

**Міністерство освіти і науки України**  
**Національний технічний університет України «Київський політехнічний**  
**інститут імені Ігоря Сікорського»**  
**Факультет інформатики та обчислювальної техніки**  
  
**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 6 з дисципліни  
«Проектування алгоритмів»

**„Пошук в умовах протидії, ігри з повною інформацією, ігри з елементом  
випадковості, ігри з неповною інформацією”**

**Виконав(ла)**

Присяжний А. О.

(шифр, прізвище, ім'я, по батькові)

**Перевірив**

Ахаладзе І. Е.

(прізвище, ім'я, по батькові)

Київ 2023

## ЗМІСТ

<b>1</b>	<b>МЕТА ЛАБОРАТОРНОЇ РОБОТИ .....</b>	<b>3</b>
<b>2</b>	<b>ЗАВДАННЯ .....</b>	<b>4</b>
<b>3</b>	<b>ВИКОНАННЯ.....</b>	<b>7</b>
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ .....	7
3.1.1	<i>Вихідний код.....</i>	7
3.1.2	<i>Приклади роботи .....</i>	54
	<b>ВИСНОВОК .....</b>	<b>55</b>
	<b>КРИТЕРІЇ ОЦІНЮВАННЯ .....</b>	<b>56</b>

## 1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи - вивчити основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Ознайомитися з підходами до програмування алгоритмів штучного інтелекту в іграх з повною інформацією, іграх з елементами випадковості та в іграх з неповною інформацією.

## 2 ЗАВДАННЯ

Для ігор з повної інформацією, згідно варіанту (таблиця 2.1) реалізувати візуальний ігровий додаток для гри користувача з комп'ютерним опонентом. Для реалізації стратегії гри комп'ютерного опонента використовувати алгоритм альфа-бета-відсікань. Реалізувати три рівні складності (легкий, середній, складний).

Для ігор з елементами випадковості, згідно варіанту (таблиця 2.1) реалізувати візуальний ігровий додаток, з користувацьким інтерфейсом, не консольним, для гри користувача з комп'ютерним опонентом. Для реалізації стратегії гри комп'ютерного опонента використовувати алгоритм мінімакс.

Для карткових ігор, згідно варіанту (таблиця 2.1), реалізувати візуальний ігровий додаток, з користувацьким інтерфейсом, не консольним, для гри користувача з комп'ютерним опонентом. Потрібно реалізувати стратегію комп'ютерного опонента, і звести гру до гри з повною інформацією (див. Лекцію), далі реалізувати стратегію гри комп'ютерного опонента за допомогою алгоритму мінімаксу або альфа-бета-відсікань.

Реалізувати анімацію процесу жеребкування (+1 бал) або реалізувати анімацію ігрових процесів (роздачі карт, анімацію ходів тощо) (+1 бал).

Реалізувати варто тільки одне з бонусних завдань.

Зробити узагальнений висновок лабораторної роботи.

Таблиця 2.1 – Варіанти

№	Варіант	Тип гри
1	Яцзи <a href="https://game-wiki.guru/published/igryi/yaczzyi.html">https://game-wiki.guru/published/igryi/yaczzyi.html</a>	3 елементами випадковості
2	Лудо <a href="http://www.iggamecenter.com/info/ru/ludo.html">http://www.iggamecenter.com/info/ru/ludo.html</a>	3 елементами випадковості
3	Генерал <a href="http://www.rules.net.ru/kost.php?id=7">http://www.rules.net.ru/kost.php?id=7</a>	3 елементами випадковості

4	Нейтріко <a href="http://www.iggamecenter.com/info/ru/neutreeko.html">http://www.iggamecenter.com/info/ru/neutreeko.html</a>	З повною інформацією
5	Тринадцять <a href="http://www.rules.net.ru/kost.php?id=16">http://www.rules.net.ru/kost.php?id=16</a>	З елементами випадковості
6	Індійські кості <a href="http://www.rules.net.ru/kost.php?id=9">http://www.rules.net.ru/kost.php?id=9</a>	З елементами випадковості
7	Dots and Boxes <a href="https://ru.wikipedia.org/wiki/Палочки_(игра)">https://ru.wikipedia.org/wiki/Палочки_(игра)</a>	З повною інформацією
8	Двадцять одне <a href="http://gamerules.ru/igry-v-kosti-part8#dvadtsat-odno">http://gamerules.ru/igry-v-kosti-part8#dvadtsat-odno</a>	З елементами випадковості
9	Тіко <a href="http://www.iggamecenter.com/info/ru/teeko.html">http://www.iggamecenter.com/info/ru/teeko.html</a>	З повною інформацією
10	Клоббер <a href="http://www.iggamecenter.com/info/ru/clobber.html">http://www.iggamecenter.com/info/ru/clobber.html</a>	З повною інформацією
11	101 <a href="https://www.durbetsel.ru/2_101.htm">https://www.durbetsel.ru/2_101.htm</a>	Карткові ігри
12	Hackenbush <a href="http://www.papg.com/show?1TMP">http://www.papg.com/show?1TMP</a>	З повною інформацією
13	Табу <a href="https://www.durbetsel.ru/2_taboo.htm">https://www.durbetsel.ru/2_taboo.htm</a>	Карткові ігри
14	Заєць і Вовки (за Зайця) <a href="http://www.iggamecenter.com/info/ru/foxh.html">http://www.iggamecenter.com/info/ru/foxh.html</a>	З повною інформацією
15	Свої козири <a href="https://www.durbetsel.ru/2_svoi-koziri.htm">https://www.durbetsel.ru/2_svoi-koziri.htm</a>	Карткові ігри
16	Війна з ботами <a href="https://www.durbetsel.ru/2_voina_s_botami.htm">https://www.durbetsel.ru/2_voina_s_botami.htm</a>	Карткові ігри
17	Domineering 8x8 <a href="http://www.papg.com/show?1TX6">http://www.papg.com/show?1TX6</a>	З повною інформацією
18	Останній гравець <a href="https://www.durbetsel.ru/2_posledny_igrok.htm">https://www.durbetsel.ru/2_posledny_igrok.htm</a>	Карткові ігри
19	Заєць и Вовки (за Вовків) <a href="http://www.iggamecenter.com/info/ru/foxh.html">http://www.iggamecenter.com/info/ru/foxh.html</a>	З повною інформацією

20	Богач <a href="https://www.durbetsel.ru/2_bogach.htm">https://www.durbetsel.ru/2_bogach.htm</a>	Карткові ігри
21	Редуду <a href="https://www.durbetsel.ru/2_redudu.htm">https://www.durbetsel.ru/2_redudu.htm</a>	Карткові ігри
22	Эльферн <a href="https://www.durbetsel.ru/2_elfern.htm">https://www.durbetsel.ru/2_elfern.htm</a>	Карткові ігри
23	Ремінь <a href="https://www.durbetsel.ru/2_remen.htm">https://www.durbetsel.ru/2_remen.htm</a>	Карткові ігри
24	Реверсі <a href="https://ru.wikipedia.org/wiki/Реверси">https://ru.wikipedia.org/wiki/Реверси</a>	З повною інформацією
25	Вари <a href="http://www.iggamecenter.com/info/ru/oware.html">http://www.iggamecenter.com/info/ru/oware.html</a>	З повною інформацією
26	Яцзи <a href="https://game-wiki.guru/published/igryi/yaczzyi.html">https://game-wiki.guru/published/igryi/yaczzyi.html</a>	З елементами випадковості
27	Лудо <a href="http://www.iggamecenter.com/info/ru/ludo.html">http://www.iggamecenter.com/info/ru/ludo.html</a>	З елементами випадковості
28	Генерал <a href="http://www.rules.net.ru/kost.php?id=7">http://www.rules.net.ru/kost.php?id=7</a>	З елементами випадковості
29	Сим <a href="https://ru.wikipedia.org/wiki/Сим_(игра)">https://ru.wikipedia.org/wiki/Сим_(игра)</a>	З повною інформацією
30	Col <a href="http://www.papg.com/show?2XLY">http://www.papg.com/show?2XLY</a>	З повною інформацією
31	Snort <a href="http://www.papg.com/show?2XM1">http://www.papg.com/show?2XM1</a>	З повною інформацією
32	Chomp <a href="http://www.papg.com/show?3AEA">http://www.papg.com/show?3AEA</a>	З повною інформацією
33	Gale <a href="http://www.papg.com/show?1TPI">http://www.papg.com/show?1TPI</a>	З повною інформацією
34	3D Noughts and Crosses 4 x 4 x 4 <a href="http://www.papg.com/show?1TND">http://www.papg.com/show?1TND</a>	З повною інформацією
35	Snakes <a href="http://www.papg.com/show?3AE4">http://www.papg.com/show?3AE4</a>	З повною інформацією

## 3 ВИКОНАННЯ

### 3.1 Програмна реалізація алгоритму

#### 3.1.1 Вихідний код

##### **Cards.h**

```
#pragma once
```

```
#include <vector>
```

```
#include <algorithm>
```

```
using namespace std;
```

```
enum Suits {  
    CLUBS,  
    DIAMONDS,  
    HEARTS,  
    SPADES,  
};
```

```
enum CardNames {  
    THREE = 1,  
    FOUR,  
    FIVE,  
    SIX,  
    SEVEN,  
    EIGHT,  
    NINE,  
    TEN,
```

```

        JACK,
        QUEEN,
        KING,
        ACE,
        TWO,
};

class Card {
    Suits suit;
    CardNames name;

public:
    Card(Suits suit, CardNames name);

    Suits getSuit() const;
    CardNames getName() const;

    bool operator==(const Card& obj) const;
    bool operator>(const Card& obj) const;
};

class Deck {
    vector<Card*> cards;

public:
    Deck();

    void shuffle();
    Card* getCard();

```



```

        void fill();
        bool isEmpty() const;

        ~Deck();
};

```

## Player.h

```

#pragma once

#include "Cards.h"
#include <iterator>

class Player {
protected:
    vector<Card*> cards;
    vector<Card*> selectedCards;
    vector<Card*> cardsToBeat;

    int findPositionForCard(Card* card);

    void removeSelectedCardsFromAllCards();
    void selectCardsToFightBack(Card* anchorCard);
public:
    void addCard(Card* card);
    vector<Card*> getCards();
    vector<Card*> getSelectedCards();
    void setCardsToBeat(vector<Card*> cards);
    vector<Card*> makeTurn();
    bool hasWon() const;
    void clearAllCards();

```

```
};
```

```
class AIPlayer : public Player {  
    void selectCardsForFirstTurn();  
    void selectCardsForBeatTurn();  
public:  
    void selectCardsForTurn();  
};
```

```
class HumanPlayer : public Player {  
public:  
    bool selectCardsForTurn(Card* selectedCard);  
    bool canFightBack();  
};
```

### **Cards.cpp**

```
#include "Cards.h"
```

```
Card::Card(Suits suit, CardNames name) {  
    this->suit = suit;  
    this->name = name;  
}
```

```
Suits Card::getSuit() const {  
    return suit;  
}
```

```
CardNames Card::getName() const {
```

```

        return name;
    }

    bool Card::operator==(const Card& obj) const {
        return this->name == obj.name;
    }

    bool Card::operator>(const Card& obj) const {
        return this->name > obj.name;
    }

Deck::Deck() {
    fill();
    shuffle();
}

void Deck::fill() {
    vector<Suits> suits = { CLUBS, DIAMONDS, HEARTS, SPADES };
    vector<CardNames> cardNames = { TWO, THREE, FOUR, FIVE, SIX,
SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE };

    for (auto suit : suits) {
        for (auto cardName : cardNames) {
            Card* card = new Card(suit, cardName);
            cards.push_back(card);
        }
    }
}

```

```
void Deck::shuffle() {
    random_shuffle(cards.begin(), cards.end());
}
```

```
Card* Deck::getCard() {
    if (cards.empty())
        return nullptr;

    Card* card = cards.back();
    cards.pop_back();
    return card;
}
```

```
bool Deck::isEmpty() const {
    return cards.empty();
}
```

```
Deck::~~Deck() {
    for (auto card : cards)
        delete card;
}
```

### **Player.cpp**

```
#include "Player.h"
```

```
int Player::findPositionForCard(Card* card) {
    int index = 0;

    while (index < cards.size() && *card > *cards.at(index))
```

```

        ++index;

    return index;
}

void Player::removeSelectedCardsFromAllCards() {
    if (selectedCards.empty())
        return;

    for (auto selectedCard : selectedCards) {
        int i = 0;

        while (i < cards.size()) {
            if (*cards.at(i) == *selectedCard && cards.at(i)->getSuit()
== selectedCard->getSuit()) {
                cards.erase(find(cards.begin(), cards.end(),
cards.at(i)));
                --i;
            }
            ++i;
        }
    }
}

void Player::selectCardsToFightBack(Card* anchorCard) {
    for (auto card : cards) {
        if (*card == *anchorCard && selectedCards.size() <
cardsToBeat.size())
            selectedCards.push_back(card);
    }
}

```

```

        if (selectedCards.size() != cardsToBeat.size())
            selectedCards.clear();
    }

    void Player::addCard(Card* card) {
        int index = findPositionForCard(card);
        cards.insert(cards.begin() + index, card);
    }

    vector<Card*> Player::getCards() {
        return cards;
    }

    vector<Card*> Player::getSelectedCards() {
        return selectedCards;
    }

    void Player::setCardsToBeat(vector<Card*> cards) {
        cardsToBeat.clear();
        copy(cards.begin(), cards.end(), back_inserter(cardsToBeat));
    }

    vector<Card*> Player::makeTurn() {
        removeSelectedCardsFromAllCards();

        vector<Card*> cardsToReturn;
        copy(selectedCards.begin(), selectedCards.end(),
back_inserter(cardsToReturn));
    }

```

```

        selectedCards.clear();
        cardsToBeat.clear();

        return cardsToReturn;
    }

    bool Player::hasWon() const {
        return cards.empty();
    }

    void Player::clearAllCards() {
        cards.clear();
        selectedCards.clear();
        cardsToBeat.clear();
    }

    void AIPlayer::selectCardsForTurn() {
        if (cardsToBeat.empty())
            selectCardsForFirstTurn();
        else
            selectCardsForBeatTurn();
    }

    void AIPlayer::selectCardsForFirstTurn() {
        int i = 0;

        while (i < cards.size() && *cards.at(0) == *cards.at(i)) {
            selectedCards.push_back(cards.at(i));

```

```

        ++i;
    }
}

void AIPlayer::selectCardsForBeatTurn() {
    int i = 0;
    while(i < cards.size()) {
        if (*cardsToBeat.at(0) > *cards.at(i) || *cardsToBeat.at(0) ==
*cards.at(i)) {
            ++i;
            continue;
        }
        selectCardsToFightBack(cards.at(i));
        if (!selectedCards.empty())
            break;

        ++i;
    }
}

bool HumanPlayer::selectCardsForTurn(Card* selectedCard) {
    if (cardsToBeat.empty()) {
        for (auto card : cards) {
            if (*card == *selectedCard)
                selectedCards.push_back(card);
        }

        return true;
    }
}

```



```

        if (!(*selectedCard > *cardsToBeat.at(0)))
            return false;

        selectCardsToFightBack(selectedCard);

        return selectedCards.size() > 0;
    }

```

```

bool HumanPlayer::canFightBack() {
    if (cardsToBeat.empty())
        return true;

    for (auto card : cards) {
        if (!(*card > *cardsToBeat.front()))
            continue;

        selectCardsToFightBack(card);

        if (!selectedCards.empty()) {
            selectedCards.clear();
            return true;
        }
    }

    return false;
}

```

## **PresidentForm.h**

```

#pragma once

```

```

#include "PresidentI.h"

namespace Lab6 {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for PresidentForm
    /// </summary>
    public ref class PresidentForm : public System::Windows::Forms::Form
    {
    private:
        PresidentI* president;

        void setCardsClickHandlers();
        Void Card_Click(Object^ sender, EventArgs^ e);
    public:
        PresidentForm(void);

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~PresidentForm();
    }
}

```

```

private: System::Windows::Forms::FlowLayoutPanel^ humanCards;
private: System::Windows::Forms::FlowLayoutPanel^ AI1Cards;
private: System::Windows::Forms::FlowLayoutPanel^ AI2Cards;
private: System::Windows::Forms::FlowLayoutPanel^ AI3Cards;
private: System::Windows::Forms::FlowLayoutPanel^ cardsOnDesk;
private: System::Windows::Forms::Label^ resultLabel;
private: System::Windows::Forms::Button^ restartBtn;

```

```
protected:
```

```
protected:
```

```
private:
```

```
    /// <summary>
```

```
    /// Required designer variable.
```

```
    /// </summary>
```

```
    System::ComponentModel::Container ^components;
```

```
#pragma region Windows Form Designer generated code
```

```
    /// <summary>
```

```
    /// Required method for Designer support - do not modify
```

```
    /// the contents of this method with the code editor.
```

```
    /// </summary>
```

```
    void InitializeComponent(void)
```

```
    {
```

```
        this->humanCards = (gcnew
```

```
System::Windows::Forms::FlowLayoutPanel());
```

```
        this->AI1Cards = (gcnew
```

```
System::Windows::Forms::FlowLayoutPanel());
```

```

        this->AI2Cards = (gcnew
System::Windows::Forms::FlowLayoutPanel());
        this->AI3Cards = (gcnew
System::Windows::Forms::FlowLayoutPanel());
        this->cardsOnDesk = (gcnew
System::Windows::Forms::FlowLayoutPanel());
        this->resultLabel = (gcnew
System::Windows::Forms::Label());
        this->restartBtn = (gcnew
System::Windows::Forms::Button());
        this->SuspendLayout();
        //
        // humanCards
        //
        this->humanCards->Location =
System::Drawing::Point(148, 797);
        this->humanCards->Name = L"humanCards";
        this->humanCards->Size = System::Drawing::Size(1628,
261);
        this->humanCards->TabIndex = 0;
        //
        // AI1Cards
        //
        this->AI1Cards->FlowDirection =
System::Windows::Forms::FlowDirection::TopDown;
        this->AI1Cards->Location = System::Drawing::Point(12,
12);
        this->AI1Cards->Name = L"AI1Cards";
        this->AI1Cards->Size = System::Drawing::Size(130, 1031);
        this->AI1Cards->TabIndex = 1;

```

```

//
// AI2Cards
//
this->AI2Cards->Location = System::Drawing::Point(148,
12);

this->AI2Cards->Name = L"AI2Cards";
this->AI2Cards->Size = System::Drawing::Size(1628, 173);
this->AI2Cards->TabIndex = 2;
//
// AI3Cards
//
this->AI3Cards->FlowDirection =
System::Windows::Forms::FlowDirection::TopDown;
this->AI3Cards->Location = System::Drawing::Point(1782,
12);

this->AI3Cards->Name = L"AI3Cards";
this->AI3Cards->Size = System::Drawing::Size(130, 1031);
this->AI3Cards->TabIndex = 3;
//
// cardsOnDesk
//
this->cardsOnDesk->Location =
System::Drawing::Point(633, 374);
this->cardsOnDesk->Name = L"cardsOnDesk";
this->cardsOnDesk->Size = System::Drawing::Size(735,
228);

this->cardsOnDesk->TabIndex = 4;
//
// resultLabel
//

```

```

        this->resultLabel->AutoSize = true;
        this->resultLabel->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 13.8F,
static_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold |
System::Drawing::FontStyle::Italic)),
        System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
        this->resultLabel->Location =
System::Drawing::Point(1180, 252);
        this->resultLabel->Name = L"resultLabel";
        this->resultLabel->Size = System::Drawing::Size(119, 29);
        this->resultLabel->TabIndex = 5;
        this->resultLabel->Text = L"Your turn";
        //
        // restartBtn
        //
        this->restartBtn->Enabled = false;
        this->restartBtn->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 13.8F,
static_cast<System::Drawing::FontStyle>((System::Drawing::FontStyle::Bold |
System::Drawing::FontStyle::Italic)),
        System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
        this->restartBtn->Location = System::Drawing::Point(1185,
295);
        this->restartBtn->Name = L"restartBtn";
        this->restartBtn->Size = System::Drawing::Size(137, 54);
        this->restartBtn->TabIndex = 6;
        this->restartBtn->Text = L"Restart";
        this->restartBtn->UseVisualStyleBackColor = true;

```

```

        this->restartBtn->Visible = false;
        this->restartBtn->Click += gcnew
System::EventHandler(this, &PresidentForm::restartBtn_Click);
        //
        // PresidentForm
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(8,
16);

        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;

        this->BackColor = System::Drawing::Color::OliveDrab;
        this->ClientSize = System::Drawing::Size(1924, 1055);
        this->Controls->Add(this->restartBtn);
        this->Controls->Add(this->resultLabel);
        this->Controls->Add(this->cardsOnDesk);
        this->Controls->Add(this->AI3Cards);
        this->Controls->Add(this->AI2Cards);
        this->Controls->Add(this->AI1Cards);
        this->Controls->Add(this->humanCards);
        this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedSingle;

        this->MaximizeBox = false;
        this->Name = L"PresidentForm";
        this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;

        this->Text = L"President";
        this->WindowState =
System::Windows::Forms::FormWindowState::Maximized;

        this->ResumeLayout(false);
        this->PerformLayout();

```

```

        }
#pragma endregion

        private: Void restartBtn_Click(Object^ sender, EventArgs^ e);
    };
}

```

## **PresidentI.h**

```

#pragma once

#include "../Lab6_code/Player.h"
#include <string>
#include <Windows.h>

using namespace System;
using namespace System::Drawing;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Collections::Generic;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Reflection;
using namespace std;

#define AMOUNT_OF_AI_PLAYERS 3
#define CARD_WIDTH 72
#define CARD_HEIGHT 130
#define PADDING 5
#define PAUSE 2000

```



```

ref class cardsContainers {
public:
    static List<FlowLayoutPanel^>^ containers = gcnew
List<FlowLayoutPanel^>;
};

ref class label {
public:
    static Label^ resultLabel;
};

enum containersNumbers{
    HUMAN_CONTAINER,
    AI1_CONTAINER,
    AI2_CONTAINER,
    AI3_CONTAINER,
    CARDS_ON_DESK_CONTAINER,
};

enum WinResults {
    NO_WIN,
    HUMAN_WIN,
    AI_WIN
};

ref class cardInfo {
public:
    Suits suit;
    CardNames name;
};

```

```

class PresidentI {
    Deck* deck;
    vector<Card*> cardsOnDesk;

    vector<AIPlayer*> AIPlayers;
    HumanPlayer* humanPlayer;
    Player* lastActingPlayer;

    void distributeCards();

    WinResults checkWinCondition() const;

    void displayCards(FlowLayoutPanel^ cardContainer, vector<Card*>
cards, RotateFlipType rotationDegree, bool hide = false);
    void clearCardsContainer(FlowLayoutPanel^ cardsContainer);
    string getCardPath(Card* card);
    PictureBox^ getPicture(String^ cardPath, RotateFlipType
rotationDegree);
    Card* getHumanCardFromCardInfo(Object^ info);
    WinResults makeAIPlayersMoves();
    void changeResultLabelText(String^ newText);
    void disableHumanPlayerCards();
    void enableHumanPlayerCards();
public:
    PresidentI();

    void displayAllCards();

    void setCardsOnDesk(vector<Card*> cards);

```

```
void removeCards(FlowLayoutPanel^ cardsContainer, vector<Card*>
cards);
```

```
WinResults makeHumanMove(Object^ info);
```

```
void restart();
```

```
~PresidentI();
```

```
};
```

### **PresidentForm.cpp**

```
#include "PresidentForm.h"
```

```
#include <stdlib.h>
```

```
#include <ctime>
```

```
using namespace Lab6;
```

```
PresidentForm::PresidentForm(void) {
```

```
    InitializeComponent();
```

```
    const int maxAmountOfCards = 13;
```

```
    int maxWidth = (CARD_WIDTH + PADDING) * maxAmountOfCards;
```

```
    int maxHeight = CARD_HEIGHT;
```

```
    humanCards->Height = maxHeight+60;
```

```
    humanCards->Width = maxWidth;
```

```
    humanCards->Location = Point(((this->Width - humanCards->Width) / 2,
this->Height - humanCards->Height));
```

```
AI1Cards->Height = maxWidth/2 + CARD_WIDTH - 6*PADDING;  
AI1Cards->Width = maxHeight*2;  
AI1Cards->Location = Point(0, (this->Height - AI1Cards->Height)/2);
```

```
AI2Cards->Height = maxHeight;  
AI2Cards->Width = maxWidth;  
AI2Cards->Location = Point((this->Width - AI2Cards->Width) / 2, 0);
```

```
AI3Cards->Height = maxWidth / 2 + CARD_WIDTH;  
AI3Cards->Width = maxHeight * 2;  
AI3Cards->Location = Point(this->Width - AI3Cards->Width +  
CARD_WIDTH, (this->Height - AI3Cards->Height) / 2);
```

```
cardsOnDesk->Height = maxHeight;  
cardsOnDesk->Width = (CARD_WIDTH + PADDING) * 4;  
cardsOnDesk->Location = Point((this->Width - cardsOnDesk->Width) /  
2, (this->Height - cardsOnDesk->Height) / 2);
```

```
srand(time(nullptr));  
president = new PresidentI;
```

```
cardsContainers::containers->Add(humanCards);  
cardsContainers::containers->Add(AI1Cards);  
cardsContainers::containers->Add(AI2Cards);  
cardsContainers::containers->Add(AI3Cards);  
cardsContainers::containers->Add(cardsOnDesk);
```

```
label::resultLabel = resultLabel;
```

```

        president->displayAllCards();
        setCardsClickHandlers();
    }

void PresidentForm::setCardsClickHandlers() {
    for each (Control ^ control in humanCards->Controls) {
        PictureBox^ pictureBoxCard =
dynamic_cast<PictureBox^>(control);
        pictureBoxCard->Click += gcnew System::EventHandler(this,
&PresidentForm::Card_Click);
    }
}

Void PresidentForm::Card_Click(Object^ sender, EventArgs^ e) {
    PictureBox^ clickedPictureBox = dynamic_cast<PictureBox^>(sender);
    auto result = president->makeHumanMove(clickedPictureBox->Tag);
    if (result == AI_WIN || result == HUMAN_WIN) {
        restartBtn->Enabled = true;
        restartBtn->Visible = true;
    }
}

Void PresidentForm::restartBtn_Click(Object^ sender, EventArgs^ e) {
    president->restart();
    setCardsClickHandlers();
    restartBtn->Enabled = false;
    restartBtn->Visible = false;
}

PresidentForm::~~PresidentForm() {

```

```

        if (components)
            delete components;

        delete president;
    }

```

### **PresidentI.cpp**

```

#include "PresidentI.h"

PresidentI::PresidentI() {
    deck = new Deck();
    humanPlayer = new HumanPlayer;

    for (int i = 0; i < AMOUNT_OF_AI_PLAYERS; ++i) {
        AIPlayer* bot = new AIPlayer;
        AIPlayers.push_back(bot);
    }

    distributeCards();
}

void PresidentI::distributeCards() {
    const int deckSize = 52;

    for (int i = 0; i < deckSize / (AMOUNT_OF_AI_PLAYERS + 1); ++i) {
        humanPlayer->addCard(deck->getCard());
        for (int j = 0; j < AMOUNT_OF_AI_PLAYERS; ++j)
            AIPlayers.at(j)->addCard(deck->getCard());
    }
}

```

```
}
```

```
WinResults PresidentI::checkWinCondition() const {
```

```
    if (humanPlayer->hasWon())
```

```
        return HUMAN_WIN;
```

```
    for (auto AIPlayer : AIPlayers) {
```

```
        if (AIPlayer->hasWon())
```

```
            return AI_WIN;
```

```
    }
```

```
    return NO_WIN;
```

```
}
```

```
void PresidentI::displayAllCards() {
```

```
    displayCards(cardsContainers::containers[0], humanPlayer->getCards(),  
RotateFlipType::RotateNoneFlipNone);
```

```
    for (int i = 0; i < AMOUNT_OF_AI_PLAYERS; ++i) {
```

```
        if(i%2 == 1)
```

```
            displayCards(cardsContainers::containers[i + 1],  
AIPlayers.at(i)->getCards(), RotateFlipType::RotateNoneFlipNone, true);
```

```
        else
```

```
            displayCards(cardsContainers::containers[i + 1],  
AIPlayers.at(i)->getCards(), RotateFlipType::Rotate90FlipNone, true);
```

```
    }
```

```
    displayCards(cardsContainers::containers[CARDS_ON_DESK_CONTAINER], cardsOnDesk, RotateFlipType::RotateNoneFlipNone);
```

```
}
```

```

void PresidentI::displayCards(FlowLayoutPanel^ cardsContainer,
vector<Card*> cards, RotateFlipType rotationDegree, bool hide) {
    clearCardsContainer(cardsContainer);

    for (auto card : cards) {
        String^ cardPath;
        if (hide)
            cardPath = "cards\\back_side.png";
        else
            cardPath = gcnew String(getCardPath(card).c_str());
        PictureBox^ picture = getPicture(cardPath, rotationDegree);
        cardInfo^ info = gcnew cardInfo;
        info->name = card->getName();
        info->suit = card->getSuit();
        picture->Tag = info;
        cardsContainer->Controls->Add(picture);
        Application::DoEvents();
    }
}

```

```

string PresidentI::getCardPath(Card* card) {
    string path = "cards\\";

    switch (card->getName()) {
        case TWO:
            path += "2";
            break;
        case THREE:
            path += "3";

```



```
        break;
case FOUR:
    path += "4";
    break;
case FIVE:
    path += "5";
    break;
case SIX:
    path += "6";
    break;
case SEVEN:
    path += "7";
    break;
case EIGHT:
    path += "8";
    break;
case NINE:
    path += "9";
    break;
case TEN:
    path += "10";
    break;
case JACK:
    path += "jack";
    break;
case QUEEN:
    path += "queen";
    break;
case KING:
    path += "king";
```

```

        break;
    case ACE:
        path += "ace";
        break;
    }

    path += "_of_";

    switch (card->getSuit()) {
    case CLUBS:
        path += "clubs";
        break;
    case DIAMONDS:
        path += "diamonds";
        break;
    case HEARTS:
        path += "hearts";
        break;
    case SPADES:
        path += "spades";
        break;
    }

    path += ".png";
    return path;
}

```

```

PictureBox^ PresidentI::getPicture(String^ cardPath, RotateFlipType
rotationDegree) {
    PictureBox^ pictureBoxCard = gcnew PictureBox();

```

```

        pictureBoxCard->Image = Image::FromFile(cardPath);
        pictureBoxCard->Image->RotateFlip(rotationDegree);
        pictureBoxCard->SizeMode = PictureBoxSizeMode::Zoom;

        if (rotationDegree == RotateFlipType::Rotate90FlipNone)
            pictureBoxCard->Size = Size(CARD_HEIGHT,
CARD_WIDTH);
        else
            pictureBoxCard->Size = Size(CARD_WIDTH,
CARD_HEIGHT);

        pictureBoxCard->Margin = Padding(0, 0, PADDING, 0);

        return pictureBoxCard;
    }

    void PresidentI::clearCardsContainer(FlowLayoutPanel^ cardsContainer) {
        while (cardsContainer->Controls->Count > 0)
            cardsContainer->Controls->Remove(cardsContainer-
>Controls[0]);
    }

    Card* PresidentI::getHumanCardFromCardInfo(Object^ info) {
        cardInfo^ clickedCardInfo = dynamic_cast<cardInfo^>(info);
        vector<Card*> cards = humanPlayer->getCards();
        for (auto card : cards) {
            if (card->getName() == clickedCardInfo->name && card-
>getSuit() == clickedCardInfo->suit)
                return card;
        }
    }

```

```

        return nullptr;
    }

void PresidentI::setCardsOnDesk(vector<Card*> cards) {
    cardsOnDesk.clear();
    copy(cards.begin(), cards.end(), back_inserter(cardsOnDesk));
    displayCards(cardsContainers::containers[CARDS_ON_DESK_CONTA
INNER], cardsOnDesk, RotateFlipType::RotateNoneFlipNone);
}

WinResults PresidentI::makeHumanMove(Object^ info) {
    Card* selectedCard = getHumanCardFromCardInfo(info);
    if (!humanPlayer->selectCardsForTurn(selectedCard)) {
        changeResultLabelText("You can't make turn using these cards");
        return NO_WIN;
    }
    disableHumanPlayerCards();
    vector<Card*> playedCards(humanPlayer->makeTurn());
    setCardsOnDesk(playedCards);
    removeCards(cardsContainers::containers[HUMAN_CONTAINER],
playedCards);
    lastActingPlayer = humanPlayer;

    if (checkWinCondition() == HUMAN_WIN) {
        changeResultLabelText("You won!");
        return HUMAN_WIN;
    }

    Sleep(PAUSE);
    if (makeAIPlayersMoves() == AI_WIN)

```

```

        return AI_WIN;
    }

    WinResults PresidentI::makeAIPlayersMoves() {
        for (int i = 0; i < AIPlayers.size(); ++i) {
            if (lastActingPlayer == AIPlayers.at(i)) {
                changeResultLabelText("No one could beat " + (i+1) + " AI
player's cards!");

                setCardsOnDesk(vector<Card*>{ });
                Sleep(PAUSE / 2);
            }

            AIPlayers.at(i)->setCardsToBeat(cardsOnDesk);
            AIPlayers.at(i)->selectCardsForTurn();
            vector<Card*> playedCards(AIPlayers.at(i)->makeTurn());

            if (!playedCards.empty()) {
                setCardsOnDesk(playedCards);
                changeResultLabelText("AI player " + (i + 1) + " made
turn");

                lastActingPlayer = AIPlayers.at(i);
            } else {
                changeResultLabelText("AI player " + (i + 1) + " can't make
turn");
            }

            removeCards(cardsContainers::containers[i+1], playedCards);

            if (checkWinCondition() == AI_WIN) {
                changeResultLabelText("AI player " + (i + 1) + " won!");
            }
        }
    }

```

```

        return AI_WIN;
    }
    Sleep(PAUSE);
}

if (lastActingPlayer == humanPlayer) {
    changeResultLabelText("No one could beat your cards!");
    setCardsOnDesk(vector<Card*>{ });
    Sleep(PAUSE / 2);
}
humanPlayer->setCardsToBeat(cardsOnDesk);
if (!humanPlayer->canFightBack()) {
    changeResultLabelText("You can't beat these cards");
    Sleep(PAUSE);
    if (makeAIPlayersMoves() == AI_WIN)
        return AI_WIN;
} else {
    changeResultLabelText("Your turn");
}
enableHumanPlayerCards();
}

void PresidentI::changeResultLabelText(String^ newText) {
    label::resultLabel->Text = newText;
    Application::DoEvents();
}

void PresidentI::removeCards(FlowLayoutPanel^ cardsContainer,
vector<Card*> cards) {
    int i = 0;

```

```

        while (i < cardsContainer->Controls->Count) {
            PictureBox^ pictureBoxCard =
dynamic_cast<PictureBox^>(cardsContainer->Controls[i]);
            cardInfo^ info = dynamic_cast<cardInfo^>(pictureBoxCard-
>Tag);

            for (auto card : cards) {
                if (card->getName() == info->name && card->getSuit() ==
info->suit) {
                    Control^ element = cardsContainer->Controls[i];
                    cardsContainer->Controls->Remove(element);
                    --i;
                }
            }
            ++i;
        }
    }
}

```

```

void PresidentI::disableHumanPlayerCards() {
    cardsContainers::containers[HUMAN_CONTAINER]->Enabled = false;

    for each (Control^ control in
cardsContainers::containers[HUMAN_CONTAINER]->Controls)
        control->Enabled = false;

    Application::DoEvents();
}

```

```

void PresidentI::enableHumanPlayerCards() {
    cardsContainers::containers[HUMAN_CONTAINER]->Enabled = true;

```

```

        for each (Control ^ control in
cardsContainers::containers[HUMAN_CONTAINER]->Controls)
            control->Enabled = true;

    Application::DoEvents();
}

```

```

void PresidentI::restart() {
    deck->fill();
    deck->shuffle();

    humanPlayer->clearAllCards();
    for (auto player : AIPlayers)
        player->clearAllCards();

    cardsOnDesk.clear();
    distributeCards();
    displayAllCards();
    changeResultLabelText("Your turn");
    enableHumanPlayerCards();
}

```

```

PresidentI::~~PresidentI() {
    delete deck;
    delete humanPlayer;
}

```



### **main.cpp**

```
#include "PresidentForm.h"

using namespace System;
using namespace Windows::Forms;
using namespace Lab6;

[STAThreadAttribute]
int main() {
    Application::SetCompatibleTextRenderingDefault(false);
    Application::EnableVisualStyles();

    PresidentForm^ mainForm = gcnew PresidentForm;
    Application::Run(mainForm);

    return 0;
}
```

### **Lab6\_tests.cpp**

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../Lab6_code/Cards.h"
#include "../Lab6_code/Player.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace Lab6tests
{
    bool areEqual(Card* card1, Card* card2) {
```

```

        return card1->getName() == card2->getName() && card1-
>getSuit() == card2->getSuit();
    }

```

```

bool contains(vector<Card*> cards, Card* card) {
    for (auto availableCard : cards) {
        if (areEqual(card, availableCard))
            return true;
    }
    return false;
}

```

```

bool areEqual(vector<Card*> cards1, vector<Card*> cards2) {
    if (cards1.size() != cards2.size())
        return false;

    for (auto card1 : cards1) {
        if (!contains(cards2, card1))
            return false;
    }
    return true;
}

```

```

TEST_CLASS(CardTests)
{
private:
    Suits suit;
    CardNames name;
    Card* card;

public:

```

```

CardTests() {
    suit = HEARTS;
    name = TEN;
    card = new Card(suit, name);
}

TEST_METHOD(getSuitTest)
{
    Assert::IsTrue(suit == card->getSuit());
}

TEST_METHOD(getNameTest)
{
    Assert::IsTrue(name == card->getName());
}

TEST_METHOD(equalOperatorTest1)
{
    Card testCard(suit, name);
    Assert::IsTrue(testCard == *card);
}

TEST_METHOD(equalOperatorTest2)
{
    Card testCard(suit, JACK);
    Assert::IsFalse(testCard == *card);
}

TEST_METHOD(greaterOperatorTest1)
{

```

```

        Card testCard(suit, name);
        Assert::IsFalse(*card > testCard);
    }

TEST_METHOD(greaterOperatorTest2)
{
    Card testCard(DIAMONDS, name);
    Assert::IsFalse(*card > testCard);
}

TEST_METHOD(greaterOperatorTest3)
{
    Card testCard(suit, JACK);
    Assert::IsFalse(*card > testCard);
}

TEST_METHOD(greaterOperatorTest4)
{
    Card testCard(DIAMONDS, JACK);
    Assert::IsFalse(*card > testCard);
}

TEST_METHOD(greaterOperatorTest5)
{
    Card testCard(suit, FIVE);
    Assert::IsTrue(*card > testCard);
}

TEST_METHOD(greaterOperatorTest6)
{

```

```

        Card testCard(DIAMONDS, FIVE);
        Assert::IsTrue(*card > testCard);
    }

    ~CardTests() {
        delete card;
    }
};

TEST_CLASS(HumanPlayerTest)
{
private:
    HumanPlayer* player;
    vector<Card*> playerCards;
public:
    HumanPlayerTest() {
        player = new HumanPlayer;
        playerCards.push_back(new Card(HEARTS, TEN));
        playerCards.push_back(new Card(SPADES, TEN));
        playerCards.push_back(new Card(DIAMONDS, TEN));
        playerCards.push_back(new Card(HEARTS, JACK));
        playerCards.push_back(new Card(SPADES, JACK));

        for (auto card : playerCards)
            player->addCard(card);
    }

    TEST_METHOD(getCardsTest) {
        vector<Card*> cards = player->getCards();
    }
};

```

```

        Assert::IsTrue(areEqual(cards, playerCards));
    }

TEST_METHOD(addCardTest) {
    Card* newCard = new Card(SPADES, ACE);
    player->addCard(newCard);
    Assert::IsTrue(contains(player->getCards(), newCard));
}

TEST_METHOD(selectCardsForTurnTest1) {
    Card* cardToSelect = new Card(HEARTS, TEN);

    player->selectCardsForTurn(cardToSelect);
    vector<Card*> selectedCards = player->getSelectedCards();

    Assert::IsTrue(selectedCards.size() == 3);
    Assert::IsTrue(contains(selectedCards, cardToSelect));
    Assert::IsTrue(contains(selectedCards, new Card(SPADES,
TEN)));

    Assert::IsTrue(contains(selectedCards, new
Card(DIAMONDS, TEN)));
}

TEST_METHOD(selectCardsForTurnTest2) {
    vector<Card*> cardsToBeat = { new Card(SPADES,
THREE) };

    Card* cardToSelect = new Card(HEARTS, TEN);

    player->setCardsToBeat(cardsToBeat);
    bool res = player->selectCardsForTurn(cardToSelect);

```

```

        vector<Card*> selectedCards = player->getSelectedCards();

        Assert::IsTrue(res);
        Assert::IsTrue(selectedCards.size() == 1);
        Assert::IsTrue(contains(selectedCards,                new
Card(DIAMONDS, TEN)));
    }

TEST_METHOD(selectCardsForTurnTest3) {
    vector<Card*> cardsToBeat = { new Card(SPADES, TWO)

};

    Card* cardToSelect = new Card(HEARTS, TEN);

    player->setCardsToBeat(cardsToBeat);
    bool res = player->selectCardsForTurn(cardToSelect);
    vector<Card*> selectedCards = player->getSelectedCards();

    Assert::IsFalse(res);
    Assert::IsTrue(selectedCards.empty());
}

TEST_METHOD(canFightBack1) {
    vector<Card*> cardsToBeat = { new Card(SPADES, TWO)

};

    player->setCardsToBeat(cardsToBeat);

    Assert::IsFalse(player->canFightBack());
}

```

```

TEST_METHOD(canFightBack2) {
    vector<Card*> cardsToBeat = { new Card(SPADES,
THREE) };

    player->setCardsToBeat(cardsToBeat);

    Assert::IsTrue(player->canFightBack());
}

TEST_METHOD(canFightBack3) {
    vector<Card*> cardsToBeat = { new Card(HEARTS,
THREE), new Card(CLUBS, THREE), new Card(SPADES, THREE), new
Card(DIAMONDS, THREE) };

    player->setCardsToBeat(cardsToBeat);

    Assert::IsFalse(player->canFightBack());
}

TEST_METHOD(makeTurnTest) {
    Card* cardToSelect = new Card(HEARTS, TEN);

    player->selectCardsForTurn(cardToSelect);
    vector<Card*> turnResult = player->makeTurn();
    vector<Card*> selectedCards = player->getSelectedCards();

    Assert::IsTrue(selectedCards.empty());
    Assert::IsTrue(turnResult.size() == 3);
    Assert::IsTrue(contains(turnResult, cardToSelect));
    Assert::IsTrue(contains(turnResult, new Card(SPADES,
TEN)));
}

```



```

        Assert::IsTrue(contains(turnResult, new Card(DIAMONDS,
TEN)));
    }

    ~HumanPlayerTest() {
        delete player;
        for (auto card : playerCards)
            delete card;
    }
};

TEST_CLASS(AIPlayerTest)
{
private:
    AIPlayer* player;
    vector<Card*> playerCards;
public:
    AIPlayerTest() {
        player = new AIPlayer;
        playerCards.push_back(new Card(HEARTS, TEN));
        playerCards.push_back(new Card(SPADES, TEN));
        playerCards.push_back(new Card(DIAMONDS, TEN));
        playerCards.push_back(new Card(HEARTS, JACK));
        playerCards.push_back(new Card(SPADES, JACK));
        playerCards.push_back(new Card(SPADES, QUEEN));

        for (auto card : playerCards)
            player->addCard(card);
    }
}

```

```

TEST_METHOD(selectCardsForTurnTest) {
    player->selectCardsForTurn();
    vector<Card*> selectedCards = player->getSelectedCards();

    Assert::IsTrue(selectedCards.size() == 3);
    Assert::IsTrue(contains(selectedCards, new Card(HEARTS,
TEN)));

    Assert::IsTrue(contains(selectedCards, new Card(SPADES,
TEN)));

    Assert::IsTrue(contains(selectedCards, new
Card(DIAMONDS, TEN)));
}

TEST_METHOD(makeMoveTest1) {
    vector<Card*> cardsToBeat = { new Card(HEARTS,
THREE) };

    player->setCardsToBeat(cardsToBeat);
    player->selectCardsForTurn();
    vector<Card*> selectedCards = player->getSelectedCards();

    Assert::IsTrue(selectedCards.size() == 1);
    Assert::IsTrue(contains(selectedCards, new
Card(DIAMONDS, TEN)));
}

TEST_METHOD(makeMoveTest2) {
    vector<Card*> cardsToBeat = { new Card(HEARTS,
THREE), new Card(CLUBS, THREE) };

    player->setCardsToBeat(cardsToBeat);
    player->selectCardsForTurn();

```

```

        vector<Card*> selectedCards = player->getSelectedCards();

        Assert::IsTrue(selectedCards.size() == 2);
        Assert::IsTrue(contains(selectedCards, new
Card(DIAMONDS, TEN)));
        Assert::IsTrue(contains(selectedCards, new Card(SPADES,
TEN)));
    }

    TEST_METHOD(makeMoveTest3) {
        vector<Card*> cardsToBeat = { new Card(HEARTS,
THREE), new Card(CLUBS, THREE), new Card(SPADES, THREE) };
        player->setCardsToBeat(cardsToBeat);
        player->selectCardsForTurn();
        vector<Card*> selectedCards = player->getSelectedCards();

        Assert::IsTrue(selectedCards.size() == 3);
        Assert::IsTrue(contains(selectedCards, new Card(HEARTS,
TEN)));
        Assert::IsTrue(contains(selectedCards, new Card(SPADES,
TEN)));
        Assert::IsTrue(contains(selectedCards, new
Card(DIAMONDS, TEN)));
    }

    TEST_METHOD(makeMoveTest4) {
        vector<Card*> cardsToBeat = { new Card(HEARTS,
THREE), new Card(CLUBS, THREE), new Card(SPADES, THREE), new
Card(DIAMONDS, THREE) };
        player->setCardsToBeat(cardsToBeat);

```

```

        player->selectCardsForTurn();
        vector<Card*> selectedCards = player->getSelectedCards();

        Assert::IsTrue(selectedCards.empty());
    }

    TEST_METHOD(makeMoveTest5) {
        vector<Card*> cardsToBeat = { new Card(HEARTS,
JACK) };

        player->setCardsToBeat(cardsToBeat);
        player->selectCardsForTurn();
        vector<Card*> selectedCards = player->getSelectedCards();

        Assert::IsTrue(selectedCards.size() == 1);
        Assert::IsTrue(contains(selectedCards, new Card(SPADES,
QUEEN))));
    }

    TEST_METHOD(makeMoveTest6) {
        vector<Card*> cardsToBeat = { new Card(HEARTS,
TWO) };

        player->setCardsToBeat(cardsToBeat);
        player->selectCardsForTurn();
        vector<Card*> selectedCards = player->getSelectedCards();

        Assert::IsTrue(selectedCards.empty());
    }

    TEST_METHOD(makeMoveTest7) {
        player->setCardsToBeat(vector<Card*> {});

```

```

        player->selectCardsForTurn();
        vector<Card*> selectedCards = player->getSelectedCards();

        Assert::IsTrue(selectedCards.size() == 3);
        Assert::IsTrue(contains(selectedCards, new Card(HEARTS,
TEN)));

        Assert::IsTrue(contains(selectedCards, new Card(SPADES,
TEN)));

        Assert::IsTrue(contains(selectedCards, new
Card(DIAMONDS, TEN)));
    }

    TEST_METHOD(clearAllCardsTest) {
        player->clearAllCards();
        vector<Card*> selectedCards = player->getSelectedCards();
        vector<Card*> cards = player->getCards();

        Assert::IsTrue(selectedCards.empty());
        Assert::IsTrue(cards.empty());
    }

    ~AIPlayerTest() {
        delete player;
        for (auto card : playerCards)
            delete card;
    }
};
}

```

### 3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

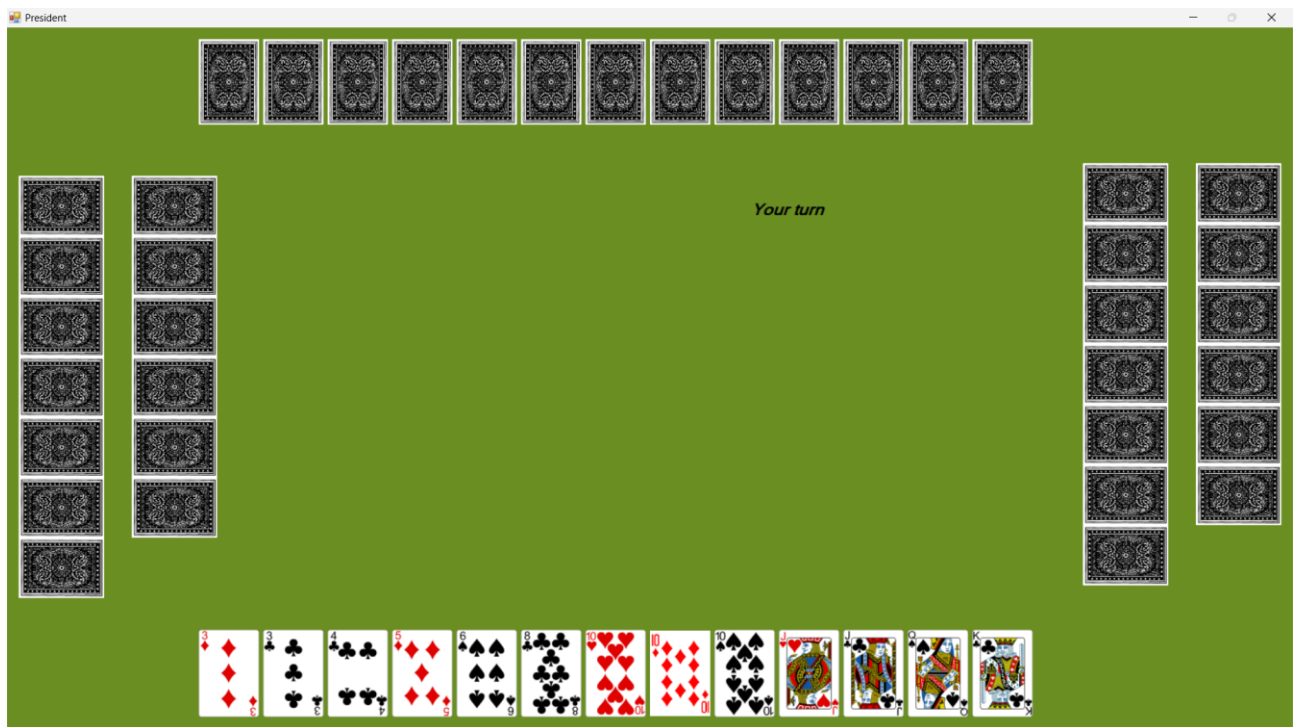


Рисунок 3.1 – Початок гри

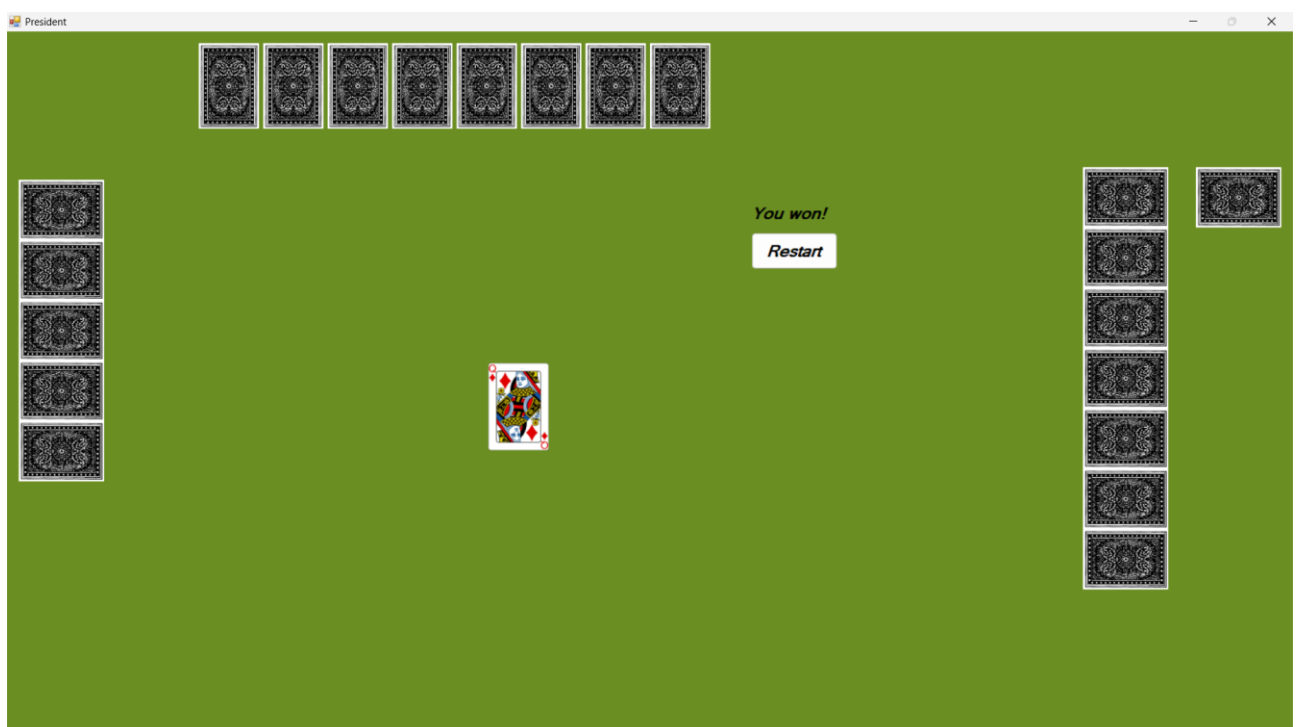


Рисунок 3.2 – Перемога

## ВИСНОВОК

В рамках даної лабораторної роботи я реалізував карткову гру Багач засобами мови програмування C++ із використанням Windows Forms для створення користувацького графічного інтерфейсу. Гра реалізована для 4 гравців: 3-х комп'ютерів та 1 людини. При тестуванні я переконався, що все працює правильно.

## КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 31.12.2023 включно максимальний бал дорівнює – 5. Після 31.12.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація – 75%;
- робота з гіт – 20%;
- висновок – 5%.

+1 додатковий бал можна отримати за реалізацію анімації ігрових процесів (жеребкування, роздачі карт, анімацію ходів тощо).

+1 додатковий бал можна отримати за виконання та захист роботи до 24.12.2023.