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

Кафедра
автоматизованих систем обробки інформації та управління
(повна назва кафедри, циклової комісії)

# КУРСОВА РОБОТА

з «Основи програмування - 2. Основи об'єктно-орієнтованого програмування»

(назва дисципліни)

на тему: «Карткова гра «Скарбнички» проти комп’ютерного опонента»

Студента 1 курсу, групи ІП-02
Василенка Павла Олександровича

Спеціальності 121 «Інженерія програмного забезпечення»

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

Кількість балів: _____
Національна оцінка _____

Члени комісії

| | |
|---|---|
| _____ | ст. в. Головченко М. М |
| (підпис) | (вчене звання, науковий ступінь, прізвище та ініціали) |
| _____ | к. т. н. доц. Муха І. П. |
| (підпис) | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ - 2021 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Основи програмування

Напрям "Інженерія програмного забезпечення"

Курс 1 Група ІП-02 Семестр 2

# ЗАВДАННЯ

на курсову роботу студента

**Василенка Павла Олександровича**

(прізвище, ім'я, по батькові)

1. Тема роботи: "Карткова гра Скарбнички"

2. Строк здачі студентом закінченої роботи: «04» червня 2021 р.

3. Вихідні дані до роботи: Технічне завдання (Додаток А)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці): Вступ, постановка задачі, теоретичні відомості, опис алгоритмів, опис архітектури програмної системи, тестування програмного забезпечення, інструкція користувача, висновок, перелік посилань.

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

6. Дата видачі завдання: 08.03.2021

КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
|---|---|---|---|
| 1. | Отримання теми курсової роботи | 08.03.2021 | |
| 2. | Підготовка технічного завдання | 10.03.2021 | |
| 3. | Аналіз предметної області | 20.03.2021 | |
| 4. | Проектування архітектури програмної системи | 02.04.2021 | |
| 5. | Узгодження з керівником інтерфейсу користувача | 03.04.2021 | |
| 6. | Розробка сценарію роботи програми | 05.04.2021 | |
| 7. | Розробка програмного забезпечення | 24.04.2021 | |
| 8. | Узгодження з керівником плану тестування | 28.04.2021 | |
| 9. | Тестування програми | 01.05.2021 | |
| 10. | Підготовка пояснювальної записки | 15.05.2021 | |
| 11. | Здача курсової роботи на перевірку | 02.06.2021 | |
| 12. | Захист курсової роботи | 04.06.2021 | |

Студент _____ Василенко П. О.

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

Керівник _____ Головченко М.М.

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

"__" _____ 2021 р

# АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 100 сторінок, 16 рисунків, 45 таблиць, 2 посилання.

Об’єкт дослідження: “Карткова гра Скарбнички”[2]. Можливість гри проти комп’ютерного опонента.

Метою курсової роботи є розробка програмного забезпечення, що передбачає можливість гри в «Скарбнички» проти комп’ютера.

В процесі розробки було сконструйовано власний алгоритм прийому кращих ходів для комп’ютерного опонента. Було створено ігровий інтерфейс. Приведені змістовні постановки задач та виконано усі задачі з переліку.

“СКАРБНИЧКИ”, КАРТКОВА ГРА, АЛГОРИТМ ПРИЙОМУ РІШЕНЬ, ВЕБ-ДОДАТОК.

ЗМІСТ

# ВСТУП

З розвитком індустрії інформаційних технологій збільшується попит на ігри. Так, збільшується і кількість роботів, що вміють грати в ці ігри. Сьогодні складно собі уявити гру в якій не буде штучного інтелекту.

Люди створюють штучний інтелект, що зможе перевершити їх. Так у 1997 році було створено Deep Blue, комп'ютера, що зміг перемогти світового чемпіона в шахах Гаррі Каспарова.

Більш складною задачею виявилася розробка штучного інтелекту для перемоги в китайську древню гру «Го». Лише у 2015 році програма AlphaGo, розроблена компанією Deepmind перемогла трьохкратного чемпіона Європи Фань Хуея з рахунком 4-1.

Усі перераховані вище роботи, що змогли перемогти людину використовували машинне навчання(machine learning), що давало їм самостійно вчитися та запам'ятовувати кращі ходи. Метою цієї роботи є розробка карткової гри «Скарбнички» проти комп'ютерного інтелекту, з заздалегідь продуматим лінійним алгоритмом. Для правильного алгоритму гри має бути проаналізовано всі можливі ситуації та виходи з них.

Шляхом практичних досліджень було встановлено всі можливі випадки в грі та те, як має діяти людина для перемоги. Перевагою людини є зчитування емоцій, що допомагає в прийнятті рішень. Комп'ютер же має підходити до всього лише математично.

# 1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що дає змогу користувачу грати в карткову гру «Скарбнички» проти комп’ютерного суперника.

Відкривши чи оновивши сторінку веб-застосунку перед користувачем з’явиться модальне вікно з кнопкою початку гри. Натиснувши на цю кнопку розпочнеться гра за відповідними правилами описаними в розділі «Теоретичні відомості». Правила порушувати неможливо. Інтерфейс користувача включає в себе можливість слідкувати за кількістю карт в колоді, в руці суперника, кількістю скарбничок обох гравців, та картами в своїй руці. По закінченню гри має виводитися вікно з оголошенням перемоги/поразки гравця з кнопкою «Грати знову».

# 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Правила гри «Скарбнички»[1]:

Кількість колод: 1

Кількість карт в колоді: 36

Кількість гравців: 2

Старшинство карт: 6,7,8,9,10,J,Q,K,A

Ціль гри: зібрати якнайбільше скарбничок, до того моменту, як карти в колоді й у обох гравців закінчаться

Хід гри: Колода ретельно тасується. Кожен гравець отримує по 4 карти, а колода, що залишилася, кладеться на центр столу сорочкою догори. Першим ходить користувач – надалі по черзі. Спочатку гравець питає у суперника про наявність карти/карт таким чином: значення карти, якщо вгадує, то вгадує кількість карт і якщо угадує то питає масті. Гравець може спитати лише карту такого значення, яке є в нього в руці. Якщо у суперника наявні всі запитані карти – він віддає їх гравцю. Якщо у суперника не виявилося жодної з запитаних карт – гравець бере верхню карту з колоди. Якщо у суперника є лише частина з загаданих карт, то він віддає їх гравцю, а гравець бере карту з колоди. Якщо у гравця чи комп’ютера в руках виявиться 4 однакові за значенням карти – вони збираються в скарбничку й відповідний гравець прибирає їх зі своєї руки. Якщо на будь-якому етапі гри у гравця виявилося менше 4-х карт – він бере з колоди рівно стільки карт, скільки йому не вистачає. Гру вважають закінченою, коли колода карт – пуста а у гравців нема карт в руках. Після цього рахуються скарбнички кожного та визначається переможець.

# 3 ОПИС АЛГОРИТМІВ

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

Таблиця 3.1 – Основні змінні та їхні призначення

| Змінна | Призначення |
|---|---|
| enemyHand | Рука комп’ютера – масив карт, які комп’ютер може використовувати |
| deckSize | Кількість карт, що залишилися в колоді (від 0 до 28) |
| imagineHand | Уявна рука гравця – комп’ютер запам’ятовує всі карти, що гравець в нього забирав, угадуючи, а також всі карти, що гравець питав, але не вгадував |
| lastUsed | Масив карт, що комп’ютер використовував нещодавно, тому немає сенсу використовувати ще раз. Максимальна к-сть таких карт – 3. |
| value | Значення карти – 6 7 8 9 10 J Q K A |
| suit | Масть карти - ♣ ♠ ♦ ♥ |
| chestsArr | Масив скарбничок, що є кожного гравця |
| chestsNum | Кількість скарбничок у кожного гравця |

## 3.1 Загальний алгоритм комп’ютера під час ходу гравця

1. ПОЧАТОК
    2. Якщо deckSize>0 і enemyHand.size>0
        2.1. Гравець запитує value карти
        2.2. Якщо гравець вгадав
            2.2.1. Гравець запитує кількість таких карт
            2.2.2. Якщо гравець вгадав
                2.2.2.1. Гравець вгадує масті карт
                2.2.2.2. Якщо гравець вгадав усі карти
                    2.2.2.2.1. Передати вгадані гравцем карти
                    2.2.2.2.2. Записати value, accurate=true, isNot=[], suit до ImagineHand
                    2.2.2.2.3. Записати value, accurate=false, до ImagineHand
                    2.2.2.2.4. Почати хід (алгоритм 3.2)
                2.2.2.3. Інакше якщо гравець вгадав частину карт
                    2.2.2.3.1. Передати вгадані гравцем карти

Продовження алгоритму 3.1

2.2.2.3.2. Записати value, accurate=true, isNot = [], suit до ImagineHand (всі карти, що гравець вгадав)

2.2.2.3.3. Додати value, accurate=false,isNot=[suit] до ImagineHand (карти що гравець назвав та не вгадав)

2.2.2.3.4. Почати хід (алгоритм 3.2)

2.2.2.4. Інакше

2.2.2.4.1. Записати value, accurate=false, isNot = [suit] до ImagineHand (всі масті названі гравцем)

2.2.2.4.2. Почати хід (алгоритм 3.2)

2.2.3. Інакше

2.2.3.1. Записати value, accurate=false до ImagineHand, якщо такої ще нема

2.2.3.2. Почати хід (алгоритм 3.2)

2.3. Інакше

2.3.1. Записати value, accurate=false до ImagineHand, якщо такої ще нема

2.3.2. Почати хід (алгоритм 3.2)

3. Інакше

3.1. Гру завершено

4. КІНЕЦЬ

**3.2 Загальний алгоритм ходу комп'ютера**

1 ПОЧАТОК

2 Поки enemyHand.size<4

2.1 Взяти карту з колоди

3 Кінець поки

4 Priority = -1, bestcardValue = null

5 Якщо в deckSize==0

5.1 bestcardValue = першій в руці карті

Продовження алгоритму 3.2

6 Інакше

6.1 Для кожної можливої карти з масиву values

6.1.1 Визначити newPriority цієї карти

6.1.2 Якщо newPriority > priority

6.1.2.1 Priority=newPriority

6.1.2.2 bestCardValue = value

7 Якщо обрана карта є у гравця

7.1 Якщо deckSize=0

7.1.1 numberToAsk = 4-(кількість цієї карти в руці комп’ютера)

7.2 Інакше

7.2.1 numberToAsk = кількість цієї карти в ImagineHand

7.2.2 Якщо numberToAsk=0

7.2.2.1 numberToAsk++

7.3 inHand = кількість карт з value == bestCardValue в руці комп’ютера

7.4 Поки (inHand+numberToAsk>4)

7.4.1 numberToAsk—

7.5 Якщо вгадали кількість карт в руці гравця

7.5.1 В imagineHand знайти карти з value==bestCardValue і accurate == true, і додаємо їх до suitsInImagineAccurate

7.5.2 Знайти всі карти з value==bestCardValue і isNot.length>0, і кожну масть з масивів isNot.length записуємо до suitsInImagineNotArr

7.5.3 Знайти всі карти в руці з value==bestCardValue, та записуємо масті до suitsInHand

7.5.4 Враховуючи інформацію з попередніх трьох пунктів створити suitsToAsk (обов'язково мають бути масті з suitsInImagineAccurateб і не має бути мастей з suitsInHand та suitsInImagineNotArr)

7.5.5 Для кожної карти, що намагаємося вгадати

7.5.5.1 Якщо ця карта є у гравця

7.5.5.1.1 Забрати карту

Продовження алгоритму 3.2

7.5.5.1.2 Покласти собі в руку

7.5.5.1.3 Видалити цю карту з imagineHand(з найменшою кількістю інформації)

7.5.5.2 Інакше

7.5.5.2.1 Якщо в ImagineHand ще нема карти з таким value, accurate=false, isNot.length=0

7.5.5.2.1.1 Додати таку карту

7.5.5.2.2 Інакше

7.5.5.2.2.1 Не додавати нічого

7.5.6 Кінець циклу

7.5.7 Якщо к-сть вгаданих карт – 0

7.5.7.1 Узяти карту з колоди

7.5.7.2 Перевірити на скарбничку (адгоритм 3.4)

7.5.8 Інакше якщо кількість вгаданих карт менша за кількість вгадуваних карт

7.5.8.1 Узяти карту з колоди

7.5.8.2 Перевірити на скарбничку

7.5.9 Інакше

7.5.9.1 Нічого не робити

7.5.10 Перевірити на наявність карти в масиві скарбничок

7.5.11 Якщо вона там є

7.5.11.1 Прибрати всі копії цієї карти з ImagineHand і LastUsed

7.6 Інакше

7.6.1 Якщо в ImagineHand 1 копія карти

7.6.1.1 Додати value=bestCardValue accurate = false isNow=[]

7.6.2 Інакше якщо 3 карти АБО 2 карти і deckSize>14

7.6.2.1 Видалити одну копію карти з ImagineHand(з найменшою кількістю інформації)

7.6.3 Інакше якщо 0 карт

Продовження алгоритму 3.2

7.6.3.1 Додати value=bestCardValue accurate = false isNow=[]. Двічі.

7.6.4 Інакше якщо 2 карти

7.6.4.1 Додати value=bestCardValue accurate = false isNow=[].

7.6.5 Взяти карту з колоди

7.6.6 Перевірити на наявність скарбничок (алгоритм 3.4)

8 Інакше

8.1 Взяти карту з колоди

8.2 Перевірити на наявність скарбничок (алгоритм 3.4)

9 Поки enemyHand.size<4

9.1 Взяти карту з колоди

9.2 Перевірити на наявність скарбнички (алгоритм 3.4)

10 КІНЕЦЬ

### 3.3 Загальний алгоритм вибору кращого ходу

1 ПОЧАТОК

2 newPriority=0;

3 Визначити кількість цієї карти в руці. (inHand)

4 Якщо inHand==0

4.1 NewPriority=-1

5 Інакше якщо inHand==1

5.1 Визначити наявність цієї карти в lastused (cardInLastUsed)

5.2 Якщо ця карта є в lastUsed

5.2.1 Порахувати кількість точних (accurate==true) мастей в ImagineHand

5.2.2 Якщо їх 0

5.2.2.1 newPriority = 0

5.2.3 Інакше якщо їх 1

5.2.3.1 newPriority = 1200

5.2.4 Інакше якщо їх 2

5.2.4.1 newPriority = 2500

Продовження алгоритму 3.3

5.2.5 Інакше якщо їх 3

5.2.5.1 newPriority = 10000

5.3 Інакше

5.3.1 Порахувати кількість цієї карти в ImagineHand

5.3.1.1 Якщо їх 0

5.3.1.1.1 newPriority = 1000

5.3.1.2 Якщо їх 1

5.3.1.2.1 Якщо це точна масть

5.3.1.2.1.1 newPriority = 1800

5.3.1.2.2 Інакше

5.3.1.2.2.1 newPriority = 1100

5.3.1.3 Якщо їх 2

5.3.1.3.1 Якщо точних мастей серед них 0

5.3.1.3.1.1 newPriority = 2100

5.3.1.3.2 Інакше якщо точних мастей 1

5.3.1.3.2.1 newPriority = 2500

5.3.1.3.3 Інакше якщо точних мастей 2

5.3.1.3.3.1 newPriority = 2800

5.3.1.4 Якщо їх 3

5.3.1.4.1 newPriority = 10000

6 Інакше якщо inHand==2

6.1 Визначити чи є ця карта в lastused(cardInLastUsed)

6.2 Якщо вона там є

6.2.1 Визначити кількість точних мастей в ImagineHand

6.2.2 Якщо їх 0

6.2.2.1 newPriority = 0

6.2.3 Інакше якщо їх 1

6.2.3.1 newPriority = 2800

6.2.4 Інакше якщо їх 2

Продовження алгоритму 3.3

6.2.4.1 newPriority = 10000

6.3 Інакше

6.3.1 Визначити кількість цієї карти в ImagineHand

6.3.2 Якщо їх 0

6.3.2.1 newPriority = 1800

6.3.3 Інакше якщо їх 1

6.3.3.1 Якщо це точна масть

6.3.3.1.1 newPriority = 2800

6.3.3.2 Інакше

6.3.3.2.1 newPriority = 2500

6.3.4 Інакше якщо їх 2

6.3.4.1 newPriority = 10000

7 Інакше якщо inHand==3

7.1 Визначити чи є ця карта в lastused(cardInLastUsed)

7.2 Якщо вона там є

7.2.1 Визначити кількість цієї карти в ImagineHand

7.2.2 Якщо їх 0

7.2.2.1 newPriority = 0

7.2.3 Інакше якщо їх 1

7.2.3.1 newPriority = 10000

7.3 Інакше

7.3.1 Визначити кількість цієї карти в ImagineHand

7.3.2 Якщо їх там 0

7.3.2.1 newPriority = 2600

7.3.3 Якщо їх там 1

7.3.3.1 newPriority = 10000

8 Інакше якщо inHand==4

8.1 Перевірити на наявність скарбнички (алгоритм 3.4)

9 КІНЕЦЬ

### 3.4 Загальний алгоритм дій, коли один з гравців збирає скарбничку

1 ПОЧАТОК

2 Counter = 0

3 Для кожної карти в руці

  3.1 Якщо її value співпадає з шуканим

    3.1.1 Counter++

4 Якщо counter==4

  4.1 Прибрати всі карти з цим value з руки, imagineHand, LastUsed

  4.2 Додати цей value до chestsArr

  4.3 chestsNum++

5 Якщо після збоору скарбнички deckSize==0 і enemyHand.size == 0

  5.1 Порахувати скарбнички кожного гравця

  5.2 Оголосити переможця

6 КІНЕЦЬ

# 4 ОПИС АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ

Опис програмного продукту представлений у двох форматах: у вигляді UML діаграм (Рисунок 4.1 — 4.7) та таблиці методів (Таблиця 4.1 — 4.2).

## 4.1 UML діаграми

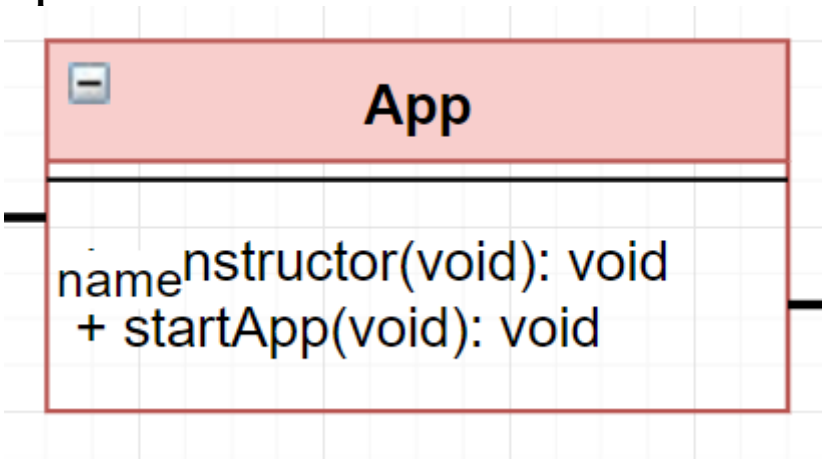


Рисунок 4.1 – Клас «App»

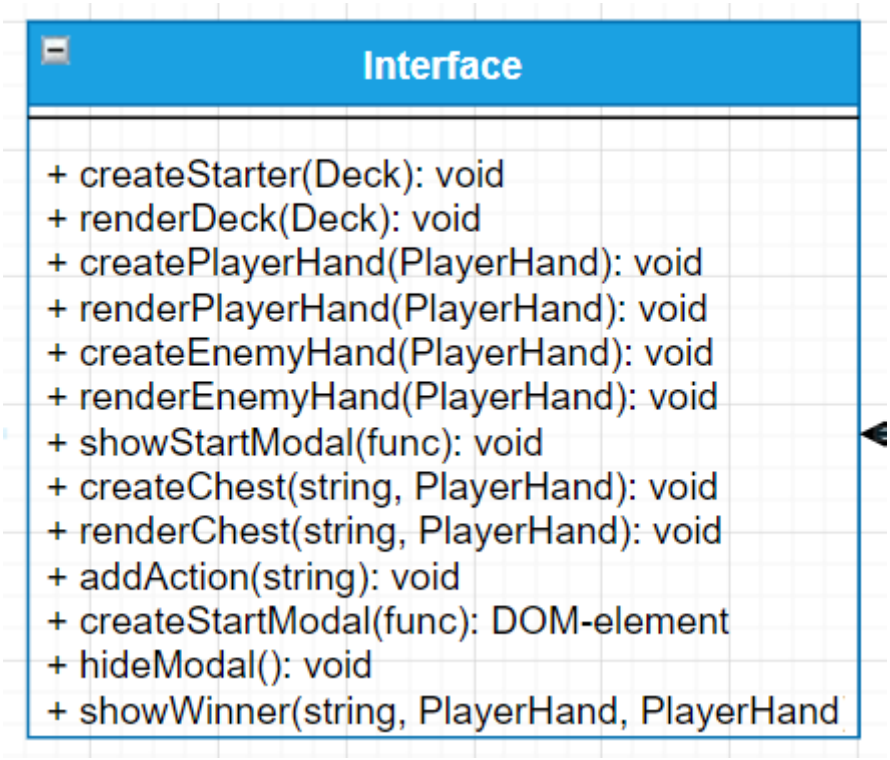


Рисунок 4.2 – Клас «Interface»

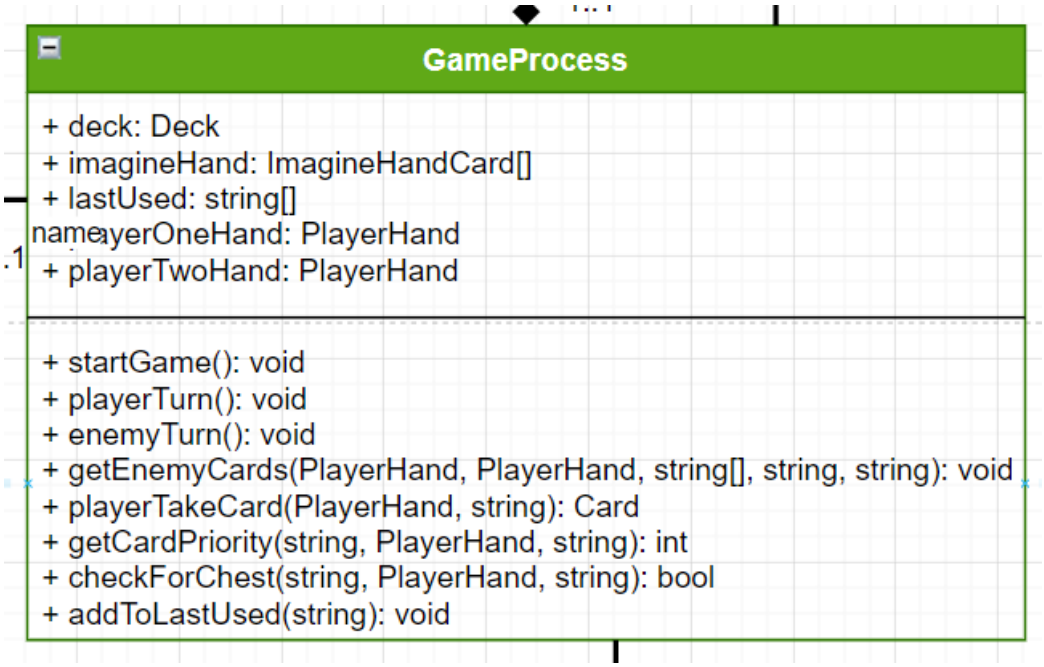


Рисунок 4.3 – Клас «GameProcess»

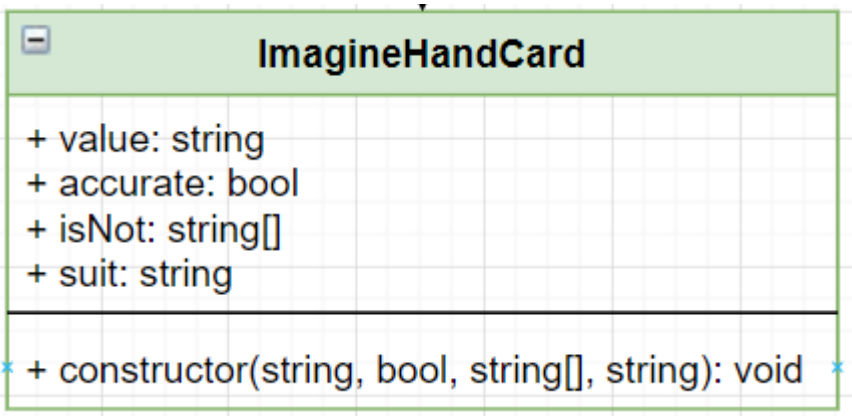


Рисунок 4.4 – Клас «ImagineHandCard»

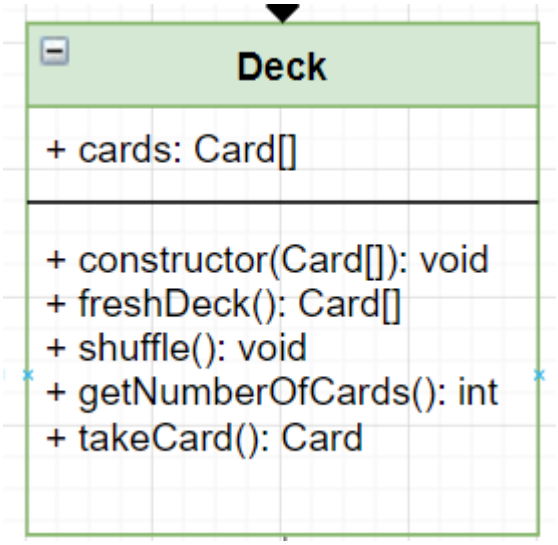


Рисунок 4.5 – Клас «Deck»

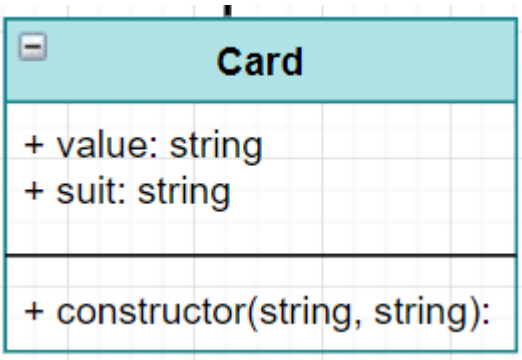


Рисунок 4.6 – Клас «Card»

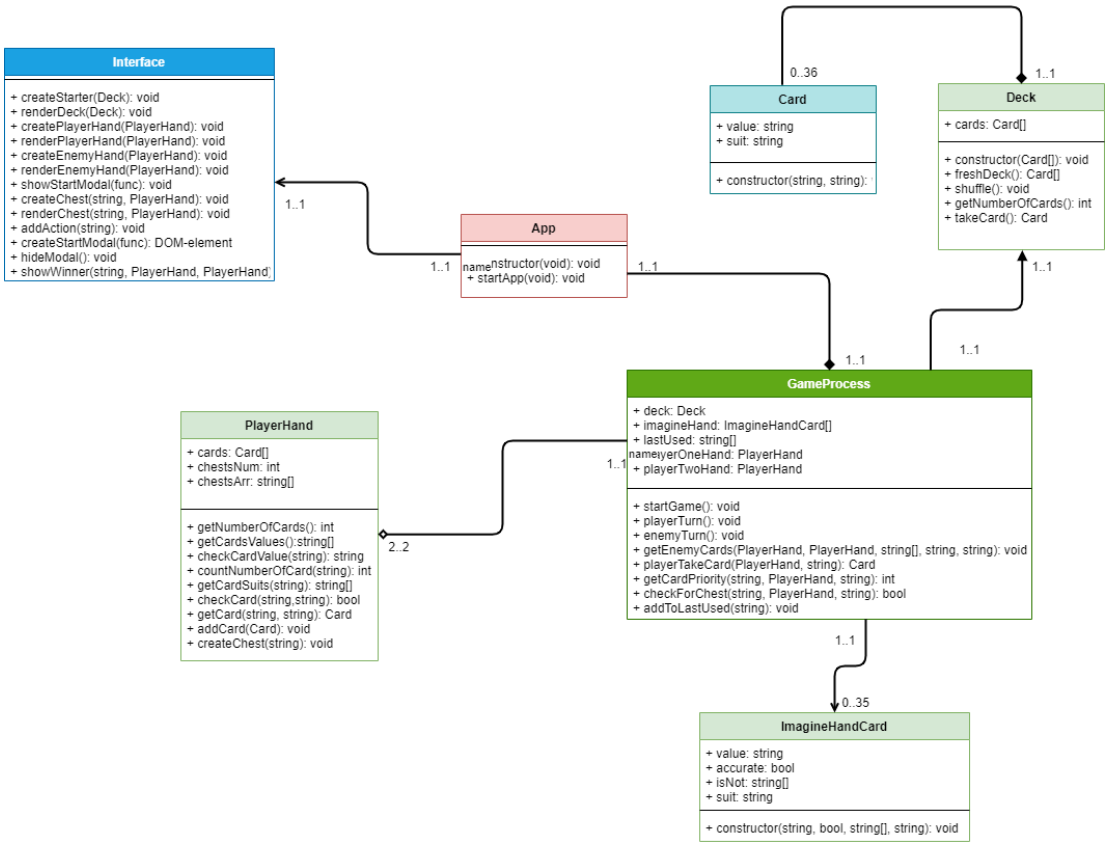


Рисунок 4.7 – Загальна діаграма класів

Робота програми починається зі створення екземпляра класу Interface і App. У своєму конструкторі App викликає метод startApp, що створює у свою чергу об’єкти класів Deck, PlayerHand(2 шт.) , а також викликає метод showStartModal класу Interface.

У конструктор класу Deck передається масив карт класу Card, створений методом freshDeck, і викликається метод shuffle. Deck не може існувати без класу Card.

У конструкторі класу PlayerHand створюється порожця рука без скарбничок, і 4 рази викликається метод взяття карти з колоди deck.

При закритті модального вікна створюється об'єкт класу GameProcess, що у своєму конструкторі приймає руки гравців та колоду карт. Також він запускає гру за допомогою метода startGame. GameProcess неможливий без рук гравців, але теоретично можливий без колоди.

Клас ImagineHandCard відповідає за спрощення роботи зі збереженням інформації про руку гравця для роботи алгоритму комп’ютера.

### 4.2 Таблиця методів

У таблиці 4.1 представлено вбудовані методи

Таблиця 4.1 – Вбудовані методи

| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
|---|---|---|---|---|---|
| 1 | Array | push | Вставка нового елемента в масив | Будь-який | Довжина новоутвореного масиву |
| 2 | Array | map | Створення нового масиву з результатами виклику вказаної функції для кожного елементу масиву | Функція, яку буде виконано до кожного елемента масиву | Новий масив, де кожен елемент це результат виклику функції |
| 3 | Array | pop | Вилучення останнього елементу масива | - | Видалений елемент |

Продовження таблиці 4.1

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | Node | append | Додати вузли в DOM-дерево | Елементи DOM-дерева | - | - |
| 5 | Node | remove | Видалити вузол з DOM-дерева | - | - | - |
| 6 | Node | appendChild | Додати вузол до DOM-дерева | - | - | - |
| 7 | Node | setAttribute | Додати властивість до елемента DOM-дерева | Перший параметр – атрибут, що треба встановити. Другий – значення цього атрибута | - | - |
| 8 | Node | createElement | Створення нового елемента DOM-дерева | DOM-string тега, що треба створити | Створений елемент | - |
| 9 | String | split | Розбиття рядка на кілька елементів для отримання масиву | Символ-роздільник | Новий масив | - |

Продовження таблиці 4.1

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | Array | filter | Отримання нового масиву з елементів попереднього масиву, що задовольняють вказану умову | Функція, за якою проводить перевірка | Новий масив | - |
| 11 | Array | forEach | Виконання одній і тієї самої функції для кожного елементу масива | Функція, яку має бути виконано | - | - |
| 12 | Object | keys | Отримати всі ключі об’єкта | Об'єкт | Масив ключів | - |
| 13 | Array | find | Знайти елемент, що задовольняє умові, вказаній у функції. Відрізняється від filter тим, що при знаходженні першого задовольняючого елемента повертає його і завершує роботу | Функція, за якою відбувається перевірка | Перший знайдений елемент або undefined | - |

Продовження таблиці 4.1

| | | | | | | |
|---|---|---|---|---|---|---|
| 14 | Array | findIndex | Знайти елемент, що задовольняє умові, вказаній у функції.Повернути його індекс | Функція, за якою відбувається перевірка | Індекс знайденого елемента або -1 | - |
| 15 | DOM-об’єкт | querySelector | Всередині елемента, що вказано знайти DOM-елемент за вказаною DOM-string | DOM-string | Знайдений елемент або null | - |
| 16 | DOM-об’єкт | getElementsByClassName | Отримати колекцію з усіх елементів всередині вказаного елементу DOM, що мають вказаний клас | Рядок з вказаним класом | Колекція елементів DOM | - |
| 17 | Array | getElementById | Отримати елемент усередині вказаного елементу DOM, що має вказаний id | Рядок з вказаним id | DOM-елемент | - |
| 18 | Таргет-об’єкт | addEventListener | Поставити на певний елемент подію, що буде відбуватися за певних умов | 1)Подія 2)функція, яку буде виконано при спрцюванні події | - | - |

Продовження таблиці 4.1

| | | | | | | |
|---|---|---|---|---|---|---|
| 19 | Location | reload | Перезавантаження сторінки | - | - | - |
| 20 | Array | includes | Перевірка на наявність вказаного елемента в масиві | Елемент, який треба перевірити | True \|\| false | - |
| 21 | Array | from | Створення з елемента будь-якого типу масив | Елемент, що треба перетворити | Отриманий масив | - |
| 22 | Math | floor | Округлення числа вниз | Число, яке треба округлити | Округлене число | - |
| 23 | Set | add | Додати елемент до Set() | Елемент який треба додати | - | - |
| 24 | Array | shift | Видалення першого елемента з масива | - | Видалений елемент | - |

У таблиці 4.2 представлено користувацькі методи

Таблиця 4.2 – Користувацькі методи

| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
|---|---|---|---|---|---|
| 1 | App | constructor | Конструктор класу. Запускає роботу додатку шляхом запуску методу цього ж класу - startApp | - | - |
| 2 | App | startApp | Метод, що запускає роботу всього додатку. Створює колоду карт, перемішує її, створює руки карт обох гравців, виводить стартове модальне вікно, у яке передає параметри закриття, а саме: функції, що мають бути викликані: створення початкового поля, заповнення картами рук гравців, створення на полі скарбничок, та свторення екземпляру класу GameProcess, чим запускає його конструктор | - | - |
| 3 | Deck | constructor | Конструктор класу. Створює колоду з 36 впорядкованих карт після чого перемішує їх | Отримує масив карт, що генерується методом freshDeck | - |
| 4 | Deck | freshDeck | З константних масивів VALUES і SUITS створює всі можливі комбінації та записує в масив. | - | Масив з 36 впорядкованих карт |
| 5 | Deck | GetnumberOfCards | Повертає кількість карт в колоді | - | Ціле число, що дорівнює кількості карт в колоді |

Продовження таблиці 4.2

| | | | | | |
|---|---|---|---|---|---|
| 6 | Deck | shuffle | Перемішування колоди. Кожна карта випадковим чином міняється місцями з іншою випадковою картою. | - | - |
| 7 | Deck | takeCard | Взяти карту з колоди, якщо вона не пуста | - | Верхня карта з колоди або undefined |
| 8 | PlayerHand | Constructor | Конструктор класу. Створює пустий масив карт, Пустий масив скарбничок, змінну для підрахунку скарбничок та бере 4 карти з колоди використовуючи метод takeCard з колоди | deck | - |
| 9 | PlayerHand | GetnumberOfCards | Визначає кількість карт в руці | - | Кількість карт в руці, ціле число |
| 10 | PlayerHand | GetcardValues | Дає можливість отримати масив зі значень карт в руці | - | Масив рядків, значень карт |
| 11 | PlayerHand | checkCardValue | Перевіряє чи є карта з вказаним value в руці | Value – рядок зі значенням карти | Повертає знайдену карту чи undefined |
| 12 | PlayerHand | CountNumberOfCard | Визначає кількість карт з вказаним value в руці | Value – рядок зі значенням карти | Ціле число, кількість карт з value в руці |
| 13 | PlayerHand | getCardSuits | Отримати всі масті карти з вказаним value , що є в руці | Value – рядок зі значенням карти | Масив рядків – мастей |
| 14 | PlayerHand | CheckCard | Перевірка на наявність карти з вказаним value і suit в руці | Value – рядок зі значенням карти<br>Suit – рядок зі значенням масті карти | True або false |

Продовження таблиці 4.2

| | | | | | |
|---|---|---|---|---|---|
| 15 | PlayerHand | getCard | Забрати карту з вказаними value і suit у гравця | Value – рядок зі значенням карти<br>Suit – рядок зі значенням масті карти | Вирізана карта, екземпляр класу Card |
| 16 | PlayerHand | addCard | Покласти карту собі в руку | Card | - |
| 17 | PlayerHand | createChest | Видаляє з руки всі карти з вказаним value та додає їх у скарбничку гравця, що викликав функцію | Value – рядок зі значенням карти | - |
| 18 | Interface | createStarter | Створює на ігровому полі колоду карт а також поле з історією гри | Deck – колода карт | - |
| 19 | Interface | renderDeck | Якщо в колоді є карти, покаже скільки їх залишилося. Якщо в колоді карт нема – видалить колоду з поля | Deck | - |
| 20 | Interface | createPlayerHand | Створює руку гравця на полі та заповнює її картами | playerHand – рука гравця | - |
| 21 | Interface | renderPlayerHand | Оновлює карти в руці гравця | playerHand – рука гравця | - |
| 22 | Interface | createEnemyHand | Створює руку комп'ютера на полі та заповнює її картами | playerHand – рука комп’ютера | - |
| 23 | Interface | renderEnemyHand | Оновлює карти в руці комп'ютера | playerHand – рука комп’ютера | - |

Продовження таблиці 4.2

| | | | | | |
|---|---|---|---|---|---|
| 24 | Interface | showStartModal | Створює модальне вікно. При закритті цього вікна викличеться функція onClose | onClose – функція, що має викликатися при закритті | - |
| 25 | Interface | createChest | Створює на полі гри блок зі скарбничками. | Player – рядок що вказує на номер гравця | - |
| 26 | Interface | renderChest | Оновлює кількість скарбничок у вказаного гравця | Player – рядок що вказує на номер гравця playerHand – рука цього гравця | - |
| 27 | Interface | addAction | Додати повідомлення message до ігрової історії | message | - |
| 28 | Interface | createStartModal | Створює DOM-елемент модального вікна | onClose – функція, що має бути викликана при закритті модального вікна | DOM-елемент |
| 29 | Interface | hideModal | Приховати модальне вікно | - | - |
| 30 | Interface | showWinner | Вивести на екран переможця та скарбнички обох гравців | Player – рядок що вказує на номер гравця playerOneHand – рука першого гравця playerTwoHand – рука другого гравця | - |

Продовження таблиці 4.2

| | | | | | |
|---|---|---|---|---|---|
| 31 | GameProcess | Constructor | Конструктор класу. Створює масиви imagineHand & lastUsed. Запам’ятовує deck, руки гравців і запускає гру | playerOneHand, playerTwoHand, deck | - |
| 32 | GameProcess | startGame | Запускає гру, перший ходить гравець | - | - |
| 33 | GameProcess | playerTurn | Метод виводить варіанти для ходу гравця, робить потрібні перевірки а в кінці ходу передає хід супернику | - | - |
| 34 | GameProcess | enemyTurn | Метод відповідає за хід комп’ютера. Робить необзідні перевірки, а в кінці передає хід гравцеві | - | - |
| 35 | GameProcess | getEnemyCards | Відповідає за передачу карт між гравцями. Перевіряє їх наявність, передає, викликає перевірки на скарбнички. А також доповнює логіку комп’ютера через додавання карт до imagineHand | ToHand – рука в яку треба класти карти FromHand – рука звідки брати карти suits- масив мастей value – значення карти player – номер гравця | - |
| 36 | GameProcess | playerTakeCard | Відповідний гравець бере карту, повідомення про це виводиться на екран | playerHand – в цю руку потрібно класти карту. player – номер гравця | Значення взятої карти |

Продовження таблиці 4.2

| | | | | | |
|---|---|---|---|---|---|
| 37 | GameProcess | getCardPriority | Обирає кращий хід для комп'ютера. Детальніше в алгоритмі 3.1 | Value – карту, що перевіряємо зараз | Int – новий пріоритет |
| 38 | GameProcess | checkForChest | Перевіряє карту з вказаним value на скарбничку. Детальніше в алгоритмі 3.4 | Value – значення карти, яке перевіряєтьс я playerHand – рука, в якій перевіряємо на скарбничку player – номер гравця | True \|\| false |
| 39 | GameProcess | addToLastUsed | Тримає в пам'яті три останні карти якими ходив комп'ютер | Value – карта яку треба записати | - |

# 5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАПЕЗПЕЧЕННЯ

Важливою частиною розробки будь якого програмного запезпечення є тестування продукту на кожному етапі створення. Так можна убезпечити коритсувачів від помилок програми, небажаних результатів роботи чи витоку інформації.

## 5.1 План тестування

а) Тестування інтерфейсу

1) Тестування відображення ігрового поля в різних браузерах

б) Тестування коректності початку гри

1) Тестування коректності створення ігрових об'єктів при натисканні на кнопку «Start»
2) Тестування випадковості перетасування колоди карт
3) Тестування випадковості виданих карт обом гравцям

в) Тестування можливості вибору значення карти гравцем

1) Тестування можливості натискання лише на карти, що є у руці гравця
2) Тестування у випадку, коли гравець угадав карту
3) Тестування у випадку, коли гравець не вгадав карту

г) Тестування можливості вибору кількості карт гравцем

1) Тестування можливості вибору 1,2 та 3-х карт
2) Тестування у випадку, коли гравець вгадав кількість карт
3) Тестування у випадку коли гравець не вгадав кількість карт

д) Тестування можливості вибору мастей гравцем

1) Тестування переліку мастей для вибору
2) Тестування кількості мастей для вибору
3) Тестування у випадку, коли гравець вгадав всі карти
4) Тестування у випадку, коли гравець вгадав частину карт
5) Тестування у випадку, коли гравець не вгадав жодної карти

е) Тестування алгоритмів комп'ютера

1) Тестування записування в ImagineHand

1.1) Тестування записування в ImagineHand під час ходу гравця

1.2) Тестування записування в ImagineHand під час ходу комп’ютера

2) Тестування видалення з ImagineHand

2.1) Тестування видалення з ImagineHand при невгадуванні кількості карт (у випадку якщо в колоді більше 14 карт)

2.2) Тестування видалення з ImagineHand при зборі скарбнички гравцем

2.3) Тестування видалення з ImagineHand при зборі скарбнички комп’ютером

3) Тестування додавання до lastUsed

3.1) Тестування додавання до lastUsed при виборі карти для ходу

3.2) Тестування додавання до lastUsed, якщо в ньому вже є 3 карти

4) Тестування видалення з lastUsed

4.1) Тестування видалення з lastUsed коли один з гравців зібрав скарбничку

Продовження Плану тестування

5) Тестування коректності вибору кращої карти

5.1) Тестування правильного підрахунку кількості карти в руці

5.2) Тестування правильності перевірки на наявність карти в lastUsed

5.3) Тестування правильного підрахунку кількості копій карти в imagineHand

5.4) Тестування правильності підрахунку кількості копій карт з точною мастю в imagineHand

5.5) Тестування вибору карти для ходу, коли в колоді не залишилося карт

6) Тестування коректності вгадування кількості карт

6.1) Тестування кількості карти в загальному випадку

6.2) Тестування кількості карт, коли в колоді не залишилося карт

7) Тестування коректності вгадування мастей

7.1) Тестування відсутності у переліку взгадуваних мастей тих мастей, що є в руці у комп’ютера

7.2) Тестування коректності вибору мастей коли в колоді не залишилося карт

7.3) Тестування випадковості вибору, коли потрбіно вгадати

7.4) Тестування відсутності тих мастей, що зазначені в isNot

ж) Тестування своєчасного взяття карт

1) Тестування взяття карти з колоди, коли один з гравців не вгадує карти
2) Тестування взяття карти з колоди, коли один з гравців не вгадує кількість карти
3) Тестування взяття карти з колоди, коли один з гравців не вгадує хочаб одну масть
4) Тестування взяття карт з колоди, коли в одного з гравців не вистачає карт

з) Тестування на своєчасну перевірку на наявність скарбничок

1) Тестування на перевірку на наявність скарбничок, коли один з гравців бере карту
2) Тестування на перевірку на наявність скарбничок, коли один з гравців угадує та забирає карти у іншого

и) Тестування на закінчення гри

1) Тестування на закінчення гри при відсутності карт в колоді та в руках гравців

к) Тестування на коректне визначення переможця

1) Тестування на визначення переможця, коли переможець гравець
2) Тестування на визначення переможця, коли переможець комп’ютер
3) Тестування на коректне виведення скарбничок обох гравців

л) Тестування подій що відбуваються при натисканні на кнопку «Play again»

1) Тестування на оновлення сторінки

### 5.2 Приклади тестування

Важливою частиною будь-якого веб застосунку є коректність його відображення на всіх браузерах. Тож спочатку перевіримо саме це. (Таблиця 5.1).

Таблиця 5.1 – Тестування кросбраузерності

| | |
|---|---|
| Мета тесту | Перевірити кросбраузерність застосунку |
| Початковий стан програми | Відкрите вікно веб-сайту |

Продовження таблиці 5.1

| Вхідні дані | - |
|---|---|
| Схема проведення тесту | 1. Відкриваємо сайт у різних браузерах<br>2. Знаходимо недоліки та неточності у відображенні, якщо такі знайдуться. |
| Очікуваний результат | Однакове відображення сайту в Chrome, Mozilla та Opera |
| Стан програми після проведення випробувань | Програма коректно відображається у всіх перелічених браузерах. |

Далі згідно нашому плану потрібно протестувати коректність початку гри. Йдеться про події, що відбуваються після натискання на кнопку «Start». (Таблиця 5.2)

Таблиця 5.2 – Тестування коректності початку гри

| Мета тесту | Перевірити коректність початку гри |
|---|---|
| Початковий стан програми | Відкрите вікно веб-сайту |
| Вхідні дані | - |
| Схема проведення тесту | 1. У відкритому додатку натискаємо кнопку «Start»<br>2. Перевіряємо правильність створення та відображення всіх необхідних об’єктів для гри |
| Очікуваний результат | На ігровому полі є колода карт, випадково згенерована, дві руки гравців, по 4 карти в кожній, блоки зі скарбничками і блок з історії гри. |
| Стан програми після проведення випробувань | 1.Колода карт випадково згенерована<br>2.Колода карт відображається, а також видно кількість карт, що залишилось<br>3.У гравців в руках є по 4 карти, що вони взяли з колоди<br>4.Відображення карт в руці є коректним<br>5.Наявні 2 блоки зі скарбничками<br>6.Блок з історією гри присутній |

Переходимо до тестування ігрового процесу, а саме коректності можливості вибору карти гравцем. (Таблиця 5.3)

Таблиця 5.3 – Тестування можливості вибору карти гравцем

| | |
|---|---|
| Мета тесту | Перевірити коректність вибору карти гравцем |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. |
| Вхідні дані | - |
| Схема проведення тесту | 1. Гравець отримує карти на початку гри<br>2. У блоці дій(зліва зверху) відкрити список карт, що доступні для вибору<br>3. Перевірити карти, що можна обрати |
| Очікуваний результат | Відкривши список карт для ходу, відображаються всі карти від 6 до А, але обрати можна лише ті, що є в руці гравця. Після вибору карти більше не можна зробити вибір |
| Стан програми після проведення випробувань | 1. При натисканні відкривається список з усіма картами<br>2. При наведенні на карти, яких нема в руці – нічого не відбувається<br>3. При наведенні на карти, які є в руці – карти підсвічуються синім кольором<br>4. При виборі карти більше не омжна зробити вибір |

Після зробленого вибору протестуємо, що відбувається коли гравець вгадує, а що – коли не вгадує. (Таблиці 5.4-5.5)

Таблиця 5.4 – Тестування можливості вгадування карти гравцем

| | |
|---|---|
| Мета тесту | Перевірити роботу програми, коли гравець вгадує карту |

Продовження таблиці 5.4

| | |
|---|---|
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав |
| Вхідні дані | Карта для ходу |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту<br>2. У полі дії обрати карту для ходу таку, що комп'ютер її має<br>3. Перевірити дії програми після вибору карти |
| Очікуваний результат | Після вибору карти, що є у комп'ютера з'являється можливість вибору кількості карт |
| Стан програми після проведення випробувань | 1. Після вибору правильної карти програма пропонує вибір кількості карт |

Таблиця 5.5 – Тестування можливості невгадування карти гравцем

| | |
|---|---|
| Мета тесту | Перевірити роботу програми, коли гравець не вгадує карту |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і не вгадав |
| Вхідні дані | Карта для ходу |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно не вгадати карту<br>2. У полі дії обрати карту для ходу таку, що комп'ютер її не має<br>Перевірити дії програми після вибору карти |
| Очікуваний результат | 1. Після вибору карти, якої нема у комп'ютера гравець бере карту та передає хід супернику |
| Стан програми після проведення випробувань | 1. Після вибору карти гравець узяв карту, про це було повідомлено у блоці історії гри<br>2. Хід було передано супернику |

Далі, згідно з планом потрібно протестувати можливість вибору кількості карт: як це відбувається, та що стається при правильному та неправильному виборі. (Таблиці 5.6-5.8)

Таблиця 5.6 – Тестування можливості вибору кількості карт

| | |
|---|---|
| Мета тесту | Перевірити коректність роботи вибору кількості карт |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав. Тепер гравець має вгадати кількість карт |
| Вхідні дані | - |
| Схема проведення тесту | 1. Коли з'являється можливість вибору кількості карт натиснути на кнопку Перевірити коректність роботи програми |
| Очікуваний результат | 2. При натисканні на кнопку з'являється список з вибором: «1», «2» або «3» карти. |
| Стан програми після проведення випробувань | 3. При натисканні на кнопку з'являється список з вибором: «1», «2» або «3» карти. |

Таблиця 5.7 – Тестування можливості вгадування кількості карт

| | |
|---|---|
| Мета тесту | Перевірити роботу програми, коли гравець вгадав кількість карт |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав. Гравець зробив вибір кількості карт і вгадав. |
| Вхідні дані | Вибрана кількість карт |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту і кількість карт<br>2. Перевірити роботу програми після правильного вгадування кількості карт |

Продовження таблиці 5.7

| | |
|---|---|
| Очікуваний результат | Після вгадування з'являється можливість вгадати масті |
| Стан програми після проведення випробувань | При правильному виборі з’являється можливість вибору мастей для вгадування |

Таблиця 5.8 – Тестування можливості невгадування кількості карт

| | |
|---|---|
| Мета тесту | Перевірити роботу програми, коли гравець не вгадав кількість карт |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав. Гравець зробив вибір кількості карт і не вгадав. |
| Вхідні дані | Вибрана кількість карт |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно не вгадати карту та кількість карт<br>2. Перевірити роботу рпограми у випадку, коли гравець не вгадує кількість карт комп'ютера |
| Очікуваний результат | Після неправильного вибору кількості карт, гравець бере карту та передає хід |
| Стан програми після проведення випробувань | Після неправильного вибору кількості карт, гравець бере карту та передає хід |

Продовжимо тестування коректності роботи ходу гравця. Нехай гравець вгадав значення та кількість карт. Необхідно протестувати вгадування мастей. (Таблиці 5.9-5.13)

Таблиця 5.9 – Тестування можливості вибору мастей.

| | |
|---|---|
| Мета тесту | Перевірити масті, що доступні для вибору гравцю |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав. Гравець зробив вибір кількості карт і вгадав. Гравець може спробувати вгадати масті карт суперника |
| Вхідні дані | - |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту та кількість карт<br>2. Перевірити роботу програми коли гравець відкриває список доступних мастей для вибору |
| Очікуваний результат | Для вибору доступні лише масті, яких нема в руці у гравця |
| Стан програми після проведення випробувань | Для вибору доступні лише масті, яких нема в руці у гравця |

Таблиця 5.10 – Тестування кількості мастей, що гравець може обрати

| | |
|---|---|
| Мета тесту | Перевірити кількість мастей, що може обирати гравець |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав. Гравець зробив вибір кількості карт і вгадав. Гравець може спробувати вгадати масті карт суперника |
| Вхідні дані | Обрана кількість карт суперника |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту та кількість карт<br>2. Перевірити роботу програми для випадку вибору 1, 2 та 3 карт |

Продовження таблиці 5.10

| | |
|---|---|
| Очікуваний результат | 1. Якщо гравець обрав 1 карту – після вибору однієї масті, вибір має бути заблоковано<br>2. Якщо гравець обрав 2 карти – після вибору двох мастей, вибір має бути заблоковано<br>3. Якщо гравець обрав 3 карти – після вибору трьох мастей, вибір має бути заблоковано |
| Стан програми після проведення випробувань | Коли кількість вибраних мастей досягає кількості обраних карт – вибір стає заблокованим |

Таблиця 5.11 – Тестування випадку, коли гравець не вгадав жодної масті

| | |
|---|---|
| Мета тесту | Перевірити роботу програми, коли гравець не вгадує жодної з мастей |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав. Гравець зробив вибір кількості карт і вгадав. Гравець спробував вгадати масті карт суперника, але не вгадав жодної |
| Вхідні дані | Введені масті |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту та кількість карт<br>2. Спеціально не вгадати жодну з мастей<br>3. Перевірити роботу програми |
| Очікуваний результат | Гравець бере одну карту і передає хід супернику |
| Стан програми після проведення випробувань | Гравець бере одну карту і передає хід супернику |

Таблиця 5.12 – Тестування випадку, коли гравець вгадав частину мастей

| | |
|---|---|
| Мета тесту | Перевірити роботу програми, коли гравець частину мастей суперника |

Продовження таблиці 5.12

| | |
|---|---|
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав. Гравець зробив вибір кількості карт і вгадав. Гравець спробував вгадати масті карт суперника, але вгадав лише частину |
| Вхідні дані | Введені масті |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту та кількість карт<br>2. Спеціально вгадати частину мастей<br>3. Перевірити роботу програми |
| Очікуваний результат | Гравець бере одну карту, забирає вгадані карти в суперника і передає хід супернику |
| Стан програми після проведення випробувань | Гравець бере одну карту, забирає вгадані карти в суперника і передає хід супернику |

Таблиця 5.13 – Тестування випадку, коли гравець вгадав всі масті

| | |
|---|---|
| Мета тесту | Перевірити роботу програми, коли гравець всі масті суперника |
| Початковий стан програми | Гра почалась. Інтерфейс поля перед гравцем, і він робить хід. Гравець зробив вибір карти для ходу і вгадав. Гравець зробив вибір кількості карт і вгадав. Гравець спробував і вгадав масті карт суперника |
| Вхідні дані | Введені масті |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту та кількість карт<br>2. Вгадати всі масті<br>3. Перевірити роботу програми |
| Очікуваний результат | Гравець забирає вгадані карти в суперника і передає хід супернику |
| Стан програми після проведення випробувань | Гравець забирає вгадані карти в суперника і передає хід супернику |

Тестування ходу гравця завершено. Далі по плану – тестування алгоритму гри комп’ютерного опонента. Спочатку перевірими записування в ImagineHand. (Таблиці 5.14-5.18)

Таблиця 5.14 – Тестування додавання до ImagineHand

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо imagineHand, коли гравець не вгадує карту |
| Початковий стан програми | Хід гравця |
| Вхідні дані | |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно не вгадати карту<br>2. Перевірити викоання алгоритму |
| Очікуваний результат | До imagineHand записано value, accurate= false |
| Стан програми після проведення випробувань | До imagineHand записано value, accurate=false |

Таблиця 5.15 – Тестування додавання до ImagineHand

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо imagineHand, коли гравець не вгадує кількість карт |
| Початковий стан програми | Хід гравця |
| Вхідні дані | |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту, але не вгадати кількість карт<br>2. Перевірити виконання алгоритму |
| Очікуваний результат | До imagineHand записано value, accurate= false |
| Стан програми після проведення випробувань | До imagineHand записано value, accurate= false |

Таблиця 5.16 – Тестування додавання до ImagineHand

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо imagineHand, коли гравець не вгадує жодної масті |
| Початковий стан програми | Хід гравця |
| Вхідні дані | Названі масті |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту, але не вгадати жодної масті<br>2. Перевірити виконання алгоритму |
| Очікуваний результат | До imagineHand записано value, accurate= false, isNot=[Названі масті] |
| Стан програми після проведення випробувань | До imagineHand записано value, accurate= false, isNot=[Названі масті] |

Таблиця 5.17 – Тестування додавання до ImagineHand

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо imagineHand, коли гравець вгадує частину мастей |
| Початковий стан програми | Хід гравця |
| Вхідні дані | Названі масті |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту, але вгадати лише частину мастей<br>2. Перевірити виконання алгоритму |
| Очікуваний результат | До imagineHand записано value, accurate= false, isNot=[Названі невгадані масті], а також value, accurate=true,isNot=[], Вгадана масть. |
| Стан програми після проведення випробувань | До imagineHand записано value, accurate= false, isNot=[Названі невгадані масті], а також value, accurate=true,isNot=[], Вгадана масть. |

Таблиця 5.18 – Тестування додавання до ImagineHand

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо imagineHand, коли гравець вгадує всі масті |
| Початковий стан програми | Хід гравця |
| Вхідні дані | Названі масті |
| Схема проведення тесту | 1. Вивести руку комп'ютера до консолі, аби точно вгадати карту, і вгадати всі масті<br>2. Перевірити виконання алгоритму |
| Очікуваний результат | До imagineHand записано value, accurate= false, а також value, accurate=true,isNot=[], Вгадана масть. |
| Стан програми після проведення випробувань | До imagineHand записано value, accurate= false, isNot=[Названі невгадані масті], а також value, accurate=true,isNot=[], Вгадана масть. |

Запис до imagineHand протестовано. По плану переходимо до тестування видалення з imagineHand. (Таблиці 5.19-5.20)

Таблиця 5.19 – Тестування видалення з ImagineHand

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо видалення з imagineHand |
| Початковий стан програми | Хід комп’ютера. Комп’ютер угадав карту гравця, але не вгадав кількість. В колоді більше 14 карт |
| Вхідні дані | Значення невгаданої карти |
| Схема проведення тесту | 1. Дочекатися коли ком'ютер допустить помилку в кількості карт за умови, що в колоді більше 14 карт<br>2. Перевірити видалення з imagineHand |
| Очікуваний результат | Елемент видалено з imagineHand |
| Стан програми після проведення випробувань | Елемент видалено з imagineHand |

Таблиця 5.20 – Тестування видалення з ImagineHand

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо видалення з imagineHand |
| Початковий стан програми | Один з гравців зібрав скарбничку |
| Вхідні дані | Значення карти, яку зібрали |
| Схема проведення тесту | 1. Зібрати скарбничку, або дозволити це зробити комп'ютеру<br>2. Перевірити видалення |
| Очікуваний результат | Елементи видалено з imagineHand |
| Стан програми після проведення випробувань | Елементи видалено з imagineHand |

Аналогічним чином протестуємо роботу з lastUsed. Додавання та видалення. (Таблиці 5.21-5.24)

Таблиця 5.21 – Тестування додавання до lastUsed

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо додавання до lastUsed, якщо в ньому нема трьох карт |
| Початковий стан програми | Комп’ютер спитав деяку карту |
| Вхідні дані | Значення карти, якою ходить комп’ютер |
| Схема проведення тесту | Після ходу комп'ютера перевірити зміни в lastUsed |
| Очікуваний результат | Елемент додано до lastUsed |
| Стан програми після проведення випробувань | Елемент додано до lastUsed |

Тестування додавання до lastUsed

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо додавання до lastUsed, якщо в ньому є 3 карти |
| Початковий стан програми | Комп’ютер спитав деяку карту |
| Вхідні дані | Значення карти, якою ходить комп’ютер |
| Схема проведення тесту | Після ходу комп'ютера перевірити зміни в lastUsed |
| Очікуваний результат | Перший елемент було видалено, а як останній додано новий |

Продовження таблиці 5.22

| | |
|---|---|
| Стан програми після проведення випробувань | Перший елемент було видалено, а як останній додано новий |

Таблиця 5.22 – Тестування видалення з lastUsed

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо видалення з lastUsed |
| Початковий стан програми | Один з гравців зібрав скарбничку |
| Вхідні дані | Значення карти, яку зібрано |
| Схема проведення тесту | Зібрати скарбничку й перевірити зміни в lastUsed |
| Очікуваний результат | Всі значення карти, що зібрано в скарбничку видалено |
| Стан програми після проведення випробувань | Всі значення карти, що зібрано в скарбничку видалено |

Таблиця 5.23 – Тестування додавання до lastUsed

| | |
|---|---|
| Мета тесту | Перевірити виконання програмою зазначених алгоритмів щодо додавання до lastUsed, якщо в ньому є 3 карти |
| Початковий стан програми | Комп’ютер спитав деяку карту |
| Вхідні дані | Значення карти, якою ходить комп’ютер |
| Схема проведення тесту | Після ходу комп'ютера перевірити зміни в lastUsed |
| Очікуваний результат | Перший елемент було видалено, а як останній додано новий |
| Стан програми після проведення випробувань | Перший елемент було видалено, а як останній додано новий |

Згідно плану переходимо до найважливішої частини курсової роботи : алгоритм комп’ютера. Спочатку перевірими чи коректно він вибирає карту для ходу в різних ситуаціях.

При виборі карти комп’ютер враховує карти в imagineHand, lastUsed, карти що знаходяться в руці а також наявність в колоді карт. (Таблиці 5.25-5.29)

Таблиця 5.24 – Тестування вибору кращого ходу

| | |
|---|---|
| Мета тесту | Перевірити правильність підрахунку кількості карт певного значення в руці комп'ютером |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Значення карти, яку треба порахувати |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи правильно комп’ютер рахує карти в своїй руці |
| Очікуваний результат | Відповідній змінній присвоєно правильне значення |
| Стан програми після проведення випробувань | Відповідній змінній присвоєно правильне значення |

Таблиця 5.25 – Тестування вибору кращого ходу

| | |
|---|---|
| Мета тесту | Перевірити правильність перевірки на наявність певної карти в lastUsed |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Значення карти, яку треба перевірити |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи правильно комп’ютер визначає наявність карти в lastUsed |
| Очікуваний результат | Комп’ютер правильно визначає чи є карта в lastUsed |
| Стан програми після проведення випробувань | Комп’ютер правильно визначає чи є карта в lastUsed |

Таблиця 5.26 – Тестування вибору кращого ходу

| | |
|---|---|
| Мета тесту | Перевірити правильність підрахунку кількості потрібних карт в imagineHand |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Значення карти, яку треба порахувати |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи правильно комп’ютер визначає кількість карт в imagineHand |

Продовження таблиці 5.27

| | |
|---|---|
| Очікуваний результат | Комп’ютер правильно визначає кількість карт в imagineHand |
| Стан програми після проведення випробувань | Комп’ютер правильно визначає кількість карт в imagineHand |

Таблиця 5.27 – Тестування вибору кращого ходу

| | |
|---|---|
| Мета тесту | Перевірити правильність підрахунку кількості потрібних карт з accurate==true в imagineHand |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Значення карти, яку треба порахувати |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи правильно комп’ютер визначає кількість точних карт в imagineHand |
| Очікуваний результат | Комп’ютер правильно визначає кількість точних карт в imagineHand |
| Стан програми після проведення випробувань | Комп’ютер правильно визначає кількість точних карт в imagineHand |

Таблиця 5.28 – Тестування вибору кращого ходу

| | |
|---|---|
| Мета тесту | Перевірити правильність вибору карти, коли в колоді не залишилось карт |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | - |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи правильно комп’ютер обирає карту для ходу, коли в колоді 0 карт |
| Очікуваний результат | Комп’ютер обирає першу карту в руці |
| Стан програми після проведення випробувань | Комп’ютер обирає першу карту в руці |

Тестування вибору карти завершено. Перевіримо коректність вибору кількості карт. (Таблиці 5.30-5.31)

Таблиця 5.29 – Тестування вибору кращої кількості карт для ходу

| | |
|---|---|
| Мета тесту | Перевірити правильність вибору кількості карт для загального випадку |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Обрана карта |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи правильно комп’ютер обирає кількість карт для вгадування |
| Очікуваний результат | Комп’ютер обирає для вгадування число цієї карти в imagineHand |
| Стан програми після проведення випробувань | Комп’ютер обирає для вгадування число цієї карти в imagineHand |

Таблиця 5.30 – Тестування вибору кращої кількості карт для ходу

| | |
|---|---|
| Мета тесту | Перевірити правильність вибору кількості карт, коли в колоді нема карт |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Обрана карта |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи правильно комп’ютер обирає кількість карт для вгадування |
| Очікуваний результат | Комп’ютер обирає для вгадування число (4-кількість карт в руці) |
| Стан програми після проведення випробувань | Комп’ютер обирає для вгадування число (4-кількість карт в руці) |

Залишилося перевірите те, як комп’ютер обирає масті для ходу. (Таблиці 5.32-5.35)

Таблиця 5.31 – Тестування вибору кращих мастей для ходу

| | |
|---|---|
| Мета тесту | Перевірити те, чи не вгадує комп'ютер ті масті, що є в нього в руці |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Обрана карта, кількість карт |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи обирає комп’ютер ті масті, що має в руці |
| Очікуваний результат | Комп’ютер ніколи не обирає ті масті, що єв нього в руці |
| Стан програми після проведення випробувань | Комп’ютер ніколи не обирає ті масті, що єв нього в руці |

Таблиця 5.32 – Тестування вибору кращих мастей для ходу

| | |
|---|---|
| Мета тесту | Перевірити правильність вибору мастей, коли в колоді нема карт |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Обрана карта, кількість карт |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи правильно комп’ютер обирає масті, коли в колоді нема карт |
| Очікуваний результат | Комп’ютер обирає для вгадування масті, яких нема в руці |
| Стан програми після проведення випробувань | Комп’ютер обирає для вгадування масті, яких нема в руці |

Таблиця 5.33 – Тестування вибору кращих мастей для ходу

| | |
|---|---|
| Мета тесту | Перевірити випадковість вибору масті, коли потрібно насправді вгадати |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Обрана карта, кількість карт |
| Схема проведення тесту | За декілька ходів, де комп'ютеру потрібно обирати переконаємося, що він обирає різні масті |
| Очікуваний результат | Комп’ютер обирає випадкові масті, коли це потрібно |

Продовження таблиці 5.34

| | |
|---|---|
| Стан програми після проведення випробувань | Комп’ютер обирає випадкові масті, коли це потрібно |

Таблиця 5.34 – Тестування вибору кращих мастей для ходу

| | |
|---|---|
| Мета тесту | Перевірити те, чи не вгадує комп'ютер масті, що пристуні в isNot |
| Початковий стан програми | Хід комп’ютера. |
| Вхідні дані | Обрана карта, кількість карт |
| Схема проведення тесту | За допомогою Debugger в DevTools перевіримо, чи обирає комп’ютер масті з isNot |
| Очікуваний результат | Комп’ютер ніколи не обирає масті з isNot |
| Стан програми після проведення випробувань | Комп’ютер ніколи не обирає масті з isNot |

Важливим елементом гри є взяття карти з колоди у правильні моменти часу: коли не вгадуєш карту, кількість карт чи масть, коли в руці менше 4-х карт. Тому тестування для цього теж є необхідним. (Таблиці 5.36-5.37)

Таблиця 5.35 – Тестування своєчасного взяття карти з колоди

| | |
|---|---|
| Мета тесту | Перевірка взяття карт гравцем |
| Початковий стан програми | Хід гравця |
| Вхідні дані | - |
| Схема проведення тесту | 1) Перевірити чи бере гравець карту, коли не вгадує карту, кількість чи масть<br>2)Перевірити чи бере гравець карти, коли в нього їх менше 4-х |
| Очікуваний результат | Гравець бере карти в усіх зазначених випадках |
| Стан програми після проведення випробувань | Гравець бере карти в усіх зазначених випадках |

Таблиця 5.36 – Тестування своєчасного взяття карти з колоди

| | |
|---|---|
| Мета тесту | Перевірка взяття карт комп'ютером |
| Початковий стан програми | Хід гравця |
| Вхідні дані | - |

Продовження таблиці 5.36

| | |
|---|---|
| Схема проведення тесту | 1) Перевірити чи бере комп'ютер карту, коли не вгадує карту, кількість чи масть<br>2)Перевірити чи бере комп'ютер карти, коли в нього їх менше 4-х |
| Очікуваний результат | Комп'ютер бере карти в усіх зазначених випадках |
| Стан програми після проведення випробувань | Комп'ютер бере карти в усіх зазначених випадках |

Скарбнички – основна одиниця гри, адже від них залежить перемога чи програш гравця. Потрібно вчасно перевіряти руки гравців та збирати скарбнички. Згідно плану це і є наша наступна область тестування. (Таблиці 5.38-5.39)

Таблиця 5.37 – Тестування своєчасної перевірки скарбничок

| | |
|---|---|
| Мета тесту | Тестування на перевірку скарбничок при зміні руки гравця |
| Початковий стан програми | Хід гравця |
| Вхідні дані | - |
| Схема проведення тесту | 1)Перевірити чи відбувається перевірка на скарбничку, коли гравець бере карту з колоди<br>2) Перевірити чи відбувається перевіка на скарбничку, коли гравець угадає якусь карту у суперника. |
| Очікуваний результат | Перевірки на скарбничку відбуваються в усіх зазначених випадках |
| Стан програми після проведення випробувань | Перевірки на скарбничку відбуваються в усіх зазначених випадках |

Таблиця 5.38 – Тестування своєчасної перевірки скарбничок

| | |
|---|---|
| Мета тесту | Тестування на перевірку скарбничок при зміні руки комп'ютера |
| Початковий стан програми | Хід комп'ютера |
| Вхідні дані | - |

Продовження таблиці 5.38

| | |
|---|---|
| Схема проведення тесту | 1)Перевірити чи відбувається перевірка на скарбничку, коли комп'ютер бере карту з колоди<br>2) Перевірити чи відбувається перевіка на скарбничку, коли комп'ютер угадає якусь карту у суперника. |
| Очікуваний результат | Перевірки на скарбничку відбуваються в усіх зазначених випадках |
| Стан програми після проведення випробувань | Перевірки на скарбничку відбуваються в усіх зазначених випадках |

Наступним за планом є тестування закінчення гри. Правильне закінчення гри: коли нема карт в колоді та в руках гравців. (Таблиця 5.40)

Таблиця 5.39 – Тестування своєчасного закінчення гри

| Мета тесту | Перевірка вчасного завершення гри |
|---|---|
| Початковий стан програми | Пуста колода, пусті руи гравців |
| Вхідні дані | - |
| Схема проведення тесту | Спустошити колоду карт та зібрати всі можливі в грі скрабнички, чим спустошити руки гравців |
| Очікуваний результат | Гру завершено |
| Стан програми після проведення випробувань | Гру завершено |

Далі за планом є тестування визначення переможця гри. Має перемагати той, у кого більше скарбничок. (Таблиця 5.41)

Таблиця 5.40 – Тестування правильного вибору переможця

| Мета тесту | Перевірка коректного визначення переможця |
|---|---|
| Початковий стан програми | Гру завершено |
| Вхідні дані | - |

Продовження таблиці 5.40

| | |
|---|---|
| Схема проведення тесту | За допомогою Debugger в DevTools порівняємо кількість скарбничок обох гравців |
| Очікуваний результат | Перемагає гравець з більшою кількістю гравців |
| Стан програми після проведення випробувань | Перемагає гравець з більшою кількістю гравців |

Тестування кнопки «Play again». (Таблиця 5.42)

Таблиця 5.41 – Тестування роботи кнопки

| | |
|---|---|
| Мета тесту | Перевірка роботи кнопки «Play again» |
| Початковий стан програми | Гру завершено |
| Вхідні дані | - |
| Схема проведення тесту | Завершити гру, натиснути на кнопку |
| Очікуваний результат | Оновлюється сторінка |
| Стан програми після проведення випробувань | Оновлюється сторінка |

# 6 ІНСТРУКЦІЯ КОРИСТУВАЧА

## 6.1 Робота з програмою

### 6.1.1 Початковий  екран

При запуску програми на екрані є модальне вікно (Рисунок 6.1)

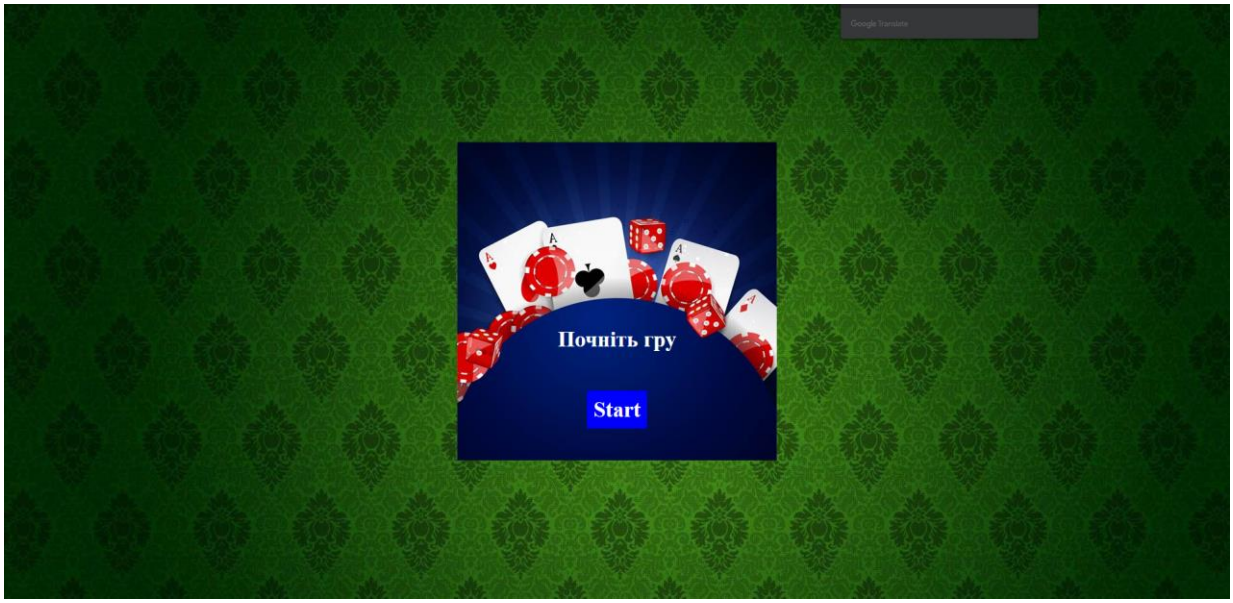


Рисунок 6.1 – Початковий екран гри

### 6.1.2 Екран з функціоналом

Весь функціонал гри користувач побачить після натискання на кнопку «Start» (Рисунок 6.2)

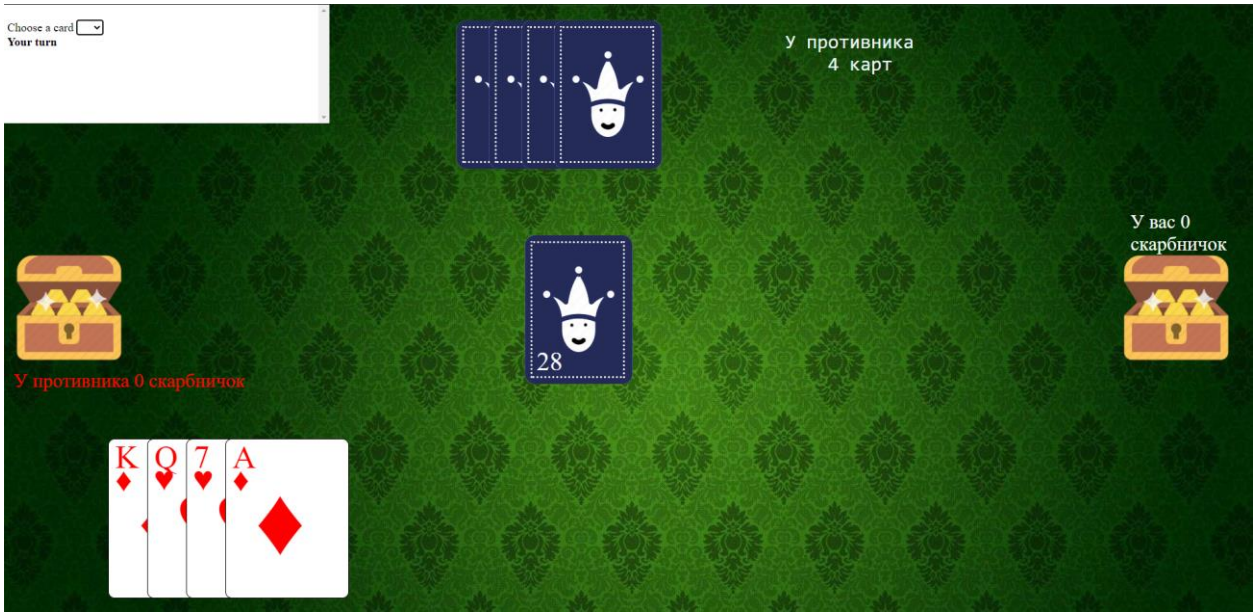


Рисунок 6.1 – Інтерфейс гри

Інтерфейс можна легко поділити на міні-блоки. Пройдемося по кожному з них.

Колода карт (Рисунок 6.3). Цей блок вказує на наявність в колоді карт, а також відображає кількість карт в колоді


Рисунок 6.2 – Колода карт

Скарбнички (Рисунки 6.3 та 6.4). Ці блоки відповідають за відображення кількості скарбничок у кожного з гравців на цей момент


Рисунок 6.3 – Блок скарбничок гравця


Рисунок 6.4 – Блок скрабничок комп'ютера

Рука гравця (Рисунок 6.6). Цей блок містить в собі карти гравця. Стиль карт було взято з стандартної колоди карт на 36.


Рисунок 6.5 – Рука гравця

Рука комп'ютера (Рисунок 6.6). Цей блок містить в собі карти суперника. Він може відображати візуально до семи карт. Про кількість карт говорить надпис біля руки.


Рисунок 6.6 – Рука комп'ютера

Блок з історією та функціоналом вибору карт (Рисунок 6.8). Цей блок відповідає за взаємодію з користувачем, а також дає йому змогу дізнатися детальніше як походив комп'ютер.


Рисунок 6.7 – Блок історії та функціоналу

### 6.1.3 Екран кінця гри

По завершенню гри виводить вікно з оголошення переможця, а також з скарбничками, що зібрали обоє гравців. (Рисунок 6.9)


Рисунок 6.1 – Екран переможця

## 6.2 Системні вимоги

| | Мінімальні | Рекомендовані |
|---|---|---|
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) | Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) |
| Процесор | Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz | Intel® Pentium® D або AMD Athlon™ 64 X2 |
| Оперативна пам'ять | 256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/ Windows 8/Windows 10) | 2 GB RAM |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 1367x600 | 1367x768 або краще |
| Прилади введення | Комп’ютерна миша | |
| Додаткове програмне забезпечення | Браузер | |

# ВИСНОВОК

У цій роботі було визначено причини її створення і було реалізовано лінійний алгоритм комп'ютера для гри. За час підготовки курсової роботи я здобув багато корисних навичок в сфері створення алгоритму, його обробки.

Перш за все, було сформульовано причину виникнення такої проблеми, а також чіткі завдання, що потрібно реалізувати. Це дало змогу зручно контролювати процес розробки ПЗ.

По-друге, було проаналізовано всі можливі ігрові ситуації, а також знайдено найкращі рішення в кожній з них. Відповідно до отриманої інформації було сформульовано та записано псевдокодом алгоритми роботи комп'ютера.

По-третє, було проведено предметний аналіз області, побудовано UML-діаграми класів, що були реалізовані в програмному коді. Всі станлартні та користувацькі методи було детально описано в таблицях.

По-четверте, кожну операцію, що має бути здійснена програмою було протестовано згідно з планом тестування. Виконання алгоритмів комп’ютера було також протестовано.

В додаток отримав досвід написання ПЗ відповідно до методології Об’єктно-Орієнтованого Програмування з інтерфейсом. Важливим у роботі також було створення інструкції користувача, що має допомагати правильному використанню програми і запобігати виникненню помилок.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Набір правил «Скарбнички» [Електронний ресурс] - https://durbetsel.ru/2_sunduchki.htm
2. «Скарбнички» [Електронний ресурс] - https://uk.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BD%D1%8C%D0%BA%D0%B8

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
автоматизованих систем обробки інформації та управління

Затвердив
Керівник Головченко М. М.
«10» березня 2021 р.

Виконавець:
Студент Василенко П. О.
«10» березня 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання курсової роботи
на тему: «Скарбнички»
з дисципліни:
«Основи програмування-2. Модульне програмування»

Київ 2021

1. *Мета*: Метою курсової роботи є розробка програмного забезпечення, що передбачає можливість гри в «Скарбнички» проти комп'ютера.
2. *Дата початку роботи*: «02»березня 2021 р.
3. *Дата закінчення роботи*: «02»червня 2021 р.
4. *Вимоги до програмного забезпечення*.
   1) Функціональні вимоги:
      - Можливість грати з комп'ютерним опонентом
      - Можливість випадкової генерації колоди карт
      - Можливість дотримування правил гри:
      - 1. Можливість вгадати карти суперника, й забрати їх собі
      - 2.Можливість вгадати лише частину набору карт суперника
      - 3.Можливість не вгадати жодної з карт суперника
      - 4.Можливість збирати 4 однакові за значенням карти в «Скарбницю»
      - 5.Можливість перемоги/поразки відповідно до кількості скарбничок гравців

   2) Нефункціональні вимоги:
      - Використання стандарту ES6 для Javascript
      - Використання системи контролю версій Git
      - Наявність зрозумілого користувачеві інтерфейсу
      - Кросбраузерність (Google Chrome, Opera, Mozilla Firefox)
      - 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 - 95 - Розробка технічної документації.

5. *Стадії та етапи розробки*:
   1) Об'єктно-орієнтований аналіз предметної області задачі (до 20.03.2021 р.)
   2) Об'єктно-орієнтоване проектування архітектури програмної системи (до 02.04.2021р.)
   3) Розробка програмного забезпечення (до 24.04.2021р.)
   4) Тестування розробленої програми (до 01.05.2021р.)
   5) Розробка пояснювальної записки (до 15.05.2021 р.).
   6) Захист курсової роботи (до 04.06.2021 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 "Додатку для створення мелодій"*

---

(Найменування програми (документа))

*HDD*

(Вид носія даних)

*36 арк,356 Кб*

(Обсяг програми (документа), арк., Кб)

*студента групи ІП-02, І курсу*

*Василенка П. О.*

Київ 2021

Index.html

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Сундучки</title>
    <link rel="stylesheet" type="text/css" href="css/style.css">
    <link rel="stylesheet" type="text/css" href="css/normalize.css">
</head>

<body>
    <h1 id="not-PC">Only PC version is availible</h1>
    <div id="root">
    </div>



    <script type="module" src="index.js"></script>

</body>

</html>
```

Style.css

```css
*,
*::after,
*::before {
    box-sizing: border-box;
}

:root {
    --redSuit: red;
    --blackSuit: black;
}
#root {
    height: 100vh;
}
body {
    user-select: none;
    overflow-x: hidden;
    overflow-y: hidden;
    width: 100vw;
    height: 100vh;
    margin: 0;
    background-image: url(../images/bg.jpg);
}

.modalStart {
    height: 100vh;
    display: flex;
    margin-top: 0vh;
    justify-content: center;
    align-items: center;
    position: relative;
}

.modalStart img {
    position: fixed;
    z-index: -1;
}
.modalStart #start {
    color: white;
    font-size: 34px;
    font-weight: bold;
    padding-top: 120px;
}

.modalStart button {
    background: blue;
    position: absolute;
    font-size: 34px;
    font-weight: bold;
    color: white;
    outline: none;
```

```
    top: 65%;
    border: none;
    padding: 10px;
}

.playground {
    position: relative;
    width: 100%;
    height: 100%;
    color: aliceblue;
    font-size: 30px;
    display: grid;
    /* display: none; */
    margin: 0;
    padding: 0;
    grid-template-columns: 27% 38% 34%;
    grid-template-rows: 30% 40% 30%;
    grid-template-
areas: 'backlog header header' 'enemyChest deck playerChest' 'footer footer footer'
;
}

.playground .enemyHand {
    height: 100%;
    width: 100%;
    grid-area: header;
    justify-self: center;
    align-items: center;
}

.playground .enemyHand ul {
    display: flex;
    height: 100%;
    width: 100%;
    list-style: none;
    margin: 0;
    padding: 0;
}

.playground .enemyHand ul li {
    position: absolute;
    top: 10px;
}

.playground p {
    position: absolute;
    top: 10px;
    left: 1200px;
}

.playground .backlog {
```

```
    background-color: #fff;
    height: 182px;
    width: 500px;
    overflow-y: scroll;
    overflow-x: hidden;
    padding: 5px;
}
.playground .backlog p {
    position: relative;
    top: 0;
    left: 0;
    color: black;
    display: inline-block;
    width: 100%;
    word-wrap: break-word;
    font-size: 18px;
}
.playground .deck {
    grid-area: deck;
    position: relative;
    justify-self: center;
    align-self: center;
}

.playground .deck img {
    display: block;
}

.playground .deck p {
    position: absolute;
    font-size: 40px;
    color: white;
    top: 145px;
    left: 63px;
}

.playground .enemyChest {
    position: relative;
    grid-area: enemyChest;
    align-self: center;
}

.playground .enemyChest p {
    color: red;
    position: absolute;
    top: 165px;
    left: 15px;
}
.playground .chest img {
    width: 200px;
}
```

```
.playground .playerChest {
    position: relative;
    grid-area: playerChest;
    align-self: center;
    justify-self: flex-end;
}

.playground .playerChest p {
    color: white;
    top: -79px;
    left: 30px;
}

.playground .playerHand {
    grid-area: footer;
    height: 100%;
    width: 100%;
    justify-self: center;
    align-self: center;
    position: relative;
}

.playground .playerHand ul {
    display: flex;
    height: 100%;
    width: 100%;
    list-style: none;
    margin: 0;
    padding: 0;
}

.playground .playerHand ul li {
    position: absolute;
    top: 10px;
}

.playground .playerHand ul li .card {
    position: absolute;
    border: 1px solid black;
    border-radius: 10px;
    width: 190px;
    height: 245px;
    background: white;
}

.playground .playerHand ul li .card .suitvalue {
    line-height: 70%;
    color: black;
    font-size: 50px;
    position: absolute;
    top: -40px;
```

```
    left: 10px;
}

.playground .playerHand ul li .card .suitBig {
    position: absolute;
    top: -110px;
    left: 45px;
    text-align: center;
    color: black;
    font-size: 150px;
}

.winner-modal {
    width: 800px;
    max-height: 600px;
    background: #fff;
    position: absolute;
    margin: 5% 29%;
    border-radius: 10px;
}
.winner__header {
    width: 100%;
    height: 50px;
    text-align: center;
}
.winner__results {
    margin-bottom: 30px;
}
.chests-list {
    list-style: none;
    display: flex;
    flex-direction: row;
    align-items: center;
    justify-items: flex-start;
}
.chests-list li {
    width: 20px;
    padding: 6px;
}
.winner__restart {
    width: 80%;
    margin: 0 auto;
    display: flex;
    justify-items: center;
    margin-bottom: 60px;
}
.winner__restart button {
    width: 100px;
    margin: 0 auto;
    background: rgb(49, 202, 49);
    color: rgb(83, 23, 139);
```

```
    outline: none;
}

#not-PC {
    display: none;
}
@media screen and (max-width: 1366px) {
    #not-PC {
        display: block;
        color: red;
        font-size: 72px;
        padding-left: 5px;
    }
}
@media (max-width: 1500px) {
    .playground .chest img {
        margin-right: -5px;
    }
    .playground p {
        left: 1123px;
    }
    .playground .enemyChest p {
        font-size: 27px;
    }
}

@media (max-height: 755px) {
    .playground .deck img {
        position: absolute;
        top: -125px;
        left: -315px;
    }
    .playground .deck p {
        top: 26px;
        left: -251px;
    }
}

@media (max-height: 600px) {
    .playground .deck img {
        width: 210px;
        top: -100px;
    }
    .playground .deck p {
        top: 26px;
        left: -251px;
        font-size: 32px;
    }
}
```

Index.js

```
import App from './js/app.js';

new App();
```

app.js

```
import { Deck } from './deck.js';
import { PlayerHand } from './PlayerHand.js';
import { myInterface } from './interface.js';
import { Gameprocess } from './gameProcess.js';

//Клас додатку
class App {
    constructor() {
        this.startApp();
    }
    static rootElement = document.getElementById('root');
    //метод початку гри
    startApp() {
        if (screen.width >= 1367) {
            const deck = new Deck();

            const playerOneHand = new PlayerHand(deck);
            const playerTwoHand = new PlayerHand(deck);
            myInterface.showStartModal(() => {
                myInterface.createStarter(deck);
                myInterface.createPlayerHand(playerOneHand);
                myInterface.createEnemyHand(playerTwoHand);
                myInterface.createChest('1', playerOneHand);
                myInterface.createChest('2', playerTwoHand);
                const gameProcess = new Gameprocess(
                    playerOneHand,
                    playerTwoHand,
                    deck,
                );
            });
        }
        //console.log(deck);
    }
}

export default App;
```

domHelper.js

```
//створення елементу HTML з вказаним тегом, класом та атрибутами
export function createElement({ tagName, className, attributes = {} }) {
    const element = document.createElement(tagName);
```

```
    if (className) {
        const classNames = className.split(' ').filter(Boolean);
        element.classList.add(...classNames);
    }

    Object.keys(attributes).forEach((key) =>
        element.setAttribute(key, attributes[key]),
    );

    return element;
}
```

Card.js

```
//клас карти, що вміщає value & suit
export class Card {
    constructor(suit, value) {
        this.suit = suit;
        this.value = value;
    }
}
```

Deck.js

```
export const SUITS = ['♠', '♣', '♥', '♦'];
export const VALUES = ['6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A'];
import { Card } from './card.js';

//Колода карт
export class Deck {
    constructor(cards = this.freshDeck()) {
        this.cards = cards;
        this.shuffle();
    }
    //створення непотасованої, нової колоди
    freshDeck() {
        return SUITS.flatMap((suit) => {
            return VALUES.map((value) => {
                return new Card(suit, value);
            });
        });
    }
    //отримати кількість карт в колоді
    get numberOfCards() {
        return this.cards.length;
    }
    //тасувати колоду
    shuffle() {
        for (let i = this.numberOfCards - 1; i > 0; i--) {
            const newIndex = Math.floor(Math.random() * (i + 1));
            const oldValue = this.cards[newIndex];
```

```
            this.cards[newIndex] = this.cards[i];
            this.cards[i] = oldValue;
        }
    }
    //взяти одну карту з колоди
    takeCard() {
        return this.cards.length > 0 ? this.cards.pop() : undefined;
    }
}
```

PlayerHand.js

```
//рука гравця
export class PlayerHand {
    // at start there are 4 cards in hand.
    constructor(deck) {
        //карти
        this.cards = [];
        //кількість скарбничок
        this.chestsNum = 0;
        //масив карт, з яких зібрано скарбничку
        this.chestsArr = [];
        //взяти 4 карти на початку
        for (let i = 0; i < 4; i++) {
            this.cards.push(deck.takeCard());
        }
    }
    //отримати кількість карт в руці
    get numberOfCards() {
        return this.cards.length;
    }
    //отримати масив значень карт
    get cardValues() {
        let cardValuesArr = this.cards.map((item) => {
            return item.value;
        });
        return cardValuesArr;
    }
    //перевірити чи є карта з вказаним value в руці
    checkCardValue(value) {
        const found = this.cardValues.find((item) => item == value);
        return found;
    }
    //визначити скільки карт з вказаним value є в руці
    countNumberOfCard(value) {
        let result = 0;
        this.cardValues.filter((item) => {
            if (item == value) {
                result++;
                return item;
            }
        });
```

```
        return result;
    }
    //визначити масті карти з вказаним value в руці
    getCardSuits(value) {
        return this.cards.map((item) => {
            if (item.value == value) {
                return item.suit;
            }
        });
    }
    //перевірити чи є вказана карта в руці
    checkCard(value, suit) {
        for (let i in this.cards) {
            if (this.cards[i].value == value && this.cards[i].suit == suit) {
                return true;
            }
        }
        return false;
    }
    //Забрати карту з руки ворога
    getCard(value, suit) {
        for (let i in this.cards) {
            if (this.cards[i].value == value && this.cards[i].suit == suit) {
                let toReturn = this.cards[i];
                this.cards.splice(i, 1);
                return toReturn;
            }
        }
    }
    //Покласти карту собі в руку
    addCard(card) {
        this.cards.push(card);
    }
    //створення скарбнички
    createChest(value) {
        for (let i = 0; i < this.numberOfCards; i++) {
            if (this.cards[i].value == value) {
                this.cards.splice(i, 1);
                i--;
            }
        }
        this.chestsNum++;
        this.chestsArr.push(value);
    }
}
```

ImagineHandCard.js

```
//карти, що знаходяться в ImagineHand
export class ImagineHandCard {
    constructor(value, accurate, isNot = [], suit) {
```

```
        this.value = value;
        this.accurate = accurate;
        this.isNot = isNot;
        if (suit) {
            this.accurate = true;
            this.suit = suit;
        }
    }
}
```

Interface.js

```
import { createElement } from './helpers/domHelper.js';
import {
    createInnerPlayedHand,
    createPlayground,
    createInnerEnemyHand,
} from './helpers/renderHelper.js';

export class Interface {
    constructor() {}
    static rootElement = document.getElementById('root');
    //створення  колоди, backlog та playground
    createStarter(deck) {
        let checkDeck = document.getElementsByClassName('deck');
        if (checkDeck[0]) {
            checkDeck[0].remove();
        }
        let playground = createPlayground(deck);
        Interface.rootElement.append(playground);
        myInterface.renderDeck(deck);
    }
    //оновлення колоди карт
    renderDeck(deck) {
        const cardsnum = deck.numberOfCards;
        if (cardsnum > 0) {
            document.querySelector('.deck p').innerText = cardsnum;
        } else {
            let deck = document.querySelector('.deck');
            if (deck) {
                deck.remove();
            }
        }
    }
    //створення  руки гравця
    createPlayerHand(playerHand) {
        const playground = document.getElementsByClassName('playground')[0];
        let playerHandDiv = createElement({
            tagName: 'div',
```

```
            className: 'playerHand',
        });
        playerHandDiv = createInnerPlayedHand(playerHand, playerHandDiv);
        playground.append(playerHandDiv);
    }
    //оновлення руки гравця
    renderPlayerHand(playerHand) {
        const playerHandDiv = document.querySelector('.playerHand');
        if (playerHandDiv) {
            playerHandDiv.innerHTML = '';
        }
        createInnerPlayedHand(playerHand, playerHandDiv);
    }
    //створення  руки суперника
    createEnemyHand(playerHand) {
        let enemyHandDiv = createInnerEnemyHand(playerHand);
        const playground = document.getElementsByClassName('playground')[0];
        playground.append(enemyHandDiv);
    }
    //оновлення руки суперника
    renderEnemyHand(playerHand) {
        const playerHandDiv = document.querySelector('.enemyHand');
        if (playerHandDiv) {
            playerHandDiv.innerHTML = '';
        }
        createInnerEnemyHand(playerHand, playerHandDiv);
    }
    //створення початкового модального вікна
    showStartModal(onClose) {
        const root = Interface.rootElement;
        const modal = this.createStartModal(onClose);

        root.append(modal);
    }
    //створення скарбничок
    createChest(player) {
        let chestBlock;
        let chestP = createElement({
            tagName: 'p',
        });
        if (player == '1') {
            chestBlock = createElement({
                tagName: 'div',
                className: 'playerChest chest',
            });
            chestP.innerHTML = `You have <span>0</span> chests`;
        } else if (player == '2') {
            chestBlock = createElement({
                tagName: 'div',
                className: 'enemyChest chest',
            });
```

```
            chestP.innerHTML = `Enemy has <span>0</span> chests`;
        }

        let chestImg = createElement({
            tagName: 'img',
            attributes: {
                src: 'images/chest.png',
                alt: 'chest',
            },
        });
        chestBlock.append(chestP, chestImg);
        document.getElementsByClassName('playground')[0].append(chestBlock);
    }
    //оновлення скарбничок
    renderChest(player, playerHand) {
        let chestP;
        if (player == '1') {
            chestP = document.querySelector('.playerChest p');
            chestP.innerHTML = `You have <span>${playerHand.chestsNum}</span> chest
s`;
        } else if (player == '2') {
            chestP = document.querySelector('.enemyChest p');
            chestP.innerHTML = `Enemy has <span>${playerHand.chestsNum}</span> ches
ts`;
        }
    }
    //додати повідомлення у текстове поле
    addAction(message) {
        const logs = document.querySelector('.backlog p');
        if (logs) {
            logs.innerHTML = message + '<br>' + logs.innerHTML;
        }
    }
    //створити стартове модальне вікно
    createStartModal(onClose) {
        let modalStart = createElement({
            tagName: 'div',
            className: 'modalStart',
        });
        let modalImg = createElement({
            tagName: 'img',
            attributes: {
                src: 'images/message_bg.jpg',
                alt: 'Start',
            },
        });
        let modalP = createElement({
            tagName: 'p',
            attributes: { id: 'start' },
        });
        let modalBtn = createElement({
```

```
            tagName: 'button',
        });
        modalP.innerText = 'Start game';
        modalBtn.innerText = 'Start';
        modalStart.append(modalImg, modalP, modalBtn);

        const close = () => {
            this.hideModal();
            onClose();
        };
        modalBtn.addEventListener('click', close);

        return modalStart;
    }
    //приховати модальне вікно
    hideModal() {
        const modal = document.getElementsByClassName('modalStart')[0];
        modal?.remove();
    }
    showWinner(player, playerOneHand, playerTwoHand) {
        const root = Interface.rootElement;
        root.innerHTML = '';
        const modal = createElement({
            tagName: 'div',
            className: 'winner-modal',
        });
        modal.innerHTML = `
        <div class="winner__header">
        <h1></h1>
        <hr>
    </div>
    <div class="winner__results">
        <ul class="chests-list player">
            <h3>Your chests:</h3>
        </ul>
        <br><br>
        <ul class="chests-list enemy">
            <h3>Enemy chests:</h3>
        </ul >

    </div>
    <div class="winner__restart">
        <button>Play again</button>
    </div>
        `;
        const winner = modal.querySelector('h1');
        if (player == '1') {
            winner.innerText = 'You won!';
        } else {
            winner.innerText = 'You lost!';
        }
```

```
        const playerChests = modal.querySelector('.player');
        const enemyChests = modal.querySelector('.enemy');
        for (let item of playerOneHand.chestsArr) {
            playerChests.innerHTML += `<li>${item}</li>`;
        }
        for (let item of playerTwoHand.chestsArr) {
            enemyChests.innerHTML += `<li>${item}</li>`;
        }
        const winnerBtn = modal.querySelector('.winner__restart');
        winnerBtn.onclick = () => {
            document.location.reload();
        };
        root.append(modal);
    }
}

export const myInterface = new Interface();
```

renderHelper.js

```
import { createElement } from './domHelper.js';
//створення playground, колоди та backlog
export function createPlayground(deck) {
    let playground = createElement({
        tagName: 'div',
        className: 'playground',
    });
    let deckDiv = createElement({
        tagName: 'div',
        className: 'deck',
    });
    let deckPicImg = createElement({
        tagName: 'img',
        attributes: {
            src: 'images/cardForPlayer1.png',
        },
    });
    let deckPicP = createElement({
        tagName: 'p',
    });
    let backlog = createElement({
        tagName: 'div',
        className: 'backlog',
    });
    let backlogP = createElement({
        tagName: 'p',
    });
    deckPicP.innerText = deck.numberOfCards;
    deckDiv.append(deckPicImg, deckPicP);
    backlog.append(backlogP);
    playground.append(backlog);
```

```
    playground.append(deckDiv);
    return playground;
}
//заповнення руки гравця картами
export function createInnerPlayedHand(playerHand, playerHandDiv) {
    let playerHandUl = createElement({
        tagName: 'ul',
    });
    if (playerHand.numberOfCards < 1) {
        playerHandDiv.appendChild(playerHandUl);
        return playerHandDiv;
    }
    const marginCard = 100;
    for (let i = 1; i < playerHand.numberOfCards + 1; i++) {
        let playerHandLi = createElement({
            tagName: 'li',
        });
        let playerHandCardDiv = createElement({
            tagName: 'div',
            className: 'card',
            attributes: {
                id: 'card' + i,
            },
        });
        let playerHandCardSuitValue = createElement({
            tagName: 'p',
            className: 'suitvalue',
        });
        playerHandCardSuitValue.innerHTML =
            playerHand.cards[i - 1]?.value +
            '<br>' +
            playerHand.cards[i - 1]?.suit;
        let playerHandCardSuitBig = createElement({
            tagName: 'p',
            className: 'suitBig',
        });
        playerHandLi.style.left = marginCard + 60 * i + 'px';
        playerHandCardSuitBig.innerText = playerHand.cards[i - 1]?.suit;
        if (
            playerHand.cards[i - 1]?.suit == '♥' ||
            playerHand.cards[i - 1]?.suit == '♦'
        ) {
            playerHandCardSuitValue.style.color = window
                .getComputedStyle(document.documentElement)
                .getPropertyValue('--redSuit');
            playerHandCardSuitBig.style.color = window
                .getComputedStyle(document.documentElement)
                .getPropertyValue('--redSuit');
        } else {
            playerHandCardSuitValue.style.color = window
                .getComputedStyle(document.documentElement)
```

```
                .getPropertyValue('--blackSuit');
            playerHandCardSuitBig.style.color = window
                .getComputedStyle(document.documentElement)
                .getPropertyValue('--blackSuit');
        }

        playerHandCardDiv.append(
            playerHandCardSuitValue,
            playerHandCardSuitBig,
        );
        playerHandLi.append(playerHandCardDiv);
        playerHandUl.append(playerHandLi);
    }
    playerHandDiv.append(playerHandUl);
    return playerHandDiv;
}
//заповнення руки ворога картами
export function createInnerEnemyHand(enemyHand, enemyHandDiv1) {
    let enemyHandDiv = enemyHandDiv1;
    if (!enemyHandDiv1) {
        enemyHandDiv = createElement({
            tagName: 'div',
            className: 'enemyHand',
        });
    }
    let enemyHandUl = createElement({
        tagName: 'ul',
    });
    if (enemyHand.numberOfCards >= 7) {
        for (let i = 1; i < 8; i++) {
            renderNumberOfCardsEnemy(enemyHandUl, i);
        }
        let enemyHandPre = createElement({
            tagName: 'pre',
        });
        let enemyHandP = createElement({
            tagName: 'p',
        });
        enemyHandP.innerHTML = `Enemy has <br>    <span>${enemyHand.numberOfCards}<
/span> cards`;
        enemyHandPre.append(enemyHandP);
        enemyHandDiv.append(enemyHandUl);
        enemyHandDiv.append(enemyHandPre);
    } else if (enemyHand.numberOfCards < 7) {
        for (let i = 1; i < enemyHand.numberOfCards + 1; i++) {
            renderNumberOfCardsEnemy(enemyHandUl, i);
        }
        let enemyHandPre = createElement({
            tagName: 'pre',
        });
        let enemyHandP = createElement({
```

```
            tagName: 'p',
        });
        enemyHandP.innerHTML = `Enemy has <br>    <span>${enemyHand.numberOfCards}<
/span> cards`;
        enemyHandPre.append(enemyHandP);
        enemyHandDiv.append(enemyHandUl);
        enemyHandDiv.append(enemyHandPre);
    }
    return enemyHandDiv;
}
//оновлення кількості карт ворога
function renderNumberOfCardsEnemy(enemyHandUl, i) {
    let enemyHandLi = createElement({
        tagName: 'li',
        className: 'card' + i,
    });
    let enemyHandImg = createElement({
        tagName: 'img',
        attributes: {
            src: 'images/cardForPlayer1.png',
        },
    });
    enemyHandLi.style.left = 600 + 50 * i + 'px';
    enemyHandLi.append(enemyHandImg);
    enemyHandUl.append(enemyHandLi);
}
//оновлення руки ворога
export function renderEnemyHand(enemyHand) {
    if (enemyHand.numberOfCards >= 7) {
        for (let i = 1; i < 8; i++) {
            let enemyHandLi = document.createElement('li');
            enemyHandLi.classList.add('card' + i.toString());
            let enemyHandImg = document.createElement('img');
            enemyHandImg.setAttribute('src', 'images/cardForPlayer1.png');
            enemyHandLi.appendChild(enemyHandImg);
            enemyHandUl.appendChild(enemyHandLi);
        }

        let enemyHandPre = document.createElement('pre');
        let enemyHandP = document.createElement('p');
        enemyHandP.innerHTML = 'У противника <br>    <span>7</span> карт';
        enemyHandPre.appendChild(enemyHandP);
        enemyHandDiv.appendChild(enemyHandUl);
        enemyHandDiv.appendChild(enemyHandPre);
    } else if (enemyHand.numberOfCards < 7) {
        for (let i = 1; i < enemyHand.numberOfCards + 1; i++) {
            let enemyHandLi = document.createElement('li');
            enemyHandLi.classList.add('card' + i.toString());
            let enemyHandImg = document.createElement('img');
            enemyHandImg.setAttribute('src', 'images/cardForPlayer1.png');
            enemyHandLi.appendChild(enemyHandImg);
```

```
            enemyHandUl.appendChild(enemyHandLi);
        }
        let enemyHandPre = document.createElement('pre');
        let enemyHandP = document.createElement('p');
        enemyHandP.innerHTML = `У противника <br>    <span>${enemyHand.numberOfCard
s}</span> карт`;
        enemyHandPre.appendChild(enemyHandP);
        enemyHandDiv.appendChild(enemyHandUl);
        enemyHandDiv.appendChild(enemyHandPre);
    }
}
```

GameProcess.js

```
import { VALUES, SUITS } from './deck.js';
import { myInterface } from './interface.js';
import { ImagineHandCard } from './imagineHandCard.js';
export class Gameprocess {
    constructor(playerOneHand, playerTwoHand, deck) {
        this.deck = deck;
        this.imagineHand = [];
        this.lastused = [];
        this.playerOneHand = playerOneHand;
        this.playerTwoHand = playerTwoHand;
        this.startGame(playerOneHand, playerTwoHand);
    }
    static rootElement = document.getElementById('root');
    //хід гравця
    playerTurn() {
        myInterface.addAction(`<strong>Your turn</strong><br>`);
        while (
            this.playerOneHand.cards.length < 4 &&
            this.deck.numberOfCards > 0
        ) {
            const takenCard = this.playerTakeCard(
                this.playerOneHand,
                'player1',
            );
            if (this.playerOneHand.cards.length == 4) {
                this.checkForChest(takenCard, this.playerOneHand, '1');
            }
        }

        const logs = document.querySelector('.backlog p');
        for (let item of this.playerTwoHand.cards) {
            console.log(item);
        }
        myInterface.addAction(
            `Choose a card  <select name="cardValue" size="1"  required autofocus><
/select>`,
        );
```

```
        const cardChoose = logs.querySelector('select[name=cardValue]');
        cardChoose.innerHTML = '<option disabled selected></option>';
        let cardValues = this.playerOneHand.cardValues;

        for (let i in VALUES) {
            if (cardValues.includes(VALUES[i])) {
                cardChoose.innerHTML =
                    cardChoose.innerHTML +
                    "<option value='" +
                    VALUES[i] +
                    "'>" +
                    VALUES[i] +
                    '</option>';
            } else {
                cardChoose.innerHTML =
                    cardChoose.innerHTML +
                    "<option value='" +
                    VALUES[i] +
                    "' disabled>" +
                    VALUES[i] +
                    '</option>';
            }
        }
        let choosedCard;
        //При виборі карти
        cardChoose.onchange = (e) => {
            e.srcElement.disabled = 'true';
            choosedCard = e.srcElement.value;
            myInterface.addAction(`You choosed ${choosedCard}`);
            //Якщо вгадали карту
            if (this.playerTwoHand.checkCardValue(choosedCard)) {
                myInterface.addAction(
                    `How many?  <select name="numberCards" size="1" required autofo
cus></select>`,
                );
                const numberChoose = logs.querySelector(
                    'select[name=numberCards]',
                );
                numberChoose.innerHTML = `
                    <option disabled selected></option>
                    <option>1</option>
                    <option>2</option>
                    <option>3</option>
                    `;
                //При виборі кількості карт
                numberChoose.onchange = (e) => {
                    e.srcElement.disabled = 'true';
                    const choosedNumber = e.srcElement.value;
                    myInterface.addAction(`You choosed ${choosedNumber}`);
                    const realNumber =
                        this.playerTwoHand.countNumberOfCard(choosedCard);
```

```
                    //Якщо вкгадали кількість карт
                    if (+choosedNumber === realNumber) {
                        myInterface.addAction(
                            `Choose suits?  <select name="cardSuits" size="2" requi
red autofocus multiple></select>`,
                        );
                        const suitsChoose = logs.querySelector(
                            'select[name=cardSuits]',
                        );
                        suitsChoose.innerHTML = `
                        <option disabled ></option>`;
                        const playerSuits =
                            this.playerOneHand.getCardSuits(choosedCard);
                        for (let i in SUITS) {
                            if (!playerSuits.includes(SUITS[i])) {
                                suitsChoose.innerHTML += `<option value="${SUITS[i]
}">${SUITS[i]}</option>`;
                            }
                        }
                        let suitsChoosed = [];
                        suitsChoose.onchange = (e) => {
                            if (suitsChoosed.length < choosedNumber) {
                                suitsChoosed.push(e.srcElement.value);
                                let choosedSuit = e.srcElement.querySelector(
                                    `option[value=${e.srcElement.value}]`,
                                );
                                choosedSuit.disabled = 'true';
                                //коли обрали всі масті
                                if (suitsChoosed.length == choosedNumber) {
                                    e.srcElement.disabled = 'true';
                                    e.srcElement.size = '1';
                                    myInterface.addAction(
                                        `You choosed ${suitsChoosed.join(' ')}`,
                                    );
                                    this.getEnemyCards(
                                        this.playerOneHand,
                                        this.playerTwoHand,
                                        suitsChoosed,
                                        choosedCard,
                                        'player1',
                                    );
                                    while (
                                        this.playerOneHand.cards.length < 4 &&
                                        this.deck.numberOfCards > 0
                                    ) {
                                        const takenCard = this.playerTakeCard(
                                            this.playerTwoHand,
                                            'player2',
                                        );
                                        this.checkForChest(
                                            takenCard,
```

```
                                this.playerTwoHand,
                                '2',
                            );
                        }
                        if (
                            this.deck.cards.length > 0 ||
                            this.playerTwoHand.numberOfCards > 0
                        ) {
                            this.enemyTurn();
                        }
                    }
                }
            };
        } else {
            myInterface.addAction(`You didn't guess`);
            const takenCard = this.playerTakeCard(
                this.playerOneHand,
                'player1',
            );
            this.checkForChest(takenCard, this.playerOneHand, '1');
            if (
                !this.imagineHand.find((item) => {
                    if (
                        item.value == choosedCard &&
                        item.accurate == false
                    )
                        return item;
                })
            ) {
                this.imagineHand.push(
                    new ImagineHandCard(choosedCard, false),
                );
            }
            while (
                this.playerOneHand.cards.length < 4 &&
                this.deck.numberOfCards > 0
            ) {
                const takenCard = this.playerTakeCard(
                    this.playerTwoHand,
                    'player2',
                );
                this.checkForChest(
                    takenCard,
                    this.playerTwoHand,
                    '2',
                );
            }
            if (
                this.deck.cards.length > 0 ||
                this.playerTwoHand.numberOfCards > 0
            ) {
```

```
                        this.enemyTurn();
                    }
                }
            };
        } else {
            myInterface.addAction("You didn't guess.");
            const takenCard = this.playerTakeCard(
                this.playerOneHand,
                'player1',
            );
            this.checkForChest(takenCard, this.playerOneHand, '1');
            if (
                !this.imagineHand.find((item) => {
                    if (item.value == choosedCard && item.accurate == false)
                        return item;
                })
            ) {
                this.imagineHand.push(
                    new ImagineHandCard(choosedCard, false),
                );
            }
            while (
                this.playerOneHand.cards.length < 4 &&
                this.deck.numberOfCards > 0
            ) {
                const takenCard = this.playerTakeCard(
                    this.playerTwoHand,
                    'player2',
                );
                this.checkForChest(takenCard, this.playerTwoHand, '2');
            }
            if (
                this.deck.cards.length > 0 ||
                this.playerTwoHand.numberOfCards > 0
            ) {
                this.enemyTurn();
            }
        }
    };
}
//Хід комп'ютера
enemyTurn() {
    //дібрати карти якщо їх менше 4
    while (
        this.playerTwoHand.cards.length < 4 &&
        this.deck.numberOfCards > 0
    ) {
        const takenCard = this.playerTakeCard(
            this.playerTwoHand,
            'player2',
        );
```

```
            this.checkForChest(takenCard, this.playerTwoHand, '2');
        }

        myInterface.addAction('<strong>Enemy turn</strong><br>');
        let priority = -1;
        let bestCardValue = null;
        if (this.deck.cards.length == 0) {
            bestCardValue = this.playerTwoHand.cards[0].value;
        } else {
            //Обрати кращу карту
            for (let value of VALUES) {
                const newPriority = this.getCardPriority(
                    value,
                    this.playerTwoHand,
                    '2',
                );
                if (priority < newPriority) {
                    priority = newPriority;
                    bestCardValue = value;
                }
            }
        }

        myInterface.addAction(`Ememy asked ${bestCardValue}`);
        this.addToLastUsed(bestCardValue);
        //Якщо у гравця є ця карта
        if (this.playerOneHand.checkCardValue(bestCardValue)) {
            //вгадати кількість
            let numberToAsk;
            if (this.deck.cards.length == 0) {
                numberToAsk =
                    4 - this.playerTwoHand.countNumberOfCard(bestCardValue);
            } else {
                numberToAsk = this.imagineHand.filter(
                    (item) => item.value == bestCardValue,
                ).length;
                if (numberToAsk == 0) numberToAsk++;
            }
            let inHand = this.playerTwoHand.countNumberOfCard(bestCardValue);
            while (inHand + numberToAsk > 4) {
                numberToAsk--;
            }
            myInterface.addAction(`Ememy asked ${numberToAsk} cards`);
            //якщо вгадали кількість
            if (
                this.playerOneHand.countNumberOfCard(bestCardValue) ==
                numberToAsk
            ) {
                //Намагаємося визначити масть
                let suitsInHand =
                    this.playerTwoHand.getCardSuits(bestCardValue);
```

```
            let suitsToAsk;
            if (this.deck.cards.length == 0) {
                suitsToAsk = [];
                for (let suit of SUITS) {
                    if (!suitsInHand.includes(suit)) {
                        suitsToAsk.push(suit);
                    }
                }
            } else {
                let suitsInImagineAccurate = this.imagineHand.map(
                    (item) => {
                        if (item.value == bestCardValue && item.accurate)
                            return item.suit;
                    },
                );
                let suitsInImagineNot = new Set();
                let inImagineNotArr = this.imagineHand.map((item) => {
                    if (
                        item.value == bestCardValue &&
                        item.isNot.length != 0
                    ) {
                        return item.isNot;
                    }
                });
                inImagineNotArr.forEach((item) => {
                    if (item) {
                        for (let i = 0; i < item.length; i++) {
                            suitsInImagineNot.add(item[i]);
                        }
                    }
                });

                inImagineNotArr = Array.from(suitsInImagineNot);
                suitsToAsk = [];
                let randomSuits = new Set();
                let suit = SUITS[0];
                while (randomSuits.size != 4) {
                    let iterator = Math.floor(Math.random() * 4);
                    suit = SUITS[iterator];
                    randomSuits.add(suit);
                }
                randomSuits = Array.from(randomSuits);
                for (let suit of randomSuits) {
                    if (suitsInImagineAccurate.includes(suit)) {
                        suitsToAsk.push(suit);
                    }
                }
                if (suitsToAsk.length != numberToAsk) {
                    for (let suit of randomSuits) {
                        if (suitsToAsk.length == numberToAsk) break;
                        else {
```

```
                        if (
                            !suitsInHand.includes(suit) &&
                            !inImagineNotArr.includes(suit) &&
                            !suitsToAsk.includes(suit)
                        ) {
                            suitsToAsk.push(suit);
                        }
                    }
                }
            }
            //suitsToAsk - масті, що ми запитаємо
        }

        this.getEnemyCards(
            this.playerTwoHand,
            this.playerOneHand,
            suitsToAsk,
            bestCardValue,
            'player2',
        );
    } else {
        myInterface.addAction("Enemy didn't guess.");
        const numInImagine = this.imagineHand.map(
            (item) => item.value == bestCardValue,
        ).length;
        if (numInImagine == 1) {
            this.imagineHand.push(
                new ImagineHandCard(bestCardValue, false, []),
            );
        } else if (
            numInImagine == 3 ||
            (numInImagine == 2 && this.deck.cards.length > 14)
        ) {
            let i = this.imagineHand.findIndex((item, index) => {
                if (
                    item.value == bestCardValue &&
                    item.isNot.length == 0 &&
                    item.accurate == false
                ) {
                    return index;
                }
            });
            if (i >= 0) {
                this.imagineHand.splice(i, 1);
            } else {
                i = this.imagineHand.findIndex((item, index) => {
                    if (
                        item.value == bestCardValue &&
                        item.accurate == false
                    )
                        return index;
```

```
                    });
                    if (i >= 0) {
                        this.imagineHand.splice(i, 1);
                    } else {
                        i = this.imagineHand.findIndex((item, index) => {
                            if (item.value == bestCardValue) return index;
                        });
                        this.imagineHand.splice(i, 1);
                    }
                }
            } else if (numInImagine == 0) {
                this.imagineHand.push(
                    new ImagineHandCard(bestCardValue, false, []),
                );
                this.imagineHand.push(
                    new ImagineHandCard(bestCardValue, false, []),
                );
            } else if (numInImagine == 2) {
                this.imagineHand.push(
                    new ImagineHandCard(bestCardValue, false, []),
                );
            }
            const takenCard = this.playerTakeCard(
                this.playerTwoHand,
                'player2',
            );
            this.checkForChest(takenCard, this.playerTwoHand, '2');
        }
    } else {
        myInterface.addAction("Enemy didn't guess.");
        const takenCard = this.playerTakeCard(
            this.playerTwoHand,
            'player2',
        );
        this.checkForChest(takenCard, this.playerTwoHand, '2');
    }
    //дібрати карти
    while (
        this.playerTwoHand.cards.length < 4 &&
        this.deck.numberOfCards > 0
    ) {
        const takenCard = this.playerTakeCard(
            this.playerTwoHand,
            'player2',
        );
        this.checkForChest(takenCard, this.playerTwoHand, '2');
    }
    //передати хід
    if (
        this.deck.cards.length > 0 ||
        this.playerTwoHand.numberOfCards > 0
```

```
        ) {
            this.playerTurn();
        }
    }
    //Перевірка наявності карт в руці опонента, та забирання їх в позитивному разі.
    getEnemyCards(toHand, fromHand, suits, value, player) {
        let actionText = '';
        for (let suit of suits) {
            actionText += value + suit + ', ';
        }
        if (player == 'player1') {
            myInterface.addAction('Cards to be checked: ' + actionText);
        } else {
            myInterface.addAction('Cards to be checked(enemy): ' + actionText);
        }
        actionText = '';
        let counter = 0;
        if (player == 'player1') {
            for (let suit of suits) {
                if (fromHand.checkCard(value, suit)) {
                    const guessedCard = fromHand.getCard(value, suit);
                    actionText += value + suit + ', ';
                    counter++;

                    if (
                        !this.imagineHand.find((item) => {
                            if (
                                item.value == value &&
                                item.accurate == true &&
                                item.isNot.length == 0 &&
                                item.suit == suit
                            )
                                return item;
                        })
                    ) {
                        this.imagineHand.push(
                            new ImagineHandCard(value, true, [], suit),
                        );
                    }
                    toHand.addCard(guessedCard);
                    this.checkForChest(value, toHand, '1');
                } else {
                    if (
                        !this.imagineHand.find((item) => {
                            if (
                                item.value == value &&
                                item.accurate == false &&
                                item.isNot.includes(suit)
                            )
                                return item;
                        })
```

```
                ) {
                    this.imagineHand.push(
                        new ImagineHandCard(value, false, [suit]),
                    );
                }
            }
        }

        if (counter == 0) {
            myInterface.addAction("You didn't guess");
            const takenCard = this.playerTakeCard(toHand, 'player1');
            this.checkForChest(takenCard, toHand, '1');
        } else {
            if (counter == suits.length) {
                if (
                    !this.imagineHand.find((item) => {
                        if (item.value == value && item.accurate == false)
                            return item;
                    })
                ) {
                    this.imagineHand.push(
                        new ImagineHandCard(value, false),
                    );
                }
            }
            myInterface.addAction("You've got cards: " + actionText);
            if (counter < suits.length) {
                let takenCard = this.playerTakeCard(toHand, 'player1');
                this.checkForChest(takenCard, toHand, '1');
            }
        }
        if (this.playerOneHand.chestsArr.includes(value)) {
            for (let i = 0; i < this.imagineHand.length; i++) {
                if (this.imagineHand[i].value == value) {
                    this.imagineHand.splice(i, 1);
                    i--;
                }
            }
            for (let i = 0; i < this.lastused.length; i++) {
                if (this.lastused[i] == value) {
                    this.lastused.splice(i, 1);
                    i--;
                }
            }
        }
        myInterface.renderPlayerHand(toHand);
        myInterface.renderEnemyHand(fromHand);
    } else {
        for (let suit of suits) {
            if (fromHand.checkCard(value, suit)) {
                const guessedCard = fromHand.getCard(value, suit);
```

```
                actionText += value + suit + ', ';
                counter++;
                toHand.addCard(guessedCard);
                let i = this.imagineHand.findIndex((item, index) => {
                    if (
                        item.value == value &&
                        item.accurate == true &&
                        item.suit == suit
                    ) {
                        return index;
                    }
                });
                if (i >= 0) {
                    this.imagineHand.splice(i, 1);
                } else {
                    i = this.imagineHand.findIndex((item, index) => {
                        if (item.value == value && item.isNot.length == 0) {
                            return index;
                        }
                    });
                    if (i >= 0) {
                        this.imagineHand.splice(i, 1);
                    } else {
                        i = this.imagineHand.findIndex((item, index) => {
                            if (item.value == value) {
                                return index;
                            }
                        });
                        this.imagineHand.splice(i, 1);
                    }
                }
                this.checkForChest(value, toHand, '2');
            } else {
                let itemIm = this.imagineHand.find((item) => {
                    if (
                        item.value == value &&
                        item.accurate == false &&
                        item.isNot.length != 0
                    ) {
                        return item;
                    }
                });
                if (itemIm) {
                    itemIm.isNot.push(suit);
                } else {
                    this.imagineHand.push(
                        new ImagineHandCard(value, false, [suit]),
                    );
                }
            }
        }
```

```
            if (counter == 0) {
                myInterface.addAction("Enemy didn't guess");
                const takenCard = this.playerTakeCard(toHand, 'player2');
                this.checkForChest(takenCard, toHand, '2');
            } else {
                myInterface.addAction('Enemy got cards: ' + actionText);
                if (counter < suits.length) {
                    let takenCard = this.playerTakeCard(toHand, 'player2');
                    this.checkForChest(takenCard, toHand, '2');
                }
            }
            if (this.playerTwoHand.chestsArr.includes(value)) {
                for (let i = 0; i < this.imagineHand.length; i++) {
                    if (this.imagineHand[i].value == value) {
                        this.imagineHand.splice(i, 1);
                        i--;
                    }
                }
                for (let i = 0; i < this.lastused.length; i++) {
                    if (this.lastused[i] == value) {
                        this.lastused.splice(i, 1);
                        i--;
                    }
                }
            }
            if (
                this.deck.cards.length > 0 &&
                this.playerTwoHand.cards.length > 0
            ) {
                myInterface.renderPlayerHand(fromHand);
                myInterface.renderEnemyHand(toHand);
            }
        }
    }
    //початок гри
    startGame(playerOneHand, playerTwoHand) {
        this.playerTurn(playerOneHand, playerTwoHand);
    }
    //Взяти карти для обраного гравця
    playerTakeCard(playerHand, player) {
        if (this.deck.numberOfCards > 0) {
            playerHand.cards.push(this.deck.takeCard());
            if (player == 'player1') {
                myInterface.renderPlayerHand(playerHand);
                myInterface.addAction(
                    `You took a card: ` +
                        playerHand.cards[playerHand.numberOfCards - 1]?.value +
                        playerHand.cards[playerHand.numberOfCards - 1]?.suit,
                );
            } else {
                myInterface.renderEnemyHand(playerHand);
```

```
                myInterface.addAction(`Enemy took a card`);
            }
            myInterface.renderDeck(this.deck);
            return playerHand.cards[playerHand.cards.length - 1].value;
        }
    }
    //Вибір кращої карти
    getCardPriority(cardValue, playerHand, player) {
        let newPriority = 0;
        let numberInHand = playerHand.countNumberOfCard(cardValue);
        if (numberInHand == 0) {
            newPriority = -1;
        } else if (numberInHand == 1) {
            let cardinlastused = this.lastused.find(
                (item) => item == cardValue,
            );
            if (cardinlastused) {
                let counter = 0;
                this.imagineHand.forEach((item) => {
                    if (item.value == cardValue && item.accurate == true)
                        counter++;
                });
                if (counter == 0) newPriority = 0;
                else if (counter == 1) newPriority = 1200;
                else if (counter == 2) newPriority = 2500;
                else if (counter == 3) newPriority = 10000;
            } else {
                let counter = 0;
                this.imagineHand.forEach((item) => {
                    if (item.value == cardValue) counter++;
                });
                if (counter == 0) newPriority = 1000;
                else if (counter == 1) {
                    if (
                        this.imagineHand.find((item) => item.value == cardValue)
                            .accurate
                    ) {
                        newPriority = 1800;
                    } else {
                        newPriority = 1100;
                    }
                } else if (counter == 2) {
                    let counterAcc = 0;
                    this.imagineHand.forEach((item) => {
                        if (item.value == cardValue && item.accurate == true)
                            counterAcc++;
                    });
                    if (counterAcc == 0) newPriority = 2100;
                    else if (counterAcc == 1) newPriority = 2500;
                    else if (counterAcc == 2) newPriority = 2800;
                } else if (counter == 3) newPriority = 10000;
```

```
            }
        } else if (numberInHand == 2) {
            let cardinlastused = this.lastused.find(
                (item) => item == cardValue,
            );
            if (cardinlastused) {
                let counter = 0;
                this.imagineHand.forEach((item) => {
                    if (item.value == cardValue && item.accurate == true)
                        counter++;
                });
                if (counter == 0) newPriority = 0;
                else if (counter == 1) newPriority = 2800;
                else if (counter == 2) newPriority = 10000;
            } else {
                let counter = 0;
                this.imagineHand.forEach((item) => {
                    if (item.value == cardValue) counter++;
                });
                if (counter == 0) newPriority = 1800;
                else if (counter == 1) {
                    if (
                        this.imagineHand.find((item) => item.value == cardValue)
                            .accurate
                    ) {
                        newPriority = 2800;
                    } else {
                        newPriority = 2500;
                    }
                } else if (counter == 2) {
                    newPriority = 10000;
                }
            }
        } else if (numberInHand == 3) {
            let cardinlastused = this.lastused.find(
                (item) => item == cardValue,
            );
            if (cardinlastused) {
                let counterim = 0;
                this.imagineHand.forEach((item) => {
                    if (item.value == cardValue) counterim++;
                });
                if (counterim == 0) newPriority = 0;
                else {
                    newPriority = 10000;
                }
            } else {
                let counterim = 0;
                this.imagineHand.forEach((item) => {
                    if (item.value == cardValue) counterim++;
                });
```

```
            if (counterim == 0) newPriority = 2600;
            else {
                newPriority = 10000;
            }
        }
    } else if (numberInHand == 4) {
        this.checkForChest(cardValue, playerHand, player);
    }

    return newPriority;
}
//Перевірка на наявність скарбнички
checkForChest(value, playerHand, player) {
    let counter = 0;
    playerHand.cards.forEach((item) => {
        if (item.value == value) counter++;
    });
    if (counter == 4) {
        playerHand.createChest(value);
        myInterface.renderChest(player, playerHand);
        if (player == '1') myInterface.renderPlayerHand(playerHand);
        else {
            myInterface.renderEnemyHand(playerHand);
        }
        myInterface.addAction(`You've got a chest from ${value}`);
        for (let i = 0; i < this.imagineHand.length; i++) {
            if (this.imagineHand[i].value == value) {
                this.imagineHand.splice(i, 1);
                i--;
            }
        }
        for (let i = 0; i < this.lastused.length; i++) {
            if (this.lastused[i] == value) {
                this.lastused.splice(i, 1);
                i--;
            }
        }
        if (this.deck.numberOfCards == 0 && playerHand.cards.length == 0) {
            if (player == '1') {
                if (playerHand.chestsNum >= 5)
                    myInterface.showWinner(
                        '1',
                        this.playerOneHand,
                        this.playerTwoHand,
                    );
                else {
                    myInterface.showWinner(
                        '2',
                        this.playerOneHand,
                        this.playerTwoHand,
                    );
```

```
                    }
                } else {
                    if (playerHand.chestsNum >= 5)
                        myInterface.showWinner(
                            '2',
                            this.playerOneHand,
                            this.playerTwoHand,
                        );
                    else {
                        myInterface.showWinner(
                            '1',
                            this.playerOneHand,
                            this.playerTwoHand,
                        );
                    }
                }
            }
            return true;
        }
        return false;
    }
    //запам'ятовування, що цю карту було використано в останні 3 ходи
    addToLastUsed(value) {
        if (this.lastused.length < 3) {
            this.lastused.push(value);
        } else {
            this.lastused.shift();
            this.lastused.push(value);
        }
    }
}
```