

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

«Розробка карткової гри TCG на Unity»

Виконала студентка групи КН-19-1
ТАШКІНОВА Олександра Олександрівна

Керівник роботи:
АРТАМОНОВ Євген Борисович

Консультант:
ТЕРЕЩУК Степан Олександрович

Рецензент:
ГРАФ Марина Сергіївна

Житомир – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ
Зав. кафедри комп'ютерних наук
Марина ГРАФ
«05» січня 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти Ташкінова Олександра Олександрівна

Керівник роботи: Артамонов Євген Борисович

Тема роботи: «Розробка карткової гри TCG на Unity»

Затверджена Наказом університету від «05» січня 2023 р. № 10/с

Термін здачі студентом закінченої роботи «10» червня 2023 р.

Вихідні дані роботи: Об'єктом дослідження є розробка ігрового додатку.

Предметом дослідження є використання рушія Unity для розробки гри.

Консультанти випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Терещук С.О.	10.02.2023р	10.02.2023р
2	Терещук С.О.	18.04.2023р	18.04.2023р
3	Терещук С.О.	16.05.2023р	16.05.2023р

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел	8 березня 2023 – 10 березня 2023	Виконано
2	Проектування структури системи	15 березня 2023 – 20 квітня 2023	Виконано
3	Написання тестового програмного коду	1 квітня 2023 – 30 квітня 2023	Виконано
4	Тестування системи	1 травня 2023 – 3 травня 2023	Виконано

5	Переписування коду	18 червня 2023 – 23 червня 2023	Виконано
6	Друге тестування	24 червня 2023 – 25 червня 2023	Виконано
7	Оформлення пояснювальної записки	16 червня 2023 – 27 червня 2023	Виконано
8	Попередній захист	1 червня 2023	

Здобувач вищої освіти: **Олександра ТАШКІНОВА**
Керівник **Євген АРТАМОНОВ**

РЕФЕРАТ

Кваліфікаційна робота бакалавра складається з пояснювальної записки та програмного комплексу гри у жанрі «TCG». Пояснювальна записка до випускної кваліфікаційної роботи містить 58 сторінок, 18 ілюстрацію та 2 таблицю.

Метою роботи є розробка гри від першої особи у жанрі «TCG» за допомогою рушія Unity.

Під час розробки проекту розглянуто та проаналізовано інші ігрові розробки у жанрі «TCG» та обґрунтовано вибір кожного рішення реалізації. Спроектowana структура програмної частини, наведені сценарії роботи, створений весь графічний контент для гри, проведено тестування ігрової розробки.

КЛЮЧОВІ СЛОВА: КАРТА, КОЛЕКЦІЙНА, ІГРОВИЙ РУШІЙ, ВИКЛИК, SUMMONED, UNITY

					КН.КР.Б – 122 – 23 – ПЗ								
Зм.	Арк.	№ докум.	Підпис	Дата									
Розроб.		О.О.ТАШКІНОВА			Розробка карткової гри TCG на Unity Пояснювальна записка				Лім.	Арк.	Аркушів		
Керівник		Є. Б. АРТАМОНОВ									4	0	
									Житомирська політехніка, група КН-19-1				
Рецензент		М.С. ГРАФ											
Зав. каф.		М.С. ГРАФ											

ABSTRACT

Bachelor's graduate work consist of software complex of game genre TCG or CCG. The explanatory note to the final qualification work contains 58 pages, 18 illustrations and 2 tables.

The goal of the work is to develop a collection card game using the Unity engine.

During development of the project, other game developments in the "TCG" genre were considered and analyzed, and the choice of each implementation solution was justified. The structure of the game was designed, the work scenarios were given, all graphic content for game was created, and the game development was tested.

KEYWORDS: CARD, COLLECTON, GAME ENGINE, UNITY

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	5
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту.....	9
1.3. Вибір архітектури ігрового додатку	13
1.4. Обґрунтування вибору інструментів та вимоги до апаратного забезпечення	15
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ТА РЕАЛІЗАЦІЯ ДОДАТКУ.....	18
2.1. Визначення варіантів використання та структура системи	18
2.1.1. Діаграма варіантів використання.....	18
2.1.2. Діаграма класів	19
2.2. Дизайн документ.....	21
2.2.1. Концепт	21
2.2.2. Геймплей	21
2.2.3. Сцена/Локація.....	22
2.2.4. Текстури, персонажі та різні гейм-об'єкти.....	22
2.2.5. Керування.....	23
2.3. Реалізація алгоритмів роботи гри	23
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ	30
3.1. Порядок встановлення та налаштування системи	30
3.2. Інтерфейс та порядок роботи з програмним кодом	30
ВИСНОВКИ.....	36
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	37
ДОДАТКИ.....	39

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

UI(User Interface) – інтерфейс користувача.

HUD(Heads Up Display) – елементи швидкого доступу інтерфейсу користувача.

UE – Unreal Engine.

ОЗП – Оперативно запам'ятовуючий пристрій.

FPS(Frames per second) – кадрів на секунду.

TCG – Trading Card Game

CCG – Collection Card Game

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	7
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Популярність комп'ютерних ігор зростає з кожним днем, діти та дорослі проводять за цим заняттям все більше часу. Це не тільки розважає, а й стає засобом втілення фантазій, допомагає на мить забути свої проблеми і приємно провести час з друзями особливо в тяжкі часи.

Мабуть кожний у дитинстві щось колекціонував будь-то фішки, вкладиші з під жувальних гумок, міні-фігурки в шоколадних яєць і звичайно карти. Всі ми пам'ятаємо карти з «Людина павук» та «Черепашки ніндзя» їх було приємно не тільки збирати, а і грати ними з друзями. Такі ігри мають назву TCG(trading card game) або CCG(collectible card game) що дослівно перекладається як «Колекційна карткова гра».

Геймплей у CCG, як правило, покрокова гра, коли кожен гравець починає з перемішаної колоди, а потім у свій хід тягне та розіграє карти, щоб атакувати іншого гравця та зменшувати його очки здоров'я до нуля, перш ніж його противник зможе зробити те саме з ним. Гральні кубики, фішки, карткові рукава або ігрові килимки використовуються для доповнення ігрового процесу. Існують турніри CCG, де досвідчені гравці змагаються за призи.

Гравці збирають карти в особисту колекцію, з якої вони створюють власні колоди карт і кидають виклик іншим гравцям у матчах. Метою більшості CCG є перемогти опонента, створивши налаштовані колоди, які грають на синергію комбінацій карт. Досконалі колоди намагатимуться врахувати випадковість, створену початковим тасуванням колоди, а також дії суперника, використовуючи додаткові та бажано ефективні карти.

Отже метою випускної кваліфікаційної роботи є створення захоплюючої колекційної карткової гри, яка надасть гравцям унікальну можливість насолоджуватись досвідом напруженості та азартом.

Об'єктом дослідження є процес розробки карткових ігор на Unity.

Предметом дослідження є використання рушія Unity для створення карткової гри.

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	8
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1. Постановка задачі

Основною метою розробки TCG гри є надання гравцеві позитивних емоцій та забезпечення гарного проведення часу під час користування додатком. Гра може як розслабляти, так і кинути виклик гравцеві, змушуючи його продумувати хід наперед.

У роботі необхідно розробити додаток цікавим та захоплюючим. Для реалізації поставленого завдання, основними етапами є:

1. Розробити дизайн користувацького інтерфейсу.

1.1. Створити загальну концепцію дизайну користувацького інтерфейсу.

1.2. За допомогою графічного редактора створити основні елементи UI:

1.2.1. Головне меню.

1.2.2. Галерея карт.

1.2.3. Інтерфейс під час раунду

1.2.4. Редактор колоди

1.2.5. Магазин.

2. Проектування та розробка основних ігрових модулів

2.1. Модуль ігрових механік.

2.2. Модуль карт.

2.2.1. Карти-герої.

2.2.2. Карти-заклинання.

2.3. Результатом реалізації поставленого завдання є додаток у вигляді гри.

1.2. Аналіз аналогів програмного продукту

На сьогоднішній день існує багато ігор у жанрі TCG проте не всі вони мають ігрову адаптацію. Буде розглянуто декілька з відомих представників цього жанру на які буде опиратись та орієнтуватись розробка, а в кінці будуть порівняні між собою.

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	9
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

«Pokemon TCG Online» - безкоштовна онлайн-версія колекційної карткової гри Pokémon для безкоштовного завантаження та гри.

Базується на колекціонуванні рідкісних покемонів та побудові власної стратегії при наборі колоди, яка потрібна для битви з іншими гравцями. Потрібно вибити всіх покемонів вашого опонента, взяти всі ваші призові карти або колоду опонента, спричинивши вичерпання його стопки карт, щоб виграти

Посилання на ресурс: <https://tcg.pokemon.com/en-us/tcgl/>



Рис.1.1.Інтерфейс Pokemon TCG Online

Legends Of Runeterra –цифрова колекційна карткова гра, в яку грають один на одного. На початку показник здоров'я обох гравців має 20. Той, хто першим впаде до нуля, програє. Гравці починають кожен матч із рукою з чотирьох карт, які вони можуть обміняти на іншу випадкову карту зі своєї колоди. У кожному раунді обидва гравці беруть по одній карті. Карти розігруються, витрачаючи ману; гравці починають з нуля мани та отримують один додатковий кристал мани за раунд до максимум десяти. Максимум три невитрачені мани автоматично зберігаються в кінці раунду як мана заклинань; це можна

використовувати в наступних раундах для заклинань, але не може викликати карти одиниць.

Однією з відмінних рис гри є її бойовий темп. У кожному раунді «жетон атаки», символ, який вказує, хто з гравців може атакувати, а хто захищатися, змінюється від гравця до гравця. Це відображається візуально на половині дошки кожного гравця зі значком меча, який символізує атаку, або щитом для захисту. Деякі карти дозволяють гравцям атакувати, якщо у них немає жетона атаки.

Посилання на ресурс: <https://playruneterra.com/ru-ru/>

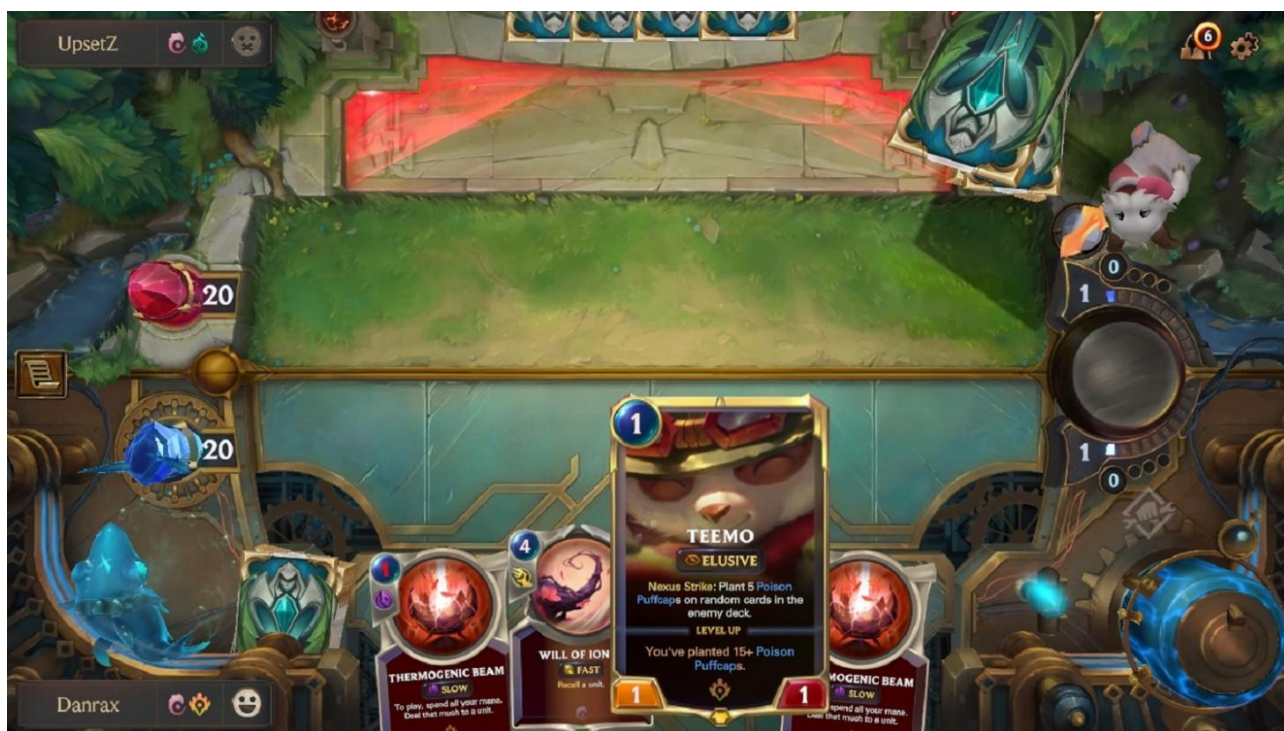


Рис.1.2 Інтерфейс Legends Of Runeterra

«Yu-Gi-Oh! Duel Links» - безкоштовна цифрова колекційна карткова гра ,розроблена Копамі на основі Yu-Gi-Oh!

У грі використовується формат, відомий як «Швидкісні дуелі», який використовує правила карткової гри з різними модифікаціями. Гравці мають 4000 очок життя, 2 фази бою, кількість зон монстрів і зон заклинань/пасток зменшено до 3, розмір основної колоди у кожній до 20-30 карт і додаткову колоду з 5 (хоча цю кількість можна збільшити до 8, виконавши певні місії), і гравці починають із 4 картками вони також можуть використовувати навички, які

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	11
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

різними способами впливають на дуелі. Ці ефекти включають, але не обмежуються додаванням карт до рук або поля, збільшенням характеристик монстрів, збільшенням очок життя, додаванням карт до колоди на початку дуелі, зміною початкової руки гравця та розкриттям вмісту обличчям вниз картки. Переможець дуелі або зменшує одиниці життя свого супротивника до 0, спустошує колоду суперника, або виграти за особливих умов. Гравці отримують бали досвіду та різні предмети після дуелі, причому більше предметів отримують за перемогу.



Рис.1.3 інтерфейс Yu-Gi-Oh! Duel Links

Посилання на ресурс: https://www.konami.com/yugioh/duel_links/en/

Підведемо підсумки

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	12
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

Функціональні можливості

	Навчання	Мультиплеєр	Одноманітний геймплей	Можливість грати як на телефоні та і на ПК	Оптимізація
Pokemon TCG Online	-	+	+	+	+
Legends Of Runeterra	+	+	+/-	-	-
Yu-Gi-Oh! Duel Links	+	+	-	-	+/-

Проаналізувавши ігри даного жанру ми склали основне розуміння та розставили пріоритети. Важливим є те, щоб вона відповідала таким критеріям:

1. Мала різноманітну кількість унікальних карт
2. Відчуття що ви граєте справжнім суперником
3. У гри мають бути правила завдяки яким гра має стати цікавішою
4. Має бути можливість зіставляти свою колоду
5. Гра повинна бути інтуїтивно зрозумілою
6. Додання магазину та системи нагороди

1.3. Вибір архітектури ігрового додатку

Для вибору відповідної архітектури, яка б найкраще підходила для розробки, розглянемо декілька з популярних патернів та підходів:

1. MVC (Model-View-Controller) – цей патерн розділяє гру на три компоненти: модель, представлення та контроллер. Модель представляє дані та бізнес-логіку, представлення відображає інтерфес користувача, а контроллер відповідає за обробку вводу від користувача та зміну моделі. Із переваг можна відмітити:

- Полегшення розробки та подальшої підтримки гри завдяки розділенню відповідальностей;
- Можливість змінювати окремі компоненти не впливаючи на інші;

Недоліки:

- Довге налаштування патерну;
- Шанс створення zombie – екземплярів, які впливають не лише на продуктивність, а можуть бути ще й причиною багів;
- При великій кількості об'єктів виникає плутанина між їхніми взаємодіями;

2. ECS (Entity-Component-System) – патерн заснований на концепції розділення сутностей, компонентів та систем. Кожна сутність складається з компонентів, які визначають її властивість на поведінку. Системи оброблюють компоненти та виконують необхідні операції. Також варто відмітити, що Unity має власний фреймворк Unity ECS який полегшує розробку гри з використанням архітектури ECS, але використання такого фреймворку ліпше розглядати у проекті зі складною логікою та великою кількістю об'єктів таких як у іграх-стратегіях. Перевагами такого підходу можна назвати:

- Високу продуктивність та ефективне використання ресурсів;
- Легке створення нових типів сутностей і компонентів;
- Ідеально підходить для гри з великою кількістю об'єктів;

Із недоліків відзначимо:

- Вимагає більше часу для задіяння патерну;
- Потребує грамотного керування життєвим циклом сутностей та компонентів;

3. MVP (Model-View-Presenter) – це архітектурний патерн, де модель представляє дані і логіку, представлення відповідає за відображення інтерфейсу користувача, а представник обробляє події від користувача та спілкується з моделлю та представленням. Серед переваг MVP є:

- Розділення обов'язків;

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	14
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

- Легко тестувати, оскільки логіка зосереджена в present, його можна легко тестувати без залежності від інтерфейсу користувача;
- Модифікування додатку є простою, без впливу на інші компоненти;

З недоліків є:

- Складність реалізації. Іноді реалізація цього патерну може бути складною і призводити до написання зайвого коду у великих проектах;
- Багато зв'язків, оскільки презентер повинен взаємодіяти з моделлю і представленням, це також збільшить кількість залежностей між компонентами;

Розглянувши декілька архітектурних патернів можна зробити висновок, що найбільш відповідним патерном є ECS, оскільки він має високу продуктивність та ефективно використовує ресурси на відміну від сімейства патернів MVC, також з ним легко створювати нові об'єкти. Більше того, Unity орієнтована на використання дизайн патерну компонентного підходу, що підходить для використання вкупі з ECS.

Отже, можна зробити висновок, що даний архітектурний шаблон спростить написання коду та підвищить продуктивність додатку, що дозволяє реалізувати всі поставлені завдання.

1.4. Обґрунтування вибору інструментів та вимоги до апаратного забезпечення

На вибір є декілька популярних доступних рушіїв: Unity, Unreal Engine та Godot. Наш пріоритет це гнучкий, інтерфейс який буде інтуїтивно зрозумілий, щоб легко засвоїти сам інструмент, під цей опис ідеально підходить Unity. Вибір даного рушія зумовлено декількома перевагами.

Перша перевага – зрозумілий та дружелюбний інтерфейс(рис. 1.4), він є легким у використанні, що дозволяє швидко орієнтуватись та приступити до розробки ігор навіть новачкам, довго в ньому не розбираючись.

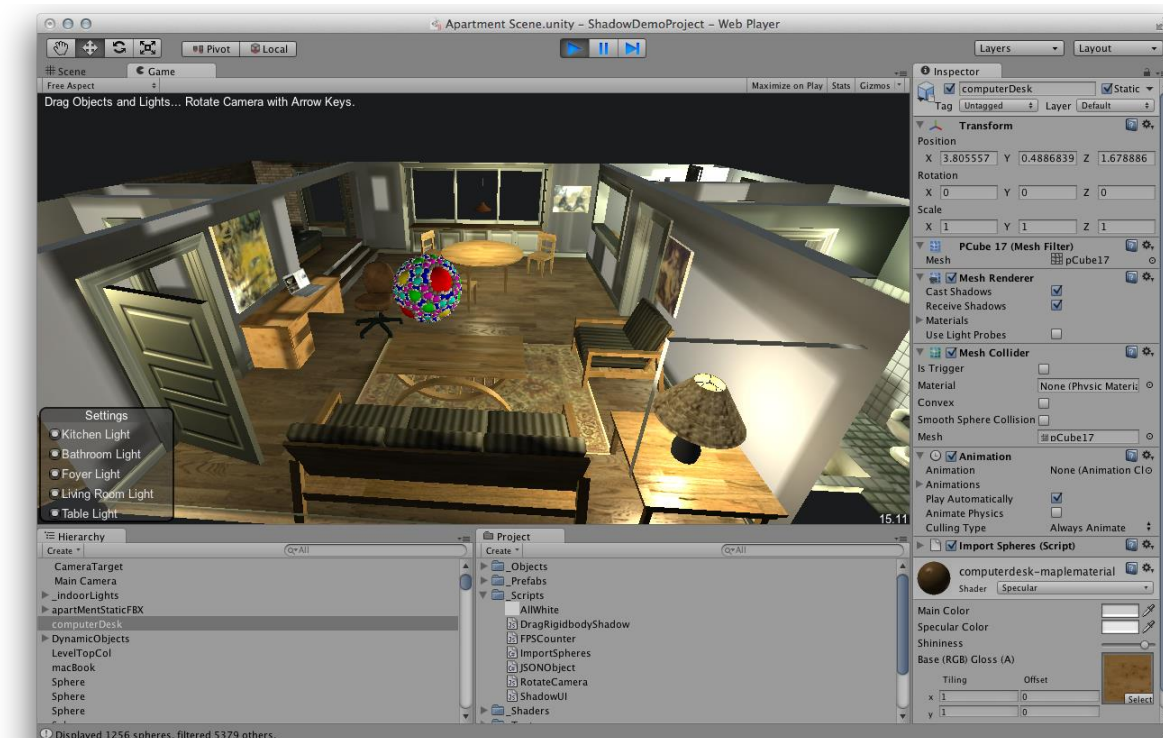


Рис.1.4. Інтерфейс Unity

Друга перевага – скрипти пишуться мовою C#. Дана мова є вискорівневою, що дозволяє легко її вивчити, засвоїти і приступити до розробки гри. Також, у Unity є ще візуальна мова програмування. C# є важливою перевагою, оскільки дозволяє швидше приступити до розробки, на відмінну від Unreal Engine, де скрипти пишуться мовою C++, що є складнішою. Так, у UE є blue prints – що є візуальним програмуванням, але так чи інакше скрипти є набагато гнучкішими та ліпшими у створенні ігрової логіки. Також у Unity реалізовано багато функцій, які б у UE довелось писати самостійно.

Ще однією перевагою є кросплатформеність додатків скомпільованих на Unity, це дозволяє випускати додатки на більшу кількість платформ з мінімальними змінами.

Іншою перевагою є документація рушія. Документація Unity є також дружелюбною та зрозумілою. Документація охоплює всі аспекти розробки, які

можуть знайти відповіді на свої питання щодо графіки, шейдерів, звуку, фізики, мережевої взаємодії та багато іншого.

В документації також практично до кожного методу/класу є велика кількість прикладу коду, що допомагає розробникам зрозуміти, як використовувати різні функції і компоненти рушія. Це особливо корисно для початківців, які можуть вивчати шаблони переглядаючи існуючі приклади.

І останньою перевагою є спільнота рушія. Unity має велику та активну спільноту розробників, яка допомагає одне одному та ділиться знаннями. На форумах Unity розробники можуть знайти відповіді на свої запитання від розробників самого рушія, або ж інших учасників, обговорювати проблеми та отримувати поради від інших досвідчених розробників.

Висновки до першого розділу

Проведений аналіз предметної області, визначено основні компоненти які має мати гра. Було розглянуто наявні аналоги та виявлення особливості, які має мати майбутній додаток: зручний UI, відмінну особливість у геймплеї та наявність хорошого аудіо, цікаві але при цьому прості у розумінні механіки, працюючу систему збереження

Визначили оптимальну архітектуру, обрано необхідний пакет програмних продуктів для реалізації поставлених задач.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ТА РЕАЛІЗАЦІЯ ДОДАТКУ

2.1. Визначення варіантів використання та структура системи

2.1.1. Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) – це діаграма яка допомагає відображати взаємодію між акторами(користувачами системи) та системою у рамках конкретних сценаріїв використання.

Діаграма варіантів використання складається з акторів, безпосередньо самих варіантів використання та взаємозв'язків між ними. Актори представляють зовнішні сутності, такі як, користувачі, системи або зовнішні сервіси, які взаємодіють з системою. Варіанти використання описують функціональні можливості системи та взаємодію між акторами та системою.

Для розробки діаграми варіантів використання було визначено одного актора – гравця.

На рис 2.1 зображено діаграму варіантів використання у процесі гри.

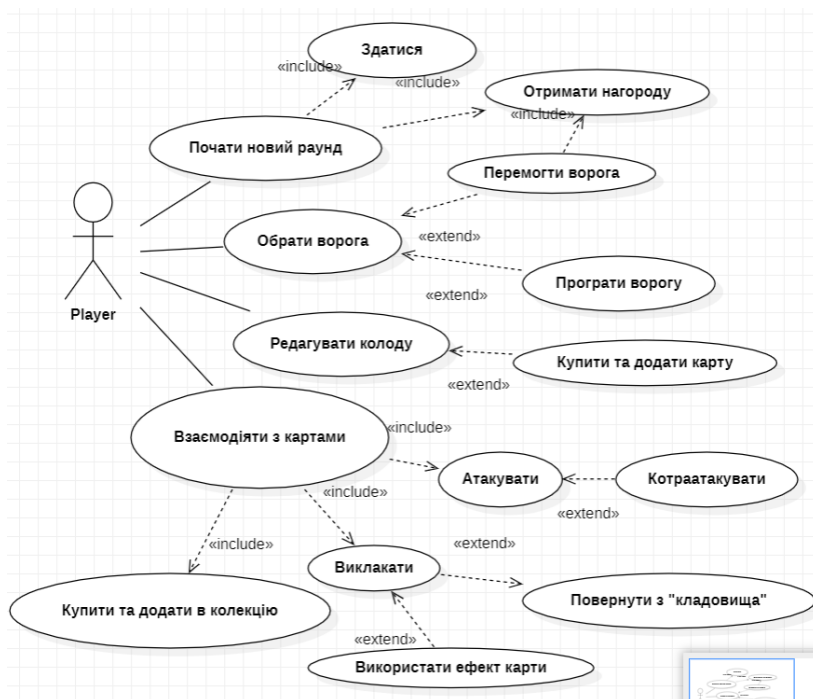


Рис.2.1. Діаграма варіантів використання

На діаграмі варіантів використання(рис.2.1) зображений один актор – гравець.

Функціональні вимоги:

1. Карти повинні: братися, кластися на поле, знищуватися;

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	18
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

2. Ви не можете повернути карту у руку якщо вона вже на столі;
3. Ви не можете атакувати ні героя, ні інші карти якщо хід не ваш;
4. Кожна карта коштує якусь к-сть мани, ви не можете призвати або використати карту якщо у вас не вистачає мани;
5. Вище сказані правила стосуються і ворога
6. Ефект карт мають спрацьовувати та не ламати гру
7. Ігрова валюта має зберігатися і видаватися після перемоги

Не функціональні вимоги:

- Продуктивність: гра повинна бути добре оптимізована та підтримувати безперебійну роботу;.

2.1.2. Діаграма класів

Проаналізувавши функціональні вимоги були побудовані сутності з яких складається гра. На рис.2.2 показана діаграма класів, сцени бою, ці класи відповідають за перемеміщення та поведінку карт на полі

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	19
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

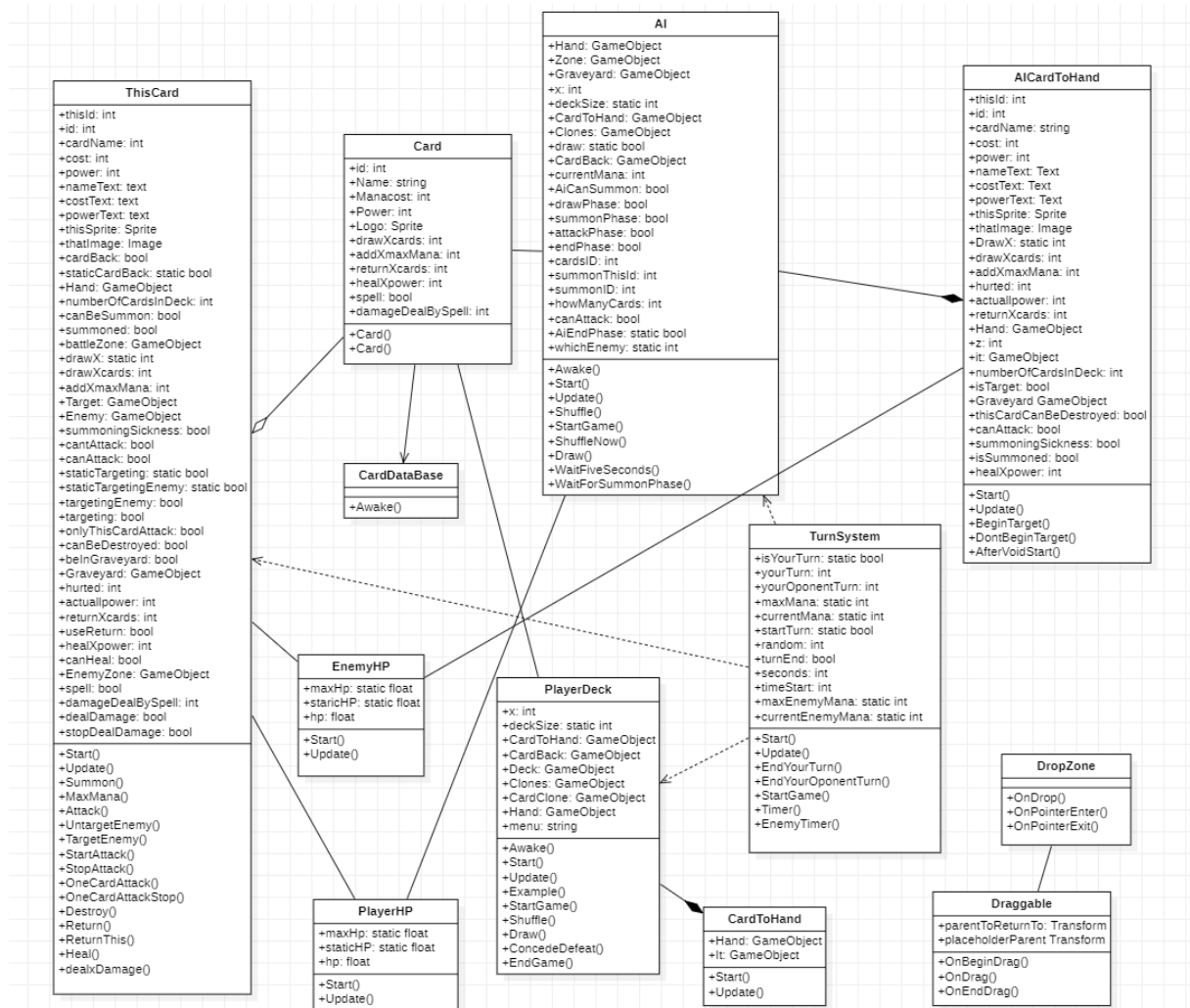


Рис.2.2. Діаграма класів на сцені бою.

Опис класів, а також їх полів та методів:

База даних усіх карт знаходиться у CardDataBase але самі поля карти ініціалізує клас Card.

ThisCard клас контролер, він відповідає за зовнішній вигляд карти та його поведінку на сцені: виклик (Summon()), атака та отримані шкоди (Attack(), TargetEnemy()) де ми звертаємось до класу EnemyHP який відповідає за збереження та візуальне відображення здоров'я ворога; знищення карти(Destroy()) і звичайно ефекти карт як наприклад повернення в руку(Return) а також окремий метод для карт заклинань (dealxDamage()).

Клас PlayerDeck обробляє масив з 30 id-карт, тасує(Shuffle()) і передає клону CardToHand який видає карту «в руки» гравцю.

TurnSystem клас, що відповідає за черговість ходів та час відведених на них, а також поповнює ману обох героїв кожний раунд.

Клас AI це інтелект ворога на сцені і створення його карт на сцені методом передачі id AICardToHand який окрім передачі карти у руки ворогу відповідає за значення булевих змінних щоб як і ми використовувати ефекти і атакувати.

Класи Draggable та DropZone відповідають за систему «Drag & Drop» де перший відповідає за поведінку карти при затисканні, а другий за місце де опускають його опускають щоб зробити дочірнім об'єкту

2.2. Дизайн документ

2.2.1. Концепт

Жанр і цільова аудиторія:

Жанр – TCG Trading Card Game

Цільова аудиторія – чоловіки та жінки від 12 до 18 років.

Основні особливості гри:

Гра буде вирізнятися серед інших унікальними картами, гарною та простою графікою.

Геймплей:

Ви з вашою колодою карт маєте подолати вашого опонента.

Платформа:

Гра буде розроблятися тільки під PC-платформу.(можливо буде мобільна версія)

Тема сеттинг

Гра розроблена в інтерактивному форматі. Тобто, від гравця залежить перемога. Кінцева мета гри – перемогти іншого героя.

Гра реалізована наступним чином: існує головний персонаж (герой) та герой-противник.

2.2.2. Геймплей

Суть ігрового процесу полягає опустити одиницю здоров'я противника до 0 шляхом завдання шкоди картці-героєм або використавши заклинання. Однак на кожну картку витрачається певна кількість очок (мана). Карта-герой має значення сили(міць), і коли вона атакує іншу картку-герой вона її знищує якщо

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	21
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

значення сили більше ніж у цілі, і навпаки карта сама знищиться і відправиться на «кладовище». Також щоб зробити гру не такою нудною у кожної карти-героя є особливі навички.

Буде розроблено карти заклинання після виконання дії вони одразу знищуються.

Герой і противник на початку мають певну одиницю здоров'я та мана, де остання поповнюється кожний хід і 5 випадкові карти з колоди які також додаються «у руку» з кожним раундом.

2.2.3. Сцена/Локація

Окрім основної сцени бою і меню, буде сцена з мазану де гравець буде поповнювати свою колекцію і сцена створення колоди яка буде використовуватися на сцені бою щось.

Перехід сцен буде такий:

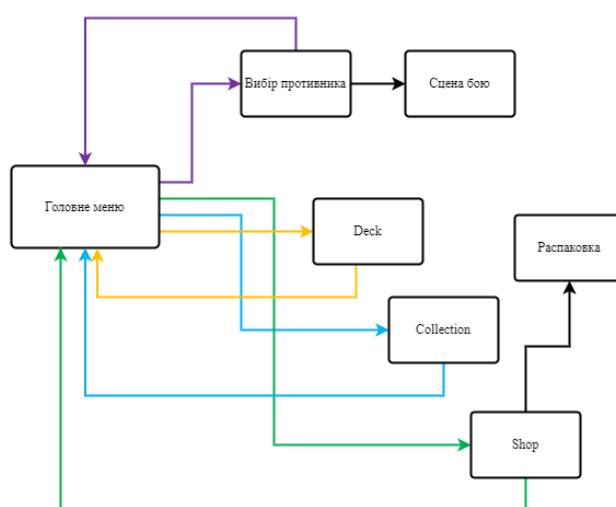


Рис.2.4. Перехід сцен

2.2.4. Текстури, персонажі та різні гейм-об'єкти

Основним гейм-об'єктом є карта (рис.2.6).

Позначки на ній:

- 1) Вартість карти;
- 2) Назви карти;
- 3) Атака карти;

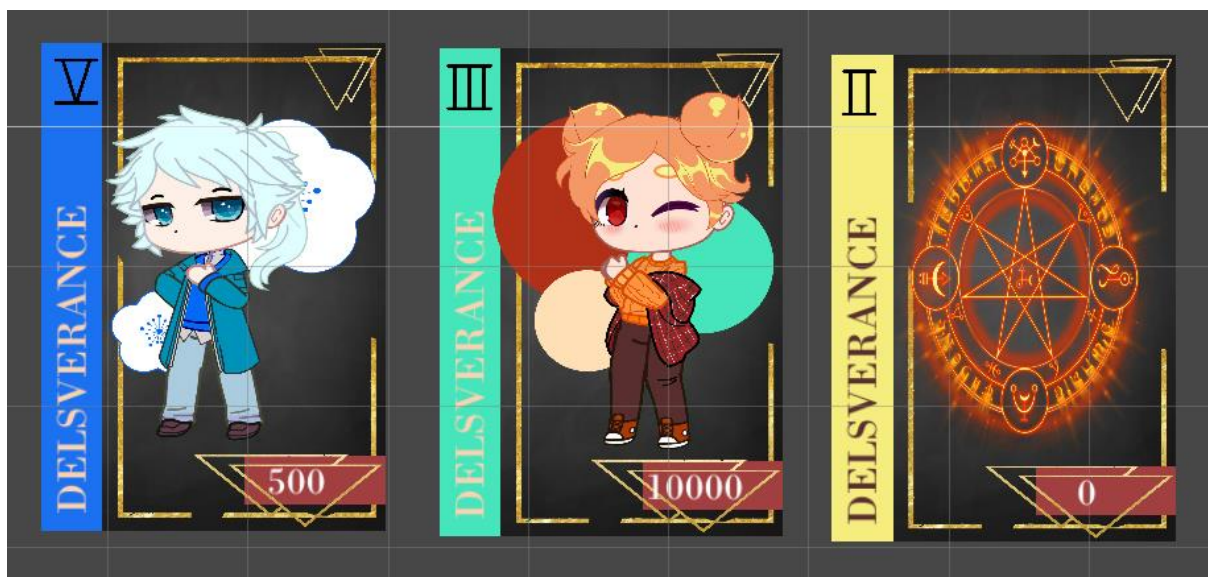


Рис.2.6 Приклад моделей карт(зліва і посередині – карта герой, з праву карта-заклинання)



Рис.2.7 Префаб карти

Графіка для гри була обрана 2D з героями у anime-chibi стилі через її зростаючу популярність, простий та приємний вигляд

2.2.5. Керування

Управління здійснюється за допомогою стрілки. Нею ми перетягуємо карти зі «своїх рук» на стіл та натискаємо на кнопки чи інтерактивні об'єкти.

2.3. Реалізація алгоритмів роботи гри

У даному розділі розглянуто частину створеного коду, що дасть основні уявлення про процес розробки. Детальніше з кодом можна ознайомитись з важливою частиною коду в додатку А.

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	23
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

Нам потрібно щоб на кожній картці були помітки для цього і був створений клас This Card, окрім візуалізації полів кожної окремої карти (Рис.2.8 №1) він міняє значення bool-х змінних (Рис.2.8 №2, №3, №4) в залежності чи ця карта знаходиться у руці супротивника чи в колоді або «кладовищі» (Рис.2.8 №2) – тоді накладається «сорочка» на карту, чи можна призвати цю карту (Рис.2.8 №3) та чи взагалі вона викликана (Рис.2.8 №4). Поле Hand (Рис.2.8 №6.1) та Battle Zone (Рис.2.8 №6.2) допомагає знайти об'єкти щоб зробити їх батьківськими елементами карти. Оскільки у нас буде колода нам потрібно буде записати «місце» у списку (Рис.2.8 №5), це значення нам допомагає додавати карти у руку не по порядку, а в випадковій послідовності.

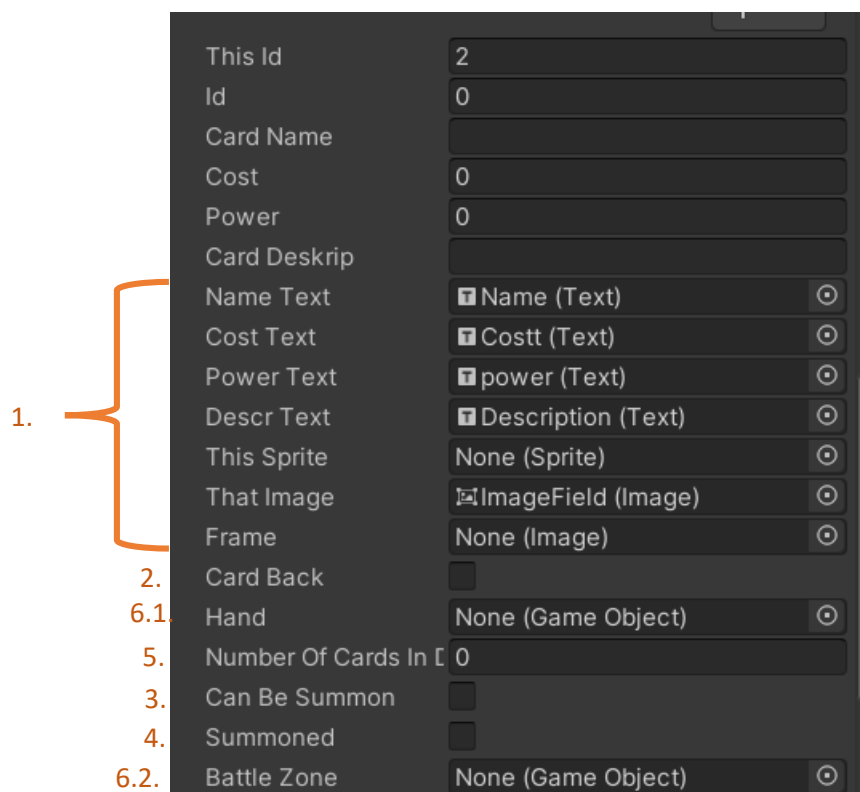


Рис.2.8 Компоненти ThisCard в інспекторі

Далі наведено реалізацію кількох важливих класів.

Спочатку було розроблено два класи Card та CardDataBase. Перший Card відповідає за ініціалізацію полів.

```
public Card(int Id, string CardName, int Cost, int Power, string CardDeskr, Sprite
ThisImage, /*string Color,*/
int DrawXcards, int AddXmaxMana, int ReturnXcards, int HealXpower, bool Spell, int
DamageDealBySpell)
{
    id = Id;
    cardName = CardName;
```

```

cost = Cost;
power = Power;
cardDeskrip = CardDeskr;

thisImage = ThisImage;

//color = Color;

drawXcards = DrawXcards;
addXmaxMana = AddXmaxMana;

returnXcards = ReturnXcards;

healXpower = HealXpower;

spell = Spell;
damageDealBySpell = DamageDealBySpell;
}

```

CardDataBase виступає в ролі бази даних всіх карт.

```

void Awake()
{
...
    cardList.Add(new Card(4, "priest", 1, 500, "heal", Resources.Load<Sprite>("14"), 0,
0, 0, 1500, false, 0));
...
    //Spell card//
    cardList.Add(new Card(7, "fireball", 4, 0, "Deal 3000 enemy or card",
Resources.Load<Sprite>("atkallcard"), 0, 0, 0, 0, true, 3000));
...
}

```

Клас ThisCard по-праву можна вважати контролером карти, окрім візуального відображення, він повідає за поведінку клону-префабу під час виклику, атаки, знищення та здібностей карт. Для прикладу візьмемо метод Attack. У ньому в залежності на кого була спрямована ціль(Target) ми або віднімемо одинцю здоров'я у противника, або перепишімо значення hurted з сили карти ворога.

```

public void Attack()
{
    if (canAttack == true && summoned==true)
    {
        if (Target != null)
        {
            if (Target == Enemy)
            {
                EnemyHP.staticHP -= power;
                targetting = false;
                cantAttack = true;
            }
        }
        else
        {
            foreach(Transform child in EnemyZone.transform)
            {
                if (child.GetComponent<AICardToHand>().isTarget == true)

```

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	25
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        {
            child.GetComponent<AICardToHand>().hurted = power;
            hurted = child.GetComponent<AICardToHand>().power;
            cantAttack = true;
        }
    }
}
}
}

```

Далі запишемо різницю сили нашої та ворога:

```
actualpower = power - hurted;
```

Та перевіримо чи ця різниця менше нуля і якщо так знищимо карту та закинемо на «кладовище»:

```

if (actualpower <= 0 && spell==false)
{
    Destroy();
}

public void Destroy()
{
    Graveyard = GameObject.Find("My Graveyard");
    canBeDestroyed = true;
    if (canBeDestroyed == true)
    {
        this.transform.SetParent(Graveyard.transform);
        canBeDestroyed = false;
        summoned = false;
        beInGraveyard = true;

        hurted = 0;

        canHeal = true;
    }
}

```

Важливим кроком було створити систему Drag & Drop щоб брати карту та класти на ігрове поле. Даний скрипт Draggeble працює завдяки трьом інтерфейсам: *IBeginDragHandler*, *IDragHandler*, *IEndDragHandler* (А також окремого скрипту *DropZone* місця куди карта кладеться). Код *IBeginDragHandler* буде виконан один раз коли ми почнемо перетягувати карту. Для реалізації перетягування карти з руки на поле була оголошена змінна *transform - DefaultParents*. В *OnBeginDrag* порівнюємо поточному батьківському елементу нашої карти, потім перемістимо нашу карту вгору по ієрархії присвоївши їй батьківський елемент його дочірнього об'єкту. А в *OnDrag* ми присвоїмо карті батьківський елемент який зберігається в *DefaultParents*.

```

public void OnBeginDrag(PointerEventData eventData)
{
    offset = transform.position - MainCamera.ScreenToWorldPoint(eventData.position);
}

```

```

DefaultParent = DefaultTempCardParent = transform.parent;

//істина працює лише тоді коли ми перетаскуємо карту з SELF_HAND
//для зручності вимкнимо карти які не можемо використати (карта яка неможна призвати
бо не вистачає мани або карта якою вже походили)
IsDraggable = GameManagerScr.Instance.IsPlayerTurn &&
(
    (DefaultParent.GetComponent<DropPlaceScr>().Type == FieldType.SELF_HAND &&
    GameManagerScr.Instance.CurrentGame.Player.Mana >= CC.Card.Manacost) ||
    (DefaultParent.GetComponent<DropPlaceScr>().Type == FieldType.SELF_FIELD &&
    CC.Card.CanAttack)
);

if (!IsDraggable) // <-- якщо false функція просто приривається
    return;
//для того щоб не можна було міняти карти в руці місцями
startID = transform.GetSiblingIndex();

//щоб цілі не підсвічувувалися коли вибираєш карти з руки
if (CC.Card.IsSpell || CC.Card.CanAttack)
{
    GameManagerScr.Instance.HighlightTargets(CC, true);
}

TempCardGO.transform.SetParent(DefaultParent);
// позиція яка приймає індекс поточної карти
TempCardGO.transform.SetSiblingIndex(transform.GetSiblingIndex());

transform.SetParent(DefaultParent.parent);
// Canv*asGroup for card cliced
GetComponent<CanvasGroup>().blocksRaycasts = false;
}

```

Код IDragHandler буде тривати кожен кадр поки ми тягнемо об'єкт вектору newPos задаємо поточні координати миші які зберігаються в IeventDataPosition ,але ці координати потрібно пропустити через ф-цію ScreenToWorldPoint так як координати миші визначаються плоскістю екрану а нам потрібно ці значення перевести в плоскість WorldCordinat.

```

public void OnDrag(PointerEventData eventData)
{
    if (!IsDraggable) // <-- якщо false функція просто приривається
        return;

    Vector3 newPos = MainCamera.ScreenToWorldPoint(eventData.position);
    transform.position = newPos + offset;

    if (!CC.Card.IsSpell)
    {
        if (TempCardGO.transform.parent != DefaultTempCardParent)
            TempCardGO.transform.SetParent(DefaultTempCardParent);

        // вимикаємо можливість гравцю міняти місцями карти викладані на столі. Для
        цього в CardMovementScr OneDrag викликаємо CheckPosition тільки якщо карта
        // знаходиться НЕ НА ПОЛЕ
        if (DefaultParent.GetComponent<DropPlaceScr>().Type != FieldType.SELF_FIELD)
        {
            CheckPosition();
        }
    }
}

```

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	27
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }
}
}
Код IEndDragHandler виконається коли ми відпустимо об'єкт
public void OnEndDrag(PointerEventData eventData)
{
    if (!IsDraggable) // <-- якщо false функція просто приривається
        return;

    GameManagerScr.Instance.HighlightTargets(CC, false);

    transform.SetParent(DefaultParent);
    // CanvasGroup for card cliced
    GetComponent<CanvasGroup>().blocksRaycasts = true;

    transform.SetSiblingIndex(TempCardGO.transform.GetSiblingIndex());
    TempCardGO.transform.SetParent(GameObject.Find("Canvas").transform);
    TempCardGO.transform.localPosition = new Vector3(2584, 0);
}

```

Для того щоб у нас була можливість створити власну колоду, зберегти та перенести в ігрову сцену ми використаємо PlayerPrefs це вбудований клас Unity який зберігає налаштування гравця між ігровими сесіями. Він може зберігати рядкові, плаваючі та цілі числа в реєстрі платформи користувача. Ми реєструємо кожну карту з масиву доступних з бази даних якщо в сумі набирається 30 ми їх нумеруємо та записуємо в масив saveDeck

```

public void CreateDeck()
{
    for(int i=0; i< numberOfCardsInDatabase; i++)
    {
        sum += cardsWithThisId[i];
    }

    if (sum == 30) //Deck
    {
        for(int i=0; i <= numberOfCardsInDatabase; i++)
        {
            PlayerPrefs.SetInt("deck" + i, cardsWithThisId[i]);
        }
    }

    sum = 0;
    numberOfDifferentCards = 0;

    for(int i=0; i <= numberOfCardsInDatabase; i++)
    {
        saveDeck[i] = PlayerPrefs.GetInt("deck" + i, 0);
    }
}

```

Висновки до другого розділу

Проведено розробу програмного забезпечення, яке створене на основі дизайн документу та технічного завдання.

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	28
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

Описано архітектуру за допомогою діаграми варіантів використання та діаграми класів.

На наступному етапі створення гри був розроблений програмний код та розглянуто деякі компоненти з нього.

Також було розглянуто деякі алгоритми гри разом із програмним кодом.

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	29
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ

3.1. Порядок встановлення та налаштування системи

Білд гри був зроблений на Unity під Windows та Linux , але щоб він працював правильно потрібно перевірити чи всі сцени завантажені і знаходяться в правильному порядку(рис.3.1), сцена з меню має бути першою:

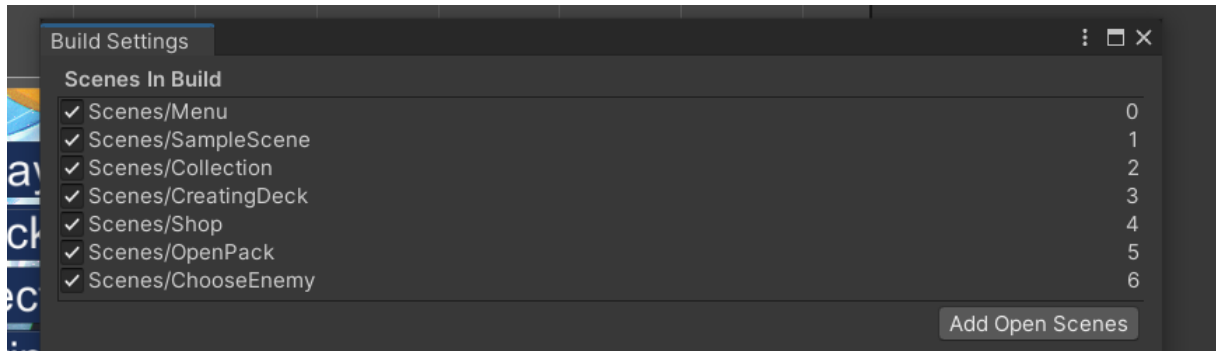


Рис.3.1 Порядок завантаження сцен

3.2. Інтерфейс та порядок роботи з програмним кодом

Після початкового запуску гри, програється інтро (рис.3.2.1) і після натискання користувач опиняється у головному меню(рис .3.2.2)

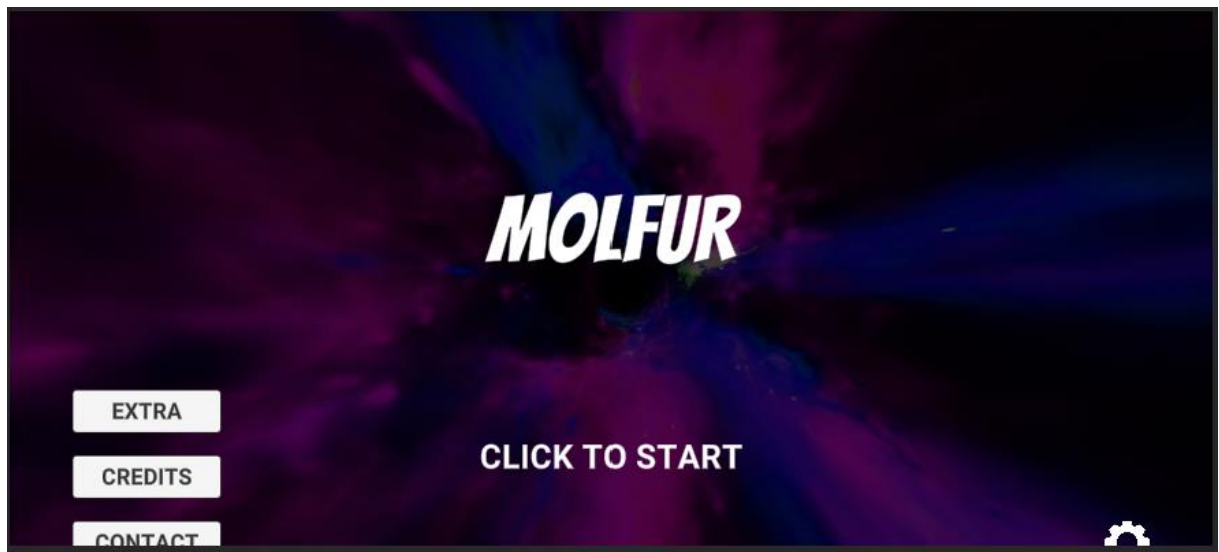


Рис. 3.2. Intro Scene

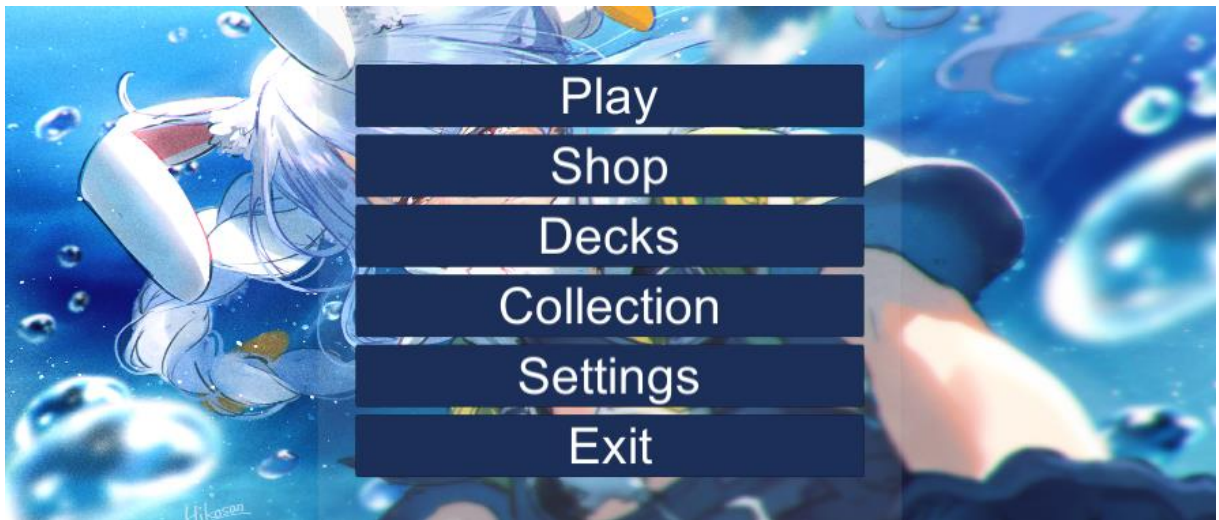


Рис. 3.3. Головне меню

Після запуску «Play» та вибору противника одразу запускається гра(рис.3.4), вам видають 4 карти, початкову к-сть мани 3 і початкова позначку здоров'я 20.

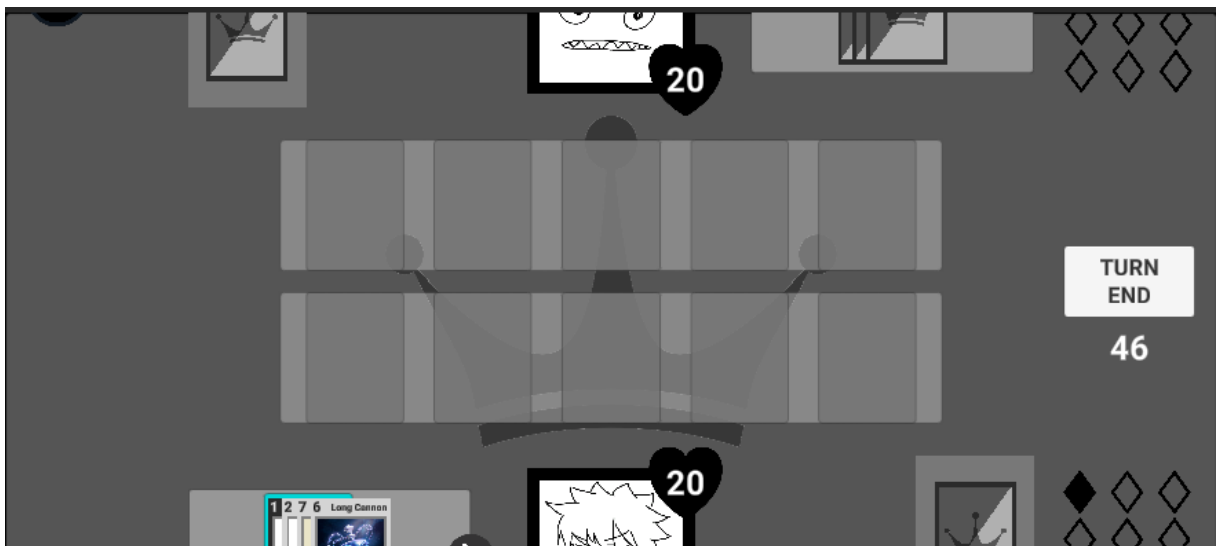


Рис.3.4 Сцена бою

Якщо в тому ж головному меню обрати «Deck» запуститься сцена(рис.3.4) де ми гортаємо нашу колекцію а щоб створити колоди ми перетягуємо карти на панель і коли карт буде 30 ми зможемо зберегти її.

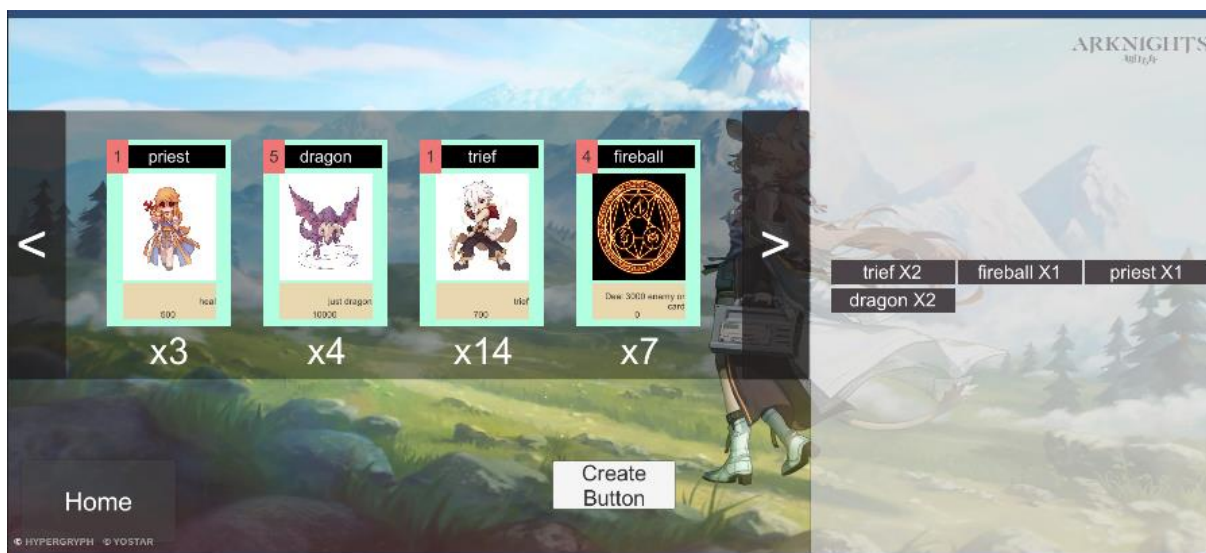


Рис.3.5 Сцена створення колоди

В сцені «Shop» (рис.3.6) ми можемо придбати блістер з 5 карт, після натискання кнопки «Buy Pack» запускається сцена «розпаковки» (рис.3.7) де вам випадає 5 випадкових карт «Сорочкою» догори і при натисканні на карту вона забирається. Після того як будуть розкриті всі карти ми повернемося назад магазин.

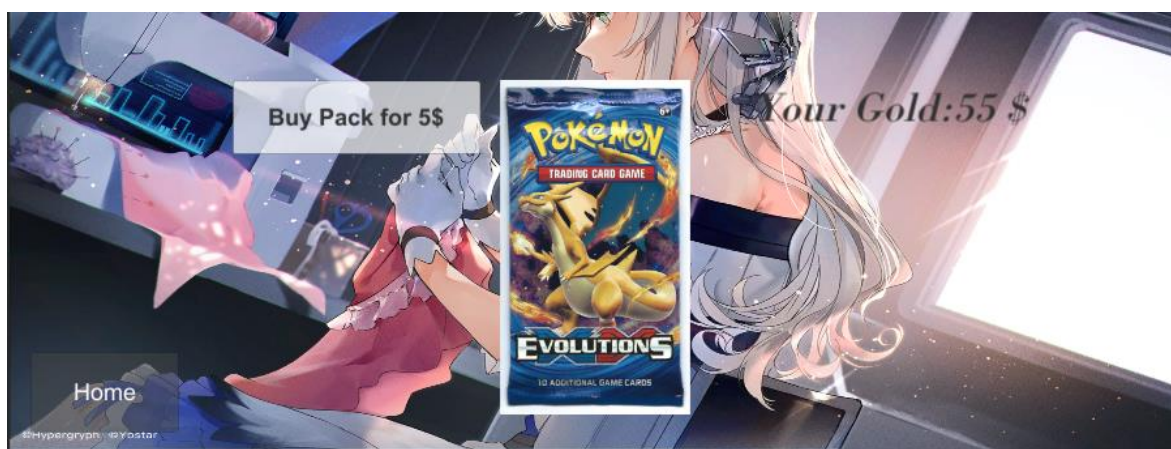


Рис.3.6 Сцена магазину

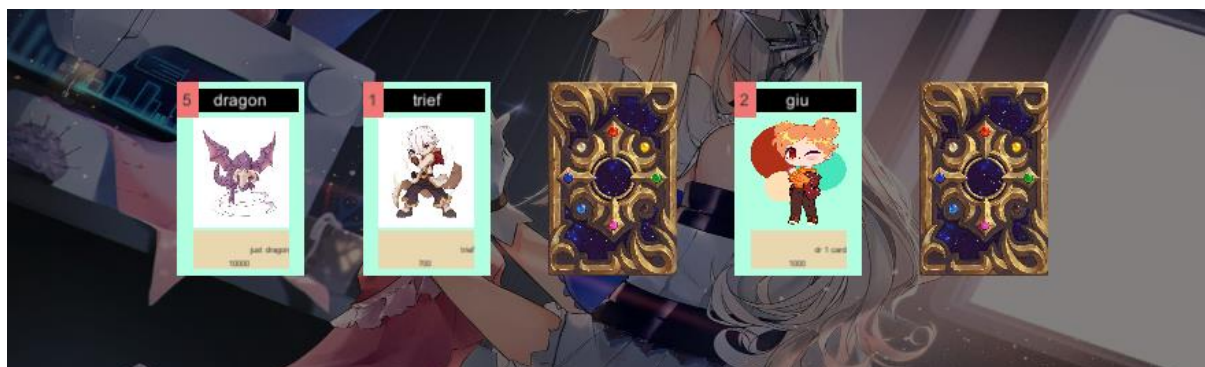


Рис.3.7 Сцена «розпаковки»

Отже, інтерфейс має всі необхідні елементи. Він достатньо зрозумілий та не має надмірного навантаження на користувача.

Проведено тестування гри для виявлення та усунення недоліків.

Таблиця 1.2

Сцена	Вхідні дані	Очікувані дані	Спостережені дані
Game	Натискаєш та затискаєш лівою кнопкою миші на карту	Карта слідує за стрілкою	Карта слідує за стрілкою
	Опускаєш карту на ігрове поле	Вона там залишається при умові що у вас достатньо мани для неї	Карта залишається на ігровому полі, але якщо ви не попадаєте по заготовці вона повертається у руку
	Натиск на кнопку «End Turn»	Хід завершується, таймер повертається до 50 сек	Хід завершується, таймер повертається до 50 сек
	Натиск на іконку білого флагу	З'являється вікно де нас запитують чи хочемо ми здатися і якщо так ви програєте	З'являється вікно де нас запитують чи хочемо ми здатися і якщо так то з'являється інше вікно «You Lose» де нам

			пропонують почати заново чи повернутись назад до головного меню
	Переніс карт на «кладовище»	Коли карти знищуються вона відправляється на «кладовище» і ми можемо переглянути його переглянути в спливаючому вікні	Карта зникає з поля і натиснувши на іконку «кладовище» ми можемо її побачити в спливаючому вікні
	Карта-воскресіння	Коли ми викликаємо карту з'являється вікно з зниженими картами натиснувши на любу з них ми повертаємо карту в руку	Коли ми викликаємо карту з'являється вікно з зниженими картами натиснувши на любу з них ми повертаємо карту в руку
Create Deck	Перетягування карт на панель	На панелі має з'явитись іконка з назвою карти і кількістю	На панелі з'являється іконка з назвою карти та цифрою що показує скільки таких копій в колоді

Create Deck/Game	Кнопка «створити колоду» + «почати гру»	Колода зберігається та переноситься в ігрову сцену	Колода зберігається та переноситься в ігрову сцену
Shop	Кнопка «Buy Pack»	Нас перекидає на сцену «Розпаковки» де нам дають 5 випадкових карт і повертають на сцену «Shop»	Нас перекидає на сцену «Розпаковки» де нам дають 5 випадкових карт і повертають на сцену «Shop» де з нашого рахунку знімають 5 умовної валюти

При тестуванні були виявлені помилки, незручності та недоліки гри. Було наприклад багато помилок зв'язаний з картами заклинання коли карті потрібно було мати ціль, а ви вибирали карту з руки то видавало помилку з нульовим значенням таргету. Також карти, що не мали таргету не спрацьовували оскільки вважали що ви намагаєтесь викласти їх на поле. Методи було пофікшено методом додання bool-х перемінних які перевіряли поточне положення карти і у випадку false просто виходити з ф-ції

Висновки до третього розділу:

У третьому розділі детально розглянуто та описано інтерфейс користувача у грі. Наведено приклади скріншотів з додатку та розказано, які дії може виконати гравець при певних обставинах.

Описано процес тестування: які дії повинні до чого призвести, та що відбувалося насправді. Виявлено деякі помилки, неточності та недоліки. Їх було проаналізовано та виправлено.

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	35
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Було досягнуто мети та створено працюючий ігровий додаток у жанрі головоломка.

Переглянуто три популярні гри жанрів, у яких було заплановано проект, визначено їх переваги та недоліки. Проаналізовано потреби гравця та поставлено задачу проекту, а саме який в загальному має бути функціонал.

Створено концепт-документ, в якому визначено жанр та аудиторію, особливості, геймплей, сюжет та платформу.

Складено дизайн-документ, у якому продумано мету гри, механіки, представлено основну графіку.

Описано архітектуру за допомогою діаграми варіантів використання та класів.

Розписано головні етапи реалізації гри з описом найбільш вагомих або складних за виконанням методів коду.

Описано можливості гравця у грі. Наведено приклади скриншотів з додатку та розказано, які дії можна виконати при яких обставинах.

Описано процес тестування: які дії повинні до чого призвести, та що відбувалося насправді.

В додатках приведено лістинг важливих функцій.

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	36
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Робота з анімаціями у Unity 2D [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=FDZxe7j6Zlc>
2. Робота зі спрайтами [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=YSMrQeBf8CM>
3. Як зробити можливим вибір мови та локалізувати увесь текст у грі [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=rLXyXUsBYZ0&list=PLdE10202Nk1MXUa_aXOTP4catRFq7prSRW&index=11
4. Меню налаштувань в Unity [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=YOaYQrN1oYQ&list=PLdE10202Nk1MXUa_aXOTP4catRFq7prSRW&index=17
5. Основи роботи зі Scriptable Objects у Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=aPXvoWVabPY>
6. Основи роботи зі SceneManager у Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=3I5d2rUJ0pE>
7. Довідник Unity а саме клас PlayerPrefs [Електронний ресурс] – Режим доступу до ресурсу <https://docs.unity3d.com/ScriptReference/PlayerPrefs.html>
8. Створення анімації кнопок та ефектів[Електронний ресурс] – Режим доступу до ресурсу <https://www.bilibili.com/video/BV1yR4y1b7NV?p=3>
9. Unity - scripting API: animator. Unity - Manual: Unity User Manual 2022.3 (LTS). URL: <https://docs.unity3d.com/ScriptReference/Animator.html>.
10. Best practices for organizing your Unity project | Unity. Unity. URL: <https://unity.com/how-to/organizing-your-project>.
11. Система Drag & Drop [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=AM7wBz9azyU>
12. Хід противника [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=pkV2fXYgbps&list=PL0oQ0JTWjALQGkiDWw_ws21fanM2za02B&index=40

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	37
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		

13. Інтелект противника [Електронний ресурс] – Режим доступу до ресурсу:
https://www.youtube.com/watch?v=cfXDwZ7PAQk&list=PLOoQ0JTWjALQGkiDWw_ws21fanM2za02B&index=32
14. Система здібностей карт і взагалі як робиться карткова гра [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.youtube.com/watch?v=IBjYKq6i45w>
15. Використання векторів у Unity [Електронний ресурс] – Режим доступу до ресурсу:
https://www.youtube.com/watch?v=wXI9_olSrqr&list=PLdE10202Nk1MXUaaXOTP4catRFq7prSRW&index=27
16. Як тестувати гру [Електронний ресурс] – Режим доступу до ресурсу:
https://www.youtube.com/watch?v=b9U1dcUS_-w&list=PLdE10202Nk1MXUaaXOTP4catRFq7prSRW&index=31

ДОДАТКИ

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		39

		Ташкінова О.О.			КН.КР.Б – 122 – 23 – ПЗ	
		Артамонов Є. Б				
Зм.	Арк.	№ докум.	Підпис	Дата		40

Файл ThisCard.cs:

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

public class ThisCard : MonoBehaviour
{
    public List<Card> thisCard = new List<Card>();
    public int thisId;

    public int id;
    public string cardName;
    public int cost;
    public int power;
    public string cardDeskrip;

    public Text nameText;
    public Text costText;
    public Text powerText;
    public Text descrText;

    public Sprite thisSprite;
    public Image thatImage;

    public Image frame; // цвет карти, пока нет

    public bool cardBack;
    public static bool staticCardBack;

    public GameObject Hand;

    public int numberOfCardsInDeck;

    public bool canBeSummon;
    public bool summoned;
    public GameObject battleZone;

    public static int drawX;
    public int drawXcards;
    public int addXmaxMana;

    public GameObject attackBorder; //<-----

    public GameObject Target;
    public GameObject Enemy;

    public bool summoningSickness;

    public bool cantAttack;
    public bool canAttack;

    public static bool staticTargeting;
    public static bool staticTargetingEnemy;

    public bool targeting;
    public bool targetingEnemy;
```

```

public bool onlyThisCardAttack;

public GameObject summonBorder;

public bool canBeDestroyed;
public GameObject Graveyard;
public bool beInGraveyard;

public int hurted;
public int actualpower;
public int returnXcards;
public bool useReturn;

public static bool UcanReturn;

public int healXpower;
public bool canHeal;

public GameObject EnemyZone;
public AICardToHand aiCardToHand;

public bool spell;
public int damageDealBySpell;
public bool dealDamage;
public bool stopDealDamage;

// Start is called before the first frame update
void Start()
{
    thisCard[0] = CardDataBase.cardList[thisId];
    numberOfCardsInDeck = PlayerDeck.deckSize;

    canBeSummon = false;
    summoned = false;

    //card in deck
    if (this.tag != "Deck")
    {
        if (TurnSystem.currentMana >= cost && false == summoned)
        {
            canBeSummon = true;
        }
        else
        {
            canBeSummon = false;
        }

        if (true == canBeSummon)
        {
            gameObject.GetComponent<Draggable>().enabled = true;
        }
        else
        {
            gameObject.GetComponent<Draggable>().enabled = false;
        }

        battleZone = GameObject.Find("Zone");

        if (false == summoned && this.transform.parent == battleZone.transform)
        {
            Summon();
        }
    }
}

```

```

    }

    drawX = 0;

    canAttack = false;
    summoningSickness = true;

    Enemy = GameObject.Find("HpEnemy");

    targetting = false;
    targettingEnemy = false;

    canHeal = true;

    EnemyZone = GameObject.Find("MyEnemyZone");
}

// Update is called once per frame
void Update()
{
    Hand = GameObject.Find("Hand");
    if(this.transform.parent == Hand.transform.parent)
    {
        cardBack = false;
    }

    id = thisCard[0].id;
    cardName = thisCard[0].cardName;
    cost = thisCard[0].cost;
    power = thisCard[0].power;
    cardDeskrip = thisCard[0].cardDeskrip;

    thisSprite = thisCard[0].thisImage;

    drawXcards = thisCard[0].drawXcards;
    addXmaxMana = thisCard[0].addXmaxMana;

    returnXcards = thisCard[0].returnXcards;

    healXpower = thisCard[0].healXpower;

    spell = thisCard[0].spell;
    damageDealBySpell=thisCard[0].damageDealBySpell;

    nameText.text = "" + cardName;
    costText.text = "" + cost;

    actuallpower = power - hurted;
    powerText.text = "" + actuallpower;
    descrText.text = "" + cardDeskrip;

    thatImage.sprite = thisSprite;

    // color here

    staticCardBack = cardBack;

    if (this.tag == "Clone")
    {
        thisCard[0] = PlayerDeck.staticDeck[numberOfCardsInDeck - 1];
        numberOfCardsInDeck -= 1;
        PlayerDeck.deckSize -= 1;
    }
}

```

```

        cardBack = false;
        this.tag = "Untagged";
    }

    if (TurnSystem.currentMana >= cost && summoned == false && beInGraveyard ==
false && TurnSystem.isYourTurn == true)
    {
        canBeSummon = true;
    }
    else canBeSummon = false;

    if (canBeSummon == true)
    {
        gameObject.GetComponent<Draggable>().enabled = true;
    }
    else gameObject.GetComponent<Draggable>().enabled = false;

    battleZone = GameObject.Find("Zone");

    if(summoned==false && this.transform.parent == battleZone.transform)
    {
        Summon();
    }

    if (canAttack == true && beInGraveyard==false)
    {
        attackBorder.SetActive(true);

        Debug.Log("Рамка атаки активировалася");
    }
    else
    {
        attackBorder.SetActive(false);
        Debug.Log("Рамка атаки активировалася");
    }

    if(TurnSystem.isYourTurn==false && summoned == true)
    {
        summoningSickness = false;
        cantAttack = false;
    }

    if(TurnSystem.isYourTurn == true && summoningSickness==false && cantAttack ==
false)
    {
        canAttack = true;
    }
    else
    {
        canAttack = false;
    }

    targeting = staticTargeting;
    targetingEnemy = staticTargetingEnemy;

    if (targetingEnemy == true)
    {
        Target = Enemy;
    }else
    {
        Target = null;
    }

```

```

    }

    if(targeting==true && onlyThisCardAttack == true)
    {
        Attack();
    }

    if (canBeSummon == true || UcanReturn == true && beInGraveyard == true)
    {
        summonBorder.SetActive(true);
    }
    else
    {
        summonBorder.SetActive(false);
    }

    if (actualpower <= 0 && spell==false)
    {
        Destroy();
    }

    if(returnXcards > 0 && summoned==true && useReturn == false &&
TurnSystem.isYourTurn == true)
    {
        Return(returnXcards);
        useReturn = true;
    }

    if (TurnSystem.isYourTurn == false)
    {
        UcanReturn = false;
    }

    if (canHeal == true && summoned == true)
    {
        Heal();
        canHeal = false;
    }

    if (damageDealBySpell > 0)
    {
        dealDamage = true;
    }

    if (dealDamage == true && this.transform.parent == battleZone.transform)
    {
        attackBorder.SetActive(true);
    }
    else
    {
        attackBorder.SetActive(false);
    }

    if(dealDamage==true && this.transform.parent == battleZone.transform)
    {
        dealxDamage(damageDealBySpell);
    }

    if (stopDealDamage == true)
    {
        attackBorder.SetActive(false);
        dealDamage = false;
    }

```

```

    }

    if(this.transform.parent==battleZone.transform && spell==true && dealDamage ==
false)
    {
        Destroy();
    }
}

public void Summon()
{
    TurnSystem.currentMana -= cost;
    summoned = true;

    MaxMana(addXmaxMana);
    drawX = drawXcards;
}

public void MaxMana(int x)
{
    TurnSystem.maxMana += x;
}

public void Attack()
{
    if (canAttack == true && summoned==true)
    {
        if (Target != null)
        {
            if (Target == Enemy)
            {
                EnemyHP.staticHP -= power;
                targeting = false;
                cantAttack = true;
            }
        }
        else
        {
            foreach(Transform child in EnemyZone.transform)
            {
                if (child.GetComponent<AICardToHand>().isTarget == true)
                {
                    child.GetComponent<AICardToHand>().hurted = power;
                    hurted = child.GetComponent<AICardToHand>().power;
                    cantAttack = true;
                }
            }
        }
    }
}

public void UntargetEnemy()
{
    staticTargetingEnemy = false;
}

public void TargetEnemy()
{
    staticTargetingEnemy = true;
}

```

```

public void StartAttack()
{
    staticTargeting = true;
}

public void StopAttack()
{
    staticTargeting = false;
}

public void OneCardAttack()
{
    onlyThisCardAttack = true;
}

public void OneCardAttackStop()
{
    onlyThisCardAttack = false;
}

public void Destroy()
{
    Graveyard = GameObject.Find("My Graveyard");
    canBeDestroyed = true;
    if (canBeDestroyed == true)
    {
        this.transform.SetParent(Graveyard.transform);
        canBeDestroyed = false;
        summoned = false;
        beInGraveyard = true;

        hurted = 0;

        canHeal = true;
    }
}

public void Return(int x)
{
    for(int i=0; i<=x; i++)
    {
        ReturnCard();
    }
}

public void ReturnCard()
{
    UcanReturn = true;
}

public void ReturnThis()
{
    if(beInGraveyard==true && UcanReturn == true)
    {
        this.transform.SetParent(Hand.transform);
        UcanReturn = false;
        beInGraveyard = false;
        summoningSickness = true;
    }
}

```

```

public void Heal()
{
    PlayerHP.staticHP += healXpower;
}

public void dealxDamage(int x)
{
    if (Target != null)
    {
        if (Target == Enemy && stopDealDamage == false && Input.GetMouseButton(0))
        {
            EnemyHP.staticHP -= damageDealBySpell;
            stopDealDamage = true;
        }
    }
    else
    {
        foreach (Transform child in EnemyZone.transform)
        {
            if (child.GetComponent<AICardToHand>().isTarget == true &&
Input.GetMouseButton(0))
            {
                child.GetComponent<AICardToHand>().hurted += damageDealBySpell;
                stopDealDamage = true;
            }
        }
    }
}
}

```

Файл PlayerDeck.cs:

```

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

public class PlayerDeck : MonoBehaviour
{
    public List<Card> deck = new List<Card>();
    public List<Card> container = new List<Card>();
    public static List<Card> staticDeck = new List<Card>();

    public int x;
    public static int deckSize;

    public GameObject cardInDeck1;
    public GameObject cardInDeck2;
    public GameObject cardInDeck3;

    public GameObject CardToHand;

    public GameObject CardBack;
    public GameObject Deck;

    public GameObject[] Clones;

    private GameObject CardClone;

```

```

public GameObject Hand;

public Text LoseText;
public GameObject LoseTextGame;

public GameObject concededWindow;
public string menu = "Menu";

void Awake()
{
    Shuffle();
}

// Start is called before the first frame update
void Start()
{
    x = 0;
    deckSize = 30;

    //for(int i = 0; i< deckSize; i++)
    //{
    //    x = Random.Range(0, 7); //зависит от количества в базе
    //    deck[i] = CardDataBase.cardList[x];
    //}

    for(int i = 1; i <= 8; i++)
    {
        if(PlayerPrefs.GetInt("deck" + i, 0) > 0)
        {
            for (int j = 1; j <= PlayerPrefs.GetInt("deck" + i, 0); j++)
            {
                deck[x] = CardDataBase.cardList[i];
                x++;
            }
        }
    }

    Shuffle();

    StartCoroutine(StartGame());
}

// Update is called once per frame
void Update()
{
    if (deckSize <= 0)
    {
        LoseTextGame.SetActive(true);
        LoseText.text = "U Lose";
    }

    staticDeck = deck;

    if(deckSize<20)
    {
        cardInDeck1.SetActive(false);
    }
    if (deckSize < 10)
    {
        cardInDeck2.SetActive(false);
    }
    if (deckSize < 2)

```

```

    {
        cardInDeck3.SetActive(false);
    }

    if (ThisCard.drawX > 0)
    {
        StartCoroutine(Draw(ThisCard.drawX));
        ThisCard.drawX = 0;
    }

    if (TurnSystem.startTurn == true)
    {
        if (CarsInHand.howMany < 7)
        {
            StartCoroutine(Draw(1));
        }
        else
        {
            // ne vutygivat krome clad
        }

        TurnSystem.startTurn = false;
    }
}

IEnumerator Example()
{
    yield return new WaitForSeconds(1);
    Clones = GameObject.FindGameObjectsWithTag("Clone");

    foreach (GameObject Clone in Clones)
    {
        Destroy(Clone);
    }
}

IEnumerator StartGame()
{
    for(int i = 0; i <=4; i++)
    {
        yield return new WaitForSeconds(1);
        Instantiate(CardToHand, transform.position, transform.rotation);
    }
}

public void Shuffle()
{
    for (int i = 0; i < deckSize; i++)
    {
        {
            container[0] = deck[i];
            int randomIndex = Random.Range(i, deckSize);
            deck[i] = deck[randomIndex];
            deck[randomIndex] = container[0];
        }
        CardClone = (GameObject)Instantiate(CardBack, transform.position,
transform.rotation);
        CardClone.tag = "Clone";
        StartCoroutine(Example());
    }
}

IEnumerator Draw(int x)

```

```

{
    for(int i = 0; i < x; i++)
    {
        yield return new WaitForSeconds(1);
        Instantiate(CardToHand, transform.position, transform.rotation);
    }
}

public void OpenWindow()
{
    concedeWindow.SetActive(true);
}

public void CloseWindow()
{
    concedeWindow.SetActive(false);
}

public void ConcedeDefeat()
{
    StartCoroutine(EndGame());
}

IEnumerator EndGame()
{
    LoseTextGame.SetActive(true);
    LoseText.text = "U Lose";
    concedeWindow.SetActive(false);

    yield return new WaitForSeconds(2.5f);

    SceneManager.LoadScene(menu);
}
}

```

Файл AI.cs:

```

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

public class AI : MonoBehaviour
{
    public List<Card> deck = new List<Card>();
    public List<Card> container = new List<Card>();
    public static List<Card> staticEnemyDeck = new List<Card>();

    public List<Card> cardsInHand = new List<Card>();
    //public bool AICanPlay;
    public List<Card> cardsInZone = new List<Card>();

    public GameObject Hand;
    public GameObject Zone;
    public GameObject Graveyard;

    public int x;
    public static int deckSize;

    public GameObject cardInDeck1;
    public GameObject cardInDeck2;
}

```

```

public GameObject cardInDeck3;

public GameObject CardToHand;

public GameObject[] Clones;

public static bool draw;

public GameObject CardBack;

public int currentMana;

public bool[] AiCanSummon;

public bool drawPhase;
public bool summonPhase;
public bool attackPhase;
public bool endPhase;

public int[] cardsID;

public int summonThisId;

public AICardToHand aiCardToHand;
public int summonID;
public int howManyCards;

public bool[] canAttack;
public static bool AiEndPhase;

public static int whichEnemy;

void Awake()
{
    Shuffle();
}
// Start is called before the first frame update
void Start()
{
    StartCoroutine(WaitFiveSeconds());

    Hand = GameObject.Find("EnemyHand");
    Zone = GameObject.Find("MyEnemyZone");
    Graveyard = GameObject.Find("Enemy Gravyard");

    x = 0;
    deckSize = 30;
    draw = true;

    //for(int i=0; i < deckSize; i++)
    //{
    //    x = Random.Range(3, 7);
    //    deck[i] = CardDataBase.cardList[x];
    //}
    //enemy 1
    if (whichEnemy == 1)
    {
        for(int i = 0; i < deckSize; i++)
        {
            if (i < 9)
            {
                deck[i] = CardDataBase.cardList[6];
            }
        }
    }
}

```

```

        }
        if (i <= 19 && i > 9)
        {
            deck[i] = CardDataBase.cardList[2];
        }
        else
        {
            deck[i] = CardDataBase.cardList[3];
        }
    }
}
//enemy 2
if (whichEnemy == 2)
{
    for (int i = 0; i < deckSize; i++)
    {
        if (i <= 19)
        {
            deck[i] = CardDataBase.cardList[5];
        }
        else
        {
            deck[i] = CardDataBase.cardList[4];
        }
    }
}
Shuffle();
StartCoroutine(StartGame());
}

// Update is called once per frame
void Update()
{
    staticEnemyDeck = deck;

    if (deckSize < 20)
    {
        cardInDeck1.SetActive(false);
    }
    if (deckSize < 10)
    {
        cardInDeck2.SetActive(false);
    }
    if (deckSize < 2)
    {
        cardInDeck3.SetActive(false);
    }

    if(AICardToHand.DrawX > 0)
    {
        StartCoroutine(Draw(AICardToHand.DrawX));
        AICardToHand.DrawX = 0;
    }

    if (TurnSystem.startTurn == false && draw == false)
    {
        StartCoroutine(Draw(1));
        draw = true;
    }

    currentMana = TurnSystem.currentEnemyMana;
}

```

```

if (0 == 0)
{
    int j = 0;
    howManyCards = 0;
    foreach(Transform child in Hand.transform)
    {
        howManyCards++;
    }
    foreach(Transform child in Hand.transform)
    {
        cardsInHand[j] = child.GetComponent<AICardToHand>().thisCard[0];
        j++;
    }

    for(int i = 0; i < 30; i++)
    {
        if(i >= howManyCards)
        {
            cardsInHand[i] = CardDataBase.cardList[0];
        }
    }
    j = 0;
}

if (TurnSystem.isYourTurn == false)
{
    for(int i = 0; i < 30; i++)
    {
        if (cardsInHand[i].id != 0)
        {
            if(currentMana >= cardsInHand[i].cost)
            {
                AiCanSummon[i] = true;
            }
        }
    }
}
else
{
    for(int i = 0; i < 30; i++)
    {
        AiCanSummon[i] = false;
    }
}

if (TurnSystem.isYourTurn == false)
{
    drawPhase = true;
}

if(drawPhase == true && summonPhase == false && attackPhase == false)
{
    StartCoroutine(WaitForSummonPhase());
}

if (TurnSystem.isYourTurn == true)
{
    drawPhase = false;
    summonPhase = false;
    attackPhase = false;
    endPhase = false;
}

```

```

    }

    //hete

    if (summonPhase == true)
    {
        summonID = 0;
        summonThisId = 0;

        int index = 0;
        for(int i = 0; i < 30; i++)
        {
            if (AiCanSummon[i] == true)
            {
                cardsID[index] = cardsInHand[i].id;
                index++;
            }
        }
        //фаза передачі

        for(int i=0; i < 30; i++)
        {
            if (cardsID[i] != 0)
            {
                if (cardsID[i] > summonID)
                {
                    summonID = cardsID[i];
                }
            }
        }
        //конец фази

        summonThisId = summonID;
        foreach(Transform child in Hand.transform)
        {
            if(child.GetComponent<AICardToHand>().id==summonThisId &&
            CardDataBase.cardList[summonThisId].cost <= currentMana)
            {
                child.transform.SetParent(Zone.transform);
                TurnSystem.currentEnemyMana -=
            CardDataBase.cardList[summonThisId].cost;
                break;
            }
        }

        summonPhase = false;
        attackPhase = true;
    }

    if(0 == 0)
    {
        int k = 0;
        int howManyCards2 = 0;
        foreach (Transform child in Zone.transform)
        {
            howManyCards2++;
        }
        foreach (Transform child in Zone.transform)
        {
            canAttack[k] = child.GetComponent<AICardToHand>().canAttack;
            k++;
        }
    }

```

```

        for (int i = 0; i < 30; i++)
        {
            if (i >= howManyCards2)
            {
                canAttack[i] = false;
            }
        }
        k = 0;
    }

    if(0 == 0)
    {
        int l = 0;
        int howManyCards3 = 0;
        foreach (Transform child in Zone.transform)
        {
            howManyCards3++;
        }
        foreach (Transform child in Zone.transform)
        {
            cardsInZone[l] = child.GetComponent<AICardToHand>().thisCard[0];
            l++;
        }

        for (int i = 0; i < 30; i++)
        {
            if (i >= howManyCards3)
            {
                cardsInHand[i] = CardDataBase.cardList[0];
            }
        }
        l = 0;
    }

    if(attackPhase==true && endPhase == false)
    {
        for(int i=0; i<30; i++)
        {
            if (canAttack[i] == true)
            {
                PlayerHP.staticHP -= cardsInZone[i].power;
            }
        }

        endPhase = true;
    }

    if (endPhase == true)
    {
        AiEndPhase = true;
    }
}

public void Shuffle()
{
    for (int i = 0; i < deckSize; i++)
    {
        container[0] = deck[i];
        int randomIndex = Random.Range(i, deckSize);
        deck[i] = deck[randomIndex];
        deck[randomIndex] = container[0];
    }
}

```

```

    }
    /*CardClone = (GameObject)*Instantiate(CardBack, transform.position,
transform.rotation);
    //CardClone.tag = "Clone";
    StartCoroutine(ShuffleNow());
}

IEnumerator StartGame()
{
    for (int i = 0; i <= 4; i++)
    {
        yield return new WaitForSeconds(1);
        Instantiate(CardToHand, transform.position, transform.rotation);
    }
}

IEnumerator ShuffleNow()
{
    yield return new WaitForSeconds(1);
    Clones = GameObject.FindGameObjectsWithTag("Clone");

    foreach(GameObject Clone in Clones)
    {
        Destroy(Clone);
    }
}

IEnumerator Draw(int x)
{
    for (int i = 0; i < x; i++)
    {
        yield return new WaitForSeconds(1);
        Instantiate(CardToHand, transform.position, transform.rotation);
    }
}

IEnumerator WaitFiveSeconds()
{
    yield return new WaitForSeconds(5);
}

IEnumerator WaitForSummonPhase()
{
    yield return new WaitForSeconds(5);
    summonPhase = true;
}
}

```

Файл DeackCreator.cs:

```

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

public class DeckCreator : MonoBehaviour
{
    public int[] cardsWithThisId;

    public bool mouseOverDeck;
    public int dragged;

    public GameObject coll;

```

```

public int numberOfCardsInDatabase;

public int sum;
public int numberOfDifferentCards;

public int[] saveDeck;

public GameObject prefab;

public bool[] alreadyCreated;

public static int lastAdded;

public int[] quantity;

// Start is called before the first frame update
void Start()
{
    sum = 0;
    numberOfCardsInDatabase = 8;
}

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

public void CreateDeck()
{
    for(int i=0; i< numberOfCardsInDatabase; i++)
    {
        sum += cardsWithThisId[i];
    }

    if (sum == 30) //Deck
    {
        for(int i=0; i <= numberOfCardsInDatabase; i++)
        {
            PlayerPrefs.SetInt("deck" + i, cardsWithThisId[i]);
        }
    }

    sum = 0;
    numberOfDifferentCards = 0;

    for(int i=0; i <= numberOfCardsInDatabase; i++)
    {
        saveDeck[i] = PlayerPrefs.GetInt("deck" + i, 0);
    }
}

public void EnterDeck()
{
    mouseOverDeck = true;
}

public void ExitDeck()
{
    mouseOverDeck = false;
}

```

```

public void Card1()
{
    dragged = Collection.x;
}
public void Card2()
{
    dragged = Collection.x+1;
}
public void Card3()
{
    dragged = Collection.x+2;
}
public void Card4()
{
    dragged = Collection.x+3;
}

public void Drop()
{
    if(mouseOverDeck == true &&
coll.GetComponent<Collection>().HowManyCards[dragged] > 0)
    {
        cardsWithThisId[dragged] ++;

        //if (cardsWithThisId[dragged] > 4)
        //{
        //    cardsWithThisId[dragged] = 4;
        //}

        if (cardsWithThisId[dragged] < 0)
        {
            cardsWithThisId[dragged] = 0;
        }

        coll.GetComponent<Collection>().HowManyCards[dragged]--;
        CalculateDrop();
    }
}

public void CalculateDrop()
{
    lastAdded = 0;
    int i = dragged;

    if(cardsWithThisId[i] > 0 && alreadyCreated[i] == false)
    {
        lastAdded = i;
        Instantiate(prefab, new Vector3(0, 0, 0), Quaternion.identity);
        alreadyCreated[i] = true;

        quantity[i] = 1;
    }
    else if(cardsWithThisId[i] > 0 && alreadyCreated[i] == true)
    {
        quantity[i]++;
    }
}
}

```