

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 комп'ютерної пригодницької гри «Spirit of the
forest»»

Виконала студентка групи КН-20-1:
БІЄМСЬКА Анна Сергіївна

Керівник роботи:
ГРАФ Марина Сергіївна

Рецензент:
ЛЕВКІВСЬКИЙ Віталій Леонідович

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СПЕЦІАЛЬНІСТЬ 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
Марина ГРАФ
«05» січня 2024 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: БІЄМСЬКА Анна Сергіївна
Керівник роботи: ГРАФ Марина Сергіївна
Тема роботи: «Розробка комп'ютерної пригодницької гри «Spirit of the forest»»,
затверджена наказом закладу вищої освіти від «25» січня 2024 р., №22/с
Термін здачі закінченої роботи: «10» червня 2024 р.
Вихідні дані роботи (зазначається предмет і об'єкт дослідження): розробка комп'ютерної
пригодницької гри «Spirit of the forest», яка базується на українській міфології та фольклорі.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Граф Марина Сергіївна	01.03.2024	20.01.2024
2	Граф Марина Сергіївна	21.04.2024	15.05.2024
3	Граф Марина Сергіївна	16.05.2024	01.06.2024

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання завдання. Опрацювання літературних джерел	25.09.23 – 10.11.23	Виконав
2	Проектування структури	11.11.23 – 20.01.24	Виконав
3	Написання програмного коду	21.01.24 – 20.04.24	Виконав
4	Тестування	21.04.24 – 25.05.24	Виконав
5	Попередній захист	10.06.2024	Виконав

Здобувач вищої освіти Анна БІЄМСЬКА
Керівник Марина ГРАФ

РЕФЕРАТ

Випускна кваліфікаційна робота складається з програмного додатку гри та пояснювальної записки.

Пояснювальна записка до випускної кваліфікаційної роботи складається з реферату, вступу, основної частини, загального висновку, списку використаної літератури та додатку.

Текстова частина викладена на 61 сторінці друкованого тексту.

Пояснювальна записка має 27 сторінок додатку. Список використаних джерел містить 20 найменувань, і займає 3 сторінки.

В роботі наведено 41 рисунок та 3 таблиці. Загальний обсяг роботи – 89 сторінок.

Ключові слова: ГРА, UNITY, 2D, SPIRIT OF THE FOREST, ПРИГОДИ, МІФОЛОГІЯ.

					КН.КР.Б – 122 – 24 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка комп'ютерної пригодницької гри «Spirit of the forest» Пояснювальна записка			Літ.	Арк.	Аркушів	
Розроб.		Біємська А.С.									
Керівник		Граф М.С.								2	
Реценз.								Житомирська політехніка, група КН-20-1			
Н. Контр.											
Затверд.		Граф М.С.									

ABSTRACT

The graduation thesis of the bachelor consists of a software application of the game and an explanatory note.

An explanatory note to the final thesis consists of an abstract, introduction, main part, general conclusion, a list of used literature and an appendix.

The text part is laid out on 61 pages of printed text.

The explanatory note has 27 pages of annexes. The list of used sources contains 20 names and occupies 3 pages.

The work contains 41 figures and 3 tables. The total volume of work is 89 pages.

Keywords: GAME, UNITY, 2D, SPIRIT OF THE FOREST, ADVENTURE, MYTHOLOGY.

		Біємська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	9
1.1 Постановка задачі.....	9
1.2 Аналіз існуючого програмного забезпечення за тематикою випускної кваліфікаційної роботи	10
1.3 Концепт-документ (Concept Document) [6]	13
1.4 Обґрунтування вибору інструментальних засобів	15
Висновки до першого розділу	16
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	17
2.1 Підготовка елементів гри.....	17
2.2 Розробка архітектури ігрового додатку	22
2.3 Основні етапи реалізації гри.....	29
Висновки до другого розділу.....	42
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ.....	43
3.1 Порядок встановлення та налаштування системи	43
3.2 Інтерфейс та порядок роботи з програмним додатком	44
3.3 Тестування роботи програмного забезпечення.....	51
Висновки до третього розділу:	57
ВИСНОВКИ	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	59
ДОДАТКИ	62
Додаток А	63
Додаток Б	66
Додаток В.....	72
Додаток Г	76
Додаток Д.....	80
Додаток Ж.....	84

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

RPG – англ. Role-Playing Game, жанр відеоігор, де основна частина ігрового процесу полягає в управлінні персонажем чи групою персонажів, які досліджують ігровий світ, виконують різноманітні завдання та розвиваються, слідуючи сюжету.

PC – англ. Personal computer, персональний комп'ютер.

UI – англ. User Interface, візуальна частина, зовнішній вигляд сайту чи застосунку.

UX – англ. User Experience, це користувацький досвід. Те, наскільки продукт інтуїтивно зрозумілий і які наші потреби вирішує.

NPC – англ. Non-player character, персонаж у комп'ютерних та у настільних рольових іграх, яким керує не гравець, а комп'ютер.

Game AI – англ. Game artificial intelligence, набір програмних методик, які використовуються у відеоіграх для створення ілюзії інтелекту в поведінці персонажів, керованих комп'ютером.

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. Сучасний світ ігрової індустрії швидко розвивається, постійно шукаючи нові шляхи для залучення гравців та створення незабутніх вражень. В цьому контексті, розробка комп'ютерної пригодницької гри «Spirit of the forest», яка базується на українській міфології та фольклорі, є надзвичайно актуальною і перспективною.

По-перше, ігрова індустрія в Україні та світі демонструє стабільне зростання. За даними Newzoo, станом на листопад 2023 року обсяг продажів ринку відеоігор становив 184 мільярди доларів, що свідчить про високу популярність та прибутковість цієї сфери [1]. В умовах такої динаміки, випуск унікальної гри, яка поєднує елементи національної культури з сучасними геймплейними рішеннями, має великі шанси на успіх.

По-друге, українська культура та міфологія залишаються недостатньо відомими на світовому рівні. Розробка такого роду гри сприятиме популяризації української культурної спадщини, зокрема, традицій, звичаїв і міфів. Використання таких елементів в іграх допомагає не тільки розважати гравців, але й навчати їх, підвищуючи обізнаність про культурне різноманіття світу.

По-третє, сучасні гравці все більше цінують оригінальні сюжети та глибокі історії в іграх. Пригодницька гра, яка пропонує занурення в магічний світ українського фольклору, де головному герою потрібно виконувати завдання, взаємодіяти з міфічними істотами та проходити різноманітні випробування, зможе забезпечити приємний ігровий досвід.

По-четверте, на фоні глобальних викликів та соціальних змін, такі ігри сприяють емоційному розвантаженню та психологічному благополуччю гравців. Занурення в чарівний світ, сповнений пригод та загадок, дозволяє відволіктися від повсякденних турбот, що є важливим аспектом сучасного життя.

Отже, розробка комп'ютерної пригодницької гри «Spirit of the forest» є актуальною з огляду на культурні, технологічні та ринкові аспекти. Вона не

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

лише задовольнить попит сучасних гравців, але й сприятиме популяризації української культури та розвитку національної індустрії розваг.

Метою випускної кваліфікаційної роботи є розробка комп'ютерної пригодницької гри, яка поєднує елементи української міфології з сучасними геймплейними механіками.

Об'єктом дослідження є методи та засоби розробки комп'ютерних ігор на основі міфологічного світу з використанням різноманітних геймплейних механік.

Предметом дослідження є специфіка розробки комп'ютерної пригодницької гри на основі української міфології та фольклору.

В рамках теми кваліфікаційної роботи було опубліковано тези:

Біємська А.С., Граф М.С., АЛГОРИТМ MINIMAX В КАРТКОВІЙ ГРІ. Актуальні проблеми та перспективи розвитку фундаментальних, прикладних, загально-технічних та безпекових наук: Тези IV Всеукраїнської науково-практичної конференції пам'яті академіка Академії наук вищої освіти України, професора Анатолія Володимировича Касперського. 27 червня 2024 року м. Київ.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Відповідно до обраної теми "Розробка комп'ютерної пригодницької гри «Spirit of the forest»", було сформовано перелік завдань для кваліфікаційної роботи. Завдання розподілені з метою систематизації та послідовного виконання етапів розробки гри [2].

1. *Аналіз існуючих 2D-ігор*: Провести огляд і аналіз існуючих 2D-ігор, зокрема тих, які базуються на міфологічних темах. Визначити їхні особливості, плюси та мінуси, що стануть основою для формулювання унікальних рішень у розробці власної гри.
2. *Дослідження міфології*: Провести дослідження української міфології та фольклору, вивчаючи міфи, легенди, казки, повір'я та інші джерела. Визначити ключові елементи української міфології, які можна інтегрувати в геймплей гри.
3. *Розробка концепції гри*: Сформувати концепцію гри, включаючи сюжет, основних персонажів, локації та основні механіки гри. Врахувати унікальні аспекти української міфології та їхнє впровадження у гру.
4. *Створення 2D-графіки*: Розробити графічний дизайн для гри, включаючи фони, персонажів, об'єкти та анімацію. Використати українську міфологію як джерело натхнення для створення графічних елементів.
5. *Програмування ігрової логіки*: Здійснити програмування ігрової логіки, включаючи механіку рух персонажів, взаємодію об'єктів та інші ігрові аспекти. Використати обрану платформу для розробки гри та мову програмування.
6. *Тестування та доопрацювання гри*: Провести тестування гри для виявлення помилок, недоліків та оптимізації геймплею. Здійснити

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

внесення необхідних змін та доопрацювання гри з урахуванням отриманих результатів тестування.

1.2 Аналіз існуючого програмного забезпечення за тематикою випускної кваліфікаційної роботи

Для вивчення особливостей, механік та інших аспектів 2D-ігор, які можуть бути корисними для розробки власної гри, було проведено аналіз трьох ігор: Machinarium, The Witcher 3: Wild Hunt та Hearthstone: Heroes of Warcraft.

Machinarium (<https://amanita-design.net/games/machinarium.html>) є пригодницькою грою з високоякісною 2D-графікою та унікальним арт-стилем. Вона відзначається складними головоломками та атмосферним сюжетом, який передається без використання слів. Гра пропонує гравцеві вирішувати різноманітні головоломки та завдання, що базуються на взаємодії з навколишнім середовищем та розумінні логіки внутрішнього світу гри. "Machinarium" демонструє, як вдало поєднати пригодницький сюжет зі складними головоломками та унікальним арт-стилем (рисунок 1.1) [3].



Рисунок 1.1 – Приклад геймплею гри «Machinarium»

До переваг гри можна віднести: чудовий стиль малювання; складні загадки, над якими доводиться довго ламати голову; загальна атмосфера. До

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

недоліків: не можна використовувати праву кнопку миші, що може бути незручним для деяких гравців; стислість гри.

The Witcher 3: Wild Hunt (<https://en.cdprojektred.com/download/?support-product=the-witcher-3>) - відома гра, що здобула визнання завдяки своєму глибокому сюжету, різноманітним персонажам та захоплюючому світу. Вона містить елементи фентезі, включаючи міфологічних істот та легенди. Головними механіками гри є бойова система, дослідження світу та прийняття рішень, що впливають на подальший сюжет. "The Witcher 3: Wild Hunt" показує, як можна успішно впровадити міфологічні елементи у світ ігор, а також як створити захоплюючий сюжет, який привертає увагу гравців (рисунок 1.2) [4].



Рисунок 1.2 – Приклад геймплею гри «The Witcher 3: Wild Hunt»



Рисунок 1.3 – Приклад гри в Гвинт («The Witcher 3: Wild Hunt»)

В гру також входить карткова гра "Гвинт" (рисунок 1.3), яка є популярною серед гравців та навіть стала окремим продуктом. У "Гвинті" вам доведеться битися в карткових поєдинках, для перемоги в яких необхідні

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

стратегічне мислення, вміння блефувати і вдало зібрана колода. Він демонструє, як можна впровадити додатковий елемент гри в існуючий світ RPG, що додає глибини та різноманітності геймплею.

До переваг гри можна віднести: величезний світ, красивий і наповнений різноманітним унікальним контентом; цікавий сюжет з безліччю персонажів та завдань; динамічна система бою, яка дозволяє гравцям використовувати різні тактики та прийоми; карткова гра "Гвинт", яка додає до гри новий елемент стратегії та розваги. До недоліків: слабкий штучний інтелект; деякі завдання та монстри можуть здатися занадто повторюваними.

Hearthstone: Heroes of Warcraft (<https://hearthstone.blizzard.com/en-us>) – це безкоштовна колекційна карткова онлайн-гра. Спочатку гравці обирають одного з дев'яти персонажів всесвіту Warcraft, а потім по черзі розігрують карти із зібраної ними колоди цього персонажа, що включає потужні заклинання, здібності та зброю героїв, або закликають сильних союзників для боротьби із противником (рисуюнок 1.4) [5].



Рисуюнок 1.4 – Приклад геймплею гри «Hearthstone: Heroes of Warcraft»

До переваг гри можна віднести: проста, але дуже захоплююча механіка; швидкий та динамічний геймплей; зручний інтерфейс; різноманітність карт та героїв, що дозволяє гравцям використовувати різні тактики та стратегії; регулярні оновлення та новий контент, що робить гру свіжою та цікавою протягом тривалого часу. До недоліків: всього три режими; немає сюжетної кампанії; залежність від удачі.

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

Отже, проведений аналіз дозволив виявити ряд важливих аспектів, які слід врахувати при розробці "Spirit of the Forest". До них належать:

- Унікальний арт-стиль та атмосфера гри.
- Захопливий сюжет та цікаві персонажі.
- Динамічний геймплей, який тримає гравця в напрузі.
- Різноманітність механік та можливостей.

Кращі практики та ідеї, отримані в результаті аналізу існуючого програмного забезпечення будуть використані під час розробки комп'ютерної пригодницької гри "Spirit of the Forest".

1.3 Концепт-документ (Concept Document) [6]

Жанр і цільова аудиторія:

Жанр: Пригодницька гра з елементами головоломки та RPG.

Цільова аудиторія: Гра буде цікавою для любителів пригодницьких ігор та фанатів міфології.

Вікова категорія: 16+.

Основні особливості гри:

Гра буде розгортатися у світі, заснованому на українській міфології та фольклорі. Гравці візьмуть на себе роль головного героя, який вирушить у подорож, щоб розкрити таємниці стародавнього лісу та врятувати світ від темних сил. Також гра буде містити різноманітні завдання, які потребуватимуть від гравців логічного мислення, кмітливості та спостережливості.

Геймплей:

Гра розпочинається з святкових забавах в селі, де гравець знайомиться з керуванням та взаємодією з персонажами. Після святкувань гравець вирушає до річки, де йому потрібно пустити віночок по воді, щоб викликати русалку. Ця зустріч обертається картковою битвою, яку гравець мусить виграти, щоб продовжити подорож.

Далі шлях веде героя до темного лісу, де він зустрічає лісовика та його трьох маленьких духів. Ця зустріч також закінчується картковою битвою, яку

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

гравець мусить подолати. Щоб продовжити, гравцеві потрібно розгадати загадку з рунами правильно підібравши комбінацію рун. На шляху герой також зустрічає мавку, на яку полює чугайстер. Гравцеві надається можливість вибрати сторону: допомогти мавці або стати на бік чугайстра. Цей вибір впливає веде до карткової битви з обраним персонажем.

Зібравши всі частини мапи, герой прямує до фінального босу - чорта, який охороняє заповітну квітку папороті. Битва з чортом складається з трьох етапів: карткової гри, гри на запам'ятовування, а також лабіринту. Після перемоги над чортом, герой нарешті здобуває квітку папороті.

Сюжет:

На свято Івана Купала молодий хлопець вирушає на пошуки цвіту папороті, що, за легендою цвіте лише раз на рік, й дарує незвичайні здібності. Це здається останньою надією. адже його село охоплене темрявою, лихо загрожує його рідній домівці та близьким людям.

Юнак вирушає на пошуки цвіту папороті, сподіваючись врятувати своє село. Проте, не все так просто. Зло, яке нависло над селом, розділило мапу до квітки на шість частин і заховало їх у трьох різних локаціях: у селі, у темних водах річки та у гущавині таємничого лісу.

Спочатку герой йде до села, де стикається з традиційними святковими забавами. Наступна локація - темні води річки, де хлопець зустрічає русалку, що оберігає частину мапи. Найнебезпечнішим випробуванням стає темний ліс, де юнак зіткнеться з небезпечними лісовими істотами та має здобути останні частини мапи.

Зібравши всі шість частин мапи, герой нарешті знаходить шлях до заповітної квітки папороті. Проте, на цьому його пригоди не закінчуються. Зло, яке хотіло завадити йому, не збирається здаватися. На шляху до квітки герою доведеться перемогти у фінальній битві з чортом, який охороняє квітку.

Після перемоги над чортом, хлопець нарешті здобуває квітку папороті і перемагає зло, яке загрожувало його селу. Село знову осяяне світлом, а люди щасливі та вільні.

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

Платформа:

Наразі планується, що гра буде розроблятися тільки під PC платформу.

1.4 Обґрунтування вибору інструментальних засобів

Для реалізації проєкту комп'ютерної пригодницької гри «Spirit of the forest» було визначено набір інструментів, які надають широкі можливості для створення високоякісного ігрового продукту, а саме:

- *Unity* - це багатоплатформовий інструмент для розроблення відеоігор і застосунків, і рушій, на якому вони працюють. Має інтуїтивний інтерфейс, що дозволяє швидко оволодіти основними функціями. Підтримує 2D- та 3D-розробку, а також широкий спектр платформ, включаючи PC, мобільні пристрої та консолі. Має вбудовані інструменти для створення рівнів, анімації, UI/UX та багато іншого. Unity Asset Store пропонує широкий спектр плагінів та асетів, які можна використовувати для розширення можливостей рушія [7].
- *C#* - це мова програмування, яка широко використовується для розробки ігор в середовищі Unity. Вона проста у вивченні та розумінні, має чіткий синтаксис. Підтримує сучасний інструментарій, який дозволяє швидко розробляти складні програмні рішення. Має високий рівень абстракції, що спрощує написання коду та розуміння програми.
- *Visual Studio* - це інтегроване середовище розробки (IDE), яке використовується для розробки програмного забезпечення, включаючи ігри. Visual Studio має вбудовану підтримку мови C#, що робить його зручним інструментом для розробки ігор на Unity.
- *Adobe Photoshop* - це програма для роботи з растровою графікою, яка використовується для створення та редагування графічних ресурсів. Має широкий спектр інструментів для малювання, редагування та обробки зображень. Підтримує роботу з різними типами файлів, включаючи PSD, PNG, JPEG та інші. Є доволі

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

простою у використанні та існують багато онлайн-ресурсів та навчальних матеріалів до неї.

- *DragonBones* - це програма для створення анімацій та роботи з рухомими скелетами, яка використовується для анімації персонажів у 2D-іграх. Має простий інтерфейс та інтуїтивно зрозумілий робочий простір. Підтримує експорт анімацій у різні формати, включаючи JSON, PNG, Spine та Flash. Має широкий набір інструментів для редагування та налаштування анімаційних об'єктів.
- *FL Studio* - це редактор-секвенсор, що використовується для створення та редагування музики та звукових ефектів. Має широкий спектр віртуальних інструментів, семплів та ефектів. Дозволяє створювати музику будь-якого жанру. Підтримує різні формати звукових файлів та можливість імпорту та експорту звукового контенту.

Висновки до першого розділу

В першому розділі було визначено перелік завдань для кваліфікаційної роботи з метою послідовного виконання етапів розробки комп'ютерної пригодницької гри "Spirit of the forest". Проведено аналіз існуючого програмного забезпечення ("Machinarium", "The Witcher 3: Wild Hunt" та "Hearthstone: Heroes of Warcraft"), наведено їх переваги та недоліки, та на основі цього виявлено ряд важливих аспектів, які слід врахувати при розробці власної гри. Створено концепт-документ гри "Spirit of the Forest", який описує жанр та цільову аудиторію, основні особливості, геймплей, сюжет та платформу. Обґрунтовано вибір інструментів для розробки гри (Unity, C#, Visual Studio, Adobe Photoshop, DragonBones, FL Studio).

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Підготовка елементів гри

Графічний дизайн є важливою складовою будь-якої гри, адже він відповідає за візуальне сприйняття та естетику. Для дизайну гри "Spirit of the forest" було обрано прості текстури та яскраві кольори для того, щоб створити атмосферу казковості. Вони допомагають поглибити імєрсію гравця у світ гри, роблячи кожен локацію захоплюючою та привабливою. Це відображається у фонових зображеннях, персонажах, об'єктах та анімації, що створює цілісну та привабливу графічну картину.

Інтерфейс гри має відповідати вибраному візуальному стилю. Це стосується кольорової гами, шрифтів, іконок та загального дизайну. Він допомагає гравцям легко орієнтуватися у грі, не відволікаючи їх від основного геймплею. А також інтерфейс має бути функціональним та відповідати потребам гравців. Він має надавати їм всю необхідну інформацію та дозволяти легко керувати грою.

Меню гри має бути чітко структурованим та зручним для навігації. В якості основних кольорів UI-елементів було обрано відтінки зеленого (рисунки 2.1), щоб підкреслити зв'язок з лісом та природними мотивами в грі.



Рисунок 2.1 – Кольорова палітра UI-елементів

Підказки (рисунки 2.2) можуть допомогти гравцям, якщо вони застрягли або не знають, що робити. Вони мають бути ненав'язливими та не псувати ігровий процес, тому їх колір є напівпрозорим сірим.



Рис. 2.2 – Приклад підказки

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

Індикатори мають бути візуально чіткими та легко зчитуватись. В якості прикладу було обрано індикатор здоров'я (рисунок 2.3) червоного кольору.

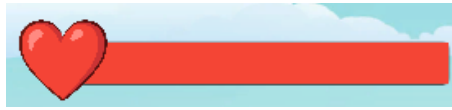


Рисунок 2.3 – Індикатор здоров'я гравця

Інвентар показує предмети, які має гравець (рисунок 2.4). Гравець може вибирати потрібний предмет та використати його.



Рисунок 2.4 – Інвентар гравця

Діалоги відображають розмови між головним героєм та іншими персонажами. Вони мають вигляд зображень персонажів з текстом біля них, знизу є інтерактивні варіанти відповіді гравця (рисунок 2.5).

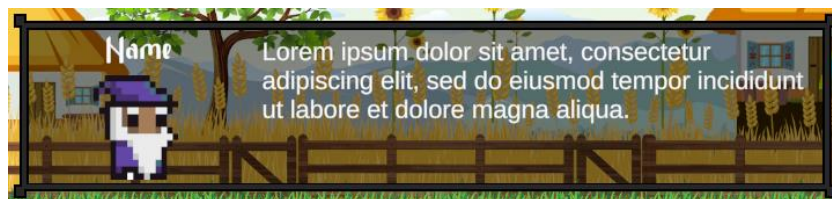


Рисунок 2.5 – Приклад діалогу

Для фонів гри було застосовано паралакс ефект, щоб зробити їх більш динамічними, додати глибини світу гри та зробити його більш візуально привабливим. Фони гри відповідають локаціям, що є в грі – це село (рисунок 2.4), річка (рисунок 2.5) та загадковий ліс (рисунок 2.6). Також під час створення локацій буде використано систему часток для додавання атмосферних ефектів, таких як легкий туман, мерехтіння зірок тощо. Принцип роботи системи часток полягає в тому, що вона створює та керує великою кількістю малих графічних об'єктів (часток), які рухаються та змінюються у часі. Кожна частка може мати свої унікальні характеристики, такі як швидкість руху, розмір, кольори, прозорість, форма та тривалість життя.



Рисунок 2.6 – Фонове зображення для локації села



Рисунок 2.7 – Фонове зображення для локації річки

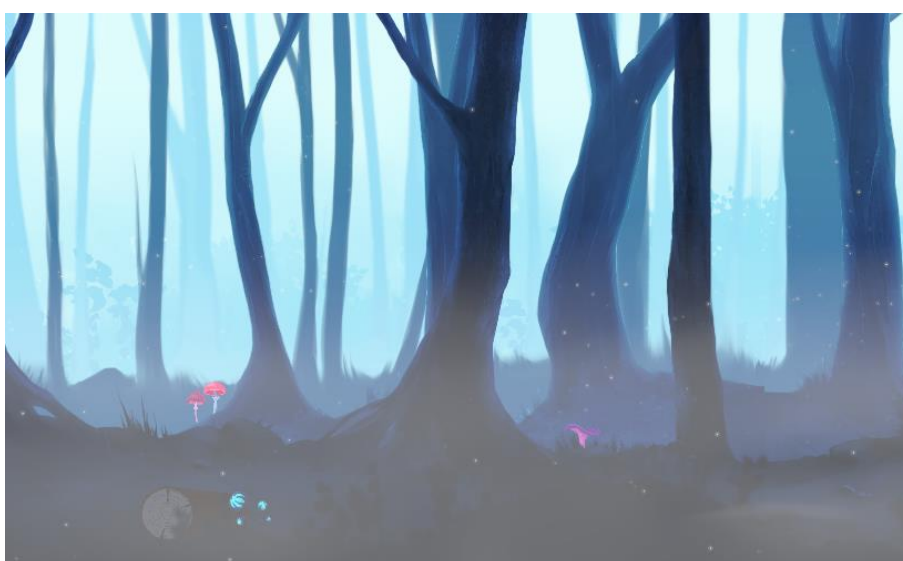


Рисунок 2.8 – Фонове зображення для локації лісу

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

До основних персонажів гри належить: головний герой (рисунок 2.9), жителі села та різні міфологічні істоти (рисунок 2.10), засновані на українській міфології та фольклорі, що трапляються на шляху героя під час подорожі. Вони можуть бути як дружніми, так і ворожими до гравця.



Рисунок 2.9 – Головний герой гри



Рисунок 2.10 – Приклад міфологічної істоти (лісовик)

Анімація (рисунок 2.11) в грі служить для підвищення динаміки та емоційної насиченості. Вона додає життя персонажам, об'єктам та оточенню, роблячи геймплей більш цікавим та захоплюючим. Кісткова анімація використовує скелетну структуру, щоб контролювати рухи персонажів чи об'єктів. Вона базується на системі кісток та з'єднань, які визначають, як частини персонажа або об'єкта повинні рухатися. Рух кісткової анімації зазвичай задається шляхом створення ключових кадрів та інтерполяції між ними.

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		20



Рисунок 2.11 – Приклад анімації співу

Основні ігрові об'єкти:

- *Сюжетні предмети* (рисунок 2.12): Необхідні для просування по сюжету. Наприклад, віночок, який потрібен, щоб викликати русалку.



Рисунок 2.12 – Приклад сюжетного предмета (віночок)

- *Лікувальне зілля* (рисунок 2.13): Відновлює здоров'я головного героя.



Рисунок 2.13 – Лікувальне зілля

- *Перешкоди* (рисунок 2.14): Різноманітні предмети, які заважають гравцеві просуватися, включаючи багаття, жалючі рослини, колоди тощо.



Рис. 2.14 – Приклад перешкоди (багаття)

Інтро служить важливим вступом до сюжету та налаштовує гравця на атмосферу гри (рисунок 2.15). Воно переносить гравця в чарівний світ

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

українських традицій та магії та створює відчуття занурення у давні обряди. Легенда про квітку папороті, яка може здолати зло, вводить гравця в основний конфлікт гри, надаючи чітке розуміння його завдання та стимулюючи до подальшої гри.



Рисунок 2.15 – Кадр з вступного ролика до гри

Музика в грі є ключовим елементом для створення настрою, емоційного зв'язку з гравцем та підсилення атмосфери гри. Використання традиційних етнічних інструментів, таких як флейти, бандури, бубни, дудки та інші, може допомогти створити аутентичну атмосферу, що відображає український фольклор. Для важливих моментів у грі можна використовувати епічну, героїчну музику з міцними ударними, що підкреслить напруження. Для моментів відпочинку музика може бути спокійною, з використанням м'яких мелодій та природних звуків. Паралельно з музикою можуть використовуватися звукові ефекти, такі як шеліт лісу, шум води, спів птахів та інші звуки природи, що додають реалізму та глибини оточенню гри.

2.2 Розробка архітектури ігрового додатку

Діаграма варіантів використання (Use Case Diagram) (рисунок 2.16) показує взаємодію між користувачами (акторами) та системою (грою) [8]. В грі «Spirit of the forest» гравець може досліджувати різні локації, такі як село, річка, ліс, щоб знайти частини мапи, необхідні для доступу до квітки папороті. Під час дослідження локацій гравець може зустрічати ворогів та NPC, вирішувати завдання, битись з ворогами, знаходити цікаві предмети та використовувати їх.

		Бісцька А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

В ігровому меню гравець може змінити налаштування гри, переглянути досягнення, бестіарій, перевірити скільки шматків мапи вже зібрано тощо.



Рисунок 2.16 – Діаграма варіантів використання

Діаграма активності (Activity Diagram) (рисунок 2.17) показує потік робіт або дій у системі. Створена діаграма відображає основні кроки, які виконує гравець протягом гри, включаючи різні взаємодії та переходи між етапами гри.

Після початку гри гравець потрапляє у головне меню, де можна змінити налаштування, переглянути досягнення, вийти з гри та продовжити або розпочати нову гру (в залежності від наявності збережень у системі). Якщо гравець почне нову гру, то він зможе переглянути вступний ролик. В якому розповідається про світ гри. Потім гравець потрапляє на локацію (село, річка або ліс) та досліджує її. Під час цього гравець може збирати предмети та використовувати їх, а також зустрічати персонажів (як дружніх, так і ворогів). Зустріч з NPC зазвичай закінчується міні-грою (наприклад «три в ряд»), в той час як зустріч з ворогом закінчується картковою битвою. Після перемоги в битві або успішного завершення міні-гри гравець отримує шматки мапи, що потрібні, щоб потрапити до боса, який охороняє квітку папороті. Коли гравець отримає всі 6 шматків мапи, він може розпочати бій з босом. Який проходить у три етапи: карткова гра, гра на запам'ятовування та лабіринт. Після перемоги

гравець отримує квітку папороті та гра завершується. Також під час дослідження локації гравець може користуватись внутрішньоігровим меню, що включає в себе перегляд бестіарію, кількості зібраних шматків мапи та зміна налаштувань.

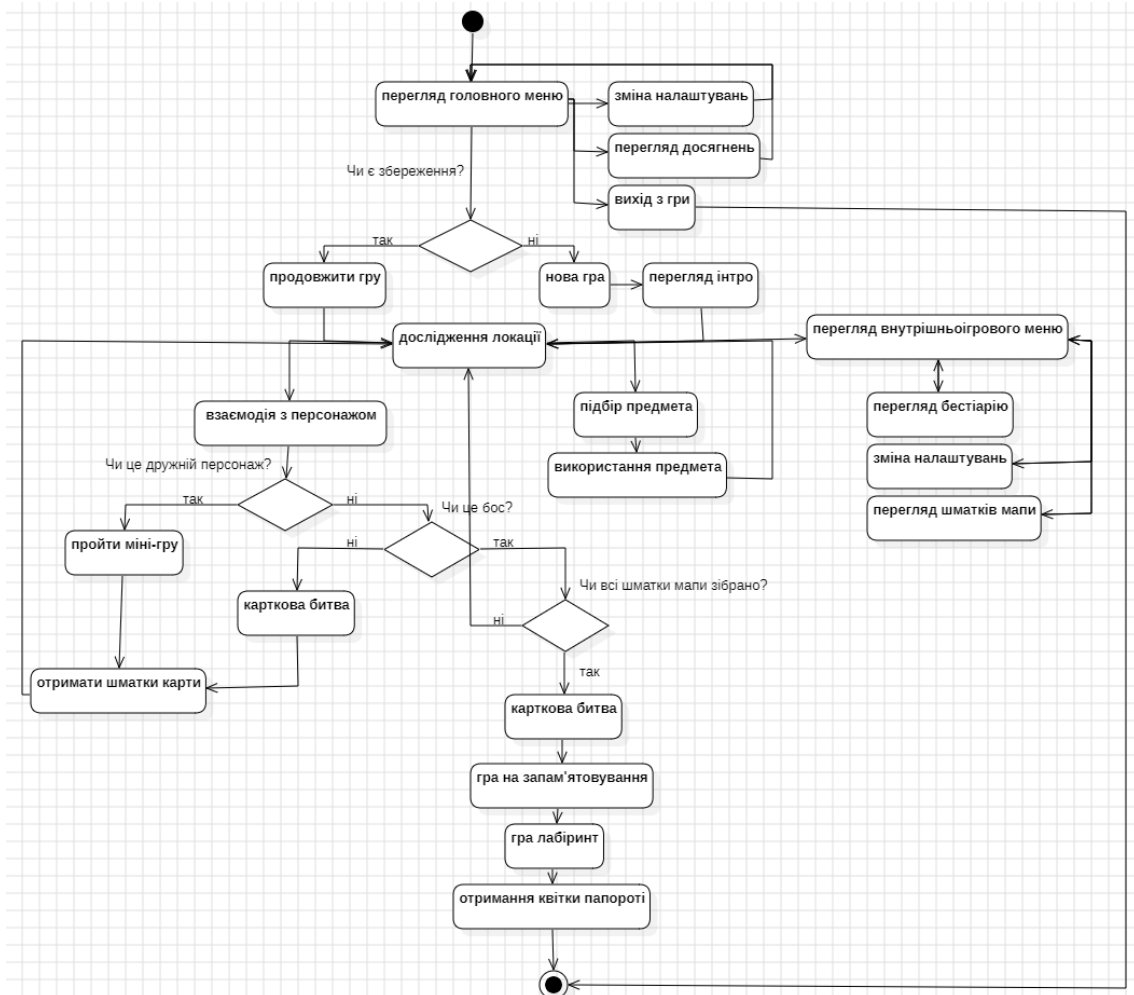


Рисунок 2.17 – Діаграма активності

Діаграма класів (Class Diagram) (рисунок 2.18) використовується для представлення структури програмного забезпечення. Вона показує класи, їх атрибути, методи та зв'язки між ними [9]. В даному випадку представлено класи, що відповідають за реалізацію карткової гри.

Опис класів:

- Game: Представляє логіку та дані всієї гри, включаючи гравців, карти, стан гри тощо.
- Player: Представляє гравця в грі з його характеристиками (здоров'я, мана, колода карт).

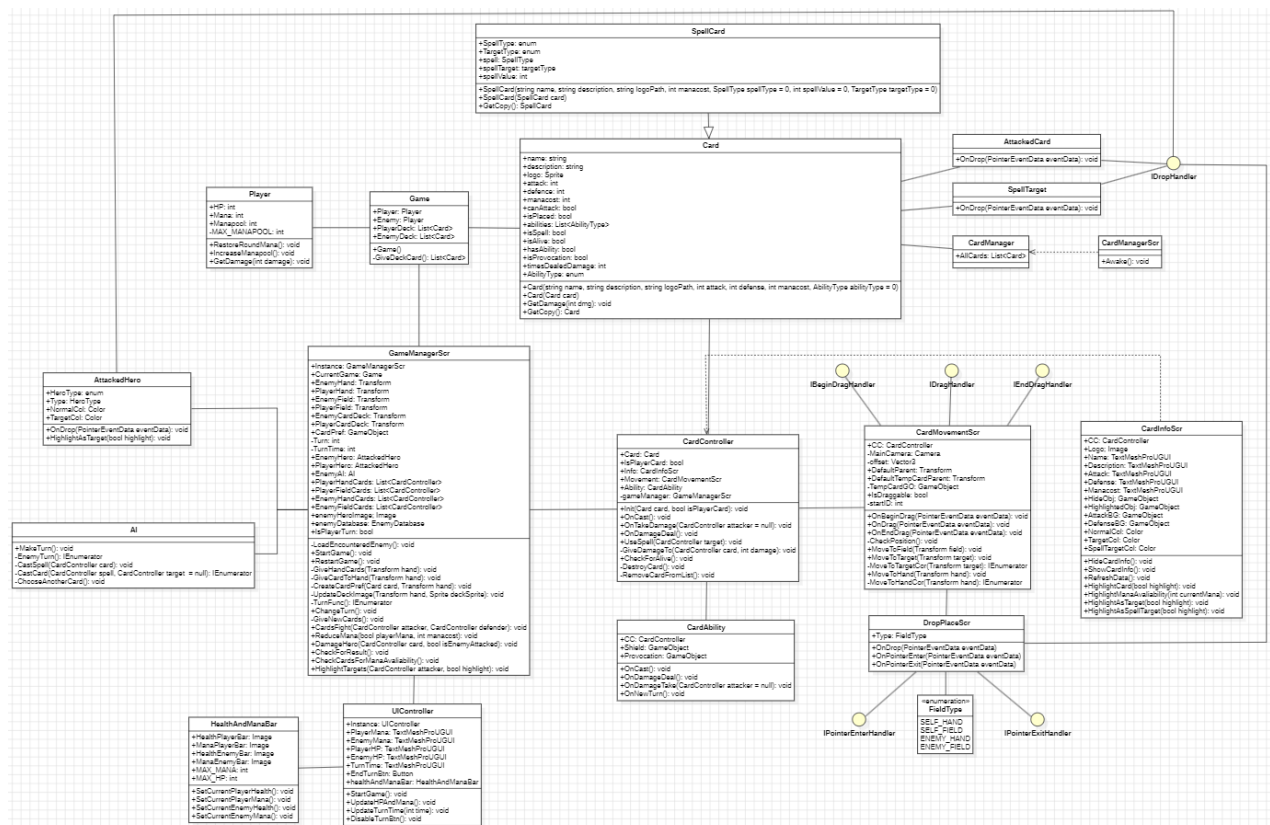


Рисунок 2.18 – Діаграма класів

- Card: Базовий клас для всіх карт у грі, містить спільні атрибути та методи.
- SpellCard: Підклас Card, що представляє карти заклинань з унікальними властивостями та ефектами.
- CardManager: Керує даними про карти, надає методи для доступу та створення карт.
- GameManagerScr: Контролює ігровий процес, реалізовує правила гри, взаємодіє з гравцями та UI.
- UIController: Керує елементами користувацького інтерфейсу (UI), оновлює їх відповідно до стану гри.
- HealthAndManaBar: Візуалізує здоров'я та ману гравців.
- CardInfoScr: Відображає інформацію про вибрану карту.
- AttackedCard: Представляє карту, яка зазнала атаки.
- SpellTarget: Визначає об'єкт, на який може бути застосовано заклинання.
- AttackedHero: Представляє героя, який зазнав атаки.

- CardManagerScr: Додатковий клас для керування картами.
- FieldType: Визначає тип поля гри (рука, поле).
- DropPlaceScr: Логіка скидання карт на ігрове поле.
- CardMovementScr: Керує рухом та візуальними ефектами карт.
- CardAbility: Описує здібності карт.
- AI: Ігровий інтелект для гри проти ворогів.

Також під час проектування діаграми класів були використані патерни для покращення структури, забезпечення гнучкості та полегшення розширення системи. Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм. Вони являють собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми [10].

Одинак (Singleton) — це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього. Цей патерн, наприклад, використовується в класі `UIController` для створення єдиного екземпляра класу протягом усього виконання програми. Це досягається за допомогою перевірки в методі `Awake`, чи існує вже екземпляр `UIController`. Якщо ні, створюється новий екземпляр і робиться його персистентним, щоб він не знищувався при переході між сценами. Якщо ж екземпляр вже існує, новий екземпляр не створюється, а поточний екземпляр використовується.

Спостерігач (Observer) — це поведінковий патерн проектування, який створює механізм підписки, що дає змогу одним об'єктам стежити й реагувати на події, які відбуваються в інших об'єктах. Цей патерн використовується неявно в декількох місцях. Наприклад, `UIController` підписується на оновлення `GameManagerScr`, щоб оновлювати інтерфейс користувача при зміні моделі (наприклад, при зміні здоров'я, мани тощо).

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

Facade (Facade) — це структурний патерн проектування, який надає простий інтерфейс до складної системи. Цей патерн використовується в класі GameManagerScr, що діє як центральний інтерфейс для взаємодії з грою.

Для графічного відображення фізичних моделей систем використовуються діаграми реалізації (Implementation Diagrams), що включають в себе діаграму компонентів (Component Diagram) та діаграму розгортання (Deployment Diagram). Діаграми реалізації розглядають взаємодію та організацію фізичних компонентів системи, надаючи уявлення про архітектуру та взаємодію між компонентами на рівні виконання.

Діаграма компонентів є одним із видів UML-діаграм, який дозволяє визначити архітектурні аспекти системи, такі як компоненти, їхні взаємозв'язки та інтерфейси. Основна мета діаграми компонентів - це надати високорівневий огляд структури системи та її компонентів на фізичному рівні.

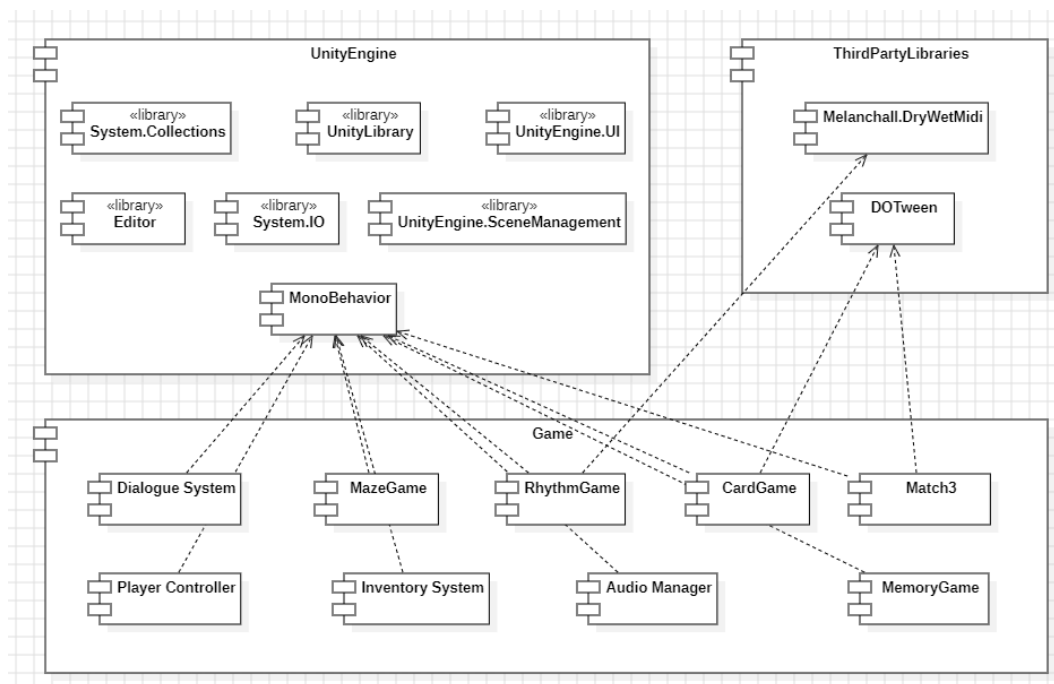


Рисунок 2.19 – Діаграма компонентів

Модуль UnityEngine містить основні бібліотеки та компоненти Unity, необхідні для роботи гри. Він поділяється на кілька підмодулів, серед яких MonoBehavior - базовий клас Unity, від якого успадковуються всі компоненти гри. Модуль ThirdPartyLibraries містить сторонні бібліотеки, які використовуються в проєкті:

- Melanchall.DryWetMidi: Бібліотека для роботи з MIDI-файлами, використовується для відтворення музики та звукових ефектів в грі.
- DOTween: Бібліотека для створення та управління анімаціями в Unity, що спрощує процес анімації об'єктів гри.

Модулі гри представляють собою окремі підсистеми, відповідальні за різні ігрові механіки (Inventory System, Player Controller, Audio Manager, Dialogue System, MazeGame, RhythmGame, CardGame, MemoryGame, Match3), що взаємодіють з MonoBehavior, а також можуть використовувати сторонні бібліотеки за необхідністю.

Діаграма розгортання надає візуальне представлення взаємозв'язків між апаратним та програмним забезпеченням, а також показує, як різні компоненти системи фізично розташовані та взаємодіють один з одним. При моделюванні діаграми розгортання враховуються такі аспекти, як сервери, обладнання, засоби зв'язку та програмне забезпечення, яке працює на цьому обладнанні. Ця інформація є ключовою для розуміння фізичної структури системи та планування її ефективного впровадження.

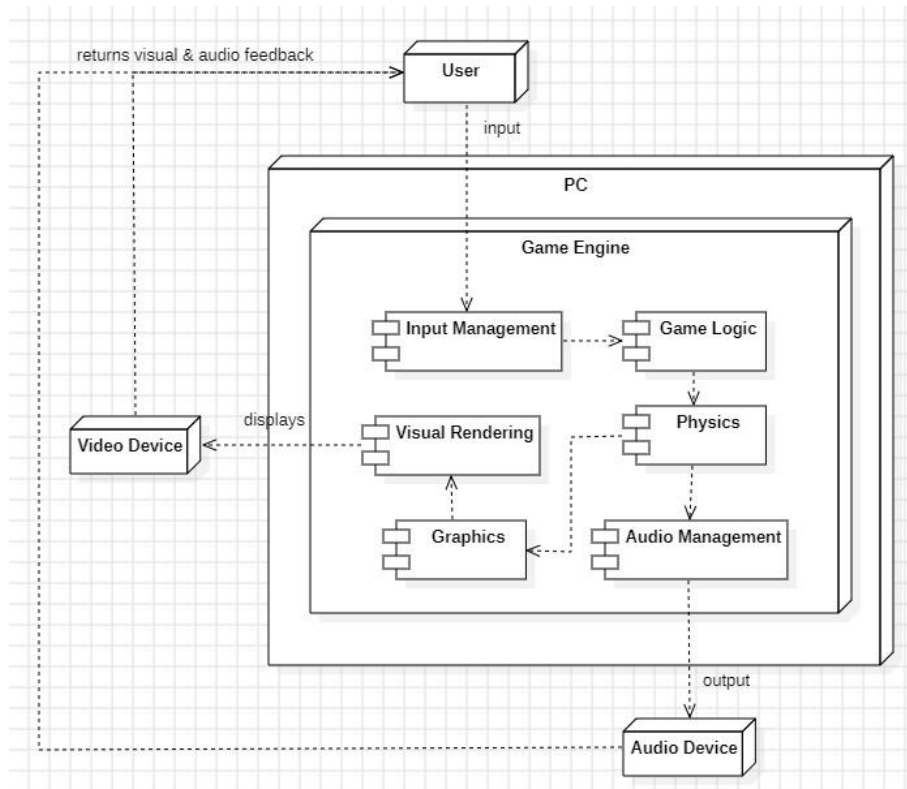


Рисунок 2.20 – Діаграма розгортання

PC - це комп'ютер, на якому запускається гра. Video Device відповідає за виведення візуального контенту гри на екран. Audio Device відповідає за відтворення звукових ефектів та музики гри. Game Engine - головний рушій гри, що включає в себе наступні підкомпоненти:

- Visual Rendering: Відповідає за відтворення графіки на екрані.
- Graphics: Обробка та рендеринг графічних об'єктів.
- Physics Engine: Обробка фізичних взаємодій у грі.
- Input Management: Управління введенням від користувача
- Game Logic: Основна логіка гри.
- Audio Management: Управління звуками та музикою у грі.

Користувач взаємодіє з грою за допомогою пристроїв введення (клавіатура, миша). Пристрої введення надсилають дані до Input Management, що обробляє отримані дані та надсилає їх до відповідних підсистем рушія. Вони в свою чергу використовують дані про введення та інші дані для оновлення стану гри.

2.3 Основні етапи реалізації гри

Реалізація ігрової логіки є ключовим етапом розробки, який визначає основні механіки геймплею та взаємодію об'єктів у грі. Почнемо з руху персонажа. Він може рухатися вліво та вправо за допомогою кнопок A та D та стрибати за допомогою Space.

Лістинг:

```
// Отримуємо вхід від гравця
float horizontalInput = Input.GetAxis("Horizontal");
float verticalInput = Input.GetAxis("Vertical");

// Рух по горизонталі та вертикалі
Vector2 movement = new Vector2(horizontalInput, verticalInput);
rb.velocity = new Vector2(movement.x * speed, rb.velocity.y);

// Визначення напрямку руху
if (movement.x > 0)
{
    transform.localScale = new Vector3(0.5f, 0.5f, 1); // Гравець
    направлений вправо
}
else if (movement.x < 0)
```

		Бісцька А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

```

    {
        transform.localScale = new Vector3(-0.5f, 0.5f, 1); // Гравець
направлений вліво
    }

    // Візуалізація OverlapCircle
    Debug.DrawLine(transform.position, transform.position + Vector3.down
* 2f, Color.red);

    // Перевірка, чи стоїть гравець на землі
    isGrounded = Physics2D.OverlapCircle(transform.position, 2f,
LayerMask.GetMask("Ground"));

    // Стрибок
    if (isGrounded && Input.GetKeyDown(KeyCode.Space))
    {
        Jump();
    }

    // Визначення стану ходьби та стрибка
    isJumping = !isGrounded;

```

При цьому відтворюється відповідна анімація та визначається певний стан: спокій, ходьба або стрибок. Кожен стан має відповідний набір правил та обмежень, які регулюють поведінку героя в цьому стані. Перехід між станами відбувається в залежності від взаємодії гравця з грою.

Лістинг:

```

public enum PlayerState
{
    Idle,
    Walking,
    Jumping,
    Dead
}
// ...
void ChangeState(PlayerState newState)
{
    currentState = newState;

    switch (newState)
    {
        case PlayerState.Idle:
            animator.SetBool("IsWalking", false);
            animator.SetBool("IsJumping", false);
            animator.Play("boy_idle");
            break;
        case PlayerState.Walking:
            animator.SetBool("IsWalking", true);
            animator.SetBool("IsJumping", false);
            animator.Play("boy_walk");
            break;
        case PlayerState.Jumping:
            animator.SetBool("IsWalking", false);

```

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

```

        animator.SetBool("IsJumping", true);
        animator.Play("boy_jump");
        break;
    case PlayerState.Dead:
        // логіка для стану "Dead"
        break;
    }
}

```

Для створення візуальної глибини та динаміки у фонових зображеннях, які рухаються повільніше, ніж об'єкти переднього плану було застосовано паралакс ефект [11]:

```

using UnityEngine;

public class ParallaxEffect : MonoBehaviour
{
    [SerializeField] Transform followingTarget;
    [SerializeField, Range(0f, 1f)] float parallaxStrenght = 0.1f;
    [SerializeField] bool disableVerticalParalax;
    Vector3 targetPreviousPosition;

    void Start()
    {
        if(!followingTarget)
        {
            followingTarget = Camera.main.transform;
        }

        targetPreviousPosition = followingTarget.position;
    }

    void Update()
    {
        var delta = followingTarget.position -
targetPreviousPosition;

        if (disableVerticalParalax)
        {
            delta.y = 0;
        }

        targetPreviousPosition = followingTarget.position;
        transform.position += delta * parallaxStrenght;
    }
}

```

Інвентар в грі дозволяє гравцям збирати, зберігати та використовувати різні предмети. Він представляє собою список предметів ('items'), який зберігається в об'єкті 'Inventory'. Кожен предмет у списку має свій унікальний ідентифікатор ('id'), назву ('name'), зображення ('img') та префаб ('prefab').

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

Предмети в інвентарі відображаються у вигляді графічних об'єктів, які відображають зображення предмета та кількість одиниць цього предмета [12].

Коли гравець збирає предмет, об'єкт 'ItemPickup', який містить інформацію про предмет, виявляє колізію з гравцем. Після цього об'єкт 'Inventory' перевіряє, чи вже є у ньому такий самий предмет. Якщо такого предмета немає, він додається до списку предметів. Якщо такий предмет вже є в інвентарі, то нові одиниці додаються до існуючої кількості.

```
public void SearchForSameItem(Item item, int count)
{
    for(int i = 0; i < maxCount; i++)
    {
        if (items[i].id == item.id)
        {
            if (items[i].count < 128)
            {
                // Додаємо кількість предмета, якщо це можливо
                items[i].count += count;
                // Якщо кількість перевищує ліміт, обрізаємо
                if (items[i].count > 128)
                {
                    count = items[i].count - 128;
                    items[i].count = 64;
                }
            }
            else
            {
                count = 0;
                i = maxCount; // Виходимо з циклу
            }
        }
    }
    // Додаємо новий предмет до інвентаря, якщо він повністю не
    вміщується
    if(count > 0)
    {
        for (int i = 0; i < maxCount; i++)
        {
            if (items[i].id == 0)
            {
                AddItem(i, item, count);
                i = maxCount; // Виходимо з циклу
            }
        }
    }
}
```

Гравець може переглянути свій інвентар, натиснувши клавішу "I". Коли гравець натискає на предмет у вікні інвентаря, він може вибрати його для подальшого використання або переміщення. Якщо гравець натискає на

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

предмет, який вже вибрано, він знову стає доступним для вибору. Гравець також може використовувати предмети з інвентаря, натискаючи клавішу "U". При цьому викликається функція `UseCurrentItem()`, яка використовує обраний предмет з інвентаря. Якщо предмет використовується, його кількість зменшується на одиницю або він повністю видаляється з інвентаря, якщо його кількість становить одиницю.

Першою міні-грою (рисунок 2.21), що прийдеться пройти гравцю є гра, що реалізує ідею музичної аркади, де гравець повинен натискати відповідні клавіші у відповідь на появу нот на екрані. Для досягнення цієї мети використовується низка скриптів з додатковою бібліотекою для роботи з MIDI-файлами - Melanchall.DryWetMidi [13].

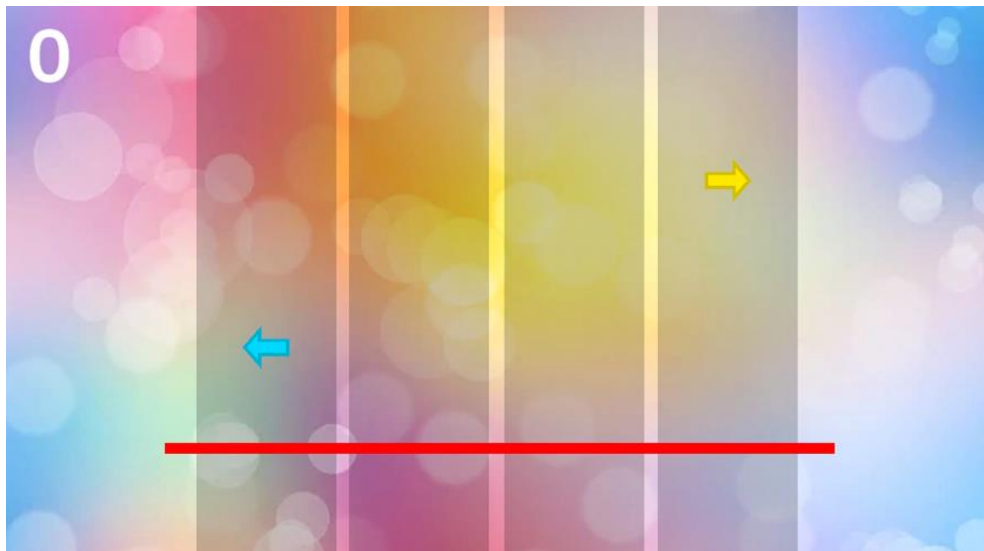


Рисунок 2.21 – Міні-гра «музична аркада»

Основні компоненти гри:

- *Lane*: Цей клас відповідає за кожну "смугу" нот, що з'являються на екрані. Він визначає, які ноти будуть відображатися на смугі, якими кнопками користувач може їх відтворювати, та контролює створення та знищення нот на сцені.

```
public void SetTimeStamps(Melanchall.DryWetMidi.Interaction.Note[]
array)
{
    foreach (var note in array)
    {
        // Перевіряємо, чи нота відповідає обмеженню за назвою ноти
        if (note.NoteName == noteRestriction)
        {
```

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

```

        // Перетворюємо час ноти у метричний час
        var metricTimeSpan =
TimeConverter.ConvertTo<MetricTimeSpan>(note.Time,
SongManager.midiFile.GetTempoMap());
        // Додаємо момент часу ноти у список часів для цієї
доріжки
        timeStamps.Add((double)metricTimeSpan.Minutes * 60f +
metricTimeSpan.Seconds + (double)metricTimeSpan.Milliseconds / 1000f);
    }
}

```

Цей метод додає моменти часу, коли ноти повинні з'явитися на доріжці, у список часів для цієї доріжки на основі нот з міді-файлу.

- *Note*: Цей клас представляє окрему ноту, що з'являється на екрані. Він відповідає за анімацію руху ноти вниз по екрану та її знищення при досягненні певної позиції.
- *ScoreManager*: Цей клас контролює рахунок гравця, обробляє правильні та неправильні натискання нот, відтворює звукові ефекти для правильних та неправильних натискань, а також відображає результати гри.
- *SongManager*: Цей клас відповідає за завантаження музичного файлу (MIDI) з різних джерел (локальний файл або з інтернету), аналізує його для отримання інформації про ноти та ритм, а також контролює відтворення музики та синхронізацію з нотами на екрані.

Кожна нота має відповідну "смугу" (Lane), а гравець має відтворювати ноти, використовуючи відповідні клавіші на клавіатурі. За кожну правильно відтворену ноту гравець отримує очки, а за неправильне відтворення - втрачає комбо. Ціль гравця - набрати певну кількість очок або завершити пісню без помилок. Гра використовує звукові ефекти та візуальну анімацію для створення ефекту взаємодії з музикою та нотами.

Наступною міні-грою (рисунок 2.22) є гра, що реалізує класичний геймплей "три в ряд", в якій гравець має обирати дві сусідні плитку для їх обміну місцями з метою створення ланцюга з трьох або більше однакових плиток у рядку чи стовпці, що призводить до їх зникнення та отримання балів [14].

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		34



Рисунок 2.22 – Міні-гра «три в ряд»

Основні класи гри та їх функції виглядають так:

- Tile: Клас, що представляє одну плитку на ігровому полі. Має координати x та y , зображення та тип. Відповідає за відображення та обмін різних типів плиток.
- TileData: Незмінна структура, що зберігає дані плитки: координати x та y , та ідентифікатор типу.
- TileDataMatrixUtility: Клас-утиліта для операцій з матрицею даних плиток, таких як обмін, пошук збігів та рухи. Містить методи для обміну плиток, пошуку збігів та найкращого руху.
- TileTypeAsset: Клас, що містить інформацію про типи плиток, включаючи їхні ідентифікатори, значення та зображення.
- Row: Клас, що представляє ряд плиток на ігровому полі.
- Move: Клас, що представляє рух, який може бути здійснений на ігровому полі.
- Match: Клас, що представляє збіг плиток, який можна здійснити на ігровому полі.
- Board: Основний клас, що керує грою. Містить методи для вибору та обміну плиток, перевірки наявності збігів та перемішування плиток.

// Метод для вибору плиток та спроби їх обміну

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

```

private async void Select(Tile tile)
{
    if (_isSwapping || _isMatching || _isShuffling) return;

    if (!_selection.Contains(tile))
    {
        if (_selection.Count > 0)
        {
            if (Math.Abs(tile.x - _selection[0].x) == 1 &&
Math.Abs(tile.y - _selection[0].y) == 0
                || Math.Abs(tile.y - _selection[0].y) == 1 &&
Math.Abs(tile.x - _selection[0].x) == 0)
                _selection.Add(tile);
        }
        else
        {
            _selection.Add(tile);
        }
    }

    if (_selection.Count < 2) return;

    await SwapAsync(_selection[0], _selection[1]);

    if (!await TryMatchAsync()) await SwapAsync(_selection[0],
_selection[1]);

    var matrix = Matrix;

    while (TileDataMatrixUtility.FindBestMove(matrix) == null ||
TileDataMatrixUtility.FindBestMatch(matrix) != null)
    {
        Shuffle();

        matrix = Matrix;
    }

    _selection.Clear();
}

// ...

// Метод для спроби обміну плиток та перевірки збігів
private async Task<bool> TryMatchAsync()
{
    var didMatch = false;

    _isMatching = true;

    var match = TileDataMatrixUtility.FindBestMatch(Matrix);

    while (match != null)
    {
        didMatch = true;

        var tiles = GetTiles(match.Tiles);

```

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

```

        var deflateSequence = DOTween.Sequence();

        foreach (var tile in tiles)
            deflateSequence.Join(tile.icon.transform.DOScale(Vector3.zero,
            tweenDuration).SetEase(Ease.InBack));

        audioSource.PlayOneShot(matchSound);

        await deflateSequence.Play().AsyncWaitForCompletion();

        var inflateSequence = DOTween.Sequence();

        foreach (var tile in tiles)
        {
            tile.Type = tileTypes[Random.Range(0,
            tileTypes.Length)];

            inflateSequence.Join(tile.icon.transform.DOScale(Vector3.one,
            tweenDuration).SetEase(Ease.OutBack));
        }

        await inflateSequence.Play().AsyncWaitForCompletion();

        OnMatch?.Invoke(Array.Find(tileTypes, tileType =>
        tileType.id == match.TypeId), match.Tiles.Length);

        match = TileDataMatrixUtility.FindBestMatch(Matrix);
    }

    _isMatching = false;

    return didMatch;
}

// ...

// Метод для перемішування плиток на дошці
private void Shuffle()
{
    _isShuffling = true;

    foreach (var row in rows)
        foreach (var tile in row.tiles)
            tile.Type = tileTypes[Random.Range(0,
            tileTypes.Length)];

    _isShuffling = false;
}

```

Для реалізації бойової системи в грі було реалізовано карткову гру (рисунки 2.23), що базується на протистоянні гравця та ігрового інтелекту. На початку гри учасникам роздаються початкові колоди з 10 випадкових карт. Гравці ходять по черзі, тривалість одного ходу 30 секунд. Після завершення

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

свого ходу гравець має натиснути відповідну кнопку. Якщо час ходу закінчився, хід автоматично переходить до наступного гравця. На ігровому полі можна розмістити максимум 6 карт. Атакувати картою, яка щойно була розміщена на полі, можна лише під час наступного ходу (крім карти моментальної атаки). Карти, які можуть атакувати, виділяються зеленим кольором. Коли одна карта атакує іншу, вона також пошкоджує себе. Кожен хід вся витрачена мана гравця відновлюється. Карти в руках, на які немає достатньо мани для використання, виділені сірим кольором. Картки із заклинаннями, що мають ціль, виділяють їх фіолетовим. Гра закінчується, якщо в одного з героїв не залишилося здоров'я або закінчилися ходи [15-16].

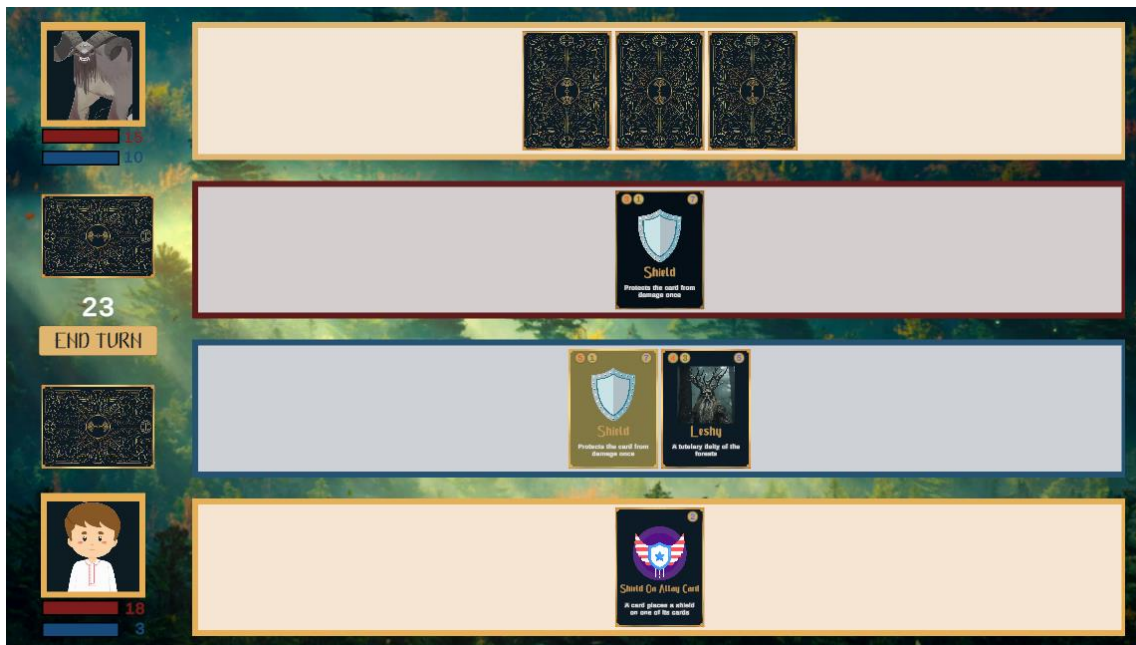


Рисунок 2.23 – Бойова система через карткову гру з ігровим інтелектом

Є такі типи карт зі здібностями:

- INSTANT_ACTIVE, // карта активна одразу після касту
- DOUBLE_ATTACK, // подвійна атака
- SHIELD, // один раз оберігає карту
- PROVOCATION, // бити інші карти не можна, окрім цієї
- REGENERATION_EACH_TURN, // +2хп кожен хід
- COUNTER_ATTACK // карта, що атакує її, отримує шкоду двічі

А також такі карти-закляття:

- HEAL_ALLY_FIELD_CARDS, // лікування всіх своїх карт на полі
- DAMAGE_ENEMY_FIELD_CARDS, // шкода всім картам ворога на полі
- HEAL_ALLAY_HERO, // лікування свого героя
- DAMAGE_ENEMY_HERO, // шкода герою ворога
- HEAL_ALLAY_CARD, // лікування своєї карти
- DAMAGE_ENEMY_CARD, // шкода карті ворога
- SHIELD_ON_ALLAY_CARD, // щит на свою карту

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

- PROVOCATION_ON_ALLAY_CARD, // провокація на свою карту
- BUFF_CARD_DAMAGE, // збільшення шкоди своєї карти
- DEBUFF_CARD_DAMAGE // зменшення шкоди карти ворога

Ігровий інтелект (Game AI) у грі використовується для керування поведінкою ворога. Основні кроки, які робить AI під час свого ходу, можна описати наступним чином:

1. *Вибір карти для гри:* AI обирає кілька карт зі своєї руки, які можна розмістити на ігровому полі, враховуючи ресурс мани.
2. *Розміщення карти на ігровому полі:* Якщо обрана карта - це заклинання, AI визначає, чи є наявні цільові об'єкти для заклинання (якщо вона передбачає цільовий об'єкт). Після цього AI розміщує обрану карту на ігровому полі.
3. *Атака:* Якщо на полі є картки, які можна атакувати, AI обирає карту, яку атакувати. Якщо на полі немає ворожих карт, AI обирає атакувати героя.
4. *Перевірка наявності можливості завершення ходу:* Після розміщення карт і можливої атаки AI перевіряє, чи закінчився вже час на цей хід. Якщо час на хід вичерпаний, хід завершується автоматично.
5. *Зміна ходу:* AI передає хід гравцю і очікує наступного свого ходу.

Ці кроки виконуються в циклі, доки гра не завершиться.

Під час битви з босом гравець зіткнеться з типовою грою на пам'ять (memory game), де гравець повинен знайти пари карт за обмежений час (рисунки 2.24). Основний механізм гри полягає у відкритті та порівнянні карток, щоб знайти пари. Гра починається зі змішування карток та їх розкладання на ігровому полі. Кожна картка має дві сторони: зображення і задню сторону (спільний дизайн для всіх карток). Клас gameManager відповідає за загальну логіку гри, включаючи ініціалізацію карток, обробку часу та відстеження стану гри. Клас cardScript управляє поведінкою окремих карток. Метод flipcard змінює стан картки між відкритою (_state == 1) і закритою (_state == 0) та змінює відповідно її зображення [17].

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		39



Рисунок 2.24 – Міні-гра на пам'ять

Під час натискання миші для вибору карток спрацьовує метод checkCards і порівнює їх:

```
void checkCards()
{
    List<int> c = new List<int>();

    for (int i = 0; i < cards.Length; i++)
    {
        if (cards[i].GetComponent<cardScript>().state == 1)
            c.Add(i);
    }

    if (c.Count == 2)
        cardComparison(c);
}

void cardComparison(List<int> c)
{
    cardScript.DO_NOT = true;

    int x = 0;

    if (cards[c[0]].GetComponent<cardScript>().cardValue ==
        cards[c[1]].GetComponent<cardScript>().cardValue)
    {
        x = 2;
        _matches--;
        if (_matches == 0 && targetTime > 0)
        {
            resultObj.SetActive(true);
            result.text = "Good job";
            PlayerPrefs.SetInt("isMemoryGameDone", 1);
            continueBtn.SetActive(true);
        }
    }
}
```

```

for (int i = 0; i < c.Count; i++)
{
    cards[c[i]].GetComponent<cardScript>().state = x;
    cards[c[i]].GetComponent<cardScript>().falseCheck();
}
}

```

Цей метод перевіряє, чи значення карток збігаються. Якщо так, вони залишаються відкритими, і кількість залишених пар зменшується. Якщо всі пари знайдено до закінчення часу, відображається повідомлення про перемогу. Якщо значення карток не збігаються, картки закриваються знову через певний час.

Останньою міні-грою під час битви з босом є лабіринт (рисунок 2.25), у якому гравець повинен знайти вихід, керуючи персонажем та рухаючись через різні клітинки. Для побудови лабіринту було використано процедурну генерацію [18].

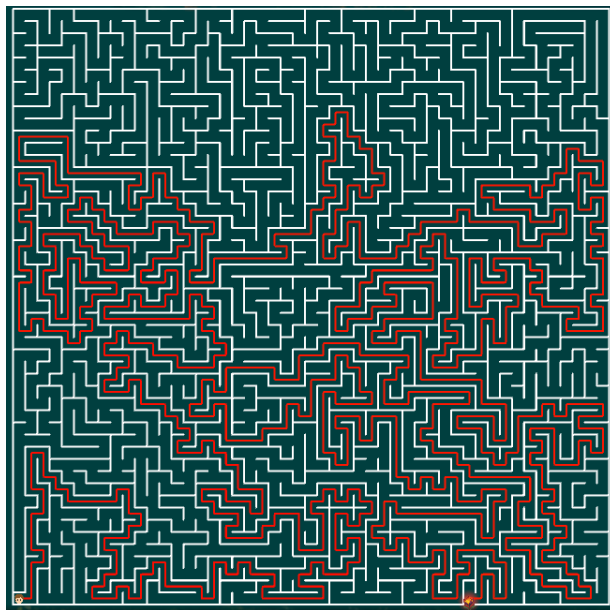


Рисунок 2.25 – Міні-гра «Лабіринт»

Лабіринт генерується методом GenerateMaze, де ініціалізуються клітинки, видаляються стіни між ними, і встановлюється вихід.:

```

public Maze GenerateMaze(GameObject exitPrefab)
{
    MazeGeneratorCell[,] cells = new MazeGeneratorCell[Width,
Height];

    for (int x = 0; x < cells.GetLength(0); x++)
    {
        for (int y = 0; y < cells.GetLength(1); y++)
        {

```

```

        cells[x, y] = new MazeGeneratorCell {X = x, Y = y};
    }
}

for (int x = 0; x < cells.GetLength(0); x++)
{
    cells[x, Height - 1].WallLeft = false;
}

for (int y = 0; y < cells.GetLength(1); y++)
{
    cells[Width - 1, y].WallBottom = false;
}

RemoveWallsWithBacktracker(cells);

Maze maze = new Maze();

maze.cells = cells;
maze.finishPosition = PlaceMazeExit(cells);

// Створюємо вихід у вказаній позиції
CreateExit(maze.finishPosition, exitPrefab);

return maze;
}

```

Лабіринт відображається за допомогою MazeSpawner, який створює об'єкти Cell для кожної клітинки та активує відповідні стіни. Гравець керується за допомогою PlayerControls, який обробляє натискання клавіш для переміщення персонажа і перевіряє, чи досягнув гравець виходу.

Висновки до другого розділу

В другому розділі було створено елементи графічного дизайну, такі як фонові зображення, персонажі, ігрові об'єкти та анімація для подальшого використання у проєкті. Архітектура ігрового додатку була представлена за допомогою діаграм варіантів використання, активності та класів. Використання патернів проектування, таких як Одинак (Singleton), Спостерігач (Observer) та Фасад (Facade), забезпечило гнучкість та розширюваність системи. Фізичну модель системи відображено за допомогою діаграм компонентів та розгортання, надає уявлення взаємодію між компонентами на рівні виконання. Також описано реалізовані основні механіки гри, такі як рух персонажа, інвентар, міні-ігри (музична аркада, "три в ряд", карткова гра, гра на пам'ять та лабіринт).

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1 Порядок встановлення та налаштування системи

Після побудови релізної версії, гру "Spirit of the forest" можна буде завантажити з цифрових платформ. Доступні версії для Windows, macOS та Linux. Детальніше про встановлення та налаштування гри можна буде дізнатися в інструкції користувача.

Для коректної роботи програми важливо враховувати вимоги до апаратного забезпечення. Мінімальні та рекомендовані параметри, які варто врахувати при виборі комп'ютерної конфігурації, було зведено до таблиці 3.1 та 3.2.

Таблиця 3.1 – Мінімальні вимоги до апаратної частини

Процесор	Intel Core i5-4460 3.2 GHz or AMD FX-8350 4.0 GHz
Оперативна пам'ять	8 GB RAM
Графіка	NVIDIA GeForce GTX 660 or AMD Radeon HD 7850 (2 GB VRAM)
Пам'ять	50 GB
DirectX	Version 11
Операційна система	Windows 7, macOS 10.13 High Sierra, Linux Ubuntu 16.04

Таблиця 3.2 – Рекомендовані вимоги до апаратної частини

Процесор	Intel Core i7-8700 3.2 GHz or AMD Ryzen 5 2600 3.4 GHz
Оперативна пам'ять	16 GB RAM
Графіка	NVIDIA GeForce GTX 1060 6GB or AMD Radeon RX 580 8GB
Пам'ять	50 GB

DirectX	Version 12
Операційна система	Windows 10, macOS 11 Big Sur, Linux Ubuntu 20.04

3.2 Інтерфейс та порядок роботи з програмним додатком

Після запуску гри гравець опиняється в головному меню (рисунок 3.1), де можна розпочати нову гру (після цього відтвориться кат-сцена), продовжити стару (якщо наявні збереження), змінити налаштування (рисунок 3.2), переглянути свої досягнення (рисунок 3.3) та вийти з гри. Налаштування включають в себе повноекранний режим, гучність звуків та музики, якість картинки та розширення екрану. При наведенні на досягнення знизу з'являється його опис. Якщо гравець отримав його, то іконка стає кольоровою, інакше вона сіра.



Рисунок 3.1 – Головне меню гри

Перша ігрова локація (рисунок 3.4) являє собою село, де гравцю треба прийняти участь в місцевих забавах напередодні Івана Купала: співи, стрибки через багаття та збирання квітів для плетіння віночків. Під час цього його супроводжує однак з місцевих жительок, яка надає інформацію через систему діалогів. В цьому, як і у всіх інших рівнях, управління персонажем здійснюється за допомогою кнопок A та D для ходьби та Space для стрибка (про

це повідомляється у підказках). Спочатку гравцю потрібно виграти міні-гру «музична аркада» (див. рисунок 2.21), де він повинен вчасно натискати відповідні клавіші у відповідь на появу нот на екрані та досягнути комбо з п'яти нот. Потім гравець має перестрибнути багаття, що наносить шкоду гравцю у разі зіткнення. Останньою міні-грою в цій локації є «три в ряд» (див. рисунок 2.22), де потрібно за хвилину назбирати 100 балів, переставляючи сусідні плитки так, щоб утворити ланцюг з трьох або більше однакових плиток у рядку чи стовпці.



Рисунок 3.2 – Налаштування гри



Рисунок 3.3 – Досягнення гравця

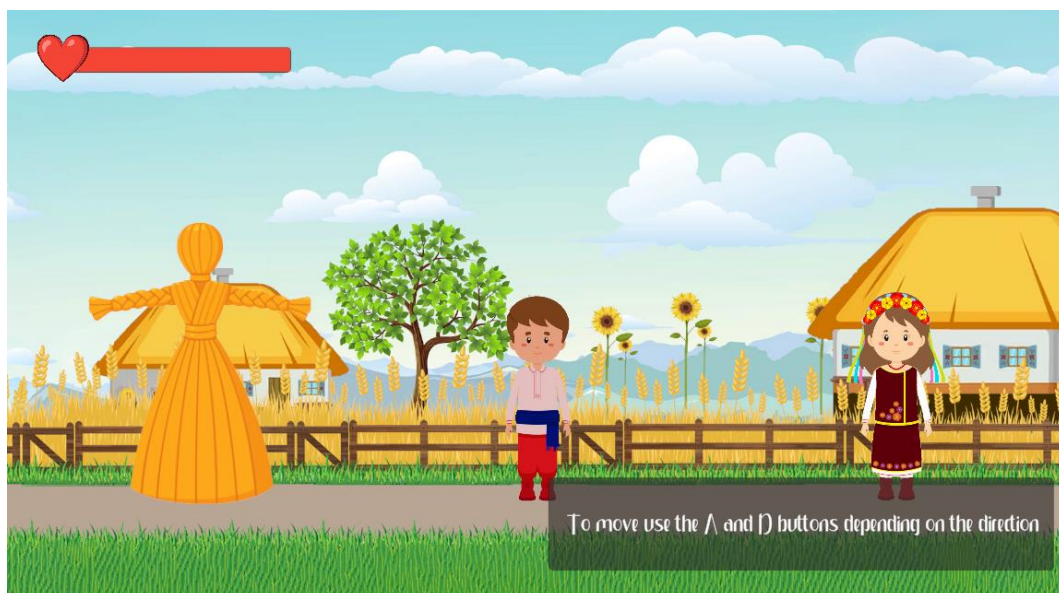


Рисунок 3.4 – Перша ігрова локація

Далі переносимось до другої локації – річки (рисунок 3.5). Тут гравця чекає останній діалог з селянкою, яка на прощання подарує віночок з квітів та діє два шматочки мапи. Підібравши його, віночок з'являється в інвентарю, що викликається при натисканні клавіші I. Щоб використати його потрібно перш за все вибрати відповідний предмет в інвентарю, та натиснути клавішу U (застосовується не тільки для віночка, а й для інших предметів). Після цього віночок попливе по воді, що призведе до появи русалки (рисунок 3.6), до якої можна дібратись плаваючи. Діалог з русалкою призведе до карткової битви, після перемоги над якою отримуємо ще два шматки мапи.

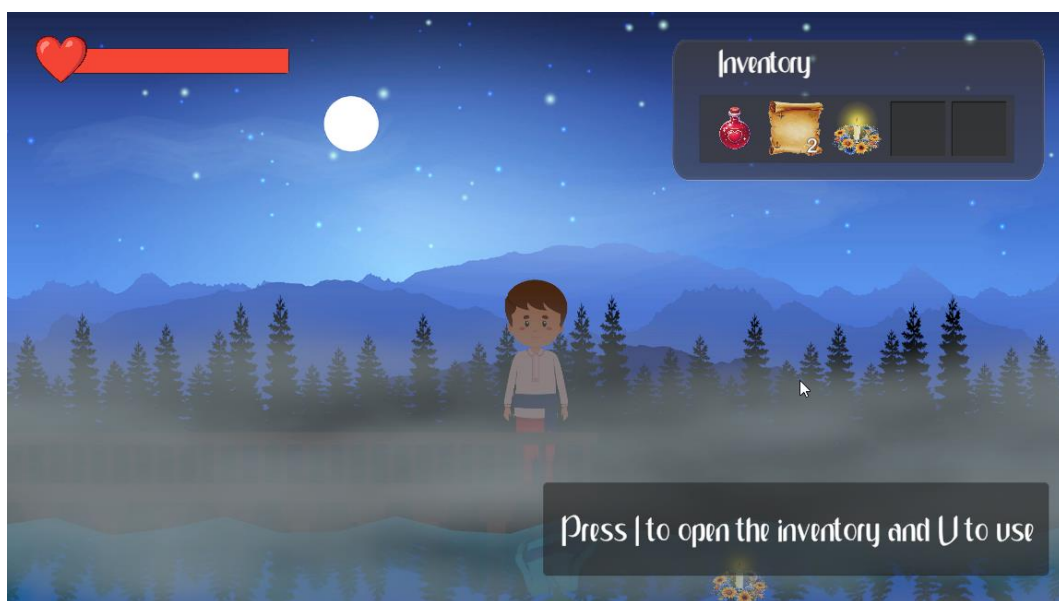


Рисунок 3.5 – Друга ігрова локація

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		46



Рисунок 3.6 – Гравець пливе до русалки

І нарешті опиняємось в останній локації – темний ліс (рисунок 3.7). Тут гравця зустрічає лісовик та його духи-помічники, що також викликає гравця на поєдинок та в якості нагороди має шматок мапи. Рухаючись далі на шляху гравця з’являється камінь з рунами (рисунок 3.8), де потрібно обрати з дванадцяти три та поставити їх у відповідні комірки. Щоб дізнатись правильну комбінацію слід у лісі натиснути на сову, що сидить на дереві. З неї випаде сувій, відкривши який гравець отримає підказку.

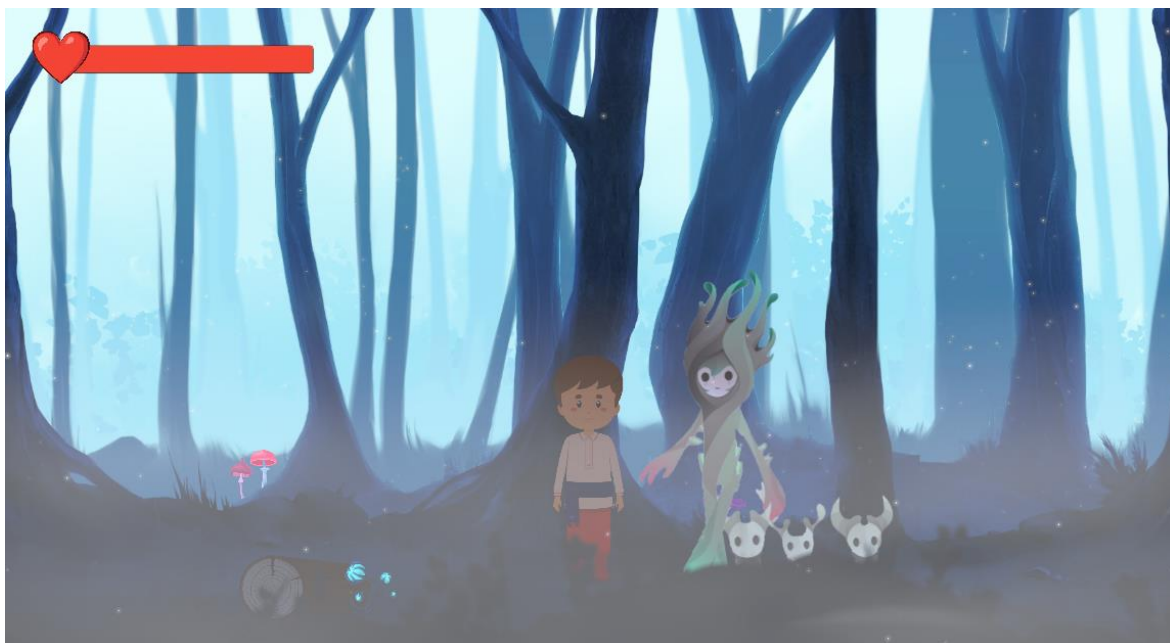


Рисунок 3.7 – Третя ігрова локація

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

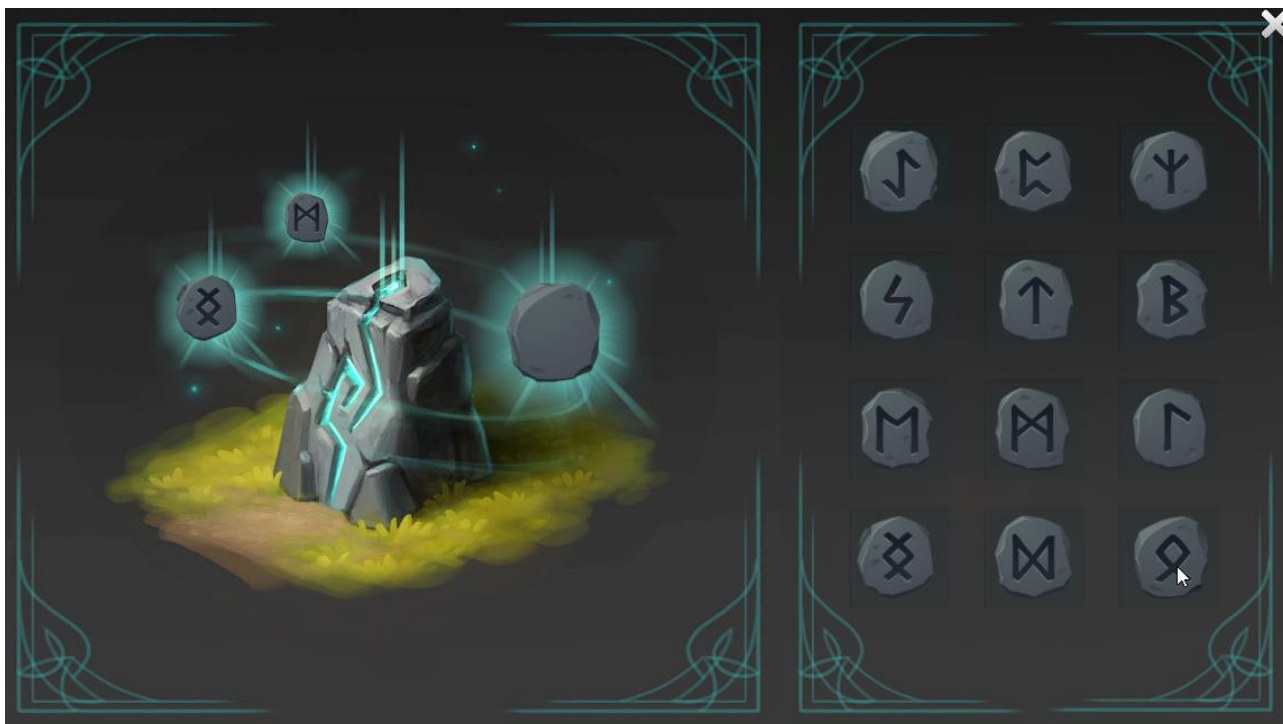
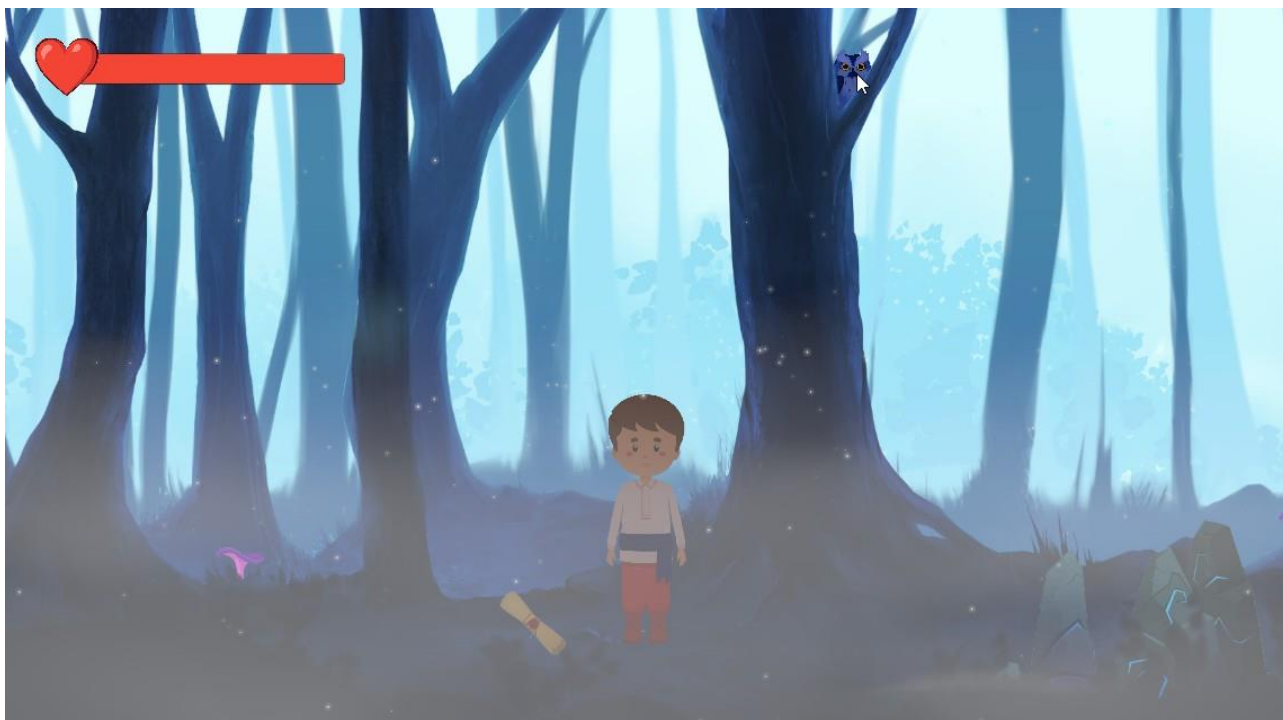


Рисунок 3.8 – Загадка з рунами

Виконавши це можна рухатись далі. На шляху герой також зустрічає мавку, на яку полює чугайстер. Гравцеві надається можливість вибрати сторону: допомогти мавці або стати на бік чугайстра (рисунок 3.9). Цей вибір впливає веде до карткової битви з обраним персонажем. В результаті забираємо останній шматочок мапи, та можна йти до боса.

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		48



Рисунок 3.9 – Діалог з мавкою та чугайстром

Битва з босом (рисунок 3.10) складається з трьох етапів: карткової гри; гри на запам'ятовування, де гравець повинен знайти пари карт за обмежений час (див. рисунок 2.24); а також лабіринту, де гравець повинен знайти вихід, керуючи персонажем (див. рисунок 2.25). Після перемоги над чортом, герой нарешті здобуває квітку папороті та гра завершується.

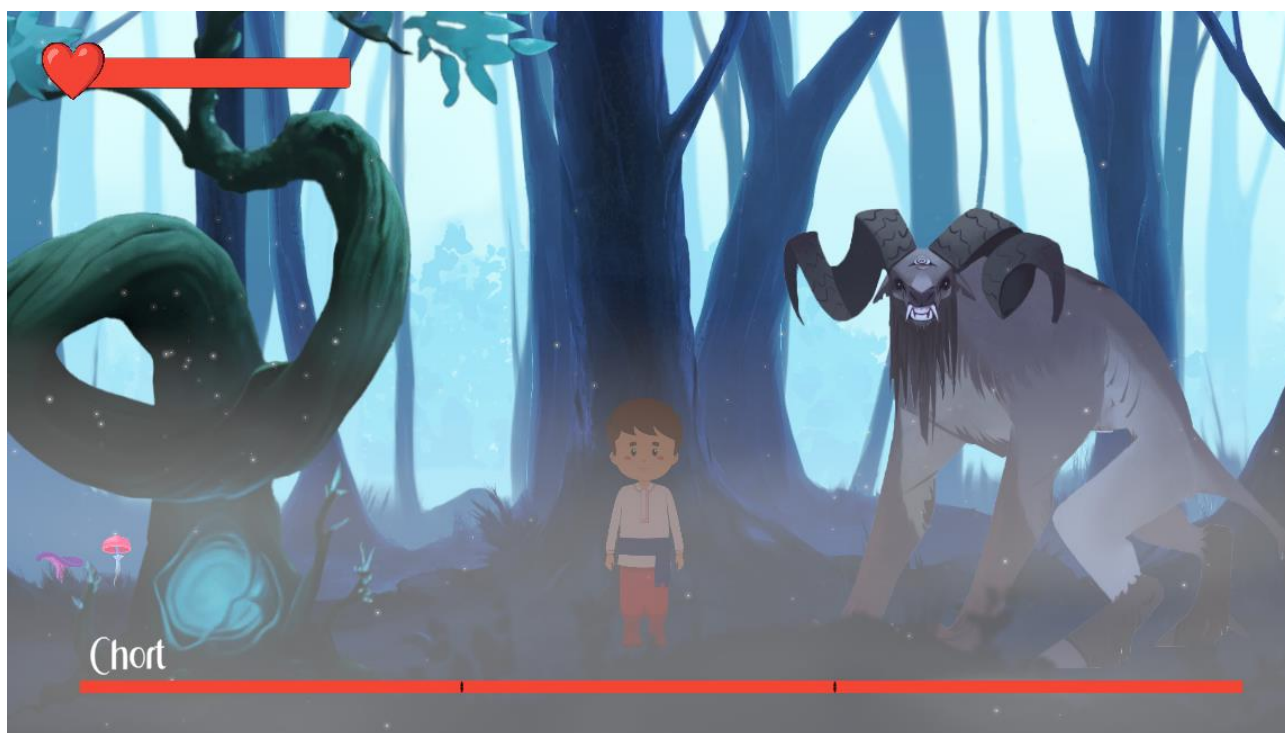


Рисунок 3.10 – Битва з чортом

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		49



Рисунок 3.11 – Бестиарій



Рисунок 3.12 – Зібранні шматки мапи

Додатково в ході гри гравець може переглянути бестиарій (рисунок 3.11), де розповідається детально про створінь, що зустрічаються в грі (відкривається при натисканні клавіші В); та переглянути зібранні шматки мапи за допомогою клавіші М (рисунок 3.12).

3.3 Тестування роботи програмного забезпечення

Проведено тестування гри для виявлення та усунення недоліків [19-20]. Протестовані пункти, було представлено у вигляді таблиці 3.3.

Таблиця 3.3 – Результат тестування гри

Сцена	Вхідні дані	Очікувані дані	Спостережені дані
Головне меню	Натискання кнопки "Нова гра"	Відтворення кат-сцени, початок нової гри	Як і очікувалося
Головне меню	Натискання кнопки "Продовжити"	Завантаження останнього збереження	Як і очікувалося
Головне меню	Натискання кнопки "Налаштування"	Відкриття меню налаштувань	Як і очікувалося
Головне меню	Натискання кнопки "Досягнення"	Відкриття списку досягнень	Як і очікувалося
Головне меню	Натискання кнопки "Вийти"	Вихід з гри	Як і очікувалося
Головне меню - Налаштування	Перемикання на повноекранний режим	Відображення гри у повноекранному режимі	Як і очікувалося
Головне меню - Налаштування	Зміна гучності	Зміна рівня гучності у грі	Як і очікувалося
Головне меню - Налаштування	Зміна якості зображення	Зміна якості зображення	Як і очікувалося

Головне меню - Налаштування	Зміна розширення екрану	Зміна розширення екрану	Як і очікувалося
Головне меню - Досягнення	Наведення на досягнення	Поява опису досягнення	Як і очікувалося
Головне меню - Досягнення	Зміна кольору відповідно до стану	При отриманні досягнення іконка стає кольоровою, інакше - сіра	Як і очікувалося
Інтро	Перегляд кат-сцени	Кат-сцена відтворюється без помилок	Як і очікувалося
Село	Натискання клавіш A та D для ходьби	Персонаж ходить вліво або вправо	Як і очікувалося
Село	Натискання клавіші Space для стрибка	Персонаж стрибає	Як і очікувалося
Село	Відображення підказок	Відображення підказок для користування грою	Як і очікувалося
Село	Розмова з селянкою	Діалог з селянкою відбувається коректно	Як і очікувалося

Село	Гравець потрапив в багаття	Нанесення шкоди здоров'ю гравця	Як і очікувалося
Село	Здоров'я гравця менше або дорівнює нулю	Відображення «екрану смерті»	Як і очікувалося
Музична аркада	Показ правил гри	На початку гри на деякий час відображаються правила	Як і очікувалося
Музична аркада	Вчасне натискання клавіш, що відповідають нотам	Зароблення комбо	Як і очікувалося
Музична аркада	Пропуск однієї ноти	Втрата заробленого комбо	Як і очікувалося
Музична аркада	Досягнення комбо з п'яти нот	Зарахування перемоги	Як і очікувалося
Музична аркада	Закінчення пісні	Зарахування поразки	Як і очікувалося
Три в ряд	Показ правил гри	На початку гри на деякий час відображаються правила	Як і очікувалося
Три в ряд	Правильна перестановка плиток	Плитки утворюють ланцюжок та зникають,	Як і очікувалося

		зараховуючи бали	
Три в ряд	Неправильна перестановка плиток	Плитки повертаються на свої місця	Як і очікувалося
Три в ряд	Закінчення часу	Зарахування поразки	Як і очікувалося
Три в ряд	Набрано 100 очків	Зарахування перемоги	Як і очікувалося
Річка	Розмова з селянкою	Діалог з селянкою відбувається коректно	Як і очікувалося
Річка	Підбір віночка	Віночок з'являється в інвентарю	Як і очікувалося
Річка	Отримання шматків мапи від селянки	Шматки мапи з'являються в інвентарю	Як і очікувалося
Річка	Використання віночка	Віночок пливе по воді, з'являється русалка	Як і очікувалося
Річка	Персонаж стрибає у річку	При потраплянні у воду персонаж плаває	Як і очікувалося
Річка	Розмова з русалкою	Діалог з русалкою відбувається коректно	Як і очікувалося

Річка	Перемога в картковій битві над русалкою	Отримання шматків мапи	Як і очікувалося
Ліс	Розмова з лісовиком	Діалог з лісовиком відбувається коректно	Як і очікувалося
Ліс	Перемога в картковій битві над лісовиком	Отримання шматків мапи	Як і очікувалося
Ліс	Гравець потрапив у кропиву	Нанесення шкоди здоров'ю гравця	Як і очікувалося
Ліс	Натиснення на сову	Поява сувія з підказкою	Як і очікувалося
Ліс	Введення правильної комбінації рун	Гравець може пройти далі	Як і очікувалося
Ліс	Зустріч з мавкою	Діалог з мавкою та чугайстром	Як і очікувалося
Ліс	Гравець обрав допомогти мавці	Початок карткової битви з чугайстром	Як і очікувалося
Ліс	Гравець обрав допомогти чугайстру	Початок карткової битви з мавкою	Як і очікувалося

Ліс	Перемога в картковій битві над мавкою/чугайстром	Отримання шматків мапи	Як і очікувалося
Ліс	Гравець зібрав всі шматки мапи	Гравець може зустрітись з босом	Як і очікувалося
Ліс	Перемога у картковій грі над чортом	Перехід до гри на запам'ятовування	Як і очікувалося
Ліс	Перемога у memory game	Перехід до лабіринту	Як і очікувалося
Ліс	Знаходження виходу з лабіринту	Завершення гри, отримання квітки папороті	Як і очікувалося
Memory game	Закінчення часу	Зарахування поразки	Як і очікувалося
Memory game	Знайдено всіх пар	Зарахування перемоги	Як і очікувалося
Memory game	Натискання на карту	Перевертання її на лицьову сторону	Як і очікувалося
Memory game	Знайдено пару однакових карток	Картки залишаються лицьовою стороною доверху	Як і очікувалося

Memory game	Обрана пара різних карток	Перевертання їх лицьовою стороною донизу	Як і очікувалося
Лабіринт	Натискання стрілочок вверх, вниз, вліво, право	Рух персонажу у відповідному напрямку	Як і очікувалося
Лабіринт	Досягнення виходу з лабіринту	Зарахування перемоги	Як і очікувалося
Всі локації	Натискання клавіші В	Відкриття бестіарію	Як і очікувалося
Всі локації	Натискання клавіші М	Відображення зібраних шматків мапи	Як і очікувалося
Всі локації	Натискання клавіші І	Відкривається інвентар гравця	Як і очікувалося
Всі локації	Натискання клавіші U після вибору предмета в інвентарі	Використання предмету	Як і очікувалося

Висновки до третього розділу:

У третьому розділі описано порядок встановлення та налаштування гри "Spirit of the forest", зазначено мінімальні та рекомендовані системні вимоги для коректної роботи гри. Також було описано інтерфейс гри та порядок роботи з нею. Інтерфейс гри інтуїтивно зрозумілий та зручний. Детально описані основні ігрові механіки та міні-ігри, які гравець може зустріти під час проходження гри. Також в межах розділу було проведено тестування гри, що охопило різні аспекти та допомогло виявити та усунути недоліки.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено комп'ютерну пригодницьку гру «Spirit of the forest», що базується на українській міфології та фольклорі, відповідно до обраної тематики. Процес розробки такої гри вимагає значних творчих та інтелектуальних зусиль, а також певного обсягу знань і навичок у різних галузях.

В першому розділі визначено завдання кваліфікаційної роботи та послідовні етапи розробки гри «Spirit of the Forest». Проведено аналіз існуючого програмного забезпечення, що допомогло визначити ключові аспекти для власної розробки. Створено концепт-документ гри, який описує жанр, цільову аудиторію, геймплей, сюжет та платформу. Обґрунтовано вибір інструментів для розробки, таких як Unity, C#, Visual Studio, Adobe Photoshop, DragonBones, та FL Studio.

В другому розділі розроблено елементи графічного дизайну, включаючи фонові зображення, персонажів, ігрові об'єкти та анімацію. Представлено архітектуру ігрового додатку за допомогою діаграм варіантів використання, активності та класів. Використання патернів проектування (Одинак, Спостерігач, Фасад) забезпечило гнучкість та розширюваність системи. Реалізовано основні механіки гри, такі як рух персонажа, інвентар, міні-ігри.

У третьому розділі описано порядок встановлення та налаштування гри, зазначено мінімальні та рекомендовані системні вимоги. Детально розглянуто інтерфейс гри та порядок роботи з нею. Проведено тестування, яке охопило різні аспекти гри та допомогло виявити та усунути недоліки..

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Last looks: The global games market in 2023 [Електронний ресурс] // Newzoo International B.V.. – 2024. – Режим доступу до ресурсу: <https://newzoo.com/resources/blog/last-looks-the-global-games-market-in-2023>. – Дата звернення: 16.05.2024.
2. How to start making a game (a guide to planning your first project) [Електронний ресурс] // gamedevbeginner. – 2022. – Режим доступу до ресурсу: <https://gamedevbeginner.com/how-to-start-making-a-game-a-guide-to-planning-your-first-project/>. – Дата звернення: 22.01.2024.
3. Machinarium Review [Електронний ресурс] // IGN. – 2016. – Режим доступу до ресурсу: <https://www.ign.com/articles/2012/09/18/machinarium-review-2>. – Дата звернення: 28.01.2024.
4. The Witcher 3 Review [Електронний ресурс] // IGN. – 2021. – Режим доступу до ресурсу: <https://www.ign.com/articles/2015/05/12/the-witcher-3-the-wild-hunt-review>. – Дата звернення: 03.02.2024.
5. Hearthstone: Heroes of Warcraft Review [Електронний ресурс] // IGN. – 2021. – Режим доступу до ресурсу: <https://www.ign.com/articles/2014/03/13/hearthstone-heroes-of-warcraft-review>. – Дата звернення: 05.02.2024.
6. Все що треба знати про геймдизайн-документацію. Поради спеціаліста [Електронний ресурс] // DOU.ua. – 2023. – Режим доступу до ресурсу: <https://gamedev.dou.ua/blogs/all-about-game-design-documentation/>. – Дата звернення: 17.02.2024.
7. Unity vs Unreal: Two Main Game Engines Compared by Game-Ace [Електронний ресурс] // Game-Ace. – 2023. – Режим доступу до ресурсу: <https://game-ace.com/blog/unity-vs-unreal/>. – Дата звернення: 20.02.2024.
8. Побудова діаграм варіантів використання (Usecase Diagrams) [Електронний ресурс] // Таврійський державний агротехнологічний університет. – 2023. – Режим доступу до ресурсу:

		Бісмська А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

- <http://www.tsatu.edu.ua/kn/wp-content/uploads/sites/16/laboratorna-robota5-diahramy-variantiv-vykorystannja.pdf>. – Дата звернення: 10.03.2024.
9. Class Diagram | Unified Modeling Language (UML) [Електронний ресурс] // GeeksforGeeks. – 2024. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/>. – Дата звернення: 14.03.2024.
10. Патерни проектування [Електронний ресурс] // Refactoring.Guru. – 2021. – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns>. – Дата звернення: 25.03.2024.
11. Unity 2D Parallax Background Effect in 100 Seconds [Електронний ресурс] // YouTube. – 2023. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=MEy-kIGE-II&ab_channel=Raycastly. – Дата звернення: 02.04.2024.
12. Let's make an Inventory System! [Електронний ресурс] // YouTube. – 2022. – Режим доступу до ресурсу: https://www.youtube.com/playlist?list=PLn1X2QyVjFVBM7Gr_pMhVt3-7rlY5hkx. – Дата звернення: 07.04.2024.
13. How to make a Rhythm Game In Unity (using MIDI) [Електронний ресурс] // YouTube. – 2022. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=ev0HsmgLScg&ab_channel=SkyanSam-SkyriftStudios. – Дата звернення: 12.04.2024.
14. Creating a Match 3 Game in Unity [Електронний ресурс] // YouTube. – 2022. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=9i_sktmLg30&ab_channel=WinterboltGames. – Дата звернення: 18.04.2024.
15. Карткова Гра на Unity [Електронний ресурс] // YouTube. – 2020. – Режим доступу до ресурсу: <https://www.youtube.com/playlist?list=PL0isw1lWHi7v1Nz91eGEnqDIAI8y9dgoN>. – Дата звернення: 22.04.2024.

		Бісмяка А.С.			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		Граф М.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

- 16.Exploring DoTween: Simplifying Animation in Unity [Електронний ресурс] // Medium. – 2023. – Режим доступу до ресурсу: <https://medium.com/@mertarar.ma/exploring-dotween-simplifying-animation-in-unity-1258c7c75d5a>. – Дата звернення: 25.04.2024.
- 17.Simple Memory Game using Unity [Електронний ресурс] // YouTube. – 2017. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=Nxt189rV08Q&ab_channel=PronayPeddiraju. – Дата звернення: 01.05.2024.
- 18Automatic Maze Generator in Unity [Електронний ресурс] // YouTube. – 2022. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=nkGwwfEPqj4&t=12s&ab_channel=WSALEvan. – Дата звернення: 11.05.2024.
- 19.How to Playtest your Game – Game Design Tips [Електронний ресурс] // Zenva Pty Ltd. – 2023. – Режим доступу до ресурсу: <https://gamedevacademy.org/game-playtest-tutorial/>. – Дата звернення: 21.05.2024.
- 20.Таблиці рішень та їх застосування в тестуванні [Електронний ресурс] // QATestLab. – 2023. – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/tables-solutions-and-their-usage-in-testing/?history=0&pfid=1&sample=679&ref=0>. – Дата звернення: 30.05.2024.

ДОДАТКИ

PlayerController.cs

```
using UnityEngine;
using UnityEngine.UI;

public enum PlayerState
{
    Idle,
    Walking,
    Jumping,
    Dead
}

public class PlayerController : MonoBehaviour
{
    public static PlayerController Instance { get; private set; }
    private PlayerState currentState = PlayerState.Idle;
    public float speed = 3f; // Швидкість руху
    public float jumpForce = 6f; // Сила стрибка
    private Animator animator;

    private Rigidbody2D rb;
    private bool isGrounded;
    private bool isJumping;

    public Image HealthPlayerBar;
    public float currentHP;
    const int MAX_HP = 100;
    private bool isDead = true;
    private float restoreHealthCount = 0.1f;
    public GameObject DeadGO;

    private void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(this);
        }
        else
        {
            Instance = this;
        }
    }

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();

        DeadGO.SetActive(false);
        currentHP = MAX_HP;
        SetCurrentPlayerHealth();

        // Відтворити анімацію спокою за замовчуванням
        ChangeState(PlayerState.Idle);
    }

    void Update()
    {

```

```

// Отримуємо вхід від гравця
float horizontalInput = Input.GetAxis("Horizontal");
float verticalInput = Input.GetAxis("Vertical");

// Рух по горизонталі та вертикалі
Vector2 movement = new Vector2(horizontalInput, verticalInput);
rb.velocity = new Vector2(movement.x * speed, rb.velocity.y);

// Визначення напрямку руху
if (movement.x > 0)
{
    transform.localScale = new Vector3(0.5f, 0.5f, 1); // Гравець
направлений вправо
}
else if (movement.x < 0)
{
    transform.localScale = new Vector3(-0.5f, 0.5f, 1); // Гравець
направлений вліво
}

// Візуалізація OverlapCircle
Debug.DrawLine(transform.position, transform.position + Vector3.down *
2f, Color.red);

// Перевірка, чи стоїть гравець на землі
isGrounded = Physics2D.OverlapCircle(transform.position, 2f,
LayerMask.GetMask("Ground"));

// Стрибок
if (isGrounded && Input.GetKeyDown(KeyCode.Space))
{
    Jump();
}

// Визначення стану ходьби та стрибка
isJumping = !isGrounded;

// Зміна анімації відповідно до стану ходьби та стрибка
if (isJumping)
{
    ChangeState(PlayerState.Jumping);
}
else if (movement.x != 0)
{
    ChangeState(PlayerState.Walking);
}
else
{
    ChangeState(PlayerState.Idle);
}

RestoreHealth();
}

void RestoreHealth()
{
    if (currentHP >= 0 && currentHP < MAX_HP)
    {
        currentHP += restoreHealthCount;
        SetCurrentPlayerHealth();
    }
}

```

```

void ChangeState(PlayerState newState)
{
    currentState = newState;

    switch (newState)
    {
        case PlayerState.Idle:
            animator.SetBool("IsWalking", false);
            animator.SetBool("IsJumping", false);
            animator.Play("boy_idle");
            break;
        case PlayerState.Walking:
            animator.SetBool("IsWalking", true);
            animator.SetBool("IsJumping", false);
            animator.Play("boy_walk");
            break;
        case PlayerState.Jumping:
            animator.SetBool("IsWalking", false);
            animator.SetBool("IsJumping", true);
            animator.Play("boy_jump");
            break;
        case PlayerState.Dead:
            // логіка для стану "Dead"
            break;
    }
}

public void DiePlayer()
{
    currentHP = 0;
    isDead = true;
    DeadGO.SetActive(true);
}

// Метод для стрибка
void Jump()
{
    rb.velocity = new Vector2(rb.velocity.x, jumpForce);
}

public void SetCurrentPlayerHealth()
{
    HealthPlayerBar.fillAmount = currentHP / MAX_HP;
}
}

```

AI.cs

```

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

public class AI : MonoBehaviour
{
    public void MakeTurn()
    {
        StartCoroutine(EnemyTurn(GameManagerScr.Instance.EnemyHandCards));
    }

    IEnumerator EnemyTurn(List<CardController> cards)
    {
        yield return new WaitForSeconds(1);

        int count = cards.Count == 1 ? 1 :
            Random.Range(0, cards.Count);

        for (int i = 0; i < count; i++)
        {
            if (GameManagerScr.Instance.EnemyFieldCards.Count > 5 ||
                GameManagerScr.Instance.CurrentGame.Enemy.Mana == 0 ||
                GameManagerScr.Instance.EnemyHandCards.Count == 0)
            {
                break;
            }

            List<CardController> cardsList = cards.FindAll(x =>
                GameManagerScr.Instance.CurrentGame.Enemy.Mana >= x.Card.Manacost);

            if (cardsList.Count == 0)
            {
                break;
            }

            if (cardsList[0] != null)
            {
                if (cardsList[0].Card.IsSpell)
                {
                    SpellCard spellCard = (SpellCard)cardsList[0].Card;

                    if ((spellCard.SpellTarget ==
                        SpellCard.TargetType.ALLAY_CARD_TARGET &&
                        GameManagerScr.Instance.EnemyFieldCards.Count == 0) ||
                        (spellCard.SpellTarget ==
                        SpellCard.TargetType.ENEMY_CARD_TARGET &&
                        GameManagerScr.Instance.PlayerFieldCards.Count == 0))
                    {
                        Debug.Log("Cannot use this spell as there are no targets
on the corresponding field. Choosing another card.");
                        ChooseAnotherCard();
                    }
                    else
                    {
                        Debug.Log("Casting spell " + cardsList[0].Card.Name);
                    }
                }
            }
        }
    }
}

```

```

        CastSpell(cardsList[0]);
    }
}
else
{
    CardMovementScr cardMovement =
cardsList[0].GetComponent<CardMovementScr>();

    if (cardMovement != null)
    {

cardMovement.MoveToField(GameManagerScr.Instance.EnemyField);
        yield return new WaitForSeconds(0.51f);

cardsList[0].transform.SetParent(GameManagerScr.Instance.EnemyField);
        cardsList[0].OnCast();
    }
    else
    {
        Debug.LogError("CardMovementScr component is null on
cardsList[0].");
    }
}
}
else
{
    Debug.LogError("cardsList[0] is null. Make sure it is properly
instantiated.");
}

/*cardsList[0].GetComponent<CardMovementScr>().MoveToField(GameManagerScr.Instance.EnemyField);

    yield return new WaitForSeconds(.51f);

cardsList[0].transform.SetParent(GameManagerScr.Instance.EnemyField);

    cardsList[0].OnCast();*/
}

yield return new WaitForSeconds(1);

while (GameManagerScr.Instance.EnemyFieldCards.Exists(x =>
x.Card.CanAttack))
{
    var activeCard = GameManagerScr.Instance.EnemyFieldCards.FindAll(x
=> x.Card.CanAttack)[0];
    bool hasProvocation =
GameManagerScr.Instance.PlayerFieldCards.Exists(x => x.Card.IsProvocation);

    if (hasProvocation || Random.Range(0, 2) == 0 &&
GameManagerScr.Instance.PlayerFieldCards.Count > 0)
    {
        CardController enemy;

        if (hasProvocation)
        {
            enemy = GameManagerScr.Instance.PlayerFieldCards.Find(x =>
x.Card.IsProvocation);

```

```

        }
        else
        {
            enemy =
GameManagerScr.Instance.PlayerFieldCards[Random.Range(0,
GameManagerScr.Instance.PlayerFieldCards.Count)];

        }

        Debug.Log(activeCard.Card.Name + " (" + activeCard.Card.Attack +
";" + activeCard.Card.Defense + ") " + "---->" + enemy.Card.Name + " (" +
enemy.Card.Attack + ";" + enemy.Card.Defense + ")");

        activeCard.Movement.MoveToTarget(enemy.transform);
        yield return new WaitForSeconds(.75f);

        GameManagerScr.Instance.CardsFight(enemy, activeCard);
    }
    else
    {
        Debug.Log(activeCard.Card.Name + " (" + activeCard.Card.Attack +
") Attacked hero");

        activeCard.GetComponent<CardMovementScr>().MoveToTarget(GameManagerScr.Instance.
PlayerHero.transform);
        yield return new WaitForSeconds(.75f);

        GameManagerScr.Instance.DamageHero(activeCard, false);
    }

    yield return new WaitForSeconds(.2f);
}

yield return new WaitForSeconds(1);
GameManagerScr.Instance.ChangeTurn();
}

void CastSpell(CardController card)
{
    switch (((SpellCard)card.Card).SpellTarget)
    {
        case SpellCard.TargetType.NO_TARGET:
            switch (((SpellCard)card.Card).Spell)
            {
                case SpellCard.SpellType.HEAL_ALLY_FIELD_CARDS:
                    if(GameManagerScr.Instance.EnemyFieldCards.Count > 0)
                    {
                        StartCoroutine(CastCard(card));
                    }

                    break;

                case SpellCard.SpellType.DAMAGE_ENEMY_FIELD_CARDS:
                    if (GameManagerScr.Instance.PlayerFieldCards.Count > 0)
                    {
                        StartCoroutine(CastCard(card));
                    }

                    break;

                case SpellCard.SpellType.HEAL_ALLAY_HERO:
                    StartCoroutine(CastCard(card));
            }
        }
    }
}

```

```

        break;

        case SpellCard.SpellType.DAMAGE_ENEMY_HERO:
            StartCoroutine(CastCard(card));
            break;
    }

    break;

    case SpellCard.TargetType.ALLAY_CARD_TARGET:
        if (GameManagerScr.Instance.EnemyFieldCards.Count > 0)
        {
            StartCoroutine(CastCard(card,
GameManagerScr.Instance.EnemyFieldCards[Random.Range(0,
GameManagerScr.Instance.EnemyFieldCards.Count)]));
        }

        break;

    case SpellCard.TargetType.ENEMY_CARD_TARGET:
        if (GameManagerScr.Instance.PlayerFieldCards.Count > 0)
        {
            StartCoroutine(CastCard(card,
GameManagerScr.Instance.PlayerFieldCards[Random.Range(0,
GameManagerScr.Instance.PlayerFieldCards.Count)]));
        }

        break;
    }
}

IEnumerator CastCard(CardController spell, CardController target = null)
{
    spell.Info.ShowCardInfo();
    if (((SpellCard)spell.Card).SpellTarget ==
SpellCard.TargetType.NO_TARGET)
    {

spell.GetComponent<CardMovementScr>().MoveToField(GameManagerScr.Instance.EnemyF
ield);

        yield return new WaitForSeconds(.51f);

        spell.OnCast();
    }
    else
    {
        //spell.Info.ShowCardInfo();

spell.GetComponent<CardMovementScr>().MoveToTarget(target.transform);
        yield return new WaitForSeconds(.51f);

        GameManagerScr.Instance.EnemyHandCards.Remove(spell);
        GameManagerScr.Instance.EnemyFieldCards.Add(spell);
        GameManagerScr.Instance.ReduceMana(false, spell.Card.Manacost);

        spell.Card.IsPlaced = true;

        spell.UseSpell(target);
    }

    string targetStr = target == null ? "no_target" : target.Card.Name;

```

```

        Debug.Log("AI spell cast: " + (spell.Card).Name + " target: " +
targetStr);
    }

    void ChooseAnotherCard()
    {
        // Assuming you have a list of cards in the AI's hand named
enemyHandCards
        List<CardController> enemyHandCards =
GameManagerScr.Instance.EnemyHandCards;

        // Choose another card randomly from the remaining cards in the AI's
hand
        while (enemyHandCards.Count > 0)
        {
            int randomIndex = Random.Range(0, enemyHandCards.Count);
            CardController chosenCard = enemyHandCards[randomIndex];

            // Check if the chosen card is a spell with ALLAY_CARD_TARGET or
ENEMY_CARD_TARGET
            if (chosenCard.Card.IsSpell)
            {
                SpellCard spellCard = chosenCard.Card as SpellCard;

                if (spellCard != null &&
                    (spellCard.SpellTarget ==
SpellCard.TargetType.ALLAY_CARD_TARGET &&
GameManagerScr.Instance.EnemyFieldCards.Count == 0 ||
                    spellCard.SpellTarget ==
SpellCard.TargetType.ENEMY_CARD_TARGET &&
GameManagerScr.Instance.PlayerFieldCards.Count == 0))
                {
                    // The chosen spell card doesn't meet the conditions, choose
another
                    enemyHandCards.RemoveAt(randomIndex);
                    continue;
                }
                else
                {
                    // The chosen spell card meets the conditions, cast the
spell
                    Debug.Log("Casting новый spell " + chosenCard.Card.Name);
                    CastSpell(chosenCard);
                    return;
                }
            }
            else
            {
                // The chosen card is not a spell, proceed with regular card
actions
                Debug.Log("Casting не spell " + chosenCard.Card.Name);

                CardMovementScr cardMovement =
chosenCard.GetComponent<CardMovementScr>();

                if (cardMovement != null)
                {
                    cardMovement.MoveToField(GameManagerScr.Instance.EnemyField);

                    chosenCard.transform.SetParent(GameManagerScr.Instance.EnemyField);
                    chosenCard.OnCast();
                }
            }
        }
    }
}

```

```

        }
        else
        {
            Debug.LogError("CardMovementScr component is null on
chosenCard.");
        }

        return;
    }
}

// Log a message indicating there are no more cards in the AI's hand
Debug.Log("No more cards in the AI's hand.");
}
}

```

CardController.cs

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

public class CardController : MonoBehaviour
{
    public Card Card;

    public bool IsPlayerCard;

    public CardInfoScr Info;
    public CardMovementScr Movement;
    public CardAbility Ability;

    GameManagerScr gameManager;

    public void Init(Card card, bool isPlayerCard)
    {
        Card = card;
        gameManager = GameManagerScr.Instance;
        IsPlayerCard = isPlayerCard;

        if(isPlayerCard)
        {
            Info.ShowCardInfo();
            GetComponent<AttackedCard>().enabled = false;
        }
        else
        {
            Info.HideCardInfo();
        }
    }

    public void OnCast()
    {
        if(Card.IsSpell && ((SpellCard)Card).SpellTarget !=
        SpellCard.TargetType.NO_TARGET)
        {
            return;
        }

        if(IsPlayerCard)
        {
            gameManager.PlayerHandCards.Remove(this);
            gameManager.PlayerFieldCards.Add(this);
            gameManager.ReduceMana(true, Card.Manacost);
            gameManager.CheckCardsForManaAvaliability();
        }
        else
        {
            gameManager.EnemyHandCards.Remove(this);
            gameManager.EnemyFieldCards.Add(this);
            gameManager.ReduceMana(false, Card.Manacost);
            Info.ShowCardInfo();
        }

        Card.IsPlaced = true;
    }
}
```

```

        if (Card.HasAbility)
        {
            Ability.OnCast();
        }

        if (Card.IsSpell)
        {
            UseSpell(null);
        }

        UIController.Instance.UpdateHPAndMana();
    }

    public void OnTakeDamage(CardController attacker = null)
    {
        CheckForAlive();
        Ability.OnDamageTake(attacker);
    }

    public void OnDamageDeal()
    {
        Card.TimesDealedDamage++;
        Card.CanAttack = false;
        Info.HighlightCard(false);

        if (Card.HasAbility)
        {
            Ability.OnDamageDeal();
        }
    }

    public void UseSpell(CardController target)
    {
        var spellCard = (SpellCard)Card;

        switch(spellCard.Spell)
        {
            case SpellCard.SpellType.HEAL_ALLY_FIELD_CARDS:
                var allyCards = IsPlayerCard ? gameManager.PlayerFieldCards :
gameManager.EnemyFieldCards;

                foreach(var card in allyCards)
                {
                    card.Card.Defense += spellCard.SpellValue;
                    card.Info.RefreshData();
                }

                break;

            case SpellCard.SpellType.DAMAGE_ENEMY_FIELD_CARDS:
                var enemyCards = IsPlayerCard ? new
List<CardController>(gameManager.EnemyFieldCards) : new
List<CardController>(gameManager.PlayerFieldCards);

                foreach(var card in enemyCards)
                {
                    GiveDamageTo(card, spellCard.SpellValue);
                }

                break;
        }
    }

```

```

case SpellCard.SpellType.HEAL_ALLAY_HERO:
    if (IsPlayerCard)
    {
        gameManager.CurrentGame.Player.HP += spellCard.SpellValue;
    }
    else
    {
        gameManager.CurrentGame.Enemy.HP += spellCard.SpellValue;
    }

    UIController.Instance.UpdateHPAndMana();

    break;

case SpellCard.SpellType.DAMAGE_ENEMY_HERO:
    if (IsPlayerCard)
    {
        gameManager.CurrentGame.Enemy.HP -= spellCard.SpellValue;
    }
    else
    {
        gameManager.CurrentGame.Player.HP -= spellCard.SpellValue;
    }

    UIController.Instance.UpdateHPAndMana();
    gameManager.CheckForResult();

    break;

case SpellCard.SpellType.HEAL_ALLAY_CARD:
    target.Card.Defense += spellCard.SpellValue;
    break;

case SpellCard.SpellType.DAMAGE_ENEMY_CARD:
    GiveDamageTo(target, spellCard.SpellValue);
    break;

case SpellCard.SpellType.SHIELD_ON_ALLAY_CARD:
    if(!target.Card.Abilities.Exists(x => x ==
Card.AbilityType.SHIELD))
    {
        target.Card.Abilities.Add(Card.AbilityType.SHIELD);
    }

    break;

case SpellCard.SpellType.PROVOCATION_ON_ALLAY_CARD:
    if (!target.Card.Abilities.Exists(x => x ==
Card.AbilityType.PROVOCATION))
    {
        target.Card.Abilities.Add(Card.AbilityType.PROVOCATION);
    }

    break;

case SpellCard.SpellType.BUFF_CARD_DAMAGE:
    target.Card.Attack += spellCard.SpellValue;
    break;

case SpellCard.SpellType.DEBUFF_CARD_DAMAGE:
    target.Card.Attack = Mathf.Clamp(target.Card.Attack -
spellCard.SpellValue, 0, int.MaxValue);

```

```

        break;
    }

    if(target != null)
    {
        target.Ability.OnCast();
        target.CheckForAlive();
    }

    DestroyCard();
}

void GiveDamageTo(CardController card, int damage)
{
    card.Card.GetDamage(damage);
    card.CheckForAlive();
    card.OnTakeDamage();
}

public void CheckForAlive()
{
    if (Card.IsAlive)
    {
        Info.RefreshData();
    }
    else
    {
        DestroyCard();
    }
}

void DestroyCard()
{
    Movement.OnEndDrag(null);

    RemoveCardFromList(gameManager.EnemyFieldCards);
    RemoveCardFromList(gameManager.EnemyHandCards);
    RemoveCardFromList(gameManager.PlayerFieldCards);
    RemoveCardFromList(gameManager.PlayerHandCards);

    Destroy(gameObject);
}

void RemoveCardFromList(List<CardController> list)
{
    if(list.Exists(x => x == this))
    {
        list.Remove(this);
    }
}
}
}

```

CardManagerScr.cs

```

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

public class Card
{
    public enum AbilityType
    {
        NO_ABILITY,
        INSTANT_ACTIVE, // карта активна одразу після касту
        DOUBLE_ATTACK, // подвійна атака
        SHIELD, // один раз оберігає карту
        PROVOCATION, // бити інші карти не можна
        REGENERATION_EACH_TURN, // +2хп кожен хід
        COUNTER_ATTACK // карта, що атакує її, отримує урон двічі
    }

    public string Name;
    public string Description;
    public Sprite Logo;
    public int Attack, Defense, Manacost;
    public bool CanAttack;
    public bool IsPlaced;

    public List<AbilityType> Abilities;

    public bool IsSpell;

    public bool IsAlive
    {
        get
        {
            return Defense > 0;
        }
    }

    public bool HasAbility
    {
        get
        {
            return Abilities.Count > 0;
        }
    }

    public bool IsProvocation
    {
        get
        {
            return Abilities.Exists(x => x == AbilityType.PROVOCATION);
        }
    }

    public int TimesDealedDamage;

    public Card(string name, string description, string logoPath, int attack,
int defense, int manacost, AbilityType abilityType = 0)
    {

```

```

        Name = name;
        Description = description;
        Logo = Resources.Load<Sprite>(logoPath);
        Attack = attack;
        Defense = defense;
        Manacost = manacost;
        CanAttack = false;
        IsPlaced = false;

        Abilities = new List<AbilityType>();

        if(abilityType != 0)
        {
            Abilities.Add(abilityType);
        }

        TimesDealedDamage = 0;
    }

    public Card(Card card)
    {
        Name = card.Name;
        Description = card.Description;
        Logo = card.Logo;
        Attack = card.Attack;
        Defense = card.Defense;
        Manacost = card.Manacost;
        CanAttack = false;
        IsPlaced = false;

        Abilities = new List<AbilityType>(card.Abilities);

        TimesDealedDamage = 0;
    }

    public void GetDamage(int dmg)
    {
        if(dmg > 0)
        {
            if (Abilities.Exists(x => x == AbilityType.SHIELD))
            {
                Abilities.Remove(AbilityType.SHIELD);
            }
            else
            {
                Defense -= dmg;
            }
        }
    }

    public Card GetCopy()
    {
        return new Card(this);
    }
}

public class SpellCard : Card
{
    public enum SpellType
    {
        NO_SPELL,
        HEAL_ALLY_FIELD_CARDS, // лікування всіх своїх карт на полі
    }
}

```

```

        DAMAGE_ENEMY_FIELD_CARDS, // шкода всім картам ворога на полі
        HEAL_ALLAY_HERO, // лікування свого героя
        DAMAGE_ENEMY_HERO, // шкода герою ворога
        HEAL_ALLAY_CARD, // лікування своєї карти
        DAMAGE_ENEMY_CARD, // шкода карті ворога
        SHIELD_ON_ALLAY_CARD, // щит на свою карту
        PROVOCATION_ON_ALLAY_CARD, // провокація на свою карту
        BUFF_CARD_DAMAGE, // збільшення урону своєї карти
        DEBUFF_CARD_DAMAGE // зменшення урону карти ворога
    }

    public enum TargetType
    {
        NO_TARGET, // викласти на поле як звичайну карту
        ALLAY_CARD_TARGET,
        ENEMY_CARD_TARGET
    }

    public SpellType Spell;
    public TargetType SpellTarget;
    public int SpellValue;

    public SpellCard(string name, string description, string logoPath, int
    manacost, SpellType spellType = 0, int spellValue = 0, TargetType targetType =
    0) : base(name, description, logoPath, 0, 0, manacost)
    {
        IsSpell = true;

        Spell = spellType;
        SpellTarget = targetType;
        SpellValue = spellValue;
    }

    public SpellCard(SpellCard card) : base(card)
    {
        IsSpell = true;

        Spell = card.Spell;
        SpellTarget = card.SpellTarget;
        SpellValue = card.SpellValue;
    }

    public new SpellCard GetCopy()
    {
        return new SpellCard(this);
    }
}

public static class CardManager
{
    public static List<Card> AllCards = new List<Card>();
}

public class CardManagerScr : MonoBehaviour
{
    public void Awake()
    {
        CardManager.AllCards.Add(new Card("Provocation", "It is not possible to
        inflict damage with other punishments except provocation",
        "Sprites/Cards/provocation", 1, 2, 3, Card.AbilityType.PROVOCATION));
    }
}

```

```

        CardManager.AllCards.Add(new Card("Regeneration", "Adds 2 hp to this
card each turn while it is on the field", "Sprites/Cards/regeneration", 4, 2, 5,
Card.AbilityType.REGENERATION_EACH_TURN));
        CardManager.AllCards.Add(new Card("Double Attack", "The card deals a
double attack", "Sprites/Cards/doubleAttack", 3, 2, 4,
Card.AbilityType.DOUBLE_ATTACK));
        CardManager.AllCards.Add(new Card("Instant Active", "The card is active
immediately after caste", "Sprites/Cards/instantActive", 2, 1, 2,
Card.AbilityType.INSTANT_ACTIVE));
        CardManager.AllCards.Add(new Card("Shield", "Protects the card from
damage once", "Sprites/Cards/shield", 5, 1, 7, Card.AbilityType.SHIELD));
        CardManager.AllCards.Add(new Card("Counter Attack", "A card that attacks
this card takes damage twice", "Sprites/Cards/counterAttack", 3, 1, 1,
Card.AbilityType.COUNTER_ATTACK));

        CardManager.AllCards.Add(new SpellCard("Heal Allay Field Cards", "The
card heals all of its cards on the field", "Sprites/Cards/healAllayCards", 2,
SpellCard.SpellType.HEAL_ALLY_FIELD_CARDS, 2, SpellCard.TargetType.NO_TARGET));
        CardManager.AllCards.Add(new SpellCard("Damage Enemy Field Cards", "The
card deals damage to all enemy cards on the field",
"Sprites/Cards/damageEnemyFieldCards", 2,
SpellCard.SpellType.DAMAGE_ENEMY_FIELD_CARDS, 2,
SpellCard.TargetType.NO_TARGET));
        CardManager.AllCards.Add(new SpellCard("Heal Allay Hero", "The card
heals its hero", "Sprites/Cards/healAllayHero", 2,
SpellCard.SpellType.HEAL_ALLAY_HERO, 2, SpellCard.TargetType.NO_TARGET));
        CardManager.AllCards.Add(new SpellCard("Damage Enemy Hero", "The card
deals damage to an enemy hero", "Sprites/Cards/damageEnemyHero", 2,
SpellCard.SpellType.DAMAGE_ENEMY_HERO, 2, SpellCard.TargetType.NO_TARGET));
        CardManager.AllCards.Add(new SpellCard("Heal Allay Card", "The card
heals one of its target cards", "Sprites/Cards/healAllayCard", 2,
SpellCard.SpellType.HEAL_ALLAY_CARD, 2,
SpellCard.TargetType.ALLAY_CARD_TARGET));
        CardManager.AllCards.Add(new SpellCard("Damage Enemy Card", "The card
deals damage to one selected enemy card", "Sprites/Cards/damageEnemyCard", 2,
SpellCard.SpellType.DAMAGE_ENEMY_CARD, 2,
SpellCard.TargetType.ENEMY_CARD_TARGET));
        CardManager.AllCards.Add(new SpellCard("Shield On Allay Card", "A card
places a shield on one of its cards", "Sprites/Cards/shieldOnAllayCard", 2,
SpellCard.SpellType.SHIELD_ON_ALLAY_CARD, 0,
SpellCard.TargetType.ALLAY_CARD_TARGET));
        CardManager.AllCards.Add(new SpellCard("Provocation On Allay Card", "The
card places a provocation on one of its chosen cards",
"Sprites/Cards/provocationOnAllayCard",
2, SpellCard.SpellType.PROVOCATION_ON_ALLAY_CARD, 0,
SpellCard.TargetType.ALLAY_CARD_TARGET));
        CardManager.AllCards.Add(new SpellCard("Buff Card Damage", "The card
increases the damage of one of its chosen cards",
"Sprites/Cards/buffCardDamage", 2, SpellCard.SpellType.BUFF_CARD_DAMAGE, 2,
SpellCard.TargetType.ALLAY_CARD_TARGET));
        CardManager.AllCards.Add(new SpellCard("Debuff Card Damage", "The card
reduces the damage of one selected enemy card",
"Sprites/Cards/debuffCardDamage", 2, SpellCard.SpellType.DEBUFF_CARD_DAMAGE, 2,
SpellCard.TargetType.ENEMY_CARD_TARGET));
    }
}

```

CardMovementScr.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using DG.Tweening;
using UnityEngine.UI;

public class CardMovementScr : MonoBehaviour, IBeginDragHandler, IDragHandler,
IEndDragHandler
{
    public CardController CC;

    Camera MainCamera;
    Vector3 offset;
    public Transform DefaultParent, DefaultTempCardParent;
    GameObject TempCardGO;
    public bool IsDraggable;

    int startID;

    void Awake()
    {
        MainCamera = Camera.allCameras[0];
        TempCardGO = GameObject.Find("TempCardGO");
    }

    public void OnBeginDrag(PointerEventData eventData)
    {
        offset = transform.position -
MainCamera.ScreenToWorldPoint(eventData.position); // відстань від центра
карочки до місця де було натиснуто мишкою

        DefaultParent = DefaultTempCardParent = transform.parent;

        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)
        {
            return;
        }

        startID = transform.GetSiblingIndex();

        if (CC.Card.IsSpell || CC.Card.CanAttack)
        {
            GameManagerScr.Instance.HighlightTargets(CC, true);
        }
    }
}

```

```

        TempCardGO.transform.SetParent(DefaultParent);
        TempCardGO.transform.SetSiblingIndex(transform.GetSiblingIndex());

        transform.SetParent(DefaultParent.parent);
        GetComponent<CanvasGroup>().blocksRaycasts = false;
    }

    public void OnDrag(PointerEventData eventData)
    {
        if (!IsDraggable)
        {
            return;
        }

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

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

            if (DefaultParent.GetComponent<DropPlaceScr>().Type !=
FieldTypes.SELF_FIELD)
            {
                CheckPosition();
            }
        }
    }

    public void OnEndDrag(PointerEventData eventData)
    {
        if (!IsDraggable)
        {
            return;
        }

        GameManagerScr.Instance.HighlightTargets(CC, false);

        transform.SetParent(DefaultParent);
        GetComponent<CanvasGroup>().blocksRaycasts = true;

        transform.SetSiblingIndex(TempCardGO.transform.GetSiblingIndex());
        TempCardGO.transform.SetParent(GameObject.Find("Canvas").transform); //
повертаємо тимчасову карту за межі екрану
        TempCardGO.transform.localPosition = new Vector3(2095, 0);
    }

    void CheckPosition()
    {
        int newIndex = DefaultTempCardParent.childCount;

        for (int i = 0; i < DefaultTempCardParent.childCount; i++)
        {
            if (transform.position.x <
DefaultTempCardParent.GetChild(i).position.x)
            {
                newIndex = i;
            }

            if (TempCardGO.transform.GetSiblingIndex() < newIndex)

```

```

        {
            newIndex--;
        }

        break;
    }
}

if (TempCardGO.transform.parent == DefaultParent)
{
    newIndex = startID;
}

TempCardGO.transform.SetSiblingIndex(newIndex);
}

public void MoveToField(Transform field)
{
    transform.SetParent(GameObject.Find("Canvas").transform);
    transform.DOMove(field.position, .5f);
}

public void MoveToTarget(Transform target)
{
    StartCoroutine(MoveToTargetCor(target));
}

IEnumerator MoveToTargetCor(Transform target)
{
    Vector3 pos = transform.position;
    Transform parent = transform.parent;
    int index = transform.GetSiblingIndex();

    if (transform.parent.GetComponent<HorizontalLayoutGroup>())
    {
        transform.parent.GetComponent<HorizontalLayoutGroup>().enabled =
false;
    }

    transform.SetParent(GameObject.Find("Canvas").transform);

    transform.DOMove(target.position, .25f);

    yield return new WaitForSeconds(.25f);

    transform.DOMove(pos, .25f);

    yield return new WaitForSeconds(.25f);

    transform.SetParent(parent);
    transform.SetSiblingIndex(index);

    if (transform.parent.GetComponent<HorizontalLayoutGroup>())
    {
        transform.parent.GetComponent<HorizontalLayoutGroup>().enabled =
true;
    }
}

public void MoveToHand(Transform hand)
{
    StartCoroutine(MoveToHandCor(hand));
}

```

```
}

IEnumerator MoveToHandCor(Transform hand)
{
    Vector3 pos = transform.position;

    transform.SetParent(GameObject.Find("Canvas").transform);

    float duration = 0.51f;

    transform.DOMove(hand.position, duration);
    yield return new WaitForSeconds(duration);

    transform.SetParent(hand);

    transform.localPosition = Vector3.zero;
}
}
```

GameManagerScr.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPPro;
using UnityEngine.UIElements;
using UnityEditor.Overlays;

public class Game
{
    public Player Player, Enemy;
    //список карт колоди гравця та ворога
    public List<Card> EnemyDeck, PlayerDeck;

    public Game()
    {
        EnemyDeck = GiveDeckCard();
        PlayerDeck = GiveDeckCard();

        Player = new Player();
        Enemy = new Player();
    }

    //створює випадкову колоду з 10 карт
    List<Card> GiveDeckCard()
    {
        List<Card> list = new List<Card>();
        for(int i = 0; i < 10; i++)
        {
            var card = CardManager.AllCards[Random.Range(0,
CardManager.AllCards.Count)];

            if (card.IsSpell)
            {
                list.Add(((SpellCard)card).GetCopy());
            }
            else
            {
                list.Add(card.GetCopy());
            }
        }

        return list;
    }
}

public class GameManagerScr : MonoBehaviour
{
    // сінглтон
    public static GameManagerScr Instance;

    public Game CurrentGame;
    public Transform EnemyHand, PlayerHand, EnemyField, PlayerField,
EnemyCardDeck, PlayerCardDeck;

    public GameObject CardPref;
    int Turn, TurnTime = 30;

```

```

public AttackedHero EnemyHero, PlayerHero;
public AI EnemyAI;

public List<CardController> PlayerHandCards = new List<CardController>();
public List<CardController> PlayerFieldCards = new List<CardController>();
public List<CardController> EnemyHandCards = new List<CardController>();
public List<CardController> EnemyFieldCards = new List<CardController>();

public bool IsPlayerTurn
{
    get
    {
        return Turn % 2 == 0;
    }
}

private void Awake()
{
    if(Instance == null)
    {
        Instance = this;
    }
}

void Start()
{
    StartGame();
}

public void RestartGame()
{
    StopAllCoroutines();

    foreach(var card in PlayerHandCards)
    {
        Destroy(card.gameObject);
    }
    foreach (var card in PlayerFieldCards)
    {
        Destroy(card.gameObject);
    }
    foreach (var card in EnemyHandCards)
    {
        Destroy(card.gameObject);
    }
    foreach (var card in EnemyFieldCards)
    {
        Destroy(card.gameObject);
    }

    PlayerHandCards.Clear();
    PlayerFieldCards.Clear();
    EnemyHandCards.Clear();
    EnemyFieldCards.Clear();

    StartGame();
}

public void StartGame()
{
    Turn = 0;
}

```

```

        CurrentGame = new Game();

        GiveHandCards(CurrentGame.EnemyDeck, EnemyHand);
        GiveHandCards(CurrentGame.PlayerDeck, PlayerHand);

        UIController.Instance.StartGame();

        StartCoroutine(TurnFunc());
    }

    void GiveHandCards(List<Card> deck, Transform hand)
    {
        int i = 0;
        while(i++ < 4)
        {
            GiveCardToHand(deck, hand);
        }
    }

    void GiveCardToHand(List<Card> deck, Transform hand)
    {
        if(deck.Count == 0)
        {
            return;
        }

        CreateCardPref(deck[0], hand);

        CardController card = hand.GetChild(hand.childCount -
1).GetComponent<CardController>();
        card.Movement.MoveToHand(hand);

        deck.RemoveAt(0);
    }

    void CreateCardPref(Card card, Transform hand)
    {
        GameObject cardGO = Instantiate(CardPref, hand, false);
        CardController cardC = cardGO.GetComponent<CardController>();

        cardC.Init(card, hand == PlayerHand);

        if(cardC.IsPlayerCard)
        {
            cardC.transform.position = PlayerCardDeck.transform.position;

            PlayerHandCards.Add(cardC);
        }
        else
        {
            cardC.transform.position = EnemyCardDeck.transform.position;

            EnemyHandCards.Add(cardC);
        }
    }

    IEnumerator TurnFunc()
    {
        // скидаємо час на таймері
        TurnTime = 30;
        UIController.Instance.UpdateTurnTime(TurnTime);
    }

```

```

foreach (var card in PlayerFieldCards)
{
    card.Info.HighlightCard(false);
}

CheckCardsForManaAvaliability();

if (IsPlayerTurn)
{
    foreach(var card in PlayerFieldCards)
    {
        card.Card.CanAttack = true;
        card.Info.HighlightCard(true);
        card.Ability.OnNewTurn();
    }

    while(TurnTime-- > 0)
    {
        UIController.Instance.UpdateTurnTime(TurnTime);
        yield return new WaitForSeconds(1);
    }

    ChangeTurn();
}
else
{
    foreach (var card in EnemyFieldCards)
    {
        card.Card.CanAttack = true;
        card.Ability.OnNewTurn();
    }

    EnemyAI.MakeTurn();

    while (TurnTime-- > 0)
    {
        UIController.Instance.UpdateTurnTime(TurnTime);
        yield return new WaitForSeconds(1);
    }

    ChangeTurn();
}
}

public void ChangeTurn()
{
    StopAllCoroutines();

    Turn++;

    UIController.Instance.DisableTurnBtn();

    if (IsPlayerTurn)
    {
        GiveNewCards();

        CurrentGame.Player.IncreaseManapool();
        CurrentGame.Player.RestoreRoundMana();

        UIController.Instance.UpdateHPAndMana();
    }
}

```

```

        else
        {
            CurrentGame.Enemy.IncreaseManapool();
            CurrentGame.Enemy.RestoreRoundMana();
        }

        StartCoroutine(TurnFunc());
    }

    void GiveNewCards()
    {
        GiveCardToHand(CurrentGame.EnemyDeck, EnemyHand);
        GiveCardToHand(CurrentGame.PlayerDeck, PlayerHand);
    }

    public void CardsFight(CardController attacker, CardController defender)
    {
        defender.Card.GetDamage(attacker.Card.Attack);
        attacker.OnDamageDeal();
        defender.OnTakeDamage(attacker);

        attacker.Card.GetDamage(defender.Card.Attack);
        attacker.OnTakeDamage();

        attacker.CheckForAlive();
        defender.CheckForAlive();
    }

    public void ReduceMana(bool playerMana, int manacost)
    {
        if (playerMana)
        {
            CurrentGame.Player.Mana -= manacost;
        }
        else
        {
            CurrentGame.Enemy.Mana -= manacost;
        }

        UIController.Instance.UpdateHPAndMana();
    }

    public void DamageHero(CardController card, bool isEnemyAttacked)
    {
        if(isEnemyAttacked)
        {
            CurrentGame.Enemy.GetDamage(card.Card.Attack);
        }
        else
        {
            CurrentGame.Player.GetDamage(card.Card.Attack);
        }

        UIController.Instance.UpdateHPAndMana();
        card.OnDamageDeal();
        CheckForResult();
    }

    public void CheckForResult()
    {
        if((CurrentGame.Enemy.HP == 0 || CurrentGame.Player.HP == 0) ||
        (EnemyHandCards.Count < 1 && EnemyFieldCards.Count < 1))
    }

```

```

        {
            StopAllCoroutines();
            UIController.Instance.ShowResult();
        }
    }

    public void CheckCardsForManaAvaliability()
    {
        foreach(var card in PlayerHandCards)
        {
            card.Info.HighlightManaAvaliability(CurrentGame.Player.Mana);
        }
    }

    public void HighlightTargets(CardController attacker, bool highlight)
    {
        List<CardController> targets = new List<CardController>();

        if (attacker.Card.IsSpell)
        {
            var spellCard = (SpellCard)attacker.Card;

            if(spellCard.SpellTarget == SpellCard.TargetType.NO_TARGET)
            {
                targets = new List<CardController>();
            }
            else if(spellCard.SpellTarget ==
SpellCard.TargetType.ALLAY_CARD_TARGET)
            {
                targets = PlayerFieldCards;
            }
        }
        else
        {
            if (EnemyFieldCards.Exists(x => x.Card.IsProvocation))
            {
                targets = EnemyFieldCards.FindAll(x => x.Card.IsProvocation);
            }
            else
            {
                targets = EnemyFieldCards;

                if (!attacker.Card.IsSpell)
                {
                    EnemyHero.HighlightAsTarget(highlight);
                }
            }
        }

        foreach(var card in EnemyFieldCards)
        {
            if (attacker.Card.IsSpell)
            {
                card.Info.HighlightAsSpellTarget(highlight);
            }
            else
            {
                card.Info.HighlightAsTarget(highlight);
            }
        }
    }
}

```