Міністерство освіти і науки України
НаціональнийтехнічнийуніверситетУкраїни«Київськийполітехнічнийінст
итутіменіІгоряСікорського"
Факультет інформатики та обчислювальноїтехніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 6 з дисципліни
«Проектування алгоритмів»

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

Виконав(ла) *ІП-13 Макарчук Лідія Олександрівна* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(прізвище, ім'я, по батькові)

Київ2022

ЗМІСТ

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 віграх 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 https://game-wiki.guru/published/igryi/yaczzyi.html	З елементами випадковості
2	Лудо http://www.iggamecenter.com/info/ru/ludo.html	З елементами випадковості
3	Генерал http://www.rules.net.ru/kost.php?id=7	З елементами випадковості

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

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

3 ВИКОНАННЯ

№	Варіант	Тип гри
16	Війна з ботами https://www.durbetsel.ru/2_voina_s_botami.htm	Карткові ігри

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

//Модуль GameAlgorithm

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GameAlgo
{
   public class GameAlgorithm
   {
      public const int CARDS_IN_HAND = 10;
      public const int BOT_NUMBER_TO_WIN = 10;
      public enum Mode
      {
         Easy,
         Medium,
         Difficult
      }
      Mode _gameMode;
      List<Card> deck;
      Player player1;
      Player player2;//IIII
      int _deadBotNumber = 0;
      int _roundNumber = 1;

      public List<Card?> Hand1 => player1.Hand;
      public List<Card?> Hand2 => player2.Hand;
      public Player p1 => player1;
      public Player p2 => player2;
      public int Score1 => player1.Score;
      public int Score2 => player2.Score;
      public bool IsDeckEmpty() => deck.Count == 0;
      public int RoundNumber
      {
         get => _roundNumber;
         //set => _roundNumber = value;
      }
      public int DeadBotNumber
      {
         get => _deadBotNumber;
        // set => _deadBotNumber = value;
      }

      public GameAlgorithm(int mode)
```

```
{
    _gameMode = (Mode)mode;
    InitiateGame();
}
public void NewRound()
{
    _roundNumber++;
    _deadBotNumber= 0;
    player1.UnchooseAllCards();
    player2.UnchooseAllCards();

    deck = Card.GenerateDeck();
    player1.Hand = HandCards();
    player2.Hand = HandCards();
    //DealCardsForPlayers();
}
private void InitiateGame()
{
    deck = Card.GenerateDeck();
    player1 = new Player(HandCards());
    player2 = new Player(HandCards());
}
private List<Card?> HandCards()
{
    List<Card?> hand = new List<Card?>();
    Random rnd = new Random();
    int index;
    for (int i=0; i< CARDS_IN_HAND; i++)
    {
        index = rnd.Next(0, deck.Count);
        hand.Add(deck[index]);
        deck.RemoveAt(index);
    }
    return hand;
}
public void AcceptMove(int? weaponIndex, int? armourIndex)
{
    player1.BotWeaponIndex = weaponIndex;
    player1.BotArmourIndex = armourIndex;
    if (weaponIndex == null)//if player1 skips move
    {
        SkipMove(1);
        NewRound();
    }
    else
        Response(player1.BotArmour);
}
public void AcceptResponse(int? weaponIndex, int? armourIndex)
{
    player1.BotWeaponIndex = weaponIndex;
    player1.BotArmourIndex = armourIndex;
    if (weaponIndex != null)
    {
        _deadBotNumber++;
    }
    player1.DeleteCardsAfterMove();
}
public bool MakeMove(out int? weaponIndex, out int? armourIndex)
{
    weaponIndex = player2.BotWeaponIndex;
    armourIndex = player2.BotArmourIndex;
    if (weaponIndex == null)//check if player2 skips
        return false;
    player2.DeleteCardsAfterMove();
    return true;
}
```

```
private void SkipMove(int playerNumber)
{
   if (playerNumber == 2)
      player1.Score += _deadBotNumber;
   else
      player2.Score += _deadBotNumber;
   _deadBotNumber = 0;
}
private void Response(Card cardToBeat)
{
   if (_gameMode == Mode.Easy)
   {
      player2.BotWeaponIndex = player2.EasyBotWeaponForBeating(cardToBeat);
      if (player2.BotWeaponIndex != null)//if has a weapon to beat the enemy
      {
         player2.BotArmourIndex = player2.EasyBotArmour();
         if (player2.BotArmourIndex == null)
         {
            player2.UnchooseAllCards();
            SkipMove(2);
            NewRound();
         }
         else {
            _deadBotNumber++;
         }
      }
      else {
         SkipMove(2);
         NewRound();
      }
   }
   else//if Mode.Medium or Mode.Difficult
   {
      player2.BotWeaponIndex = player2.FindMinBlackCard(cardToBeat);
      if (player2.BotWeaponIndex != null)//if has a weapon to beat the enemy
      {
         player2.BotArmourIndex = player2.FindMinRedCard();
         if (player2.BotArmourIndex == null)
         {
            player2.UnchooseAllCards();
            SkipMove(2);
            NewRound();
         }
         else
         {
            _deadBotNumber++;
         }
      }
      else
      {
         SkipMove(2);
         NewRound();
      }
   }
   player1.DeleteCardsAfterMove();
}
public void CreateBot()
{
   switch (_gameMode)
   {
      case Mode.Easy:
         {
            player2.BotWeaponIndex = player2.EasyBotWeaponForCreation();
            if (player2.BotWeaponIndex != null)//if there is a black card
            {
               player2.BotArmourIndex = player2.EasyBotArmour();
```

```
                    if (player2.BotArmourIndex == null)
                        player2.UnchooseAllCards();
                }
                break;
            }
        case Mode.Medium:
            {
                player2.BotWeaponIndex = player2.FindMinBlackCard(null);
                if (player2.BotWeaponIndex != null)//if there is a black card
                {
                    player2.BotArmourIndex = player2.MediumBotArmour(this._deadBotNumber);
                    if (player2.BotArmourIndex == null)
                        player2.UnchooseAllCards();
                }
                break;
            }
        case Mode.Difficult:
            {
                player2.BotWeaponIndex = player2.FindMinBlackCard(null);
                if (player2.BotWeaponIndex != null)//if there is a black card
                {
                    player2.BotArmourIndex = player2.minMax(this._deadBotNumber, player1.Score,
BOT_NUMBER_TO_WIN);
                    if (player2.BotArmourIndex == null)
                        player2.UnchooseAllCards();
                }
                break;
            }
    }
    if (player2.BotWeaponIndex == null)
    {
        SkipMove(2);
        NewRound();
    }
}

public void DealCardsForPlayers()
{
    player1.DealCards(deck);
    player2.DealCards(deck);
}
}
}
```

//Модуль Player

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GameAlgo
{
  public class Player
  {
    List<Card?> _hand;
    int? _botWeaponIndex;
    int? _botArmourIndex;
    int _score = 0;
    public List<Card?> Hand
    {
      get { return _hand; }
      set { _hand = value; }
    }
    public int? BotWeaponIndex
    {
```

```
    get { return _botWeaponIndex; }
    set { _botWeaponIndex = value; }
}
public int? BotArmourIndex
{
    get { return _botArmourIndex; }
    set { _botArmourIndex = value;}
}
public int Score
{
    get { return _score; }
    set { _score = value; }
}
public Player(List<Card?> hand)
{
    this._hand = hand;
}
public Card BotWeapon => (Card)_hand[(int)_botWeaponIndex];
public Card BotArmour => (Card)_hand[(int)BotArmourIndex];
public void DeleteCardsAfterMove()//
{
    if (_botArmourIndex == null || _botWeaponIndex == null)
        return;
    _hand[(int)_botArmourIndex] = null;
    _hand[(int)_botWeaponIndex] = null;
}
public void UnchooseAllCards()
{
    _botArmourIndex = null;
    _botWeaponIndex = null;
}
public bool hasArmour()
{
    for(int i=0; i< _hand.Count; i++)
    {
        if (_hand[i] != null && _hand[i].IsRed())
            return true;
    }
    return false;
}
public bool hasWeapon()
{
    for (int i = 0; i < _hand.Count; i++)
    {
        if (_hand[i] != null && _hand[i].IsBlack())
            return true;
    }
    return false;
}
public void DealCards(List<Card> deck)
{
    Random rnd = new Random();
    int index;
    for (int i = 0; i < _hand.Count; i++)
    {
        if (_hand[i] == null)
        {
            if (deck.Count == 0)
                return;//
            index = rnd.Next(deck.Count);
            _hand[i] = deck[index];
            deck.RemoveAt(index);
        }
    }
}
```

```
//FIND MIN OR MAX CARDS
public int? FindMinRedCard()
{
   int? minCardIndex = null;
   for (int i = 0; i < _hand.Count; i++)
   {
      if (_hand[i] == null)
         continue;
      if (_hand[i].IsRed())
      {
         if (minCardIndex == null || _hand[(int)minCardIndex].CardRank > _hand[i].CardRank)
            minCardIndex = i;
      }
   }
   return minCardIndex;
}
public int? FindMinBlackCard(Card? cardToBeat )
{
   int? minCardIndex = null;
   for (int i = 0; i < _hand.Count; i++)
   {
      if (_hand[i] == null)
         continue;
      if (_hand[i].IsBlack())
      {
         if (minCardIndex == null || _hand[(int)minCardIndex].CardRank > _hand[i].CardRank)
         {
            if (cardToBeat==null || _hand[i].CardRank >= cardToBeat.CardRank)
               minCardIndex = i;
         }
      }
   }
   return minCardIndex;
}
public int? FindMaxRedCard()
{
   int? maxCardIndex = null;
   for (int i = 0; i < _hand.Count; i++)
   {
      if (_hand[i] == null)
         continue;
      if (_hand[i].IsRed())
      {
         if (maxCardIndex == null || _hand[(int)maxCardIndex].CardRank < _hand[i].CardRank)
            maxCardIndex = i;
      }
   }
   return maxCardIndex;
}
public int? FindMaxBlackCard()
{
   int? maxCardIndex = null;
   for (int i = 0; i < _hand.Count; i++)
   {
      if (_hand[i] == null)
         continue;
      if (_hand[i].IsBlack())
      {
         if (maxCardIndex == null || _hand[(int)maxCardIndex].CardRank < _hand[i].CardRank)
            maxCardIndex = i;
      }
   }
   return maxCardIndex;
}
private int? FindBeforeMaxRedCard(int numberBeforeMax)
{
```

```
    List<(Card c, int ind)> redCardIndexes= new List<(Card, int)>();
    int? maxCardIndex = null;
    for (int i = 0; i < _hand.Count; i++)
    {
        if (_hand[i] == null)
            continue;
        if (_hand[i].IsRed())
        {
            redCardIndexes.Add((_hand[i], i));
        }
    }
    if (redCardIndexes.Count == 0)//if there is no red cards
        return null;
    redCardIndexes.OrderBy(x => x.c.CardSuit).ToList();
    int resultIndex = redCardIndexes.Count - numberBeforeMax - 1;
    if (resultIndex < 0 || resultIndex >= redCardIndexes.Count)
        resultIndex = redCardIndexes.Count-1;
    return redCardIndexes[resultIndex].ind;
}
public int FindMaxCard()
{
    int maxCardIndex = 0;
    for (int i=0;i< _hand.Count;i++)
    {
        if (_hand[i] == null) continue;
        if (_hand[i].CardRank > _hand[maxCardIndex].CardRank)
            maxCardIndex = i;
    }
    return maxCardIndex;
}

//EASY LEVEL
public int? EasyBotWeaponForCreation()
{
    return FindMaxBlackCard();
}
public int? EasyBotWeaponForBeating(Card cardToBeat)
{
    int? index = FindMaxBlackCard();
    if (index == null || _hand[(int)index].CardRank < cardToBeat.CardRank)
        return null;
    return index;
}
public int? EasyBotArmour()
{
    List<int> redCardsIndexes = new List<int>();
    for (int i = 0; i < _hand.Count;i++)
    {
        if (_hand[i] == null)
            continue;
        if (_hand[i].IsRed())
            redCardsIndexes.Add(i);
    }
    if (redCardsIndexes.Count == 0)
        return null;
    Random rnd = new Random();
    return redCardsIndexes[rnd.Next(redCardsIndexes.Count)];
}

//MEDIUM
public int? MediumBotArmour(int deadBotNumber)
{
    if(deadBotNumber > 1)
        return FindMaxRedCard();
    return FindBeforeMaxRedCard(1);
}
```

```
//HARD
public int? minMax(int deadBotNumber, int enemyScore, int BOT_NUMBER_TO_WIN)
{
    if(enemyScore > _score || enemyScore + deadBotNumber >= BOT_NUMBER_TO_WIN || deadBotNumber > 2)
    {
        if (deadBotNumber > 0)//if there are dead bots
            return FindMaxRedCard();
        return FindMinRedCard();
    }
    else
    {
        int bigNum, smallNum;
        CountBigAndSmallCards(out bigNum, out smallNum);
        if (bigNum > 2)
        {
            if(deadBotNumber > 0)
            {
                int? redMinIndex = FindMinRedCard();
                int? blackMaxIndex = FindMaxBlackCard();
                if ((redMinIndex != null && (int)_hand[(int)redMinIndex].CardRank < 10) || CountWeapon() <= 3
                    || (blackMaxIndex != null && (int)_hand[(int)blackMaxIndex].CardRank >= 10))
                    return FindMaxRedCard();
                else
                    return FindBeforeMaxRedCard(1);
            }
            else
            {
                Random rnd = new Random();
                if(rnd.Next(2) == 0)
                    return FindBeforeMaxRedCard(1);
                else
                    return FindBeforeMaxRedCard(2);
            }
        }
        else
        {
            if ((smallNum > 3 && deadBotNumber > 0 && bigNum > 0) || CountWeapon() > smallNum + bigNum)
                return FindMaxRedCard();
            return FindMinRedCard();
        }
    }
}
private void CountBigAndSmallCards(out int bigNum, out int smallNum)
{
    const int BIG_CARD_RANK = 10;
    bigNum = 0;
    smallNum = 0;
    for (int i=0; i< _hand.Count; i++)
    {
        if (_hand[i] != null && _hand[i].IsRed())
        {
            if ((int)_hand[i].CardRank >= BIG_CARD_RANK)
                bigNum++;
            else
                smallNum++;
        }
    }
}
private int CountWeapon()
{
    int n = 0;
    for(int i = 0; i< _hand.Count; i++)
    {
        if (_hand[i].IsBlack())
            n++;
```

```
            }
            return n;
        }
        //OTHER METHODS
        public bool CanBeatBot(Card card)
        {
            int? blackCardIndex = FindMinBlackCard(card);
            if (blackCardIndex == null)
                return false;
            else return true;
        }
    }
}
```

//Модуль Card

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GameAlgo
{
    public class Card
    {
        public enum Rank
        {
            r2, r3, r4, r5, r6, r7, r8, r9, r10,
            J, Q, K, Ace,
            Joker
        }
        public enum Suit
        {
            clubs,
            diamonds,
            hearts,
            spades
        }
        Rank _cardRank;
        Suit _cardSuit;
        public Rank CardRank => _cardRank;
        public Suit CardSuit => _cardSuit;
        public Card(Suit s, Rank r)
        {
            _cardSuit = s;
            _cardRank = r;
        }
        public static List<Card> GenerateDeck()
        {
            const int RANK_NUM = 12;
            const int SUIT_NUM = 4;
            List<Card> deck = new List<Card>();
            for (int suit=0; suit<SUIT_NUM; suit++)
            {
                for (int rank = 0; rank < RANK_NUM; rank++)
                    deck.Add(new Card((Suit)suit, (Rank)rank));
            }
            deck.Add(new Card(Suit.spades, Rank.Joker));
            deck.Add(new Card(Suit.clubs, Rank.Joker));
            return deck;
        }
        public bool IsRed()
        {
            if (_cardSuit == Suit.diamonds || _cardSuit == Suit.hearts)
                return true;
            return false;
```

```
        }
        public bool IsBlack()
        {
            if (_cardSuit == Suit.clubs || _cardSuit == Suit.spades)
                return true;
            return false;
        }
    }

}
```

3.1.2 Приклади роботи

На наступних рисунках показані приклади роботи програми.

Рисунок 3.1 – Початкова форма

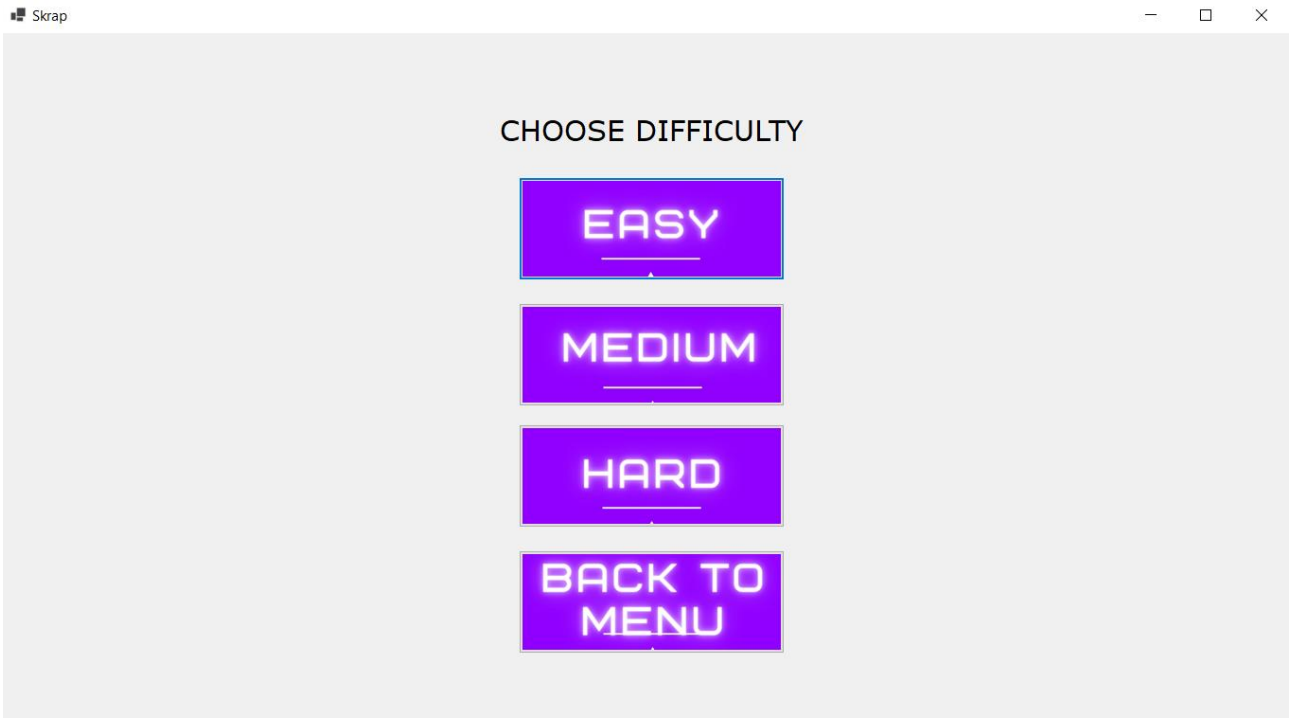

Рисунок 3.2 – Меню вибору складності

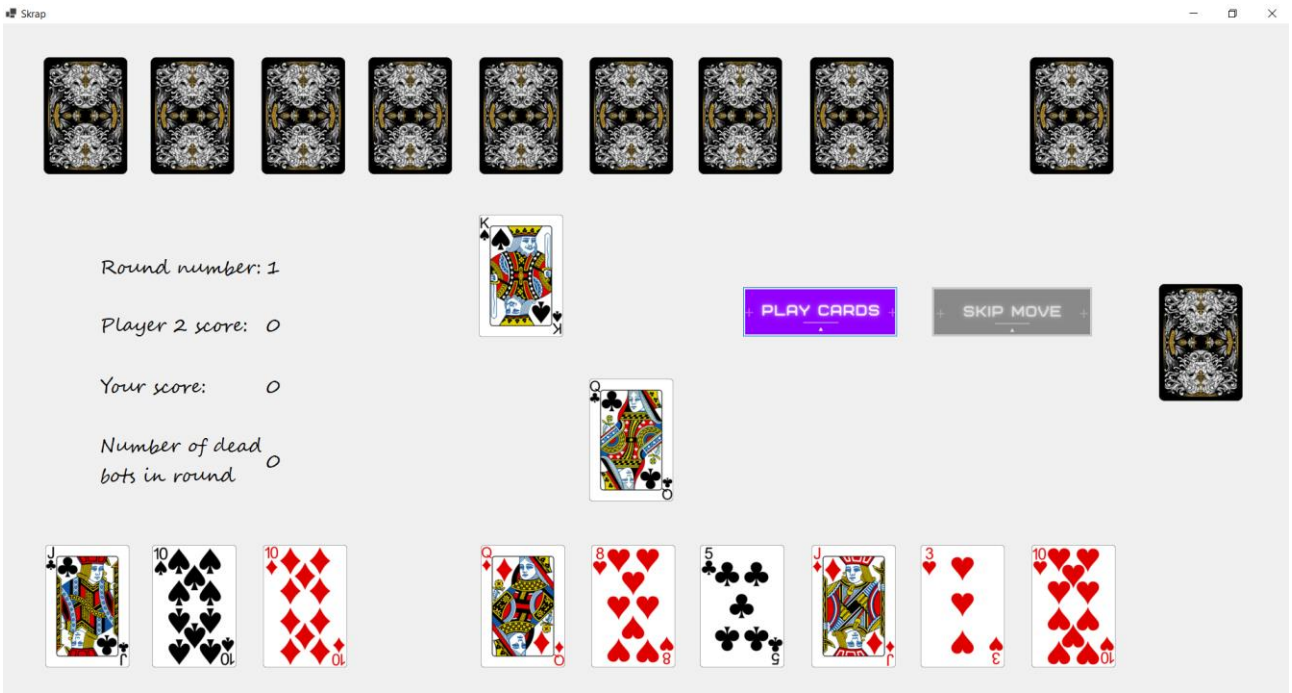

Рисунок 3.3 – Вирішення, який гравець робить хід першим (за величиною вибраних карт, або ж рандомом, якщо вибрані карти однакові за рангом)

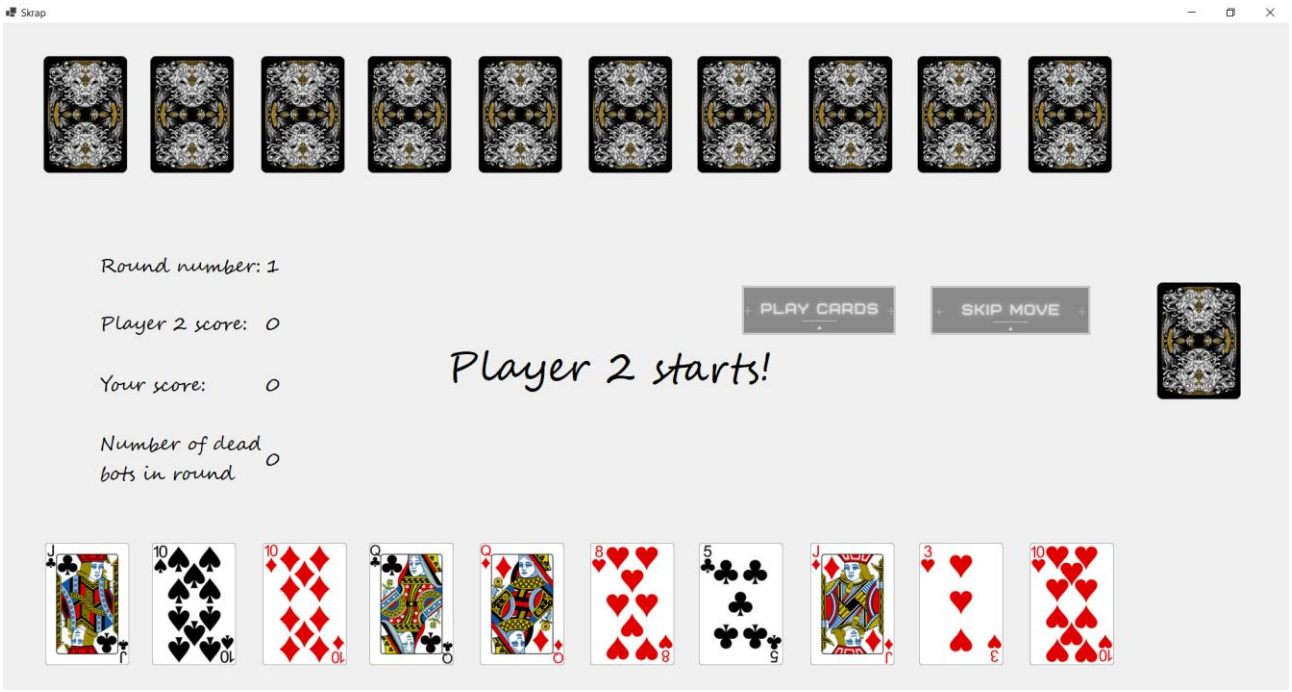

Рисунок 3.4 – Результати вирішення, хто з гравців починатиме гру (у даному випадку гравець 2, тобто, комп’ютерний опонент)

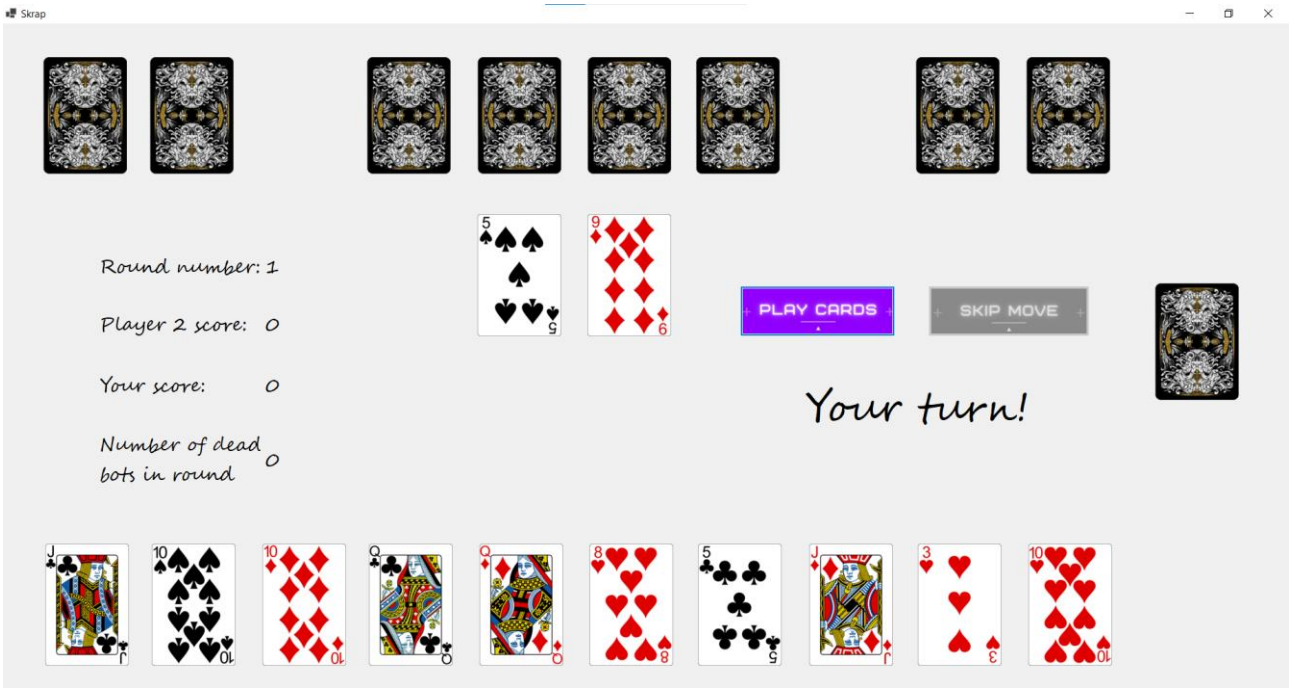

Рисунок 3.5 – Гравець 2 починає гру (виставляє бота зі зброєю 5 та бронею 9)

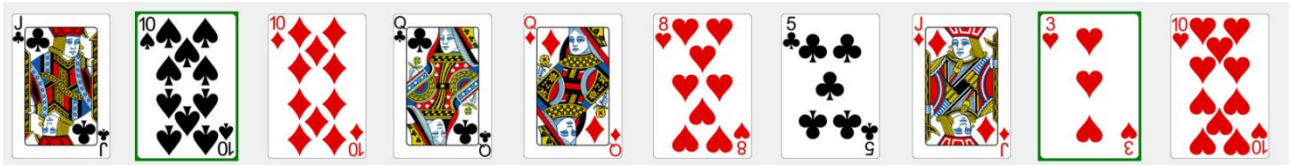

Рисунок 3.6 – Вибір карт користувачем для створення бота (навколо вибраних карт з’являється зелена рамка)

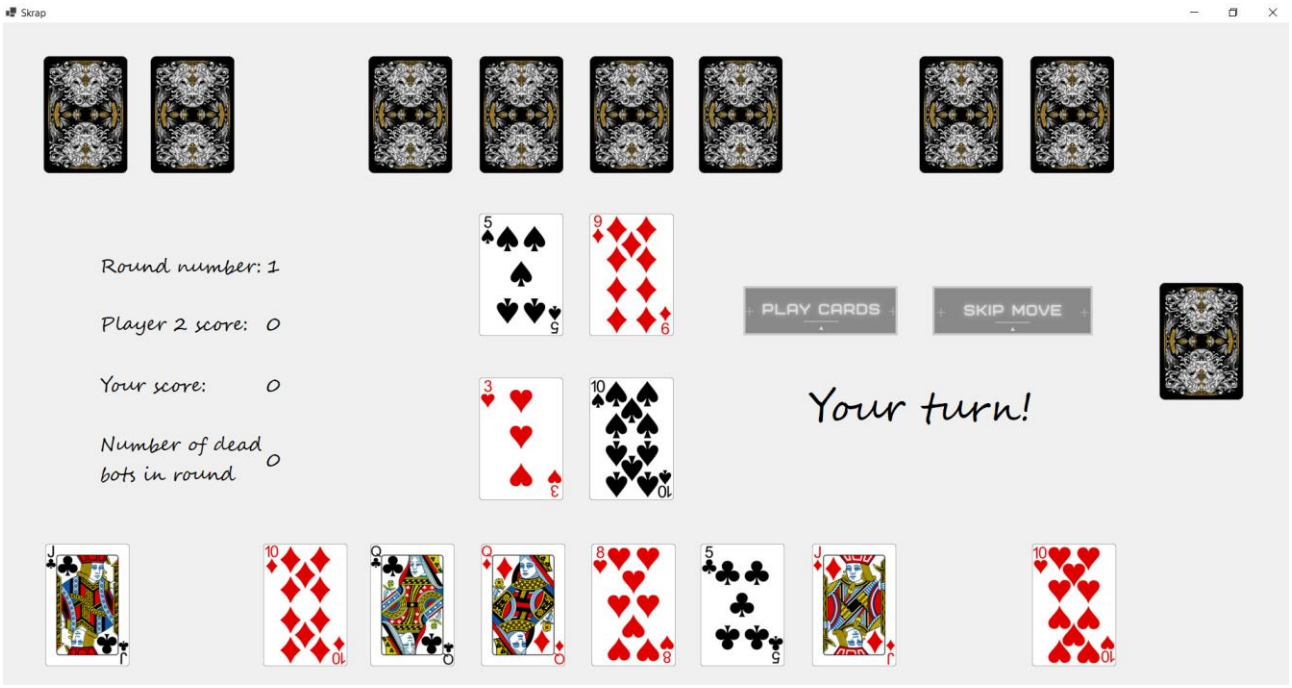

Рисунок 3.7 – Користувач б’є бота, виставленого комп’ютерним опонентом

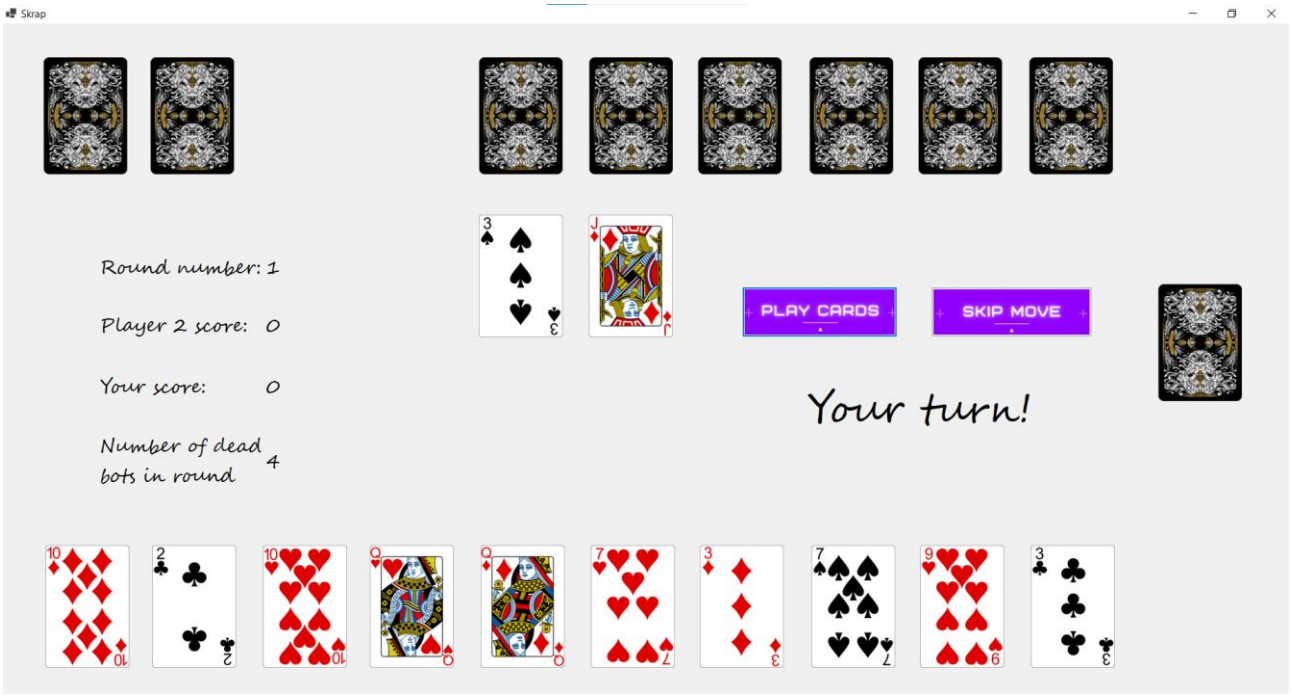

Рисунок 3.8 – Кнопка ‘Skip Move’ стає активною, коли гравець не може відбитися

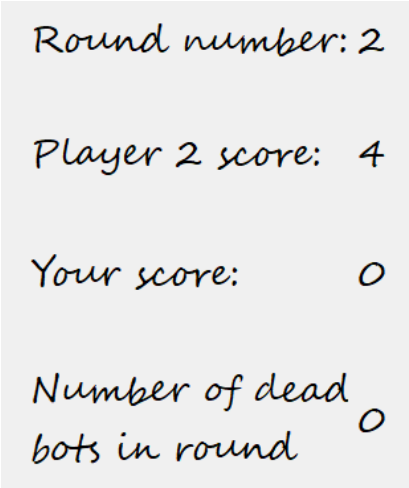

Рисунок 3.9 – Після того, як один з гравців не зміг побити бота розпочинається новий раунд з новою роздачею карт, бали гравців оновлюються, а к-сть вбитих ботів дорівнює нулю

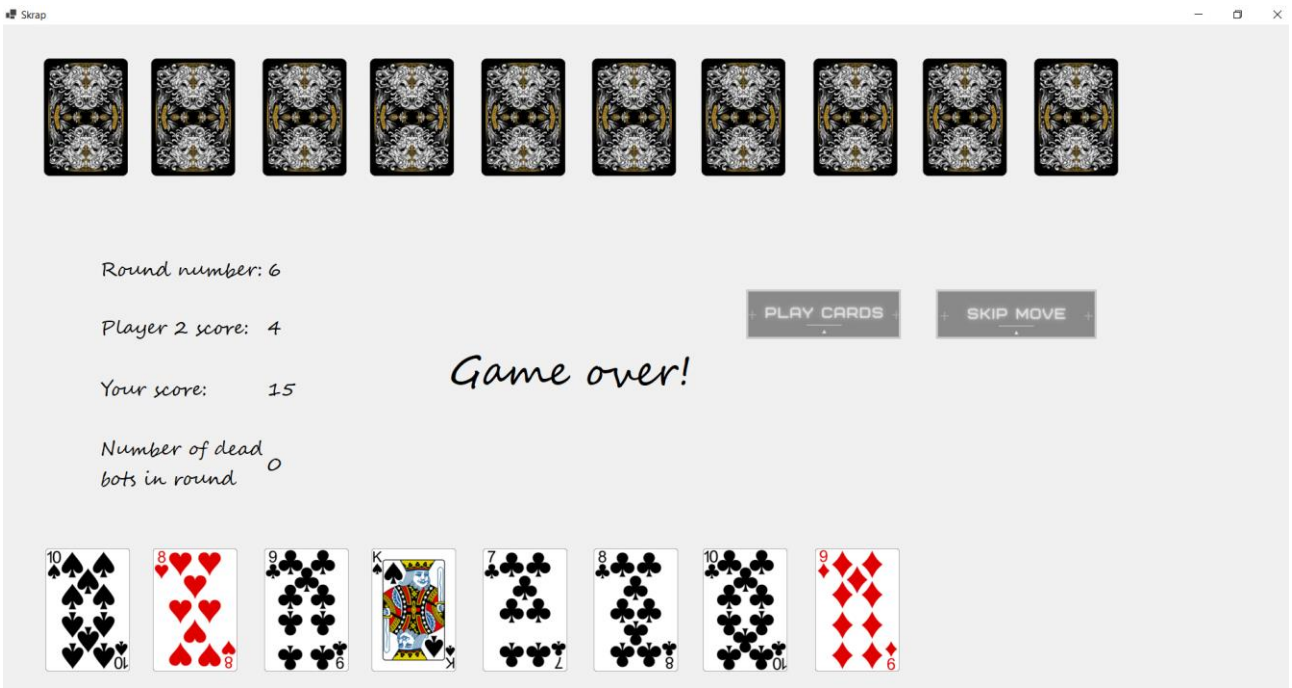

Рисунок 3.10 – Коли в одного з гравців к-сть балів більша або рівна 10, гра припиняється

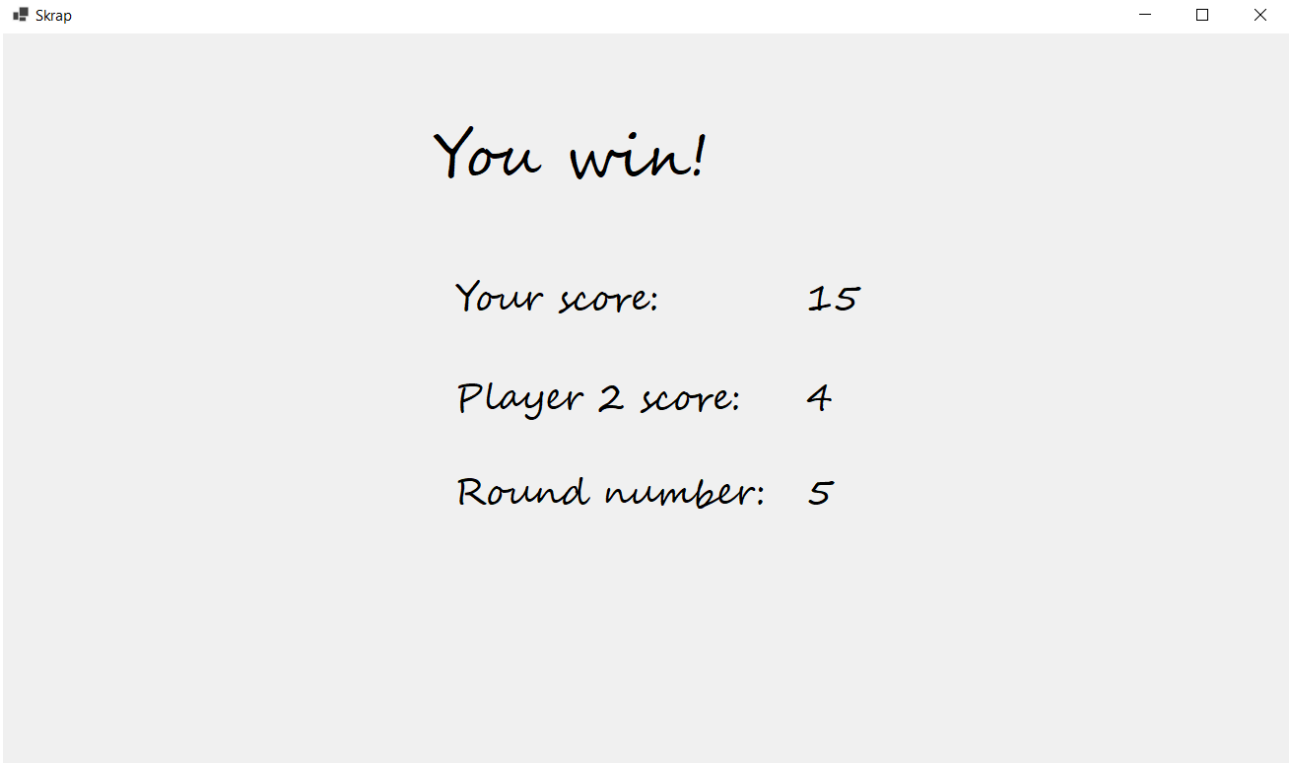

Рисунок 3.11 – Виведення кінцевих результатів та переможця

ВИСНОВОК

В рамках даної лабораторної роботи я вивчила основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Результатом виконання стала створена карткова гра “Війна з ботами” (“Skrap”). Дана гра дозволяє користувачу позмагатися з комп’ютерним опонентом на 3-х рівнях складності: легкому, середньому та складному. На кожному з рівнів комп’ютерний опонент діє за різними алгоритмами. На легкому рівні він намагається прийти до поразки, коли ж на складному намагається отримати перемогу.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 25.12.2022 включно максимальний балдорівнює – 5. Після 25.12.2022максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація– 95%;
- висновок – 5%.

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