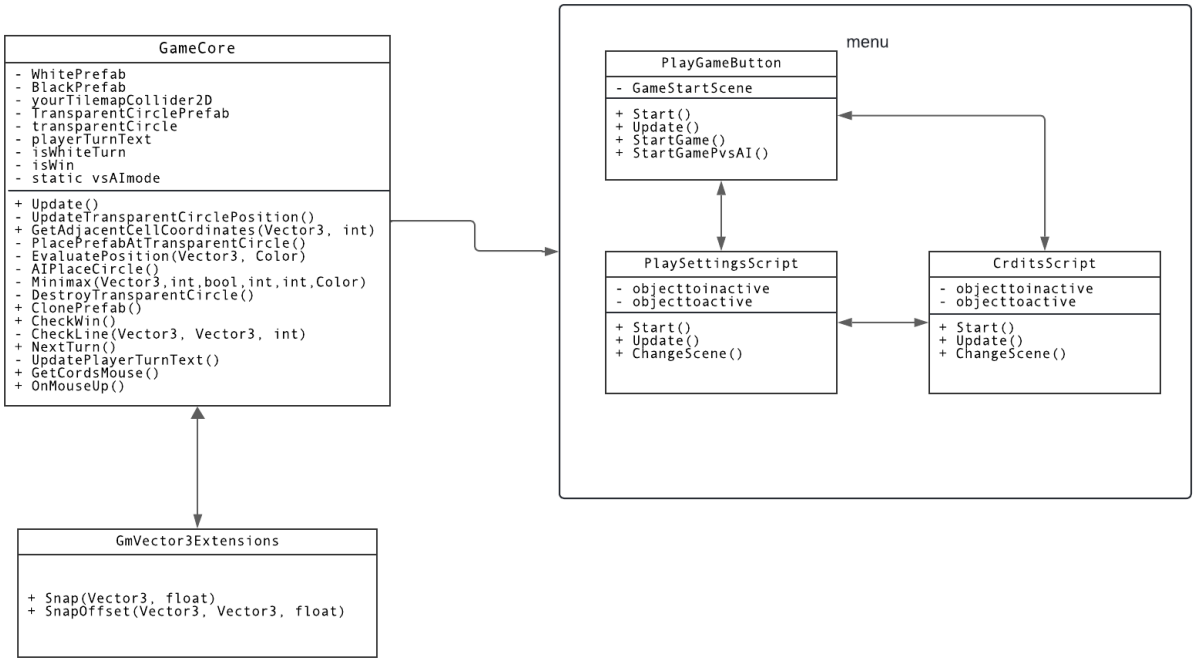

Рис. 2.2. UML діаграма логіки гри

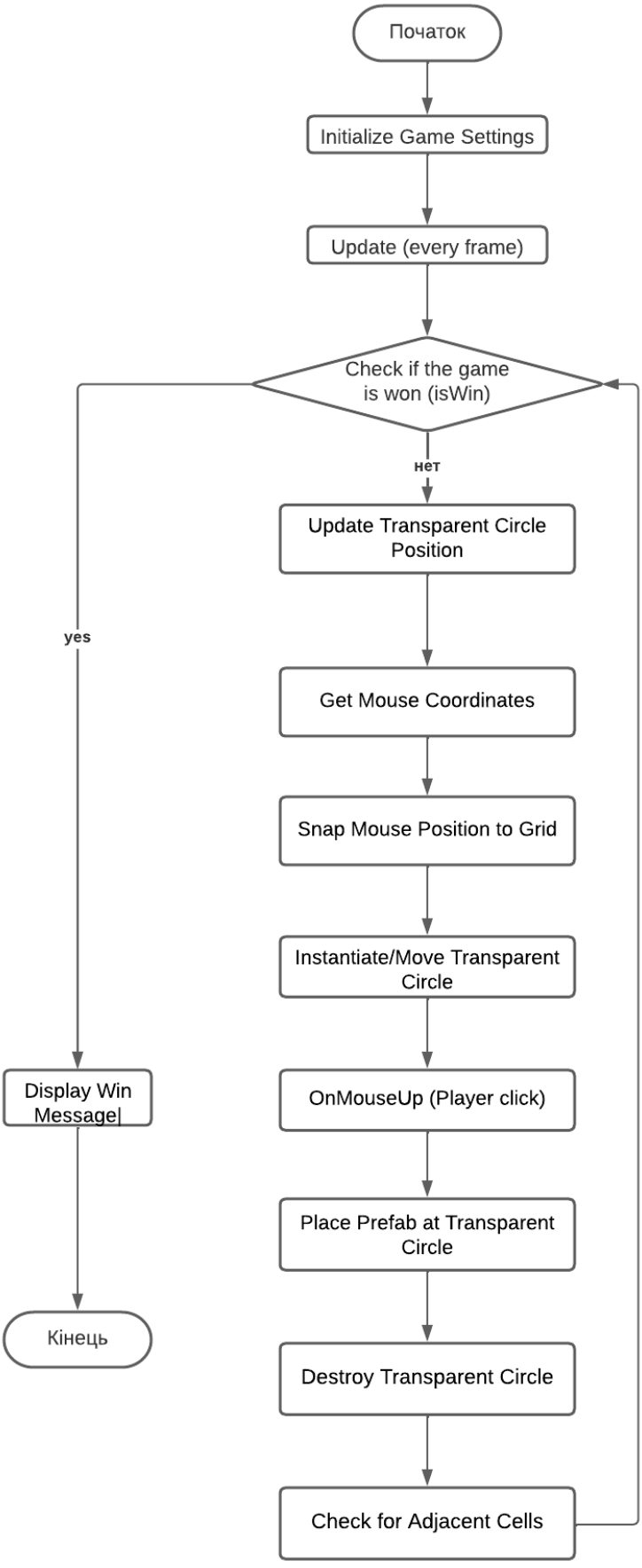

Рис 2.3. Flow chart ігрового циклу

Логічна схема додатку базується на взаємодії різних компонентів, що забезпечують виконання правил гри та управління. Основні компоненти включають графічні об'єкти та логіка гри.

Для представлення ігрового поля та шашок використовуються дошка, що являє собою tilemap. Кожна клітинка поля може бути зайнята шашкою гравця або залишитися порожньою.

Логіка гри реалізована в класі GameCore. Вона включає в себе розміщення шашок на полі, перевірку на перемогу, а також алгоритми для режиму "гравець проти комп'ютера".

Для визначення найкращого ходу комп'ютера було вибрано алгоритм Мінімакс з альфа-бета відсіканням. Цей алгоритм дозволяє оцінити різні можливі варіанти ходів на певну глибину та вибрати найкращий хід для комп'ютера, максимізуючи його шанси на перемогу.

Логічна схема додатку базувалася на послідовному виконанні дій: від вибору режиму гри до завершення гри з перемогою одного з гравців або заповнення всіх полів. Головний клас GameCore керував усіма аспектами гри, включаючи взаємодію з об'єктами, логіку ходів та перевірку на перемогу.

Ця логічна схема забезпечувала функціональність та ігровий процес, що відповідали правилам гри "Гомоку".

2.5. Обгрунтування та організація вхідних та вихідних даних програми

Програмне забезпечення отримує вхідні дані шляхом зчитування інформації з UI – елементу:

Вхідні дані:

- Вибір гравця:

Гравець обирає клітинку для розміщення свого символу (біла або чорна кулька) на ігровому полі.

Вихідні дані:

-Стан ігрового поля:

Поточний стан ігрового поля після ходу гравця та комп'ютера.

-Результат гри:

Визначення переможця або нічиєї, якщо гра закінчилася.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Рекомендовані вимоги для клієнтських технічних засобів:

-Операційна система: Android версії 10 та вище.

-Оперативна пам'ять: Мінімум 500 МБ оперативної пам'яті для оптимальної продуктивності.

-Вільне місце на диску: Рекомендується мати не менше 60 МБ вільного місця для інсталяції та збереження додатку та додаткових даних.

Ці вимоги забезпечать ефективну роботу програмного виробу на пристроях з Android та забезпечать зручне користування грою без зайвих перешкод або обмежень.

2.6.2. Використані програмні засоби

Використані програмні засоби та технології:

Unity: Середовище розробки ігор, яке забезпечує інтеграцію графічних компонентів, обробку взаємодії, анімацію та створення ігрового інтерфейсу.

C#: Мова програмування, використовувана для розробки функціональності гри та логіки взаємодії об'єктів.

Tilemap: Використовується для створення грального дошки та поля гри.

Text: Використовується для відображення повідомлень гравцям.

SpriteRenderer: Використовується для відображення символів гравців на полі гри.

2.6.3. Виклик та завантаження програми

На даний момент, завантаження гри можливе лише шляхом завантаження .apk файлу на пристрій, на якому потім буде встановлено програму.

Гра буде автоматично додана до головного меню після встановлення, дозволяючи користувачам легко запускати її одним натисканням.

2.6.4. Опис інтерфейсу користувача

Інтерфейс взаємодії з користувачем реалізований з використанням компонентів Unity UI. Текстове поле відображає поточного гравця та повідомлення про перемогу.

При розробці інтерфейсу гри "Гомоку" було приділено особливу увагу зручності та інтуїтивності користування. Інтерфейс повинен бути "візитною карткою" гри та надавати користувачу можливість легко засвоїти правила гри та взаємодіяти з додатком.

2.6.4.1 Головне меню

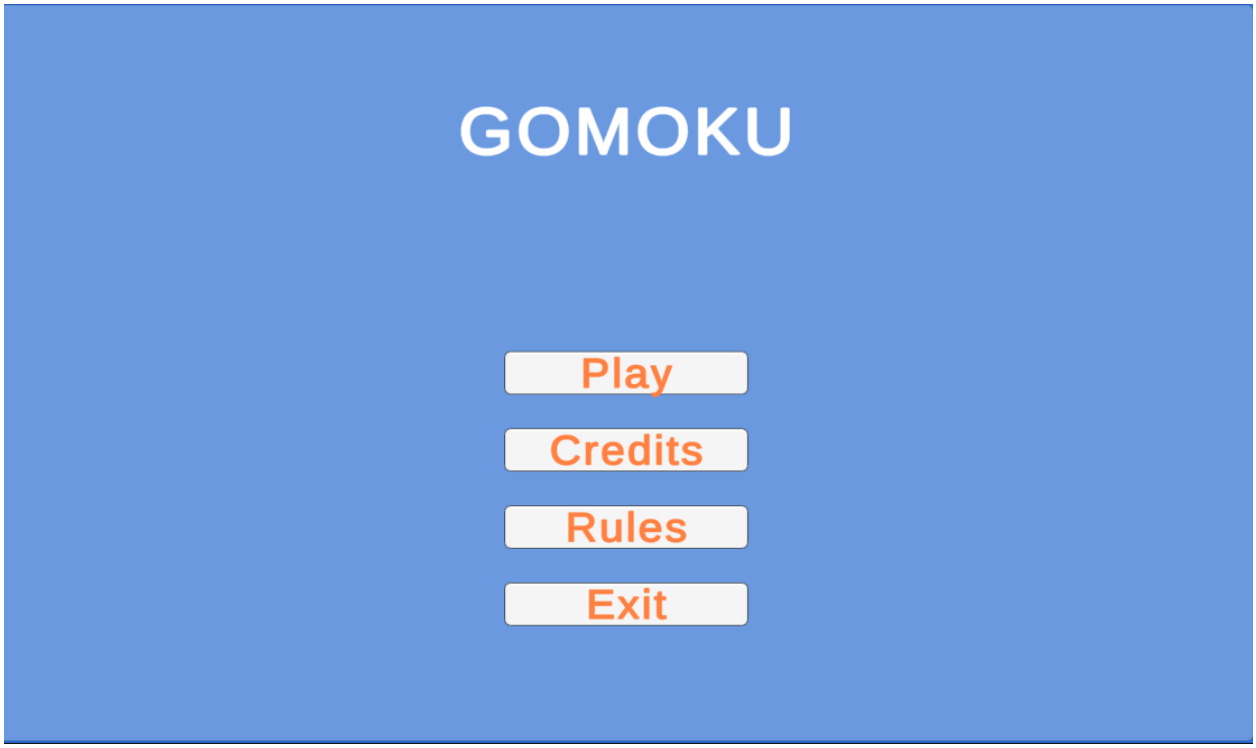

Рис. 2.2 – Головне меню

При запуску додатку відображається головне меню, де користувач може перейти далі (рис. 2.3), дізнатися більше про авторів грі (рис. 2.4), почитати правила гри (рис. 2.5), та вийти з додатку (кнопка “Exit”).

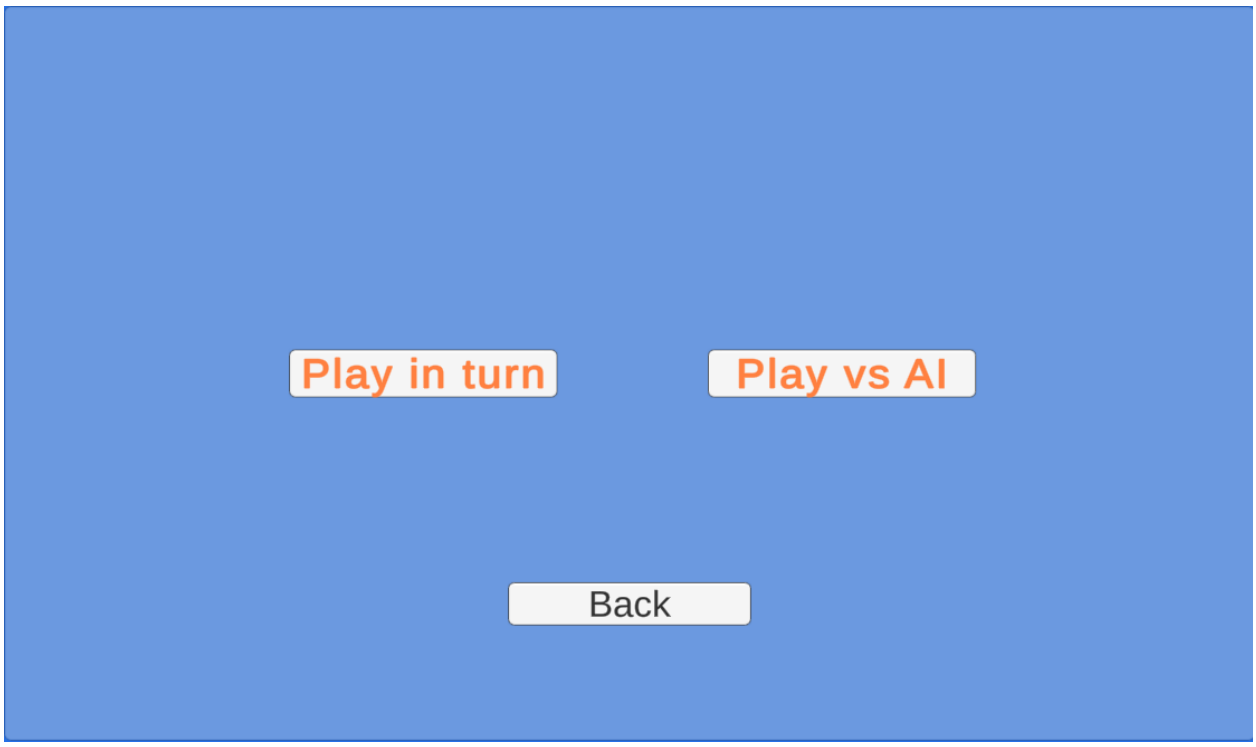

Рис. 2.3 – Меню вибора режима гри (кнопка play)

Рис. 2.4 – Автор гри (кнопка credits)

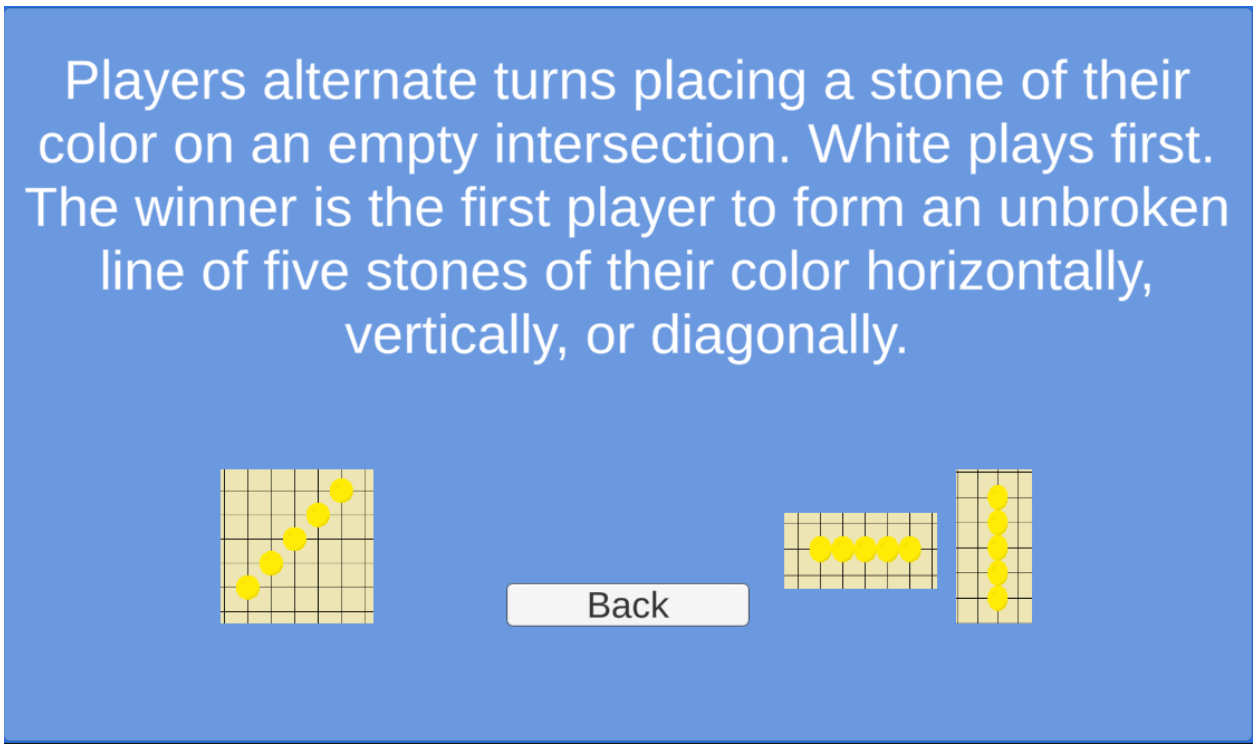

Рис. 2.5 – Правила гри «Гомоку»(кнопка rules)

2.6.4.2 Ігрове поле

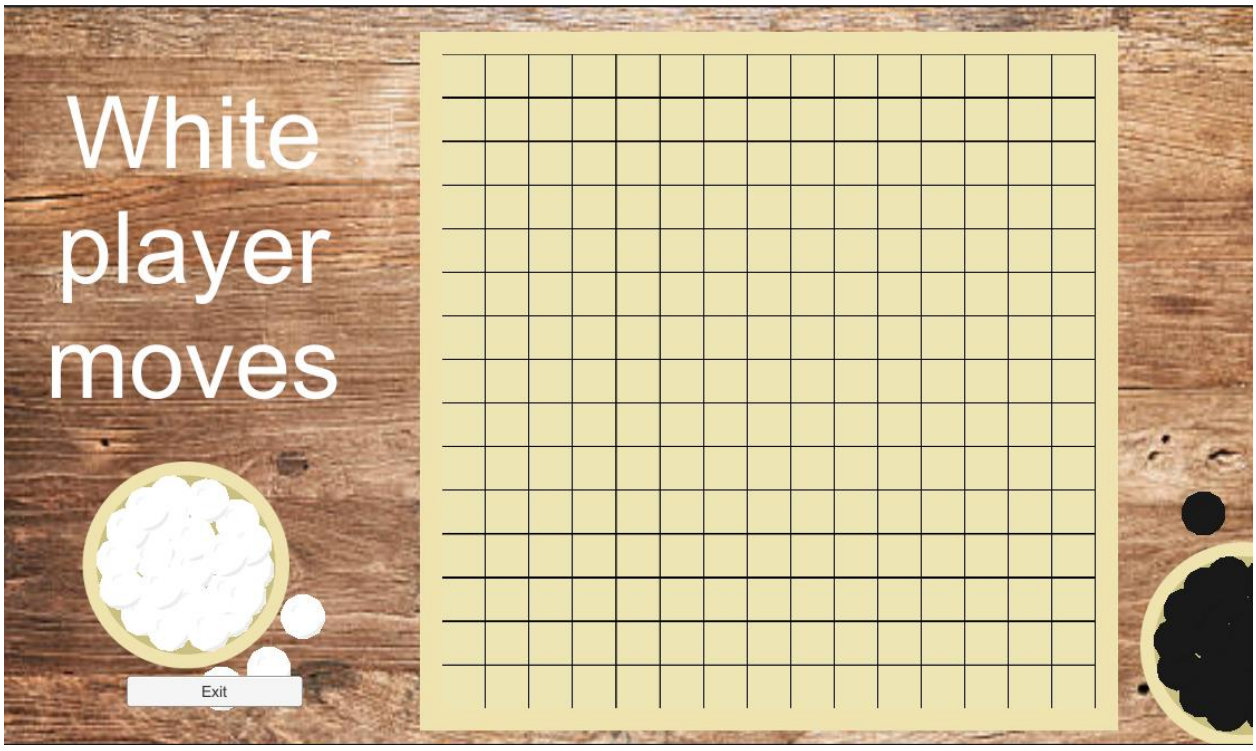

Рис. 2.6 – Ігрове поле

Ігрове поле представлено у вигляді квадратної матриці, розміром 15x15 клітинок. Кожна клітинка може містити шашку гравця або бути порожньою. Гравець може робити ходи, тапаючи на порожні клітинки і розміщуючи на них свої шашки. Зліва знизу є кнопка “Exit”, яка, дозволяє вийти назад у головне меню.

2.6.4.3 Панель стану

У верхній лівій частині екрану ігрового поля розташована панель стану, де відображається інформація про поточного гравця. Також на цій панелі може бути показано повідомлення про перемогу одного з гравців або про нічию.

2.6.4.4 Взаємодія з ігровим полем

Користувач може взаємодіяти з ігровим полем, розміщуючи свої шашки на порожніх клітинках. Це досягається шляхом тапання на бажану клітинку. Після вибору клітинки може здійснюватися перевірка на правильність ходу та здійснення самого ходу.

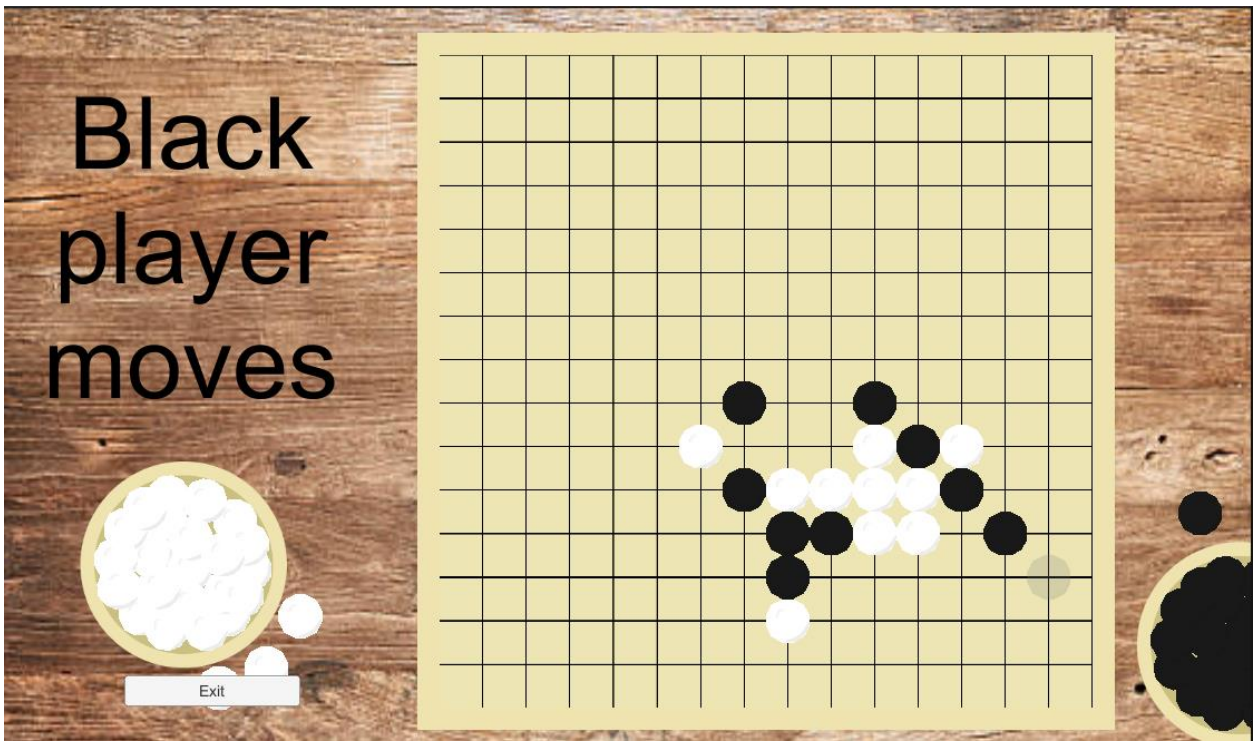

Рис. 2.7 Ігровий процес

2.6.4.5 Графічний дизайн і анімації

При розробці графічного інтерфейсу були використані яскраві та зрозумілі кольори, що допомагають користувачу зосередитися на грі та полегшують розрізнення шашок різних гравців.

Розробка графічного інтерфейсу для гри "Гомоку" виявилася важливим етапом проекту. Створений інтерфейс є зручним та інтуїтивно зрозумілим, забезпечуючи гравцю зручну можливість взаємодії з грою. Яскравий дизайн та анімації додають грі привабливості та динамічності, покращуючи загальний користувацький досвід.

ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

– передбачуване число операторів - 600;

– коефіцієнт складності програми - 1,25;

– коефіцієнт корекції програми в ході розробки - 0,05;

– годинна заробітна плата розробника – 240 грн/год;

Згідно зі статистикою з сайту DOU [14] – Середня зарообітна плата програміста становить 950 доларів за місяць, тобто за 160 робочих годин, годинна заробітна плата розробника становить 5.625 доларів за годину. Згідно поточного курсу долару до гривні (рис. 3.1), заробітна плата розробника становить 240 гривень за годину.

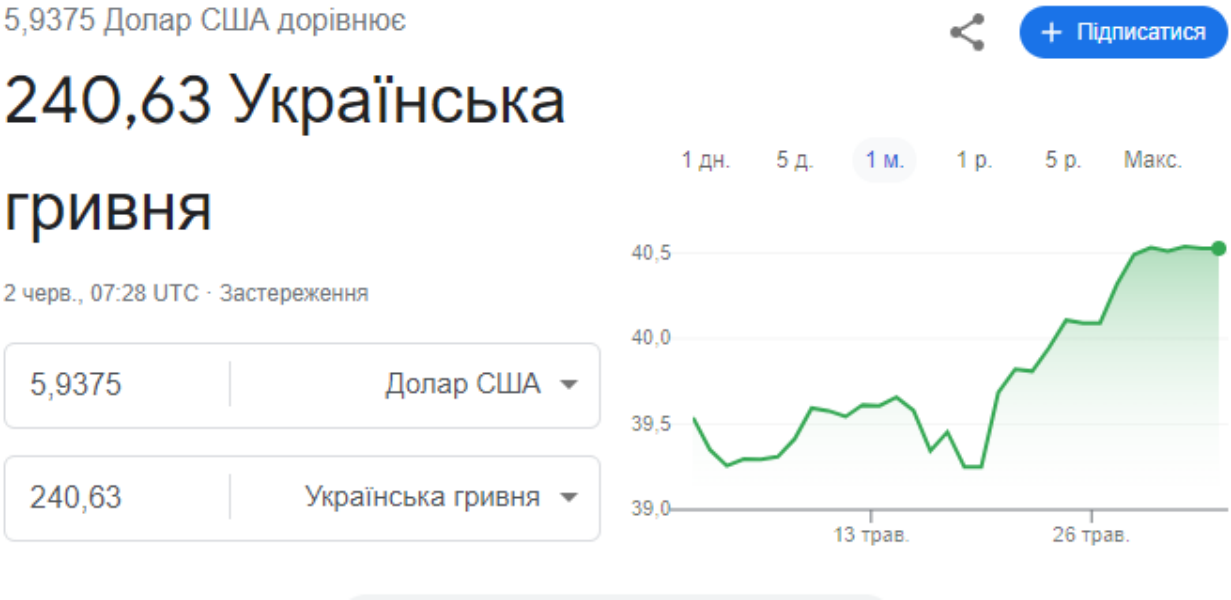

Рис 3.1

– коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі - 1,2;

– коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності - 1,1;

– вартість машино-години ЕОМ – 0.948 грн/год.

Розрахувалось наступним чином :
Згідно з офіційного сайту постачальника електроенергії «Yasno» [15], на момент розробки базовий тариф для населення до 31.05.2024 коштував 2,64 грн/кВт * год.

Всередьному, мій комп’ютер та монітор витрачають 200 ватт/год при роботі з неважкими програмами. Тоді за годину комп’ютер споживає електроенергії на 0,20 * 2,64 = 0.528 грн

Плата за інтернет в моєму місті, становить 330 гривень в місяць [16], або приблизно 0.42 грн/год

Таким чином, вартість машино-години комп'ютера становила 0.528 + 0.42 = 0.948 грн/год

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою

$$t = t_о + t_и + t_а + t_п + t_{отл} + t_д$$

де t_o - витрати праці на підготовку та опис поставленої задачі (приймається 50);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_{д}$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p)$$

де q - передбачуване число операторів;

С - коефіцієнт складності програми;

р - коефіцієнт кореляції програми в ході її розробки.

Після підстановки відповідних значень умовне число операторів дорівнює:

$$Q = 600 * 1{,}25 * (1 + 0{,}05) = 787{,}5$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_и = \frac{Q*B}{(75..85)*k}, \text{ людино-годин,}$$

де В - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

$$t_u = (787{,}5*1{,}2) / (85 * 1{,}1) = 10{,}10 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_а = \frac{Q}{(20..25)*k}, \text{ людино-годин.}$$

Підставивши значення:

t_a = 787,5 / (25 * 1,1) = 28.63 людино-годин.

Витрати на складання програми по готовій блок-схемі розраховується за формулою:

$$t_п = \frac{Q}{(20..25)*k}, \text{ людино-годин.}$$

Після підставлення значень отримуємо витрати на складання програми по готовій блок-схемі:

t_n = 787,5 / (20 * 1,1) = 35.79 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)*k}, \text{ людино-годин.}$$

Підставивши значення:

$t_{отл}$ = 787,5 / (5 * 1,1) = 143.18 людино-годин.

– за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 * t_{отл}, \text{ людино-годин.}$$

$t^k_{отл}$ = 1,5 * 143.18 = 214.77 людино-годин.

Витрати праці на підготовку документації:

$t_д = t_{др} + t_{до}$, людино-годин,

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15..20)*k}, \text{ людино-годин.}$$

$t_{др}$ = 787,5 / (20 *1,1) = 35.79 людино-годин.

$T_{до}$ - трудомісткість редагування, печатки й оформлення документації

$t_{до}$ = 0,75 * $t_{др}$, людино-годин.

$$t_{до} = 0,75 \cdot 35.79 = 26.84 \text{ людино-годин.}$$

Виходить що витрати праці на підготовку документації:

$$t_{д} = 35.79 + 26.84 = 62.63 \text{ людино-годин.}$$

Отже трудомісткість розробки ПЗ:

t = 50 + 62,63 + 143,18 + 35,79+ 28,63 +10,10 = 330.33 людино-години.

3.2 Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн}$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t * С_{ПР}, \text{ грн}$$

де t - загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата розробника становить 240 грн / год, отримуємо:

$$З_{ЗП} = 330,33 * 240 = 79279 \text{ грн.}$$

Вартість машинного часу $З_{МВ}$, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} * С_{МЧ}, \text{ грн,}$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$З_{МВ}$ = 143,18 * 0,948 = 135,73 грн.

Звідси витрати на створення програмного продукту:

$К_{ПО}$ = 79279 + 135,73 = 79414,73 грн.

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k * F_p}, \text{ міс,}$$

де B_k– число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні

F_p = 176 годин).

Звідси, за формулою (3.14) витрати на створення програмного продукту:

T = 330,33 / 1 * 176 = 1,876 міс.

Висновок: Розробка та реалізація комп'ютерної гри "Гомоку" для платформи Android обійдеться в 79414,73 грн. При цьому, розробку здійснюватиме один розробник рівня Junior з трьома роками досвіду у розробці програмного забезпечення. Приблизний час, необхідний для завершення проекту, становить 1,876 місяців при стандартному 40-годинному робочому тижні. Загальна кількість людино-годин, витрачених на розробку, становить 330.33 години.

ВИСНОВКИ

Під час роботи було розроблено комп'ютерну гру "Гомоку" для платформи Android. Гру реалізовано на мові програмування C# з використанням ігрового рушія Unity. Для досягнення поставленої задачі необхідно вирішити основні завдання:

-Вивчення предметної області

-Проектування інтерфейсу

- Реалізація гри

- Імплементація алгоритму для гри проти комп’ютера

-Тестування та відладка

В цілому, цей проект спрямований на створення захоплюючої гри, яка не лише забезпечить розвагу, а й сприятиме розвитку розумових навичок та стратегічного мислення у користувачів.

Під час виконання даної кваліфікаційної роботи також було встановлено складність розробленої системи і здійснено розрахунок вартості роботи зі створення програми, використовуючи середню заробітну плату розробника рівня Junior з трьома роками досвіду у розробці програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nasa, R., et al. (2018) Alpha-Beta Pruning in Mini-Max Algorithm—An Optimized Approach for a Connect-4 Game. International Research Journal of Engineering and Technology, 5, 1637-1641.

2. Ian Milling, J. F. (2009). Articial Intelligence for Games. Elsevier Inc, Miami.

3. Vardi, A. (1992). New minimax algorithm. Journal of Optimization Theory and Applications, 75(3):613–634.

4. Donald E.Knuth, R. W. (1975). An analysis of alpha-beta pruning. Artificial Intelligence, 6(4):293– 326.

5. G.C.Stockman (1979). A minimax algorithm better than alpha-beta. Artificial Intelligence, 12(2):179–196.

6. Philipp Rohlfshagen, Stephen Tavener, D. P. S. S. and Colton, S. (2012). A survey of monte carlo tree search methods. IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, 4(1).

7. Coulom, R. (2007). Efficient selectivity and backup operators in monte-carlo tree search. In van den Herik, H. J., Ciancarini, P., and Donkers, H. H. L. M. J., editors, Computers and Games, pages 72–83, Berlin, Heidelberg. Springer Berlin Heidelberg

8. Guillaume Chaslot, S. r. and Istvan Szita, P. S. (2008). Monte-carlo tree search: A new framework for game artificial intelligence.

9. Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.-A., Puz, N., Weaver, D., and Yerneni, R. (2008). Pnuts: Yahoo!’s hosted data serving platform. Proc. VLDB Endow., 1(2):1277–1288.

10. Rongxiao, Z. (2016). Convolutional and recurrent neural network for gomoku. Master’s thesis, Stanford University.

11. Zhentao Tang, Zhao, D., Kun Shao, and Le Lv (2016). Adp with mcts algorithm for gomoku. In 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pages 1–7.

12. Silver David, H. A. (2016). Mastering the game of go with deep neural networks and tree search. Nature, 529:484–489.

13.Anton, R. (2018). Reevaluation of artificial intelligence engine alpha zero, a self-learning algorithm, reveals lack of proof of best engine, and an advancement of artificial intelligence via multiple roots. Mathematical and Theoretical Physics, 1(2):32–40

14. Середня заробітна плата розробника рівня Junior. URL: https://jobs.dou.ua/salaries/?period=2023-12&position=Junior%20SE&experience=0-5

15. Yasno. Базовий тариф для населення до 31.05.2024. URL: https://yasno.com.ua/bazovij-tarif-dlya-naselennya-2024

16. Київстар. Домашній Інтернет у Марганці. URL: https://kyivstar.ua/home-internet/region/marhanets

17. Методичні рекомендації до виконання кваліфікаційних робіт здобувачів першого рівня вищої освіти спеціальності 122 Комп'ютерні науки/ В.В. Спірінцев, П.О. Іщук, О.С. Шевцова; Д : НТУ «Дніпровська політехніка», 2021. – 59 с.

ДОДАТОК А

КОД ПРОГРАМИ

GameCore.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;
using UnityEngine.UI;

public class GameCore : MonoBehaviour
{
    public GameObject WhitePrefab;
    public GameObject BlackPrefab;
    public Tilemap yourTilemapCollider2D;

    public GameObject TransparentCirclePrefab;
    private GameObject transparentCircle;

    public Text playerTurnText;

    private bool isWhiteTurn = true; // Флаг
    public bool isWin = false;
    public static bool vsAImode = false;

    void Update()
    {
        if (!isWin)
        {
            UpdateTransparentCirclePosition();
        }

    }

    private void UpdateTransparentCirclePosition()
    {
        Vector3 mousePosition = GetCordsMouse();
        Vector3 snappedMousePosition =
GmVector3Extensions.Snap(mousePosition) + new Vector3(0, 0, 5);

        if (transparentCircle == null)
        {
            transparentCircle = Instantiate(TransparentCirclePrefab,
snappedMousePosition, Quaternion.identity);
            transparentCircle.name = "TransparentCircle";
        }
        else
        {
            transparentCircle.transform.position =
snappedMousePosition;
        }
    }


    public Vector3[] GetAdjacentCellCoordinates(Vector3 initialCell,
int radius)
    {
        List<Vector3> adjacentCoordinates = new List<Vector3>();
```

```
            for (int xOffset = -radius; xOffset <= radius; xOffset++)
            {
                for (int yOffset = -radius; yOffset <= radius; yOffset++)
                {
                    Vector3 adjacentCell = initialCell + new
Vector3(xOffset, yOffset, 0);
                    if (Vector3.Distance(initialCell, adjacentCell) <=
radius)
                    {
                        adjacentCoordinates.Add(adjacentCell);
                    }
                }
            }

            return adjacentCoordinates.ToArray();
        }

        private void PlacePrefabAtTransparentCircle()
        {
            if (transparentCircle != null)
            {
                Vector3 transparentCirclePosition =
transparentCircle.transform.position;

                if (vsAImode)
                {
                    isWhiteTurn = true;
                }

                if (transparentCirclePosition.x < 0 ||
transparentCirclePosition.x > 16 ||
                    transparentCirclePosition.y < -17 ||
transparentCirclePosition.y > -2)
                {
                    return; // невозвр
                }



                GameObject existingClone = GameObject.Find($"Clone
{transparentCirclePosition}");
                if (existingClone == null)
                {


                    GameObject prefabToInstantiate = isWhiteTurn ?
WhitePrefab : BlackPrefab;
                    var Clone = Instantiate(prefabToInstantiate,
transparentCirclePosition, Quaternion.identity);
                    Clone.name = $"Clone {transparentCirclePosition}";
                    Clone.tag = "Clone";



                    isWhiteTurn = !isWhiteTurn; // Перек флага


                    if (!vsAImode)
                    {
                        NextTurn();
                    }

                    if (vsAImode)
```

```
                    {

                        AIPlaceCicrle();
                    }

                    //Check3InPodryad();
                    CheckWin();
                }

            }
        }

        private int EvaluatePosition(Vector3 position, Color playerColor)
        {
            int score = 0;

            // Define weights for different scenarios
            int consecutiveWeight = 90;
            int blockedWeight = 40;

            Vector3[] directions = { Vector3.right, Vector3.up, new
Vector3(1, 1, 0), new Vector3(1, -1, 0) };

            foreach (Vector3 dir in directions)
            {
                int count = 0;
                int blockedCount = 0;
                for (int i = 1; i <= 4; i++)
                {
                    Vector3 currentCell = position + i * dir;
                    GameObject currentClone = GameObject.Find($"Clone
{currentCell}");

                    if (currentClone != null)
                    {
                        SpriteRenderer spriteRenderer =
currentClone.GetComponent<SpriteRenderer>();
                        if (spriteRenderer != null)
                        {
                            if (spriteRenderer.color == playerColor)
                            {
                                count++;
                            }
                            else
                            {
                                blockedCount++;
                                break;
                            }
                        }
                    }
                }

                if (count == 3 && blockedCount == 1)
                {
                    score += blockedWeight;
                }
                else
                {
                    score += count * consecutiveWeight;
                }
            }

            return score;
        }
```

```
          void AIPlaceCicrle()
          {
              Vector3              transparentCirclePosition              =
transparentCircle.transform.position;
              int adjacentRadius = Random.Range(1, 3);
              Vector3[]                adjacentCoordinates                =
GetAdjacentCellCoordinates(transparentCirclePosition, adjacentRadius);

              GameObject prefabToInstantiate2 = isWhiteTurn ? WhitePrefab :
BlackPrefab;

              int bestScore = int.MinValue;
              Vector3 bestCoordinate = Vector3.zero;

              foreach (Vector3 coord in adjacentCoordinates)
              {
                  GameObject   existingClone2   =   GameObject.Find($"Clone
{coord}");

                  if (coord.x >= 0 && coord.x < 16 && coord.y >= -17 && coord.y
<= -2 && existingClone2 == null)
                  {
                      int  score  =  Minimax(coord,  5,  false,  int.MinValue,
int.MaxValue, prefabToInstantiate2.GetComponent<SpriteRenderer>().color);
                      if (score > bestScore)
                      {
                          bestScore = score;
                          bestCoordinate = coord;
                      }
                  }
              }

              if (bestScore != int.MinValue)
              {
                  var     Clone2     =     Instantiate(prefabToInstantiate2,
bestCoordinate, Quaternion.identity);
                  Clone2.name = $"Clone {bestCoordinate}";
                  Clone2.tag = "Clone";
              }
          }

          int Minimax(Vector3 position, int depth, bool isMaximizingPlayer,
int alpha, int beta, Color playerColor)
          {
              if (depth == 0 || isWin)
              {
                  return EvaluatePosition(position, playerColor);
              }

              int maxScore = isMaximizingPlayer ? int.MinValue : int.MaxValue;

              Vector3[]                adjacentCoordinates                =
GetAdjacentCellCoordinates(position, 1);

              foreach (Vector3 coord in adjacentCoordinates)
              {
                  GameObject   existingClone   =   GameObject.Find($"Clone
{coord}");
```

```
                if (coord.x >= 0 && coord.x < 16 && coord.y >= -17 && coord.y
<= -2 && existingClone == null)
                {
                    GameObject prefabToInstantiate = isMaximizingPlayer ?
WhitePrefab : BlackPrefab;

                    int score = Minimax(coord, depth - 1,
!isMaximizingPlayer, alpha, beta, playerColor);

                    if (isMaximizingPlayer)
                    {
                        maxScore = Mathf.Max(maxScore, score);
                        alpha = Mathf.Max(alpha, score);
                    }
                    else
                    {
                        maxScore = Mathf.Min(maxScore, score);
                        beta = Mathf.Min(beta, score);
                    }

                    if (beta <= alpha)
                    {
                        break;
                    }
                }
            }

            return maxScore;
        }



        private void DestroyTransparentCircle()
        {
            if (transparentCircle != null)
            {
                Destroy(transparentCircle);
            }
        }


        public void ClonePrefab()
        {


        }




        public void CheckWin()
        {
            int winCount = 5;

            GameObject[] clones =
GameObject.FindGameObjectsWithTag("Clone");

            foreach (GameObject clone in clones)
            {
                if (CheckLine(clone.transform.position, Vector3.right,
winCount))
                {

                    playerTurnText.text = "Horizontal win!";
```

```
                    SpriteRenderer            spriteRenderer            =
clone.GetComponent<SpriteRenderer>();
                    if (spriteRenderer != null)
                    {

                        spriteRenderer.color = Color.yellow;
                    }
                    isWin = true;

                    return;
                }

                if    (CheckLine(clone.transform.position,    Vector3.up,
winCount))
                {

                    playerTurnText.text = "Vertical win!";
                    SpriteRenderer            spriteRenderer            =
clone.GetComponent<SpriteRenderer>();
                    if (spriteRenderer != null)
                    {

                        spriteRenderer.color = Color.yellow;
                    }
                    isWin = true;

                    return;
                }

                if (CheckLine(clone.transform.position, new Vector3(1, 1,
0), winCount))
                {

                    playerTurnText.text = "Diagonal win!";
                    SpriteRenderer            spriteRenderer            =
clone.GetComponent<SpriteRenderer>();
                    if (spriteRenderer != null)
                    {

                        spriteRenderer.color = Color.yellow;
                    }
                    isWin = true;

                    return;
                }

                if (CheckLine(clone.transform.position, new Vector3(1, -1,
0), winCount))
                {

                    playerTurnText.text = "Diagonal win!";
                    SpriteRenderer            spriteRenderer            =
clone.GetComponent<SpriteRenderer>();
                    if (spriteRenderer != null)
                    {

                        spriteRenderer.color = Color.yellow;
                    }
                    isWin = true;

                    return;
                }
            }
        }
```

```
        private bool CheckLine(Vector3 startCell, Vector3 direction, int
count)
        {
            List<GameObject> winningClones = new List<GameObject>();
            Color firstCloneColor = Color.white; // Цвет первого клона в
линии

            for (int i = 0; i < count; i++)
            {
                Vector3 currentCell = startCell + i * direction;
                GameObject    currentClone    =    GameObject.Find($"Clone
{currentCell}");

                if (currentClone == null || currentClone.tag != "Clone")
                {
                    return false;
                }

                SpriteRenderer               spriteRenderer               =
currentClone.GetComponent<SpriteRenderer>();
                if (spriteRenderer != null)
                {
                    if (i == 0)
                    {
                        firstCloneColor    =    spriteRenderer.color;    //
Сохраняем цвет первого клона
                    }
                    else if (spriteRenderer.color != firstCloneColor)
                    {
                        return false;
                    }
                }

                winningClones.Add(currentClone);
            }

            foreach (GameObject clone in winningClones)
            {
                SpriteRenderer               spriteRenderer               =
clone.GetComponent<SpriteRenderer>();
                if (spriteRenderer != null)
                {
                    spriteRenderer.color = Color.yellow;
                }
            }

            return true;
        }

        public void NextTurn()
        {
            //isWhiteTurn = !isWhiteTurn;
            UpdatePlayerTurnText();


        }

        private void UpdatePlayerTurnText()
        {
            if (isWhiteTurn)
            {
                playerTurnText.text = "White player moves";
                playerTurnText.color = Color.white;
```

```
            }
            else
            {
                playerTurnText.text = "Black player moves";
                playerTurnText.color = Color.black;
            }
        }

        public Vector3 GetCordsMouse()
        {
            Vector3                    mousePosition                    =
Camera.main.ScreenToWorldPoint(Input.mousePosition);
            return mousePosition;
        }

        public void OnMouseUp()
        {
            if (!isWin)
            {
                PlacePrefabAtTransparentCircle();
                DestroyTransparentCircle();
            }
        }

    }
```

GmVector3Extensions.cs:

```
    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;

    public static class GmVector3Extensions
    {
        /// <summary>
        /// Snap Vector3 to nearest grid position
        /// </summary>
        /// <param name="vector3">Sloppy position</param>
        /// <param name="gridSize">Grid size</param>
        /// <returns>Snapped position</returns>
        public static Vector3 Snap(this Vector3 vector3, float gridSize =
1.0f)
        {
            return new Vector3(
                Mathf.Round(vector3.x / gridSize) * gridSize,
                Mathf.Round(vector3.y / gridSize) * gridSize,
                Mathf.Round(vector3.z / gridSize) * gridSize);
        }

        /// <summary>
        /// Snap Vector3 to nearest grid position with offset
        /// </summary>
        /// <param name="vector3">Sloppy position</param>
        /// <param name="gridSize">Grid size</param>
        /// <returns>Snapped position</returns>
        public  static  Vector3  SnapOffset(this  Vector3  vector3,  Vector3
offset, float gridSize = 1.0f)
        {
            Vector3 snapped = vector3 + offset;
            snapped = new Vector3(
                Mathf.Round(snapped.x / gridSize) * gridSize,
                Mathf.Round(snapped.y / gridSize) * gridSize,
                Mathf.Round(snapped.z / gridSize) * gridSize);
```

```
        return snapped - offset;
    }
}
```

PlayerSettingsScript.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlaySettingsScript : MonoBehaviour
{
    public GameObject objecttoinacvive;
    public GameObject objecttoactive;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void CnageScene()
    {
        objecttoinacvive.SetActive(false);
        objecttoactive.SetActive(true);
    }
}
```

Playgamebutton.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class playgamebutton : MonoBehaviour
{


    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public int GameStartScene;
    public void StartGame()
    {
        SceneManager.LoadScene(GameStartScene);
        GameCore.vsAImode = false;
```

```
    }

    public void StartGamePvsAI()
    {
        SceneManager.LoadScene(GameStartScene);
        GameCore.vsAImode = true;
    }

}
```

CrditsScript.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CrditsScript : MonoBehaviour
{
    public GameObject objecttoinacvive;
    public GameObject objecttoactive;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }



    public void CnageScene()
    {
        objecttoinacvive.SetActive(false);
        objecttoactive.SetActive(true);
    }

}
```

ДОДАТОК Б

ВІДГУК

Керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

«Розробка комп'ютерної гри "Гомоку" на платформі Андроїд»

Студента групи 122-20-3 Калінкіна Богдана Павловича

Керівник економічного розділу **Л.В. Касьяненко**
доц. каф. ПЕП та ПУ, к.е.н

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Калінкін Б.П.диплом.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Калінкін Б.П.диплом.pdf	Пояснювальна записка до кваліфікаційної роботи у форматі PDF
Програма	
Калінкін Б.П.диплом.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Калінкін Б.П.диплом.ppt	Презентація кваліфікаційної роботи