Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Розробка казуальної комп'ютерної гри*
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Вдовиченка В. В.*
(прізвище, ініціали)
4 курсу *КІ-542* групи

Керівники: *доцент Нєнов О.Л.*
(посада, прізвище та ініціали)
асистент Колумба І.В.
(посада, прізвище та ініціали)

Консультанти: *проф. Басюркіна Н. Й.*
(посада, прізвище та ініціали)
доцент Нєнов О.Л.
(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *10.06* 2023 р., протокол № *8*

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса - 2023 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Розробка ігор та інтерактивних медіа у віртуальній реальності*

ЗАТВЕРДЖУЮ

Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« *10* » *квітня* 2023 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Вдовиченка Віктора Віталійовича

1. Тема роботи *Розробка казуальної комп'ютерної гри*

Затверджена наказом університету від « *10* » *04* 2023 р., наказ № *146-03*

2 Термін здачі здобувачем закінченої роботи *5 червня 2023 р.*

3. Вихідні дані роботи

1. Жанр комп'ютерної гри.

2. Документація на середовище розробки Unity.

4. Перелік питань, які потрібно розробити

1. Вступ. 2. Передпроектний аналіз. 3. Постановка завдання. Концепт-документ.
4. Проектування гри. Дизайн-документ. 5. Створення контенту і програмна реалізація.
6. Економічні розрахунки. 7. Охорона праці. 8. Загальні висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайд 1. Характеристика кваліфікаційної роботи. Слайд 2. Огляд подібних ігор на ринку. Слайд 3. Вимоги до програмного проекту. Слайд 4. Архітектура проекту. Слайд 6. Програмні алгоритми. Слайд 7. Елементи інтерфейсу користувача. Слайд 8. Техніко-економічні показники. Слайд 9. Загальні висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Басюркіна Н.Й., д.е.н., проф.*		
Охорона праці	*Нєнов О.Л., к.т.н., доц.*		
Нормоконтроль	*Нєнов О.Л., к.т.н., доц.*		

7. Дата видачі завдання *10.04.2023*

Керівники _____ *Олексій НЄНОВ*

_____ *Ірина КОЛУМБА*

Завдання прийняв до виконання _____ *Віктор ВДОВИЧЕНКО*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Передпроектний аналіз предметної області.*	*18.04.2023*	
2.	*Постановка завдання на розробку.*	*21.04.2023*	
3.	*Проектування архітектури проекту гри.*	*28.04.2023*	
4.	*Проектування компонентів проекту гри.*	*05.05.2023*	
5.	*Проектування алгоритмів гри.*	*10.05.2023*	
6.	*Програмна реалізація гри.*	*24.05.2023*	
7.	*Тестування гри.*	*28.05.2023*	
8.	*Техніко-економічний аналіз проекту.*	*02.06.2023*	
9.	*Опрацювання питань охорони праці.*	*03.06.2023*	
10.	*Оформлення пояснювальної записки і презентації.*	*05.06.2023*	

Керівники роботи _____ *Олексій НЄНОВ*

_____ *Ірина КОЛУМБА*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач - дипломник _____ *Віктор ВДОВИЧЕНКО*

АНОТАЦІЯ

Дана робота присвячена розробці казуальної комп’ютерної гри «Пасьянс Клондайк» для платформи Windows за допомогою ігрового рушія Unity. Розроблено технічне завдання, виконано проектування гри, а також програмна реалізація її першого варіанту. Здійснено обґрунтований вибір інструментальних засобів розробки. Крім того, досліджено суміжні питання економічного обґрунтування проекту та охорони праці.

Пояснювальна записка складається з 83 аркушів основної текстової частини і 20 аркушів додатків.

Ключові слова: *комп’ютерна гра, пасьянс, Unity, C#.*

ABSTRACT

This work is devoted to the development of a casual computer game "Klondike Solitaire" for the Windows platform using the Unity game engine. The technical task was developed, the design of the game was completed, as well as the software implementation of its first version. A well-founded selection of development tools was made. In addition, related issues of project economic justification and labor protection were investigated.

The explanatory note consists of 83 sheets of the main text part and 20 sheets of appendices.

Keywords: *computer game, solitaire, Unity, C#.*

ЗМІСТ

стор.

ВСТУП

Індустрія комп'ютерних ігор є дуже актуальною сьогодні. Ця галузь швидко розвивається, залучаючи мільйони гравців з усього світу та генеруючи мільярди доларів доходів щороку.

У світі є багато компаній, які створюють ігри на різних платформах, таких як персональні комп'ютери, консолі, мобільні пристрої та інші. Крім того, індустрія комп'ютерних ігор постійно вдосконалюється та розвивається, включаючи нові технології, графіку та інші функції.

Комп'ютерні ігри також мають значний вплив на культуру та суспільство, що дозволяє їм залишатися актуальними та важливими в наш час. Наприклад, деякі ігри допомагають розвивати навички та здібності, інші пропонують віртуальні світи, які забезпечують місце для експериментування та відпочинку, а деякі використовуються в освіті та бізнесі.

Таким чином, індустрія комп'ютерних ігор не тільки актуальна, але й продовжує розширювати свої межі та впливати на наше життя в різних його аспектах. Все це доводить актуальність даної роботи.

Одним з типів комп’ютерних ігор є казуальні ігри (Casual game). Це дуже популярний тип відеоігор, який зазвичай має простий геймплей та доступний для гравців з будь-яким рівнем досвіду. Однією з причин популярності казуальних ігор є, власне, їх доступність та простота. Вони приваблюють новачків в геймінгу та тих, хто шукає просту та розважальну гру, що не потребує складних стратегій або глибокого знання ігрової механіки. Казуальні ігри також дозволяють гравцям грати на мобільних пристроях, що дозволяє їм грати в будь-який час і в будь-якому місці. Крім того, казуальні ігри зазвичай є досить короткими та не вимагають великих витрат часу, що робить їх ідеальними для відпочинку від роботи чи навчання. Казуальні ігри також зазвичай мають низьку ціну або можуть

бути безкоштовними з можливістю внутрішньоігрових покупок, що зробило їх ще більш доступними для гравців.

Об’єктом дослідження в даній роботі є узагальнений процес розробки комп’ютерних казуальних ігор.

Предметом дослідження є складові розробки проекту комп’ютерної казуальної гри.

Основною прикладною метою даної роботи є розробка проекту і реалізація альфа-версії комп’ютерної казуальної гри.

Для здійснення поставленої мети вирішується ряд завдань, серед яких:

- узагальнена постановка завдання;
- аналітичний огляд ринку комп’ютерних казуальних ігор;
- розробка технічного завдання на розробку;
- створення проекту комп’ютерної казуальної гри певного спрямування;
- реалізація проекту у тестовий прототип застосунку;
- техніко-економічне обґрунтування розробки;
- дослідження суміжних питань охорони праці;
- оформлення пояснювальної записки та підготування презентації.

РОЗДІЛ 1
ПЕРЕДПРОЕКТНИЙ АНАЛІЗ І УЗАГАЛЬНЕНА ПОСТАНОВКА ЗАВДАННЯ

1.1 Огляд предметної області комп'ютерних казуальних ігор і обґрунтування доцільності розробки

Казуальна гра (від англ. *Casual* – повсякденний) – це відеогра, орієнтована на масову аудиторію, на відміну від хардкорної гри, орієнтованої на геймерів. Казуальні ігри можуть мати будь-який тип ігрового процесу та відноситися до будь-якого жанру. Як правило, вони використовують прості правила, короткі сеанси і вимагають від гравця невеликого обсягу набутих навичок [5]. Вони не очікують від гравця знайомства зі стандартним набором механік та елементів керування.

За всю історію відеоігор поряд з хардкорними іграми було розроблено та опубліковано незліченну кількість казуальних ігор. У 1990-х і 2000-х роках посилилися злагоджені зусилля щодо отримання вигоди з казуальних ігор, оскільки багато розробників та видавців називали себе компаніями з виробництва казуальних ігор, видаючи ігри спеціально для персональних комп'ютерів, веб-браузерів і, після 2007 року, смартфонів.

У більшості казуальних ігор є:

- захоплюючий, простий, інтуїтивно зрозуміти геймплей;
- простий інтерфейс користувача, керований за допомогою торкання і перегортання мобільного телефону або однією кнопкою миші;
- короткі сеанси, завдяки чому у гру можна грати під час перерв на роботі, у громадському транспорті чи під час очікування у черзі;
- знайомі візуальні елементи, такі як гральні карти або сітчасте ігрове поле.

Казуальні ігри зазвичай коштують менше, ніж хардкорні ігри, що є частиною стратегії із залучення якомога більшої кількості гравців [15]. Можна використовувати будь-який метод монетизації гри, від роздрібного поширення до безкоштовної гри та підтримки реклами.

Термін «гіперказуальна гра» або «миттєва гра» виник у 2017 році для опису надзвичайно простих в освоєнні ігор, які не вимагають завантаження, в які можна грати в існуючому застосунку, такому як веб-браузер або програма для обміну повідомленнями, і які зазвичай монетизуються через покази реклами гравцю [9].

Однією з перших казуальних відеоігор була аркадна гра *Namco Pac-Man* 1980 року, яка дебютувала в золоте століття аркадних відеоігор. За деякими оцінками, протягом XX століття в неї грали більше десяти мільярдів разів, що робить її найприбутковішою відеогрою всіх часів [13].

У 1989 році *Nintendo* випустила пристрій *Game Boy* з безкоштовною грою *Tetris*. Вона дуже швидко набула великої популярності, і цьому приписують успіх молодої портативної ігрової системи *Nintendo* [14].

Гра *Solitaire* від *Microsoft* (1990 р.), яка постачалася безкоштовно разом із *Microsoft Windows*, вважається першою популярною «казуальною грою» на комп'ютері: станом на 2007 р. в неї грало понад 400 мільйонів чоловік [7]. Наступні версії Windows включали казуальні ігри *Minesweeper*, *FreeCell* і *Spider Solitaire*. Компанія випустила чотири пакети *Microsoft Entertainment Pack* для казуальних ігор на офісних комп'ютерах з 1990 до 1992 року.

Казуальні ігри почали процвітати онлайн у 1990-х роках разом з появою всесвітньої павутини, коли карткові та настільні ігри були доступні в платних сервісах, таких як *AOL* та *Prodigy*, а потім на веб-порталах, таких як *Yahoo! Games* та *Microsoft's Gaming Zone*. У середині 2000-х з'явилося більше сайтів, що спеціалізуються на розміщенні та публікації ігор, таких як *Gamesville* та *RealNetworks*. Деякі видавці та розробники затаврували себе спеціально як компанії, що займаються казуальними іграми, такі як *Big Fish Games*, *PopCap Games* та *MumboJumbo*. Поява технологій Shockwave і Flash породила бум веб-ігор, спонукаючи дизайнерів створювати прості ігри, які

можна було розмістити на багатьох різних веб-сайтів і в які можна було пройти всю гру за один короткий присід. Одна з найвідоміших казуальних ігор *Bejeweled* починалася як флеш-гра, яку можна було завантажити за певну плату або купити в роздріб. В 2009 році все ще існував ринок казуальних ігор за 20 доларів США, куплених у роздріб або для скачування.

У 2008 та 2009 роках казуальні ігри для соціальних мереж швидко набули масової популярності після випуску *Mafia Wars* для Facebook та *Happy Farm* у Китаї. *Happy Farm* надихнула багатьох клонів, у тому числі найпопулярнішу гру для соціальних мереж *FarmVille* (2009), яка досягла піку в 83,76 мільйона активних користувачів на місяць у березні 2010 [12].

Казуальні ігри стали популярними на смартфонах відразу після їх дебюту: телефони з сенсорним екраном, такі як iPhone 2007 року, відрізнялися великими кольоровими дисплеями, доступністю протягом усього дня для власника телефону та інтуїтивно зрозумілим інтерфейсом користувача, заснованим на дотику і перетягуванні.

Основна аудиторія консолей для відеоігор – затяті геймери, але на кожній ігровій консолі є кілька казуальних ігор, а унікальний контролер консолі *Wii* від *Nintendo*, що розпізнає рух, сподобався більш звичайній аудиторії, яку, можливо, лякали пристрої введення геймпада інших консолей. *Wii Sports* (2006), колекція з п’яти простих спортивних ігор, у яких гравці використовували ігровий контролер, щоб розмахувати тенісною ракеткою чи бейсбольною битою, постачалася в комплекті з консоллю *Wii* у більшості країн і була продана понад 82 мільйонами копій станом на 2019 рік [10].

1.2 Жанрова класифікація казуальних ігор та вибір жанру гри для розробки

Казуальні ігри можна знайти в багатьох ігрових жанрах. Компанія з виробництва казуальних ігор *Big Fish Games* [6] і сайт огляду казуальних ігор *Gamezebo* [8] назвали сім популярних жанрів казуальних ігор:

- головоломки: серія *Bejeweled*, серія *Collapse!*, серія *Luxor*;
- ігри з пошуком предметів: серія *Mystery Case Files*, серія *Mortimer Beckett*, серія *Hidden Expedition*;
- пригодницькі ігри: серії *Dream Chronicles*, серії *Aveyond*, серії *Nancy Drew*;
- стратегічні ігри (включаючи тайм-менеджмент): серія *Diner Dash*, серія *Delicious*, серія *Cake Mania*;
- аркади та бойовики: *Plants vs. Zombies*, серія *Peggle*, серія *Feeding Frenzy*;
- ігри зі словами та вікторинами: *Bookworm*, серія *Bookworm Adventures*, *Bonnie's Bookstore*;
- карткові та настільні ігри: *Slingo Quest*, *Lottso! Deluxe*, *Luxor Mahjong*.

У подальшому було вирішено розробляти карткову гру – пасьянс. Далі цей жанр описується більш докладно.

1.3 Характеристика карткових ігор

Одним з популярних жанрів казуальних ігор є карткові ігри.

Взагалі карткова (або картярська) гра – це гра із застосуванням гральних карт. Більшість карткових ігор характеризуються випадковим початковим станом, для визначення якого використовується набір (колода) карт для тієї чи іншої гри. Існує безліч наборів гральних карт, створених під конкретні ігри.

Одна з існуючих класифікацій [16] підрозділяє карткові ігри на:

- сімейні – ті, які мають нескладні правила і доступні для будь-якого віку (скриньки, дурень, мавр або 101, п’яниця, дев’ятка та інші);
- комерційні – виграш у яких залежить від уміння гравця (кріббідж, бридж, преферанс, віст, мушка та інші);
- азартні – виграш в яких залежить, в першу чергу, від випадку і частково – від уміння гравця, тому найбільш важливим стає розрахунок

ймовірностей (покер, блекджек, сека, бура (тридцять одне), очко (двадцять одне), бакара, п'яниця, макао, фараон (штос) та інші.

Окремим видом карткових ігор є пас'янси. Пасьянс (від французької *patience* – терпіння) – це карткова гра для однієї людини, рідше – двох. Граючий розкладає карти, дотримуючись певних правил і, найчастіше, переслідуючи певну мету – розмістити карти в якомусь систематичному порядку. Залежно від правил мета може бути досяжна тією чи іншою мірою завдяки інтелектуальним зусиллям граючого (граючих) і завдяки випадковості (що залежить від розкладу). Останнє дозволяє використовувати пасьянси для ворожіння.

Існує три класи пасьянсів.

Побудова послідовностей. Найбільш часто метою пасьянсу є побудова карт у послідовності, зазвичай по масті. Карти перетасованої колоди роздаються у порядку на столі. Потім карти переміщаються і розміщуються таким чином, щоб зрештою розташувати їх у висхідній (наприклад, від туза до короля) або спадної послідовності. Іноді обидва варіанти поєднані в одній грі. Карта, з якої починається необхідна послідовність, називається базовою картою, а побудованим послідовність – сімейством. У деяких випадках базові карти розміщуються на місці на початку гри; в інших вони складаються лише в міру того, як вони потрапляють під руку під час гри. В одних пасьянсах карти послідовності мають бути тієї ж масті, що й базова карта; в інших масть не береться до уваги. Деякі пасьянси дозволяють тимчасово створювати допоміжні послідовності (за спаданням або зростанням), тобто групи карт, наступних одна за одною, але ще не готових для розіграшу в сімейства або послідовності.

Поєднання. Другою метою багатьох пасьянсів є «спарювання» карт, а потім їхнє скидання. Якщо гравцеві вдається викинути всі карти, гра завершується виграшем.

Підсумовування. Третя мета полягає в тому, щоб скинути будь-які дві карти, загальне значення яких становить певну суму, скажімо, одинадцять чи тринадцять; гра завершується, якщо гравець може таким чином позбавитися всіх карт.

Очевидно, що намагання розташувати, поєднати у пари або об'єднати за значенням карти цілої колоди є важким завданням, ступінь якого змінюється відповідно до правил конкретної гри. Тому гравець повинен бути готовий до багатьох невдач, навіть коли він майже досяг мети успіху. Звідси сенс назви пас'янсу – «терпіння» – є відтворенням цього опису.

Оскільки пасьянси дуже популярні, існує безліч їх різновидів і реалізацій у вигляді комп'ютерних ігор для різних обчислювальних платформ. У [17] представлений перелік з більш ніж 300 відомих пасьянсів. З появою і розповсюдженням персональних комп'ютерів пасьянси пережили свого роду ренесанс завдяки тому, що їх можна легко перенести у формат програмного забезпечення та грати на комп'ютерах чи інших електронних пристроях

Найпопулярніші типи пасьянсів мають багато спільного, особливо тому, що всі вони призначені для гри лише одним гравцем (звідси і термін «пасьянс»). Тим не менш, їх крихітні відмінності власними силами роблять їх унікальними і складними.

Косинка або Клондайк залишається найпопулярнішою версією пасьянсів уже багато років. А пасьянс «Косинка онлайн» є найвідомішим варіантом серед усіх видів карткових пасьянсів.

Пасьянс Павук посідає перше місце серед двоколодних пасьянсів. Він отримав свою назву від 8 основ, які необхідно побудувати, щоб виграти гру, оскільки у павуків зазвичай 8 ніг. Шанс виграти в пасьянсі Павук складає приблизно 1 з кожних 3 ігор.

Три піки – ця головоломка отримала свою назву від трьох вершин піраміди, утворених відображенням карток на ігровому полі. Він також відомий як «Три вежі» та «Потрійна піка». Гра була винайдена Робертом

Хогом у 1989 році. Хог провів комп'ютерний статистичний аналіз своєї оригінальної гри та дійшов висновку, що приблизно понад 90% усіх ігор Tripeaks можна виграти.

У міру того, як все більше гравців приєднується до світу пасьянсів, менш відомі ігри починають з'являтися як альтернатива нинішній великій трійці, що є одним з плюсів цих ігор. Кожен може знайти завдання, яке найкраще відповідає його смакам та навичкам, у різних її типах.

Можливо, найвпливовішим з ранніх втілень пасьянсів був «*Solitaire*» від Microsoft – реалізація відомого пасьянсу «Клондайк». Його включення в Microsoft Windows з 1990 року мало особливо великий вплив на популяризацію пасьянсів серед широкої громадськості. Було створено безліч електронних версій цих ігор, в тому числі під запатентованими назвами, відмінними від справжньої назви, такими як «Solitaire» від Microsoft. Солітер вважається більш стратегічним варіантом, ніж інші реалізації Клондайку, оскільки вимагає більш уважного та вдумливого підходу. Тим не менш, це один з пасьянсів з найвищою ймовірністю рішення. За оцінками, близько 99 % розкладок є вирішуваними.

1.4 Потенційні конкуренти

1.4.1 Пасьянс «Косинка» з поставки Microsoft Windows

Існує дуже багато цифрових реалізацій пасьянсу «Клондайк» для, мабуть, усіх існуючих платформ. Однією з найпопулярніших є стандартна версія гри, яка входить у комплект програмного забезпечення операційної системи Windows версій 7 та більш ранніх (рис. 1.1). До речі, у редакції Windows 7 Professional стандартні ігри вимкнені. Якщо вони потрібні, їх треба самостійно увімкнути через Панель управління – Програмні компоненти – Компоненти Windows – Ігри. Там можна обрати потрібні ігри і включити – система автоматично їх інсталює.

У Windows 10 натомість є вбудований застосунок *Microsoft Solitaire Collection*, який можна знайти у списку всіх застосунків і в якому містяться пасьянси «Павук» (*Spider*), «Косинка» (*Klondike*), «Вільна комірка» (*Free Cell*) та інші.

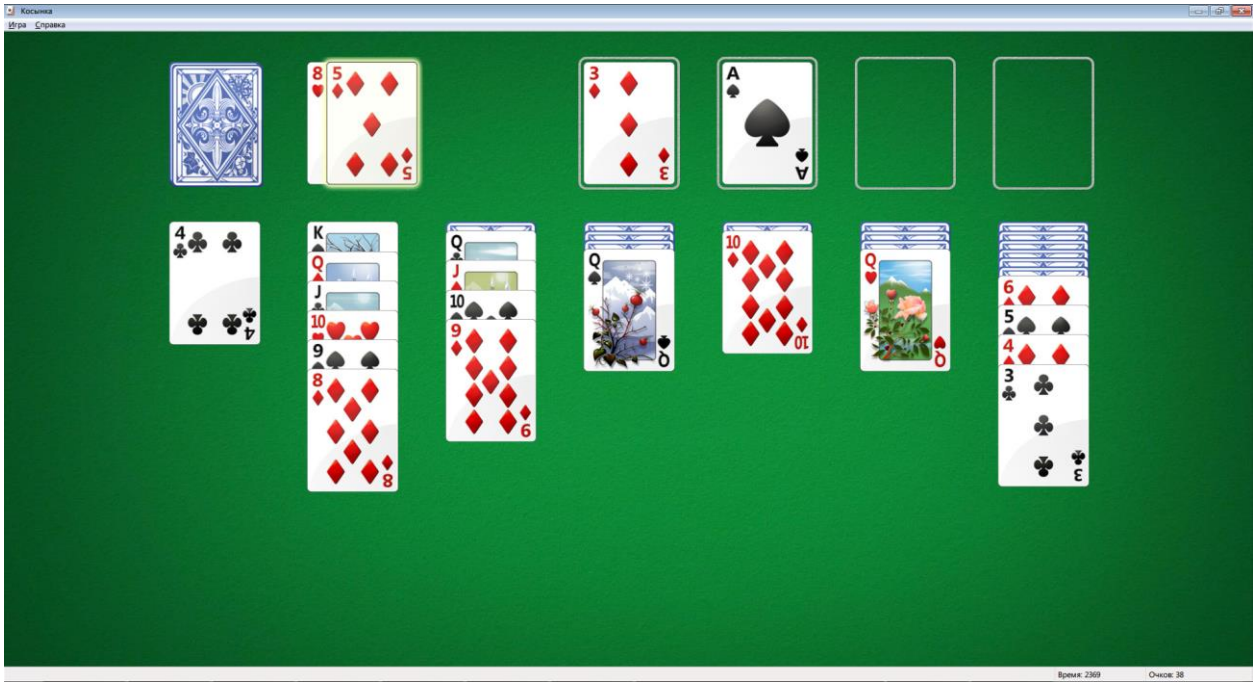

Рис. 1.1 – Стандартний пасьянс «Косинка» з набору програм Windows 7

Стандартна «Косинка» Windows 7 приємно виглядає, має звукове супроводження дій гравця, анімацію перегортання карт та підказки. Ходи здійснюються перетягуванням карт за допомогою миші. Існує можливість налаштувати гру: здавати карти по одні чи по три, вести або не вести рахунок і облік часу проходження, чи треба зберігати гру при виході і продовжувати збережену гру наступного разу, а також чи треба застосовувати звуки і анімацію. Є також можливість відміняти ходи і переглядати статистику минулих ігор.

Гра «Косинка» зі стандартної поставки Windows 7 є невимогливою до системних ресурсів навіть при використанні анімації і звуків. Вважається, що замість випадкових генерацій в ній використовується обмежений набір

розкладок карт, які можуть бути складені. Таким чином, гравець може бути впевнений в тому, що не тратить час завідомо марно.

Якщо немає бажання встановлювати гру у вигляді окремого компонента операційної системи, Microsoft дає можливість пограти у «стандартний» пасьянс «Косинка» з набору *Microsoft Solitaire Collection* онлайн за веб-посиланням *https://zone.msn.com/gameplayer/gameplayerHTML.aspx?game=mssolitairecollection* (рис. 1.2).

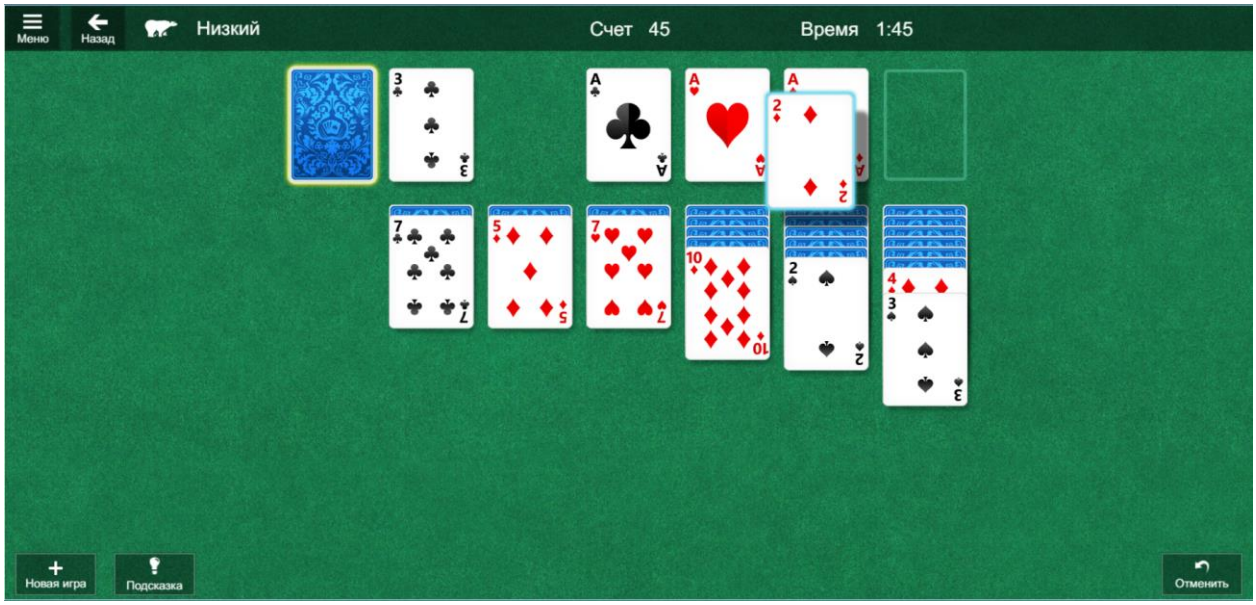

Рис. 1.2 – Онлайн-версія пасьянсу «Косинка» з набору Microsoft Solitaire Collection

Але, на відміну від офлайн-версії, веб-гра помітно підгальмовує на відносно «слабкому» апаратному забезпеченні (такому, наприклад, як двоядерний CPU Pentium E2200 на 2,2 ГГц із вбудованою графікою Intel G33/G31 Express і 4 ГБ оперативної пам’яті DDR2 PC2-6400). Аналіз показав, що завантажуються саме ресурси процесора. Скоріше за все, причиною цього є графічні ефекти: гра використовує тіні і градієнти. Проте, складається загальне враження про недостатню оптимізацію і (або) завищені вимоги. Гра також пропонує налаштування, підказку наступного ходу, можливість відміняти ходи, веде рахунок ігрових балів і облік часу проходження гри.

1.4.2 123 Free Solitaire

Ще однією з численних реалізацій пас'янсу «Клондайк» є програма «123 Free Solitaire» (доступна для завантаження за веб-адресою *https://www.123freesolitaire.com/*). Гра містить у собі декілька пасьянсів, у тому числі, варіанти Клондайку зі здачею по 1 та по 3 карти. Вона використовує красиві анімації, пропонує налаштування зовнішнього вигляду карт і фону, а також деякі особливості управління мишею. Зовнішній вигляд гри показаний на рис. 1.3.

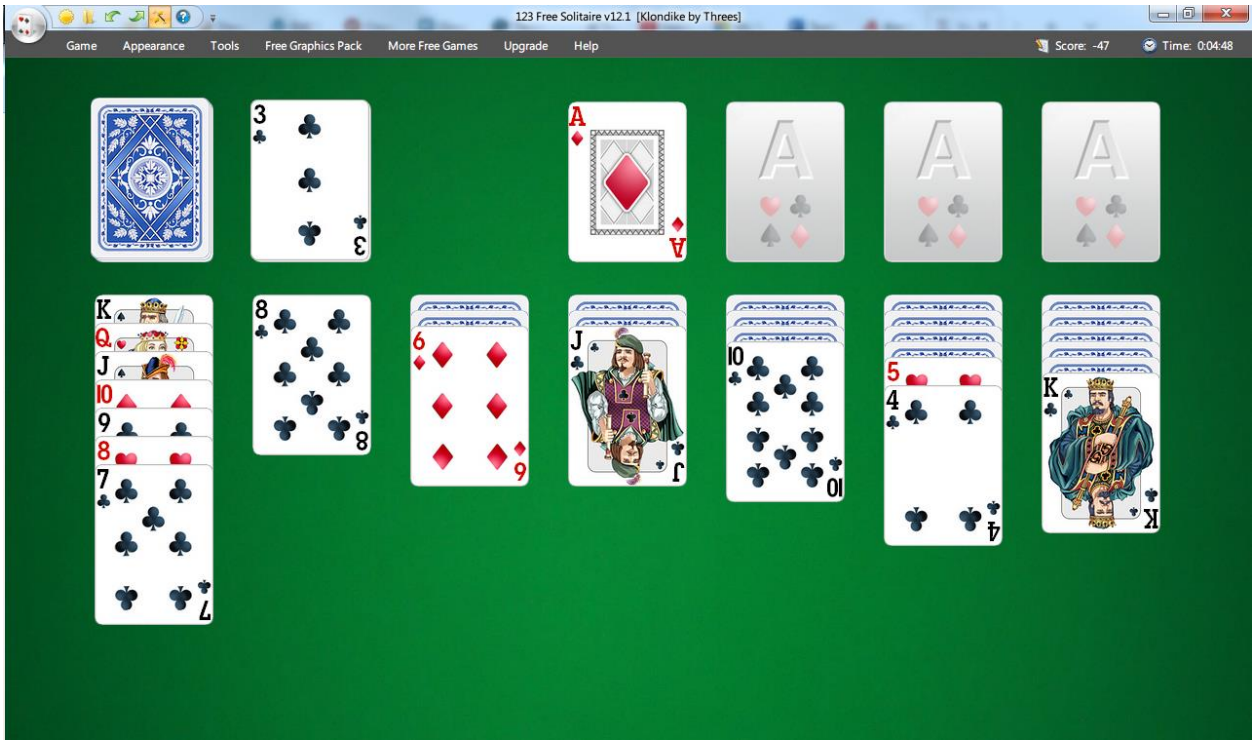

Рис. 1.3 – Пасьянс «Klondike» у грі 123 Free Solitaire

Існує також онлайн-версія гри «123 Free Solitaire». Вона пропонує підказку наступного ходу, відміну зробленого ходу, рахунок і облік часу.

Незважаючи на численну кількість пасьянсів, реалізованих для різних платформ, не було знайдено жодного різновиду цього пасьянсу на 36 карт замість класичних 52.

1.5 Узагальнена постановка завдання

Розроблювана гра (умовна назва «*MySolitaire*») розрахована на платформу Windows і у базовому варіанті реалізуватиме класичний варіант пасьянсу «Клондайк» із зняттям з колоди по три карти. Управління в грі має здійснюватися за допомогою миші; опційно можливо додавання інших способів управління.

Висновки першого розділу

Проведені дослідження, результати яких представлені в розділі, обґрунтовують актуальність і доцільність подальшої роботи. Вони дозволили здійснити загальну постановку завдання на дипломну роботу. На основі отриманих результатів і узагальненої постановки виконується подальше проектування і реалізація казуальної карткової гри.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ЗАСТОСУНКУ КОМП'ЮТЕРНОЇ ГРИ

2.1 Технічне завдання на розробку комп'ютерної гри

2.1.1 Загальна інформація про проект

Комп'ютерна гра пасьянс «Клондайк» за загальними класичними правилами, які описані далі.

Робоча назва – *MySolitaire*.

Цільова платформа – *Windows*.

2.1.2 Опис і правила гри

Класичний варіант пасьянсу «Клондайк» розкладається однією колодою в 52 карти. Мета гри – розкласти карти по мастях в порядку від туза до короля в чотири стопки (їх іноді називають базовими, або «будинками»). Карту можна перекладати на іншу рангом вище, але іншого кольору (чорного або червоного). У кожну з чотирьох базових стопок (будинків), в які необхідно розкласти всі карти, спочатку кладуться тузи, потім двійки, трійки і так далі до короля.

Карти можна здавати (знімати) з колоди, що залишилася від роздачі (у лівому верхньому куті), по три штуки. У вільну нижню комірку можна покласти лише короля. Гра закінчується, коли всі карти розкладено. Мета гри полягає в тому, щоб розкласти всі карти у верхні чотири стопки по зростанню, починаючи з туза, так, щоб карти однієї масті знаходилися в одній стопці.

2.1.3 Додаткові вимоги до ігрового застосунку

Нефункціональні вимоги до гри:

- невимогливість до обчислювальних ресурсів пристроїв (оперативної пам’яті, місця на постійному носії, ресурсу процесора тощо);
- стабільність роботи;
- приємний зовнішній вигляд і зручний інтерфейс управління.

В якості додаткових можливостей гра може:

- підтримувати управління подвійним кліком миші: якщо гравець двічі клацне карту, яка може потрапити в одну з верхніх позиціх, вона повинна автоматично злетіти туди;
- забезпечувати інші способи керування картами на ігровому полі: з сенсорного екрану, з клавіатури тощо;
- підтримувати портретну (вертикальну) та ландшафтну (горизонтальну) орієнтацію екрану пристрою;
- дати можливість користувачеві вибирати та налаштовувати колірні теми графічного інтерфейсу.

2.1.4 Організаційні оцінки

Планується розвивати даний проект за інкрементною стратегією, що передбачає створення все більш функціональної робочої версії на кожній ітерації. На першу ітерацію передбачається витратити 3 місяці і близько 500 годин роботи менеджера, програміста та дизайнера.

Проект може приносити прибуток шляхом розміщення реклами в додатку або через механізм внутрішньоігрових покупок.

2.2 Вибір засобів проектування та інструментів реалізації

Першочерговим питанням стосовно вибору інструментів є питання вибору основного середовища розробки. Взагалі, існує два підходи до розробки комп’ютерних ігор: використовувати універсальні середовища розробки нативних програм для певних платформ або користуватися спеціалізованими ігровими рушіями і конструкторами, які підтримують декілька (часто – багато) платформ. В даній роботі обрано другий варіант, оскільки він є переважним внаслідок більшої продуктивності розробки ігор, особливо – для декількох платформ. Хоча, слід відзначити, що він поступається оптимізованістю за стосунків у порівнянні з програмами, створеними за допомогою класичних універсальних IDE. Далі розглянемо варіанти ігрових рушіїв, які дозволять реалізувати нашу гру.

Unity – це ігровий рушій і популярне середовище розробки, яке широко використовується для створення комп’ютерних і мобільних ігор. Unity надає розробникам потужні інструменти для створення ігор різних жанрів, включаючи відеоігри, аркади, стратегії, ігри на рольових основах та багато інших.

На сьогоднішній день на ринку існує багато альтернатив Unity, які також використовуються для розробки комп'ютерних і мобільних ігор. Далі коротко описані деякі з них.

Unreal Engine – популярний ігровий рушій, який забезпечує потужність та гнучкість у розробці ігор. Unreal Engine має розширені функціональні можливості, особливо у графіці та фізиці, і широкий спектр підтримуваних платформ.

Godot Engine – безкоштовний та відкритий ігровий рушій з активною спільнотою розробників. Godot Engine надає досить широкі можливості розробки ігор, включаючи підтримку 2D та 3D графіки, анімації, фізики та інших функцій.

CRYENGINE – інший доволі потужний ігровий рушій, який використовується для створення ігор з вражаючою графікою. CRYENGINE пропонує високоякісний рендеринг, фізику, освітлення та інші функції, що допомагають розробникам створювати візуально привабливі ігрові світи.

GameMaker Studio – це інструмент, спрямований на швидку та просту розробку ігор, особливо для початківців. GameMaker Studio надає зручний інтерфейс та дозволяє швидко створювати 2D ігри з використанням графічного редактора та мови скриптування.

Кожен з цих рушіїв має свої переваги та особливості, і вибір залежить від конкретних потреб розробника, типу гри та інших факторів. В якості основного інструменту розробки був обраний Unity. Програмування ігрової логіки здійснюватиметься у Visual Studio. Серед переваг обраного рушія Unity:

- гнучкість та ефективність у розробці – дозволяє програмістам, художникам і дизайнерам ефективно працювати разом над проектом, використовуючи потужну мову програмування C#;
- підтримка різних платформ, включаючи Windows, macOS, iOS, Android, Xbox, PlayStation та інших, що дає можливість створювати ігри для різних пристроїв і операційних систем;
- багатий набір інструментів для моделювання ігрових об’єктів, реалістичного рендерингу, фізики, анімації, штучного інтелекту та інших аспектів геймдевелопменту;
- наявність великої спільноти розробників, яка ділиться знаннями, ресурсами та плагінами для полегшення роботи та розширення можливостей розробки ігор.

Для проектування та візуалізації проектних рішень використовується уніфікована мова моделювання UML. Це універсальна мова, яка базується на відкритому стандарті ISO/IEC 19501:2005 і використовує графічні символи для створення абстрактної моделі системи, відомої як UML-

модель. За допомогою UML можна визначати, проектувати, візуалізувати та документувати програмні системи будь-якої архітектури та призначення. Спеціальні CASE-засоби дозволяють генерувати шаблони програмного коду на основі вхідних UML-моделей.

Для створення електронних UML-діаграм в даній роботі використовується онлайн-платформа *diagrams.net*. Цей безкоштовний інструмент, заснований на відкритому вихідному коді, дозволяє створювати діаграми безпосередньо в браузері. Ресурс *diagrams.net* може інтегруватися з популярними хмарними сховищами, такими як Google Drive, Dropbox, Microsoft OneDrive, а також хмарними хостингами проектів, такими як GitHub та GitLab.

2.3 Структура класів проекту

Для створення програмної частини проект гри в Unity будемо використовувати об’єктно-орієнтовану мову C# і компоненти стандартної бібліотеки рушія.

Для реалізації проекту *MySolitaire* сформуємо такі класи:

- *Гра* (*Solitaire*) – основний клас, що відповідає за облік карт на ігровій сцені і реалізує основні операції з картами;
- *Карта* (*Selectable*) – клас карти, зберігає масть і значення карти, а також її поточне положення;
- *Зображення карти* (*UpdateSprite*) – клас, який відповідає за відображення зовнішнього вигляду карти;
- *Обробник користувацького вводу* (*UserInput*) – клас, що обробляє дії користувача (клацання миші).

Відповідна концептуальна діаграма класів наведена на рис. 2.1.

Класи *UserInput* та *UpdateSprite* міститимуть поля класу *Solitaire*, тому зв’язані з ними агрегацією.

Клас *UpdateSprite* міститиме поля не тільки класу *Solitaire*, а й *UserInput* та *Selectable*, тому він також зв’язаний з ними агрегацією.

Клас *Solitaire* управляє картами і звертається до класу *Selectable*, тому зв’язаний з ним відношенням залежності.

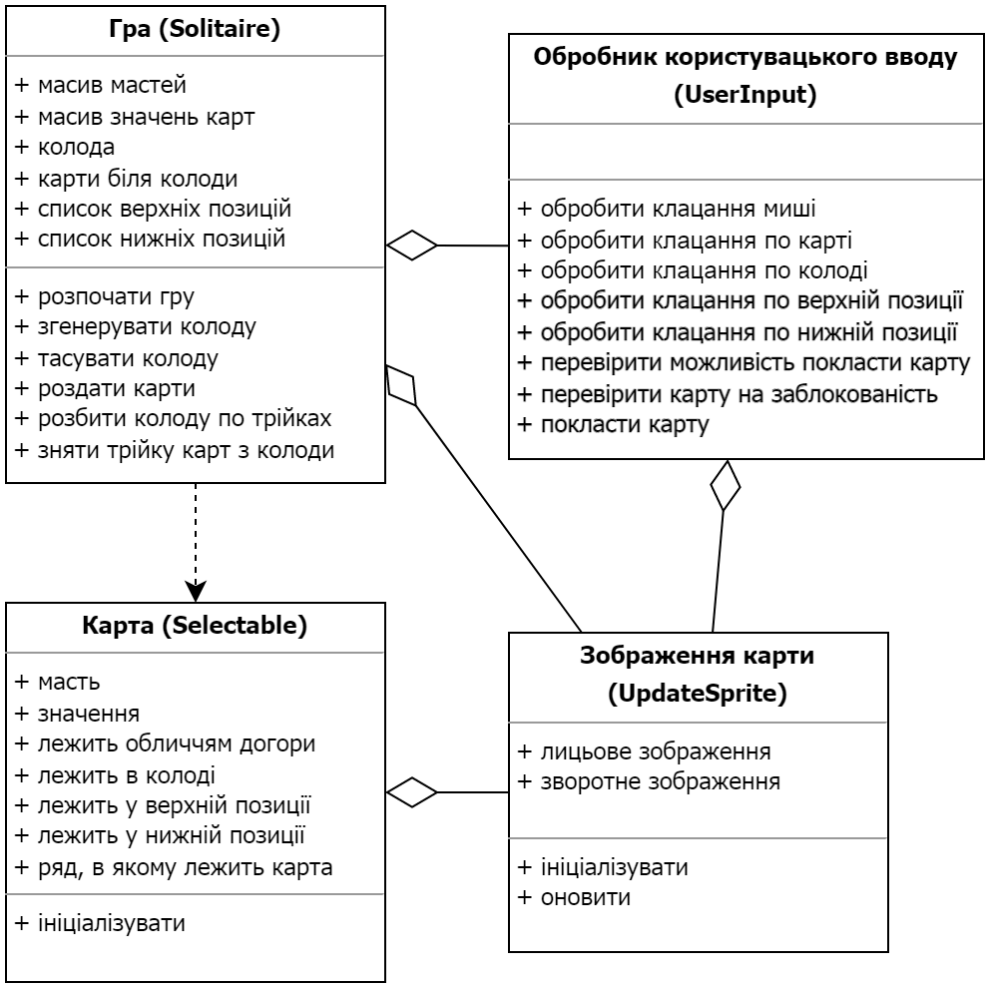

Рис. 2.1 – Класи застосунку *MySolitaire* – концептуальна діаграма

2.4 Елементи функціонування застосунку

2.4.1 Алгоритм обробки натискання на карту

Переміщення карт будемо реалізовувати наступним способом: гравець має спочатку клацнути по карті, яку хоче перемістити, а потім по позиції, куди він хоче її перемістити. Таким чином, коли гравець клікатиме по карті,

є три варіанти обробки цієї дії: або перевернути карту, або обрати (позначити жовтим), або покласти (перемістити) на неї раніше обрану карту.

Перевернути можна закриту карту. Якщо карта відкрита, залишається два варіанти. Якщо на цю відкриту карту можна покласти карту, яка вибиралася раніше, то кладемо. Якщо не можна, то вибираємо поточну карту.

Алгоритм обробки клацання по карті у вигляді блок-схеми показаний на рис. 2.2.

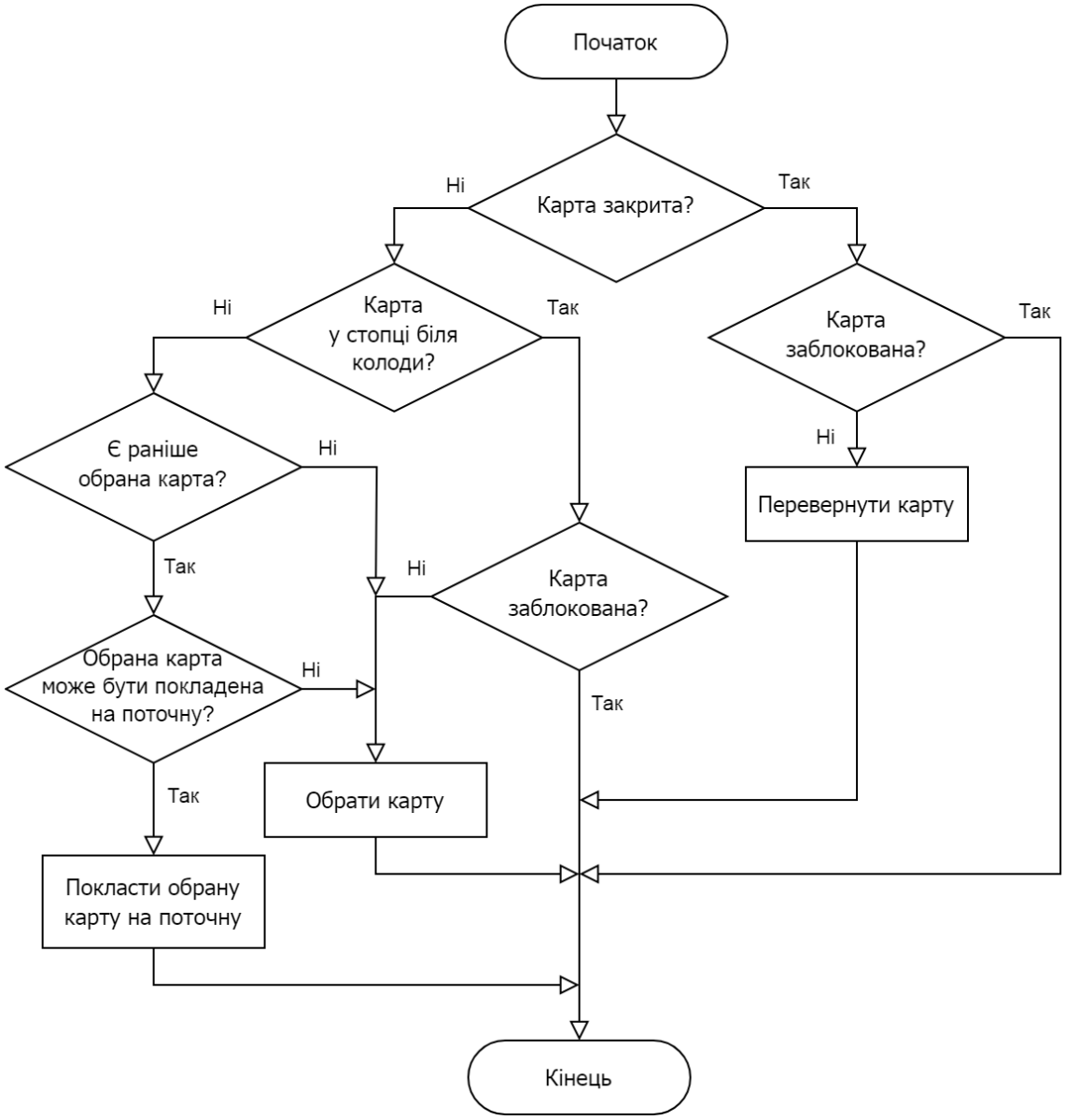

Рис. 2.2 — Блок-схема алгоритму обробки клацання по карті

2.4.2 Алгоритм перевірки можливості покласти одну карту на іншу

У попередньому алгоритмі є перевірка: чи можна покласти карту, яка вибиралася раніше, на натиснуту відкриту карту. Ця перевірка здійснюється за алгоритмом, блок-схема якого показана на рис. 2.3.

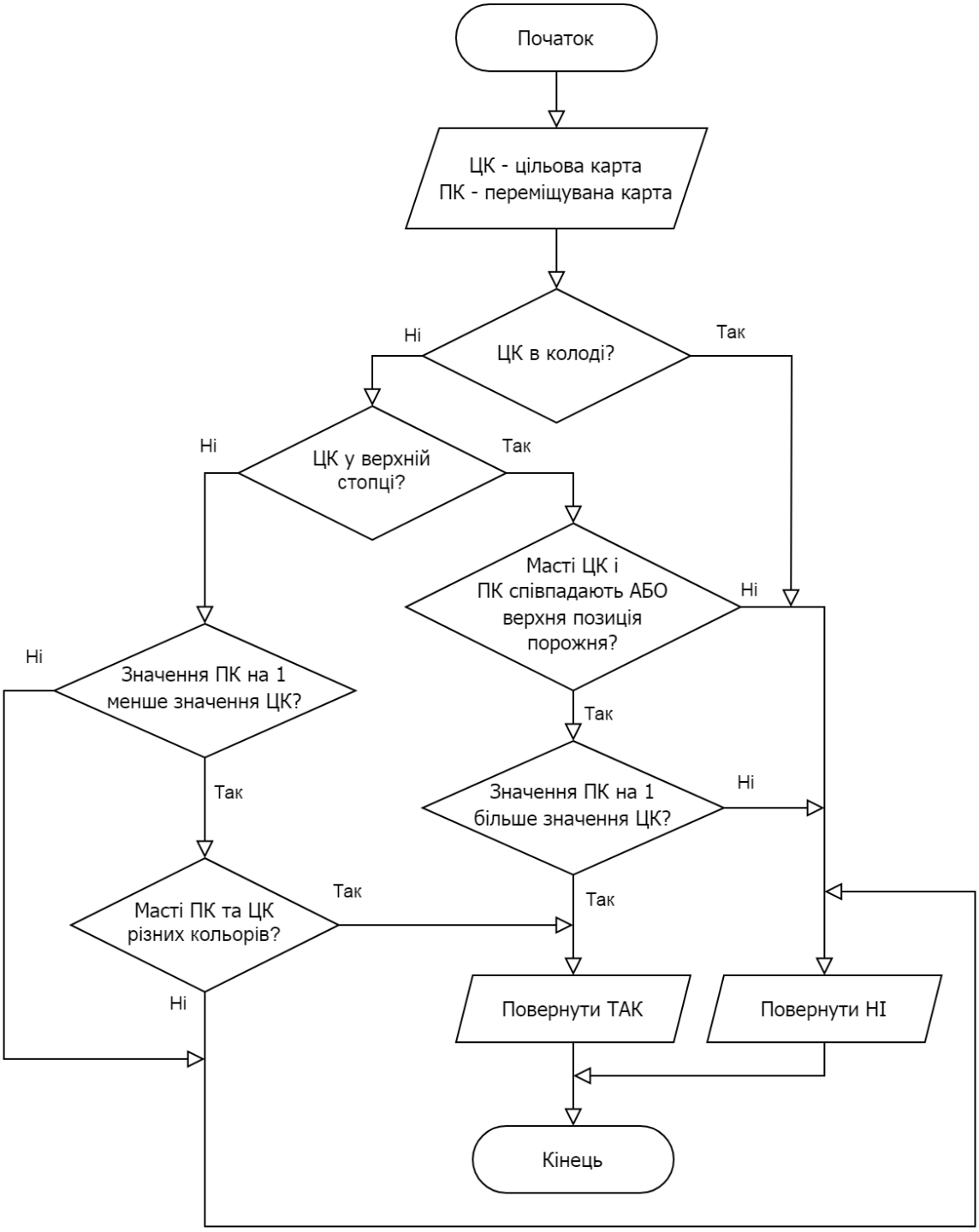

Рис. 2.3 — Блок-схема алгоритму перевірки можливості покласти одну карту на іншу

Висновки другого розділу

В результаті виконання даного етапу роботи були розроблені технічне завдання, вибрані засоби проектування і реалізації проекту казуальної гри *MySolitaire*. В цьому розділі представлені проектні рішення, які стосуються структурних та поведінкових аспектів ігрового проекту. На базі цих рішень можливо приступити до практичної реалізації гри з використанням відповідних інструментальних засобів в обраному середовищі розробки Unity.

РОЗДІЛ 3
РОЗРОБКА ЗАСТОСУНКУ

3.1 Створення і початкове налаштування проекту Unity

Цей проект починається з нуля, тому перше, що нам потрібно зробити, це створити новий проект Unity (рис. 3.1).

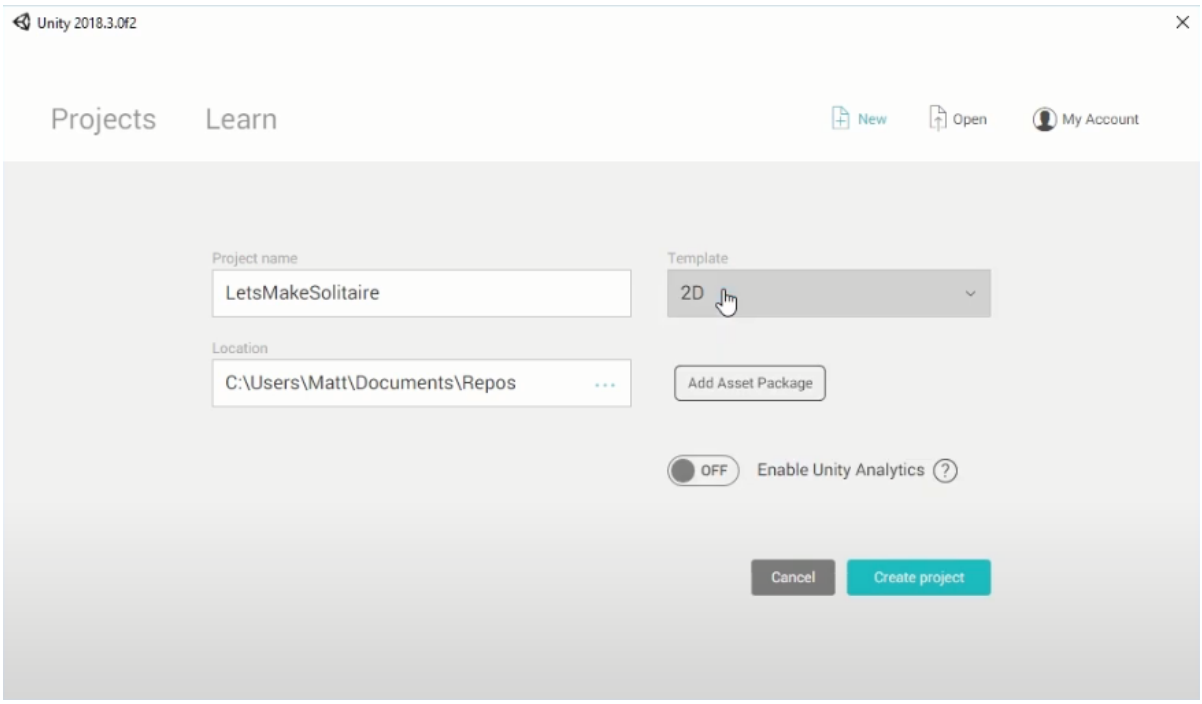

Рис. 3.1 — Створення нового проекту Unity

Оскільки в нас буде двовимірна гра, встановлюємо шаблон 2D.

Після завантаження Unity переходимо на вкладку гри та вибираємо співвідношення сторін 16:9, оскільки це найпоширеніше співвідношення сторін для комп'ютерних моніторів.

Замість додавання фонового зображення налаштуємо фоновий колір камери. Для цього вибираємо основну камеру та натискаємо колір фону – для класичного вигляду пасьянсу виберемо зелений (рис. 3.2).

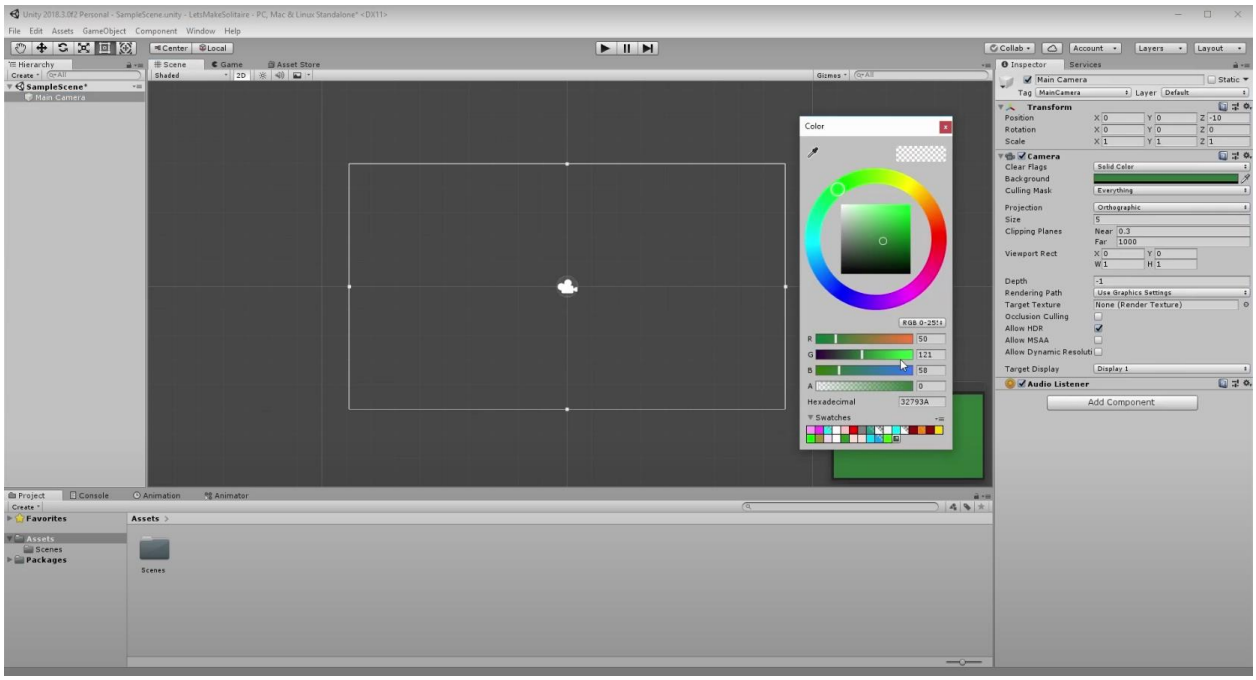

Рис. 3.2 — Налаштування фонового кольору сцени

Одразу переконаємося, що камера переведена у ортографічний режим і що для осі *Z* під час перетворення встановлено значення –10 одиниць.

3.2 Підготування ресурсів для проекту

Для гри нам знадобляться графічні ресурси. Спрайти для гральних карт було завантажено з веб-сайту *OpenGameArt.org* (рис. 3.3). Ці спрайти позначено як загальнодоступні. Пакет містить як растрові png, так і векторні формати файлів – нам знадобляться лише png.

Після завантаження та розпакування перетягуємо файли карт в Unity Editor, у створену папку під назвою *Sprites* (рис. 3.4).

3.3 Створення основних ігрових об’єктів

Далі ми можемо додати деякі ігрові об’єкти до сцени. Перший назвемо *Deck* (колода), другий — *Top* (верхній), а третій — *Bottom* (нижній). Додавання прямокутника до сцени перетягуванням його з папки *Sprites* до

сцени показало, що він трохи завеликий. Для зміни його розміру встановлюємо масштаб 0,4 на осях *X* і *Y*.

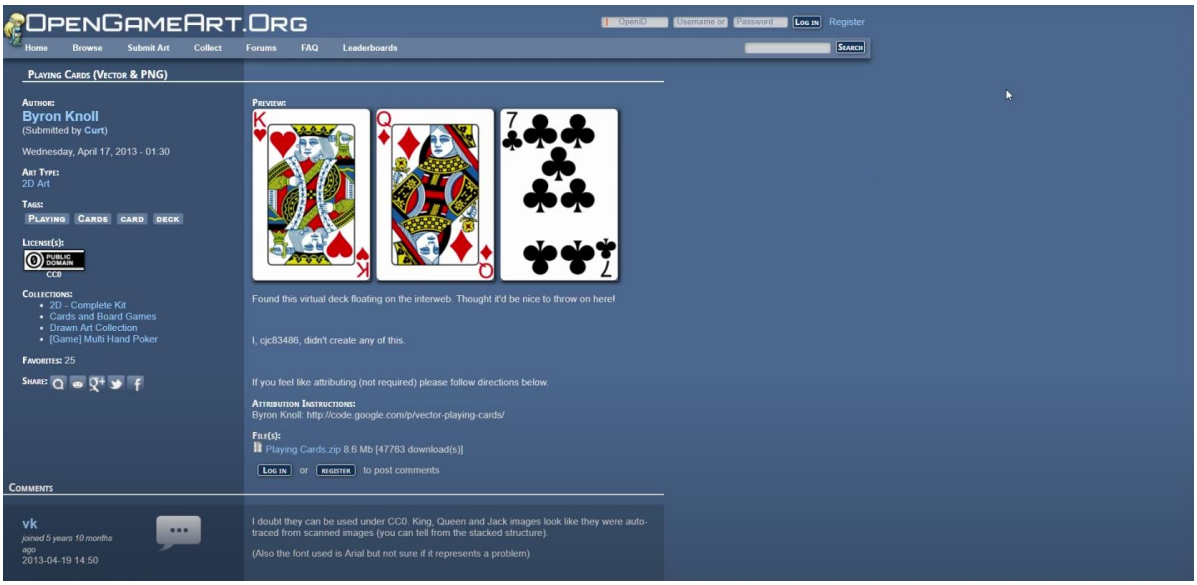

Рис. 3.3 — Отримання графічних ресурсів для гри

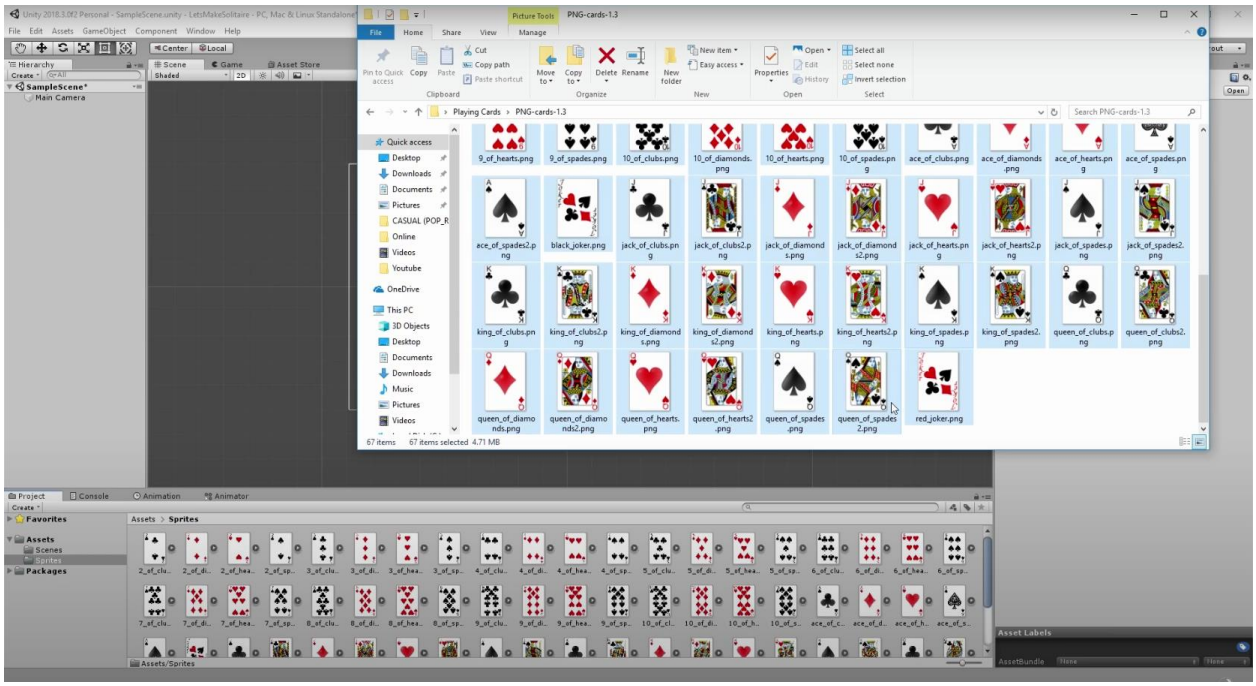

Рис. 3.4 — Перенесення графічних ресурсів у проект Unity

Потім будуємо сцену так, як вона має виглядати. Будемо вручну прописувати розміри, щоб об’єкти акуратно вишикувались в ряди.

Далі перейменовуємо прямокутники послідовно, починаючи з 0, щоб можна було легко відстежувати їх і звертатися у коді, зокрема, коли знадобиться зв’язати їх із масивами або списками.

Далі встановлюємо чотири верхні позиції як дочірні до ігрового об’єкту *Top*, а сім нижніх – як дочірні для ігрового об’єкту *Bottom* (рис. 3.5).

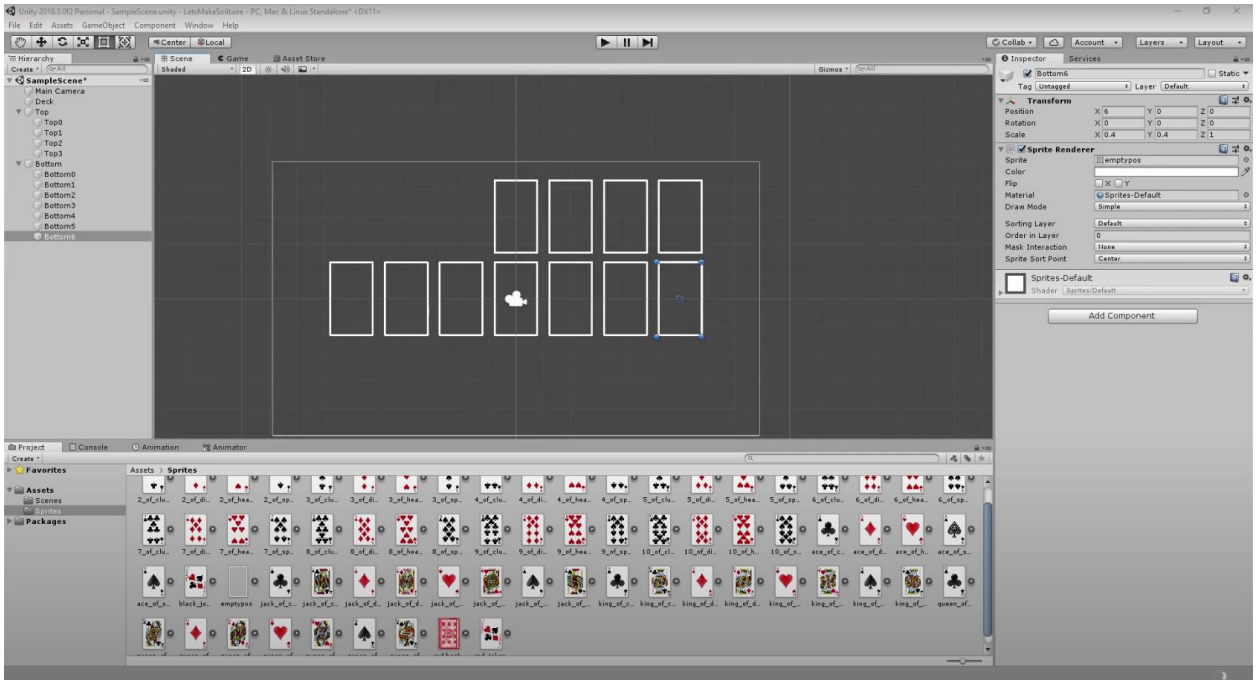

Рис. 3.5 — Формування верхнього і нижнього ряду для стопок карт

Потім додаємо прямокутник до дочірнього елементу до ігрового об’єкта *Deck* та вимикаємо спрайт для зворотної частини зображення карти. Це буде кнопка, яка роздає карти з колоди – назвемо її *DeckButton*.

Замість створення окремих об’єктів для 52 різних карт створюємо префаб *Card*, який стане основою для всіх карт, і розташуємо його поза сценою (рис. 3.6). Кожного разу, коли ми захочемо створити карту, ми можемо створити копію цього префабу та оновити спрайт залежно від того, якою буде карта. Для реалізації цієї логіки створюємо новий сценарій оновлення спрайтів (*UpdateSprite*) та прикріплюємо його до префабу *Card*.

Для сценаріїв і префабів створюємо, відповідно, папки *Scripts* та *Prefabs*.

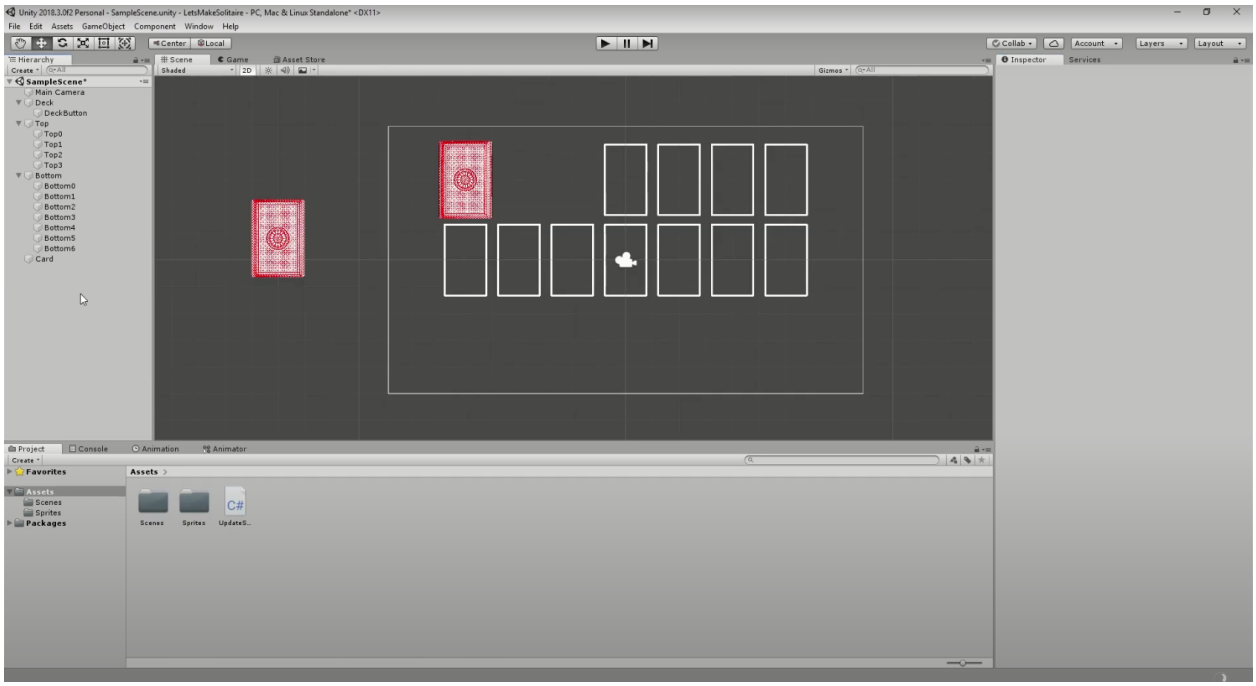

Рис. 3.6 — Створення префабу для карт

3.4 Створення сценаріїв для реалізації ігрової логіки

У грі поки ще нічого немає, тому нам потрібно релізувати деякі функції. В першу чергу, створимо ігровий об’єкт під назвою *SolitaireGame* для обробки ігрової логіки, і додамо до нього сценарій пасьянсу *Solitaire.cs* як компонент. Нам потрібно, щоб цей ігровий об’єкт обробляв будь-які вхідні дані від гравця. Для цього створюємо другий сценарій під назвою *UserInput* та долучаємо його до цього об’єкту.

Ще один сценарій – *Solitaire* – має підтримувати ігровий процес, виконуючи кілька ключових речей: він створить колоду карт, перетасує колоду та правильно роздасть карти.

Для організації карт у програмному коді ми будемо використовувати масиви, цикли та списки. Перш за все, нам потрібен масив мастей. Оскільки є лише чотири масті – трефи, бубни, черви та піки, робимо це статичним рядковим масивом *suits*. Ініціалізуємо його значеннями «*C*» для треф, «*D*» для бубен, «*H*» для черв і «*S*» для пік.

Для бубнових, червових і пікових значень (старшинств) карт заведемо загальнодоступний статичний масив рядків *values*, який проініціалізуємо значеннями *"A"* для туза, і далі *"2"*, *"3"*, *"4"*, *"5"*, *"6"*, *"7"*, *"8"*, *"9"*, *"10"*, *"J"* для валета, *"Q"* для дами та *"K"* для короля.

```
public class Solitaire : MonoBehaviour
{
    public Sprite[] cardFaces;
    public Gameobject cardPrefab;
    public static string[] suits = new string[] { "C", "D", "H", "S" };
    public static string[] values = new string[] { "A", "2", "3", "4", "5",
"6", "7", "8", ”9”, "10", "3", "Q", "K" };
    public List<string> deck;

    // Start is called before the first frame update
    void Start()
    {
        PlayCards();
    }
```

Тепер треба зіставити елементам масиву *cardFaces* зображення карт. Для цього в редакторі прописуємо розмірність цього масиву 52 і перетягуємо кожен спрайт карти на його місце, відсортовуючи по мастях по зростанню значення карти від туза до короля (рис. 3.7).

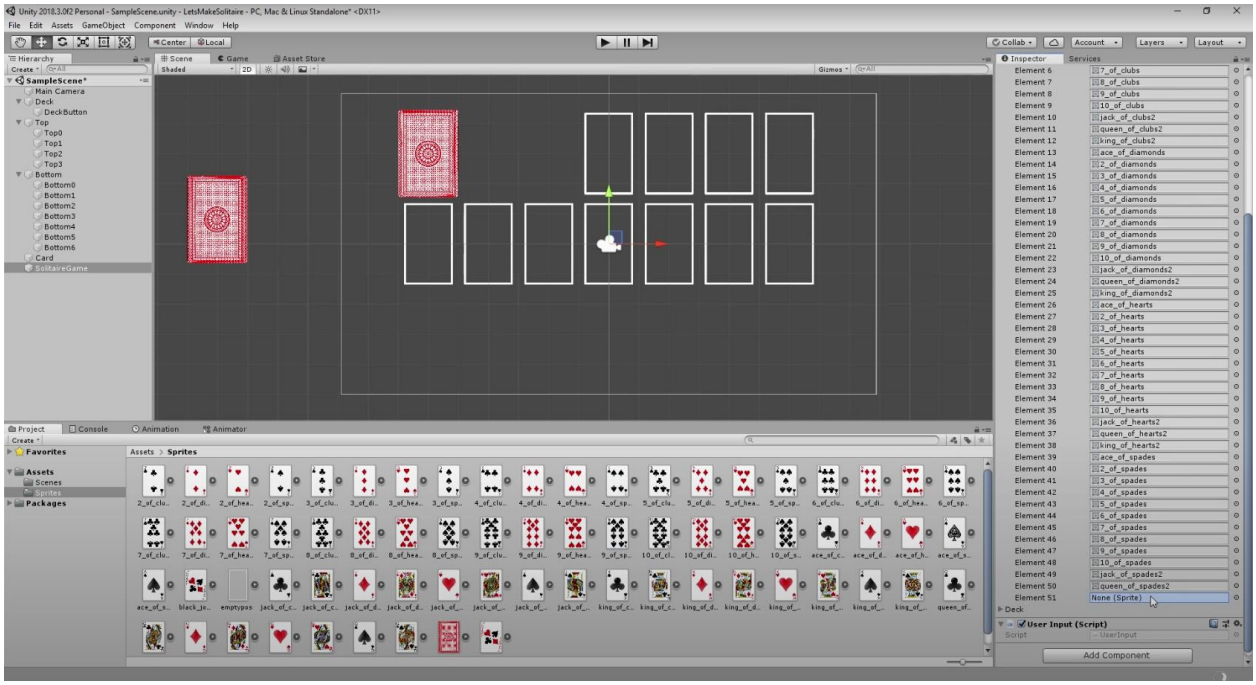

Рис. 3.7 — Заповнення масиву зображень карт

3.5 Генерування, тасування і роздача карт

Оскільки колода карт автоматично генеруватиметься з масивів рядків, створимо новий загальнодоступний статичний метод *GenerateDeck()*, який повертатиме список рядків, кожен з яких відповідатиме певній карті:

```
public static List<string> GenerateDeck()
{
    List<string> newDeck = new List<string>();

    foreach (string s in suits) // цикл по мастях
    {
        foreach (string v in values) // цикл по значеннях карт
        {
            newDeck.Add(s + v); // формуємо рядок "масть + значення" для карти
        }
    }
    return newDeck; // повертаємо список карт
}
```

Далі нам треба перетасувати карти. Існує декілька методів генерації випадкового перемішування. Метод, який ми використовуємо, взятий з відомого сайту Stackoverflow і заснований на алгоритмі тасування Фішера – Йєтса. Відповідна функція тасування *Shuffle()* приймає список елементів довільного типу *T*:

```
    void Shuffle<T>(List<T> list)
    {
        System.Random random = new System.Random();
        int n = list.Count;
        while (n > 1)
        {
            int k = random.Next(n);
            n--;
            T temp = list[k];
            list[k] = list[n];
            list[n] = temp;
        }
    }
```

Тепер, коли ми можемо створити список імен карт і перетасувати його, ми можемо задіяти його для створення карт, які відображатимуться на екрані.

Далі нам потрібен метод роздачі карт – назвемо його *SolitaireDeal()*.

```
void SolitaireDeal()
{
    foreach (string card in deck)
    {
        // затримка перед роздачею чергової карти
        yield return new WaitForSeconds(0.02f);
        // створюємо об'єкт-карту
        GameObject newCard = Instantiate(cardPrefab,
            new Vector3(bottomPos[i].transform.position.x,
                transform.position.y, transform.position.z);
        newCard.name = card;
    }
}
```

При тестуванні методу здається, що на екрані з'являється лише одна карта. Однак якщо подивитися на ієрархію, можна побачити, що тут є усі 52 карти з окремими іменами та перетасовані, просто вони всі займають однакову позицію. І наразі відображається лише задня сторона карт.

Щоб отримати краще візуальне представлення карток, надаємо їм зміщення по осях *y* та *z*:

```
float yOffset = 0;
float zOffset = 0.03f;
. . .
yOffset = yOffset + 0.5f;
zOffset = zOffset + 0.03f;
```

Тепер усі 52 карти створені в одній стопці (рис. 3.8).

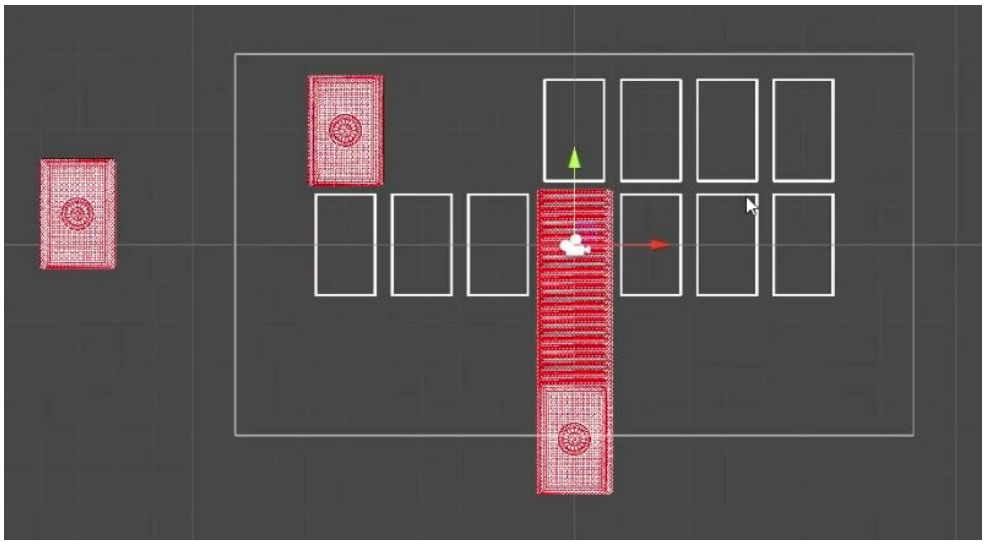

Рис. 3.8 — Стопка розданих карт

У тривимірному режимі можна побачити, що чим нижче карта, тим ближче вона до камери (рис. 3.9). Це може допомогти з ідентифікацією та вирішенням проблем візуалізації та виявленням звернень.

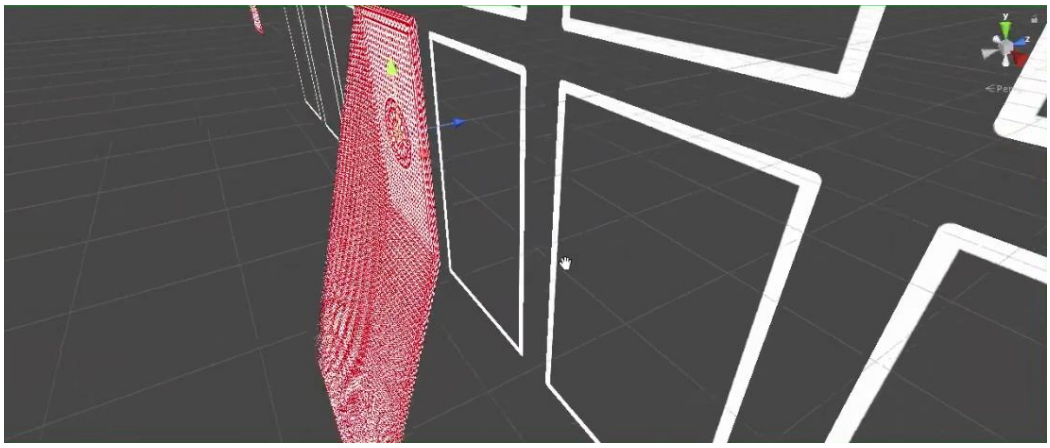

Рис. 3.9 — Стопка розданих карт у 3D-режимі

Тепер, коли у нас є карти на екрані, ми можемо оновити спрайти, щоб зовнішній вигляд карт відповідав їхнім назвам.

Ми вже створили сценарій оновлення спрайтів *UpdateSprite()* та прикріпили його до об’єкту карти. Прикріплюємо тепер спрайт зворотної сторони карт до слоту параметра *Card Back* об’єкту *Card* у редакторі.

Для полегшення реалізації ігрового процесу буде зручно, якщо інтерактивні об’єкти матимуть об’єднуючий сценарій, який буде стежити за їхніми значеннями. Для цього створюємо сценарій *Selectable* і додаємо його до префабу *Card*. В нас інтерактивними будуть карти, тому клас *Selectable* описуватиме саме карти. Додамо до цього класу поле *faceUp*, яке показуватиме, якою стороною лежить карта:

```
public class Selectable : MonoBehaviour // клас для карти
{
    public bool faceUp = false; // чи лежить карта обличчям догори
```

У сценарій оновлення спрайтів *UpdateSprite()* оголошуємо змінні для доступу до необхідних ігрових об’єктів: рендера спрайтів, а також об’єктів класів *Selectable* та *Solitaire*:

```
public Sprite cardFace;
public Sprite cardBack;
private SpriteRenderer spriteRenderer;
private Selectable selectable;
private Solitaire solitaire;
```

Щоб призначити правильну лицьову карту для новостворених карток, ми будемо використовувати масив, створений у сценарії *Solitaire*. Екземпляр префабу карти названо на основі рядка з масиву *Deck*. Місце у списку, де ім'я карти відповідає рядку в порожній колоді, збігається з номером розташування правильного спрайту у масиві спрайтів. Оскільки метод генерації колоди знаходиться у сценарії *Solitaire*, ми можемо просто викликати його. Ці дії треба виконати одноразово, тому виконуємо їх у методі *Start()*, який викликається перед оновленням першого кадру на сцені.

```
void Start()
{
    List<string> deck = Solitaire.GenerateDeck();
    solitaire = FindObjectOfType<Solitaire>();

    int i = 0;
    foreach (string card in deck)
    {
        if (this.name == card)
        {
            cardFace = solitaire.cardFaces[i];
            break;
        }
        i++;
    }
    spriteRenderer = GetComponent<SpriteRenderer>();
    selectable = GetComponent<Selectable>();
}
```

Далі нам треба запрограмувати коректне відображення карт. Якщо карта лежить обличчям догори, треба коректно відобразити її спрайт, інакше – спрайт задньої сторони. Якщо користувач клацнув по карті (тобто обрав її), пофарбуємо її у жовтий колір. Ці дії треба виконувати постійно, тому ми розміщую відповідний код в методі *Update()*, який викликається з кожним кадром оновлення зображення на сцені:

```
void Update()
{
    if (selectable.faceUp == true)
    {
        spriteRenderer.sprite = cardFace;
    }
    else {
        spriteRenderer.sprite = cardBack;
    }

    if (userInput.slot1)
    {
        if (name == userInput.slot1.name)
        {
            spriteRenderer.color = Color.yellow; // помічаємо карту
        }
        else {
            spriteRenderer.color = Color.white;
        }
    }
}
```

Після збереження сценаріїв оновлюємо префаб.

Тепер всі 52 карти перетасовано та роздано лицьовою стороною, хоча вони поки що розміщені у одному місці (рис. 3.10). Якщо клацнути по властивості *faceUp* карти в редакторі, вона повертається іншою стороною.

Тепер нам потрібен спосіб правильно розташувати карти після роздачі. Для цього створюємо масиви для стопок, де можуть розміщуватися карти.

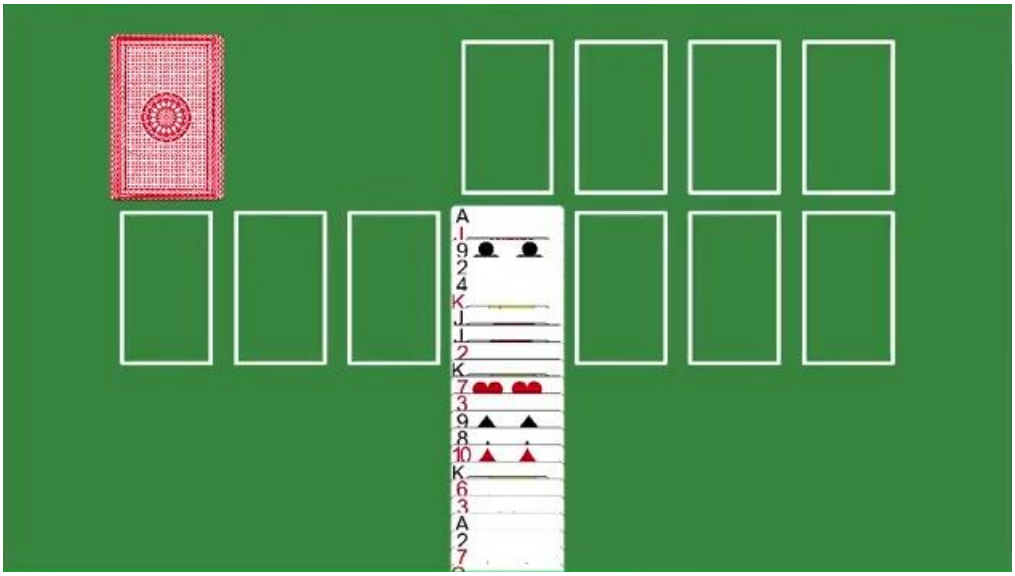

Рис. 3.10 — Стопка розданих карт обличчями догори

```
    public GameObject[] bottomPos;
    public GameObject[] topPos;

    // масив мастей
    // C - трефи, D - бубни, H - черви, H - піки
    public static string[] suits = new string[] { "C", "D", "H", "S" };
    // масив значень карт (A - туз, 2, ..., J - валет, Q - дама, K - король)
    public static string[] values = new string[] { "A", "2", "3", "4", "5",
"6", "7", "8", "9", "10", "J", "Q", "K" };
    public List<string>[] bottoms;
    public List<string>[] tops;

    private List<string> bottom0 = new List<string>();
    private List<string> bottom1 = new List<string>();
    private List<string> bottom2 = new List<string>();
    private List<string> bottom3 = new List<string>();
    private List<string> bottom4 = new List<string>();
    private List<string> bottom5 = new List<string>();
    private List<string> bottom6 = new List<string>();
```

В методі *Start()* створюємо список рядків для нижніх стопок карт:

```
    bottoms = new List<string>[] { bottom0, bottom1, bottom2, bottom3,
bottom4, bottom5, bottom6 };
```

Далі нам знадобиться метод розкладання карт по нижніх стопках:

```
void SolitaireSort()
{
    for (int i = 0; i < 7; i++)
    {
        for (int j = i; j < 7; j++)
        {
            // додаємо у нижню стопку верхню карту з колоди
            bottoms[j].Add(deck.Last<string>());
            deck.RemoveAt(deck.Count - 1); // видаляємо верхню карту з колоди
        }
    }
}
```

Коли змінна $i = 0$, в сім нижніх стопок розкладається по одній карті. В наступній ітерації, коли $i = 1$, нульова (ліва) стопка пропускається, і в шість правих стопок роздається ще по одній карті, і т. д. Таким чином, кожна наступна стопка матиме на одну карту більше.

Далі ставимо у відповідність елементам списків *Top Pos* і *Bottom Pos* зі скрипту *Solitaire* відповідні позиції *Top* і *Bottom* на сцені (рис. 3.11):

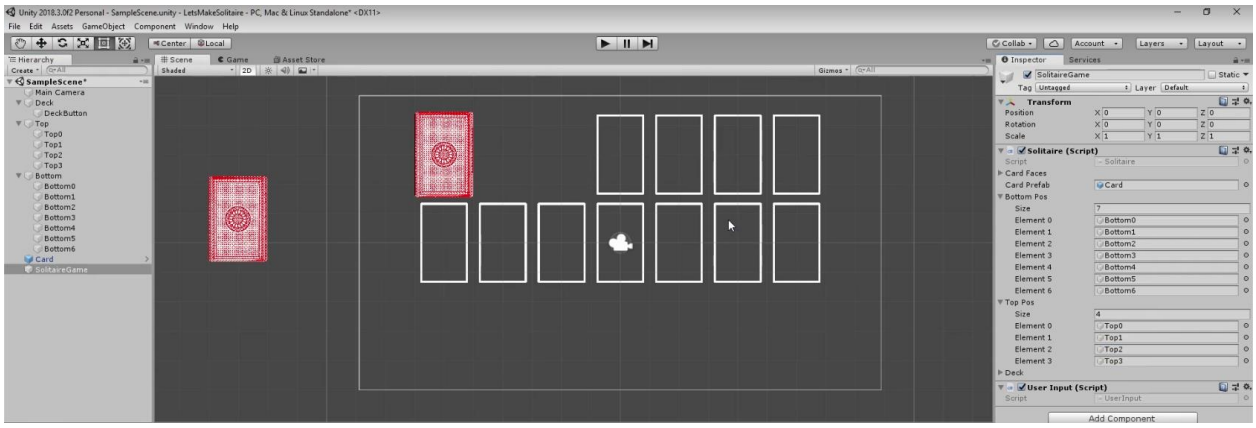

Рис. 3.11 — Зв’язування позицій карт на сцені з елементами програмних списків

Тепер треба переписати метод роздачі карт *SolitaireDeal()*: він має розташовувати карти по нижніх стопках. Крім того, він реалізуватиме тепер інтерфейс *IEnumerator*, який надає певний функціонал для перебору внутрішніх об’єктів у контейнері. Нам це знадобиться для того, щоб додати оператор *yield*, який використаний для затримки перед роздаванням наступної карти:

```
IEnumerator SolitaireDeal()
{
    for (int i = 0; i < 7; i++)
    {
        foreach (string card in bottoms[i])
        {
            yield return new WaitForSeconds(0.02f); // затримка
            // створюємо об’єкт-карту
            GameObject newCard = Instantiate(cardPrefab,
                new Vector3(bottomPos[i].transform.position.x,
                    bottomPos[i].transform.position.y - yOffset,
                    bottomPos[i].transform.position.z - zOffset),
                    Quaternion.identity, bottomPos[i].transform);
            newCard.name = card;
            newCard.GetComponent<Selectable>().row = i;
            if (card == bottoms[i][bottoms[i].Count - 1])
            {
                // лише остання карта має бути відкритою
                newCard.GetComponent<Selectable>().faceUp = true;
            }
            yOffset = yOffset + 0.5f;
            zOffset = zOffset + 0.03f;
        }
    }
```

Метод *PlayCards()*, який ми викликаємо у методі *Start()* скрипту, послідовно запускає генерацію карток в колоді, тасування колоди, сортування і роздачу карт:

```
public void PlayCards(){
    deck = GenerateDeck(); // генеруємо колоду
    Shuffle(deck); // тасуємо

    foreach (string card in deck) {
        print(card); // перевірка карт в колоді
    }
    SolitaireSort();
    StartCoroutine(SolitaireDeal());
}
```

3.6 Відстеження натискань по об’єктах

Відстеження натискань по ігрових об’єктах на сцені потребує додавання до них певних компонентів. Додаємо до об’єкту карткової колоди компоненти *Rigidbody 2D* та *Box Collider 2D* і налаштовуємо їх (рис. 3.12).

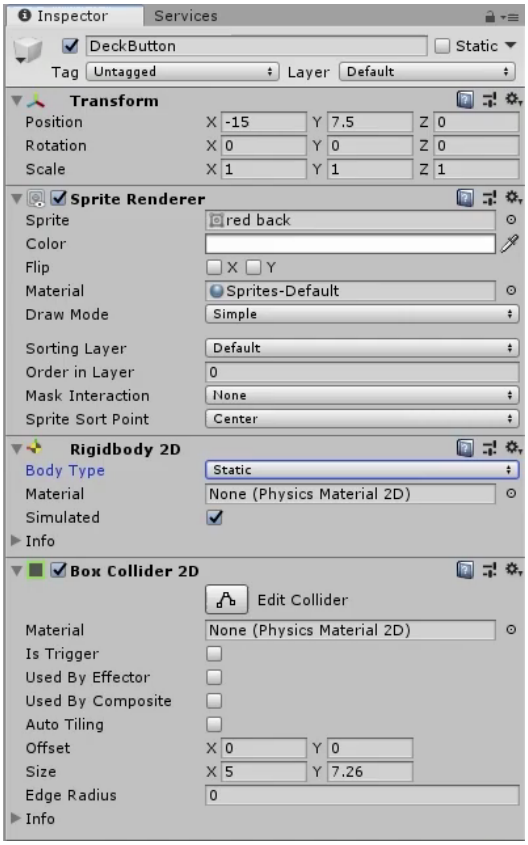

Рис. 3.12 — Параметри компонентів *Rigidbody 2D* та *Box Collider 2D* об’єкта *Deck Button*

Далі програмуємо початковий функціонал у сценарії користувацького вводу *UserInput.cs*. Нам знадобиться метод обробки клацання миші:

```
    void GetMouseClick()
    {
        if (Input.GetMouseButtonDown(0))
        {
            Vector3 mousePosition = Camera.main.ScreenToWorldPoint(new
Vector3(Input.mousePosition.x, Input.mousePosition.y, -10));
            // отримуємо об'єкт з параметрами клацання
            RaycastHit2D hit =
Physics2D.Raycast(Camera.main.ScreenToWorldPoint(Input.mousePosition),
Vector2.zero);
            if (hit)
            {
                // визначаємо, по чому клацнули: по колоді, карті,
                // верхній або нижній позиції
                if (hit.collider.CompareTag("Deck"))
                {
                    // клацнули по колоді
                    Deck();
                }
                else if (hit.collider.CompareTag("Top"))
                {
                    // клацнули по верхній позиції
                    Top(hit.collider.gameObject);
                }
                else if (hit.collider.CompareTag("Bottom"))
                {
                    // клацнули по нижній позиції
                    Bottom(hit.collider.gameObject);
                }
                else if (hit.collider.CompareTag("Card"))
                {
                    // клацнули по карті
                    Card(hit.collider.gameObject);
                }
            }
        }
    }
```

Метод *GetMouseClick()* нам треба викликати постійно, тому поміщаємо його у метод *Update()* – він викликатиметься з кожним кадром на сцені:

```
    void Update()
    {
        GetMouseClick();
    }
```

Будемо відстежувати натискання на колоду, карту, верхню або нижню порожню позицію за допомогою тегів. Тому створюємо теги «*Deck*»,

«*Card*», «*Top*» та «*Bottom*» (рис. 3.13) і вказуємо їх у параметрах відповідних об’єктів.

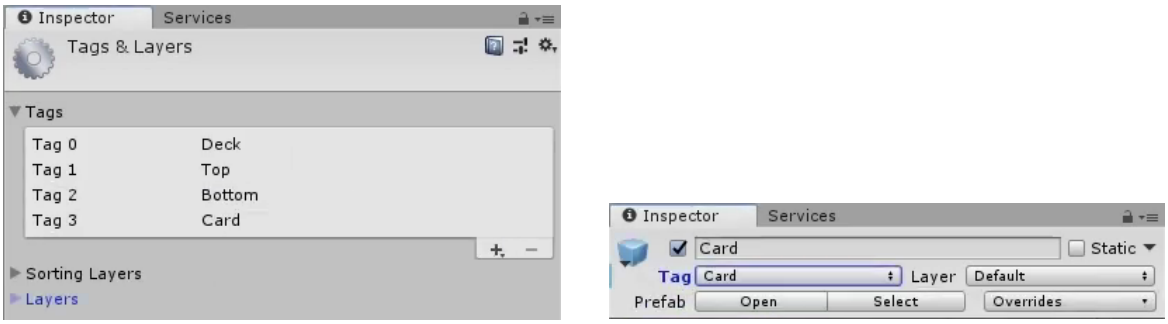

Рис. 3.13 — Додавання тегів для ідентифікації компонентів, які реагують на клацання

Для обробки натискання на колоду, карту, порожню верхню або нижню позицію створюємо заготівки відповідних функцій: *Deck(), Card(), Top()* та *Bottom()*.

До префабу карти, а також об’єктів верхніх і нижніх позицій додаємо компоненти *Rigidbody 2D* та *Box Collider 2D*.

3.7 Реалізація зняття карт з колоди

Клацання по колоді карт має знімати три карти з неї. Введемо для цього такі змінні (поля класу): список трійок карт в колоді, список знятих карт, список відкритих карт, знятих з колоди, кількість трійок карт в колоді, кількість знятих трійок з колоди і кількість у залишку карт окрім повних трійок в колоді.

```
// відкрита трійка карт
public List<string> tripsOnDisplay = new List<string>();
// масив трійок в колоді
public List<List<string>> deckTrips = new List<List<string>>();
// стопка, знята з колоди
public List<string> discardPile = new List<string>();
private int deckLocation; // кількість знятих трійок з колоди
private int trips; // кількість повних трійок карт в колоді
private int tripsRemainder; // залишок карт після зняття трійок
```

Процедуру розбиття колоди по трійках запрограмуємо у методі *SortDeckIntoTrips()*:

```
public void SortDeckIntoTrips()
{
    trips = deck.Count / 3;
    tripsRemainder = deck.Count % 3;
    deckTrips.Clear();
    // формуємо групи трійок в колоді
    int modifier = 0;
    for (int i = 0; i < trips; i++)
    {
        List<string> myTrips = new List<string>();
        for (int j = 0; j < 3; j++)
        {
            myTrips.Add(deck[j + modifier]);
        }
        deckTrips.Add(myTrips);
        modifier = modifier + 3;
    }
    // формуємо залишок карт після зняття трійок (1 або 2 карти)
    if (tripsRemainder != 0)
    {
        List<string> myRemainders = new List<string>();
        modifier = 0;
        for (int k = 0; k < tripsRemainder; k++)
        {
            myRemainders.Add(deck[deck.Count - tripsRemainder + fier]);
            modifier++;
        }
        deckTrips.Add(myRemainders);
        trips++;
    }
    deckLocation = 0;
}
```

Після розбиття карт в колоді по трійках можна їх роздавати. Для цього створюємо метод *DealFromDeck()*:

```
    public void DealFromDeck()
    {
        // додаємо карти, що залишилися, до відкритої купки
        foreach (Transform child in deckButton.transform)
        {
            if (child.CompareTag("Card"))
            {
                deck.Remove(child.name);
                discardPile.Add(child.name);
                Destroy(child.gameObject);
            }
        }

        if (deckLocation < trips)
        {
            tripsOnDisplay.Clear();
```

```
            float xOffset = 2.5f;
            float zOffset = -0.2f;

            foreach (string card in deckTrips[deckLocation])
            {
                GameObject newTopCard = Instantiate(cardPrefab,
                    new Vector3(deckButton.transform.position.x + xOffset,
deckButton.transform.position.y, deckButton.transform.position.z + zOffset),
                        Quaternion.identity, deckButton.transform);
                xOffset = xOffset + 0.5f;
                zOffset = zOffset - 0.2f;
                newTopCard.name = card;
                tripsOnDisplay.Add(card);
                newTopCard.GetComponent<Selectable>().faceUp = true;
                newTopCard.GetComponent<Selectable>().inDeckPile = true;
            }
            deckLocation++;
        }
        else {
            RestackTopDeck();
        }
    }
```

Також дещо допрацюємо метод початкової роздачі карт *SolitaireDeal()*:

```
        . . .
        foreach (string card in discardPile)
        {
            if (deck.Contains(card))
            {
                deck.Remove(card);
            }
        }
        discardPile.Clear();
    }
```

Метод *RestackTopDeck()* повертатиме зняті карти назад у колоду, коли більше немає карт для зняття в закритій колоді. Після цього можна буде знову знімати трійки карт з основної колоди.

```
    void RestackTopDeck()
    {
        deck.Clear();
        foreach (string card in discardPile)
        {
            deck.Add(card);
        }
        discardPile.Clear();
        SortDeckIntoTrips();
    }
```

При клацанні на колоді ми створили заготівку методу *Deck()*. Звідти і викликатимемо наш метод роздачі карт з колоди:

```
void Deck()
{
    solitaire.DealFromDeck();
}
```

Виклик методу *SortDeckIntoTrips()* здійснюємо в кінці методу *PlayCards()*:

```
public void PlayCards()
{
    . . .
    SortDeckIntoTrips();
}
```

3.8 Вибір і переміщення карт

Переміщення карт будемо реалізовувати наступним способом: гравець має спочатку клацнути по карті, яку хоче перемістити, а потім по позиції, куди він хоче її перемістити. Таким чином, клацання по карті може призводити до різних дій в залежності від того, чи була до цього моменту обрана якась інша карта. Отже, нам треба завести програмну змінну, призначену для відслідковування такої обраної карти:

```
public GameObject slot1;
```

Коли користувач клацає по карті, ми передаватимемо у метод *Card()* об’єкт, по якому було здійснено клацання:

```
. . .
else if (hit.collider.CompareTag("Card"))
{
    // клацнули по карті
    Card(hit.collider.gameObject);
}
```

Для перевірки того, чи може бути карта, обрана раніше, покладена на карту, по якій клацнув гравець, у сценарії *UserInput.cs* створюємо метод *Stackable()*:

```
bool Stackable(GameObject selected)
{
    // карта, яка була обрана раніше (що хочемо покласти)
    Selectable s1 = slot1.GetComponent<Selectable>();
    // цільова карта, по якій клацнули (на яку хочемо покласти обрану раніше)
    Selectable s2 = selected.GetComponent<Selectable>();
```

Для реалізації перевірки можливості класти одну карту на іншу введемо до класу *Selectable* додаткові поля:

```
public class Selectable : MonoBehaviour
{
    public bool top = false; // чи лежить карта у верхній стопці
    public string suit; // масть
    public int value; // значення
    public int row; // ряд, в якому лежить карта
                    // (зверху 4 ряди (0..3), знизу - 7 рядів (0..6))
    public bool faceUp = false; // чи лежить карта обличчям догори
    public bool inDeckPile = false; // чи лежить карта в колоді

    private string valueString; // значення карти у строковому вигляді
```

Дані для заповнення цих полів можна отримати, коли користувач клацнув карту. Вводимо до методу *Start()* логіку, яка відслідковує, що натиснута карта, і заповнює поля класу:

```
    void Start()
    {
        if (CompareTag("Card"))
        {
            suit = transform.name[0].ToString();

            for (int i = 1; i < transform.name.Length; i++)
            {
                char c = transform.name[i];
                valueString = valueString + c.ToString();
            }

            if (valueString == "A") {
                value = 1;
            }
            if (valueString == "2") {
                value = 2;
            }
            if (valueString == "3") {
                value = 3;
            }
            if (valueString == "4") {
                value = 4;
            }
            if (valueString == "5") {
                value = 5;
```

```
        }
        if (valueString == "6") {
            value = 6;
        }
        if (valueString == "7") {
            value = 7;
        }
        if (valueString == "8") {
            value = 8;
        }
        if (valueString == "9") {
            value = 9;
        }
        if (valueString == "10") {
            value = 10;
        }
        if (valueString == "J") {
            value = 11;
        }
        if (valueString == "Q") {
            value = 12;
        }
        if (valueString == "K") {
            value = 13;
        }
    }
}
```

Далі реалізуємо метод *Stackable()*:

```
if (!s2.inDeckPile) { // якщо цільова карта не в колоді
    if (s2.top) {
    // у верхній стопці повинна складатися одна масть від туза до короля
        if (s1.suit == s2.suit || (s1.value == 1 && s2.suit == null)) {
            if (s1.value == s2.value + 1) {
                return true;
            }
        }
        else {
            return false;
        }
    }
    else {
    // у нижній стопці повинні складатися кольори, що чергуються
    // від короля до туза
        if (s1.value == s2.value - 1) {
            bool card1Red = true;
            bool card2Red = true;

            if (s1.suit == "C" || s1.suit == "S") {
            // якщо масть першої карти - трефи або піки
                card1Red = false; // перша карта НЕ червона
            }
            if (s2.suit == "C" || s2.suit == "S") {
            // якщо масть другої карти - трефи або піки
                card2Red = false; // друга карта НЕ червона
            }

            if (card1Red == card2Red) {
```

```
                    // якщо обидві карти одного кольору,
                    // у нижній стопці вони не складаються
                    print("Not stackable");
                    return false;
                }
                else {
                    // інакше у нижній стопці вони можуть складатися
                    print("Stackable");
                    return true;
                }
            }
        }
    }
    return false;
}
```

Тепер ми можемо реалізувати логіку методу натискання по карті *Card()* у відповідності до алгоритму з рис. 2.2.

```
// функція обробки клацання на карті (об'єкт selected)
void Card(GameObject selected)
    {
        print("Clicked on Card");

        if (!selected.GetComponent<Selectable>().faceUp) {
        // якщо карту закрито (обличчям вниз)
            if (!Blocked(selected)) { // і при цьому не заблокована
                // перегортаємо її
                selected.GetComponent<Selectable>().faceUp = true;
                slot1 = this.gameObject;
            }
        }
        // якщо карта відкрита
        else if (selected.GetComponent<Selectable>().inDeckPile) {
        // якщо карта, по якій клацнули, знаходиться у стопці біля колоди
            if (!Blocked(selected)) { // і якщо вона не заблокована
                slot1 = selected; // обираємо її
            }
            else {
                // print("Карта в трійці заблокована");
                return;
            }
        }

        // якщо зараз немає обраних карт
        if (slot1 == this.gameObject) {
            slot1 = selected; // обираємо карту
        }

        // якщо зараз обрана якась інша карта
        else if (slot1 != selected) {
            if (Stackable(selected)) {
            //  якщо нова (кланцута) карта може бути покладена на обрану
                Stack(selected); // кладемо обрану раніше карту
            }
            else {
                slot1 = selected; // інакше обираємо цю карту
            }
```

```
        }
    }
```

Тепер створюємо методи, які ще не реалізовані. По-перше реалізуємо метод *Stack()* укладання карти до стопки, на яку вона переміщується.

```
void Stack(GameObject selected)
{
    Selectable s1 = slot1.GetComponent<Selectable>(); // карта, яку беремо
    Selectable s2 = selected.GetComponent<Selectable>(); // карта, на яку
                                                         // кладемо
    float yOffset = 0.5f;

    if (s2.top || (!s2.top && s1.value == 13))
    // якщо карта кладеться у верхню стопку або у вільну нижню стопку
    // (тоді ця карта - король), то кладемо її без зміщення по вертикалі
    {
        yOffset = 0;
    }

    slot1.transform.position = new Vector3(selected.transform.position.x,
                               selected.transform.position.y - yOffset,
                               selected.transform.position.z - 0.01f);
    // рухаємо "дітей" разом з "батьками"
    slot1.transform.parent = selected.transform;

    if (s1.inDeckPile) // якщо карта береться з колоди
    {
        // видаляємо карти з трійки біля колоди, щоб запобігти
        // дублювання карт
        solitaire.tripsOnDisplay.Remove(slot1.name);
    }
    // переміщення карти між верхніми стопками
    else if (s1.top && s2.top && s1.value == 1)
    // якщо між верхніми позиціями переміщається туз
    {
        solitaire.topPos[s1.row].GetComponent<Selectable>().value = 0;
        solitaire.topPos[s1.row].GetComponent<Selectable>().suit = null;
    }
    else if (s1.top) // якщо знімаємо карту з однієї з верхніх стопок,
    // оновлюємо значення верхньої карти у колоді після зняття карти
    {
      solitaire.topPos[s1.row].GetComponent<Selectable>().value = s1.value - 1;
// (по суті, у верхніх стопках у нас немає множини карт;
// так як вони завжди зверху відсортовані за значеннями,
// досить знати (і оновлювати) значення верхньої карти в стопці,
// щоб представляти і підтримувати всю стопку)
    }
    else // якщо картка береться з нижньої стопки
    {
        // видаляємо рядок картки із відповідного нижнього списку
        solitaire.bottoms[s1.row].Remove(slot1.name);
    }

    s1.inDeckPile = false; // не можна додавати карти в стопку трійок
    s1.row = s2.row; // "цільова" карта і карта, що переміщується,
                     // знаходяться в одній стопці
```

```
    if (s2.top) // якщо кладемо карту на карту у верхній стопці АБО
                // на порожнє місце
    {
        // фіксуємо значення нової верхньої карти у верхній стопці
        solitaire.topPos[s1.row].GetComponent<Selectable>().value = s1.value;
        // фіксуємо масть нової верхньої карти у верхній стопці
        solitaire.topPos[s1.row].GetComponent<Selectable>().suit = s1.suit;
        s1.top = true; // фиксируем, что карта лежит в "доме"
    }
    else
    {
        s1.top = false;
    }
    slot1 = this.gameObject;
}
```

Другий метод, який необхідно розробити, – метод *Blocked()*, який перевіряє, чи є передана карта заблокованою для розміщення на ній іншої карти:

```
    bool Blocked(GameObject selected)
    {
        Selectable s2 = selected.GetComponent<Selectable>();
        if (s2.inDeckPile == true)
        {
            if (s2.name == solitaire.tripsOnDisplay.Last())
            // якщо це остання карта у відкритій трійці, вона не блокується
            {
                return false;
            }
            else
            {
                print(s2.name + " is blocked by " +
                    solitaire.tripsOnDisplay.Last());
                return true;
            }
        }
        else
        {
            if (s2.name == solitaire.bottoms[s2.row].Last())
            // перевіряємо, чи внизу ця карта
            {
                return false;
            }
            else
            {
                return true;
            }
        }
    }
```

Для того, що поміщати карти у верхні стопки, нам треба створити метод *Top()*, який оброблятиме клацання по порожніх верхніх позиціях:

```
void Top(GameObject selected)
// обробка клацання по порожній верхій позиції
{
    print("Clicked on Top");
    if (slot1.CompareTag("Card"))
    {
        if (slot1.GetComponent<Selectable>().value == 1)
        // якщо карта – туз, то кладемо її на верхню позицію
        {
            Stack(selected);
        }
    }
}
```

У редакторі Unity додаємо скрипт *Selectable* до об’єктів верхніх і нижніх позицій і прописуємо у їх поля *Row* відповідні номери позицій, рахуючи з нуля зліва направо.

Аналогічно, щоб мати можливість поміщати карти у нижні стопки, створюємо метод *Bottom()*, який буде обробляти клацання по нижніх позиціях:

```
void Bottom(GameObject selected)
// обробка клацання по порожній нижній позиції
{
    print("Clicked on Bottom");

    if (slot1.CompareTag("Card"))
    {
        if (slot1.GetComponent<Selectable>().value == 13)
        // якщо карта - король, то кладемо її
        {
            Stack(selected);
        }
    }
}
```

На даному етапу вже можна повноцінно грати у пасьянс (рис. 3.14). Залишається обробити завершення гри і реалізувати деякі додаткові функції.

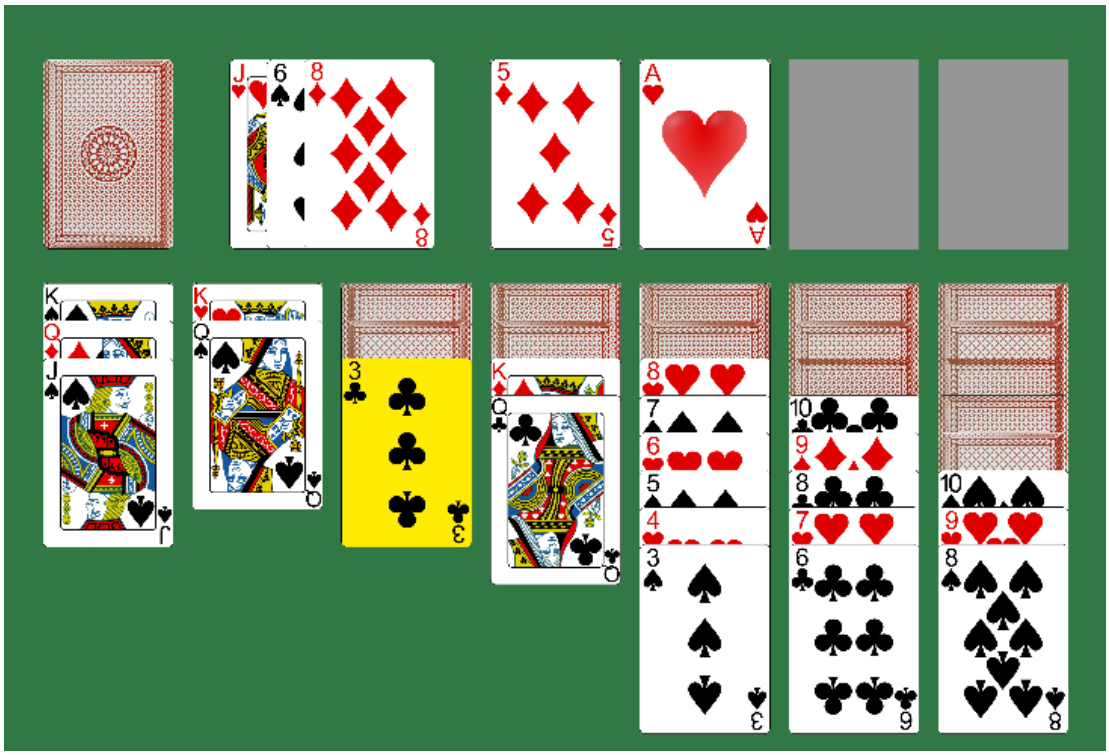

Рис. 3.14 — Зовнішній вигляд сцени в процесі гри

Висновки третього розділу

В даному розділі була виконана реалізація першого (прототипного) варіанту гри «Пасьянс Клондайк» за поставленим завданням. У наступних релізах застосунок може бути доповнений новими можливостями, такими як анімація, більш природне перетягування карт, підказки щодо наступного ходу гравця, таблицю рекордів тощо. В кінці гри можна додати феєрверк або іншу анімацію, якщо гравець збере пасьянс.

Проте, навіть у такому вигляді у гру можна грати і отримувати задоволення.

РОЗДІЛ 4
ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ПРОЕКТУ

4.1 Організаційно-економічна характеристика роботи

Об’єктом розробки в даній дипломній роботі є програмний продукт – комп’ютерна карткова гра для платформи Windows. Інструмент розробки – ігровий рушій і середовище розробки Unity.

Карткові ігри сьогодні є одним із жанрів, що найшвидше розвиваються в індустрії. Кращі комп’ютерні карткові ігри пропонують гравцям потенційно безмежну реграбельність і доступні для завантаження – не дивно, чому вони так «злетіли» в останні роки.

Карткові ігри все ще набирають сили. Їхні творці змішують батлери, карткові ігри та механіку в стилі roguelike, і на виході виходять неймовірно захоплюючі ігри на кшталт Slay the Spire або прийдешньої Griftlands.

На ринку існує багато відеоігор-пасьянсів, які відповідають різним потребам користувачів. Серед них:

- 123 FreeSolitaire – програма, що пропонує дуже естетичну графіку;
- Free Solitaire 3D – гра тривимірними картами, що можуть редагуватися і налаштовуватися;
- Big Solitaires 3D – близько 40 різновидів пасьянсу;
- Quick Bridge Portable – програма для гри в бридж з комп’ютером, яку можна легко запустити з флешки.

Областю використання даного проекту є сфера розваг, зокрема – галузь відеоігор. За обігом коштів галузь відеоігор рік у рік випереджає музичну індустрію та кіновиробництво разом узяті.

Індустрія інтерактивних розваг дає сьогодні своїм учасникам колосальний дохід. За весь час існування відеоігор змінювалася бізнес-модель

їхнього поширення, впроваджувалися різні схеми придбання контенту, а деякі компанії розробили власні економічні системи. Бюджети під час існування промисловості зростають рік у рік; і хоч кількість гравців також збільшилася, видавці вже не хочуть отримувати менше грошей. Тому економічні системи в іграх точно розвиватимуться й надалі.

За типом комп'ютерна гра, яка розробляється в даному проекті, є казуальною. Гра не потребує спеціальних пристроїв для грання.

Таблиця 4.1

Класифікаційна оцінка проекту

Клас	Монопроект
Тип	Технічний
Вид	Дослідницький
Тривалість	Короткостроковий
Ступінь складності	Проект середньої складності
Рівень	Корпоративний

На основі наведених статистичних даних можна зробити висновок, що розробка даної казуальної гри має потенціал комерційного успіху.

Ринком збуту даного продукту є світовий ринок. Потенційним споживачем може бути будь-який користувач, який хоча б іноді грає у комп'ютерні ігри. Очікуваною конкурентною перевагою являється низька ціна продукту.

Для досягнення мети були проаналізовані сучасні тенденції у розробці програмних застосунків, досліджені засоби розробки програмних застосунків і деякі проекти конкурентів.

Передбачається поширення продукту за допомогою розміщення на платформах – ігрових магазинах. Гра виконана однією людиною за короткий строк, планується виставлення мінімально ціни на усіх платформах. Реклама продукту не передбачена.

Етапи виконання розділів кваліфікаційної роботи бакалавра:

1. Постановка технічного завдання. Термін виконання – чотири тижні.
2. Розробка робочого проекту: розробка логічної моделі програми шляхом складання алгоритму. На стадії робочого проекту створюється вся необхідна документація об’єкта (специфікації, пояснювальна записка, інструкції та ін.). Термін виконання – один місяць.
3. Техніко-економічна частина: розрахунок витрат та обґрунтування економічної доцільності впровадження даного програмного продукту. Термін виконання – три тижні.

У кваліфікаційній роботі бакалавра представлений наступний склад робіт:

1. Технічне завдання. Термін виконання – від 10 до 14 днів.
2. Розробка алгоритму програмного продукту. Термін виконання – від 20 до 30 днів.
3. Розробка робочого проекту. Термін виконання – від 1 місяця до 45 днів.
4. Впровадження проекту. Термін виконання – за 1 тиждень.

Розрахунок ефективності оцінки системи управління полягає у знанні характеристик системи на момент розрахунку. Для кожної з оцінок створюється таблиця переводу діапазону можливих значень у шкалу балів в 1 до 5.

Для створення програмного продукту був виконаний аналіз існуючих методів збору та аналізу цінової політики торгових мереж.

Очікувані конкурентні переваги.

Даний продукт, являє собою простий і зручний у застосуванні інтерфейс, проста реалізація функцій, ефективність, що має забезпечити його затребуваність.

4.2 Розрахунок науково-технічної ефективності проекту

Ефективністю називається здатність приносити ефект, результативність процесу, проекту, які визначаються як відношення ефекту, результату до витрат, що забезпечили результат.

Основні категорії ефективності науково-дослідних та дослідно-конструкторських робіт (НДДКР) є економічний, науково-технічний, соціальний, маркетинговий та екологічний ефект.

Економічний ефект розробки та впровадження проекту оцінюється системою вартісних показників і проявляється у підвищенні економічної ефективності роботи підприємства, раціональному використанні ресурсів, збільшенні прибутку від реалізації продукції, покращенні використання виробничих потужностей, впровадженні корисних винаходів.

Науково-технічний ефект полягає у прирості практично орієнтовних науково-технічних знань і умінь. На етапах практичного використання науково-технічний ефект:

- підвищення науково-технічного рівня виробництва;
- підвищення організаційного рівня виробництва;
- кількість зареєстрованих охоронних документів;
- збільшення частки використання нових інформаційних технологій;
- збільшення частки нових технологічних процесів;
- зростання кількості і статусу науково-технічних публікацій;
- підвищення конкурентоспроможності підприємства та його продукції.

Соціальний ефект сприяє розвитку суспільства та задовольняє його потреби, проявляється у досягненні якісно нового рівня життя населення,

підвищенні рівня освіти та створенні нових нематеріальних цінностей й оцінюється в основному якісними показниками. Крім позитивних впливів також треба враховувати потенційні негативні впливи тривалого використання техніки під час гри. Щоб їх знизити, треба обмежувати час, який користувач (особливо, якщо він дитина) витрачає на гру.

Екологічний ефект – впливання інновацій на розв'язання проблем охорони довкілля, що особливо важливо у реалізації інноваційних проектів, які можуть змінювати рівень екологічної безпеки території.

Маркетинговий ефект відображає потреби ринку в наукових дослідженнях, реалізації маркетингової стратегії, сукупності заходів, що проводяться, виражений через збільшення показників задоволеності споживачів, поліпшення бізнес-показників компанії.

«Науково-технічну ефективність (НТЕ) результатів прикладних робіт визначаємо на основі показників науково-технічного рівня. Оцінка науково-технічної ефективності НДДКР відбувається на основі показника ($O_{НТЕ}$), який представляє собою ступінь досягнення максимально можливого рівня, значення якого дорівнює 1 (одиниці)» [1]:

$$O_{НТЕ} = K^{Ф}_{НТЕ} / K^{П}_{НТЕ}, \qquad (4.1)$$

де $K^{Ф}_{НТЕ}$ – показник (коефіцієнт) фактичного рівня науково-технічної ефективності;

$K^{П}_{НТЕ}$ – показник (коефіцієнт) потенційно можливого рівня науково-технічної ефективності (дорівнює одиниці).

Значення показника $K^{Ф}_{НТЕ}$ визначаємо на основі шкали експертних оцінок (табл. 4.2) [1].

Таблиця 4.2

Шкала експертних оцінок для виміру рівня науково-технічної ефективності проектів

№	Групи показників	Характеристика показників	Інтервал рейтингового числа	Коефіцієнт значущості показників
1	Науково-технічний рівень	Перевищує кращі світові аналоги	10	0,30
		Відповідає світовому рівню	7 – 9	
		Нижче кращих світових аналогів	5 – 6	
		Перевищує кращі вітчизняні аналоги	3 – 4	
		Відповідає вітчизняному рівню	1 – 2	
		Нижче вітчизняного рівня	0	
2	Перспективність	Першочергова значущість	8 – 10	0,25
		Значущий	5 – 7	
		Корисний	1 – 4	
3	Потенційний масштаб практичного використання	Світовий ринок	10	0,20
		Галузі національної економіки	7 – 9	
		Галузь (регіон)	3 – 6	
		Окремі підприємства (об’єднання)	1 – 2	
4	Ступінь вірогідності досягнення позитивних результатів	Великий	10	0,25
		Середній	5 – 9	
		Малий	1 – 4	

(Таблиця узята з [1])

«Об'єкт оцінки і аналоги, які порівнюють за однаковими показниками, наведеними у співставленому вигляді відхилення в значеннях кожного з показників, мають бути однаковими для варіантів, що порівнюються». [1]

4.3 Проведення оцінки науково-технічного рівня розробки

Визначаємо $К^{Ф}_{НТЕ}$ на основі експертної оцінки науково-технічного рівня розробки.

«З цією метою:

- розробляємо перелік специфічних показників, необхідних для виміру науково-технічного рівня розробки;
- формуємо групу аналогів, які реалізовані на світовому і вітчизняному ринках;
- здійснюємо відповідні розрахунки для співставлення показників і визначення балів по таблиці 4.2.

До числа специфічних показників відносять:

- для нової техніки: продуктивність, споживання інженерних ресурсів на виробітку одиниці продукції, потреба в робочих, які обслуговують обладнання, експлуатаційні витрати на одиницю продукції;
- для нових матеріалів і речовин: вміст корисних речовин для виробітки готової продукції, питома вага відходів у загальному обсязі переробленої сировини, вартість одиниці нового матеріалу, додаткові витрати на екологічну компенсацію;
- для нових технологій: якість виробленої продукції, енергоємність і трудомісткість продукції, собівартість одиниці продукції.» [1]

На основі співставлення даних таблиці 4.3 встановлюємо бали по характеристиках чотирьох груп і на цій основі розраховуємо значення інтегрального показника *НТЕ* [1]:

$$НТЕ = \Sigma Б_i \cdot К_i^3 , \qquad (4.2)$$

де $i = 1 \div 4$,

$Б_i$ – бали (рейтингове число),

$К$ – коефіцієнт значущості показників.

Таблиця 4.3

Порівняльні показники для виконання оцінки НТЕ

ПОКАЗНИКИ	Варіанти технології	
	Розробленої	співвідносної (аналога)
Рівень новизни	Високий	Середній
Якість продукції	Висока	Середня
Споживання на 1 версію продукту – тепла, Гкал – електроенергії, кВт·годину – води, $м^3$	 5 900 3	 8 1200 6
Трудомісткість виробництва, людино-годин	70	60

Рівень науково-технічної ефективності НДДКР розраховано на основі наведених даних прикладу (таблиця 4.4).

Таблиця 4.4

Експертна оцінка і розрахунок величини інтегрального показника НТЕ

№	Групи показників	Рейтинг експертів			Середня за експертними оцінками	НТЕ
		1	2	3		
1	Науково-технічний рівень	8	7	8	7,7	2,7 (7,7 · 0,35)
2	Перспективність	7	6	8	7	2,45 (7 · 0,35)
3	Потенційний масштаб практичного використання	7	8	7	7,3	1,46 (7,3 х 0,20)
4	Ступінь вирогідності досягнення позитивних результатів	9	8	9	8,66	0,87 (8,66 · 0,1)
Всього						7,48

$$НТЕ = 7,7 \cdot 0,35 + 7 \cdot 0,35 + 7,3 \cdot 0,20 + 8,66 \cdot 0,1 = 2,7 + 2,45 + 1,46 + 0,87 = 7,48.$$

Отриманий результат слід порівняти з максимально можливим значенням, яке дорівнює 10 балам ($10 \cdot 0{,}35 + 10 \cdot 0{,}35 + 10 \cdot 0{,}20 + 10 \cdot 0{,}1$).

Отже, оцінка рівня *HTE* може бути зроблена за допомогою інтегрального коефіцієнта оцінки *HTE* (K_{HTE}) [1]:

$$K_{HTE} = \frac{\text{HTE}}{10} \cdot 100\ \%. \qquad (4.3)$$

На основі даних таблиці 4.3, можна дійти до висновку, що K_{HTE} відповідає 74,8 %, тобто:

$$\frac{7{,}48}{10} \cdot 100 = 74{,}8\ \%.$$

В тому випадку, коли значення K_{HTE} перевищує середнє значення, яке дорівнює 5,0, має бути зроблено висновок про достатній рівень HTE:

- цілком достатній 5,0 – 6,0;
- достатній 6,1 – 8,0;
- достатньо високий 8,1 – 9,0;
- високий 9,1 – 10.

Рівень HTE технології визначаємо достатнім. Отже, розроблений ПП пропонується випускати на ринок.

4.4 Розрахунок економічної ефективності проекту

4.4.1 Визначення складності розробки програмного продукту

Тривалість розробки програмного продукту (ПП) залежить від кількості його компонентів, складності його розробки, кваліфікації персоналу, а також від запланованих термінів, продиктованих ринковими умовами. Вихідними даними для визначення складності розробки ПП є кількість програмного забезпечення в тисячах умовних машинних команд аналогової програми. Вибравши аналог програмного засобу (ПЗ), що містить V_0

умовних машинних команд. У даному проекту розробляється новий програмний продукт, який відповідає аналогу ПЗ оптимізаційних розрахунків с $V_0 = 5500$ умовних машинних команд із трудомісткістю $T_р = 312$ чол/год.

Трудомісткість розробки ПП повинна включати розробку наступних етапів:

- технічного завдання – ТЗ;
- технічного проекту – ТП;
- робочого проекту – РП;
- впровадження – ВП.

Трудомісткість розроблювального ПП визначається на кожному етапу окремо на підставі трудомісткості аналога, з урахуванням складності розробки, ступеня новизни та ступеня використання в розробці стандартних модулів на підставі формул 4.4 – 4.7:

$$T_{тз} = T_р \cdot L_1 \cdot K_н \quad (4.4)$$

$$T_{тп} = T_р \cdot L_2 \cdot K_н \quad (4.5)$$

$$T_{рп} = T_р \cdot L_3 \cdot K_н \cdot K_т \quad (4.6)$$

$$T_{вн} = T_р \cdot L_4 \cdot K_н \quad (4.7)$$

де $T_р$ – укрупнена норма часу на розробку аналога ПЗ, чол/год, що коректується поправочним коефіцієнтом, що враховує умови розробки ПЗ, тобто в умовах комп'ютера, $K_н = 0{,}8$. У даному проекті $T_р = 312 \cdot 0{,}8 = 250$ чол/год;

L_j – питома вага і-го етапу розробки. У даному проекті залежно від ступеня новизни проекту (B): $L_1 = 0{,}12$; $L_2 = 0{,}15$; $L_3 = 0{,}5$; $L_4 = 0{,}12$;

$K_н$ – поправочний коефіцієнт, що враховує ступінь новизни, у даній роботі $K_н = 1{,}2$;

$К_т$ – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм, $К_т = 0,6$.

При розрахунках прийняті наступні об’єми розробленої документації по етапах проекту:

- $N_{тз} = 6$ – кількість сторінок технічного завдання;
- $N_{тп} = 9$ – кількістьсторінок технічного проекту;
- $N_{рп} = 27$ – кількість сторінок робочого проекту;
- $N_{інстр} = 5$ – кількість сторінок інструкції по налагодженню та впровадженню;
- $N_{пр} = 62$ – кількість сторінок пояснювальної записки.

4.4.2 Розрахунок трудомісткості розробки програмного продукту

Технічне завдання:

$$T_{тз} = T_р \cdot L_1 \cdot К_н = 250 \cdot 0,12 \cdot 1,2 = 36;$$

$$\underline{T}_{кк} = 0,7 \cdot N_{тз} = 0,7 \cdot 6 = 4,2;$$

$$T_{нк} = 0,15 \cdot N_{тз} = 0,15 \cdot 6 = 0,9.$$

Розробка технічного проекту (алгоритму й блок-схеми).

$$T_{тз} = T_р \cdot L_2 \cdot К_н = 250 \cdot 0,15 \cdot 1,2 = 45;$$

$$T_{кк} = 0,7 \cdot N_{тп} = 0,7 \cdot 9 = 6,3;$$

$$T_{нк} = 0,15 \cdot N_{тп} = 0,15 \cdot 9 = 1,35.$$

Розробка робочого процесу:

$$T_{тз} = T_р \cdot L_3 \cdot К_н \cdot К_т = 250 \cdot 0,5 \cdot 1,2 \cdot 0,6 = 90;$$

$$T_{кк} = 0,7 \cdot N_{тп} = 0,7 \cdot 9 = 6,3;$$

$$T_{нк} = 0{,}15 \cdot N_{тп} = 0{,}15 \cdot 9 = 1{,}35.$$

Налагодження і впровадження:

$$T_{тз} = T_{р} \cdot L_4 \cdot K_{н} = 250 \cdot 0{,}12 \cdot 1{,}2 = 36;$$

$$T_{кк} = 0{,}7 \cdot N_{інст} = 0{,}7 \cdot 5 = 3{,}5;$$

$$T_{нк} = 0{,}15 \cdot N_{інст} = 0{,}15 \cdot 5 = 1{,}5.$$

Пояснювальна записка:

$$T_{тз} = 1{,}5 \cdot N_{пз} = 1{,}5 \cdot 62 = 93;$$

$$T_{кк} = 0{,}7 \cdot N_{пз} = 0{,}7 \cdot 62 = 43{,}4;$$

$$T_{нк} = 0{,}15 \cdot N_{пз} = 0{,}15 \cdot 62 = 9{,}3.$$

Всього:

$$T_{р} = 36 + 45 + 90 + 36 + 93 = 300;$$

$$T_{кк} = 4{,}2 + 6{,}3 + 6{,}3 + 3{,}5 + 43{,}4 = 63{,}7;$$

$$T_{нк} = 0{,}9 + 1{,}35 + 1{,}35 + 0{,}75 + 9{,}3 = 13{,}65;$$

$$T_{общ} = 300 + 63{,}7 + 13{,}65 = 377{,}35.$$

Тривалість розробки ПП у днях визначається за формулою 4.8:

$$T_{пп} = \sum T_{ij} / (8{,}0 \cdot 0{,}73) \text{ (днів)}, \qquad (4.8)$$

де $\sum T_{ij}$ – сумарна тривалість розробки, год;

8 – тривалість робочого дня (коефіцієнт переведення в робочі дні), год;

0,73 – коефіцієнт переведення в календарні дні;

T_{ij} – трудомісткість j-го виду робіт по i-му етапу.

$$T_{пп} = 377{,}35 / (8 \cdot 0{,}73) = 64{,}61 \text{ днів.}$$

Порівнюючи отримані результати з розрахунками тривалості розробки ПП за допомогою мережевої моделі, вибираємо для подальших розрахунків останнє значення, так як воно, на наш погляд, ближче до фактичної тривалостірозробки ПП.

4.4.3 Визначення вартості програмного продукту

Ціна визначається за формулою:

$$Ц = К \cdot В_{ндр} + П_{р}, \quad (4.9)$$

де $В_{ндр}$ – витрати на розробку програмної продукції (кошторисна собівартість);

$К = 1{,}1$;

$П_{р}$ – нормативний прибуток, розраховується за формулою:

$$П_{р} = С_{м} \cdot Р_{н} / 100, \quad (4.10)$$

де $Р_{н}$ – норматив рентабельності, %; $Р_{н} = 25$ %;

$С_{м}$ – матеріальні витрати.

$В_{ндр}$ визначаються на підставі складання кошторису витрат на проведення НДР у таблиці 4.9.

При визначенні витрат на матеріали враховують: вартість сировини та матеріалів для проведення досліджень з урахуванням додаткових накладних витрат (витрат на транспорт, комісійних зборів тощо), вартість канцелярських матеріалів (паперів тощо), вартість ліцензій інтернет-сервісів (табл. 4.5).

Таблиця 4.5

Витрати на матеріали

Найменування матеріалів	Ціна, грн/од	Кількість	Сума, грн.
Папір	330	2	660
Ручки	22,4	2	44,8
Інформаційні носії	550	1	550
Чемодан фасилітатора	14495	1	14495
Разом			15749,8

Витрати по заробітній платі визначаються як сума заробітної плати усіх учасників НДР. Орієнтовний склад учасників, ступінь їх участі у НДР та заробітна плата наведені у таблиці 4.6.

Таблиця 4.6

Орієнтовний склад учасників НДР, їх заробітна плата та ступінь участі

Учасник НДР	Місячна заробітна плата, грн	Тривалість роботи, днів	Ступінь участі, %
Студент – дослідник	7000	150	100
Науковий керівник роботи	12000	44	25

Основна заробітна плата виконавця з урахуванням окладу і часу.

$$В_{зп} = \frac{\sum З_i \cdot K_0 \cdot \tau_i}{Д_р} = \frac{(7000 \cdot 150) + (1200 \cdot 0{,}25 \cdot 44)}{22} = 52344{,}64 \text{ грн},$$

де $З_i$ – середньомісячний оклад;

$Д_р$ – середня кількість робочих днів ($Д_р = 22$);

τ_i – трудомісткість робіт ($\tau_i = 150$);

K_0 – коефіцієнт обліку окладу керівників і консультантів проекту ($K_0 = 0{,}1$).

$$B_{зд} = 52344,64 \cdot 0,1 = 5234,5 \text{ грн.}$$

Єдиний соціальний внесок беруть у розмірі 22 % від величини заробітної плати $C_{сс} = 0,22 \cdot C_{зд}$.

Додаткова заробітна плата $C_{зд} = C_{зо} \cdot K_{д}$,

де $K_{д}$ – коефіцієнт відрахувань на заробітну плату, $K_{д} = 0,1$.

$$C_{сс} = 0,22 \cdot (52344,64 + 5234,5) = 12667,4 \text{ грн.}$$

Амортизаційні відрахування $C_{а}$ беруть від вартості основних виробничих фондів за встановленими нормативами до кожної групи фондів, які використовують при проведенні НДР (основного та додаткового обладнання, комп'ютерної техніки, інших фондів, крім приміщення). Амортизаційні відрахування необхідно розраховувати, виходячи з терміну їх використання.

Таблиця 4.7

Розрахунок сум амортизаційних відрахувань за відповідний рік

Група ОЗ	Елементи основних засобів	Первісна вартість, грн	На, %	Вартість, грн.
1	Комп'ютерна техніка	20000	25	5000
Всього				5000

Річна вартість споживаної електроенергії $C_{ел}$ визначається по формулі:

$$C_{ел} = M_{у} \cdot T_{ко} \cdot Ц_{е} \cdot K_{і},$$

де $M_{у}$ – установлена сумарна потужність комп'ютерного устаткування, кВт;

$T_{ко}$ – річний фонд роботи ЕОМ з урахуванням часу на профілактичні огляди;

$Ц_{е}$ – вартість 1 квт-години електричної енергії;

$K_{і}$ – коефіцієнт інтенсивного використання потужності (рекомендоване значення $K_{і} = 0,9$).

$$C_{ел} = 0,6 \cdot 1224 \cdot 3,45 \cdot 0,9 = 2280,31 \text{ грн.}$$

Отже, витрати на електроенергію складуть 2280,31 грн.

Таблиця 4.8

Кошторис витрат на проведення прикладних НДР

Найменування статей витрат	Сума витрат, грн
1. Матеріали	15749,8
2. Заробітна плата (основна і додаткова)	52344,64
3. Єдиний соціальний внесок	12667,4
4. Амортизаційні відрахування	5000
5. Витрати на електроенергію	2280,31
6. Виробнича собівартість	88042,15
7. Адміністративні витрати 3,5% від виробничої собівартості	3081,5
8. Витрати на збут – 1% від виробничої собівартості	880,4
Всього повна собівартість	92004,05

Нормативний прибуток:

$$П_р = (92004,05 – 15749,8) \cdot 0,25 = 19063,5 \text{ грн.}$$

Ціна програмного продукту складе:

$$Ц = 1,1 \cdot 92004,05 + 19063,5 = 120267,95 \text{ грн.}$$

4.5 Визначення капітальних витрат

Розрахунок капітальних витрат, пов’язаних з впровадженням (вдосконаленням) ПП здійснюється за формулою:

$$К = К_п + К_{ко} + К_{во} + К_с, \qquad (4.11)$$

де $К_п$ – довиробничі витрати; $К_п$ = Ц = 120267,95 грн;

$К_{ко}$ – вартість комп'ютерного устаткування: $К_{ко}$ = 5000 грн;

$К_{во}$ – вартість допоміжного устаткування: $К_{во}$ = 250 грн;

$К_с$ – вартість будівництва у зв'язку з впровадженням: $К_с$ = 0.

$$К = 120267,95 + 5000 + 250 + 0 = 125517,95 \text{ грн.}$$

Розрахунок поточних (експлуатаційних) витрат:

$$С = С_{опл} + С_а + С_{ел} + С_п + С_р + С_{всп}, \quad (4.12)$$

де $С_{опл}$ – річний фонд основної і додаткової оплати праці персоналу;

$С_а$ – сума річних амортизаційних відрахувань від вартості основного і допоміжного устаткування;

$С_{ел}$ – вартість витрат на енергію за рік;

$С_р$ – вартість річного ремонту: 6 % $К_{ко}$ = 300 грн;

$С_{всп}$ – річна вартість допоміжних матеріалів, пов'язаних з експлуатацією ПП: 2 % $К_{ко}$ = 100 грн;

$С_п$ – вартість утримання приміщень; $С_п$ = 2500 грн.

Моє програмне забезпечення буде корисне для компаній та звичайних користувачів, що шукають вигідну пропозицію для придбання товарів загального вживання в мережі Інтернет.

Річний фонд заробітної плати. До впровадження програмного продукту дана задача вирішувалася двома співробітниками з окладом по 12000 грн/міс:

$$З_{осн}^{доп} = (12000 \cdot 2) \cdot 12 = 288000 \text{ грн.}$$

Після впровадження програмного продукту чисельність фахівців скоротилася до одного фахівця – оператора комп'ютерного набору 8000 грн.:

$$З_{осн}^{доп} = 8000 \cdot 12 = 96\,000 \text{ грн.}$$

Фонд додаткової заробітної плати:

$$З_{доп} = З_{осн} \cdot К_{доп},$$

де $К_{доп}$ – коефіцієнт додаткової заробітної плати: визначається в розмірі $К_{доп} = 0,1$.

$$З_{осн}^{до} = 288\,000 \cdot 0,1 = 28800 \text{ грн.}$$

$$З_{осн}^{після} = 96\,000 \cdot 0,1 = 9600 \text{ грн.}$$

Єдиний соціальний внесок 22 %:

$$З_{єсв}^{до} = (288\,000 + 28800) \cdot 0,22 = 69696 \text{ грн.}$$

$$З_{єсв}^{після} = (96\,000 + 9600) \cdot 0,22 = 23232 \text{ грн.}$$

Загальні витрати на оплату праці:

$$С_{опл} = З_{осн} + З_{доп} + З_{єсв}, \qquad (4.13)$$

Разом:

$$С_{опл}^{до} = 288000 + 28800 + 69696 = 386496 \text{ грн.}$$

$$С_{опл}^{після} = 96000 + 9600 + 23232 = 128832 \text{ грн.}$$

Розрахунок амортизаційних відрахувань визначається за формулою:

$$С_а = К_{кко} \cdot Н_а / 100$$

де $Н_а$ – норма амортизаційних відрахувань ($Н_а = 60$ %).

$$С_а = 5000 \cdot 0,6 = 3000 \text{ грн.}$$

Річна вартість споживаної електроенергії, визначається за формулою:

$$C_{ел} = M_у \cdot T_{ко} \cdot Ц_э \cdot K_и \quad (4.14)$$

де $M_у$ – установлена сумарна потужність комп'ютерного устаткування: $M_у$ = 0,6 кВт;

$T_{ко}$ – річний фонд роботи ЕОМ з урахуванням часу на профілактичні огляди: $T_{ко}$ = 5500 год;

$Ц_э$ – вартість 1 кВт – години ел. енергії 3,45 грн;

$K_и$ – коефіцієнт інтенсивного використання потужності, $K_и$ = 0,9.

$$C_{ел} = 0,6 \cdot 5500 \cdot 3,45 \cdot 0,9 = 10246,5 \text{ грн.}$$

Разом:

$$C = C_{опл} + C_а + C_{ел} + C_р + C_п:$$

$$C^{до} = 386496 \text{ грн (тільки вартість труда).}$$

$$C^{після} = 128832 + 3000 + 10246,5 + 300 + 100 + 2500 = 144950,5 \text{ грн.}$$

Річна економія на поточних витратах: $Э_г = (C^{до} - C^{після}) + \Delta П$,

де $\Delta П$ – приріст прибутку господарчого суб'єкта та його структурного розподілу ΔП = 0.

$$Э_г = 386496 - 144950,5 + 0 = 241545 \text{ грн.}$$

$$Э_о = 241545,2 - 0,25 \cdot 125517,95 = 210165,7 \text{ грн.}$$

Рентабельність інвестицій:

$$Е = \frac{Э_г}{К_п} \cdot 100\% = \frac{241545,2}{К_п} \cdot 100\% = 199\% \quad (4.15)$$

199 % > 25 % – відповідно, проект ефективний.

Строк окупності витрат проекту складає до одного року.

Таблиця 4.9

Техніко-економічні показники програмного продукту

Найменування показників	Одиниця виміру	Значення показника	
		до впрова-дження проекту	після впрова-дження проекту
Трудомісткість розробки проекту	дні		151
Ціна ПП	грн		115 339,45
Капітальні витрати	грн	–	125517,95
Поточні витрати	грн/рік	386494	144950
Економічний ефект від реалізації проекту	грн/рік		241545,2
Строк окупності	роки		0,5
Рентабельність	%		199

Висновки четвертого розділу

Техніко-економічні показники проекту показують, що:

- трудомісткість розробки не перевищують запланований термін;
- ціна ПП, капітальні та поточні витрати дозволяють окупити проект в термін до одного року;
- рентабельність проекту показує високу якість цінової та комерційної стійкості проекту;

Економічна ефективність цього проекту є перевагою порівняно зі замовленням комп’ютерної гри у сторонньої організації.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ НА РОБОЧОМУ МІСЦІ

Дана дипломна робота присвячена створенню комп'ютерної гри. Оскільки об'єкт розробки є типовим представником програмного забезпечення, а розробники ігор працюють за персональним комп'ютером (ПК) багато годин, вони мають слідувати правилам і рекомендаціям, що стосуються охорони праці з комп'ютером.

5.1 Загальні положення

Продуктивність та характер діяльності людини прямо пов'язані з виконанням конкретних робіт та ефективністю праці. Остання визначається як впливом людського фактору, так і використанням засобів виробництва, а також умовами, пов'язаними з технологією та організацією праці. В сучасній промисловій сфері більшість працівників займається роботою, пов'язаною з використанням комп'ютерної техніки. Працюючи за комп'ютером, людина стикається з різноманітними факторами, такими як електромагнітні поля, інфрачервоне та іонізуюче випромінювання, шум, вібрації та статична електрика.

Робота за комп'ютером супроводжується нервовим навантаженням та потребує значної ментальної напруги для операторів. Вона також вимагає високого рівня зорової активності та значного фізичного навантаження на руки під час взаємодії з клавіатурою. Раціональна конструкція та розташування елементів робочого місця мають велике значення для забезпечення оптимальної робочої позиції під час праці за комп'ютером.

Під час роботи з комп'ютером важливо дотримуватись належного режиму праці та відпочинку. В іншому випадку працівники можуть

відчувати незадоволення роботою, головний біль, роздратування, порушення сну, втому та відчувати біль в очах, спині, шиї та руках.

Обчислювальна техніка виділяє тепло, що може призвести до підвищення температури та зниження вологості у приміщенні.

Рівень шуму на робочому місці оператора не повинен перевищувати встановлені норми. Для зниження рівня шуму стіни та стеля в приміщенні, де знаходяться комп'ютери, повинні бути облицьовані матеріалами, що поглинають звук.

Оптичне випромінювання включає ультрафіолетове (УФ), видиме світло та інфрачервоне випромінювання.

В першу чергу, ультрафіолетове (УФ) випромінювання впливає на шкіру та очі людини. Проте, аналіз досліджень робочих місць користувачів комп'ютерів показує, що у 86 % випадків УФ-випромінювання не було виявлено.

Світлове випромінювання, переважно, впливає на очі і може викликати втому та запалення райдужної оболонки. Однак ці симптоми швидко зникають і не спричиняють патологічних змін.

Електромагнітне випромінювання (ЕМВ) у радіочастотному діапазоні є основним джерелом ЕМВ в робочому середовищі, зокрема від монітора. Тому при обиранні місця для комп'ютера необхідно враховувати, що задня і бокові стінки можуть бути джерелом значно більшого ЕМВ, ніж сам екран.

Наукові дослідження свідчать про те, що радіочастотне випромінювання впливає на центральну нервову систему (ЦНС) і є значним стрес-фактором.

Щоб зменшити вплив згаданих видів випромінювання, рекомендується використовувати монітори з низьким рівнем випромінювання, а також дотримуватись регламентованого режиму праці та відпочинку.

5.2 Способи зниження впливу шкідливих та небезпечних факторів при роботі з комп'ютером

Щодо зниження впливу шкідливих та небезпечних факторів під час роботи з комп'ютером, необхідно, зокрема, правильно розташовувати обладнання та електричні кабелі, щоб уникнути ризику ураження електричним струмом. Інші заходи, пов'язані з електробезпекою, відповідають загальним вимогам щодо пожежної та електробезпеки.

Екран монітора повинен бути розташованим перпендикулярно до напрямку погляду. Якщо він нахилений, це може спричинити погіршення постави. Відстань між очима та екраном повинна трохи перевищувати звичайну відстань між очима та книгою. Якщо на моніторі, особливо на старих моделях, відсутній захисний екран, необхідно сидіти на відстані витягнутої руки від нього. Ще одним аспектом, що стосується зору, є створення неоднорідного поля зору. Це можна досягти, розмістивши на стінах плакати або картини з спокійними кольорами, наприклад, пейзажі або натюрморти.

Форма спинки крісла має відповідати формі спини. Висота крісла має бути така, щоб користувач не відчував тиску на стегна або куприк. Крісло бажано обладнати підлокітниками, а його розташування має бути зручним, щоб не доводилося напружуватись, щоб дістатися до клавіатури. Дуже важливими є регулярні переміщення та зміна положення тіла, а також перерви у роботі.

Під час напруженої роботи за комп'ютером рекомендується робити перерви тривалістю 15 хвилин кожну годину і займатися іншими справами. Кілька разів на годину корисно виконати серію легких вправ для розслаблення.

Якщо не дотримуватись заходів безпеки під час роботи за комп'ютером, можуть виникнути такі наслідки, як:

– проблеми з органами зору (спостерігаються у 60 % користувачів);

- захворювання серцево-судинної системи (20 % користувачів);
- захворювання шлунково-кишкового тракту (10 % користувачів);
- шкіряні проблеми (5 % користувачів);
- ризик виникнення різноманітних пухлин.

Якщо у приміщенні використовується більше одного комп'ютера, важливо враховувати, що на користувача може впливати випромінювання від інших комп'ютерів, зокрема з бокових, та задньої стінки сусідніх дисплеїв. Тому необхідно встановити спеціальні фільтри та забезпечити, щоб користувач розміщувався на відстані не менше одного метра від бічних і задніх стінок інших дисплеїв.

Отже, для запобігання негативним впливам важливо бути ознайомленим з небезпечними аспектами самого комп'ютера, правилами безпечної роботи з ним, засобами запобігання ризикам, особливо пов'язаним з відомими загрозами, такими як електричні ураження та пожежна небезпека.

5.3 Правила безпеки при роботі з комп'ютером

Перед початком роботи на комп'ютері користувач повинен переконатися у цілісності корпусу та компонентів комп'ютера, а також перевірити наявність заземлення, стан та цілісність живильних кабелів та їх правильне підключення. У разі виявлення несправностей вмикати комп'ютер та розпочинати роботу заборонено.

Під час роботи, після переконання у справності обладнання, можна увімкнути живлення комп'ютера і почати роботу, дотримуючись умов, зазначених у відповідному посібнику з експлуатації.

Заборонено:

- замінювати різні деталі або компоненти під час роботи комп'ютера;
- з'єднувати або від'єднувати вилки та розетки живильної мережі, які знаходяться під напругою;

- відкривати кришки, що закривають доступ до струмопровідних частин живильної мережі під час роботи обладнання;
- використовувати паяльник з не заземленим корпусом;
- замінювати запобіжники під напругою;
- залишати комп'ютер увімкненим без нагляду.

Після закінчення робочого дня необхідно:

- вимкнути живлення комп'ютера, натиснувши відповідну кнопку та вийнявши вилку живильного кабелю з розетки, дотримуючись інструкцій з експлуатації;
- прибрати робоче місце користувача комп'ютера, відклавши використане обладнання та матеріали на призначені для них місця;
- у разі виявлення дефектів під час роботи комп'ютера – повідомити відповідним посадовим особам та спеціалістам.

5.4 Пожежна профілактика

Для забезпечення пожежної безпеки потрібно, в першу чергу, визначити типи та кількість первинних засобів пожежогасіння. При цьому необхідно враховувати фізико-хімічні та пожежонебезпечні властивості горючих речовин, їх взаємодію з вогнегасниками, а також площу виробничих приміщень, установок та відкритих майданчиків.

Для загасання невеликих вогнищ пожеж, де горіння не може відбуватися без доступу повітря призначені азбестові полотна, грубошерстяні тканини та повсті розміром не менше 1 кв. м.

У пожежному стенді мають бути ємності для піску об'ємом не менше 0,1 куб. м. Конструкція ящика з піском має забезпечувати зручний доступ до нього і запобігати потраплянню опадів або проникненню вологи іншими шляхами.

Комплектація технологічного обладнання вогнегасниками повинна відповідати вимогам технічних умов (паспортів) на це обладнання або відповідним правилам пожежної безпеки.

Необхідно вибирати тип вогнегасників і розраховувати їх необхідну кількість залежно від їх вогнегасної здатності, максимальної площі, класу пожежі горючих речовин і матеріалів у закритих приміщеннях або на об'єктах згідно з ISO N 3941-77.

У замкнутих приміщеннях об'ємом до 50 куб. м для загасання пожеж можна використовувати портативні вогнегасники або додатково використовувати порошкові вогнегасники.

При виборі вогнегасника з врахуванням відповідної температурної межі використання необхідно враховувати кліматичні умови, в яких будуть експлуатуватися будівлі та споруди.

Вогнегасники, які були відправлені на перезарядку з підприємства, повинні бути замінені на відповідну кількість заряджених вогнегасників.

При захисті приміщень з ПК слід враховувати особливості взаємодії вогнегасних речовин з обладнанням, виробами, матеріалами та іншими елементами. Для цих приміщень рекомендується встановлювати вуглекислотні вогнегасники, враховуючи максимально допустиму концентрацію вогнегасної речовини.

Приміщення, які обладнані автоматичними стаціонарними установками пожежогасіння, повинні мати вогнегасники на 50 % від їх розрахункової кількості. Відстань від можливого вогнища пожежі до місця розташування вогнегасника не повинна перевищувати: 20 м для громадських будівель і споруд, 30 м для приміщень категорій А, Б і В, 40 м для приміщень категорії Г та 70 м для приміщень категорії Д.

Використання первинних засобів пожежогасіння для господарських та інших потреб, які не пов'язані з гасінням пожежі, заборонено.

З урахуванням типу будівлі (громадське приміщення) та можливого класу пожежі (Е), оскільки у приміщенні є багато комп'ютерів, визначаємо необхідну кількість вогнегасників – один порошковий вогнегасник об'ємом 5 літрів.

Висновки п’ятого розділу

У даній частині дипломного проекту були розглянуті питання щодо гігієнічних норм організації і обладнання робочих місць користувачів персональних комп’ютерів, питання пожежної профілактики приміщень, виконані необхідні розрахунки об’єму недоторканого запасу води для зовнішнього пожежогасіння. Отримані результати дозволяють обладнати приміщення для роботи користувачів обчислювальної техніки з дотриманням вимог безпеки їх праці.

ЗАГАЛЬНІ ВИСНОВКИ

В результаті виконаної роботи

. . .

Також були досліджені суміжні питання охорони праці і виконано техніко-економічне обґрунтування проекту.

Незважаючи на те, що перший рівень реалізації гри ще не дозволяє викласти її в загальний доступ, закладені ідеї і механізми гри можуть стати основою її майбутньої популярності.

Перспективи розвитку гри для конкуренції з аналогами:

- покращення графічного оформлення ігрового поля, фону та інших елементів зовнішнього вигляду, створення для деяких з них анімації, можливо, розробка *3D*-моделі ігрового поля;
- реалізація підказок у грі;
- розробка штучного інтелекту для аналізу партій;
- реалізація рейтингу і режиму змагань користувачів;
- у майбутньому можна зробити гру мережевою і багатокористувацькою, додавши можливість декільком гравцям діяти в одному ігровому просторі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Басюркіна Н. Й., Свистун Т. В. Методичні вказівки до оцінки науково-технічної ефективності розробки нової технології, нового обладнання та інших інновацій. Для студентів всіх спеціальностей СВО «бакалавр» і «магістр» денної і заочної форм навчання. – Одеса : ОНТУ, 2022. – 18 с.
2. Вігуржинська С. Ю, Колесник В. І. Дипломне проектування економічної частини проекту: Методичні вказівки для студентів, що навчаються за комп'ютерними спеціальностями "Інформаційні управляючі системи та технології", "Інформаційні технології проектування", "Комп'ютерні системи та мережі" та "Спеціалізовані комп'ютерні системи". – Одеса: ОНАХТ, 2016. – 22 с.
3. Вісловух А. М. Охорона праці користувачів персональних комп'ютерів (ПК): Навчальний посібник. – К.: ІПК ДСЗУ, 2007. – 55с.
4. Князєва Н. О. Дипломне проектування: Методичні вказівки до дипломної роботи спеціаліста / Н. О. Князєва, С. В. Шестопалов, С. Л. Жуковецька. — Одеса : ОНАХТ, 2016. — 46 с.
5. Boyes, Emma. GDC '08: Are casual games the future?– 19.02.2008. [Електронний ресурс]. — Режим доступу: *https://www.cnet.com/tech/gaming/gdc-08-are-casual-games-the-future/.*
6. Casual Game Genres on Big Fish Games [Електронний ресурс] // Big Fish Games. – 1 March 2002. — Режим доступу: *http://www.bigfishgames.com/download-games/top-pc-games.html.*
7. Casual Gaming Worth $2.25 Billion, and Growing Fast [Електронний ресурс] // VentureBeat. 29 October 2007. — Режим доступу: *https://venturebeat.com/business/casual-gaming-worth-225-billion-and-growing-fast/.*

8. Gamezebo staff. Casual Game Genres on Gamezebo [Електронний ресурс]. – 1 March 2006. — Режим доступу: *http://www.gamezebo.com/games/find*.
9. Hyper-casual: Mobile gaming's newest genre [Електронний ресурс] // AppLovin Blog. 3 August 2018. — Режим доступу: *https://blog.applovin.com/hyper-casual-mobile-gamings-newest-genre/*.
10. IR Information : Financial Data – Top Selling Title Sales Units – Wii Software [Електронний ресурс] // Nintendo Co., Ltd., 2019. — Режим доступу: *http://www.nintendo.co.jp/ir/en/finance/software/wiiu.html*.
11. Kent, Steve L. The ultimate history of video games: from Pong to Pokémon and beyond : the story behind the craze that touched our lives and changed the world // Prima, 2001. – p. 143. [K]
12. Kohler, Chris. Q&A: 14. Happy Farm [Електронний ресурс] // Wired. – 2008. — Режим доступу: *https://www.wired.com/gamelife/2009/12/the-15-most-influential-games-of-the-decade/2/*.
13. Kohler, Chris. Q&A: Pac-Man Creator Reflects on 30 Years of Dot-Eating [Електронний ресурс] // Wired. – 21 May 2010. — Режим доступу: *https://www.wired.com/gamelife/2010/05/pac-man-30-years/*.
14. Kharif, Olga. Tetris' Maker Has His "A" Game [Електронний ресурс] // Bloomberg. – 23 Nov 2005. — Режим доступу: *https://www.bloomberg.com/news/articles/2005-11-22/tetris-maker-has-his-a-game*.
15. Welcome To Gaming Lite // Computer Gaming World. – September, 1992. – p. 74.
16. Card game [Електронний ресурс] // Wikipedia. – Режим доступу: *https://en.wikipedia.org/wiki/Card_game*.
17. List of patience games [Електронний ресурс] // Wikipedia. – Режим доступу: *https://en.wikipedia.org/wiki/List_of_patience_games*.