МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з «Основ програмування – 2. Методології програмування»
(назва дисципліни)

на тему: Карткова гра 101

Студент 1 курсу, групи ІП-22
Христосенко Андрій Степанович
Спеціальності 121 «Інженерія програмного забезпечення»

Керівник
ст. вик. Головченко М. М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____
Національна оцінка _____

Члени комісії

_____ к. т. н., доц. Муха І. П.
(підпис) (посада, вчене звання, науковий ступінь, прізвище та ініціали)

_____ асистент Вовк. Є. А.
(підпис) (посада, вчене звання, науковий ступінь, прізвище та ініціали)

Київ 2023 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-22 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Христосенка Андрія Степановича

(прізвище, ім'я, по батькові)

1. Тема роботи Карткова гра 101

2. Строк здачі студентом закінченої роботи 28 червня 2023

3. Вихідні дані до роботи Додаток «А» Технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)
Постановка задачі, теоретичні відомості, опис алгоритмів, опис програмного забезпечення тестування програмного забезпечення, інструкція користувача.

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

6. Дата видачі завдання «12» лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	12.02.2023	
2.	Підготовка ТЗ	17.03.2023	
3.	Пошук та вивчення літератури з питань курсової роботи	17.04.2023	
4.	Розробка сценарію роботи програми	25.04.2023	
5.	Узгодження сценарію роботи програми з керівником	26.04.2023	
6.	Розробка (вибір) алгоритму рішення задачі	05.04.2023	
6.	Узгодження алгоритму з керівником	08.04.2023	
7.	Узгодження з керівником інтерфейсу користувача	08.04.2023	
8.	Розробка програмного забезпечення	18.04.2023	
9.	Налагодження розрахункової частини програми	25.04.2023	
10.	Розробка та налагодження інтерфейсної частини програми	12.05.2023	
11.	Узгодження з керівником набору тестів для контрольного прикладу	13.05.2023	
12.	Тестування програми	30.05.2023	
13.	Підготовка пояснювальної записки	03.06.2023	
14.	Здача курсової роботи на перевірку	27.06.2023	
15.	Захист курсової роботи	28.06.2023	

Студент _____

(підпис)

Керівник _____ Головченко М. М.

(підпис) (прізвище, ім'я, по батькові)

"12" лютого 2023р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 79 сторінка, 8 рисунків, 17 таблиць, 2 посилання.

Мета роботи: якісна реалізація карткової гри 101.

Виконана програмна реалізація алгоритму карткової гри 101.

ЗМІСТ

ВСТУП

На сучасному етапі розвитку ігрової індустрії, дивовижні можливості з'являються перед розробниками, відкриваючи нові горизонти для творчості. Карткові ігри також стали одними із найпопулярніших ігор, у які продовжують грати мільйони людей у світі.

Гра "Карткова гра 101" є старою радянською грою і була популярною серед колишніх держав СНД. Особливо важливим є той факт, що розробка такої програми надає розробнику великий досвід. Процес розробки гри дозволяє навчитися працювати з об'єктами у графічному інтерфейсі, розробляти алгоритми та поєднувати ці компоненти між собою.

У рамках цього проекту будуть описані алгоритми, які використовуються у грі "Карткова гра 101", а також наведена реалізація цієї головоломки на мові програмування Python з усією необхідною документацією.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення для реалізації карткової гри "101".

Задача гри полягає у тому, щоб набрати якомога менше очок шляхом гри і розігрування карт. Кожна карта має свою вартість, і гравець набирає очки в залежності від вартості розіграної карти. Гра триває до досягнення більше 101 очок.

Вхідними даними для даної роботи є імена гравців у текстовому файлі, порожнє ігрове поле.

Вихідними даними для програми є набрані очки гравцем та комп’ютером у текстовому файлі.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

У картковій грі 101 використовуються 1 колода, 36 ігрових карт та 4 гравці.

Для реалізації гри були написані власні алгоритми: загальний запуск гри, запуск раундів, перевірка правил гри, алгоритм гри комп'тером, обрахунок кількості очок та алгоритм оновлення дизайну.

Задачею гри є викинути всі карти і тим самим закінчити гру з найменшою кількістю очок. Це досягається тим, що потрібно викидати карти з найбільшою кількістю очок, а також використовуючи різні стратегії.

Старшинство карт у грі: 6, 7, 8, 9, 10, В, Д, К, Т. Вартість карт у очках: туз – 11 очок, 10 – 10 очок, 8 – 8 очок, 7 – 7 очок, 6 – 6 очок, король – 4 очки, дама – 3 очки, валет – 2 очки, 9 – 0 очок. Але є окремий випадок: якщо гравець закінчив гру на будь-яку даму, то у нього забирається 20 очок, якщо гравець закінчив гру на пікову даму, то у нього забирається 40 очок.

Щоб перемагати у грі потрібно викидати карти з найбільшою кількістю очок, або якщо у вас є 9-ка й інша карта такої самої масті то можна спочатку викинути одну або більше дев'яток, а потім покласти зверху будь яку іншу карту - тим самим ви за один хід викинете більше карт.

Також можна приберегти даму, якщо в руках є інша карта якою можна зіграти, а даму залишити на момент, якщо не буде інших карт, щоб зіграти.

Тривалість раунда залежить від того, який гравець першим покладе останню карту з руки на кон. Тривалість усієї гри залежить від того, коли щонайменше один гравець набере більше ніж 101 очко. Але, якщо будь-який гравець набере 101 очко - його рахунок обнуляється.

3 ОПИС АЛГОРИТМІВ

Перелік всіх основних змінних та їх призначення наведено в таблиці 3.1.

Таблиця 3.1 — Основні змінні та їх призначення

Змінна	Призначення
choosed_card	Обрана карта через інтерфейс користувача
modes	Обраний режим інтерфейсу
num_players	Задана кількість гравців
players	Масив гравців
trick_pile	Масив колоди карт
current_player	Номер гравця який має ходити
round_number	Номер раунду
current_suit	Обрана масть після гри дамою
ai_choosed	Номер для масті, яку обирає комп’ютер
is_end_game	Перевіряє чи закінчувати раунд
names	Список з іменами гравців

3.1. Загальний алгоритм

1. ПОЧАТОК
2. ЯКЩО натиснуто кнопку «Почати гру», ТО алгоритм запуску гри
3. ЯКЩО натиснуто кнопку «FAQ», ТО вивести інформацію про гру та правила гри
4. ЯКЩО натиснуто кнопку «Вийти», ТО закрити програму
5. ЯКЩО натиснуто кнопку «Режим розробника», ТО алгоритм оновлення дизайну для розробника
6. ЯКЩО натиснуто кнопку «Вибрати або взяти карту», ТО відкрити вікно з вибором карти
7. ЯКЩО натиснуто кнопку «Обрати масть», ТО відкрити вікно з вибором масті
8. ЯКЩО натиснуто кнопку «Завершити гру», ТО відкрити вікно із завершенням гри

9. КІНЕЦЬ

3.2. Алгоритм початку гри

1. ПОЧАТОК
2. Перетасувати колоду та видати карти гравцям, запустити алгоритм початку гри
3. ПОВТОРИТИ
4. ПРИСВОЇТИ game_scores нульові значення
5. ПРИСВОЇТИ game_over = False
6. ПОКИ НЕ game_over
    - 3.1. ПРИСВОЇТИ last_scores нульові значення
    - 3.2. Запустити алгоритм початку раунду
    - 3.3. ПРИСВОЇТИ last_scores результат роботи алгоритму обрахунку очок.
    - 3.4 Запустити алгоритм зкинути налаштування
    - 3.5 ПРИСВОЇТИ game_scores = game_scores + last_scores
    - 3.6 ПРИСВОЇТИ game_scores результат роботи алгоритму перевірки очок.
    - 3.7 ПРИСВОЇТИ game_over результат роботи алгоритму перевірки завершення гри.
    - 3.8 Оновити користувацький інтерфейс
7. ВСЕ ПОВТОРИТИ
8. ПРИСВОЇТИ final_winners, final_scores результат роботи алгоритму знаходження переможця.
9. Зберегти результати гри
10. Вимкнути всі кнопки окрім як кнопки завершення гри
11. КІНЕЦЬ

3.3. Алгоритм початку раунду

1. ПОЧАТОК

2. ПРИСВОЇТИ current_player результат роботи алгоритму перевірки правил гри
3. Оновити користувацький інтерфейс ігрового поля
4. ПРИЗНАЧИТИ check_round = True
5. ПОКИ check_round
    5.1. ЯКЩО перевірка завершення гри ТО вийти з циклу
    5.2. ЯКЩО перевірка на завершення гри ТО вийти з циклу
    5.3. ЯКЩО гравець не користувач ТО
        5.3.1. ЯКЩО комп’ютер зіграв ТО перевірка на зіграну карту ІНАКШЕ хід наступного гравця
    5.4. ІНАКШЕ
        5.6.1. ПРИСВОЇТИ last_card = self.trick_pile[-1]
        5.6.2. ЯКЩО last_card[0] == 'Q' ТО вивести в заголовок програми, що масть змінена
        5.6.3. ПРИСВОЇТИ choosing_action = True
        5.6.4. ПОКИ choosing_action
            5.6.4.1. ЧЕКАТИ поки гравець не обере карту
            5.6.4.2. ЯКЩО action == "1" ТО
                5.6.4.2.1. ЯКЩО результат алгоритму зіграння картою ТО перевірка на зіграну карту
            5.6.4.3. ЯКЩО action == "2" ТО запустити алгоритм взяття карти з колоди ІНАКШЕ запустити алгоритм відіграння користувачем
        5.6.6. ВСЕ ПОВТОРИТИ
    5.5. ЯКЩО перевірка завершення гри ТО вийти з циклу
    5.6. ЯКЩО перевірка на завершення гри ТО вийти з циклу
    5.7. Оновити користувацький інтерфейс ігрового поля
6. ВСЕ ПОВТОРИТИ
7. КІНЕЦЬ

3.4. Алгоритм відігратися гравцем

1. ПОЧАТОК
2. ПРИСВОЇТИ choosing_action2 = True
3. ПОКИ choosing_action2

    3.1. ПРИСВОЇТИ action результат вибору карти користувачем

    3.2. ЯКЩО action є "1" ТО

        3.2.1. ЯКЩО результат алгоритму зіграння картою ТО перевірка на зіграну карту, ПРИСВОЇТИ choosing_action2 = False

    3.3. ЯКЩО action є "2" ТО пропустити хід, ПРИСВОЇТИ choosing_action2 = False

4. ВСЕ ПОВТОРИТИ
5. КІНЕЦЬ

3.5. Алгоритм зіграння картою гравцем

1. ПОЧАТОК
2. ПРИСВОЇТИ card_choosing = True
3. ПОКИ card_choosing

    3.1. ПРИСВОЇТИ card_number вибрану карту користувачем

    3.2. ПРИСВОЇТИ card_number = int(card_number) – 1

    3.3. ЯКЩО card_number == -1 ТО

        3.3.1. ПРИСВОЇТИ self.choose_card.choosed_card_2 = 0

            3.3.2. ЯКЩО self.choose_card.is_take_card == 1 ТО ПОВЕРНУТИ False

    3.4. ІНАКШЕ

        3.4.1. ЯКЩО перевірка на коректність введеної карти гравцем ТО

            3.4.1.1. ПРИСВОЇТИ card карту користувача

            3.4.1.2. ЯКЩО перевірка на гру картою ТО зіграти картою, повернути True ІНАКШЕ ПОВЕРНУТИ False

3.4.2. ІНАКШЕ ПОВЕРНУТИ False

3.4.3. Оновити користувацький інтерфейс

3.5. ПРИСВОЇТИ self.choose_card.is_take_card = 3

3.6. ПРИСВОЇТИ self.choose_card.choosed_card_2 = 0

4. ВСЕ ПОВТОРИТИ

4. КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Діаграма класів програмного забезпечення

Клас UI відповідає за ініціалізацію робочого вікна та обробку подій у програмі. Клас також містить програмну реалізацію алгоритму гри, запуску раундів та обробки ходів.

Клас ChooseSuit виконує роботу коли натискається кнопка Обрати масть і служить для того, щоб користувач обрав масть, коли користувач поклав даму або дама була першою картою після запуску гри і користувач є роздатчиком карт.

Клас ChooseCard запускає вікно вибору карти або взяття карти з колоди або пропуску ходу. Вікно відкривається коли користувачу дається можливість обрати карту для гри при натисканні на кнопку “Вибрати або взяти карту”.

Клас Player служить для зберігання гравців та їхніх карт у руках. Атрибути та методи класа дозволяють обробляти дії з картами гравців.

Клас Deck містить атрибути та методи, які дозволяють користувачу взаємодіяти з колодою карт, генерує колоду карт, перемішує колоду, видає карту з колоди, видає інформацію про кількість карт у колоді та містить інформацію про кожну карту.

Клас Gameisover запускається після натискання на кнопки “Завершити гру”, яка стає доступною, коли гра закінчується, і служить діалоговим вікном з виведенням інформації, що гру завершено і результати збережені.

На рисунку 4.1 відображено діаграму класів програмного забезпечення з усіма класами та зв’язками між ними.

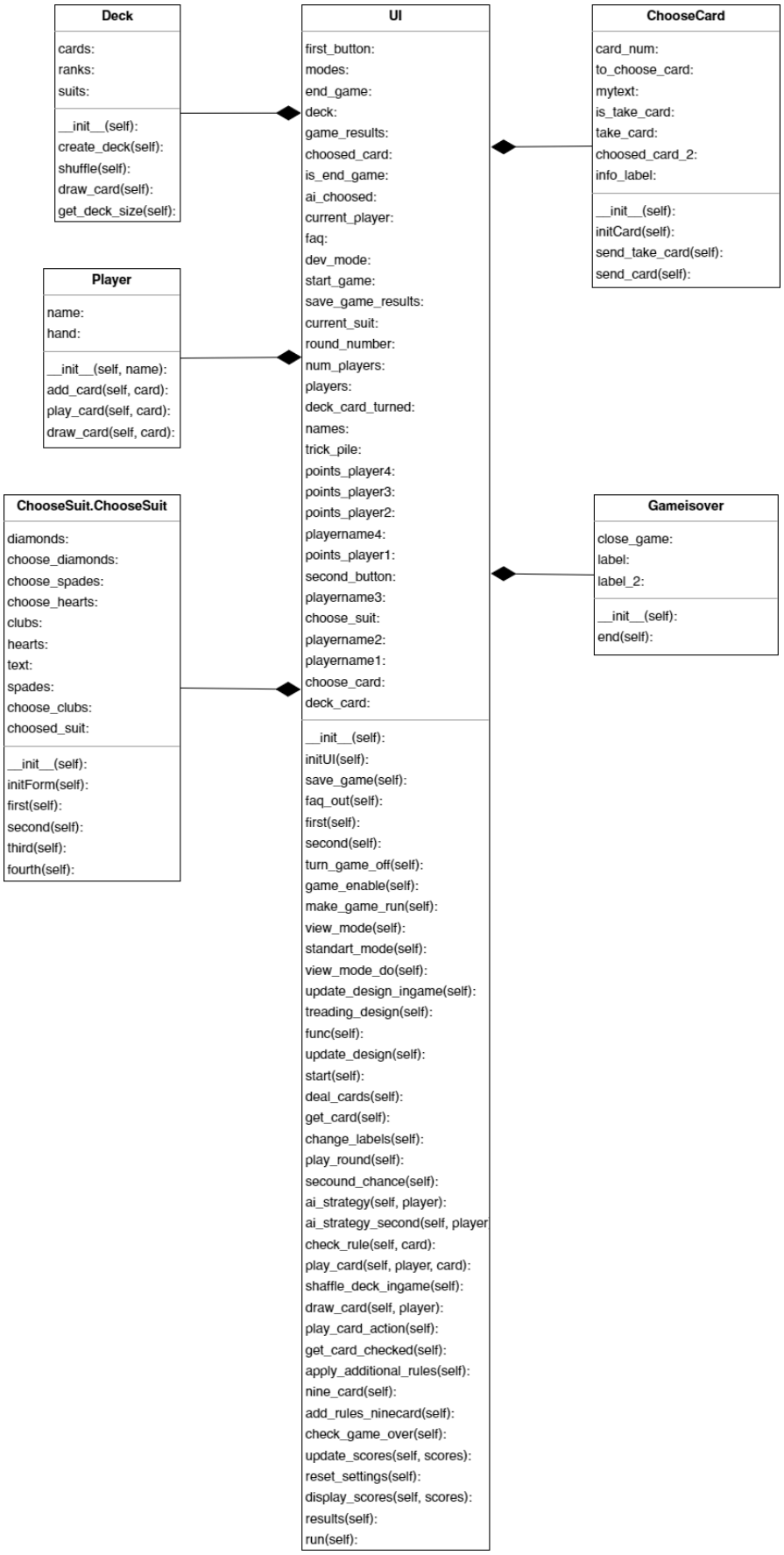

Рисунок 4.1 — Діаграма класів програмного забезпечення

4.2. Опис методів частин програмного забезпечення

4.2.1. Стандартні методи

У таблиці 4.1 наведено стандартні методи, що були використанні при розробці програмного забезпечення. Перелічено використані методи зі стандартної бібліотеки мови програмування Python та набору кросплатформенних модулів PyQt5.

Таблиця 4.1 — Стандартні методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	uic	loadUi	Ініціалізація дизайну програми	Рядок	-
2	QWidget	setWindowTitle	Ініціалізація заголовок вікна	Рядок	-
3	QWidget	setWindowIcon	Ініфіалізація іконки програми	Рядок	-
4	QObject	findChild	Ініціалізація об’єкта дизайну в програмний код	Назва класу об’єкта, назва об’єкта класу	-
5	QWidget	setEnabled	Змінити режим кнопки	Булеве значення	-
6	QLabel	setText	Встановити текст для об’єкта	Рядок	-

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
7	QPushButton	clicked	Дія при натисканні кнопки	Функція	-
8	QWidget	show	Відкриття діалогового вікна програми	-	-
9	QMessageBox	information	Вивід інформаційного вікна	-	-
10	QWidget	close	Закрити програму	-	-
11	threading	Thread	Викликати функцію потоково	Функція, булеве значення	-
12	threading	start	Запуск потоку	-	-
13	threading	join	Дочекатися завершення потоку	-	-

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
14	time	sleep	Призупиняє виконання програми на певний період часу	Ціле число	-
15	QPixmap	setPixmap	Встановлення картинки об’єкту	Об’єкт QPixmap	-
16	-	randint	Генерація числа у заданому діапазоні	Ціле число, ціле число	Ціле число
17	-	choice	Вибір рандомного об’єкта із заданих даних	Список	Список
18	-	shuffle	Перемішати елементи списку	Список	Список
19	-	exit	Вихід з програми	Рядок	-
20	-	readlines	Зчитати рядок з файлу	-	Рядок

4.2.2. Користувацькі методи

У таблиці 4.2 наведено перелік користувацьких методів, що були розроблені для функціонування програмного забезпечення та виконання поставленої задачі.

Таблиця 4.2 — Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	UI	__init__	Ініціалізування графічного інтерфейсу та атрибутів для гри	-	-
2	UI	add_rules_nin ecard	Обробка подій	-	-
3	UI	ai_strategy	Обробка гри за комп’ютер	Ціле число	Булеве значення
4	UI	ai_strategy_se cond	Обробка гри за комп’ютер №2	Ціле число	Булеве значення
5	UI	apply_additio nal_rules	Перевірка правил гри	-	Ціле число
6	UI	change_labels	Оновити дизайн користувача	-	-
7	UI	check_game_ over	Перевірка на завершення гри	-	Булеве значення
8	UI	check_rule	Перевірка чи можна грати картою	Список	Булеве значення
9	UI	deal_cards	Роздача карт	-	-

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
10	UI	display_scores	Оновити дизайн очок гравців	Список	-
11	UI	draw_card	Видати карту гравцю	-	Список
12	UI	faq_out	Вивести вікно інформації про правила гри	-	-
13	UI	first	Дія при натисканні кнопки “Вибрати або взяти карту”	-	-
14	UI	func	Функція оновлення дизайну	-	-
15	UI	game_enable	Запускає функцію початку гри при натисканні кнопки “Почати гру”	-	-
16	UI	get_card	Бере з користувацького інтерфейсу карту, яку обрав користувач	-	Рядок
17	UI	get_card_checked	Бере карту з колоди	-	-
18	UI	initUI	Ініціалізує об’єкти інтерфейсу	-	-

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
19	UI	make_game_run	Запускає алгоритми гри	-	-
20	UI	nine_card	Перевірка правила про карту дев’ятка	-	-
21	UI	play_card	Зіграти картою	Ціле число, список	-
22	UI	play_card_action	Алгоритм зіграння картою користувачем	-	Булеве значення
23	UI	play_round	Алгоритм запуску раунда	-	-
24	UI	reset_settings	Функція зкидання налаштувань	-	-
25	UI	run	Алгоритм запуску гри	-	-
26	UI	save_game	Функція відкриття вікна, яке доступне після закінчення гри	-	-
27	UI	second	Відкриття вікна з вибором масті	-	-
28	UI	secound_chance	Шанс відігратися гравцю	-	-
29	UI	shaffle_deck_ingame	Перемішати колоду під час раунду	-	-

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
30	UI	standart_mode	Функція для запуску дизайну в звичайному режимі гри	-	-
31	UI	start	Функція ініціалізування початку гри	-	-
32	UI	treading_design	Запуск дизайну в потоковому режимі	-	-
33	UI	turn_game_off	Функція закриття програми після натискання кнопки “Вийти”	-	-
34	UI	update_design	Оновити користувацький дизайн гри	-	-
35	UI	update_design_ingame	Оновити користувацький дизайн гри під час раунду	-	-
36	UI	update_scores	Обрахунок кількості очок гравців	Список	Список
37	UI	view_mode	Функція, яка вмикає режим розробника в потоковому режимі	-	-

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
38	UI	view_mode_do	Оновлення користувацького інтерфейсу	-	-
39	Choose Suit	__init__	Ініціалізування дизайну	-	-
40	Choose Suit	initForm	Ініціалізування об’єктів класу	-	-
41	Choose Suit	first	Функція при натисканні на першу кнопку	-	-
42	Choose Suit	second	Функція при натисканні на другу кнопку	-	-
43	Choose Suit	third	Функція при натисканні на третю кнопку	-	-
44	Choose Suit	fourth	Функція при натисканні на четверту кнопку	-	-
45	Choose Card	initCard	Ініціалізування дизайну	-	-
46	Choose Card	send_take_card	Взяття карти або пропуску ходу	-	-
47	Choose Card	send_card	Відправити обрану карту	-	-

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
48	Deck	__init__	Ініціалізування колоди карт	-	-
49	Deck	create_deck	Створення колоди карт	-	-
50	Deck	draw_card	Видати карту гравцю	-	Список
51	Deck	get_deck_size	Отримати розмір колоди	-	Ціле число
52	Deck	shuffle	Перемішати колоду	-	-
53	Gameis over	__init__	Ініціалізування дизайну	-	-
54	Choose Suit	end	Вихід з програми	-	-
55	Player	__init__	Ініціалізування класу Player	-	-
56	Player	add_card	Видати карту в руки гравцю	Список	-
57	Player	draw_card	Взяти карту з руки гравця	Список	-
58	Player	play_card	Зіграти картою	Список	-

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. План тестування

а) Тестування процесу гри

1) Тестування роботи програми після запуску гри.
2) Тестування роботи програми при грі комп’тером
3) Тестування роботи програми при достроковому виході з гри

б) Тестування взяття карти

1) Тестування роботи програми при взятті карти користувачем.
2) Тестування роботи програми при обранні карти користувачем.

в) Тестування вибору масті

г) Тестування при аналізуванні вхідних даних

д) Тестування переходу в режим розробника

1) Тестування роботи програми при вході в режим розробника.
2) Тестування роботи програми після виходу з режиму розробника.

е) Тестування програми при збереженні результатів програми

ж) Тестування завершення гри

1) Тестування роботи програми коли набирається 101 очко
2) Тестування роботи програми завершенням через кнопку “Завершити гру”

5.2. Приклади тестування

У даному підрозділі міститься опис тестування основних функцій програми, включаючи ситуації, що можуть призвести до некоректної роботи програми та продемонстровано очікувані результати роботи у кожному з випадків.

У таблицях 5.1 – 5.12 наведено приклади тестування програми.

Таблиця 5.1 – Приклад роботи програми після запуску гри

Мета тесту	Перевірити правильність генерації поля, карт гравців, правильної здачі карт
Початковий стан програми	Відкрите вікно програми, доступні кнопки “Запуск гри”, “FAQ”, “Вийти”
Вхідні дані	Користувач натиснув кнопку запуск гри
Схема проведення тесту	Обрати кнопку запуск гри та дочекатися поки на полі з’являться ігрові карти, колода та кон
Очікуваний результат	На полі з’явилися карти у гравців та запустилася гра
Стан програми після проведення випробувань	Вікно зі згенерованим полем для гри та картами у гравців

Таблиця 5.2 – Приклад роботи програми при грі комп’тером

Мета тесту	Перевірити як комп’ютер працює при грі картою
Початковий стан програми	Відкрите вікно програми, виведено поле карти у гравців

Продовження таблиці 5.2

Вхідні дані	Кожен гравець має по 5 карт, а роздатчик 4, на коні лежить карта взята з колоди карт
Схема проведення тесту	Дочекатись доки алгоритм не перевірить чи не потрібно ходити гравцю або обрати ним масть
Очікуваний результат	Гравцю потрібно обрати карту, якщо на столі лежить 9, або обрати масть, якщо на коні лежить дама. Інакше хід переходить до комп’ютера.
Стан програми після проведення випробувань	Хід переходить до комп’ютера, комп’ютер продовжує класти карту на кон і хід переходить далі.

Таблиця 5.3 – Приклад роботи програми при достроковому виході з гри

Мета тесту	Перевірити як програма завершує роботу при достроковому виході
Початковий стан програми	Відкрите вікно програми, виведено поле та фігури поряд, хід комп’ютера
Вхідні дані	Користувач натискає кнопку “Вийти”
Схема проведення тесту	Під час роботи алгоритмів користувач виходить із гри, натискаючи на кнопку “Вийти”
Очікуваний результат	Вікно закривається з кодом 0
Стан програми після проведення випробувань	Вікно закривається з кодом 0

Таблиця 5.4 – Приклад роботи програми при взятті карти користувачем.

Мета тесту	Перевірити як програма працює коли користувач бере карту з кону
Початковий стан програми	Відкрите вікно програми, виведено поле, хід користувача
Вхідні дані	Користувач натискає на кнопку “Вибрати або взяти карту” і обирає взяти карту
Схема проведення тесту	Коли хід доходить до користувача, йому відкривається можливість обрати карту, або обрати масть, у нашому випадку випадає можливість зіграти картою і користувач обирає взяти карту
Очікуваний результат	На вікні виведеться вікно вибору карти, користувач вибере взяти карту і користувач отримає карту, а потім дасть користувачу відігратись
Стан програми після проведення випробувань	Користувач отримав карту, гра очікує наступного вибора користувача

Таблиця 5.5 – Приклад роботи програми при при обранні карти

Мета тесту	Перевірити як програма працює при обранні карти користувачем
Початковий стан програми	Відкрите вікно програми, виведено поле, хід користувача
Вхідні дані	Користувач натискає на кнопку “Вибрати або взяти карту” і обирає існуючу карту

Продовження таблиці 5.5

Схема проведення тесту	На вікні виведеться вікно вибору карти, користувач вибере карту і натисне обрати - користувач отримає карту, а потім хід перейде до наступного гравця
Очікуваний результат	Користувач зіграє картою, якщо карта була 9 або дама, то необхідно буде зіграти далі або обрати масть, інакше хід переходить наступному ігроку
Стан програми після проведення випробувань	Гравець обрав карту, хід перейшов до наступного гравця

Таблиця 5.6 – Приклад роботи програми при обранні масті

Мета тесту	Перевірити як програма працює при обранні масті користувачем
Початковий стан програми	Відкрите вікно програми, виведено поле, хід користувача
Вхідні дані	Користувач натискає на кнопку “Обрати масть” і обирає масть
Схема проведення тесту	Натиснути на кнопку “Обрати масть”
Очікуваний результат	Користувач обере масть, хід перейде до наступного гравця і гравець має накрити загаданою мастю
Стан програми після проведення випробувань	Користувач обрав масть, хід перейшов до наступного гравця і гравець зіграв обраною мастю користувача

Таблиця 5.7 – Приклад роботи програми при аналізуванні вхідних даних

Мета тесту	Перевірити як програма працює при аналізуванні вхідних даних
Початковий стан програми	Програма закрита
Вхідні дані	Користувач вводить у файл “players.txt” імена гравців з кожного рядка
Схема проведення тесту	Перед запуском гри користувач вводить 4 імена гравців з кожного рядка
Очікуваний результат	На ігровому полі виводяться задані користувачем імена гравців
Стан програми після проведення випробувань	Задані користувачем відображаються на ігровому полі

Таблиця 5.8 – Приклад роботи програми при вході в режим розробника.

Мета тесту	Перевірити як програма переходить у режим розробника
Початковий стан програми	Відкрите вікно програми, згенеровано ігрове поле, хід гравця
Вхідні дані	Користувач обрає «Режим розробника»
Схема проведення тесту	Натиснути на кнопку «Режим розробника»
Очікуваний результат	Карти всіх гравців стають відкритими для перегляду
Стан програми після проведення випробувань	Карти гравців відкриті

Таблиця 5.9 – Приклад роботи програми після виходу з режиму розробника.

Мета тесту	Перевірити як програма виходить із режима розробника та переходить у стандартний режим
Початковий стан програми	Відкрите вікно програми, згенеровано ігрове поле, всі карти гравців видимі
Вхідні дані	Користувач обраєє «Стандартний режим»
Схема проведення тесту	Натиснути на кнопку «Стандартний режим»
Очікуваний результат	Карти всіх гравців окрім користувача стають закритими для перегляду
Стан програми після проведення випробувань	Карти всіх гравців окрім користувача закриті

Таблиця 5.10 – Приклад роботи програми при збереженні результатів програми

Мета тесту	Перевірити як програма працює при збереженні результатів програми
Початковий стан програми	Гра зіграна, один з гравців набрав більше 101 очка
Вхідні дані	Обраховані результати гравців
Схема проведення тесту	Гра завершується після виконання умови, що щонайменше у одного гравця більше 101 очок
Очікуваний результат	Гру завершено, доступна лиш одна кнопка “Завершити гру”, результати збережені у файл “results.txt”

Продовження таблиці 5.10

Стан програми після проведення випробувань	Результати успішно збереглися у файлі “results.txt”

Таблиця 5.11 – Приклад роботи програми коли набирається 101 очко

Мета тесту	Перевірити як програма працює якщо один з гравців набирає 101 очко
Початковий стан програми	Раунд завершено, один з гравців набрав 101 очко
Вхідні дані	Один з гравців набрав рівно 101 очко
Схема проведення тесту	Набрати рівно 101 очко
Очікуваний результат	Онулювання очок гравця
Стан програми після проведення випробувань	Очки гравця онулювались і він має тепер 0 очок

Таблиця 5.12 – Приклад роботи програми при завершенні через кнопку “Завершити гру”

Мета тесту	Перевірити як програма працює при завершенні програми через кнопку “Завершити гру”
Початковий стан програми	Відкрите вікно програми, доступна лише одна кнопка “Завершити гру”
Вхідні дані	Користувач натискає на кнопку “Завершити гру”
Схема проведення тесту	Натиснути на кнопку “Завершити гру” і після цього натискає на кнопку виходу з програми
Очікуваний результат	Програма успішно завершується
Стан програми після проведення випробувань	Програма успішно завершилася

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1. Робота з програмою

Перед запуском програми інсталюється модуль PyQt5 через команду “pip install PyQt5”. Після інсталювання пакету запускаємо файл main з розширенням .py відкривається головне вікно програми (рис. 6.1).

Рисунок 6.1 – Головне вікно програми

Після цього користувач має почати гру, натиснувши на кнопку Почати гру та дочекатись поки згенерується колода та ініціалізується дизайн (рис. 6.2).

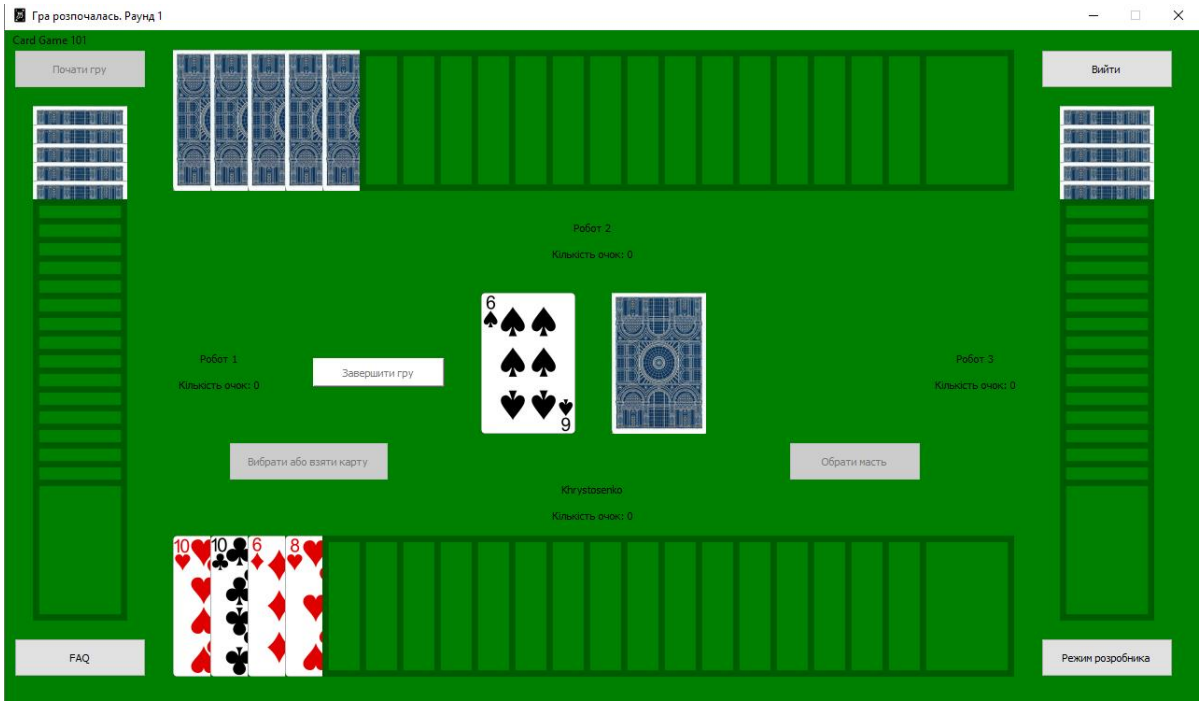

Рисунок 6.2 – Натиснуто кнопку «Почати гру» та ініціалізовано ігрове поле

Після цього в гру необхідно грати згідно правил гри, які можна подивитись, обравши кнопку “FAQ” (рис. 6.3).

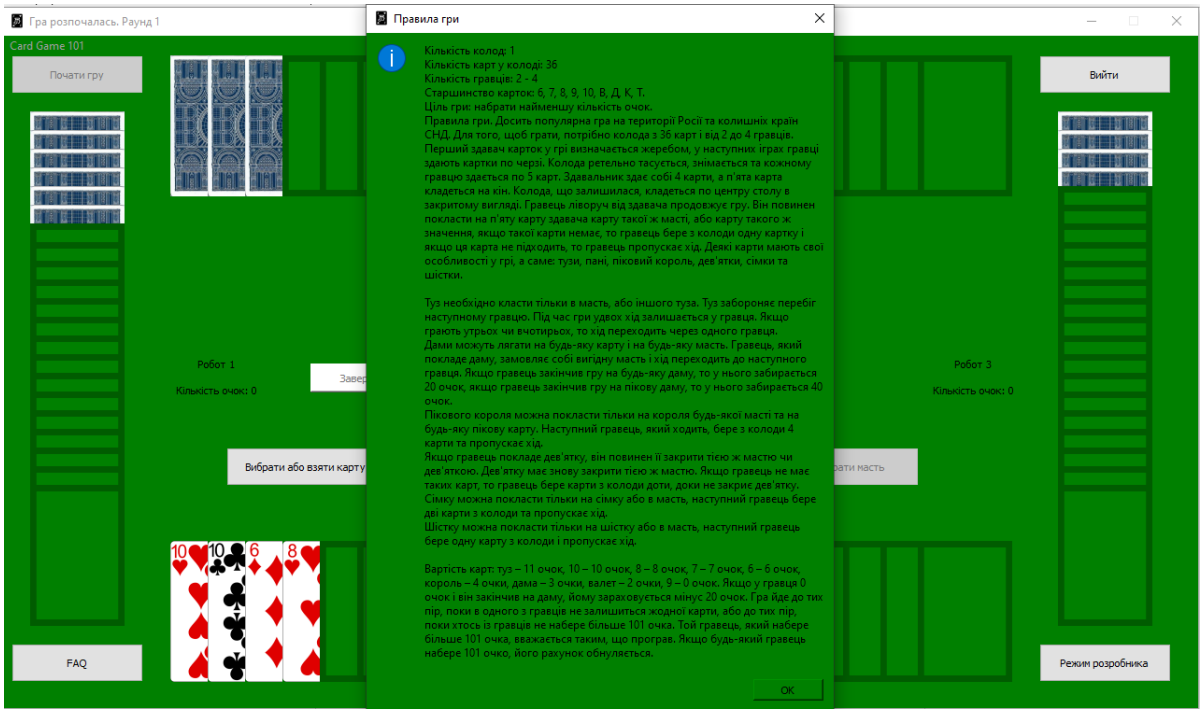

Рисунок 6.3 – Відкрито меню правил гри

В ході гри користувач може обирати між варіантами як взяття карти або зіграти картою, якщо гравець хоче зіграти картою, тоді він обирає номер карти та натискає на кнопку “обрати” (рис. 6.4).

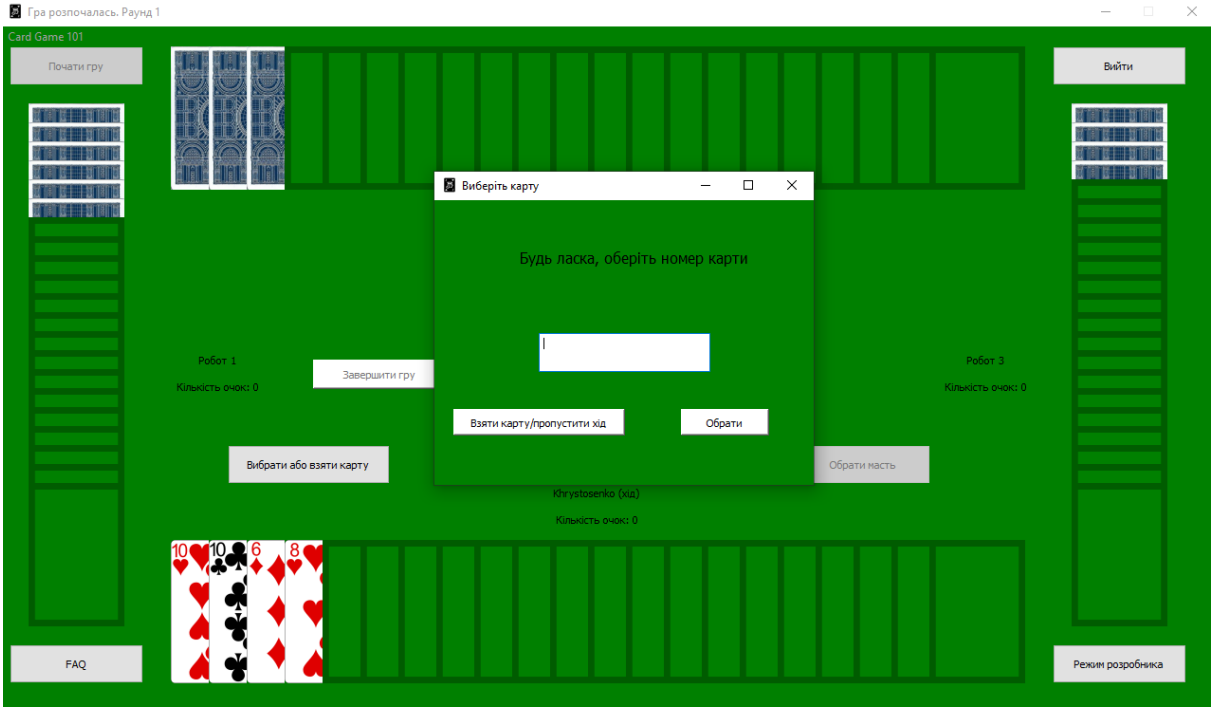

Рисунок 6.4 – Обрання дії над картами

Якщо користувач викинув даму, тоді користувач повинен обрати масть натиснувши на відповідну активну кнопку “Обрати масть” (рис. 6.5).

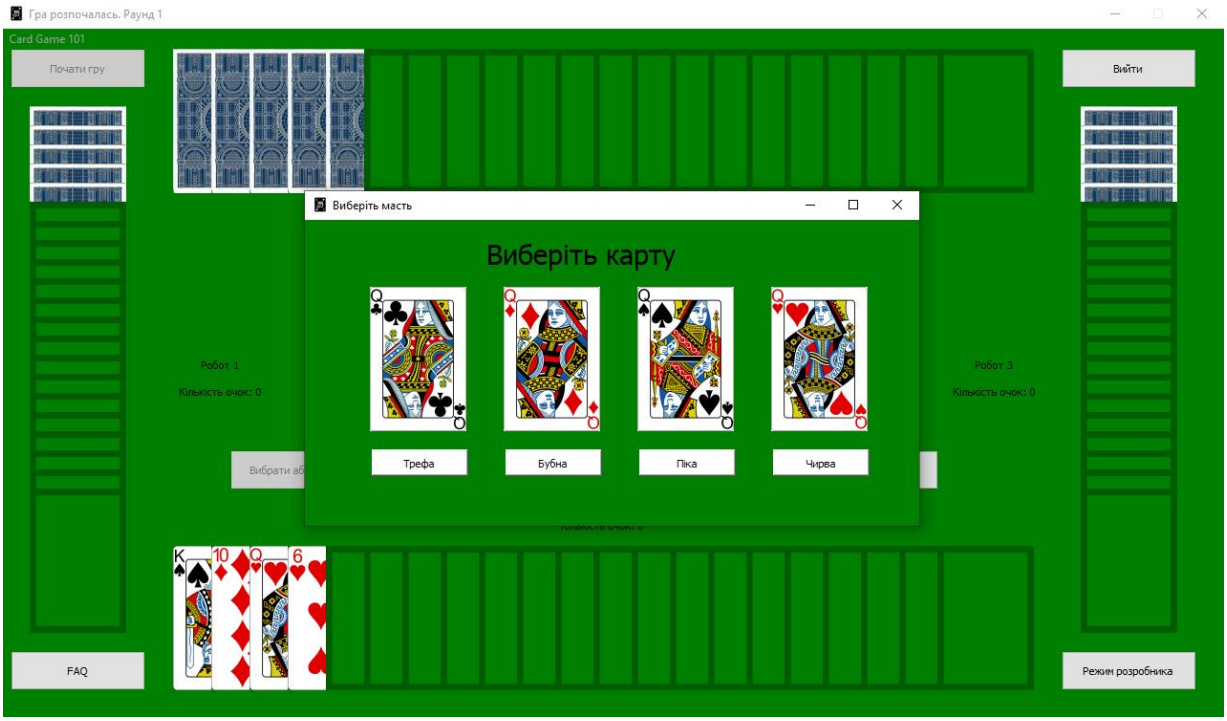

Рисунок 6.5 – Вікно з обранням масті

Якщо ви захотіли перейти в режим розробника, ви можете натиснути на відповідну кнопку “Режим розробника” і побачити карти інших гравців (рисунок 6.6).

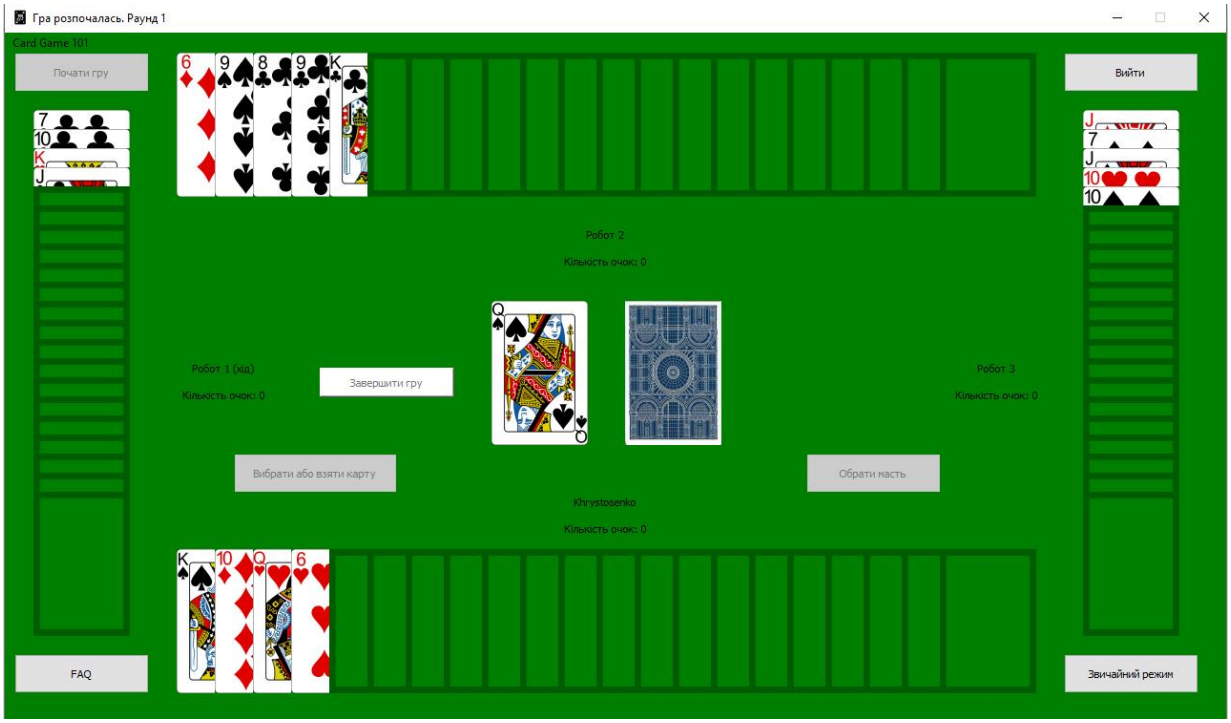

Рисунок 6.6 – Виведено карти інших гравців

Після закінчення раундів, ви маєте можливіть зберегти результати гри. Приклад закінчення раунду (рис. 6.7)

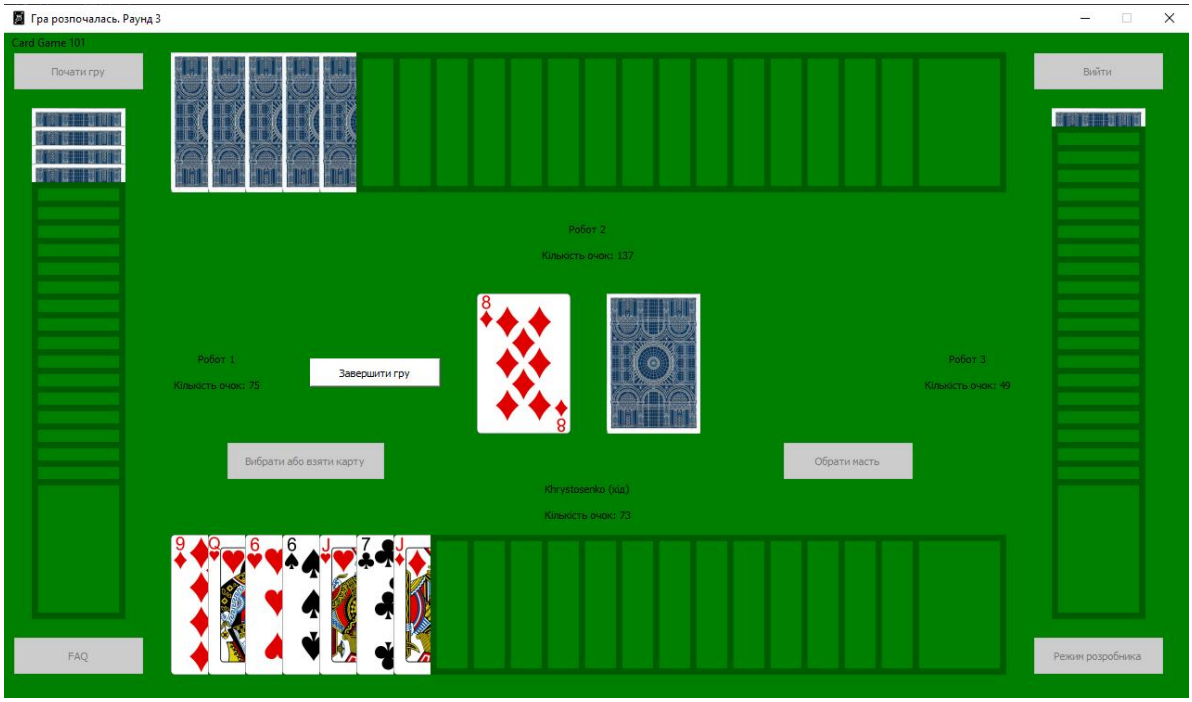

Рисунок 6.7 – Закінчення раунду з активною кнопкою “Завершити гру”

Для виходу з програми користувач може або натиснути крестик зправа вгорі або натиснути кнопку “Завершити гру” і гра завершиться успішно.

6.2 Формат вхідних та вихідних даних

Перед початком гри вхідними даними є імена гравців у файлі “players.txt”. Вихідними даними є результати збережені у файлі “results.txt”.

6.3 Системні вимоги

У таблиці 6.1 наведено системні вимоги до програмного забезпечення.

Таблиця 6.1 – Системні вимоги до програми

	Мінімальні	Рекомендовані
Операційна система	Windows XP	Windows 7, 8, 10, 11
Процесор	Intel Core i3, 1.0 GHz	Intel Core i5, i7, 2.0 GHz
Оперативна пам’ять	2 GB	4 GB і більше
Відеоадаптер	Intel GMA 950 з відеопам’яттю об'ємом не менше 64 МБ (або сумісний аналог)	
Дисплей	800x600	1024x768
Прилади введення	Клавіатура і комп’ютерна мишка	
Додаткове програмне забезпечення	-	

ВИСНОВКИ

У перших двох розділах висвілено теоретичні відомості необхідні для подальшого розуміння ходу гри та підходів до алгоритму розв’язання. У третьому розділі описані основні алгоритми використані у програмі. Четвертий – вміщує діаграму класів та опис використаних методів. У п’ятому розділі розроблено алгоритм тестування та описано бажані результати. Шостий – містить інструкцію користувача з того, що і в якій послідовності треба робити під час користування програмою.

У ході даної курсової роботи було розроблено якісне програмне забезпечення із застосуванням об’єктно-орієнтованої парадигми програмування. Створено програму гри в карткову гру 101. Розроблена програма дозволяє пограти в гру 101 з комп’ютером.

Розроблене програмне забезпечення надає великий багаж знань розробнику та може бути використане в комерційних цілях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Карточная игра 101. cards-igri.ru. Архів оригіналу за 8 червня 2017. Процитовано 4 червня 2017.
2. http://www.durbetsel.ru. Сто одно (101) - карточная игра. www.durbetsel.ru. Архів оригіналу за 23 жовтня 2018. Процитовано 4 червня 2017.

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив

Керівник *Головченко М.М.*

«23» березня 2023 р.

Виконавець:

Студент *Христосенко А.С.*

«31» травня 2023р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: Карткова гра 101

з дисципліни:

«Основи програмування»

Київ 2023

1. *Мета*: Метою курсової роботи є розробка якісного та коректного програмного забезпечення карткової гри 101.
2. *Дата початку роботи*: «12» лютого 2023 р.
3. *Дата закінчення роботи*: «28» червня 2023 р.
4. *Вимоги до програмного забезпечення*.
   1) Функціональні вимоги:
      – Реалізація тривалості гри. Тривалість однієї гри до моменту коли в одного з гравців не залишиться жодної карти.
      – Реалізація анімаціх роздачі карт.
      – Можливість рандомної генерації колоди карт.
      – Можливість грати тільки згідно правил гри.
      – Можливіть обрахунку кількості очок у гравців та запису їх у файл.
      – Вивід поля для гри, ігрових карт гравців, імен користувачів, колоди та кону.
      – Збереження результатів гри у файл.
   2) Нефункціональні вимоги:
      – Можливість запуску програми на операційних системах Windows 10 та Windows 11.
      – Реалізація програми на мові Python з використанням бібліотек PyQt5.
      – Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:
      ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.
      ГОСТ 19.106 - 78 - Вимоги до програмної документації.
      ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.
5. *Стадії та етапи розробки*:

1) Об'єктно-орієнтований аналіз предметної області задачі (до 17.04.2023р.)
2) Об'єктно-орієнтоване проектування архітектури програмної системи (до 24.04.2023р.)
3) Розробка програмного забезпечення (до 18.04.2023р.)
4) Тестування розробленої програми (до 30.05.2023р.)
5) Розробка пояснювальної записки (до 03.06.2023 р.).
6) Захист курсової роботи (до 26.06.2023 р.).

6. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду програмного забезпечення

Карткової гри 101

(Найменування програми (документа))

Електронний носій

(Вид носія даних)

37 арк, 164 Мб

(Обсяг програми (документа), арк.,

студента групи ІП-22 І курсу

Христосенко А.С.

main.py

```
from UI import *
from PyQt5.QtWidgets import QApplication



if __name__ == '__main__':
    app = QApplication(sys.argv)
    ui = UI()
    sys.exit(app.exec_())
```

UI.py

```
from Gameisover import *
from Deck import *
from Player import *
from ChooseCard import *
from ChooseSuit import *
import threading
import time
from functions import *
from PyQt5.QtWidgets import QMainWindow



class UI(QMainWindow):
    def __init__(self):
        super(UI, self).__init__()

        self.deck = Deck()
        self.choose_suit = ChooseSuit()
        self.choose_card = ChooseCard()
        self.game_results = Gameisover()
```

```
self.choosed_card = 0
self.modes = 0
self.num_players = 4
self.players = []
self.trick_pile = []
self.current_player = 0
self.round_number = 1
self.current_suit = None
self.ai_choosed = None
self.is_end_game = None
self.names = []

# Ініціалізація UI
uic.loadUi("designs/window.ui", self)
self.setWindowTitle("101 Card game")
self.setWindowIcon(QIcon(f"{icon}"))

players = ["player1", "player2", "player3", "player4"]
cards = ["card" + str(i) for i in range(1, 22)]

for player in players:
    for card in cards:
        label_name = card + "_" + player
        setattr(self, label_name, self.findChild(QLabel, label_name))

self.deck_card = self.findChild(QLabel, "deck_card")
self.deck_card_turned = self.findChild(QLabel, "deck_card_turned")

self.playername1 = self.findChild(QLabel, "playername1")
self.playername2 = self.findChild(QLabel, "playername2")
```

```
        self.playername3 = self.findChild(QLabel, "playername3")
        self.playername4 = self.findChild(QLabel, "playername4")

        self.points_player1 = self.findChild(QLabel, "points_player1")
        self.points_player2 = self.findChild(QLabel, "points_player2")
        self.points_player3 = self.findChild(QLabel, "points_player3")
        self.points_player4 = self.findChild(QLabel, "points_player4")

        self.faq = self.findChild(QPushButton, "faq")
        self.start_game = self.findChild(QPushButton, "start_game")
        self.end_game = self.findChild(QPushButton, "end_game")
        self.dev_mode = self.findChild(QPushButton, "dev_mode")
        self.first_button = self.findChild(QPushButton, "first_button")
        self.second_button = self.findChild(QPushButton, "second_button")
        self.save_game_results = self.findChild(QPushButton,
"save_game_results")

        self.initUI()
        UI.show(self)

    def initUI(self):
        with open("players.txt", 'r') as file:
            self.names = [line.strip() for line in file.readlines()]

        for i in range(4):
            self.players.append(Player(self.names[i]))

        self.second_button.setEnabled(False)
        self.first_button.setEnabled(False)
        self.save_game_results.setEnabled(False)
```

```
        self.dev_mode.setEnabled(False)

        self.playername1.setText(self.names[0])
        self.playername2.setText(self.names[1])
        self.playername3.setText(self.names[2])
        self.playername4.setText(self.names[3])

        self.save_game_results.clicked.connect(self.save_game)
        self.start_game.clicked.connect(self.game_enable)
        self.faq.clicked.connect(self.faq_out)
        self.end_game.clicked.connect(self.turn_game_off)
        self.dev_mode.clicked.connect(self.view_mode)
        self.first_button.clicked.connect(self.first)
        self.second_button.clicked.connect(self.second)

    def save_game(self):
        self.game_results.show()

    def faq_out(self):
        QMessageBox.information(self, "Правила гри", text)

    def first(self):
        self.choose_card.show()

    def second(self):
        self.choose_suit.show()

    def turn_game_off(self):
        # NEW CODE
        UI.close(self)
```

```
        def game_enable(self):
            self.setWindowTitle(f"Гра розпочалась. Раунд {self.round_number}")
            self.make_game_run()
            game = threading.Thread(target=self.run, name="game_logic",
daemon=True)
            game.start()

        def make_game_run(self):
            self.start()
            self.trick_pile.append(self.deck.draw_card())
            ui_threads = threading.Thread(target=self.treading_design,
name="kurwa")
            ui_threads.start()
            ui_threads.join()
            self.start_game.setEnabled(False)
            time.sleep(1)

        def view_mode(self):
            if self.modes == 0:
                self.modes = 1
                self.dev_mode.setText("Звичайний режим")
                view_mode_0 = threading.Thread(target=self.view_mode_do)
                view_mode_0.start()
            else:
                self.modes = 0
                self.dev_mode.setText("Режим розробника")
                view_mode_1 = threading.Thread(target=self.update_design)
                view_mode_1.start()
```

```
        def standart_mode(self):
            self.modes = 0
            self.dev_mode.setText("Режим розробника")
            defaultmode_thread = threading.Thread(target=self.update_design)
            defaultmode_thread.start()

        def view_mode_do(self):
            if self.modes == 1:
                pass
            else:
                self.standart_mode()
                return

            card = self.trick_pile[-1]
            pixmap = QPixmap(f"images/{card[0]}_of_{card[1]}.png")
            self.deck_card.setPixmap(pixmap)

            empty = QPixmap(f"images/empty.png")
            self.change_labels()
            try:
                deck_card_2 = self.trick_pile[-2]
                another =
QPixmap(f"images/{deck_card_2[0]}_of_{deck_card_2[1]}.png")
                self.deck_card_turned.setPixmap(another)
            except IndexError:
                pass

            if self.deck.get_deck_size() == 0:
                deck_pix = QPixmap(f"images/empty.png")
                self.deck_card_turned.setPixmap(deck_pix)
```

```
else:
    deck_pix2 = QPixmap(f"{turned}")
    self.deck_card_turned.setPixmap(deck_pix2)

player_hand = self.players[0].hand

for i, card in enumerate(player_hand[:21]):
    if len(card) > 1:
        pixmap1 = QPixmap(f"images/{card[0]}_of_{card[1]}.png")
        getattr(self, f"card{i + 1}_player1").setPixmap(pixmap1)

for index in range(len(player_hand), 21):
    getattr(self, f"card{index + 1}_player1").setPixmap(empty)

player_hand = self.players[1].hand

for i, card in enumerate(player_hand[:21]):
    if len(card) > 1:
        pixmap2 = QPixmap(f"images/{card[0]}_of_{card[1]}.png")
        getattr(self, f"card{i + 1}_player2").setPixmap(pixmap2)

for index in range(len(player_hand), 21):
    getattr(self, f"card{index + 1}_player2").setPixmap(empty)

player_hand = self.players[2].hand

for i, card in enumerate(player_hand[:21]):
    if len(card) > 1:
        pixmap3 = QPixmap(f"images/{card[0]}_of_{card[1]}.png")
        getattr(self, f"card{i + 1}_player3").setPixmap(pixmap3)
```

```
            for index in range(len(player_hand), 21):
                getattr(self, f"card{index + 1}_player3").setPixmap(empty)

            player_hand = self.players[3].hand

            for i, card in enumerate(player_hand[:21]):
                if len(card) > 1:
                    pixmap4 = QPixmap(f"images/{card[0]}_of_{card[1]}.png")
                    getattr(self, f"card{i + 1}_player4").setPixmap(pixmap4)

            for index in range(len(player_hand), 21):
                getattr(self, f"card{index + 1}_player4").setPixmap(empty)

        def update_design_ingame(self):
            self.change_labels()
            if self.modes == 0:
                random_number = random.randint(1, 10000)
                thread_name = f"thread_update_{random_number}"
                thread_update = threading.Thread(target=self.update_design,
name=thread_name)
                thread_update.start()
            else:
                random_number = random.randint(1, 10000)
                thread_name = f"thread_update_{random_number}"
                thread_view = threading.Thread(target=self.view_mode_do,
name=thread_name)
                thread_view.start()

        def treading_design(self):
```

```
    self.update_design()

def func(self):
    empty = QPixmap("images/empty.png")

    player_hand = self.players[0].hand

    for i, card in enumerate(player_hand[:21]):
        if len(card) > 1:
            pixmap = QPixmap(f"images/{card[0]}_of_{card[1]}.png")
            getattr(self, f"card{i + 1}_player1").setPixmap(pixmap)

    for index in range(len(player_hand), 21):
        getattr(self, f"card{index + 1}_player1").setPixmap(empty)

    pixmap = QPixmap(f"{turned}")

    player_hand = self.players[1].hand

    for i, card in enumerate(player_hand[:21]):
        if len(card) > 1:
            getattr(self, f"card{i + 1}_player2").setPixmap(pixmap)

    for index in range(len(player_hand), 21):
        getattr(self, f"card{index + 1}_player2").setPixmap(empty)

    player_hand = self.players[2].hand

    for i, card in enumerate(player_hand[:21]):
        if len(card) > 1:
```

```
                getattr(self, f"card{i + 1}_player3").setPixmap(pixmap)

        for index in range(len(player_hand), 21):
            getattr(self, f"card{index + 1}_player3").setPixmap(empty)

        player_hand = self.players[3].hand

        for i, card in enumerate(player_hand[:21]):
            if len(card) > 1:
                getattr(self, f"card{i + 1}_player4").setPixmap(pixmap)

        for index in range(len(player_hand), 21):
            getattr(self, f"card{index + 1}_player4").setPixmap(empty)

    def update_design(self):
        card = self.trick_pile[-1]
        pixmap = QPixmap(f"images/{card[0]}_of_{card[1]}.png")
        self.deck_card.setPixmap(pixmap)

        if self.deck.get_deck_size() == 0:
            pixmap = QPixmap(f"images/empty.png")
            self.deck_card_turned.setPixmap(pixmap)
        else:
            pixmap = QPixmap(f"{turned}")
            self.deck_card_turned.setPixmap(pixmap)

        self.func()

    def start(self):
        self.deck.shuffle()
```

```
        self.deal_cards()

    def deal_cards(self):
        for i in range(self.num_players):
            num_cards = 5 if i != self.current_player else 4

            for _ in range(num_cards):
                card = self.deck.draw_card()
                self.players[i].add_card(card)

    def get_card(self):
        time.sleep(2)
        action = "0"
        if self.choose_card.is_take_card == 1:
            action = "2"
        elif self.choose_card.is_take_card == 0:
            action = "1"
        elif self.choose_card.is_take_card != 0 and self.choose_card.is_take_card
!= 1:
            pass
        self.choose_card.is_take_card = 3
        return action

    def change_labels(self):
        if self.current_player == 0:
            self.playername1.setText(f"{self.names[0]} (хід)")
            self.playername2.setText(self.names[1])
            self.playername3.setText(self.names[2])
            self.playername4.setText(self.names[3])
        elif self.current_player == 1:
```

```
            self.playername1.setText(self.names[0])
            self.playername2.setText(f"{self.names[1]} (хід)")
            self.playername3.setText(self.names[2])
            self.playername4.setText(self.names[3])
        elif self.current_player == 2:
            self.playername1.setText(self.names[0])
            self.playername2.setText(self.names[1])
            self.playername3.setText(f"{self.names[2]} (хід)")
            self.playername4.setText(self.names[3])
        elif self.current_player == 3:
            self.playername1.setText(self.names[0])
            self.playername2.setText(self.names[1])
            self.playername3.setText(self.names[2])
            self.playername4.setText(f"{self.names[3]} (хід)")

    def play_round(self):
        self.setWindowTitle(f"Гра розпочалась. Раунд {self.round_number}")
        time.sleep(1)
        self.current_player = self.apply_additional_rules()
        self.update_design_ingame()
        check_round = True
        while check_round:
            time.sleep(2)
            if self.check_game_over():
                break

            if self.is_end_game:
                break

            if self.current_player != 0:
```

```
                if self.ai_strategy(self.current_player):
                    self.current_player = self.apply_additional_rules()
                    self.change_labels()
                else:
                    self.current_player = (self.current_player + 1) % 4
                    self.change_labels()
            else:
                last_card = self.trick_pile[-1]
                if last_card[0] == 'Q':
                    self.change_labels()
                    self.setWindowTitle(f"Увага! Масть змінено на: {self.deck.suits[self.ai_choosed]}")
                self.first_button.setEnabled(True)
                choosing_action = True
                while choosing_action:
                    time.sleep(2)
                    action = self.get_card()
                    if action == "1":
                        choosing_action = False
                        if self.play_card_action():
                            self.current_player = self.apply_additional_rules()
                            choosing_action = False
                    elif action == "2":
                        if not self.draw_card(self.current_player):
                            check_round = False
                            choosing_action = False
                        else:
                            time.sleep(0.5)
                            self.update_design_ingame()
                            time.sleep(0.5)
```

```
                    self.first_button.setEnabled(False)
                    self.secound_chance()
                    choosing_action = False
            self.setWindowTitle(f"Гра розпочалась. Раунд {self.round_number}")

            if self.check_game_over():
                break

            if self.is_end_game:
                break

            self.update_design_ingame()
            time.sleep(2)

        def secound_chance(self):
            time.sleep(1)
            self.first_button.setEnabled(True)
            choosing_action2 = True
            while choosing_action2:
                action = self.get_card()
                if action == "1":
                    if self.play_card_action():
                        self.current_player = self.apply_additional_rules()
                        self.first_button.setEnabled(False)
                        choosing_action2 = False
                elif action == "2":
                    self.current_player = (self.current_player + 1) % self.num_players
                    self.first_button.setEnabled(False)
                    choosing_action2 = False
```

```
def ai_strategy(self, player):
    valid_cards = [card for card in self.players[player].hand if
                   card[1] == self.trick_pile[-1][1] or
                   card[0] == self.trick_pile[-1][0] or
                   card[1] == self.current_suit or
                   card[0] == 'Q']

    if valid_cards:
        card = random.choice(valid_cards)
        if self.check_rule(card):
            self.play_card(player, card)
            return True
        else:
            if not self.draw_card(player):
                return False
            if self.ai_strategy_second(player):
                return True
            else:
                return False
    else:
        if not self.draw_card(player):
            return False
        if self.ai_strategy_second(player):
            return True
        else:
            return False

def ai_strategy_second(self, player):
    valid_cards = [card for card in self.players[player].hand if
```

```
                    card[1] == self.trick_pile[-1][1] or
                    card[0] == self.trick_pile[-1][0] or
                    card[1] == self.current_suit
                    ]

            if len(valid_cards) > 0:
                card = random.choice(valid_cards)
                if self.check_rule(card):
                    self.play_card(player, card)
                    return True
            return False

        def check_rule(self, card):
            last_card = self.trick_pile[-1]
            if card[0] == 'Q':
                return True
            if last_card[0] == 'Q' and card[1] == self.current_suit:
                return True
            if last_card[0] != 'Q' and (card[1] == self.trick_pile[-1][1] or card[0] ==
self.trick_pile[-1][0]):
                return True
            return False

        def play_card(self, player, card):
            self.players[player].play_card(card)
            self.trick_pile.append(card)

        def shaffle_deck_ingame(self):
            last_card = self.trick_pile.pop(-1)
            random.shuffle(self.trick_pile)
```

```
        self.deck.cards = self.trick_pile + self.deck.cards
        self.trick_pile = [last_card]

    def draw_card(self, player):
        if self.deck.get_deck_size() <= 0:
            if len(self.trick_pile) > 1:
                self.shaffle_deck_ingame()
                card = self.deck.draw_card()
                self.players[player].draw_card(card)
                return True
            else:
                return False
        else:
            card = self.deck.draw_card()
            self.players[player].draw_card(card)
            return True

    def play_card_action(self):
        self.first_button.setEnabled(True)
        card_choosing = True
        while card_choosing:
            time.sleep(2)
            card_number = self.choose_card.choosed_card_2
            card_number = int(card_number) - 1
            if card_number == -1:
                self.choose_card.choosed_card_2 = 0
                if self.choose_card.is_take_card == 1:
                    return False
            else:
                if card_number < len(self.players[self.current_player].hand):
```

```
                card = self.players[self.current_player].hand[card_number]
                if self.check_rule(card):
                    self.play_card(self.current_player, card)
                    if card[0] == '9':
                        self.current_player = self.apply_additional_rules()
                    self.choose_card.choosed_card_2 = 0
                    self.choose_card.is_take_card = 3
                    self.first_button.setEnabled(False)
                    return True
                else:
                    self.choose_card.choosed_card_2 = 0
                    self.choose_card.is_take_card = 3
                    if self.choose_card.is_take_card == 1:
                        return False
            else:
                self.choose_card.is_take_card = 3
                self.choose_card.choosed_card_2 = 0
                if self.choose_card.is_take_card == 1:
                    return False
            self.update_design_ingame()
        self.choose_card.is_take_card = 3
        self.choose_card.choosed_card_2 = 0
        time.sleep(2)
    self.first_button.setEnabled(False)

def get_card_checked(self):
    if self.deck.get_deck_size() <= 0:
        if len(self.trick_pile) > 1:
            self.shaffle_deck_ingame()
            card = self.deck.draw_card()
```

```
            self.players[self.current_player].draw_card(card)
        else:
            self.is_end_game = True
    else:
        card = self.deck.draw_card()
        self.players[self.current_player].draw_card(card)

def apply_additional_rules(self):
    check = False
    for player in self.players:
        if len(player.hand) == 0:
            last_card = self.trick_pile[-1]
            if last_card[0] == '9':
                check = False
            else:
                check = True
        else:
            check = False
    if check:
        return (self.current_player + 1) % self.num_players

    card = self.trick_pile[-1]
    rank = card[0]
    suit = card[1]

    if rank == 'A':
        self.current_player = (self.current_player + 1) % self.num_players

    elif rank == 'Q':
        self.update_design_ingame()
```

```
        suit_choosing = True
        while suit_choosing:
            time.sleep(2)
            if self.current_player == 0:
                self.second_button.setEnabled(True)
                suit = self.choose_suit.choosed_suit
                if suit == -1:
                    pass
                else:
                    self.current_suit = self.deck.suits[suit]
                    self.choose_suit.choosed_suit = -1
                    suit_choosing = False
            elif self.current_player != 0:
                suit = random.randint(0, 3)
                self.current_suit = self.deck.suits[suit]
                self.ai_choosed = suit
                suit_choosing = False
        self.second_button.setEnabled(False)

    elif rank == 'K' and suit == 'spades':
        self.current_player = (self.current_player + 1) % self.num_players
        for _ in range(4):
            self.get_card_checked()

    elif rank == '9':
        self.nine_card()

    elif rank == '7':
        self.current_player = (self.current_player + 1) % self.num_players
```

```
            for _ in range(2):
                self.get_card_checked()

        elif rank == '6':
            self.current_player = (self.current_player + 1) % self.num_players
            self.get_card_checked()

        # Возвращает следующего игрока
        time.sleep(0.5)
        self.update_design_ingame()
        time.sleep(0.5)
        return (self.current_player + 1) % self.num_players

    def nine_card(self):
        self.update_design_ingame()
        time.sleep(1)
        if self.current_player == 0:
            self.first_button.setEnabled(True)
            choosing_action = True
            while choosing_action:
                time.sleep(2)
                action = self.get_card()
                if action == "1":
                    if self.play_card_action():
                        self.add_rules_ninecard()
                        choosing_action = False
                elif action == "2":
                    if not self.draw_card(self.current_player):
                        choosing_action = False
                    else:
```

```
                self.nine_card()
                choosing_action = False
        elif self.current_player != 0:
            time.sleep(2)
            valid_cards = [card for card in self.players[self.current_player].hand if
                           card[1] == self.trick_pile[-1][1] or
                           card[0] == self.trick_pile[-1][0] or
                           card[0] == 'Q']
            if len(valid_cards) > 0:
                card = random.choice(valid_cards)
                self.play_card(self.current_player, card)
                if card[0] == '9':
                    time.sleep(1)
                    self.update_design_ingame()
                    self.nine_card()
                else:
                    self.add_rules_ninecard()
                    self.update_design_ingame()
                    time.sleep(1)
            else:
                if self.is_end_game:
                    time.sleep(1)
                    self.update_design_ingame()
                    time.sleep(1)
                else:
                    self.get_card_checked()
                    self.update_design_ingame()
                    time.sleep(1)
                    self.nine_card()
```

```
def add_rules_ninecard(self):
    self.update_design_ingame()
    time.sleep(0.5)
    card = self.trick_pile[-1]
    rank = card[0]
    suit = card[1]

    if rank == 'A':
        self.current_player = (self.current_player + 1) % self.num_players

    elif rank == 'Q':
        self.update_design_ingame()
        suit_choosing = True
        while suit_choosing:
            time.sleep(2)
            if self.current_player == 0:
                self.second_button.setEnabled(True)
                suit = self.choose_suit.choosed_suit
                if suit == -1:
                    pass
                else:
                    self.current_suit = self.deck.suits[suit]
                    self.choose_suit.choosed_suit = -1
                    suit_choosing = False
            elif self.current_player != 0:
                suit = random.randint(0, 3)
                self.current_suit = self.deck.suits[suit]
                self.ai_choosed = suit
                suit_choosing = False
```

```
                self.second_button.setEnabled(False)

            elif rank == 'K' and suit == 'spades':
                self.current_player = (self.current_player + 1) % self.num_players
                self.update_design_ingame()
                for _ in range(4):
                    self.get_card_checked()

            elif rank == '7':
                self.current_player = (self.current_player + 1) % self.num_players
                self.update_design_ingame()

                for _ in range(2):
                    self.get_card_checked()

            elif rank == '6':
                self.current_player = (self.current_player + 1) % self.num_players
                self.update_design_ingame()

                self.get_card_checked()

        def check_game_over(self):
            for player in self.players:
                if len(player.hand) == 0:
                    return True

            if len(self.deck.cards) == 0 and all(len(player.hand) == 0 for player in
self.players):
                return True
            return False
```

```
        def update_scores(self, scores):
            for i, player in enumerate(self.players):
                score = calculate_score(player.hand)
                scores[i] = score
            self.current_player = 0
            self.current_player = (self.current_player + self.round_number) %
self.num_players
            self.round_number = self.round_number + 1
            return scores

        def reset_settings(self):
            player_names = [player.name for player in self.players]

            self.players.clear()

            for name in player_names:
                self.players.append(Player(name))

            self.choosed_card = 0
            self.modes = 0
            self.num_players = 4
            self.trick_pile = []

            self.current_suit = None
            self.ai_choosed = None
            self.is_end_game = None
            self.deck = Deck()
            self.deck.shuffle()
            self.deal_cards()
```

```
        self.trick_pile.append(self.deck.draw_card())

    def display_scores(self, scores):
        self.points_player1.setText(f"Кількість очок: {scores[0]}")
        self.points_player2.setText(f"Кількість очок: {scores[1]}")
        self.points_player3.setText(f"Кількість очок: {scores[2]}")
        self.points_player4.setText(f"Кількість очок: {scores[3]}")

    def run(self):
        self.dev_mode.setEnabled(True)
        game_scores = [0, 0, 0, 0]

        game_over = False
        while not game_over:
            last_scores = [0, 0, 0, 0]
            time.sleep(1)
            self.play_round()
            last_scores = self.update_scores(last_scores)
            self.reset_settings()
            for i in range(0, 4):
                game_scores[i] = game_scores[i] + last_scores[i]
            game_scores = check_scores(game_scores)
            game_over = is_game_over(game_scores)
            self.display_scores(game_scores)

        final_winners, final_scores = determine_winner(game_scores)
        save_results(final_winners, final_scores, self.players)
        self.setWindowTitle(f"Гра закінчилась. Переміг: {final_winners}")

        self.faq.setEnabled(False)
```

```
        self.start_game.setEnabled(False)
        self.end_game.setEnabled(False)
        self.dev_mode.setEnabled(False)
        self.first_button.setEnabled(False)
        self.second_button.setEnabled(False)
        self.save_game_results.setEnabled(True)
```

Player.py

```
class Player:
    def __init__(self, name):
        self.name = name
        self.hand = []

    def add_card(self, card):
        self.hand.append(card)

    def play_card(self, card):
        self.hand.remove(card)

    def draw_card(self, card):
        self.hand.append(card)
```

Deck.py

```
import random


class Deck:
    def __init__(self):
        self.cards = []
        self.suits = ['spades', 'hearts', 'diamonds', 'clubs']
        self.ranks = ['6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
        self.create_deck()
```

```
            self.shuffle()

        def create_deck(self):
            self.cards = [(rank, suit) for suit in self.suits for rank in self.ranks]

        def shuffle(self):
            random.shuffle(self.cards)

        def draw_card(self):
            return self.cards.pop(0)

        def get_deck_size(self):
            return len(self.cards)
```

ChooseCard.py

```
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QLabel, QPushButton, QMessageBox, QWidget, QTextEdit
from PyQt5 import uic
from config import *


class ChooseCard(QWidget):
    def __init__(self):
        super(ChooseCard, self).__init__()
        self.choosed_card_2 = 0
        self.is_take_card = 3
        self.mytext = None

        uic.loadUi("designs/choose_card.ui", self)
        self.setWindowTitle("Виберіть карту")
```

```
        self.setWindowIcon(QIcon(f"{icon}"))

        self.info_label = self.findChild(QLabel, "info_label")
        self.to_choose_card = self.findChild(QPushButton, "to_choose_card")
        self.card_num = self.findChild(QTextEdit, "card_num")
        self.take_card = self.findChild(QPushButton, "take_card")

        self.initCard()

    def initCard(self):
        self.to_choose_card.clicked.connect(self.send_card)
        self.take_card.clicked.connect(self.send_take_card)

    def send_take_card(self):
        self.is_take_card = 1
        ChooseCard.hide(self)

    def send_card(self):
        self.mytext = self.card_num.toPlainText()
        res = self.mytext.isnumeric()
        if res:
            self.is_take_card = 0
            self.choosed_card_2 = self.mytext
            self.card_num.setPlainText("")
            self.mytext = None
            ChooseCard.hide(self)
        else:
            QMessageBox.information(self, "Увага!", "Неправильне використання, спробуйте знову")
```

ChooseSuit.py

```
from PyQt5.QtGui import QPixmap, QIcon
from PyQt5.QtWidgets import QLabel, QPushButton, QWidget
from PyQt5 import uic
from config import *


class ChooseSuit(QWidget):
    def __init__(self):
        super(ChooseSuit, self).__init__()
        self.choosed_suit = -1

        uic.loadUi("designs/choose_suit.ui", self)
        self.setWindowTitle("Виберіть масть")
        self.setWindowIcon(QIcon(f"{icon}"))

        self.text = self.findChild(QLabel, "text")

        self.clubs = self.findChild(QLabel, "clubs")
        self.diamonds = self.findChild(QLabel, "diamonds")
        self.spades = self.findChild(QLabel, "spades")
        self.hearts = self.findChild(QLabel, "hearts")

        self.choose_clubs = self.findChild(QPushButton, "choose_clubs")
        self.choose_diamonds = self.findChild(QPushButton, "choose_diamonds")
        self.choose_spades = self.findChild(QPushButton, "choose_spades")
        self.choose_hearts = self.findChild(QPushButton, "choose_hearts")

        self.initForm()

    def initForm(self):
```

```
        pixmap1 = QPixmap(f"images/Q_of_clubs.png")
        pixmap2 = QPixmap(f"images/Q_of_diamonds.png")
        pixmap3 = QPixmap(f"images/Q_of_spades.png")
        pixmap4 = QPixmap(f"images/Q_of_hearts.png")
        self.clubs.setPixmap(pixmap1)
        self.diamonds.setPixmap(pixmap2)
        self.spades.setPixmap(pixmap3)
        self.hearts.setPixmap(pixmap4)

        self.choose_clubs.clicked.connect(self.first)
        self.choose_diamonds.clicked.connect(self.second)
        self.choose_spades.clicked.connect(self.third)
        self.choose_hearts.clicked.connect(self.fourth)

    def first(self):
        self.choosed_suit = 3
        ChooseSuit.hide(self)

    def second(self):
        self.choosed_suit = 2
        ChooseSuit.hide(self)

    def third(self):
        self.choosed_suit = 0
        ChooseSuit.hide(self)

    def fourth(self):
        self.choosed_suit = 1
        ChooseSuit.hide(self)
```

functions.py

```
def calculate_score(hand):
    score = 0
    minus_points = 0

    for card in hand:
        rank = card[0]
        if rank.isdigit():
            if rank == '9':
                score += 0
            else:
                score += int(rank)
        elif rank in ['J', 'Q', 'K']:
            score += 10
        elif rank == 'A':
            score += 11

    if score == 0:
        for card in hand:
            rank = card[0]
            suit = card[1]
            if rank == 'Q':
                if suit == 'spades':
                    minus_points = 20
                else:
                    minus_points = 40

    score -= minus_points
    return score
```

```
def check_scores(scores):
    if scores[0] == 101:
        scores[0] = 0
    if scores[1] == 101:
        scores[1] = 0
    if scores[2] == 101:
        scores[2] = 0
    if scores[3] == 101:
        scores[3] = 0
    return scores


def save_results(winners, scores, players):
    with open("results.txt", "w") as file:
        file.write("Результати гри:\n")
        file.write("----------------\n")
        for i, player in enumerate(players):
            file.write(f"Гравець {player.name}: {scores[i]} очок\n")
        file.write("----------------\n")
        file.write("Переможці:\n")
        file.write("----------------\n")
        for winner in winners:
            file.write(f"Гравець {players[winner - 1].name}\n")


def determine_winner(final_scores):
    min_score = min(final_scores)
    winners = [i + 1 for i, score in enumerate(final_scores) if score == min_score]
    return winners, final_scores
```

```
def is_game_over(scores):
    max_score = max(scores)
    if max_score > 101:
        return True
    else:
        return False
```

Gameisover.py

```
from main import *
import sys
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QLabel, QPushButton, QWidget
from PyQt5 import uic
from config import *


class Gameisover(QWidget):
    def __init__(self):
        super(Gameisover, self).__init__()

        uic.loadUi("designs/gameover.ui", self)
        self.setWindowTitle("Гру завершено!")
        self.setWindowIcon(QIcon(f"{icon}"))

        self.label = self.findChild(QLabel, "label")
        self.label_2 = self.findChild(QLabel, "label_2")
        self.close_game = self.findChild(QPushButton, "close_game")

        self.close_game.clicked.connect(self.end)
```

```
    def end(self):
        self.hide()
        sys.exit(app.exec_())
```

config.py

```
text = '''Кількість колод: 1
Кількість карт у колоді: 36
Кількість гравців: 2 - 4
Старшинство карток: 6, 7, 8, 9, 10, В, Д, К, Т.
Ціль гри: набрати найменшу кількість очок.
```

Правила гри. Досить популярна гра на території Росії та колишніх країн СНД. Для того, щоб грати, потрібно колода з 36 карт і від 2 до 4 гравців. Перший здавач карток у грі визначається жеребом, у наступних іграх гравці здають картки по черзі. Колода ретельно тасується, знімається та кожному гравцю здається по 5 карт. Здавальник здає собі 4 карти, а п'ята карта кладеться на кін. Колода, що залишилася, кладеться по центру столу в закритому вигляді. Гравець ліворуч від здавача продовжує гру. Він повинен покласти на п'яту карту здавача карту такої ж масті, або карту такого ж значення, якщо такої карти немає, то гравець бере з колоди одну картку і якщо ця карта не підходить, то гравець пропускає хід. Деякі карти мають свої особливості у грі, а саме: тузи, пані, піковий король, дев'ятки, сімки та шістки.

Туз необхідно класти тільки в масть, або іншого туза. Туз забороняє перебіг наступному гравцю. Під час гри удвох хід залишається у гравця. Якщо грають утрьох чи вчотирьох, то хід переходить через одного гравця.

Дами можуть лягати на будь-яку карту і на будь-яку масть. Гравець, який покладе даму, замовляє собі вигідну масть і хід переходить до наступного гравця. Якщо гравець закінчив гру на будь-яку даму, то у нього забирається 20 очок, якщо гравець закінчив гру на пікову даму, то у нього забирається 40 очок.

Пікового короля можна покласти тільки на короля будь-якої масті та на будь-яку пікову карту. Наступний гравець, який ходить, бере з колоди 4 карти та пропускає хід.

Якщо гравець покладе дев'ятку, він повинен її закрити тією ж мастю чи дев'яткою. Дев'ятку має знову закрити тією ж мастю. Якщо гравець не має таких карт, то гравець бере карти з колоди доти, доки не закриє дев'ятку.

Сімку можна покласти тільки на сімку або в масть, наступний гравець бере дві карти з колоди та пропускає хід.

Шістку можна покласти тільки на шістку або в масть, наступний гравець бере одну карту з колоди і пропускає хід.

Вартість карт: туз – 11 очок, 10 – 10 очок, 8 – 8 очок, 7 – 7 очок, 6 – 6 очок, король – 4 очки, дама – 3 очки, валет – 2 очки, 9 – 0 очок. Якщо у гравця 0 очок і він закінчив на даму, йому зараховується мінус 20 очок. Гра йде до тих пір, поки в одного з гравців не залишиться жодної карти, або до тих пір, поки хтось із гравців не набере більше 101 очка. Той гравець, який набере більше 101 очка, вважається таким, що програв. Якщо будь-який гравець набере 101 очко, його рахунок обнуляється.'"

turned = "images/turned_card.jpg"

icon = "images/icon.png"

Gameisover.py

```
from main import *
import sys
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QLabel, QPushButton, QWidget
from PyQt5 import uic
from config import *


class Gameisover(QWidget):
```

```
def __init__(self):
    super(Gameisover, self).__init__()

    uic.loadUi("designs/gameover.ui", self)
    self.setWindowTitle("Гру завершено!")
    self.setWindowIcon(QIcon(f"{icon}"))

    self.label = self.findChild(QLabel, "label")
    self.label_2 = self.findChild(QLabel, "label_2")
    self.close_game = self.findChild(QPushButton, "close_game")

    self.close_game.clicked.connect(self.end)

def end(self):
    self.hide()
    sys.exit(app.exec_())
```