

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**До захисту допущено:
В.о. Завідувача кафедри
Михайло НОВОТАРСЬКИЙ**

_____ (підпис)

“ ” _____ 2025 р.

**Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою “ Комп’ютерні системи та
мережі”
спеціальності 123 “Комп’ютерна інженерія”**

на тему: Застосунок для карткової гри з використанням штучного інтелекту

Виконав : студент 4 курсу, групи ІО-11
(шифр групи)

_____ **Чуян А. В.** _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____ **асистент кафедри ОТ Гончаренко О. О.** _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ **асистент кафедри ОТ Гончаренко О. О.** _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ **старший викладач кафедри ІСТ Яланецький В. А.** _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025 р.

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ

В.о. Завідувача кафедри

Михайло НОВОТАРСЬКИЙ

_____ (підпис)

“ ” _____ 2025 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Чуяна А. В.

1. Тема проєкту Застосунок для карткової гри з використанням штучного інтелекту керівник проєкту Гончаренко Олександр Олексійович, асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 р. №1705-с
2. Термін здачі студентом закінченого проєкту 11 червня 2025 р.
3. Вихідні дані до проєкту технічна документація, теоретичні дані.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Розділ 1. Огляд існуючих рішень.
Розділ 2. Огляд технологій для розробки системи.
Розділ 3. Деталі розробки системи.
Розділ 4. Дослідження та аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Гончаренко О. О.		

7. Дата видачі завдання «7» лютого 2025 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	<i>Затвердження теми проєкту</i>	12.02.2025-17.02.2025	
2.	<i>Вивчення та аналіз завдання</i>	01.03.2025-20.03.2025	
3.	<i>Розробка архітектури та загальної структури системи</i>	21.03.2025-30.03.2025	
4.	<i>Розробка структур окремих підсистем</i>	14.04.2025-20.04.2025	
5.	<i>Програмна реалізація системи</i>	21.04.2025-14.05.2025	
6.	<i>Оформлення пояснювальної записки</i>	15.05.2025-01.06.2025	
7.	<i>Захист програмного продукту</i>	05.06.2025	
8.	<i>Передзахист</i>	10.06.2025	
9.	<i>Захист</i>	18.06.2025	

Студент-дипломник _____ Андрій ЧУЯН
(підпис)

Керівник проєкту _____ Олександр ГОНЧАРЕНКО
(підпис)

АНОТАЦІЯ

Цей проект присвячений розробці застосунку для карткової гри з використанням штучного інтелекту, здатного приймати рішення на основі моделювання поведінки гравця. Ігрове середовище було створено за допомогою рушія Unity, а для навчання агентів використовувався інструментарій the Unity Machine Learning Agents Toolkit (ML-Agents). Увага була зосереджена на розробці ігрової логіки, системі оцінювання ситуацій та прийняттю стратегічних рішень агентом на основі підкріплювального навчання. Розроблений ШІ здатний адаптувати свою поведінку залежно від змін у грі.

Ключові слова: карткова гра, штучний інтелект, Unity, ML-агенти, підкріплювальне навчання.

ANNOTATION

This project is dedicated to developing an application for a card game that employs artificial intelligence capable of making decisions based on modeling player behavior. The game environment was created using the Unity engine, and the Unity Machine Learning Agents Toolkit (ML-Agents) was utilized for training the agents. The focus was on developing the game logic, the system for evaluating situations, and the agent's strategic decision-making based on reinforcement learning. The developed AI is capable of adapting its behavior in response to in-game changes.

Keywords: card game, artificial intelligence, Unity, ML-agents, reinforcement learning.

справки	Формат	Значення	Найменування	Кіл. листів	№ екземпля	Додаток
			Документація загальна			
			Знову розроблена			
	<i>A4</i>	<i>ІАЛЦ.467200.002 ТЗ</i>	Застосунок для карткової	3		
			гри з використанням			
			штучного інтелекту			
			Технічне завдання			
	<i>A4</i>	<i>ІАЛЦ.467200.003 ПЗ</i>	Застосунок для карткової	66		
			гри з використанням			
			штучного інтелекту			
			Пояснювальна записка			
	<i>A4</i>	<i>ІАЛЦ.467200.004 Д1</i>	Застосунок для карткової	1		
			гри з використанням			
			штучного інтелекту			
			Структурна модель			
			системи			
	<i>A4</i>	<i>ІАЛЦ.4672008.005 Д2</i>	Застосунок для карткової	1		
			гри з використанням			
			штучного інтелекту			
			Функціональна схема			
			(діаграма класів)			
	<i>A4</i>	<i>ІАЛЦ.467200.006 Д3</i>	Застосунок для карткової	1		
			гри з використанням			
			штучного інтелекту			
			Алгоритм дії			
			розробленої системи			
	<i>A4</i>	<i>ІАЛЦ.467200.007 Д4</i>	Застосунок для карткової	65		
			гри з використанням			
			штучного інтелекту			
			Текст програмного коду			

					<i>ІАЛЦ.467200.001 ОА</i>						
Зм	Лист	№ докум.	Підп	Дата							
Розроб		Чуян А.В.			Застосунок для карткової гри з використанням штучного інтелекту Опис альбому		Літ.	Аркуш	Аркушів		
Перев		Гончаренко О.О.						1	1		
							<i>КПІ ім. Ігоря Сікорського ФІОТ ІО-11</i>				

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Застосунок для карткової гри з використанням штучного
інтелекту»

Київ – 2025

ЗМІСТ

НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ	2
ПІДСТАВИ ДЛЯ РОЗРОБКИ	2
МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....	2
ДЖЕРЕЛА РОЗРОБКИ.....	2
ТЕХНІЧНІ ВИМОГИ.....	3
Вимоги до розробленого продукту	3
Вимоги до програмного забезпечення.....	3
Вимоги до апаратної частини	3
ЕТАПИ РОЗРОБКИ	3

					ІАЛЦ.467200.003 ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив	Чуян А. В.				Застосунок для карткової гри з застосуванням штучного інтелекту Технічне завдання	Літ.	Аркуш	Аркушів
Перевірив	Гончаренко О.О.						1	
Реценз.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11		
Н. Контр.								
Затвердив								

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку застосунку для карткової гри з використанням штучного інтелекту, а також на подальшу підтримку та вдосконалення розробленої системи.

Областю застосування даного рішення є інтерактивні розважальні платформи, мобільні додатки, онлайн-ігри, а також освітні проекти, у яких застосування штучного інтелекту сприятиме аналізу стратегій гри, покращенню навчального процесу та розвитку логічного мислення користувачів.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка застосунку для карткової гри, в якому реалізовано штучний інтелект, здатний приймати рішення на основі логіки гри, ймовірностей та ігрової стратегії, що забезпечує цікаве та варіативне ігрове середовище для користувача.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

					ІАЛЦ.467200.002 ТЗ	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Інтуїтивно зрозумілий інтерфейс користувача для гри.
- Можливість гри проти штучного інтелекту зі змінним рівнем складності.
- Реалізація логіки гри згідно з правилами обраної карткової гри
- Візуалізація карт та дій гравців
- Надати вичерпну та зрозумілу документацію для розробленого продукту.
- Наявність документації щодо встановлення, запуску та використання застосунку.

5.2. Вимоги до програмного забезпечення

- OC Windows
- Visual Studio 2015 with C++ Tools component or later and Windows 10+SDK

5.3. Вимоги до апаратної частини

- DX10, DX11, DX12 capable GPUs
- x86, x64 architecture with SSE2 instruction set support

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	12.02.2025-17.02.2025
Вивчення та аналіз завдання	01.03.2025-20.03.2025
Розробка архітектури та загальної	21.03.2025-30.03.2025
Розробка структур окремих частин	14.04.2025-20.04.2025
Програмна реалізація системи	21.04.2025-14.05.2025
Виправлення помилок	15.05.2025-01.06.2025
Оформлення пояснювальної записки	05.06.2025

					ІАЛЦ.467200.002 ТЗ	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

ПОЯСНЮВАЛЬНА ЗАПISKA
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Застосунок для карткової гри з використанням штучного
інтелекту»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ	4
ВСТУП	5
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ	7
1.1 Інтелектуальні системи в комп'ютерних іграх	7
1.1.1 Основні поняття	7
1.2 Огляд способів реалізації штучного інтелекту в комп'ютерних іграх	8
1.2.1 Скінченні автомати станів (FSM).....	8
1.2.2 Поведінкові дерева.....	9
1.2.3 Нейронні мережі та машинне навчання	10
1.2.4 Еволюційні алгоритми.....	13
1.3 Приклади застосування різних підходів для реалізації штучного інтелекту в комп'ютерних іграх	14
1.3.1 Half-Life (1998).....	14
1.3.2 Halo 2	15
1.3.3 Gran Turismo Sophy	17
1.3.4 Galactic Arms Race	18
1.4 Огляд жанрів карткових ігор	19
1.4.1 Ігри на взятки	19
1.4.2 Ігри на скидання карт	20
1.4.3 Ігри на збір комбінацій	21
1.4.4 Ігри на побудову колоди	21
1.4.5 Рольові карткові ігри	22
1.4.6 Колекційні карткові ігри	23
1.4.7 Пасьянси.....	23
РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ	27
2.1 Огляд технологій розробки	27
2.1.1 Рушій Unity	27

					ІАЛЦ.467200.003 ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив	Чуян А. В.				Застосунок для карткової гри з застосуванням штучного інтелекту Пояснювальна записка	Літ.	Аркуш	Аркушів
Перевірив	Гончаренко О.О.						1	66
Реценз.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11		
Н. Контр.								
Затвердив								

2.1.2 The Unity Machine Learning Agents Toolkit (ML-Agents)	28
2.1.3 DOTween	29
2.2 Основні функціональні можливості.....	30
2.3 Загальна архітектура системи.....	31
ВИСНОВОК ДО РОЗДІЛУ 2	33
РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ.....	34
3.1 Опис функціонала системи	34
3.1.1 Вибір карткової гри	34
3.1.2 Функціонал системи	35
3.1.3 Опис користувацького інтерфейсу.....	37
3.2 Розробка основних елементів системи	38
3.2.1 Моделювання карт.....	38
3.2.2 Моделювання руки гравця	39
3.2.3 Управління ставками	41
3.2.4 Оцінювання сили покерної руки	42
3.2.5 Модуль відстеження поточного стану гри	43
3.2.6 Архітектура ігрового агента	44
3.2.7 Реалізація ігрового контролера	45
3.2.8 Реалізація агента з підкріплювальним навчанням.....	47
3.2.9 Реалізація агента з використанням метода Монте-Карло	51
3.2.10 Структура сцени в Unity.....	52
ВИСНОВОК ДО РОЗДІЛУ 3	55
РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ	56
4.1 Огляд розробленої системи системи.....	56
4.1.1 Головне меню	56
4.1.2 Меню вибору складності.....	57
4.1.3 Інтерфейс гри.....	58
4.1.4 Меню паузи.....	60

4.1.5 Меню завершення гри	60
4.2 Порівняння з альтернативними підходами	61
ВИСНОВОК ДО РОЗДІЛУ 4	63
ВИСНОВКИ	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	66

					ІАЛЦ.467200.003 ПЗ	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК СКОРОЧЕНЬ

ІІІ	Штучний інтелект
FSM	(Finite State Machine) Скінченний автомат станів
BT	(Behavior Tree) Поведінкове дерево
NPC	(Non-Player Character) Неігровий персонаж
RL	(Reinforcement Learning) Навчання з підкріпленням
DQN	(Deep Q-Network) Глибока Q-мережа
A*	(A-star) Алгоритм пошуку шляху
BFS	(Breadth-First Search) Пошук у ширину

					ІАЛЦ.467200.003 ПЗ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

У сучасному світі програмного забезпечення дедалі більшої популярності набувають інтелектуальні ігрові застосунки, які поєднують розвагу з технологічними інноваціями. Карткові ігри, що здавна користуються популярністю серед людей різного віку, є ідеальним середовищем для реалізації й випробування алгоритмів штучного інтелекту. Особливо цікавим є поєднання класичних елементів геймплею з сучасними методами моделювання поведінки гравців, аналізу ситуацій та прийняття рішень.

Штучний інтелект у контексті карткових ігор виконує не лише функцію віртуального супротивника, але й забезпечує реалістичність і складність гри, що значно підвищує її навчальний і розважальний потенціал. Алгоритми ШІ дозволяють створити динамічних опонентів, які адаптуються до ігрового процесу, аналізують історію ходів, виявляють закономірності та навіть можуть застосовувати стратегії, близькі до людських, включаючи блеф, оцінку ймовірностей і ризику.

Розробка застосунку для карткової гри з використанням ШІ охоплює низку важливих напрямів: створення ігрового інтерфейсу, моделювання правил гри та реалізація системи прийняття рішень для віртуальних гравців.

В умовах швидкого розвитку обчислювальних можливостей та популярності мобільних і веб-застосунків створення інтелектуальних ігор відкриває нові можливості як для досліджень у галузі штучного інтелекту, так і для широкої аудиторії користувачів, які прагнуть більш глибокого та реалістичного ігрового досвіду.

Метою даної роботи є розробка застосунку для карткової гри з використанням елементів штучного інтелекту, який забезпечуватиме складного, динамічного та адаптивного супротивника. У рамках реалізації буде досліджено сучасні методи ШІ, такі як Монте-Карло симуляції, машинне

					ІАЛЦ.467200.003 ПЗ	Арк.
						5
Зм.	Арк.	№ докум.	Підпис	Дата		

навчання, евристичні правила прийняття рішень, а також проєктування гнучкої структури гри, що дозволяє легко масштабувати її функціональність. Такий застосунок матиме як розважальне, так і навчальне значення, а його розробка сприятиме глибшому розумінню сучасних підходів до створення ігрових систем на основі ШІ.

					ІАЛЦ.467200.003 ПЗ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Інтелектуальні системи в комп'ютерних іграх

1.1.1 Основні поняття

Інтелектуальні системи (або системи зі штучним інтелектом, ШІ) у комп'ютерних іграх - це програмні модулі, які здатні автономно приймати рішення, адаптувати поведінку персонажів до дій гравця, оптимізувати стратегічні дії та імітувати людський інтелект в ігровому середовищі. Такі системи забезпечують динамічність, гнучкість та реалістичність ігрового процесу, особливо в стратегічних та карткових іграх, де важливими є логіка, прогнозування дій суперника, управління ресурсами та прийняття рішень в умовах невизначеності.

Однією з основних характеристик інтелектуальних систем у карткових іграх є стратегічне мислення, яке проявляється в умінні планувати послідовність дій для забезпечення вигравної позиції в майбутньому. До того ж, системи демонструють високий рівень адаптивності, що дозволяє їм навчатися на попередньому досвіді та змінювати свою стратегію відповідно до нової інформації. Також важливою є здатність до імовірнісного моделювання, що дає можливість ефективно працювати з неповною інформацією, як, наприклад, у ситуаціях, коли гравець не знає карти суперника. Крім цього, ці системи характеризуються оптимізацією рішень, що виражається у виборі найбільш ефективного ходу із багатьох можливих дій з урахуванням поточного стану гри.

					ІАЛЦ.467200.003 ПЗ	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

1.2 Огляд способів реалізації штучного інтелекту в комп'ютерних іграх

Штучний інтелект у комп'ютерних іграх є одним із ключових компонентів, що визначає інтелектуальну поведінку неігрових персонажів (NPC), суперників або партнерів гравця. Його основне завдання - забезпечення виклику, реалізму та варіативності ігрового процесу.

Далі буде розглянуто підходи до реалізації штучного інтелекту в іграх.

1.2.1 Скінченні автомати станів (FSM)

Скінченні автомати станів - це потужна математична модель, яка дозволяє описати поведінку об'єкта як набір конкретних "станів" з визначеними правилами переходу між ними. В ігровому ШІ часто використовується саме ця концепція для управління поведінкою NPC [1].

Одним із центральних елементів моделі є стан, який відображає конкретну ситуацію або конфігурацію поведінки персонажа. Наприклад, охоронець може перебувати в одному з декількох станів, таких як патрулювання, настороження, атака чи відступ, що дозволяє системі точно визначати необхідну реакцію на задані умови. Передбачені переходи між цими станами виконуються згідно з певними умовами або подіями, які виступають у ролі тригерів для зміни поточного стану.

Переходи між станами реалізуються на підставі зовнішніх чи внутрішніх умов, що можуть включати як сигнали з навколишнього середовища (наприклад, близькість гравця, звуковий сигнал або зміна освітлення), так і параметри самого персонажа (як-от рівень здоров'я, емоційний стан або інші внутрішні ознаки).

Окрім того, система включає операції, що виконуються при вході до певного стану, під час перебування в ньому або при виході з нього. Наприклад,

					ІАЛЦ.467200.003 ПЗ	Арк.
						8
Зм.	Арк.	№ докум.	Підпис	Дата		

при активації стану атаки NPC може завдати шкоди супернику або видавати специфічні звукові сигнали.

1.2.2 Поведінкові дерева

Поведінкові дерева являють собою методологію організації логіки ігрового штучного інтелекту, що дозволяє створювати динамічні, адаптивні та іноді нелінійні сценарії поведінки NPC [2]. На відміну від класичних підходів, таких як скінченні автомати, поведінкові дерева характеризуються підвищеною реактивністю завдяки безперервній перевірці умов під час виконання, що дає змогу агенту оперативно адаптуватися до змін, наприклад, змінювати свою дію у разі виникнення нової загрози. Крім того, система вирізняється високою модульністю, оскільки кожна поведінка реалізується як окремий блок, який можна змінювати або повторно використовувати без необхідності перебудови всієї системи. При цьому слід зазначити, що масштабування класичних методів, зокрема FSM, може призводити до ускладнення структури системи, у той час як ієрархічна організація поведінкових дерев спрощує як розробку, так і підтримку проекту.

Структурною базою поведінкового дерева є кореневий вузол, який виконує функцію стартової точки, ініціюючи послідовне опитування своїх дочірніх вузлів. Далі йдуть вузли, що відповідають за регулювання логіки виконання, а також спеціалізовані вузли-декоратори, які змінюють поведінку конкретного дочірнього вузла шляхом впровадження додаткових умов або обмежень, що може проявлятися, наприклад, у повторенні дії або інверсії результату її виконання. Завершальним етапом у структурі є вузли, що безпосередньо містять конкретні дії або перевірки умов, які спрямовані на взаємодію зі світом гри, такі як виконання анімації, пересування, атака або аналіз стану оточення.

					ІАЛЦ.467200.003 ПЗ	Арк.
						9
Зм.	Арк.	№ докум.	Підпис	Дата		

Під час виконання поведінкового дерева кореневий вузол звертається до свого прямого нащадка, який, у свою чергу, може мати власних дітей у ієрархічній структурі. Кожний вузол системи повертає один із трьох можливих станів: успіх, невдачу або стан виконання. Такий підхід вирішує завдання обробки ситуацій, коли певна дія потребує часу для виконання, наприклад, пересування до визначеної цілі, або коли необхідно здійснювати вибір альтернативних варіантів дій. Зокрема, якщо контрольний вузол обирає певну дію, система розпочинає з першої альтернативи і переходить до наступної лише у випадку невдачі попереднього виконання.

1.2.3 Нейронні мережі та машинне навчання

Нейронні мережі та машинне навчання відкривають нові можливості для створення динамічних, адаптивних та «живих» агентів. Замість заздалегідь заданих правил поведінки, сучасні системи можуть навчатися на власному досвіді або аналізі великої кількості даних, що дозволяє їм змінювати свою стратегію у відповідь на дії гравця або змінене оточення.

У сучасних іграх застосовуються різноманітні підходи машинного навчання.

Одним з найефективніших методів у цьому контексті є навчання із підкріпленням (Reinforcement Learning, RL). Його сутність полягає у тому, що агенти взаємодіють із оточенням, отримують винагороди або штрафи за виконання певних дій, і з часом навчаються максимально збільшувати суму отриманих винагород [3]. Такий підхід дозволяє створювати адаптивні стратегії поведінки, коли ШІ вчиться оптимально реагувати на нетривіальні ситуації у грі. Прикладами є алгоритми Deep Q-Networks (DQN), Actor-Critic методи чи Proximal Policy Optimization (PPO).

					ІАЛЦ.467200.003 ПЗ	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

Іншим підходом є навчання з учителем (Supervised Learning), коли модель, використовуючи набір даних з прикладами «правильних» рішень (як, наприклад, записи ігрових сесій чи експертних стратегій), може навчитися імітувати ці стратегії [4]. Такий підхід застосовується для прогнозування поведінки гравця, адаптації рівня складності гри або навчання NPC відтворювати людську логіку.

У межах мовних інтерфейсів застосовуються нейромовленнєві технології, що базуються на глибинних нейронних мережах. Вони дозволяють здійснювати аналіз, генерацію та інтерпретацію мовлення. Це забезпечує природну мовну взаємодію між гравцем і персонажами гри, зокрема через голосові команди або діалогові системи.

У тих випадках, коли пряме програмування або підкріплювальне навчання є надто складними, доцільним є використання імітаційного навчання. У цьому підході агенти навчаються відтворювати поведінку, спостерігаючи за демонстраціями експерта (людини або іншого агента) [5].

Окрему нішу займає застосування генеративних нейронних мереж для створення ігрового контенту. Моделі на кшталт Generative Adversarial Networks (GAN) та Variational Autoencoders (VAE) здатні автоматично генерувати візуальні об'єкти, аудіоефекти, сценарії та ігрові рівні, що допомагає урізноманітнити ігровий процес.

За допомогою нейронних мереж і машинного навчання ігровий ІІІ може адаптуватися до стилю гри конкретного користувача, змінювати рівень складності в реальному часі та створювати більш непередбачувану поведінку NPC. Це сприяє більш глибокому зануренню гравця у віртуальний світ.

Замість того, щоб заздалегідь прописувати всі можливі сценарії, розробники можуть покластися на алгоритми, що самостійно «вчаться» на основі даних. Це зменшує обсяг ручного кодування та спрощує оновлення і оптимізацію ІІІ у процесі розробки гри.

					ІАЛЦ.467200.003 ПЗ	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

Нейронні мережі можуть працювати у гармонії з класичними підходами, такими як поведінкові дерева або алгоритми пошуку шляху (A*, Дейкстра, BFS). Наприклад, вони можуть аналізувати «гру в цілому», тоді як традиційні алгоритми відповідають за конкретні завдання, як-от переміщення персонажа.

Незважаючи на численні переваги, впровадження методів машинного навчання в ігрову індустрію супроводжується низкою проблем, що обмежують їхнє широке застосування на практиці. Однією з головних технічних складностей є висока потреба в обчислювальних ресурсах. Навчання нейронних мереж, зокрема глибоких, потребує значної обчислювальної потужності та великого обсягу оперативної пам'яті, а також доступу до спеціалізованого апаратного забезпечення, як-от графічні процесори (GPU) або тензорні процесори (TPU). Це створює додаткові фінансові та інфраструктурні витрати для розробників.

Не менш важливою проблемою є необхідність формування або збирання великих і якісних масивів даних, які є критично важливими для ефективного навчання моделей. У випадках, коли відповідні дані відсутні або їхня кількість є недостатньою, навчання моделей втрачає ефективність або стає зовсім неможливим. Цей виклик є особливо актуальним для вузькоспеціалізованих ігрових жанрів або нових типів ігрових механік, де історичні дані практично відсутні.

Крім того, ігрове середовище часто є високодинамічним і непередбачуваним, що вимагає від систем штучного інтелекту здатності швидко реагувати на зміну умов. У таких ситуаціях може виникати потреба у регулярному перенавчанні моделей або оперативному оновленні їхніх параметрів для збереження релевантності поведінки агента. Це, у свою чергу, ускладнює експлуатацію таких систем у режимі реального часу та потребує додаткових зусиль з боку розробників щодо забезпечення стабільності та продуктивності штучного інтелекту в умовах постійної зміни ігрового контексту.

					ІАЛЦ.467200.003 ПЗ	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

1.2.4 Еволюційні алгоритми

Еволюційні алгоритми представляють собою клас оптимізаційних методів, що імітують природні процеси - відбору, репродукції, мутації та схрещування - для знаходження ефективних рішень у великих і складних просторах можливих варіантів [6]. У контексті ігрового ШІ вони використовуються для автоматичного вдосконалення поведінки агентів, налаштування параметрів системи або навіть для генерації ігрового контенту.

Типовий еволюційний підхід передбачає існування популяції кандидатних рішень, кожне з яких може бути представлене у вигляді бітових рядків, числових векторів або структурованих об'єктів, таких як дерева в межах генетичного програмування. Початкова популяція зазвичай формується випадково або з використанням попередніх емпіричних знань. Якість кожного кандидата оцінюється за допомогою спеціально визначеної фітнес-функції, що слугує критерієм ефективності. У контексті ігрового ШІ така функція може визначати рівень досягнення цілей, ефективність виконання завдань або збалансованість між конфліктуючими поведінковими стратегіями.

Далі здійснюється відбір кращих представників популяції, які мають найвищі показники за фітнес-функцією. Ці рішення передають свої характеристики новому поколінню шляхом операцій кросинговеру, що комбінує генетичні ознаки, та мутації, яка вводить випадкові зміни, забезпечуючи генетичну різноманітність і запобігаючи передчасному сходженню до локальних екстремумів. Процес є ітеративним: нові популяції формуються, знову оцінюються, і цикл повторюється до досягнення бажаного рівня оптимальності або до виконання заздалегідь визначеного критерію зупинки.

Еволюційні алгоритми мають низку суттєвих переваг. Їхня здатність до постійної адаптації дозволяє системам штучного інтелекту ефективно пристосовуватися до нових ігрових умов, а також виявляти нестандартні або

несподівані стратегії, що не були передбачені розробниками. Крім того, такі алгоритми легко інтегруються в модульну архітектуру ШІ, функціонуючи у поєднанні з іншими підходами, зокрема машинним навчанням або поведінковими деревами.

Водночас існують і певні обмеження. Значні обчислювальні витрати, пов'язані з багатьма поколіннями та складністю фітнес-функцій, можуть стати серйозною перешкодою для використання еволюційних методів у реальному часі. Ще одним викликом є необхідність точного формулювання фітнес-функції, яка повинна адекватно відображати бажану поведінку у складному ігровому середовищі. Також слід враховувати ризик зниження генетичної різноманітності, що виникає внаслідок неправильно підібраних параметрів мутації або відбору і може призвести до застрягання в локальних оптимумах без подальшого прогресу.

1.3 Приклади застосування різних підходів для реалізації штучного інтелекту в комп'ютерних іграх

1.3.1 Half-Life (1998)

Штучний інтелект у грі Half-Life (1998) вважається знаковим проривом у розробці NPC-поведінки в іграх від першої особи. Розроблений на рушії GoldSrc, цей ШІ поєднував прості FSM із сценарно-контекстною адаптацією [7], що дозволяло ворогам і союзникам діяти автономно, логічно й реалістично.

Основна мета полягала не тільки у створенні небезпечних супротивників, а й у формуванні NPC, які б виявляли когерентну логіку дій: взаємодіяли між собою, адаптувалися до змін обстановки та реагували на дії гравця. Для цього використовувалась модель поведінки, яка включала стани патрулювання,

					ІАЛЦ.467200.003 ПЗ	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

пошуку, атаки та відступу. Перемикання між цими станами відбувалося в залежності від сенсорних даних, таких як поява гравця у полі зору або отримання пошкоджень, що дозволяло зберігати обчислювальну ефективність і водночас підтримувати складну поведінкову структуру.

Сенсорний модуль, що визначав реакцію NPC, включав виявлення гравця за лінією зору, аналіз рівня шуму (включаючи звуки кроків або пострілів), а також оцінку умов освітлення, що впливали на здатність розпізнавати об'єкти. Навігація персонажів через складні просторові середовища здійснювалася за допомогою попередньо побудованої сітки вузлів у поєднанні з алгоритмом пошуку шляху A*. Це забезпечувало рух до цілі з урахуванням змін у навколишньому середовищі, включаючи динамічні перешкоди.



Рисунок 1.1 – Скріншот гри Half-Life (1998)

1.3.2 Halo 2

Штучний інтелект у грі Halo 2 вважається одним з найбільш інноваційних на момент її виходу (2004). З технічної перспективи, ШІ в цій грі був спрямований не лише на симуляцію поведінки противників, а й на побудову

стратегічної ієрархії та командної взаємодії серед NPC. Система ШІ створювала ілюзію «живого світу», де вороги взаємодіяли, ухилялися, відступали або атакували.

У центрі архітектури штучного інтелекту Halo 2 були поведінкові дерева, що забезпечували адаптивне прийняття рішень на основі поточного контексту. Ці дерева дозволяли NPC обирати дії з урахуванням наявності або відсутності укриттів, здійснювати внутрішньоігрову комунікацію, а також оцінювати рівень загрози та відповідно змінювати тактичну модель поведінки.

Система включала реакцію на поведінку гравця - зміну позицій, застосування гранат, захоплення транспортних засобів. Вороги могли виявляти обхідні шляхи, змінювати патрульні маршрути [8].



Рисунок 1.2 – Скріншот гри Halo 2

1.3.3 Gran Turismo Sophy

Gran Turismo Sophy - це високорівневий ШІ, розроблений компанією Sony AI у співпраці з Polyphony Digital, який демонструє надлюдський рівень навичок керування в реалістичному автосимуляторі Gran Turismo 7. Його розвиток став проривом у застосуванні глибокого навчання з підкріпленням в динамічному симуляційному середовищі з багатьма агентами.

У процесі навчання GT Sophy використовувався алгоритм Quantile-Regression Soft Actor-Critic (QR-SAC), який є сучасною варіацією глибокого навчання з підкріпленням. Цей алгоритм оптимізує поведінкову політику агента з метою досягнення кількох ключових цілей: мінімізації часу проходження кола, підтримання стабільності транспортного засобу на трасі та забезпечення адаптивної реакції на дії інших водіїв, незалежно від того, чи є вони реальними гравцями чи іншими агентами. Для підвищення ефективності навчального процесу команда розробників створила спеціалізований симулятор, який точно відтворює фізичну модель автомобіля у грі Gran Turismo, але не містить графічного інтерфейсу, що дозволяє здійснювати швидке моделювання та прискорене тренування моделей.

У ході тестування GT Sophy продемонстрував здатність до реакцій, що переважають швидкість і точність більшості професійних гравців [9]. Це підтверджує значну перевагу агентів, побудованих на основі глибокого навчання з підкріпленням, порівняно з традиційними підходами до керування, які спираються на заздалегідь визначені алгоритми. Таким чином, Gran Turismo Sophy репрезентує собою один із найпереконливіших прикладів інтеграції сучасних методів машинного навчання у ігрові середовища.

					ІАЛЦ.467200.003 ПЗ	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		



Рисунок 1.3 – Скріншот гри Gran Turismo 7

1.3.4 Galactic Arms Race

Galactic Arms Race - це науково-фантастична гра, яка стала новаторською завдяки інтеграції ШІ для автоматичної генерації ігрового контенту. Її унікальність полягає в тому, що ігрова зброя еволюціонує за допомогою поведінки гравців, використовуючи еволюційні алгоритми - зокрема модифікацію NEAT (NeuroEvolution of Augmenting Topologies), яка отримала назву cgNEAT.

У GAR не використовуються заздалегідь задані шаблони для створення зброї. Замість цього застосовується еволюційна нейромережа, яка аналізує, якими видами зброї користуються гравці, обчислює їхню «популярність» і ефективність та створює нову зброю, яку додає у гру як мутації попередніх версій.

Fitness-функції для зброї базуються не на прямій «смертності», а на використанні та тривалості застосування конкретної зброї. Таким чином, кожна

дія гравця (вибір зброї, успішність в бою) стає вхідними даними для еволюційного процесу [10].



Рисунок 1.4 – Скріншот гри Galactic Arms Race

1.4 Огляд жанрів карткових ігор

1.4.1 Ігри на взятки

Гра на взятки - це жанр карткових ігор, в якому крок гри побудований навколо здобуття так званої взятки. Взятка - це набір карт, викладених на стіл під час одного ходу, які в кінцевому підсумку отримує певний гравець або команда за правилами конкретної гри. Наприклад, у деяких іграх гравець повинен прикладати карту тієї ж масті, що й перша карта у взятці, або використовувати спеціальну карту (трибуну), яка має вищий ранг [11].

Серце жанру полягає в поєднанні елементів випадковості та стратегії. Роздача карт робить кожну партію непередбачуваною, а вибір, коли і яку карту викласти, вимагає ретельного аналізу ситуації та прогнозування ходів

суперників. Деякі ігри цього жанру проводяться індивідуально, а інші - командами, де партнерська взаємодія, комунікація та спільне стратегічне планування відіграють вирішальну роль.

Популярні варіанти ігор на взятки включають, зокрема, Брідж, Хартс, Спейдс, Віст, Euchre та інші. Хоча кожна з них має власні нюанси та додаткові правила, базова логіка залишається незмінною.

1.4.2 Ігри на скидання карт

Ігри на скидання карт - це жанр карткових ігор, у яких головна мета гравця полягає в тому, щоб позбутися всіх карт із своєї руки, дотримуючись певних правил щодо їх викладання на стіл.

У таких іграх кожен крок передбачає ретельний вибір карт, які можна легально "скинути" відповідно до встановлених критеріїв. Зазвичай правила вимагають відповідності за мастю, рангом чи іншими характеристиками, а іноді передбачають можливість застосування спеціальних карт, які змінюють хід гри, наприклад, змушують суперника брати додаткові карти або змінюють напрям обертання гри.

Жанр характеризується балансом між випадковістю та стратегією. Роздача карт задає початковий рівень невизначеності, тоді як грамотне управління рукою дозволяє гравцю впливати на хід гри. Гравцям необхідно аналізувати ситуацію, враховувати вже зіграні карти, а інколи й передбачати ходи суперників.

Серед найвідоміших представників цього жанру в країнах Східної Європи - популярна гра «Дурень». Однак ігри на скидання карт не обмежуються лише традиційними варіантами, їх популярність спонукала розробку сучасних цифрових версій, де класична механіка поєднується з новими елементами, зручними інтерфейсами та онлайн-змаганнями.

1.4.3 Ігри на збір комбінацій

Ігри на збір комбінацій - жанр карткових ігор, в якому головною метою є збір певних наборів карт, що відповідають визначеним умовам перемоги. У цих іграх основною механікою є формування комбінацій - це можуть бути набори карт однакового рангу, послідовних значень або інші специфічні поєднання, прописані правилами гри. Завдяки поєднанню випадковості, що виникає від роздачі карт, і необхідності ретельного стратегічного планування, гравці постійно аналізують ситуацію, обирають оптимальні ходи та намагаються передбачити дії суперників.

Класичні приклади цього жанру показують, як різні ігри інтегрують механіку збору комбінацій для створення унікальних стратегічних можливостей. Наприклад, у Rummy гравці формують комбінації з послідовних або однакових карт, тоді як ігри на зразок Ticket to Ride, Splendor чи Azul використовують збір карт для побудови маршрутів, накопичення ресурсів або створення візуально привабливих композицій. Простота основних правил робить ці ігри доступними для широкої аудиторії, водночас відкриваючи простір для глибокого стратегічного мислення, що дозволяє досвідченим гравцям віднаходити нові тактичні підходи.

1.4.4 Ігри на побудову колоди

Ігри на побудову колоди - це жанр карткових ігор, у яких гравці поступово формують власну колоду карт у процесі гри, додаючи нові карти та вдосконалюючи її. На відміну від традиційних карткових ігор, де гравці починають із фіксованої колоди, у колодобудівних іграх стартовий набір карт зазвичай невеликий і слабкий, а основна механіка полягає у придбанні нових карт, які покращують можливості гравця.

					ІАЛЦ.467200.003 ПЗ	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

Кожен хід гравець тягне кілька карт зі своєї колоди, використовує їх для виконання дій або отримання ресурсів, а потім може купувати нові карти з доступного ринку. У міру розвитку гри колода стає потужнішою, дозволяючи виконувати складніші комбінації та стратегічні маневри. Важливим аспектом жанру є баланс між розширенням колоди та її оптимізацією - надмірне додавання карт може зробити її менш ефективною, тому гравці часто мають можливість видаляти слабкі або непотрібні карти.

Однією з перших ігор, що популяризувала цей жанр, стала Dominion, яка встановила основні принципи колодобудівних ігор. Згодом з'явилися такі відомі представники жанру, як Star Realms, Ascension, Clank!, Legendary: A Marvel Deck Building Game та багато інших. Деякі ігри поєднують механіку побудови колоди з іншими елементами, такими як пригодницькі або кооперативні механіки, що додає глибини ігровому процесу.

1.4.5 Рольові карткові ігри

Рольові карткові ігри - це жанр, який поєднує елементи традиційних рольових ігор із механікою використання карт як основного засобу взаємодії. У цих іграх кожна карта може символізувати персонажа, здібність, подію або локацію

Завдяки інтеграції аспектів створення та розвитку персонажа, вибору стратегічних ходів і прийняття рішень, рольові карткові ігри дозволяють створити атмосферу занурення у фантастичний світ

Прикладами рольових карткових ігор є Munchkin, Pathfinder Adventure Card Game, Arkham Horror: The Card Game, Lord of the Rings: The Card Game та багато інших.

					ІАЛЦ.467200.003 ПЗ	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

1.4.6 Колекційні карткові ігри

Колекційні карткові ігри представляють собою жанр, у якому основна ідея полягає в тому, що кожен гравець формує власну унікальну колоду за допомогою карт, здобутих шляхом покупки, обміну чи отримання у вигляді нагород. У цих іграх карти мають різноманітні властивості, силу та рівні рідкісності, що створює складну мережу стратегічних взаємозв'язків. Гравець використовує свою колоду для участі в поєдинках, де важливу роль відіграє синергія між картами, здатність адаптувати стратегію під конкретного суперника та постійне вдосконалення власного ігрового арсеналу. Завдяки регулярним розширенням і виходу нових наборів карт, жанр постійно розвивається.

Яскравим прикладом цього жанру є Magic: The Gathering - перша комерційно успішна колекційна карткова гра, що започаткувала сучасний підхід до побудови колод.

1.4.7 Пасьянси

Пасьянси - це жанр карткових ігор, орієнтований на індивідуальну гру, де головним завданням гравця є впорядкування карт згідно з певними правилами для досягнення визначеної мети. Ці ігри відрізняються від змагальних форматів тим, що кожна партія проходить у самотньому режимі, і успіх залежить виключно від логічного мислення, уважності та передбачення наслідків прийнятих рішень. Гравець зазвичай маніпулює картами так, щоб послідовно перемістити їх з однієї зони в іншу або створити стрічку впорядкованих комбінацій, що дозволяє «очистити» ігрове поле.

Одним із класичних прикладів пасьянсів є «Клондайк», знайомий майже кожному користувачеві комп'ютера, який став синонімом цього жанру завдяки своїй добре продуманій механіці, що поєднує випадковість стартового розкладу

					ІАЛЦ.467200.003 ПЗ	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

з логічним плануванням ходів. Інший популярний варіант - «Павук», де використовується кілька колод для побудови послідовних рядків.

					ІАЛЦ.467200.003 ПЗ	Арк.
						24
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було розглянуто фундаментальні підходи до побудови інтелектуальних систем у комп'ютерних іграх, що забезпечують адаптивну, стратегічну та реалістичну поведінку віртуальних гравців. Проаналізовано ключові методи реалізації штучного інтелекту - скінченні автомати станів, поведінкові дерева, нейронні мережі, методи машинного навчання та еволюційні алгоритми. Кожен із них має свої переваги, обмеження й сфери застосування: від простого управління NPC до створення самонавчальних агентів, здатних адаптуватися до непередбачуваної поведінки гравця.

Огляд показав, що ефективність ШІ в іграх значною мірою залежить від контексту - жанру, цілей гравця, типу взаємодії та технічних обмежень. Наприклад, класичні підходи, як-от FSM і поведінкові дерева, продемонстрували високу результативність у таких іграх, як Half-Life чи Halo 2, завдяки своїй передбачуваності та низьким обчислювальним витратам. Водночас сучасні методи, як глибоке навчання з підкріпленням у Gran Turismo Sophy чи еволюційні алгоритми в Galactic Arms Race, виявили потенціал для створення складної, динамічної й адаптивної поведінки.

Особливої уваги заслуговує поєднання класичних і сучасних підходів, що дозволяє досягти балансу між контрольованістю, продуктивністю та варіативністю ігрового процесу. Це відкриває перспективи для використання комбінованих моделей, які можуть ефективно працювати в умовах складних сценаріїв.

Окремий інтерес становлять карткові ігри, що створюють унікальні виклики для систем ШІ: необхідність роботи з неповною інформацією, стратегічне прогнозування, ймовірнісне мислення та симуляція людської поведінки. Розділ 1.4 класифікує жанри карткових ігор - від ігор на взятки до

					ІАЛЦ.467200.003 ПЗ	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

пасьянсів - що демонструє різноманітність механік і завдань для інтелектуальних агентів.

Таким чином, цей розділ створює цілісну теоретичну основу для подальшої розробки ігрового застосунку з використанням ШІ, адаптованого до специфіки карткових ігор.

					ІАЛЦ.467200.003 ПЗ	Арк.
						26
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ

2.1 Огляд технологій розробки

2.1.1 Рушій Unity

Для розробки застосунку буде використано рушій Unity.

Unity є оптимальним вибором для розробки застосунку для карткової гри з інтеграцією штучного інтелекту завдяки своїй гнучкості, багатофункціональності та широким можливостям як у сфері графіки, так і у програмній логіці. По-перше, рушій надає надзвичайно потужний інструментарій для розробки 2D-візуальних рішень, що особливо важливо для карткових ігор.

Крім того, використання мови програмування C# спрощує реалізацію складної логіки гри та інтеграцію компонентів штучного інтелекту. Завдяки таким рішенням, як ML-Agents, розробники можуть впроваджувати алгоритми, які дозволяють AI адаптувати свою поведінку, аналізувати дії користувача та постійно вдосконалювати свою стратегічну реакцію. Такий підхід надає можливість створити динамічного противника, що робить гру більш інтригуючою та варіативною.

Ще одна вагома перевага Unity полягає у його мультиплатформеності. Розробка застосунку на цій платформі дозволяє експортувати проєкт на широкий спектр операційних систем і пристроїв, таких як Windows, macOS, iOS, Android, WebGL та інші. Це значно розширює аудиторію та забезпечує доступність продукту на різних ринках.

Інфраструктура Unity також підтримується величезною спільнотою розробників та ресурсною базою через Asset Store. Можливість швидкого

					ІАЛЦ.467200.003 ПЗ	Арк.
						27
Зм.	Арк.	№ докум.	Підпис	Дата		

доступу до готових моделей, звукових ефектів, анімаційних рішень та інших ресурсів дозволяє прискорити процес розробки та звільнити час для впровадження унікальних функцій гри. Крім того, модульна архітектура рушія сприяє ізольованій розробці різних компонентів, що дозволяє легко масштабувати продукт і оптимізувати його продуктивність.

Таким чином, Unity забезпечує високий рівень інтуїтивного користувацького інтерфейсу, надійну основу для впровадження адаптивних AI-рішень, підтримку мультиплатформеності і ефективну організацію коду, що робить його ідеальним інструментом для розробки карткових ігор.

2.1.2 The Unity Machine Learning Agents Toolkit (ML-Agents)

Для створення агенту на базі машинного навчання буде використано The Unity Machine Learning Agents Toolkit.

The Unity Machine Learning Agents Toolkit (ML-Agents) представляє собою потужне розширення для рушія Unity, яке дозволяє інтегрувати сучасні методи машинного навчання безпосередньо у процес розробки ігор. У контексті карткової гри застосування цього інструменту відкриває можливості для створення адаптивного штучного інтелекту, який може ефективно навчатися приймати рішення на основі численних симуляцій і самостійно вдосконалювати свою стратегію.

Використання ML-Agents дає змогу симулювати тисячі ігрових сценаріїв у контрольованому середовищі Unity, що є особливо корисним для карткових ігор, де результат матчу залежить від комбінацій карт, змінних умов гри та стратегічних рішень. Такий підхід дозволяє уникнути жорстко заданих алгоритмів та обмежених сценаріїв, адже алгоритми навчання з підкріпленням поступово формують оптимальну тактику за рахунок аналізу результатів численних ітерацій гри.

					ІАЛЦ.467200.003 ПЗ	Арк.
						28
Зм.	Арк.	№ докум.	Підпис	Дата		

Однією з вагомих переваг ML-Agents є його тісна інтеграція з самим Unity, що дозволяє розробнику використовувати єдине середовище для створення як графічного інтерфейсу, так і логіки гри. Розробка та налагодження штучного інтелекту відбувається в тому ж редакторі, що і основний застосунок, що значно спрощує процес тестування, моніторингу поведінки агента та інтеграції результатів навчання в реальний продукт.

До того ж, ML-Agents має широку підтримку спільноти та безліч практичних прикладів, що істотно скорочує час, необхідний для впровадження інноваційних AI-рішень. Розробники можуть використовувати готові шаблони для початкових налаштувань, а потім адаптувати їх до специфіки карткової гри, експериментуючи з різними моделями для визначення оптимального варіанту поведінки супротивника.

2.1.3 DOTween

Для реалізації ігрових анімацій у процесі розробки буде використано бібліотеку DOTween - популярне tweening-рішення для Unity, яке суттєво спрощує створення динамічних і плавних візуальних ефектів. У контексті розробки карткової гри ця бібліотека є особливо доцільною завдяки низці її технічних і функціональних переваг.

Однією з ключових характеристик DOTween є висока оптимізованість, що дозволяє зберігати продуктивність навіть за наявності великої кількості одночасних анімаційних об'єктів. Завдяки ефективному управлінню ресурсами можлива безперебійна реалізація таких ефектів, як переміщення, масштабування або обертання карт, що забезпечує загальну плавність і чутливість ігрового процесу.

Бібліотека також вирізняється простотою використання. Її API побудовано таким чином, щоб забезпечити легкість у створенні складних анімаційних

					ІАЛЦ.467200.003 ПЗ	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

ланцюгів без необхідності у глибокому зануренні в низькорівневу реалізацію. Це дозволяє розробникам швидко досягати бажаного візуального результату, спрощуючи етап прототипування та подальшої інтеграції.

Крім того, DOTween надає широкий спектр можливостей для налаштування поведінки анімацій. Розробники можуть застосовувати різноманітні функції згладжування (easing), визначати затримки, а також поєднувати декілька анімацій у послідовності або паралельні групи

Інтеграція DOTween із Unity відбувається без зайвих ускладнень, оскільки бібліотека створена спеціально з урахуванням архітектури рушія. Це дозволяє безпосередньо працювати як з UI-компонентами, так і з геймплейними об'єктами, зберігаючи єдину систему анімацій

Завдяки активній підтримці спільноти та наявності ґрунтовної документації, DOTween є стабільним інструментом, що супроводжується великою кількістю прикладів, навчальних матеріалів і відповідей на найпоширеніші питання. Така інформаційна база сприяє зменшенню часу на вирішення проблемних ситуацій під час розробки.

У підсумку, застосування DOTween дозволяє значно підвищити візуальну привабливість карткової гри, створюючи плавний і захопливий користувацький досвід. Зокрема, можливість легко реалізовувати ефекти перевертання, масштабування чи переміщення карт дозволяє зробити інтерфейс динамічним, зрозумілим та естетично привабливим.

2.2 Основні функціональні можливості

Основна мета проекту - створити інтерактивний застосунок для карткової гри з інтегрованим штучним інтелектом, що забезпечуватиме динамічний і адаптивний ігровий процес. Основні функції, які планується реалізувати, включають:

					ІАЛЦ.467200.003 ПЗ	Арк.
						30
Зм.	Арк.	№ докум.	Підпис	Дата		

Проект має на меті створення інтерактивного застосунку для карткової гри з інтегрованим модулем штучного інтелекту, який забезпечуватиме динамічний, адаптивний ігровий процес. Ключовим завданням є реалізація повноцінної ігрової логіки, що регулюватиме правила, визначатиме результати, контролюватиме послідовність ходів і виявлятиме переможця. Ця логіка повинна бути стабільною та перевіреною на різноманітних сценаріях для забезпечення її надійності та коректного функціонування.

Важливим компонентом проекту є розробка користувацького інтерфейсу, який дозволить гравцеві взаємодіяти з грою у зручний та інтуїтивний спосіб. Візуальне середовище включатиме динамічне розміщення карт, плавні анімації переходів, візуальні відгуки на дії користувача та відображення результатів гри, що забезпечить загальну цілісність ігрового досвіду.

Особливу увагу приділено реалізації модуля штучного інтелекту, призначеного для аналізу поточної ігрової ситуації та прийняття стратегічно обґрунтованих рішень. AI-модуль має враховувати стан гри, історію попередніх ходів, динаміку ігрового процесу та адаптувати свою поведінку відповідно до змін у діях користувача, демонструючи елементи навчання й адаптації.

2.3 Загальна архітектура системи

Архітектура застосунку проектується за принципами модульності, відповідно до яких кожна підсистема відповідає за виконання окремого функціонального набору завдань. Основною складовою візуальної частини є ігровий інтерфейс, реалізований у середовищі Unity. Цей компонент відповідає за відображення всіх візуальних елементів, керування анімацією, сповіщеннями та забезпечення інтерактивної взаємодії з користувачем. Усі події, ініційовані гравцем, зокрема натискання, переміщення курсора або вибір ігрових об'єктів, опрацьовуються саме на цьому рівні.

					ІАЛЦ.467200.003 ПЗ	Арк.
						31
Зм.	Арк.	№ докум.	Підпис	Дата		

Логіка гри виступає як центральна обчислювальна підсистема, що регулює основні правила карткової гри, здійснює перевірку допустимості ходів, виконує розрахунок балів і визначає результат кожної ігрової ситуації. Реалізація цієї компоненти відбувається мовою програмування C#, що дозволяє забезпечити необхідну гнучкість і продуктивність внутрішньої логіки.

Інтегрований модуль штучного інтелекту тісно взаємодіє з логікою гри та виконує функції аналізу поточного стану ігрового процесу. Він формує стратегічні рішення на основі заданих правил або шляхом застосування алгоритмів машинного навчання, реагуючи на зміну ігрових умов у режимі реального часу.

Ефективна взаємодія між усіма структурними модулями досягається за допомогою внутрішніх інтерфейсів та механізму обробки подій, що забезпечує узгоджений обмін даними та стабільну роботу застосунку як єдиного функціонального цілого.

					ІАЛЦ.467200.003 ПЗ	Арк.
						32
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було обґрунтовано вибір технологій, визначено функціональні вимоги та сформульовано загальну архітектуру застосунку. Unity було обрано як основну платформу для розробки завдяки її широким можливостям у сфері графіки, підтримці C#, модульності, мультиплатформеності та активній спільноті. Інструмент ML-Agents, що інтегрується з Unity, забезпечує ефективну реалізацію адаптивного штучного інтелекту, здатного навчатися на основі симуляцій та підкріплення. Додатково використання DOTween дозволяє створити плавний, привабливий і динамічний інтерфейс, що позитивно впливає на ігровий досвід користувача.

Було визначено основні функціональні блоки системи: реалізація ігрової логіки, створення користувацького інтерфейсу, впровадження модуля штучного інтелекту та організація збереження даних. Проєкт передбачає модульну архітектуру з чітким розмежуванням відповідальностей між компонентами, що сприятиме масштабованості, гнучкості в розробці та легкості в налагодженні системи. Таким чином, закладено надійне підґрунтя для подальшої реалізації інтелектуального застосунку в жанрі карткових ігор.

					ІАЛЦ.467200.003 ПЗ	Арк.
						33
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1 Опис функціонала системи

3.1.1 Вибір карткової гри

У контексті розробки застосунку, що поєднує карткову гру та штучний інтелект із використанням машинного навчання, вибір конкретної гри відіграє ключову роль. [12, 13] Гра повинна бути достатньо складною, щоб вимагати глибокого стратегічного мислення, мати широку варіативність ситуацій для навчання агента, а також бути добре задокументованою в академічному та ігровому середовищі. Ідеально, якщо обрана гра включає як елементи неповної інформації, так і багатоступеневий процес прийняття рішень, що дозволяє ефективно демонструвати роботу алгоритмів ШІ у складних умовах.

Саме тому для даної дипломної роботи було обрано семикартковий стад покер. Цей різновид покеру поєднує у собі достатню глибину стратегії, динамічність розіграшів і обмежену, але прогресивно зростаючу інформацію про карти суперників. На відміну від популярного техаського холдему, стад покер не використовує спільні карти, що робить кожну ситуацію унікальною та ускладнює передбачення рук опонентів. Це створює сприятливі умови для побудови і навчання штучного інтелекту, здатного адаптуватися до мінливої ігрової інформації. Крім того, стад покер включає кілька раундів ставок, що дозволяє реалізувати в агенті здатність до поступового оновлення стратегії та оцінки шансів на перемогу залежно від дій інших гравців. Така структура гри робить її придатною як для реалізації логіки на основі ймовірнісних методів (наприклад, Монте-Карло симуляцій), так і для навчання агентів за допомогою алгоритмів підкріплення. Таким чином, стад покер є оптимальним вибором для

					ІАЛЦ.467200.003 ПЗ	Арк.
						34
Зм.	Арк.	№ докум.	Підпис	Дата		

дослідження можливостей штучного інтелекту в контексті складних багатокрокових стратегічних ігор із неповною інформацією.

3.1.2 Функціонал системи

Функціонал системи, розробленої в межах дипломної роботи, охоплює повний цикл гри в семикартковий стад покер з участю як людського гравця, так і штучного інтелекту, з можливістю спостереження та аналізу ігрового процесу. Система реалізує механіку гри згідно з правилами стад покеру: автоматичну роздачу карт, відображення відкритих та прихованих карт гравців, послідовність етапів гри (включаючи кожен з раундів торгів), обчислення сили покерних комбінацій та визначення переможця.

Штучний інтелект у системі представлений кількома типами гравців: простими алгоритмічними супротивниками, що керуються базовими евристиками, та навчуваним агентом, який використовує алгоритми машинного навчання, зокрема підкріплювальне навчання. Агент аналізує поточний стан гри, включаючи власні карти, відкриті карти опонентів, історію ставок та позицію за столом, і на основі цих даних приймає рішення про наступну дію - фолд, кол, рейз або чек.

Спершу створимо блок-схему, що відображає послідовність основних кроків стад покеру.

					ІАЛЦ.467200.003 ПЗ	Арк.
						35
Зм.	Арк.	№ докум.	Підпис	Дата		

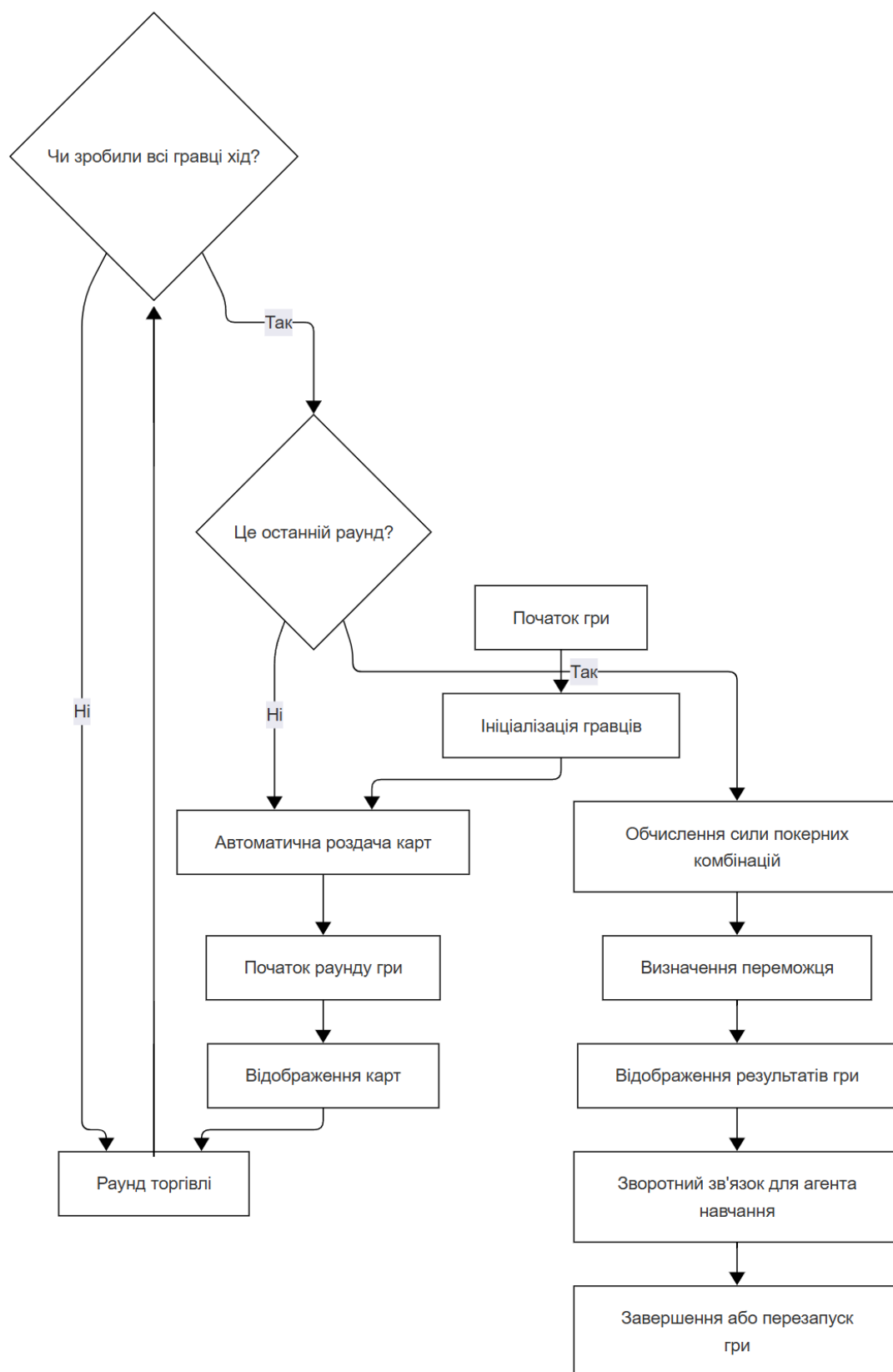


Рисунок 3.1 – Блок схема логіки гри

Зм.	Арк.	№ докум.	Підпис	Дата

Дана блок-схема демонструє повний цикл гри. Спочатку відбувається ініціалізація гри та гравців, після чого система автоматично роздає карти. Далі генерується основний цикл гри, який включає відображення як відкритих, так і прихованих карт, проведення раундів торгів із прийняттям рішень (фолд, кол, рейз або чек) і перевіркою, чи зробили всі учасники свій хід. Після завершення торгів, за допомогою модуля обчислення сили покерних комбінацій, визначається переможець. Нарешті, результати гри відображаються, а агент отримує зворотний зв'язок для подальшого вдосконалення.

3.1.3 Опис користувацького інтерфейсу

Користувацький інтерфейс розробленої системи семикарткового стад покеру забезпечує інтерактивне та інтуїтивно зрозуміле середовище для взаємодії користувача з ігровим процесом. Його структура побудована з урахуванням традиційної візуальної організації покерного столу, що дозволяє гравцеві легко орієнтуватися в елементах керування та спостерігати за діями супротивників.

Центральним елементом інтерфейсу є ігровий стіл, візуально поданий у вигляді овальної або прямокутної області з розташованими по периметру позиціями гравців. У верхній частині кожної позиції відображається аватар гравця з ім'ям та числовим індикатором поточного балансу фішок. Поряд розміщено карти, які поділяються на відкриті та закриті згідно з правилами стад покеру. Відкриті карти орієнтовані до інших гравців, у той час як закриті карти подані у вигляді перевернутих сорочкою догори [14].

У центрі столу розташовано віджет з цифровим індикатором загального банку, який динамічно оновлюється в процесі гри. Він відіграє важливу роль у зворотному зв'язку для користувача, дозволяючи відстежувати внески гравців у поточному раунді ставок. Кожен гравець також має власний індикатор кількості

					ІАЛЦ.467200.003 ПЗ	Арк.
						37
Зм.	Арк.	№ докум.	Підпис	Дата		

фішок, що розташований безпосередньо під або над його аватаром, забезпечуючи постійне інформування про стан кожного учасника [14].

Панель керування користувача розміщено в нижній частині екрана. Вона містить набір інтерактивних кнопок, які відповідають можливим ігровим діям: "Fold" (скинути карти), "Call" (зрівняти ставку), "Check" (пропустити хід без ставки), "Raise" (підвищити ставку), "Small Bet" та "Big Bet" (розмірені ставки згідно з поточним етапом гри). Кожна кнопка реалізована як окремий графічний елемент з можливістю активації або деактивації залежно від ігрового контексту. Наприклад, кнопки "Check" або "Call" стають недоступними, якщо на попередньому кроці було зроблено рейз, що потребує відповіді.

3.2 Розробка основних елементів системи

3.2.1 Моделювання карт

Фундаментальним класом у системі є Card, який реалізовано як серіалізований об'єкт ScriptableObject. Такий підхід дозволяє зберігати метаінформацію про карти як окремі об'єкти у проєкті Unity, що спрощує створення колоди та інспектування карт вручну [15].

Карта містить властивості масті (CardSign), значення (CardValue), графічного зображення (Sprite artWork) та стану: перевернута чи ні (isFlipped), а також чи доступна для перегляду іншими гравцями (isVisibleToOthers). Також реалізовані методи CompareTo і Equals, що дозволяє порівнювати карти для сортування та визначення рівності

Клас CardDisplay відповідає за відображення однієї карти у сцені. Зображення карти (арти) встановлюється відповідно до властивості artWork об'єкта Card. Метод Flip() реалізує анімацію перевертання карти за допомогою бібліотеки DOTween.

					ІАЛЦ.467200.003 ПЗ	Арк.
						38
Зм.	Арк.	№ докум.	Підпис	Дата		

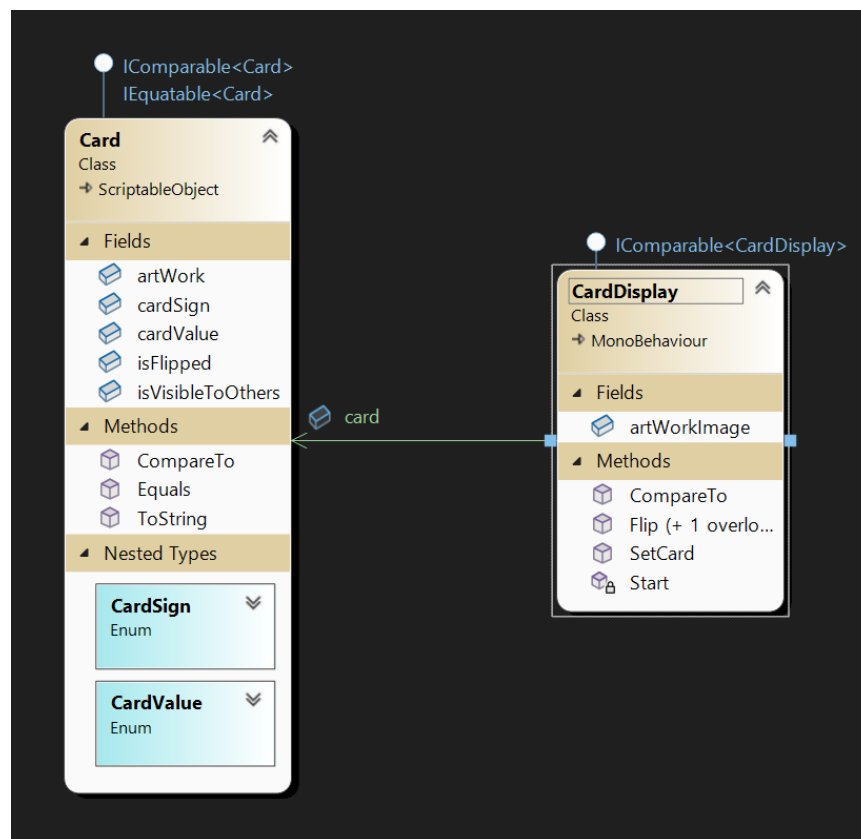


Рисунок 3.2 – Діаграма класів моделі карт

3.2.2 Моделювання руки гравця

Клас CardStack моделює руку гравця - послідовність карт, з якою той бере участь у грі. Важливою особливістю є підтримка подій – вони інформують систему візуалізації про зміни стану руки, дозволяючи динамічно оновлювати інтерфейс. Методи Push() і Pop() реалізують додавання та вилучення карт, а метод Shuffle() - перемішування руки, що може використовуватись при ініціалізації нової гри.

Клас CardStackView інкапсулює логіку оновлення графічного представлення руки гравця. У ньому реалізовано механізм обробки подій зі CardStack, створення об'єктів-карт з префабу, сортування карт та їх розміщення за допомогою сплайнів (SplineContainer).

Кожна карта розміщується вздовж заданого сплайну, що дозволяє формувати зручну та естетично привабливу викладку карт. Під час відкриття руки (Open()) змінюється контейнер сплайнів, що оновлює форму руки та її орієнтацію, відповідно до стадії гри.

Метод SortHandCards() також враховує видимість карт при сортуванні - відкриті карти завжди подаються окремо та з пріоритетом у відображенні.

Таким чином, взаємодія між логічними та візуальними компонентами у розробленій системі реалізована через подійно-орієнтовану архітектуру. Кожна зміна у стані руки гравця відображається у UI за допомогою взаємодії між CardStack, CardDisplay та CardStackView. Подібна декомпозиція забезпечує можливість, за необхідності, легко змінювати способи відображення руки.

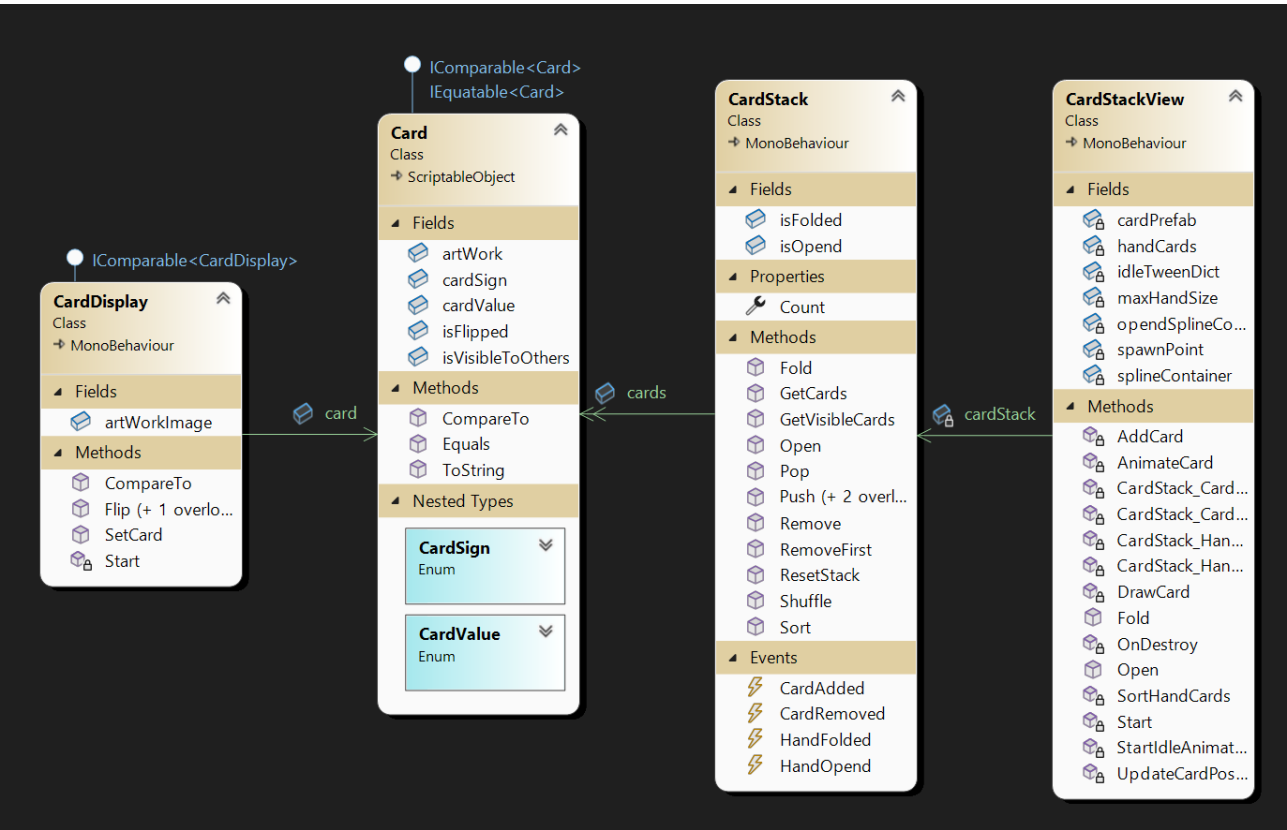


Рисунок 3.3 – Діаграма класів моделі руки гравця

3.2.3 Управління ставками

Одним із ключових аспектів реалізації покерної гри є моделювання механіки ставок. Для досягнення цього було реалізовано модуль керування ставками, представлений у вигляді трьох основних класів: BetManager, BetView та PotManager.

Клас BetManager відповідає за логіку розміщення ставки гравцем, зберігає поточну кількість фішок, актуальну ставку гравця в поточному раунді (currentBet), а також кількість зроблених ставок у раунді (betCounter). Метод Bet() забезпечує основну функціональність - перевірку можливості ставки, зміну стану гравця та додавання фішок до банку. Важливим є використання події PlayerBet, яка дозволяє іншим компонентам системи реагувати на зміну ставки без прямої залежності від логіки управління нею. Такий підхід забезпечує низький рівень зв'язності між модулями.

Клас PotManager реалізує управління ігровим банком, дозволяючи додавати ставки гравців через метод Add(int amount) та скидати банк за допомогою ResetPot(). Завдяки прямій візуалізації через елемент інтерфейсу TextMeshProUGUI, значення банку динамічно оновлюється в реальному часі.

Для відображення поточної ставки окремого гравця використовується клас BetView, який підписується на подію PlayerBet, що генерується в BetManager. Це дозволяє відокремити логіку гри від візуального представлення.

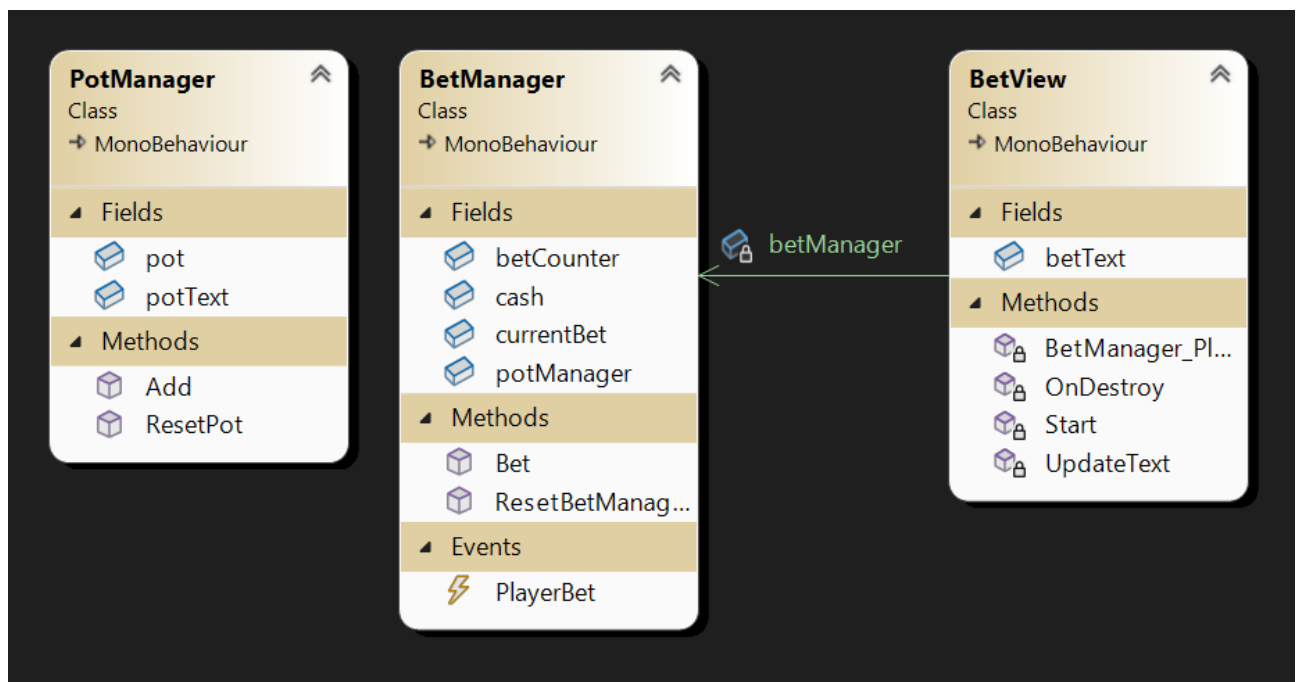


Рисунок 3.4 – Діаграма класів системи управління ставками

3.2.4 Оцінювання сили покерної руки

У розробці інтелектуальних систем для покеру критичною складовою є алгоритм оцінювання сили руки гравця. Він не лише забезпечує логіку визначення переможця, але й слугує основою для прийняття стратегічних рішень ботами, таких як ставка, фолд чи рейз. Для цього в проєкті реалізовано окремий модуль `PokerHandEvaluator`, який забезпечує детерміновану та ймовірнісну оцінку покерних рук.

Основним об'єктом моделі є перерахування `HandRank`, що описує всі можливі типи комбінацій у покері: від старшої карти (`HighCard`) до стріт-флеша (`StraightFlush`). Структура `HandValue` інкапсулює ранжування руки (`Rank`) та список кікерів (`Kickers`) - додаткових карт, що використовуються для розв'язання ситуацій із рівносильними комбінаціями. Реалізований інтерфейс `Comparable<HandValue>` дозволяє порівнювати дві руки між собою за їхньою

силою, що є необхідним як для визначення переможця, так і для побудови стратегій агентів.

Метод `EvaluateHand(List<Card> cards)` виконує повний аналіз руки на основі кількості однакових карт, наявності послідовностей і мастей. Він повертає об'єкт `HandValue`, що описує тип комбінації та ключові карти. Наприклад, якщо рука містить чотири карти однакового рангу, буде повернуто `FourOfAKind` з відповідним кикером.

3.2.5 Модуль відстеження поточного стану гри

Важливим компонентом адаптивного ШІ є модуль відстеження поточного стану гри. Клас `TableState` реалізує централізовану модель даних для представлення поточного стану ігрового столу. Його роль полягає у збиранні, збереженні та поширенні інформації про гравців, їхні ставки, карти та загальний банк. Така абстракція є ключовою для забезпечення когерентної взаємодії ШІ-агентів із ігровим середовищем.

Завдяки використанню патерну спостерігач (observer pattern) через подію `StateChanged`, система дозволяє іншим компонентам реагувати на зміну стану в реальному часі [16]. Це дає можливість інтегрувати механізми адаптивної поведінки, які спрацьовують на зміну візуальних карт, ставок або фолдів інших гравців. Така архітектура є передумовою для реалізації реактивного ШІ, здатного змінювати свою поведінку в залежності від динаміки ігрового процесу.

У цілому, `TableState` є критично важливою інфраструктурною одиницею, що поєднує інформацію про гру у зручному форматі для алгоритмічного аналізу та прийняття рішень.

3.2.6 Архітектура ігрового агента

Клас Player є центральною одиницею у представлення логіки поведінки як людського гравця, так і бота. Його структура демонструє чітке розділення відповідальностей між інтерфейсом користувача, контролем поточної інформації про гру та делегуванням стратегічних рішень агенту штучного інтелекту.

Під час кожного ходу метод StartTurn ініціалізує процес прийняття рішення. Залежно від того, чи є гравець основним (людським), система активує графічний інтерфейс (UI), або - у випадку бота - запускає корутину BotTurn, яка викликає відповідну логіку ШІ. У цьому контексті важливу роль відіграє метод UpdateTableInfo, який збирає актуальні дані про відомі карти: власні, відкриті суперників та ті, що залишились у колоді. Завдяки цим даним формується інформаційний стан агента, необхідний для подальшого аналізу та вибору оптимальної дії.

Особливої уваги заслуговує обробка ШІ-агента у залежності від реалізації. Якщо PlayerAI.IsMLAgent встановлено в true, агент користується методом RequestDecisionFromML, що дозволяє взаємодіяти з навченою моделлю. Якщо ж використовується класичний (евристичний) ШІ, задіюється метод MakeDecision.

Прийняте рішення реалізується методом OnBotDecision, який мапує вибір на конкретну дію (фолд, кол, бет, рейз тощо). Після виконання дії викликається колбек onTurnComplete, який сигналізує про завершення ходу.

Класи BetManager та CardStack інтегруються безпосередньо через методи Bet, Fold, ResetStack, формуючи єдину поведінкову модель гравця. Така композиційна архітектура дозволяє відокремлювати логіку ухвалення рішень від механіки їх реалізації, що є важливою умовою для підтримки багатоагентних систем і можливості розширення.

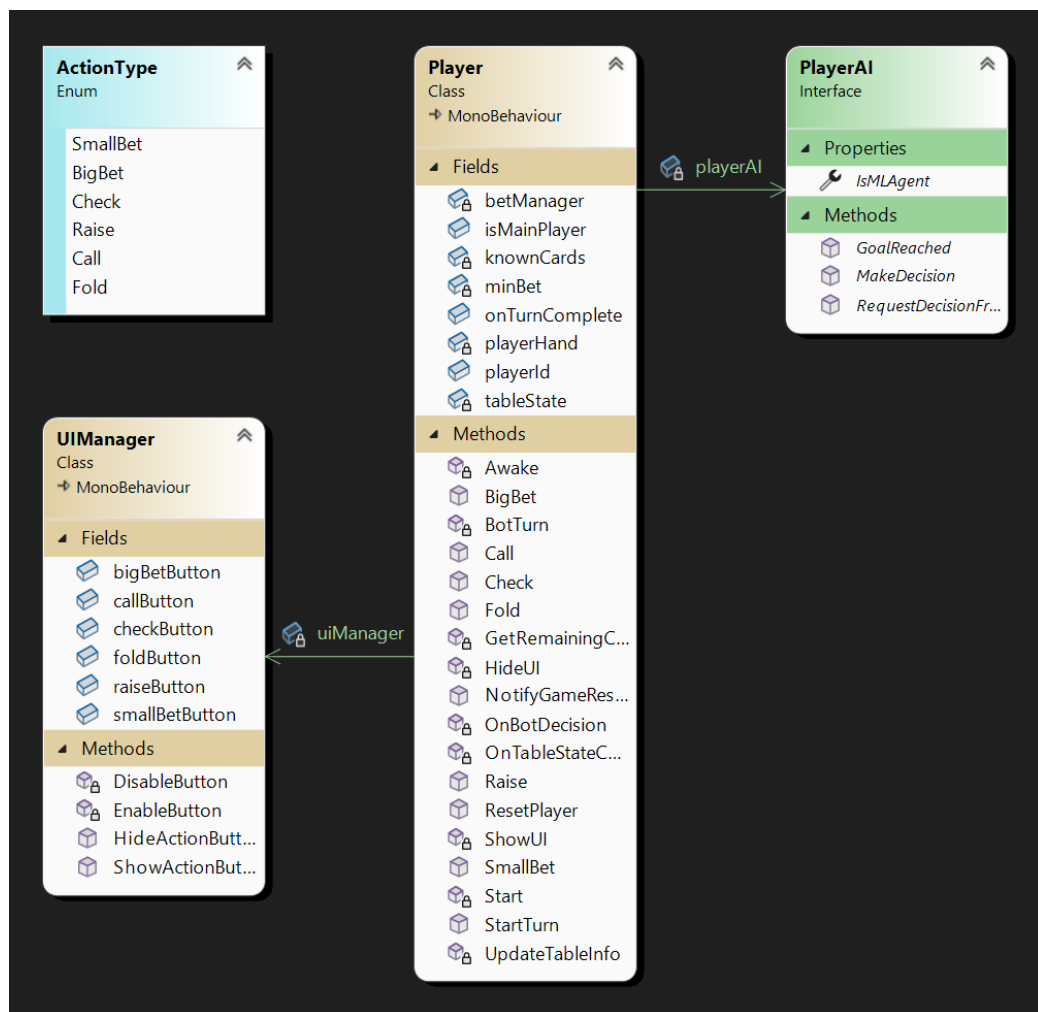


Рисунок 3.5 – Діаграма класів ігрового агента

3.2.7 Реалізація ігрового контролера

Центральним елементом управління ігровим процесом у реалізованій моделі є клас GameController, який виконує роль диспетчера ігрових станів, координатора роздачі карт, а також послідовного керування черговістю дій гравців. Цей клас взаємодіє з іншими компонентами системи, зокрема GameStateManager, CardStack, Player, BetManager.

Клас GameController реалізує повний життєвий цикл гри — від ініціалізації колоди до визначення переможця. На початку гри ініціалізується нова колода методом GenerateDeck(), після чого вона перетасовується та роздаються перші

карти у корутині FirstStage(). Метод ResetGame() забезпечує повторне скидання гри з урахуванням затримки, необхідної для візуального відображення анімацій.

Кожен етап гри (від ThirdStreet до SeventhStreet) супроводжується роздачею додаткових карт гравцям, які ще не скинулися, після чого обирається гравець із найсильнішою (або найслабшою на ThirdStreet) відкритою рукою для початку торгів.

Черговість дій гравців координується методами StartTurn() та NextTurn(). Основною логікою прийняття рішень є визначення можливих дій гравця за допомогою методу getPossibleActions(), що повертає список доступних дій типу ActionType. Алгоритм враховує кілька факторів: наявність поточної ставки (isFacingBet), обмеження на кількість підвищень (raiseCount < 3), а також ситуацію "bring-in" на третьому етапі гри. У випадку, якщо усі ставки гравців рівні (AreAllBetsEqual()), відбувається автоматичний перехід до наступного стану гри за допомогою gameManager.UpdateGameState().

Після завершення SeventhStreet, система переходить до фази Showdown, в якій відкриваються всі карти гравців, що не скинулися, і визначається переможець гри. Метод Showdown() використовує той самий клас PokerHandEvaluator для оцінки сили рук та визначення переможця.

Окремий клас GameStateManager відповідає за зберігання та зміну глобального стану гри, реалізуючи патерн спостерігача через подію OnGameStateChange. Це дозволяє іншим об'єктам (зокрема, GameController) реагувати на зміни стану гри в реальному часі та ініціювати відповідні дії. Така архітектура сприяє гнучкому масштабуванню та полегшує інтеграцію нових механік або фаз гри.

					ІАЛЦ.467200.003 ПЗ	Арк.
						46
Зм.	Арк.	№ докум.	Підпис	Дата		

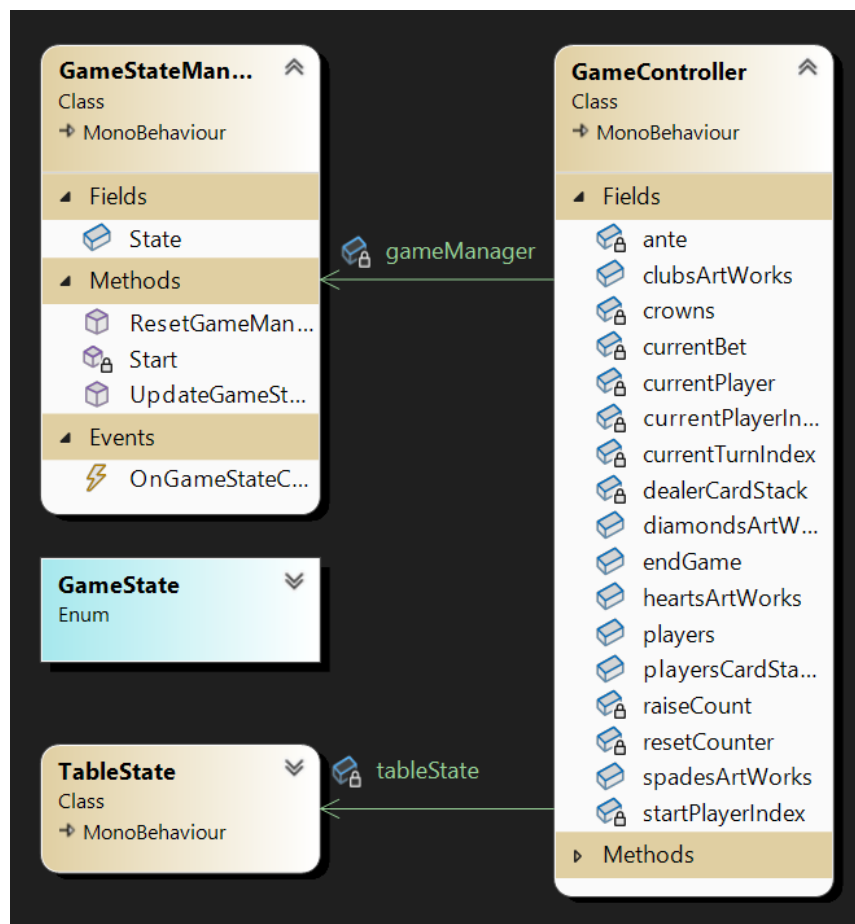


Рисунок 3.6 – Діаграма класів ігрового контролера

3.2.8 Реалізація агента з підкріплювальним навчанням

Клас MLBasedAI успадковується від Agent – базового класу ML-Agents, який реалізує усі необхідні функції для взаємодії з середовищем (реєстрація агентів, управління епізодами, збір спостережень тощо) [17]. Додатково реалізується інтерфейс PlayerAI, що дозволяє використовувати агента як звичайного гравця.

Методи Initialize() та OnEpisodeBegin() використовуються для задання початкових значень та оновлення стану агента при кожному новому епізоді тренування. Якщо активовано trainingMode, то викликається ResetGame(), що повертає гру у початковий стан [17].

Метод `CollectObservations(VectorSensor sensor)` є центральним компонентом збирання даних для навчання. У ньому агент спостерігає:

- Власну руку – до 7 карт, кожна з яких кодується через нормалізовані значення рангу та масті;
- Відкриті карти опонентів – до 4 карт для кожного суперника;
- Поточні ставки – індивідуальні та загальна сума у банку;
- Стан гри – кодується у вигляді нормалізованого значення, що відповідає стадії (від `ThirdStreet` до `SeventhStreet`);
- Кількість суперників, які вже скинули карти – дозволяє агенту враховувати ймовірну силу залишених суперників.

Ці ознаки дозволяють агенту мати контекстно багате уявлення про поточний стан гри, з якого формується вектор спостереження для нейронної мережі політики.

Після збирання спостережень, агент отримує від моделі дію - `ActionType`, яку потім передає у зовнішню гру через делегат `decisionCallback`. Метод `WriteDiscreteActionMask` забезпечує виключення недопустимих дій з простору можливостей агента, тим самим запобігаючи вибору нелегітимних ходів у поточному контексті гри.

Метод `GoalReached` відповідає за підрахунок винагороди агента після завершення епізоду. Якщо агент виграє банк, він отримує позитивну винагороду, пропорційну виграшу; в іншому випадку - штраф того ж розміру. Такий підхід стимулює агента грати на максимізацію прибутку, а не лише на перемогу [3].

Для забезпечення ефективного навчання штучного інтелекту в середовищі стад-покеру була застосована конфігурація, описана у файлі `configuration.yaml`.

Обрано метод `Proximal Policy Optimization (PPO)` - один з найефективніших алгоритмів політик у підкріплювальному навчанні. Його перевага полягає в стабільності та обмеженні надмірних оновлень політик за рахунок використання

					ІАЛЦ.467200.003 ПЗ	Арк.
						48
Зм.	Арк.	№ докум.	Підпис	Дата		

відсічної функції втрат (clipped objective) [18], що особливо важливо в умовах частково спостережуваних середовищ, таких як покер.

Були обрані наступні гіперпараметри навчання:

batch_size: 256

buffer_size: 10240

learning_rate: 0.0003

beta: 0.005

epsilon: 0.2

lambda: 0.95

num_epoch: 3

shared_critic: true

- Розмір мініпакету з досвіду (batch_size), який використовується для кожної оптимізації. Значення 256 дозволяє ефективно балансувати між швидкістю та точністю градієнта.
- Розмір буфера для накопичення досвіду перед оновленням політики (buffer_size). Буфер 10240 забезпечує достатню різноманітність ситуацій.
- Швидкість навчання (learning_rate), встановлена на 0.0003 - це стандартне значення для PPO, яке забезпечує повільне, але стабільне оновлення політики.
- Сила регуляризації ентропії (beta), яка робить політику «більш випадковою». Значення 0.005 гарантує, що агенти належним чином досліджують простір дій під час навчання.
- Коефіцієнт відсічення в PPO (epsilon), який обмежує зміну політики між оновленнями. Значення 0.2 є стандартним.
- Параметр регуляризації (lambda), що використовується під час розрахунку узагальненої оцінки переваг. Значення є стандартним.

					ІАЛЦ.467200.003 ПЗ	Арк.
						49
Зм.	Арк.	№ докум.	Підпис	Дата		

- `shared_critic` використовується для пришвидшення навчання в середовищах з великою кількістю агентів, що працюють паралельно.

Налаштування нейронної мережі:

`hidden_units`: 512

`num_layers`: 3

`memory`:

`sequence_length`: 64

`memory_size`: 256

Модель складається з трьох шарів по 512 нейронів, що забезпечує достатню здатність для обробки складних ігрових ситуацій.

Вбудована рекурентна пам'ять (RNN) дозволяє агенту зберігати контекст дій у часі [20] (наприклад, пам'ятати ставки супротивника на попередніх вулицях), що є критичним для стратегічних ігор з неповною інформацією. Використання LSTM-пам'яті з довжиною послідовності 64 та розміром 256 допомагає в обробці тимчасових залежностей.

Сигнали винагороди:

`gamma`: 0.99

`strength`: 1.0

`network_settings`:

`hidden_units`: 128

`num_layers`: 2

Значення `gamma`: 0.99 встановлює майже повне врахування майбутніх вигід, що актуально в довготривалих іграх (як стад-покер). Нейромережа для оцінки винагород має 2 шари по 128 нейронів

Щоб стимулювати агента досліджувати середовище при рідкісних зовнішніх винагородах, додано внутрішню мотивацію через механізм Curiosity: агент отримує додаткову винагороду за те, що правильно прогнозує, до якого

					ІАЛЦ.467200.003 ПЗ	Арк.
						50
Зм.	Арк.	№ докум.	Підпис	Дата		

стану перейде, використовуючи свою власну модель переходу, що заохочує пошук нових стратегій [19, 21].

3.2.9 Реалізація агента з використанням метода Монте-Карло

Один із підходів до моделювання поведінки штучного гравця в покері полягає у використанні методу Монте-Карло для оцінки ймовірності перемоги на основі поточної руки та доступної інформації про опонентів. Реалізація такого агента представлена в класі `MonteCarloBasedAI`, який реалізує інтерфейс `PlayerAI`.

У серці даного агента лежить метод `MakeDecision`, що відповідає за вибір дії з-поміж доступних (`possibleActions`) у поточній ігровій ситуації. Актор оцінює шанси на перемогу за допомогою симуляцій, викликаючи функцію `SimulateWinProbability` із класу `PokerSimulator`. Цей метод виконує серію випробувань (зазвичай 1000, хоча передбачено можливість зміни кількості симуляцій) для побудови ймовірнісної моделі успішності поточної руки з урахуванням частково відомих карт суперників і залишку колоди.

Функція `SimulateWinProbability` працює за класичним методом Монте-Карло. В кожній симуляції випадковим чином добираються відсутні карти як гравцю, так і опонентам, після чого рука гравця порівнюється з руками суперників за допомогою оцінювача покерних комбінацій. Якщо рука агента переважає всі інші, вважається, що він виграв симуляцію. Після завершення усіх симуляцій визначається частка переможних, що і є оцінкою ймовірності виграшу `winRate`.

На основі `winRate` агент здійснює подальше коригування через систему адаптивних коефіцієнтів:

- `opponentFactor` зменшує впевненість при збільшенні кількості суперників;

- `dynamicThreshold` - динамічний поріг агресивності, що залежить від `potOdds`;
- `riskFactor` - модифікатор ризику, що посилює агресію, коли шанси вищі за `potOdds`;
- `stageFactor` - обережність, що зростає в пізніших раундах торгів;

3.2.10 Структура сцени в Unity

У межах об'єкта `Canvas` розташовано основні UI-компоненти, зокрема:

- `MainPlayerUI` - містить кнопки дій для головного гравця: `FoldButton`, `CallButton`, `RaiseButton`, `CheckButton`, `SmallBetButton`, `BigBetButton` - відповідають за користувацьке управління під час торгів.
- `MainPlayerBet`, `Player1Bet`, `Player2Bet`, `Player3Bet` - елементи для відображення ставок відповідних гравців. Вони включають: `Image`, `Name`, `Crown` - графічні та текстові елементи для виведення інформації про гравця. `TextBg1`, `TextBg2`, `Text` - стилізовані текстові поля для сум ставки або додаткової інформації.
- `Pot` - UI-елемент, що показує загальну суму в банку, структурується аналогічно до блоків ставок гравців.

`GameResultsMenu` та `PauseMenu` - окремі підменю для завершення гри або її призупинення. Кожне меню включає стандартні елементи: `MenuButton`, `QuitButton`, `RestartButton` / `ResumeButton`, `Title`.

`Deck` - об'єкт, що відповідає за управління колодою, містить `SpawnPoint` та `ArtWork`.

`GameController` - головний керуючий об'єкт, реалізує логіку гри (роздача, ставки, черговість тощо).

`MainPlayerPlace`, `Player1Place`, `Player2Place`, `Player3Place` — позиції гравців за столом, кожна з яких має:

- HandSpline - крива для викладення карт у руці;
- OpenedSpline - крива для відкритих карт.

MainPlayer, Player1, Player2, Player3 - логічні представлення гравців, які містять скрипти з інтелектом, станом рук та контролем ставок.

UIManager, Table, GameManager — централізовані керуючі об'єкти.

На скріншоті представлена ієрархія об'єктів сцени в Unity Editor, яка відображає організацію UI-компонентів та ігрових сутностей для реалізації 7-карткового стад-покеру.



Рисунок 3.7 – Структура проекту в Unity Editor

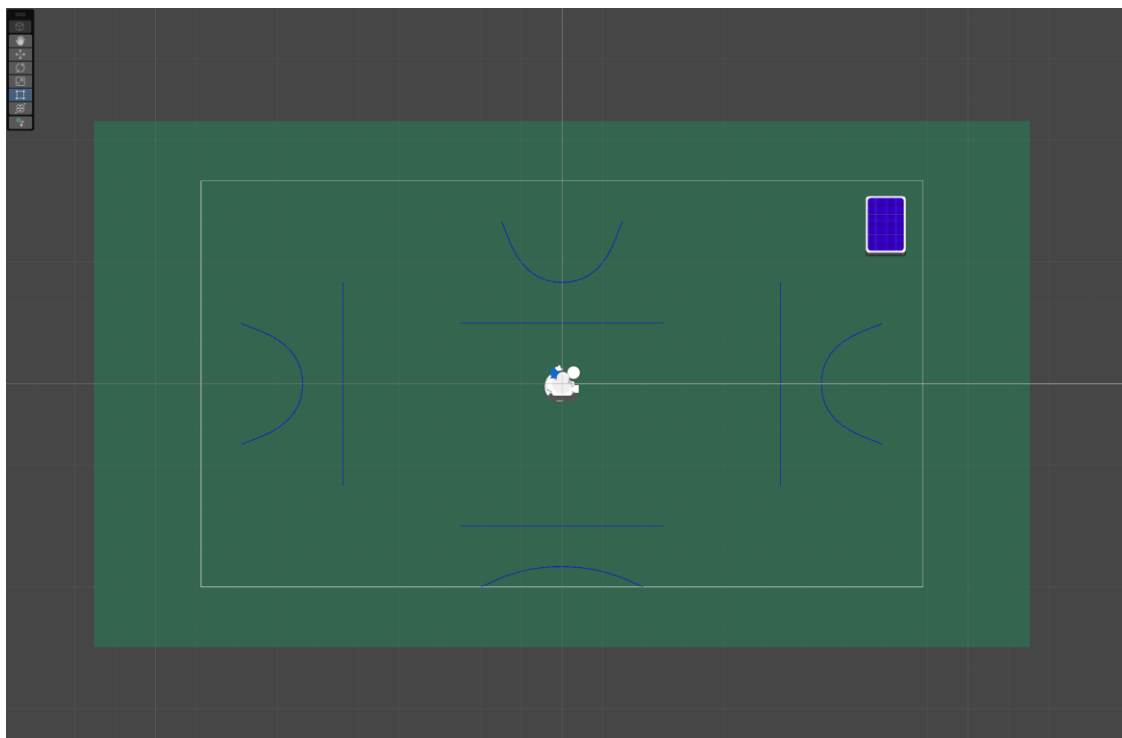


Рисунок 3.8 – скріншот World space сцени в Unity Editor

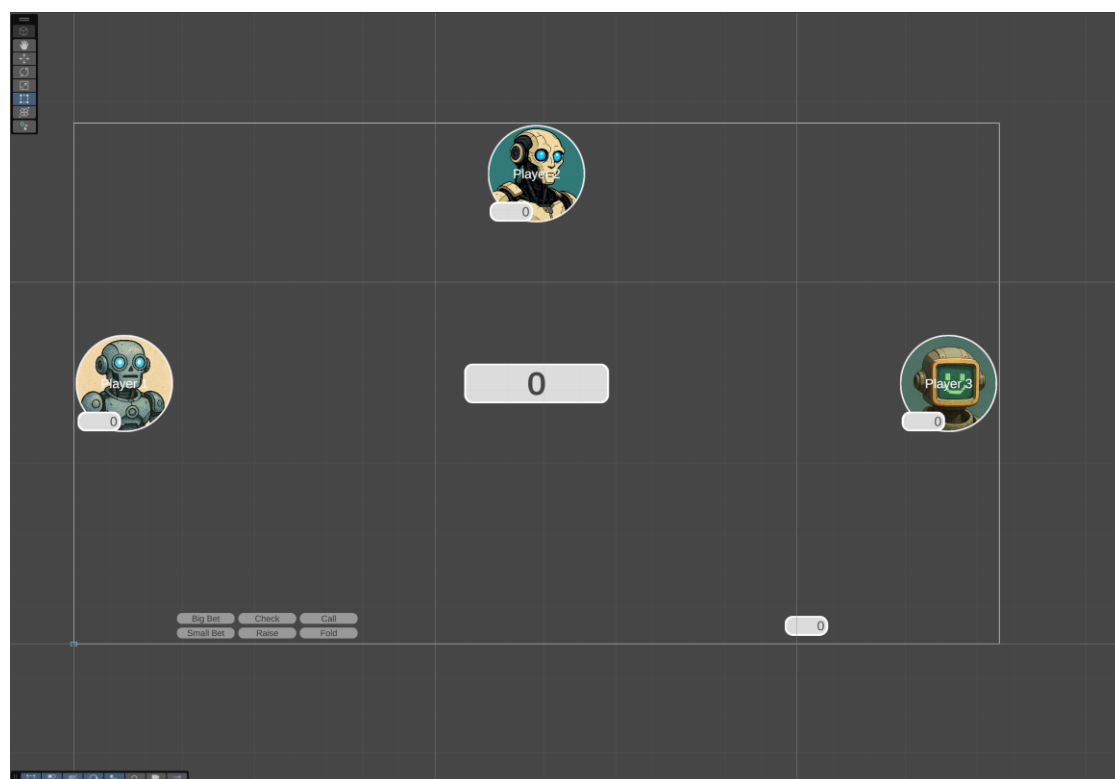


Рисунок 3.9 – скріншот Screen space сцени в Unity Editor

ВИСНОВОК ДО РОЗДІЛУ 3

У ході виконання дипломної роботи було поставлено мету розробити програмну систему, що моделює гру в семикартковий стад покер із залученням штучного інтелекту, здатного до навчання та прийняття рішень на основі поточного стану гри. План передбачав створення повнофункціонального середовища з інтерактивним інтерфейсом, ігровою логікою, модулем ставок, механізмами оцінювання комбінацій та підтримкою інтелектуального агента на основі алгоритмів підкріплювального навчання.

Фактично, усі ключові компоненти системи було успішно реалізовано. Було створено повноцінний функціональний ігровий цикл, що включає логіку роздачі карт, багатораундові ставки, визначення переможця та оновлення балансу гравців. Штучний інтелект реалізовано у двох варіантах: як прості евристичні агенти, так і навчуваний агент, здатний приймати рішення на основі поточного стану гри. Інтерфейс користувача повністю відповідає структурі класичного покерного столу, забезпечуючи зручну візуалізацію ігрового процесу та інтерактивність.

Окрему увагу приділено архітектурній структурі проєкту. Завдяки використанню патернів спостерігача, подійної моделі та чіткої декомпозиції відповідальностей між класами, було досягнуто високої модульності та гнучкості системи, що створює умови для її подальшого масштабування.

Таким чином, реалізований програмний продукт не лише відповідає запланованому функціоналу, але й демонструє розширені можливості у напрямі побудови адаптивних ігрових систем з елементами штучного інтелекту.

					ІАЛЦ.467200.003 ПЗ	Арк.
						55
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Огляд розробленої системи системи

4.1.1 Головне меню

Після запуску застосунку користувач потрапляє до головного меню, зображеного на рисунку 4.1. Інтерфейс цього вікна є максимально простим і зрозумілим. У центрі екрана розміщена назва гри - AI Stud Poker, що одразу інформує користувача про формат гри та її інтелектуальну основу. Під назвою розташована кнопка "Play", яка призначена для початку гри. Нижче знаходиться кнопка "Exit", яка дозволяє завершити роботу застосунку.

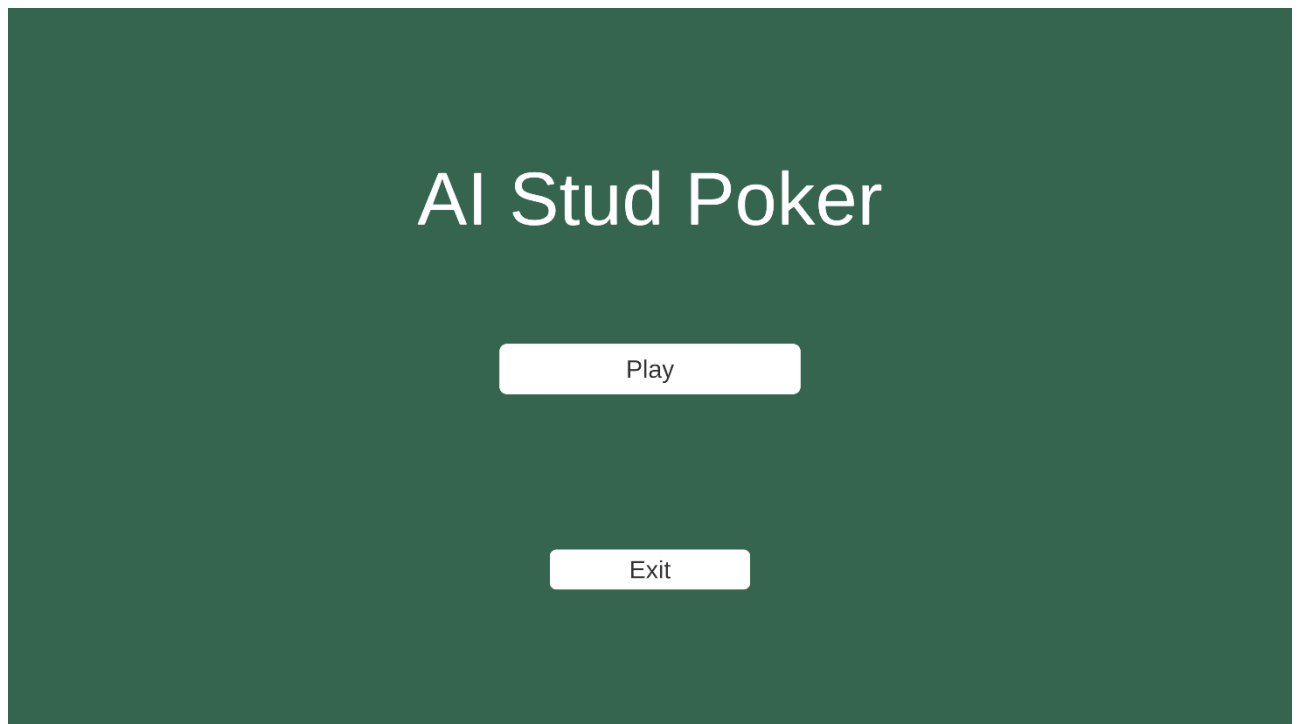


Рисунок 4.1 – скріншот головного меню гри

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		56

4.1.2 Меню вибору складності

Після натискання кнопки "Play" користувач переходить до вікна вибору складності гри, як показано на рисунку 4.2. У верхній частині екрана розміщено заголовок "Оберіть складність", який дає чітке розуміння мети цього інтерфейсу. Доступні три варіанти рівня: "Easy (Monte Carlo)", де суперники базуються на алгоритмі Монте-Карло; "Medium (AI)" і "Hard (AI)", де реалізовано різні рівні складності штучного інтелекту. Це дозволяє гравцеві самостійно обрати бажаний рівень виклику відповідно до власного досвіду. Кнопка "Back" повертає користувача до головного меню.

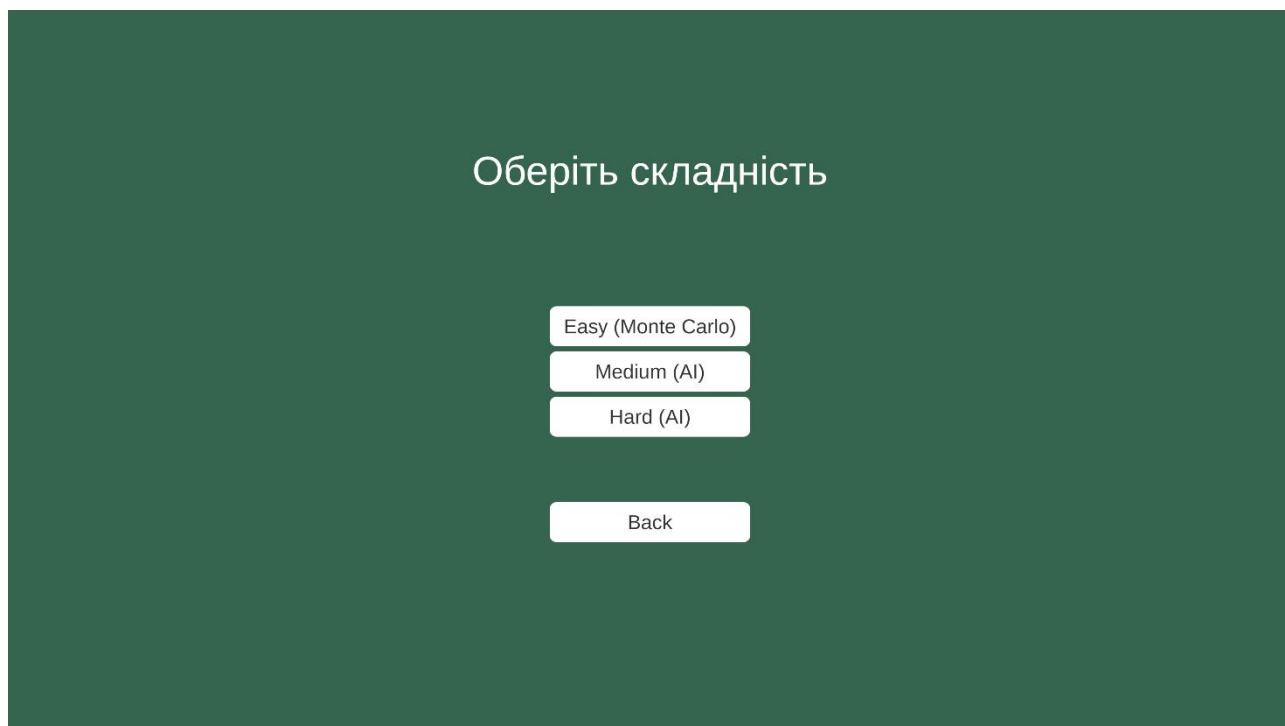


Рисунок 4.2 – скріншот меню вибору складності

4.1.3 Інтерфейс гри

На рисунку 4.3 зображено процес роздачі карт під час початку гри, що супроводжується відповідною анімацією. Гравці, представлені аватарами з підписами, отримують відкриті карти, а також кілька закритих. У центрі екрана відображено поточний банк - 200 фішок, що змінюється залежно від ставок гравців. У нижній частині знаходиться панель керування діями: гравець може зробити ставку, підняти її, зробити чек або пас. Власні карти гравця розташовані в нижній частині екрана.



Рисунок 4.3 – скріншот інтерфейсу гри

На рисунку 4.4 представлено інтерфейс гри, де система динамічно адаптує елементи управління відповідно до поточного стану ігрового сеансу.



Рисунок 4.4 – скріншот адаптації інтерфейсу до стану ігрового сеансу

На рисунку 4.5 видно, як один із гравців вийшов з гри.



Рисунок 4.5 – скріншот гри, де один гравець вийшов з гри

4.1.4 Меню паузи

Меню паузи забезпечує можливість тимчасового призупинення ігрової сесії. У центрі екрану великим шрифтом розміщено заголовок «Гру Призупинено». Під заголовком розташовані три основні кнопки. Перша кнопка з написом «Продовжити» дозволяє повернутися до гри. Друга кнопка - «Головне Меню», яка перенаправляє гравця до стартового екрана гри. Третя кнопка - «Вийти», що, ймовірно, закриває гру або повертає до операційної системи.

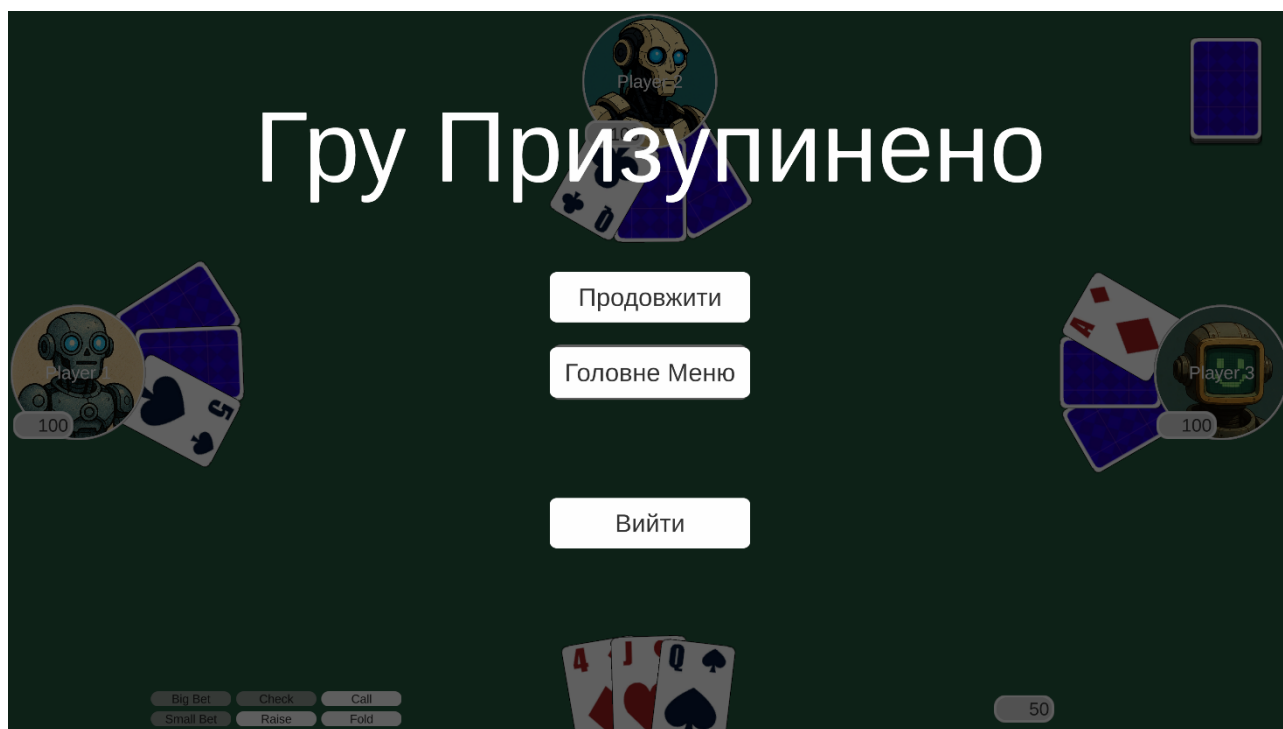


Рисунок 4.6 – скріншот меню паузи

4.1.5 Меню завершення гри

На рисунку 4.6 показано меню завершення гри. У верхній частині екрана великим білим шрифтом відображається повідомлення про результат партії.

					ІАЛЦ.467200.003 ПЗ	Арк.
						60
Зм.	Арк.	№ докум.	Підпис	Дата		

Нижче знаходиться напис, що вказує на переможця та виграшну комбінацію карт.

Центральна частина інтерфейсу містить три основні кнопки. Кнопка "Перезапустити" дозволяє негайно почати нову гру. Кнопка "Головне Меню" повертає користувача на стартовий екран гри. Остання кнопка, "Вийти", розташована нижче й завершує сеанс, закриваючи гру.

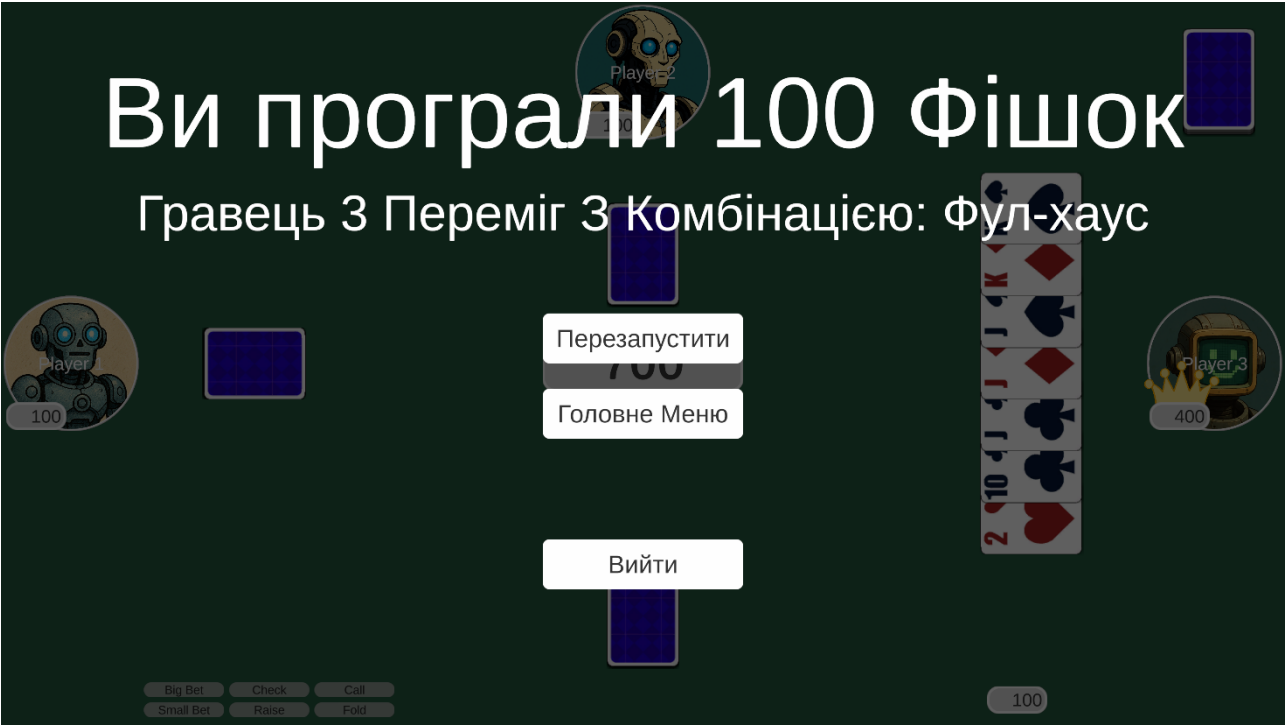


Рисунок 4.6 – скріншот меню завершення гри

4.2 Порівняння з альтернативними підходами

Для оцінки ефективності реалізованого підходу було виконано порівняння з класичною симуляцією на основі методу Монте-Карло.

Таблиця 4.1 – порівняння параметрів Монте-Карло проти навченої моделі

Параметр	Метод Монте-Карло	Підкріплювальне навчання
Середній час ухвалення рішення	~1200 мс	~50 мс
Якість блефів	Відсутні	Часто використовуються
Виграш у симульованих партіях	Приблизно однаковий	
Адаптація до нових гравців	Неможлива	Присутня (через спостереження)
Необхідні ресурси	Високі (в залежності від кількості симуляцій)	Низькі

Хоча метод Монте-Карло забезпечує теоретично точніші оцінки комбінацій, підхід із навчанням агентів демонструє вищу швидкість та адаптивність до суперників, що є критичним для ігор із неповною інформацією, таких як покер.

Головною цінністю розробки є не просто симуляція гри в покер, а демонстрація того, що агент із підкріплювальним навчанням здатен грати на рівні вище випадкової стратегії, використовувати стратегію блефу, навіть не маючи прямої інформації про карти опонента, адаптуватися до зміни стилю гри суперників, чого не може класичний скриптовий AI або Монте-Карло та розпізнавати ситуації, в яких варто уникнути ризику, навіть коли це не очевидно статистично.

ВИСНОВОК ДО РОЗДІЛУ 4

У цьому розділі було здійснено детальний аналіз функціональних можливостей та інтерфейсних рішень розробленої системи гри AI Stud Poker. Представлений інтерфейс вирізняється інтуїтивною структурою, зручністю у використанні та логічною послідовністю екранів: від головного меню до завершення гри. Особливу увагу приділено адаптивності інтерфейсу до різних станів ігрового процесу, що сприяє кращому зануренню користувача у гру.

Крім візуального огляду, проведено порівняльний аналіз ефективності використаного підходу з класичним методом Монте-Карло. Результати свідчать про переваги підкріплювального навчання: значне зниження часу на ухвалення рішень, вища адаптивність до поведінки нових гравців, а також наявність стратегій, зокрема блефу.

Таким чином, розроблена система не лише забезпечує повноцінний ігровий досвід, але й демонструє високий рівень інтелектуальної складової. Це підтверджує доцільність використання сучасних методів штучного інтелекту для створення адаптивних, реалістичних і стратегічно обґрунтованих ігрових агентів.

					ІАЛЦ.467200.003 ПЗ	Арк.
						63
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У ході виконання даної роботи було досягнуто поставленої мети — створено інтелектуальний застосунок для карткової гри на основі сучасних методів штучного інтелекту, зокрема підкріплювального навчання. У межах дослідження проаналізовано теоретичні основи побудови інтелектуальних систем у контексті ігрових застосунків, розглянуто класичні та сучасні підходи до моделювання поведінки агентів, а також проведено класифікацію карткових ігор з точки зору викликів, які вони ставлять перед ШІ. Було визначено, що особливості таких ігор - зокрема неповна інформація, стратегічне прогнозування, ймовірнісний аналіз і симуляція поведінки гравця — створюють складне, але водночас надзвичайно продуктивне середовище для дослідження інтелектуальних агентів.

Практична частина роботи зосереджувалась на розробці гри у жанрі семикарткового стада з реалізацією функціонального ігрового циклу, адаптивного інтерфейсу та модуля штучного інтелекту. Для реалізації застосунку було обрано платформу Unity, яка завдяки своїй гнучкості, підтримці C# та можливості інтеграції з ML-Agents надала всі необхідні інструменти для створення як візуальної, так і логічної складової гри. Архітектура системи побудована з використанням сучасних підходів до організації коду, що забезпечило високу модульність, масштабованість і простоту у налагодженні.

Особливу увагу приділено порівнянню різних підходів до реалізації агентів: від евристичних правил до навчуваних моделей. У результаті підтверджено, що агент, навчений за допомогою підкріплювального навчання, здатен демонструвати складну поведінку, включаючи блеф і адаптацію до змін середовища. Експериментальні результати свідчать про підвищену ефективність таких агентів порівняно з традиційними методами, зокрема у контексті реакції на нові стратегії супротивника.

					ІАЛЦ.467200.003 ПЗ	Арк.
						64
Зм.	Арк.	№ докум.	Підпис	Дата		

Загалом, розроблений застосунок виконує не лише розважальну, але й навчально-дослідницьку функцію, демонструючи потенціал поєднання ігрових технологій і штучного інтелекту. Отримані результати підтверджують перспективність подальших досліджень у цьому напрямі, зокрема щодо розвитку самонавчальних систем у динамічному ігровому середовищі та створення більш реалістичних і варіативних моделей поведінки.

					ІАЛЦ.467200.003 ПЗ	Арк.
						65
Зм.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Devang J. Finite State Machine in Game Development. International Journal of Advanced Research in Science, Communication and Technology (IJARSCT). 2021. Т. 10, вип. 1. С. 384–387. ISSN 2581-9429. URL: https://www.researchgate.net/profile/Devang-Jagdale/publication/355518086_Finite_State_Machine_in_Game_Development/links/6178c8caa767a03c14b9df98/Finite-State-Machine-in-Game-Development.pdf (дата звернення: 22.05.2025).
2. Michele C., Petter Ögren. Behavior Trees in Robotics and AI: An Introduction. Boca Raton : Chapman & Hall/CRC Press, 2018. 208 с. ISBN 9781138593732. URL: <https://doi.org/http://dx.doi.org/10.1201/9780429489105>.
3. Richard S. S., Andrew G. B. Reinforcement Learning: An Introduction / ред. B. Francis. Cambridge, Massachusetts : The MIT Press, 2015. 1–7 с. URL: <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf> (дата звернення: 23.05.2025).
4. Olivier C., Bernhard S., Alexander Z. Semi-Supervised Learning. Cambridge, Massachusetts : The MIT Press, 2006. 1–5 с.
5. Jens K., Jan P. Imitation and Reinforcement Learning. IEEE Robotics & Automation Magazine. 2010. С. 4. URL: <https://doi.org/http://dx.doi.org/10.1109/MRA.2010.936952>.
6. Melanie M. An Introduction to Genetic Algorithms. Cambridge, Massachusetts : The MIT Press, 1999. 1–10 с.
7. Steven W. Game AI: The State of the Industry. Game Developer. 1998. 20 листоп. URL: <https://www.gamedeveloper.com/design/game-ai-the-state-of-the-industry> (дата звернення: 23.05.2025).

8. Damian I. GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI. Game Developer. 2005. 11 безп. URL:
<https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai> (дата звернення: 23.05.2025).
9. Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning / F. Florian та ін. Institute of Electrical and Electronics Engineers. 2021. Т. 6, вип. 3 : IEEE Robotics and Automation Letters. С. 4257 – 4264. ISSN 2377-3766. URL: <https://doi.org/10.1109/LRA.2021.3064284>.
10. Hastings, Erin, "Automatic Graphics And Game Content Generation Through Evolutionary Computation" (2009). Electronic Theses and Dissertations. 3997. <https://stars.library.ucf.edu/etd/3997>
11. David P. The Penguin Book of Card Games. Penguin Books, 2009. 658 с.
12. Dimitris B., Alex P. Machine Learning. Springer. 2022. Т. 111 : World-class interpretable poker. С. 3063–3083. URL: <https://doi.org/10.1007/s10994-022-06179-8> (дата звернення: 08.06.2025).
13. Juho K. Recording and Describing Poker Hands. ArXiv. 2023. С. 8. URL: <https://doi.org/10.48550/arXiv.2312.11753>.
14. Juho K. PokerKit: A Comprehensive Python Library for Fine-Grained Multi-Variant Poker Game Simulations. ArXiv. 2023. С. 8. URL: <https://doi.org/10.48550/arXiv.2308.07327>.
15. Joe H. Unity in Action: Multiplatform Game Development in C# with Unity 5. Manning Publications, 2015. 352 с.
16. Robert N. Game Programming Patterns. Genever Benning, 2014. 354 с.
17. Unity ML-Agents Toolkit. Unity ML-Agents Toolkit Documentation. URL: <https://unity-technologies.github.io/ml-agents/ML-Agents-Toolkit-Documentation/> (дата звернення: 26.05.2025).

18. Proximal Policy Optimization with Mixed Distributed Training / Z. Zhenyu та ін. ArXiv. 2019. С. 5. URL: <https://doi.org/10.48550/arXiv.1907.06479> (дата звернення: 26.05.2025).
19. Exploration by Random Network Distillation / B. Yuri та ін. ArXiv. 2018. С. 17. URL: <https://doi.org/10.48550/arXiv.1810.12894>.
20. Training Agents using Upside-Down Reinforcement Learning / S. Rupesh Kumar та ін. ArXiv. 2021. С. 31. URL: <https://doi.org/10.48550/arXiv.1912.02877> (дата звернення: 26.05.2025).
21. Curiosity-driven Exploration by Self-supervised Prediction / P. Deepak та ін. ArXiv. 2017. С. 12. URL: <https://doi.org/10.48550/arXiv.1705.05363>.

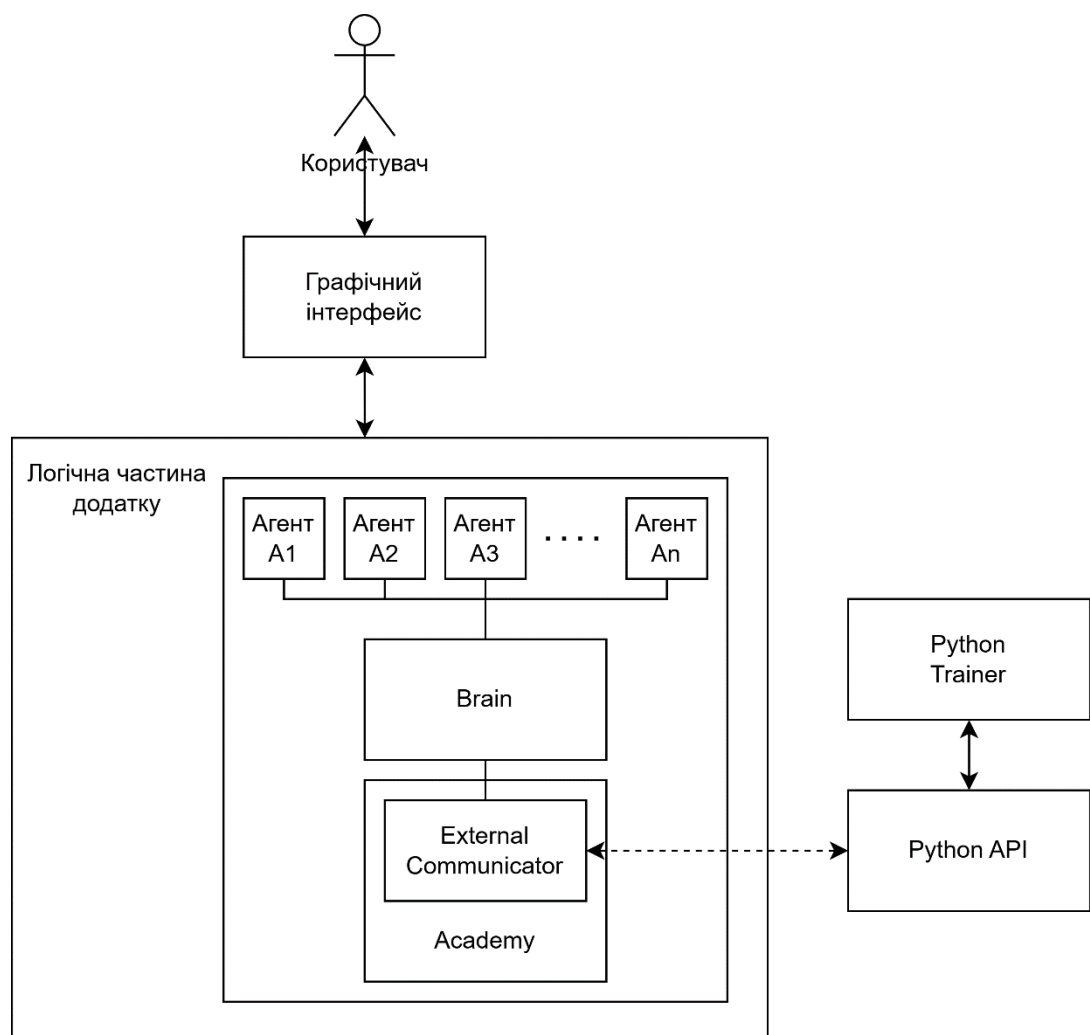
ДОДАТОК А

Застосунок для карткової гри з використанням штучного
інтелекту

Структурна модель системи
ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2025



					ІАЛЦ.467200.004 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив	Чуян А. В.				Застосунок для карткової гри з застосуванням штучного інтелекту Структурна модель системи	Літ.	Аркуш
Перевірив	Гончаренко О.О.					1	1
Реценз.						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Н. Контр.							
Затвердив							

ДОДАТОК Б

Застосунок для карткової гри з використанням штучного
інтелекту

Діаграма класів (функціональна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2025

ДОДАТОК В

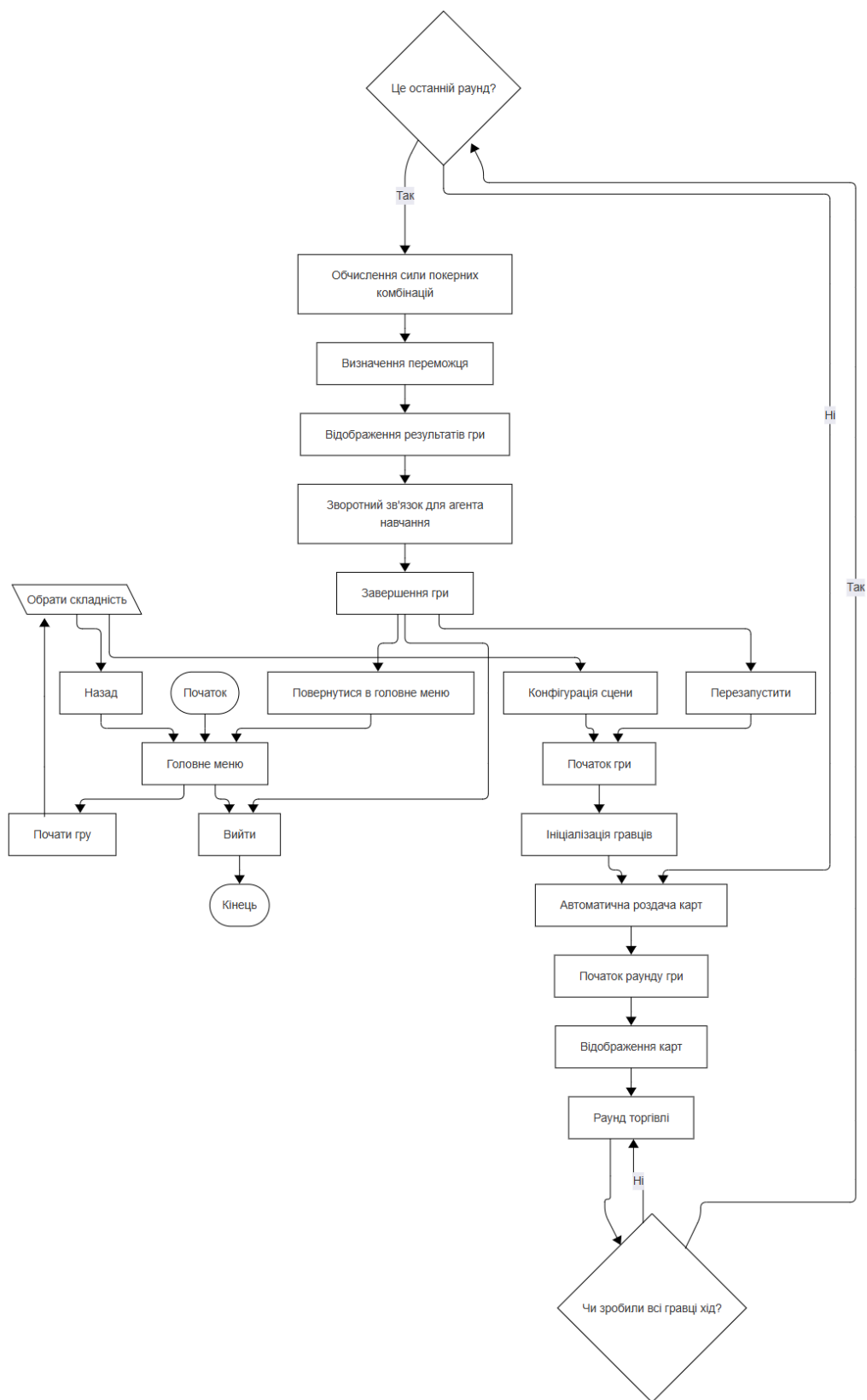
Застосунок для карткової гри з використанням штучного
інтелекту

Алгоритм дії розробленої системи

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ 2025



					ІАЛЦ.467200.006 ДЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Застосунок для карткової гри з застосуванням штучного інтелекту Алгоритм дії розробленої системи			
Розробив	Чуян А. В.							
Перевірив	Гончаренко О.О.							
Реценз.								
Н. Контр.								
Затвердив					Літ. Аркуш Аркушів 1 1 КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11			

ДОДАТОК Г

**Застосунок для карткової гри з використанням штучного
інтелекту**

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 65

Київ 2025

MainMenu.cs:

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    public void PlayPressed(int sceneId)
    {
        SceneManager.LoadSceneAsync(sceneId);
    }

    public void ExitPressed()
    {
        Application.Quit();
    }
}

```

BetManager.cs:

```

using UnityEngine;

public class BetManager : MonoBehaviour
{
    public int cash = 1000;
    public int currentBet = 0;
    public int betCounter = 0;
    public PotManager potManager;

    public event PlayerBetEventHandler PlayerBet;

    public bool Bet(int amount)
    {
        if (amount > cash) return false;
        potManager.Add(amount);
        currentBet += amount;
        cash -= amount;
        betCounter++;
        PlayerBet?.Invoke(this, new PlayerBetEventArgs(currentBet, cash));
        return true;
    }
}

```

					ІАЛЦ.467200.007 Д4										
Зм.	Арк.	№ докум.	Підпис	Дата	Застосунок для карткової гри з застосуванням штучного інтелекту Текст програмного коду					Літ.	Аркуш	Аркушів			
Розробив	Чуян А. В.											1	65		
Перевірив	Гончаренко О.О.									КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11					
Реценз.															
Н. Контр.															
Затвердив															

```

    }

    public void ResetBetManager()
    {
        cash = 1000;
        currentBet = 0;
        betCounter = 0;
        PlayerBet?.Invoke(this, new PlayerBetEventArgs(currentBet, cash));
    }
}

```

BetView.cs:

```

using TMPro;
using UnityEngine;

[RequireComponent(typeof(BetManager))]
public class BetView : MonoBehaviour
{
    public TextMeshProUGUI betText;
    BetManager betManager;

    void Start()
    {
        betManager = GetComponent<BetManager>();
        UpdateText();
        betManager.PlayerBet += BetManager_PlayerBet;
    }

    private void OnDestroy()
    {
        betManager.PlayerBet -= BetManager_PlayerBet;
    }

    private void UpdateText()
    {
        betText.text = betManager.currentBet.ToString();
    }

    private void BetManager_PlayerBet(object sender, PlayerBetEventArgs e)

```

					ІАЛЦ.467200.007 Д4	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        {
            UpdateText();
        }
    }
}

```

Card.cs:

```

using System;
using UnityEngine;

namespace CardGame
{
    [CreateAssetMenu(fileName = "NewCard", menuName = "Card")]
    public class Card : ScriptableObject, IComparable<Card>, IEquatable<Card>
    {
        public enum CardSign
        {
            Club,
            Diamond,
            Heart,
            Spade
        }

        public enum CardValue
        {
            Two = 2,
            Three,
            Four,
            Five,
            Six,
            Seven,
            Eight,
            Nine,
            Ten,
            Jack,
            Queen,
            King,
            Ace
        }

        public CardSign cardSign;
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        public CardValue cardValue;

        public Sprite artWork;

        public bool isFlipped = false;
        public bool isVisibleToOthers = false;

        public int CompareTo(Card other)
        {
            if (other == null) return 1;
            if (!isFlipped) return -1;

            return (int)cardValue * 10 + (int)cardSign - ((int)other.cardValue * 10 +
(int)other.cardSign);
        }

        public bool Equals(Card other)
        {
            return (cardValue == other.cardValue && cardSign == other.cardSign);
        }

        public override string ToString()
        {
            return cardValue.ToString() + " of " + cardSign.ToString();
        }
    }
}

```

CardAddDelegate.cs:

```
public delegate void CardAddedEventHandler(object sender, CardAddedEventArgs e);
```

CardAddedEventArgs.cs:

```
using CardGame;
```

```
public class CardAddedEventArgs
```

```

{
    public CardAddedEventArgs(Card card, bool flip, bool isVisibleToOthers)
    {
        Card = card;
        Flip = flip;
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        isVisibleToOthers = isVisibleToOthers;
    }

    public Card Card { get; private set; }
    public bool Flip { get; private set; }
    public bool isVisibleToOthers { get; private set; }
}

CardDisplay.cs:
using CardGame;
using DG.Tweening;
using System;
using UnityEngine;
using static CardGame.Card;

public class CardDisplay : MonoBehaviour, IComparable<CardDisplay>
{
    public Card card;

    public SpriteRenderer artWorkImage;
    void Start()
    {
        artWorkImage.sprite = card.artWork;
    }

    public void SetCard(Card card)
    {
        this.card = card;
        if (card.isFlipped)
            artWorkImage.sprite = card.artWork;
    }

    public void Flip()
    {
        card.isFlipped = !card.isFlipped;
        card.isVisibleToOthers = card.isFlipped;
        transform.DORotate(new(0, 180f, 0), 0.25f, RotateMode.LocalAxisAdd);
    }

    public void Flip(bool isVisibleToOthers)

```

					ІАЛЦ.467200.007 Д4	Арк.
						5
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    {
        card.isFlipped = !card.isFlipped;
        card.isVisibleToOthers = isVisibleToOthers;
        transform.DORotate(new(0, 180f, 0), 0.25f, RotateMode.LocalAxisAdd);
    }

    public int CompareTo(CardDisplay other)
    {
        if (other == null) return 1;

        return (int)card.cardValue * 10 + (int)card.cardSign - ((int)other.card.cardValue *
10 + (int)other.card.cardSign);
    }
}

```

CardRemoveDelegate.cs:

```
public delegate void CardRemovedEventHandler(object sender, CardRemovedEventArgs e);
```

CardRemovedEventArgs.cs

```
using CardGame;
```

```
public class CardRemovedEventArgs
{
    public CardRemovedEventArgs(int cardIndex)
    {
        this.cardIndex = cardIndex;
    }

    public int cardIndex { get; private set; }
}

```

CardStack.cs

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

public class CardStack : MonoBehaviour
{
    public List<Card> cards = new();
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

```

public bool isFolded = false;
public bool isOpend = false;

public event CardRemovedEventHandler CardRemoved;
public event CardAddedEventHandler CardAdded;
public event HandOpendEventHandler HandOpend;
public event HandFoldedEventHandler HandFolded;

public IEnumerable<Card> GetCards()
{
    foreach (Card card in cards)
        yield return card;
}

public IEnumerable<Card> GetVisibleCards()
{
    foreach (Card card in cards)
        if (card.isVisibleToOthers) yield return card;
}

public void Fold()
{
    isFolded = true;
    HandFolded?.Invoke(this, new HandFoldedEventArgs());
}

public void Open()
{
    isOpend = true;
    HandOpend?.Invoke(this, new HandOpendEventArgs());
}

public void Push(Card card)
{
    if (card != null)
    {
        cards.Add(card);
        CardAdded?.Invoke(this, new CardAddedEventArgs(card, true, true));
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

```

public void Push(Card card, bool flip)
{
    if (card != null)
    {
        cards.Add(card);
        CardAdded?.Invoke(this, new CardAddedEventArgs(card, flip, flip));
    }
}

public void Push(Card card, bool flip, bool isVisibleToOthers)
{
    if (card != null)
    {
        cards.Add(card);
        CardAdded?.Invoke(this, new CardAddedEventArgs(card, flip, isVisibleToOthers));
    }
}

public Card Pop()
{
    if (cards.Count == 0)
        return null;

    Card firstCard = cards[0];
    cards.RemoveAt(0);

    CardRemoved?.Invoke(this, new CardRemovedEventArgs(0));
    return firstCard;
}

public void RemoveFirst()
{
    if (cards.Count == 0)
        return;

    Card firstCard = cards[0];
    cards.RemoveAt(0);

    CardRemoved?.Invoke(this, new CardRemovedEventArgs(0));
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						8
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }

    public bool Remove(Card card)
    {
        return cards.Remove(card);
    }

    public void Sort()
    {
        cards.Sort();
    }

    public int Count { get { return cards.Count; } }

    public void Shuffle()
    {
        for (int i = cards.Count - 1; i > 0; i--)
        {
            int randomIndex = Random.Range(0, i + 1);
            (cards[i], cards[randomIndex]) = (cards[randomIndex], cards[i]);
        }
    }

    public void ResetStack()
    {
        isFolded = false;
        isOpend = false;

        int n = cards.Count;
        for (int i = 0; i < n; i++)
        {
            Card c = Pop();
        }
    }
}

```

CardStackView.cs

```

using CardGame;
using DG.Tweening;
using System.Collections.Generic;

```

					ІАЛЦ.467200.007 Д4	Арк.
						9
Зм.	Арк.	№ докум.	Підпис	Дата		

```

using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.Splines;

[RequireComponent(typeof(CardStack))]
public class CardStackView : MonoBehaviour
{
    [SerializeField] private int maxHandSize;
    [SerializeField] private GameObject cardPrefab;
    [SerializeField] private SplineContainer splineContainer;
    [SerializeField] private SplineContainer openSplineContainer;
    [SerializeField] private Transform spawnPoint;

    CardStack cardStack;
    private List<GameObject> handCards = new();

    private Dictionary<GameObject, Tween> idleTweenDict = new();

    private void Start()
    {
        cardStack = GetComponent<CardStack>();
        UpdateCardPosition();

        cardStack.CardRemoved += CardStack_CardRemoved;
        cardStack.CardAdded += CardStack_CardAdded;
        cardStack.HandOpen += CardStack_HandOpen;
        cardStack.HandFolded += CardStack_HandFolded;
    }

    private void OnDestroy()
    {
        cardStack.CardRemoved -= CardStack_CardRemoved;
        cardStack.CardAdded -= CardStack_CardAdded;
        cardStack.HandOpen -= CardStack_HandOpen;
        cardStack.HandFolded -= CardStack_HandFolded;
    }

    private void CardStack_HandFolded(object sender, HandFoldedEventArgs e)
    {
        if (!cardStack.isOpen)

```

					ІАЛЦ.467200.007 Д4	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        Fold();
    }

    private void CardStack_HandOpend(object sender, HandOpendEventArgs e)
    {
        if (!cardStack.isFolded)
            Open();
    }

    private void CardStack_CardAdded(object sender, CardAddedEventArgs e)
    {
        AddCard(e.Card, e.Flip, e.IsVisibleToOthers);
    }

    private void CardStack_CardRemoved(object sender, CardRemovedEventArgs e)
    {
        GameObject removedCard = handCards[e.cardIndex];
        if (idleTweenDict.ContainsKey(removedCard))
        {
            idleTweenDict[removedCard].Kill();
            idleTweenDict.Remove(removedCard);
        }
        Destroy(removedCard);
        handCards.RemoveAt(e.cardIndex);
        UpdateCardPosition();
    }

    void AddCard(Card card, bool flip, bool isVisibleToOthers)
    {
        DrawCard(card, flip, isVisibleToOthers);
    }

    private void DrawCard(Card drawnCard, bool flip, bool isVisibleToOthers)
    {
        if (handCards.Count >= maxHandSize) return;
        if (drawnCard == null) return;

        GameObject g = Instantiate(cardPrefab, spawnPoint.position, spawnPoint.rotation);
        CardDisplay cardDisplay = g.GetComponent<CardDisplay>();

```

					ІАЛЦ.467200.007 Д4	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        if (cardDisplay != null)
        {
            cardDisplay.SetCard(drawnCard);
            if (flip)
                cardDisplay.Flip(isVisibleToOthers);
        }
        handCards.Add(g);
        SortHandCards(handCards);

        UpdateCardPosition();
    }

    private void SortHandCards(List<GameObject> hc)
    {
        hc.Sort((a, b) =>
        {
            CardDisplay cardA = a.GetComponent<CardDisplay>();
            CardDisplay cardB = b.GetComponent<CardDisplay>();

            if (cardA.card.isVisibleToOthers != cardB.card.isVisibleToOthers)
                return cardA.card.isVisibleToOthers ? 1 : -1;

            return cardA.CompareTo(cardB);
        });
    }

    public void Open()
    {
        splineContainer = openSplineContainer;
        foreach(GameObject g in handCards)
        {
            DOTween.Kill(g.transform);

            if (idleTweenDict.ContainsKey(g))
            {
                idleTweenDict[g].Kill();
                idleTweenDict.Remove(g);
            }
        }
    }

```

					ІАЛЦ.467200.007 Д4	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        CardDisplay cardDisplay = g.GetComponent<CardDisplay>();
        cardDisplay.card.isVisibleToOthers = true;
        if(cardDisplay != null && !cardDisplay.card.isFlipped)
            cardDisplay.Flip();
    }
    SortHandCards(handCards);
    UpdateCardPosition(opendSplineContainer.Spline, false);
}

public void Fold()
{
    foreach (GameObject g in handCards)
    {
        DOTween.Kill(g.transform);

        if (idleTweenDict.ContainsKey(g))
        {
            idleTweenDict[g].Kill();
            idleTweenDict.Remove(g);
        }

        CardDisplay cardDisplay = g.GetComponent<CardDisplay>();
        if (cardDisplay != null && cardDisplay.card.isFlipped)
            cardDisplay.Flip();
    }

    if (handCards.Count == 0) return;
    float cardSpacing = 0f;
    float firstCardPosition = 0.5f;
    UpdateCardPosition(cardSpacing, firstCardPosition, opendSplineContainer.Spline,
false);
}

private void UpdateCardPosition(bool startIdleAnimation = true)
{
    if (handCards.Count == 0) return;
    float cardSpacing = 1f / maxHandSize;
    float firstCardPosition = 0.5f - (handCards.Count - 1) * cardSpacing / 2;
    Spline spline = splineContainer.Spline;
    for (int i = 0; i < handCards.Count; i++)

```

					ІАЛЦ.467200.007 Д4	Арк.
						13
Зм.	Арк.	№ докум.	Підпис	Дата		

```

{
    GameObject card = handCards[i];

    if (idleTweenDict.ContainsKey(card))
    {
        idleTweenDict[card].Kill();
        idleTweenDict.Remove(card);
    }

    card.transform.localScale = Vector3.one;

    float p = firstCardPosition + i * cardSpacing;
    Vector3 splinePosition = spline.EvaluatePosition(p);
    Vector3 forward = spline.EvaluateTangent(p);
    Vector3 up = spline.EvaluateUpVector(p);
    Quaternion rotation = Quaternion.LookRotation(up, Vector3.Cross(up,
forward).normalized);
    card.GetComponent<SortingGroup>().sortingOrder = i;

    if (card.GetComponent<CardDisplay>() != null &&
card.GetComponent<CardDisplay>().card.isFlipped)
    {
        rotation *= Quaternion.Euler(0, 180, 0);
    }

    AnimateCard(card, splinePosition, rotation, startIdleAnimation);
}

}

private void UpdateCardPosition(float cardSpacing, float firstCardPosition, Spline spline,
bool startIdleAnimation = true)
{
    if (handCards.Count == 0) return;
    for (int i = 0; i < handCards.Count; i++)
    {
        GameObject card = handCards[i];

        if (idleTweenDict.ContainsKey(card))
        {
            idleTweenDict[card].Kill();

```

					ІАЛЦ.467200.007 Д4	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        idleTweenDict.Remove(card);
    }

    card.transform.localScale = Vector3.one;

    float p = firstCardPosition + i * cardSpacing;
    Vector3 splinePosition = spline.EvaluatePosition(p);
    Vector3 forward = spline.EvaluateTangent(p);
    Vector3 up = spline.EvaluateUpVector(p);
    Quaternion rotation = Quaternion.LookRotation(up, Vector3.Cross(up,
forward).normalized);
    card.GetComponent<SortingGroup>().sortingOrder = i;

    if (card.GetComponent<CardDisplay>() != null &&
card.GetComponent<CardDisplay>().card.isFlipped)
    {
        rotation *= Quaternion.Euler(0, 180, 0);
    }

    AnimateCard(card, splinePosition, rotation, startIdleAnimation);
}

}

private void UpdateCardPosition(Spline spline, bool startIdleAnimation = true)
{
    if (handCards.Count == 0) return;
    float cardSpacing = 1f / maxHandSize;
    float firstCardPosition = 0.5f - (handCards.Count - 1) * cardSpacing / 2;
    for (int i = 0; i < handCards.Count; i++)
    {
        GameObject card = handCards[i];

        if (idleTweenDict.ContainsKey(card))
        {
            idleTweenDict[card].Kill();
            idleTweenDict.Remove(card);
        }

        card.transform.localScale = Vector3.one;
    }
}

```

```

        float p = firstCardPosition + i * cardSpacing;
        Vector3 splinePosition = spline.EvaluatePosition(p);
        Vector3 forward = spline.EvaluateTangent(p);
        Vector3 up = spline.EvaluateUpVector(p);
        Quaternion rotation = Quaternion.LookRotation(up, Vector3.Cross(up,
forward).normalized);
        card.GetComponent<SortingGroup>().sortingOrder = i;

        if (card.GetComponent<CardDisplay>() != null &&
card.GetComponent<CardDisplay>().card.isFlipped)
        {
            rotation *= Quaternion.Euler(0, 180, 0);
        }

        AnimateCard(card, splinePosition, rotation, startIdleAnimation);
    }
}

private void AnimateCard(GameObject card, Vector3 targetPosition, Quaternion
targetRotation, bool startIdleAnimation, float moveDuration = 0.5f, float rotateDuration =
0.25f)
{
    card.transform.DOMove(targetPosition, moveDuration).OnComplete(() =>
    {
        if (startIdleAnimation)
        {
            StartIdleAnimation(card);
        }
    });
    card.transform.DOLocalRotateQuaternion(targetRotation, rotateDuration);
}

private void StartIdleAnimation(GameObject card)
{
    if (idleTweenDict.ContainsKey(card))
    {
        idleTweenDict[card].Kill();
        idleTweenDict.Remove(card);
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						16
Зм.	Арк.	№ докум.	Підпис	Дата		

```

Vector3 startPos = card.transform.position;
Vector3 startScale = card.transform.localScale;
Vector3 targetScale = startScale * 1.05f;
float targetRotationAngle = 2f;

Sequence idleSequence = DOTween.Sequence();
idleSequence.Append(card.transform.DOMoveY(startPos.y + 0.1f, 1f)
    .SetEase(Ease.InOutSine))
    .Join(card.transform.DOScale(targetScale, 1f)
    .SetEase(Ease.InOutSine))
    .Join(card.transform.DOBlendableLocalRotateBy(new Vector3(0, 0,
targetRotationAngle), 1f)
    .SetEase(Ease.InOutSine));
idleSequence.Append(card.transform.DOMoveY(startPos.y, 1f)
    .SetEase(Ease.InOutSine))
    .Join(card.transform.DOScale(startScale, 1f)
    .SetEase(Ease.InOutSine))
    .Join(card.transform.DOBlendableLocalRotateBy(new Vector3(0, 0, -
targetRotationAngle), 1f)
    .SetEase(Ease.InOutSine));

float randomDelay = Random.Range(0f, 0.5f);
idleSequence.SetDelay(randomDelay)
    .SetLoops(-1, LoopType.Restart);

idleTweenDict[card] = idleSequence;
}
}

```

GameController.cs

```

using CardGame;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Unity.VisualScripting;
using UnityEngine;

public class GameController : MonoBehaviour
{

```

					ІАЛЦ.467200.007 Д4	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

```

[SerializeField] private GameStateManager gameManager;
[SerializeField] private TableState tableState;
[SerializeField] private List<GameObject> crowns;
[SerializeField] private CardStack dealerCardStack;

int ante = 50;

public Action<int, PokerHandEvaluator.HandRank, bool, int> endGame;

public List<GameObject> players;

public List<CardStack> playersCardStacks;

public Sprite[] clubsArtWorks;
public Sprite[] diamondsArtWorks;
public Sprite[] heartsArtWorks;
public Sprite[] spadesArtWorks;

private GameObject currentPlayer;
private int currentPlayerIndex;
private int startPlayerIndex;
private int currentTurnIndex;
private int raiseCount;
private int currentBet;

private int resetCounter = 0;

public void ResetGame()
{
    if (resetCounter == 0)
    {
        dealerCardStack.ResetStack();
        tableState.ResetState();
        GenerateDeck();
        dealerCardStack.Shuffle();
        gameManager.ResetGameManager();
    }
    resetCounter += 1;
    if (resetCounter >= 4) resetCounter = 0;
}

```

```

private void StartTurn()
{
    if (players.Count == 0) return;
    currentTurnIndex = 0;
    NextTurn();
}

private void NextTurn()
{
    if (currentTurnIndex >= players.Count && AreAllBetsEqual())
    {
        gameManager.UpdateGameState(gameManager.State + 1);
        return;
    }

    List<ActionType> possibleActions = getPossibleActions();
    if (!playersCardStacks[currentPlayerIndex].isFolded)
    {
        currentPlayer.GetComponent<Player>().StartTurn(() =>
        {
            if (currentPlayer.GetComponent<BetManager>().currentBet != currentBet &&
!currentPlayer.GetComponent<CardStack>().isFolded)
            {
                currentBet = currentPlayer.GetComponent<BetManager>().currentBet;
                raiseCount++;
            }
            currentTurnIndex++;
            currentPlayerIndex = (startPlayerIndex + currentTurnIndex) % players.Count;
            tableState.currentPlayerIndex = currentPlayerIndex;
            currentPlayer = players[currentPlayerIndex];
            NextTurn();
        }, possibleActions, currentBet);
    }
    else
    {
        currentTurnIndex++;
        currentPlayerIndex = (startPlayerIndex + currentTurnIndex) % players.Count;
        tableState.currentPlayerIndex = currentPlayerIndex;
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						19
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        currentPlayer = players[currentPlayerIndex];
        NextTurn();
    }

}

private bool AreAllBetsEqual()
{
    int bet = -1;
    foreach (var player in players)
    {
        var cardStack = player.GetComponent<CardStack>();
        if (cardStack.isFolded) continue;

        var currentBet = player.GetComponent<BetManager>().currentBet;
        if (bet == -1)
        {
            bet = currentBet;
        }
        else if (bet != currentBet)
        {
            return false;
        }
    }
    return true;
}

private List<ActionType> getPossibleActions()
{
    List<ActionType> actions = new List<ActionType>();

    bool isBringIn = (gameManager.State == GameState.ThirdStreet && currentTurnIndex ==
0);

    bool isFacingBet = (currentBet !=
currentPlayer.GetComponent<BetManager>().currentBet);
    bool canRaise = raiseCount < 3;

    int foldedCounter = 0;
    foreach (CardStack cardStack in playersCardStacks)

```

					ІАЛЦ.467200.007 Д4	Арк.
						20
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    {
        if (cardStack.isFolded) foldedCounter++;
    }

    bool isOnePlayerLeft = (foldedCounter == players.Count - 1);

    if (isBringIn)
    {
        actions.Add(ActionType.SmallBet);
        return actions;
    }

    if (!isOnePlayerLeft)
        actions.Add(ActionType.Fold);

    if (isFacingBet)
    {
        actions.Add(ActionType.Call);
        if (canRaise)
            actions.Add(ActionType.Raise);
        return actions;
    }
    else
    {
        if ((int) gameManager.State < 3 && canRaise)
        {
            actions.Add(ActionType.SmallBet);
        }
        else if (canRaise)
        {
            actions.Add(ActionType.BigBet);
        }
        actions.Add(ActionType.Check);
    }
    return actions;
}

private void Open()
{
    foreach (CardStack cardStack in playersCardStacks)

```

					ІАЛЦ.467200.007 Д4	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        cardStack.Open();
    }

    public void Fold()
    {
        playersCardStacks[0].Fold();
    }

    private IEnumerator FirstStage()
    {
        foreach (GameObject player in players)
            if (!player.GetComponent<BetManager>().Bet(ante))
                NotEnoughCashHandle();

        yield return new WaitForSeconds(0.15f);
        playersCardStacks[0].Push(dealerCardStack.Pop(), true, false);
        yield return new WaitForSeconds(0.15f);
        playersCardStacks[0].Push(dealerCardStack.Pop(), true, false);
        yield return new WaitForSeconds(0.15f);
        playersCardStacks[0].Push(dealerCardStack.Pop());

        for (int i = 1; i < playersCardStacks.Count; i++)
        {
            yield return new WaitForSeconds(0.15f);
            playersCardStacks[i].Push(dealerCardStack.Pop(), false);
            yield return new WaitForSeconds(0.15f);
            playersCardStacks[i].Push(dealerCardStack.Pop(), false);
            yield return new WaitForSeconds(0.15f);
            playersCardStacks[i].Push(dealerCardStack.Pop());

        }

        gameManager.UpdateGameState(GameState.ThirdStreet);
    }

    private void NotEnoughCashHandle()
    {
        throw new NotImplementedException();
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

```

private void Awake()
{
    gameManager.OnGameStateChange += GameManager_OnGameStateChange;
    foreach(GameObject player in players)
    {
        playersCardStacks.Add(player.GetComponent<CardStack>());
    }
}

private void OnDestroy()
{
    gameManager.OnGameStateChange -= GameManager_OnGameStateChange;
}

private void GameManager_OnGameStateChange(GameState state)
{
    if (state == GameState.StartGame)
        StartCoroutine(FirstStage());
    if (state == GameState.ThirdStreet)
        ThirdStreet();
    if (state == GameState.FourthStreet)
        StartCoroutine(FourthStreet());
    if (state == GameState.FifthStreet)
        StartCoroutine(FifthStreet());
    if (state == GameState.SixthStreet)
        StartCoroutine(SixthStreet());
    if (state == GameState.SeventhStreet)
        StartCoroutine(SeventhStreet());
    if (state == GameState.Showdown)
    {
        Showdown();
    }
}

private void ThirdStreet()
{
    CardStack weakestHand =
PokerHandEvaluator.GetWeakestHand(GetUnfoldedCardStacks(playersCardStacks).ToList());
    currentPlayerIndex = playersCardStacks.IndexOf(weakestHand);
    tableState.currentPlayerIndex = currentPlayerIndex;
}

```

```

        startPlayerIndex = currentPlayerIndex;
        currentPlayer = players[currentPlayerIndex];
        raiseCount = 0;
        currentBet = 50;
        StartTurn();
    }

```

```

private IEnumerator FourthStreet()
{
    yield return new WaitForSeconds(0.05f);
    foreach (CardStack cardStack in playersCardStacks)
    {
        if (!cardStack.isFolded)
        {
            yield return new WaitForSeconds(0.15f);
            cardStack.Push(dealerCardStack.Pop());
        }
    }
}

```

```

        CardStack strongestHand =
PokerHandEvaluator.GetStrongestHand(GetUnfoldedCardStacks(playersCardStacks).ToList());
        currentPlayerIndex = playersCardStacks.IndexOf(strongestHand);
        tableState.currentPlayerIndex = currentPlayerIndex;
        startPlayerIndex = currentPlayerIndex;
        currentPlayer = players[currentPlayerIndex];
        raiseCount = 0;
        StartTurn();
    }

```

```

private IEnumerator FifthStreet()
{
    yield return new WaitForSeconds(0.05f);
    foreach (CardStack cardStack in playersCardStacks)
    {
        if (!cardStack.isFolded)
        {
            yield return new WaitForSeconds(0.15f);
            cardStack.Push(dealerCardStack.Pop());
        }
    }
}

```

```

        CardStack                                strongestHand                                =
PokerHandEvaluator.GetStrongestHand(GetUnfoldedCardStacks(playersCardStacks).ToList());
        currentPlayerIndex = playersCardStacks.IndexOf(strongestHand);
        tableState.currentPlayerIndex = currentPlayerIndex;
        startPlayerIndex = currentPlayerIndex;
        currentPlayer = players[currentPlayerIndex];
        raiseCount = 0;
        StartTurn();
    }

    private IEnumerator SixthStreet()
    {
        yield return new WaitForSeconds(0.05f);
        foreach (CardStack cardStack in playersCardStacks)
        {
            if (!cardStack.isFolded)
            {
                yield return new WaitForSeconds(0.15f);
                cardStack.Push(dealerCardStack.Pop());
            }
        }
    }

        CardStack                                strongestHand                                =
PokerHandEvaluator.GetStrongestHand(GetUnfoldedCardStacks(playersCardStacks).ToList());
        currentPlayerIndex = playersCardStacks.IndexOf(strongestHand);
        tableState.currentPlayerIndex = currentPlayerIndex;
        startPlayerIndex = currentPlayerIndex;
        currentPlayer = players[currentPlayerIndex];
        raiseCount = 0;
        StartTurn();
    }

    private IEnumerator SeventhStreet()
    {
        yield return new WaitForSeconds(0.05f);
        if (!playersCardStacks[0].isFolded)
        {
            yield return new WaitForSeconds(0.15f);
            playersCardStacks[0].Push(dealerCardStack.Pop(), true, false);
        }
    }

```

					ІАЛЦ.467200.007 Д4	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }

    for (int i = 1; i < playersCardStacks.Count; i++)
    {
        if (!playersCardStacks[i].isFolded)
        {
            yield return new WaitForSeconds(0.15f);
            playersCardStacks[i].Push(dealerCardStack.Pop(), false);
        }
    }

    CardStack strongestHand =
PokerHandEvaluator.GetStrongestHand(GetUnfoldedCardStacks(playersCardStacks).ToList());
    currentPlayerIndex = playersCardStacks.IndexOf(strongestHand);
    tableState.currentPlayerIndex = currentPlayerIndex;
    startPlayerIndex = currentPlayerIndex;
    currentPlayer = players[currentPlayerIndex];
    raiseCount = 0;
    StartTurn();
}

private void Showdown()
{
    foreach (CardStack cardStack in playersCardStacks)
    {
        if (!cardStack.isFolded)
        {
            cardStack.Open();
        }
    }

    CardStack strongestHand =
PokerHandEvaluator.GetStrongestHand(GetUnfoldedCardStacks(playersCardStacks).ToList());
    int winnerIndex = playersCardStacks.IndexOf(strongestHand);
    crowns[winnerIndex].SetActive(true);

    for (int i = 0; i < players.Count; i++)
    {
        Player player = players[i].GetComponent<Player>();
        if (player != null)
        {

```

```

        if (i == winnerIndex)
            player.NotifyGameResults(true);
        else
            player.NotifyGameResults(false);
    }
}

PokerHandEvaluator.HandRank                                rank                                =
PokerHandEvaluator.EvaluateHand(strongestHand.GetVisibleCards().ToList()).Rank;
bool isMainPlayer = players[winnerIndex].GetComponent<Player>().isMainPlayer;
if (isMainPlayer)
    endGame?.Invoke(winnerIndex, rank, isMainPlayer, tableState.GetPot());
else
    endGame?.Invoke(winnerIndex, rank, isMainPlayer, tableState.GetMainPlayerBet());
}

private void Start()
{
    GenerateDeck();
    dealerCardStack.Shuffle();
}

public int getNumberOfFoldedPlayers()
{
    int folded_num = 0;
    foreach (CardStack cardStack in playersCardStacks)
        if (cardStack.isFolded) folded_num++;
    return folded_num;
}

void GenerateDeck()
{
    foreach (Card.CardSign sign in System.Enum.GetValues(typeof(Card.CardSign)))
    {
        foreach (Card.CardValue value in System.Enum.GetValues(typeof(Card.CardValue)))
        {
            Card newCard = ScriptableObject.CreateInstance<Card>();
            newCard.cardSign = sign;
            newCard.cardValue = value;
            newCard.artWork = AssignArtWork(sign, (int)value - 2);
        }
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						27
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        dealerCardStack.Push(newCard);
    }
}

private IEnumerable<CardStack> GetUnfoldedCardStacks(List<CardStack> cardStacks)
{
    foreach (CardStack cardStack in cardStacks)
    {
        if (!cardStack.isFolded)
        {
            yield return cardStack;
        }
    }
}

Sprite AssignArtWork(Card.CardSign sign, int index)
{
    switch (sign)
    {
        case Card.CardSign.Club:
            return clubsArtWorks[index];
        case Card.CardSign.Diamond:
            return diamondsArtWorks[index];
        case Card.CardSign.Heart:
            return heartsArtWorks[index];
        case Card.CardSign.Spade:
            return spadesArtWorks[index];
        default:
            return null;
    }
}
}

```

GameResultsMenu.cs

```

using System;
using CardGame;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;

```

					ІАЛЦ.467200.007 Д4	Арк.
						28
Зм.	Арк.	№ докум.	Підпис	Дата		

```

public class GameResultsMenu : MonoBehaviour
{
    [SerializeField] private GameController debugDealer;
    [SerializeField] private TextMeshProUGUI resultText;
    [SerializeField] private TextMeshProUGUI detailsText;
    [SerializeField] private GameObject resultsMenu;

    void Start()
    {
        debugDealer.endGame += OnGameEnd;
    }

    private void OnDestroy()
    {
        debugDealer.endGame -= OnGameEnd;
    }

    private void OnGameEnd(int playerId, PokerHandEvaluator.HandRank rank, bool isMainPlayer,
int amount)
    {
        if (isMainPlayer)
        {
            resultText.text = "Ви виграли " + amount.ToString() + " Фішок";
            detailsText.text = "Комбінація: " + HandTankToUkr(rank);
        }
        else
        {
            resultText.text = "Ви програли " + amount.ToString() + " Фішок";
            detailsText.text = "Гравець " + playerId.ToString() + " Переміг з Комбінацією: "
+ HandTankToUkr(rank);
        }
        resultsMenu.SetActive(true);
    }

    private String HandTankToUkr(PokerHandEvaluator.HandRank rank)
    {
        switch (rank)
        {
            case PokerHandEvaluator.HandRank.HighCard:

```

					ІАЛЦ.467200.007 Д4	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return "Старша Карта";
    case PokerHandEvaluator.HandRank.OnePair:
        return "Пара";
    case PokerHandEvaluator.HandRank.TwoPair:
        return "Дві Пари";
    case PokerHandEvaluator.HandRank.ThreeOfAKind:
        return "Трійка";
    case PokerHandEvaluator.HandRank.Straight:
        return "Стріт";
    case PokerHandEvaluator.HandRank.Flush:
        return "Флеш";
    case PokerHandEvaluator.HandRank.FullHouse:
        return "Фул-хаус";
    case PokerHandEvaluator.HandRank.FourOfAKind:
        return "Карє";
    case PokerHandEvaluator.HandRank.StraightFlush:
        return "Стріт-флеш";
    default:
        return "Невідома комбінація";
    }
}

public void RestartPressed(int sceneId)
{
    SceneManager.LoadScene(sceneId);
}

public void ReturnToMainMenuPressed()
{
    SceneManager.LoadSceneAsync(0);
}

public void QuitPressed()
{
    Application.Quit();
}
}

```

GameStateManager.cs

```

using System;
using UnityEngine;

```

					ІАЛЦ.467200.007 Д4	Арк.
						30
Зм.	Арк.	№ докум.	Підпис	Дата		

```

public class GameStateManager : MonoBehaviour
{
    public GameState State;
    public event Action<GameState> OnGameStateChange;

    public void ResetGameManager()
    {
        UpdateGameState(GameState.StartGame);
    }

    private void Start()
    {
        UpdateGameState(GameState.StartGame);
    }

    public void UpdateGameState(GameState newState)
    {
        State = newState;

        switch (newState)
        {
            case GameState.StartGame:
                break;
            case GameState.ThirdStreet:
                break;
            case GameState.FourthStreet:
                break;
            case GameState.FifthStreet:
                break;
            case GameState.SixthStreet:
                break;
            case GameState.SeventhStreet:
                break;
            case GameState.Showdown:
                break;
            case GameState.Idle:
                break;
            case GameState.Lose:
                break;
        }
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						31
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        default:
            throw new ArgumentOutOfRangeException(nameof(newState), newState, null);
    }

    OnGameStateChange?.Invoke(newState);
}
}

public enum GameState
{
    StartGame,
    ThirdStreet,    // 1
    FourthStreet,   // 2
    FifthStreet,    // 3
    SixthStreet,    // 4
    SeventhStreet,  // 5
    Showdown,
    Idle,
    Lose
}

```

HandFoldDelegate.cs

```
public delegate void HandFoldedEventHandler(object sender, HandFoldedEventArgs e);
```

HandFoldedEventArgs.cs

```
public class HandFoldedEventArgs
{
}

```

HandOpenDelegate.cs

```
public delegate void HandOpendEventHandler(object sender, HandOpendEventArgs e);
```

HandOpendEventArgs.cs

```
public class HandOpendEventArgs
{
}

```

MLBasedAI.cs

```
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;

```

					ІАЛЦ.467200.007 Д4	Арк.
						32
Зм.	Арк.	№ докум.	Підпис	Дата		

```

using Unity.MLAgents.Actuators;
using System.Data;
using Unity.VisualScripting;
using System;
using System.Collections.Generic;
using CardGame;
using System.Linq;

public class MLBasedAI : Agent, PlayerAI
{
    public bool IsMLAgent { get; } = true;

    [SerializeField] private GameController debugDealer;
    [SerializeField] private GameStateManager gameManager;
    [SerializeField] private bool trainingMode = false;

    private List<ActionType> possibleActions;
    private List<Card> playerHand;
    private List<List<Card>> allVisibleOpponentCards;
    private int pot;
    private List<int> bets;
    private int playerBet;
    private Action<ActionType> decisionCallback;

    private int _currentEpisode = 0;
    private float _cumulativeReward = 0f;

    public override void Initialize()
    {
        _currentEpisode = 0;
        _cumulativeReward = 0f;
    }

    public override void OnEpisodeBegin()
    {
        _currentEpisode++;
        _cumulativeReward = 0f;
        if (trainingMode)
        {
            debugDealer.ResetGame();
        }
    }
}

```

```

    }
}

public override void CollectObservations(VectorSensor sensor)
{
    if (sensor != null)
    {
        for (int i = 0; i < 7; i++)
        {
            if (i >= playerHand.Count)
            {
                sensor.AddObservation(-1.0f);
                sensor.AddObservation(-1.0f);
            }
            else
            {
                AddCardObservations(playerHand[i], sensor);
            }
        }

        foreach (List<Card> cards in allVisibleOpponentCards)
        {
            for (int i = 0; i < 4; i++)
            {
                if (i >= cards.Count)
                {
                    sensor.AddObservation(-1.0f);
                    sensor.AddObservation(-1.0f);
                }
                else
                {
                    AddCardObservations(cards[i], sensor);
                }
            }
        }

        sensor.AddObservation(NormalizeBet(playerBet));

        foreach (int bet in bets)
            sensor.AddObservation(NormalizeBet(bet));
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						34
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        sensor.AddObservation(NormalizeBet(pot));

        sensor.AddObservation((float)playerBet / (pot + 1));

        float gameStateNormalized = ((float)gameManager.State - 1f) / 4f * 2f - 1f;
        sensor.AddObservation(gameStateNormalized);

        float handValue = PokerHandEvaluator.EvaluateHandStrengthNormalized(playerHand);
        sensor.AddObservation(handValue);

        sensor.AddObservation((float)debugDealer.getNumberOfFoldedPlayers() / 4f * 2f -
1f);
    }
}

public override void OnActionReceived(ActionBuffers actions)
{
    int actionIndex = actions.DiscreteActions[0];
    ActionType selected = (ActionType)actionIndex;

    float handStrength = PokerHandEvaluator.EvaluateHandStrengthNormalized(playerHand);
    bool lateGame = gameManager.State >= GameState.FifthStreet;

    decisionCallback?.Invoke(selected);
}

public ActionType MakeDecision(List<ActionType> possibleActions, List<Card> playerHand,
List<List<Card>> visibleOpponentCards, List<Card> deck, float potOdds)
{
    throw new NotSupportedException();
}

public void RequestDecisionFromML(List<ActionType> possibleActions, List<Card>
playerHand, List<List<Card>> allVisibleOpponentCards, int pot, List<int> bets, int playerBet,
Action<ActionType> callback)
{
    this.possibleActions = possibleActions;
    this.playerHand = playerHand;
    this.allVisibleOpponentCards = allVisibleOpponentCards;
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						35
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        this.pot = pot;
        this.bets = bets;
        this.decisionCallback = callback;

        RequestDecision();
    }

    public override void WriteDiscreteActionMask(IDiscreteActionMask actionMask)
    {
        int totalActions = Enum.GetValues(typeof(ActionType)).Length;

        for (int i = 0; i < totalActions; i++)
        {
            ActionType action = (ActionType)i;
            if (!possibleActions.Contains(action))
            {
                actionMask.SetActionEnabled(0, i, false);
            }
        }
    }

    public void GoalReached(bool won, int amount, int id)
    {
        if (won)
        {
            AddReward(amount);
        }
        else
        {
            AddReward(-amount);
        }

        _cumulativeReward = GetCumulativeReward();
    }

    private float NormalizeBet(int bet)
    {
        int smallBet = 50;
        int bigBet = 100;
        int playersNumber = 4;
    }

```

					ІАЛЦ.467200.007 Д4	Арк.
						36
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        int ante = 50;

        int maxPot = playersNumber * ante + playersNumber * (smallBet * 8 + bigBet * 12);

        return (float)bet / (float)maxPot * 2.0f - 1.0f;
    }

    private void AddCardObservations(Card card, VectorSensor sensor)
    {
        sensor.AddObservation(((float)card.cardValue - 1f) / 13f * 2f - 1f);
        sensor.AddObservation(((float)card.cardSign + 1) / 5f * 2f - 1f);
    }
}

```

MonteCarloBasedAI.cs

```

using CardGame;
using System;
using System.Collections.Generic;
using UnityEngine;

public class MonteCarloBasedAI : MonoBehaviour, PlayerAI
{
    [SerializeField] private GameStateManager gameManager;

    public int SimulationCount = 1000;
    private float confidenceAdjustment = 1.0f;

    public bool IsMLAgent { get; } = false;

    public void GoalReached(bool won, int pot, int id)
    {
        throw new NotImplementedException();
    }

    public ActionType MakeDecision(List<ActionType> possibleActions, List<Card> playerHand,
    List<List<Card>> visibleOpponentCards, List<Card> deck, float potOdds)
    {
        if (gameManager.State == GameState.ThirdStreet && possibleActions.Count == 1)
            return ActionType.SmallBet;
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						37
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        float winRate = PokerSimulator.SimulateWinProbability(playerHand,
visibleOpponentCards, deck, SimulationCount);

        int opponentCount = visibleOpponentCards.Count;
        float opponentFactor = 1f - (opponentCount * 0.08f);
        float dynamicThreshold = Mathf.Lerp(0.3f, 0.6f, potOdds);
        float riskFactor = Mathf.Lerp(0.7f, 1.3f, winRate - potOdds);
        float stageFactor = 1f - (int)gameManager.State * 0.05f;

        confidenceAdjustment *= (winRate > potOdds) ? 1.05f : 0.95f;
        confidenceAdjustment = Mathf.Clamp(confidenceAdjustment, 0.8f, 1.2f);

        float adjustedWinRate = winRate * opponentFactor * riskFactor * stageFactor *
confidenceAdjustment;

        if (adjustedWinRate > dynamicThreshold && possibleActions.Contains(ActionType.Raise))
            return ActionType.Raise;
        if (adjustedWinRate > dynamicThreshold * 0.9f &&
possibleActions.Contains(ActionType.SmallBet))
            return ActionType.SmallBet;
        if (adjustedWinRate > dynamicThreshold * 1.1f &&
possibleActions.Contains(ActionType.BigBet))
            return ActionType.BigBet;
        if ((adjustedWinRate > 0.25f || winRate > potOdds) &&
possibleActions.Contains(ActionType.Check))
            return ActionType.Check;
        if ((adjustedWinRate > 0.25f || winRate > potOdds) &&
possibleActions.Contains(ActionType.Call))
            return ActionType.Call;

        return ActionType.Fold;
    }

    public void RequestDecisionFromML(List<ActionType> possibleActions, List<Card>
playerHand, List<List<Card>> allVisibleOpponentCards, int pot, List<int> bets, int playerBet,
Action<ActionType> callback)
    {
        throw new NotSupportedException();
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						38
Зм.	Арк.	№ докум.	Підпис	Дата		

PauseMenu.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    [SerializeField] private GameObject pauseMenuUI;

    public bool GameIsPaused = false;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    private void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }
}
```

					ІАЛЦ.467200.007 Д4	Арк.
						39
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        public void ReturnToMainMenuPressed()
        {
            SceneManager.LoadSceneAsync(0);
        }

        public void QuitPressed()
        {
            Application.Quit();
        }
    }

```

Player.cs

```

using CardGame;
using JetBrains.Annotations;
using NUnit.Framework;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEditor;
using UnityEngine;
using static CardGame.PokerHandEvaluator;

public class Player : MonoBehaviour
{
    [SerializeField] private TableState tableState;
    [SerializeField] private UIManager uiManager;

    private PlayerAI playerAI;

    public int playerId;
    public bool isMainPlayer;
    public Action onTurnComplete;
    private BetManager betManager;
    private CardStack playerHand;
    private int minBet;

    private List<Card> knownCards = new List<Card>();

    public void ResetPlayer()

```

					ІАЛЦ.467200.007 Д4	Арк.
						40
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    {
        if (betManager == null) Debug.Log("betManager is null");
        betManager.ResetBetManager();
        playerHand.ResetStack();
        knownCards = new List<Card>();
        HideUI();
    }

    private void Awake()
    {
        playerAI = GetComponent<PlayerAI>();
        betManager = GetComponent<BetManager>();
        playerHand = GetComponent<CardStack>();
    }

    private void Start()
    {
        tableState.StateChanged += OnTableStateChanged;
    }

    private void OnTableStateChanged()
    {
        UpdateTableInfo();
    }

    private void UpdateTableInfo()
    {
        List<Card> myHand = playerHand.GetCards().ToList();
        knownCards = knownCards.Union(myHand).ToList();
        knownCards = knownCards.Union(tableState.GetAllVisibleCards()).ToList();
    }

    public void StartTurn(Action turnEndCallback, List<ActionType> possibleActions, int
currentBet)
    {
        onTurnComplete = turnEndCallback;
        minBet = currentBet;
        if (isMainPlayer)
        {
            ShowUI(possibleActions);

```

					ІАЛЦ.467200.007 Д4	Арк.
						41
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }
    else
    {
        StartCoroutine(BotTurn(possibleActions));
    }
}

public void NotifyGameResults(bool won)
{
    if (playerAI.IsMLAgent)
    {
        if (won)
            playerAI.GoalReached(won, tableState.potManager.pot - betManager.currentBet,
playerId);
        else
            playerAI.GoalReached(won, betManager.currentBet, playerId);
    }
}

private IEnumerator BotTurn(List<ActionType> possibleActions)
{
    yield return new WaitForSeconds(0.0f);

    UpdateTableInfo();

    List<Card> myHand = playerHand.GetCards().ToList();
    List<Card> remainingCards = GetRemainingCards(knownCards);
    float potOdds = (float)(minBet - betManager.currentBet) / tableState.potManager.pot;

    if (playerAI.IsMLAgent)
    {
        playerAI.RequestDecisionFromML(possibleActions, myHand,
tableState.GetAllPlayersVisibleCardsExcept(playerId).ToList(), tableState.potManager.pot,
tableState.GetAllPlayersBetsExcept(playerId).ToList(), betManager.currentBet,
OnBotDecision);
    }
    else
    {
        yield return new WaitForSeconds(0.2f);
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						42
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        ActionType decision = playerAI.MakeDecision(possibleActions, myHand,
tableState.GetPlayersVisibleCardsExcept(playerId).ToList(), remainingCards, potOdds);
        OnBotDecision(decision);
    }
}

private void OnBotDecision(ActionType decision)
{
    switch (decision)
    {
        case ActionType.SmallBet:
            SmallBet();
            break;
        case ActionType.BigBet:
            BigBet();
            break;
        case ActionType.Check:
            Check();
            break;
        case ActionType.Raise:
            Raise();
            break;
        case ActionType.Call:
            Call();
            break;
        case ActionType.Fold:
            Fold();
            break;
    }
}

private static List<Card> GetRemainingCards(List<Card> cardList)
{
    List<Card> deck = new List<Card>();

    foreach (Card.CardSign sign in System.Enum.GetValues(typeof(Card.CardSign)))
    {
        foreach (Card.CardValue value in System.Enum.GetValues(typeof(Card.CardValue)))
        {
            Card newCard = ScriptableObject.CreateInstance<Card>();

```

					ІАЛЦ.467200.007 Д4	Арк.
						43
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        newCard.cardSign = sign;
        newCard.cardValue = value;
        if (!cardList.Contains(newCard))
        {
            deck.Add(newCard);
        }
    }
}
return deck;
}

```

```

public void Check()
{
    if (isMainPlayer)
    {
        HideUI();
    }
    onTurnComplete?.Invoke();
}

```

```

public void SmallBet()
{
    betManager.Bet(50);
    if (isMainPlayer)
    {
        HideUI();
    }
    onTurnComplete?.Invoke();
}

```

```

public void BigBet()
{
    betManager.Bet(100);
    if (isMainPlayer)
    {
        HideUI();
    }
    onTurnComplete?.Invoke();
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						44
Зм.	Арк.	№ докум.	Підпис	Дата		

```

public void Raise()
{
    betManager.Bet(minBet - betManager.currentBet + 100);
    if (isMainPlayer)
    {
        HideUI();
    }
    onTurnComplete?.Invoke();
}

public void Call()
{
    betManager.Bet(minBet - betManager.currentBet);
    if (isMainPlayer)
    {
        HideUI();
    }
    onTurnComplete?.Invoke();
}

public void Fold()
{
    GetComponent<CardStack>().Fold();

    if (isMainPlayer)
    {
        HideUI();
    }
    onTurnComplete?.Invoke();
}

private void ShowUI(List<ActionType> possibleActions)
{
    uiManager.ShowActionButtons(possibleActions);
}

private void HideUI()
{
    uiManager.HideActionButtons();
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						45
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

public enum ActionType
{
    SmallBet,
    BigBet,
    Check,
    Raise,
    Call,
    Fold
}

```

PlayerAI.cs

```

using CardGame;
using System;
using System.Collections.Generic;

internal interface PlayerAI
{
    bool IsMLAgent { get; }

    void GoalReached(bool won, int pot, int id);
    public ActionType MakeDecision(List<ActionType> possibleActions, List<Card> playerHand,
List<List<Card>> visibleOpponentCards, List<Card> deck, float potOdds);
    void RequestDecisionFromML(List<ActionType> possibleActions, List<Card> playerHand,
List<List<Card>> allVisibleOpponentCards, int pot, List<int> bets, int playerBet,
Action<ActionType> callback);
}

```

PlayerBetDelegate.cs

```

public delegate void PlayerBetEventHandler(object sender, PlayerBetEventArgs e);

```

PlayerBetEventArgs.cs

```

using CardGame;

public class PlayerBetEventArgs
{
    public PlayerBetEventArgs(int currentBet, int playerCash)
    {

```

					ІАЛЦ.467200.007 Д4	Арк.
						46
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        CurrentBet = currentBet;
        PlayerCash = playerCash;
    }

    public int CurrentBet { get; private set; }
    public int PlayerCash { get; private set; }
}

```

PokerHandEvaluator.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

namespace CardGame
{
    public class PokerHandEvaluator
    {
        public enum HandRank
        {
            HighCard, OnePair, TwoPair, ThreeOfAKind, Straight, Flush, FullHouse, FourOfAKind,
            StraightFlush
        }

        public struct HandValue : IComparable<HandValue>
        {
            public HandRank Rank;
            public List<Card.CardValue> Kickers;

            public int CompareTo(HandValue other)
            {
                if (Rank != other.Rank)
                    return Rank.CompareTo(other.Rank);

                for (int i = 0; i < Kickers.Count; i++)
                {
                    if (Kickers[i] != other.Kickers[i])
                        return Kickers[i].CompareTo(other.Kickers[i]);
                }
                return 0;
            }
        }
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						47
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }
}

public static CardStack GetWeakestHand(List<CardStack> hands)
{
    if (hands == null || hands.Count == 0)
        throw new ArgumentException("The list cannot be empty");

    CardStack minHand = hands[0];

    for (int i = 1; i < hands.Count; i++)
    {
        HandValue minHandRank = EvaluateHand(minHand.GetVisibleCards().ToList());
        HandValue handRank = EvaluateHand(hands[i].GetVisibleCards().ToList());

        int cmp = minHandRank.CompareTo(handRank);

        if (cmp > 0)
        {
            minHand = hands[i];
        }
    }
    return minHand;
}

public static CardStack GetStrongestHand(List<CardStack> hands)
{
    if (hands == null || hands.Count == 0)
        throw new ArgumentException("The list cannot be empty");

    CardStack maxHand = hands[0];

    for (int i = 1; i < hands.Count; i++)
    {
        HandValue minHandRank = EvaluateHand(maxHand.GetVisibleCards().ToList());
        HandValue handRank = EvaluateHand(hands[i].GetVisibleCards().ToList());

        int cmp = minHandRank.CompareTo(handRank);

        if (cmp < 0)

```

					ІАЛЦ.467200.007 Д4	Арк.
						48
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        {
            maxHand = hands[i];
        }
    }
    return maxHand;
}

public static float EvaluateHandStrengthNormalized(List<Card> cards)
{
    if (cards == null || cards.Count == 0)
        throw new ArgumentException("Hand must contain at least one card.");

    HandValue handValue = EvaluateHand(cards);

    int totalRanks = Enum.GetValues(typeof(HandRank)).Length;

    float baseRankStrength = (float)handValue.Rank / (totalRanks - 1);

    float kickerScore = 0f;
    float maxKickerValue = Enum.GetValues(typeof(Card.CardValue)).Cast<int>().Max();
    float kickerWeight = 1f;

    foreach (var kicker in handValue.Kickers)
    {
        kickerScore += ((float)kicker / maxKickerValue) * kickerWeight;
        kickerWeight *= 0.1f;
    }

    float totalStrength = Mathf.Clamp01(baseRankStrength + kickerScore * 0.05f);

    return totalStrength * 2f - 1f;
}

public static HandValue EvaluateHand(List<Card> cards)
{
    if (cards.Count == 0)
        throw new ArgumentException("Hand must contain at least one card.");

    var sortedCards = cards.OrderByDescending(c => c.cardValue).ToList();

```

					ІАЛЦ.467200.007 Д4	Арк.
						49
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        var rankGroups = sortedCards.GroupBy(c => c.cardValue).OrderByDescending(g =>
g.Count()).ThenByDescending(g => g.Key);
        var suitGroups = sortedCards.GroupBy(c => c.cardSign);
        bool isFlush = suitGroups.Any(g => g.Count() >= 5);
        List<Card.CardValue> straight = GetLongestStraight(sortedCards);

        if (isFlush && straight.Count >= 5)
            return new HandValue { Rank = HandRank.StraightFlush, Kickers =
straight.Take(5).ToList() };

        if (rankGroups.First().Count() == 4)
            return new HandValue { Rank = HandRank.FourOfAKind, Kickers =
GetKickers(rankGroups, 4) };

        if (rankGroups.First().Count() == 3 && rankGroups.ElementAtOrDefault(1)?.Count()
>= 2)
            return new HandValue { Rank = HandRank.FullHouse, Kickers =
GetKickers(rankGroups, 3, 2) };

        if (isFlush)
            return new HandValue { Rank = HandRank.Flush, Kickers = GetFlush(sortedCards)
};

        if (straight.Count >= 5)
            return new HandValue { Rank = HandRank.Straight, Kickers =
straight.Take(5).ToList() };

        if (rankGroups.First().Count() == 3)
            return new HandValue { Rank = HandRank.ThreeOfAKind, Kickers =
GetKickers(rankGroups, 3) };

        if (rankGroups.First().Count() == 2 && rankGroups.ElementAtOrDefault(1)?.Count()
== 2)
            return new HandValue { Rank = HandRank.TwoPair, Kickers =
GetKickers(rankGroups, 2, 2) };

        if (rankGroups.First().Count() == 2)
            return new HandValue { Rank = HandRank.OnePair, Kickers =
GetKickers(rankGroups, 2) };

```

					ІАЛЦ.467200.007 Д4	Арк.
						50
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return new HandValue { Rank = HandRank.HighCard, Kickers = sortedCards.Select(c
=> c.cardValue).Take(5).ToList() };
    }

    private static List<Card.CardValue> GetLongestStraight(List<Card> sortedCards)
    {
        var uniqueRanks = sortedCards.Select(c => c.cardValue).Distinct().ToList();
        List<Card.CardValue> bestStraight = new List<Card.CardValue>();
        List<Card.CardValue> currentStraight = new List<Card.CardValue>();

        foreach (var rank in uniqueRanks)
        {
            if (currentStraight.Count == 0 || (int)currentStraight.Last() == (int)rank +
1)
                currentStraight.Add(rank);
            else
                currentStraight = new List<Card.CardValue> { rank };

            if (currentStraight.Count > bestStraight.Count)
                bestStraight = new List<Card.CardValue>(currentStraight);
        }

        return bestStraight;
    }

    private static List<Card.CardValue> GetFlush(List<Card> sortedCards)
    {
        return sortedCards.GroupBy(c => c.cardSign).Where(g => g.Count() >=
5).First().Select(c => c.cardValue).Take(5).ToList();
    }

    private static List<Card.CardValue>
GetKickers(IOrderedEnumerable<IGrouping<Card.CardValue, Card>> rankGroups, params int[]
counts)
    {
        List<Card.CardValue> kickers = new List<Card.CardValue>();
        HashSet<Card.CardValue> addedKickers = new HashSet<Card.CardValue>();

        foreach (var count in counts)
        {

```

					ІАЛЦ.467200.007 Д4	Арк.
						51
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        var group = rankGroups.Where(g => g.Count() == count)
                                .OrderByDescending(g => g.Key)
                                .FirstOrDefault();
        if (group != null && addedKickers.Add(group.Key))
        {
            kickers.Add(group.Key);
        }
    }

    foreach (var group in rankGroups)
    {
        if (!addedKickers.Contains(group.Key))
        {
            kickers.Add(group.Key);
            if (kickers.Count == 5) break;
        }
    }

    return kickers;
}

```

```

public static Dictionary<HandRank, float> EvaluateHandProbabilities(List<Card>
partialHand, List<Card> remainingCards)
{
    if (partialHand == null || remainingCards == null)
        throw new ArgumentNullException("Card lists cannot be null.");

    if (partialHand.Count < 3 || partialHand.Count > 7)
        throw new ArgumentException("Partial hand must have between 3 and 7 cards.");

    Dictionary<HandRank, int> handRankCounts = new Dictionary<HandRank, int>();
    foreach (HandRank rank in Enum.GetValues(typeof(HandRank)))
    {
        handRankCounts[rank] = 0;
    }

    int cardsNeeded = 7 - partialHand.Count;
    var combinations = GetCombinations(remainingCards, cardsNeeded);
    int totalCombinations = 0;
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						52
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        foreach (var combo in combinations)
        {
            var fullHand = new List<Card>(partialHand);
            fullHand.AddRange(combo);
            HandValue evaluatedHand = EvaluateHand(fullHand);
            handRankCounts[evaluatedHand.Rank]++;
            totalCombinations++;
        }

        Dictionary<HandRank, float> handRankProbabilities = new Dictionary<HandRank,
float>();

        foreach (var rank in handRankCounts.Keys)
        {
            handRankProbabilities[rank] = (float)handRankCounts[rank] /
totalCombinations;
        }

        return handRankProbabilities;
    }

    private static IEnumerable<IEnumerable<T>> GetCombinations<T>(IEnumerable<T> items,
int count)
    {
        if (count == 0)
        {
            yield return new List<T>();
        }
        else
        {
            int i = 0;
            foreach (var item in items)
            {
                var remainingItems = items.Skip(i + 1);
                foreach (var combination in GetCombinations(remainingItems, count - 1))
                {
                    yield return new List<T> { item }.Concat(combination);
                }
                i++;
            }
        }
    }

```

					ІАЛЦ.467200.007 Д4	Арк.
						53
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }

    }
}

PokerSimulator.cs
using CardGame;
using System.Collections.Generic;
using System;
using UnityEngine;

public class PokerSimulator
{
    private const int SimulationCount = 100000;

    public static float SimulateWinProbability(List<Card> playerHand, List<List<Card>>
visibleOpponentCards, List<Card> remainingCards)
    {
        if (playerHand == null || visibleOpponentCards == null || remainingCards == null)
            throw new ArgumentNullException("Card lists cannot be null.");

        if (playerHand.Count < 3 || playerHand.Count > 7)
            throw new ArgumentException("Player hand must have between 3 and 7 cards.");

        int opponentCount = visibleOpponentCards.Count;
        int playerWins = 0;

        for (int i = 0; i < SimulationCount; i++)
        {
            var deck = new List<Card>(remainingCards);
            deck.Shuffle();

            var playerFullHand = new List<Card>(playerHand);
            while (playerFullHand.Count < 7)
            {
                playerFullHand.Add(deck.Pop());
            }

            List<List<Card>> opponentHands = new List<List<Card>>();

```

```

for (int j = 0; j < opponentCount; j++)
{
    var opponentHand = new List<Card>(visibleOpponentCards[j]);
    while (opponentHand.Count < 7)
    {
        opponentHand.Add(deck.Pop());
    }
    opponentHands.Add(opponentHand);
}

var playerHandValue = PokerHandEvaluator.EvaluateHand(playerFullHand);
bool playerWinsRound = true;

foreach (var opponentHand in opponentHands)
{
    var opponentHandValue = PokerHandEvaluator.EvaluateHand(opponentHand);
    try
    {
        if (opponentHandValue.CompareTo(playerHandValue) >= 0)
        {
            playerWinsRound = false;
            break;
        }
    }
    catch (Exception ex)
    {
        foreach (Card card in playerFullHand)
            Debug.Log(card.ToString());
        foreach (Card card in opponentHand)
            Debug.Log(card.ToString());
        throw ex;
    }
}

if (playerWinsRound)
{
    playerWins++;
}
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						55
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return (float)playerWins / SimulationCount;
    }

    public static float SimulateWinProbability(List<Card> playerHand, List<List<Card>>
visibleOpponentCards, List<Card> remainingCards, int simulationCount)
    {
        if (playerHand == null || visibleOpponentCards == null || remainingCards == null)
            throw new ArgumentNullException("Card lists cannot be null.");

        if (playerHand.Count < 3 || playerHand.Count > 7)
            throw new ArgumentException("Player hand must have between 3 and 7 cards.");

        int opponentCount = visibleOpponentCards.Count;
        int playerWins = 0;

        for (int i = 0; i < simulationCount; i++)
        {
            var deck = new List<Card>(remainingCards);
            deck.Shuffle();

            var playerFullHand = new List<Card>(playerHand);
            while (playerFullHand.Count < 7)
            {
                playerFullHand.Add(deck.Pop());
            }

            List<List<Card>> opponentHands = new List<List<Card>>();

            for (int j = 0; j < opponentCount; j++)
            {
                var opponentHand = new List<Card>(visibleOpponentCards[j]);
                while (opponentHand.Count < 7)
                {
                    opponentHand.Add(deck.Pop());
                }
                opponentHands.Add(opponentHand);
            }

            var playerHandValue = PokerHandEvaluator.EvaluateHand(playerFullHand);
            bool playerWinsRound = true;

```

					ІАЛЦ.467200.007 Д4	Арк.
						56
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        foreach (var opponentHand in opponentHands)
        {
            var opponentHandValue = PokerHandEvaluator.EvaluateHand(opponentHand);
            try
            {
                if (opponentHandValue.CompareTo(playerHandValue) >= 0)
                {
                    playerWinsRound = false;
                    break;
                }
            }
            catch (Exception ex)
            {
                Debug.Log("main player:");
                foreach (Card card in playerFullHand)
                    Debug.Log(card.ToString());
                Debug.Log("bot:");
                foreach (Card card in opponentHand)
                    Debug.Log(card.ToString());
                throw ex;
            }
        }

        if (playerWinsRound)
        {
            playerWins++;
        }
    }

    return (float)playerWins / simulationCount;
}

public static class Extensions
{
    private static System.Random rng = new System.Random();

    public static void Shuffle<T>(this IList<T> list)
    {

```

					ІАЛЦ.467200.007 Д4	Арк.
						57
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        int n = list.Count;
        while (n > 1)
        {
            n--;
            int k = rng.Next(n + 1);
            T value = list[k];
            list[k] = list[n];
            list[n] = value;
        }
    }

    public static T Pop<T>(this IList<T> list)
    {
        var item = list[list.Count - 1];
        list.RemoveAt(list.Count - 1);
        return item;
    }
}

```

PotManager.cs

```

using TMPro;
using UnityEngine;

public class PotManager : MonoBehaviour
{
    public int pot = 0;
    public TextMeshProUGUI potText;

    public void ResetPot()
    {
        pot = 0;
        potText.text = pot.ToString();
    }

    public void Add(int amount)
    {
        pot += amount;
        potText.text = pot.ToString();
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						58
Зм.	Арк.	№ докум.	Підпис	Дата		

TableState.cs

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

public class TableState : MonoBehaviour
{
    public event Action StateChanged;

    public List<GameObject> players;
    public List<CardStack> playersCardStacks;
    public List<BetManager> playersBetManagers;
    public PotManager potManager;
    public int currentPlayerIndex;

    public void ResetState()
    {
        foreach(GameObject player in players)
        {
            player.GetComponent<Player>().ResetPlayer();
        }
        potManager.ResetPot();
    }

    private void Awake()
    {
        foreach(GameObject player in players)
        {
            CardStack cardStack = player.GetComponent<CardStack>();
            cardStack.CardRemoved += OnCardAdded;
            cardStack.CardAdded += OnStateRemoved;
            playersCardStacks.Add(cardStack);
            playersBetManagers.Add(player.GetComponent<BetManager>());
        }
    }

    private void OnCardAdded(object sender, CardRemovedEventArgs e)
    {
    }
```

```

        StateChanged?.Invoke();
    }

    private void OnStateRemoved(object sender, CardAddedEventArgs e)
    {
        StateChanged?.Invoke();
    }

    public int GetPlayerBet(int index)
    {
        return playersBetManagers[index].currentBet;
    }

    public int GetMainPlayerBet()
    {
        return GetPlayerBet(players.FindIndex(p => p.GetComponent<Player>().isMainPlayer ==
true));
    }

    public IEnumerable<List<Card>> GetPlayersVisibleCards()
    {
        foreach(CardStack cardStack in playersCardStacks)
        {
            if (!cardStack.isFolded)
                yield return cardStack.GetVisibleCards().ToList();
        }
    }

    public IEnumerable<List<Card>> GetPlayersVisibleCardsExcept(int playerId)
    {
        foreach (GameObject player in players)
        {
            if (player.GetComponent<Player>().playerId != playerId)
            {
                CardStack cardStack = player.GetComponent<CardStack>();
                if (!cardStack.isFolded)
                    yield return cardStack.GetVisibleCards().ToList();
            }
        }
    }
}

```

```

public IEnumerable<List<Card>> GetAllPlayersVisibleCardsExcept(int playerId)
{
    foreach (GameObject player in players)
    {
        if (player.GetComponent<Player>().playerId != playerId)
        {
            CardStack cardStack = player.GetComponent<CardStack>();
            yield return cardStack.GetVisibleCards().ToList();
        }
    }
}

public IEnumerable<int> GetAllPlayersBetsExcept(int playerId)
{
    foreach (GameObject player in players)
    {
        if (player.GetComponent<Player>().playerId != playerId)
        {
            BetManager betManager = player.GetComponent<BetManager>();
            yield return betManager.currentBet;
        }
    }
}

public IEnumerable<int> getPlayersBetsExcept(int playerId)
{
    foreach (GameObject player in players)
    {
        if (player.GetComponent<Player>().playerId != playerId)
        {
            CardStack cards = player.GetComponent<CardStack>();
            if (!cards.isFolded)
            {
                yield return player.GetComponent<BetManager>().currentBet;
            }
        }
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						61
Зм.	Арк.	№ докум.	Підпис	Дата		

```

public List<Card> GetAllVisibleCards()
{
    List<Card> cards = new List<Card>();
    foreach (CardStack cardStack in playersCardStacks)
    {
        if (!cardStack.isFolded)
        {
            foreach (Card card in cardStack.GetVisibleCards()) { cards.Add(card); }
        }
    }
    return cards;
}

public IEnumerable<int> GetPlayersBetCounters()
{
    foreach (GameObject player in players)
    {
        CardStack cards = player.GetComponent<CardStack>();
        if (!cards.isFolded)
        {
            yield return player.GetComponent<BetManager>().betCounter;
        }
    }
}

public IEnumerable<int> GetPlayersBetCountersExcept(int playerId)
{
    foreach (GameObject player in players)
    {
        if (player.GetComponent<Player>().playerId != playerId)
        {
            CardStack cards = player.GetComponent<CardStack>();
            if (!cards.isFolded)
            {
                yield return player.GetComponent<BetManager>().betCounter;
            }
        }
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						62
Зм.	Арк.	№ докум.	Підпис	Дата		

```

public IEnumerable<int> GetPlayersBets()
{
    foreach (GameObject player in players)
    {
        CardStack cards = player.GetComponent<CardStack>();
        if (!cards.isFolded)
        {
            yield return player.GetComponent<BetManager>().currentBet;
        }
    }
}

public int GetPot()
{
    return potManager.pot;
}
}

```

UIManager.cs

```

using NUnit.Framework;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class UIManager : MonoBehaviour
{
    public Button foldButton;
    public Button callButton;
    public Button raiseButton;
    public Button checkButton;
    public Button smallBetButton;
    public Button bigBetButton;

    public void ShowActionButtons(List<ActionType> possibleActions)
    {
        foreach (ActionType action in possibleActions)
        {
            switch (action)

```

					ІАЛЦ.467200.007 Д4	Арк.
						63
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        {
            case ActionType.Fold:
                EnableButton(foldButton);
                break;
            case ActionType.Call:
                EnableButton(callButton);
                break;
            case ActionType.Raise:
                EnableButton(raiseButton);
                break;
            case ActionType.Check:
                EnableButton(checkButton);
                break;
            case ActionType.SmallBet:
                EnableButton(smallBetButton);
                break;
            case ActionType.BigBet:
                EnableButton(bigBetButton);
                break;
            default:
                break;
        }
    }
}

```

```

public void HideActionButtons()
{
    DisableButton(foldButton);
    DisableButton(callButton);
    DisableButton(raiseButton);
    DisableButton(checkButton);
    DisableButton(smallBetButton);
    DisableButton(bigBetButton);
}

```

```

private void EnableButton(Button btn)
{
    btn.interactable = true;
    var colors = foldButton.colors;
    colors.normalColor = Color.white;
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						64
Зм.	Арк.	№ докум.	Підпис	Дата		

```
        foldButton.colors = colors;
    }

    private void DisableButton(Button btn)
    {
        btn.interactable = false;
        var colors = foldButton.colors;
        colors.normalColor = Color.grey;
        foldButton.colors = colors;
    }
}
```

					ІАЛЦ.467200.007 Д4	Арк.
						65
Зм.	Арк.	№ докум.	Підпис	Дата		