**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**”





Київ 2022

Зміст

[**1**](#_heading=h.30j0zll) **Мета лабораторної роботи 3**

[**2**](#_heading=h.1fob9te) **Завдання 4**

[**3**](#_heading=h.2et92p0) **Виконання 6**

[3.1](#_heading=h.tyjcwt) Програмна реалізація алгоритму 6

[*3.1.1*](#_heading=h.3dy6vkm) *Вихідний код 6*

[*3.1.2*](#_heading=h.1t3h5sf) *Приклади роботи 6*

[**Висновок 7**](#_heading=h.4d34og8)

[**Критерії оцінювання 8**](#_heading=h.2s8eyo1)

# 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.

# 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 | З повною інформацією |
| 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 | Dots and Boxes 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 8х8 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) http://www.iggamecenter.com/info/ru/foxh.html | З повною 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 |
| Інтерпретація правил, використана при створенні програми:  Кількість карт: 52  Кількість гравців: 2  На початку гри кожному гравцеві дають 4 карти з колоди.  У свій хід можна:  а) викласти карту на стіл  б) якщо на столі є 3 чи 4 карти одного значення, взяти їх собі, виклавши на стіл відповідно 3 чи 4 карти з руки, старші за значенням.  У кінці ходу гравець добирає карти з колоди до чотирьох.  Гра закінчується, коли в колоді та в руках гравців більше немає карт.  Бали нараховуються так:  Кожен набір по 3 карти - 1 бал  Кожен набір по 4 карти - 2 бали  Виграє той, хто має більше балів. | | |
| 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 Noughts and Crosses 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 |

# Виконання

## Системні вимоги

Повинен бути встановлений інтерпретатор Python 3 та модуль pygame.

## Програмна реалізація алгоритму

### Вихідний код

BeltCardGame.py

import Mouse

import ScrSwitcher

import pygame

def main():

"""Основний файл програми для гри 'Ремінь' """

pygame.init()

pygame.font.init()

dimensions = (1300, 750)

scr = pygame.display.set\_mode(dimensions)

background = pygame.image.load('assets/background.png')

pygame.display.set\_caption("Ремінь")

clock = pygame.time.Clock()

mouse = Mouse.Mouse()

scrSwitcher = ScrSwitcher.ScrSwitcher(scr, mouse)

FPS = 60

run = True

while run == True:

scr.blit(background, (0, 0))

mouse.update()

scrSwitcher.update()

pygame.display.update()

clock.tick(FPS)

for myEvent in pygame.event.get():

if myEvent.type == pygame.QUIT:

pygame.quit()

run = False

if \_\_name\_\_ == '\_\_main\_\_':

main()

Mouse.py

import pygame

class Mouse():

"""Записує позицію курсору та натискання ЛКМ"""

def \_\_init\_\_(self):

self.x : int

self.y : int

self.left = False

self.right = False

def update(self):

self.x = pygame.mouse.get\_pos()[0]

self.y = pygame.mouse.get\_pos()[1]

if pygame.mouse.get\_pressed()[0]:

if self.isDown == False:

self.isDown = True

elif self.isDown:

self.isDown = None

else:

self.isDown = False

def intersects(self, coords):

if (self.x >= coords[0]) and (self.x <= coords[2]) and (self.y >= coords[1]) and (self.y <= coords[3]):

return True

else:

return False

ScrSwitcher.py

import Menu

import InGame

class ScrSwitcher(object):

"""Клас, що перемикає між головним меню та ігровим полем"""

def \_\_init\_\_(self, scr, mouse):

self.state = 0 # 0 - menu, 1 - game

self.menu = None

self.inGame = None

self.scr = scr

self.mouse = mouse

self.p2difficulty = None

self.p3difficulty = None

self.p4difficulty = None

def update(self):

"""Оновлення ігрового поля та меню або перехід між ними"""

if self.state == 0:

if self.menu == None:

self.menu = Menu.Menu(self.scr, self.mouse)

menuUpdate = self.menu.update()

if menuUpdate == 0:

pass

elif isinstance(menuUpdate, tuple):

self.state = 1

self.menu = None

self.p2difficulty = menuUpdate[0]

self.p3difficulty = menuUpdate[1]

self.p4difficulty = menuUpdate[2]

if self.state == 1:

if self.inGame == None:

self.inGame = InGame.InGame(self.scr, self.mouse, self.p2difficulty, self.p3difficulty, self.p4difficulty)

inGameUpdate = self.inGame.update()

if inGameUpdate == 0:

pass

elif inGameUpdate == 1:

self.state = 0

self.inGame = None

Menu.py

import TextBox

import Button

import Deck

import Controls

class Menu(object):

"""Головне меню гри"""

def \_\_init\_\_(self, scr, mouse):

self.scr = scr

self.mouse = mouse

self.p2\_difficulty = 1

self.p3\_difficulty = None

self.p4\_difficulty = None

self.title = TextBox.TextBox(self.scr, 'Ремінь', (520, 190, 540, 210), (255, 255, 255), -1, 48)

self.playButton = Button.Button(self.mouse, self.scr,'Почати гру',(190, 265, 490, 335), (255, 255, 255), 2, 28)

self.controlsWindow = None

self.controlsButton = Button.Button(self.mouse, self.scr, 'Правила', (560, 265, 860, 335), (255, 255, 255), 2, 28)

self.p1\_text = TextBox.TextBox(self.scr, 'Гравець 1 – Ви', (125, 355, 375, 405), (255, 255, 255), -1, 20)

self.p2\_text = TextBox.TextBox(self.scr, 'Гравець 2', (150, 415, 350, 465), (255, 255, 255), -1, 20)

self.p2\_easyButton = Button.Button(self.mouse, self.scr, 'Легкий', (150, 490, 350, 540), (175, 175, 175), 2, 20)

self.p2\_normalButton = Button.Button(self.mouse, self.scr, 'Середній', (150, 565, 350, 615), (175, 175, 175), 2, 20)

self.p2\_hardButton = Button.Button(self.mouse, self.scr, 'Важкий', (150, 640, 350, 690), (175, 175, 175), 2, 20)

#self.p3\_text = TextBox.TextBox(self.scr, 'Гравець 3', (425, 355, 625, 405), (255, 255, 255), 2, 20)

#self.p3\_offButton = Button.Button(self.mouse, self.scr, 'Вимкнено', (425, 415, 625, 465), (175, 175, 175), 2, 20)

#self.p3\_easyButton = Button.Button(self.mouse, self.scr, 'Легкий', (425, 490, 625, 540), (175, 175, 175), 2, 20)

#self.p3\_normalButton = Button.Button(self.mouse, self.scr, 'Середній', (425, 565, 625, 615), (175, 175, 175), 2, 20)

#self.p3\_hardButton = Button.Button(self.mouse, self.scr, 'Важкий', (425, 640, 625, 690), (175, 175, 175), 2, 20)

#self.p4\_text = TextBox.TextBox(self.scr, 'Гравець 4', (700, 355, 900, 405), (255, 255, 255), 2, 20)

#self.p4\_offButton = Button.Button(self.mouse, self.scr, 'Вимкнено', (700, 415, 900, 465), (175, 175, 175), 2, 20)

#self.p4\_easyButton = Button.Button(self.mouse, self.scr, 'Легкий', (700, 490, 900, 540), (175, 175, 175), 2, 20)

#self.p4\_normalButton = Button.Button(self.mouse, self.scr, 'Середній', (700, 565, 900, 615), (175, 175, 175), 2, 20)

#self.p4\_hardButton = Button.Button(self.mouse, self.scr, 'Важкий', (700, 640, 900, 690), (175, 175, 175), 2, 20)

def update(self):

if self.controlsWindow is not None:

controlsWindowUpdate = self.controlsWindow.update()

if controlsWindowUpdate == 0:

pass

elif controlsWindowUpdate == 1:

self.controlsWindow = None

return 0

elif self.controlsWindow is None:

self.title.update()

self.playButton.update()

self.controlsButton.update()

self.p1\_text.update()

self.p2\_text.update()

self.p2\_easyButton.update(self.p2\_difficulty == 0)

self.p2\_normalButton.update(self.p2\_difficulty == 1)

self.p2\_hardButton.update(self.p2\_difficulty == 2)

#self.p3\_text.update()

#self.p3\_offButton.update(self.p3\_difficulty is None)

#self.p3\_easyButton.update(self.p3\_difficulty == 0)

#self.p3\_normalButton.update(self.p3\_difficulty == 1)

#self.p3\_hardButton.update(self.p3\_difficulty == 2)

#self.p4\_text.update()

#self.p4\_offButton.update(self.p4\_difficulty is None)

#self.p4\_easyButton.update(self.p4\_difficulty == 0)

#self.p4\_normalButton.update(self.p4\_difficulty == 1)

#self.p4\_hardButton.update(self.p4\_difficulty == 2)

if self.p2\_easyButton.isPressed:

self.p2\_difficulty = 0

if self.p2\_normalButton.isPressed:

self.p2\_difficulty = 1

if self.p2\_hardButton.isPressed:

self.p2\_difficulty = 2

#if self.p3\_offButton.isPressed:

# self.p3\_difficulty = None

#if self.p3\_easyButton.isPressed:

# self.p3\_difficulty = 0

#if self.p3\_normalButton.isPressed:

# self.p3\_difficulty = 1

#if self.p3\_hardButton.isPressed:

# self.p3\_difficulty = 2

#if self.p4\_offButton.isPressed:

# self.p4\_difficulty = None

#if self.p4\_easyButton.isPressed:

# self.p4\_difficulty = 0

#if self.p4\_normalButton.isPressed:

# self.p4\_difficulty = 1

#if self.p4\_hardButton.isPressed:

# self.p4\_difficulty = 2

if self.controlsButton.isPressed:

self.controlsWindow = Controls.Controls(self.scr, self.mouse)

if self.playButton.isPressed:

return (self.p2\_difficulty, self.p3\_difficulty, self.p4\_difficulty)

else:

return 0

Controls.py

import Button

import TextBox

class Controls(object):

"""Вікно з правилами та поясненням як грати"""

def \_\_init\_\_(self, scr, mouse):

self.scr = scr

self.mouse = mouse

self.title = TextBox.TextBox(self.scr, 'Правила гри', (340, 40, 710, 140), (255, 255, 255), -1, 48)

self.text = []

self.text.append(TextBox.TextBox(self.scr, 'На початку гри кожному гравцеві дають 4 карти з колоди.', (350, 150, 550, 190), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'У свій хід можна:', (150, 190, 350, 230), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'а) викласти карту на стіл (ЛКМ по карті в руці)', (315, 230, 500, 270), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'б) якщо на столі є 3 чи 4 карти одного значення, взяти їх собі,', (390, 270, 560, 310), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'виклавши на стіл відповідно 3 чи 4 карти з руки,', (377, 310, 500, 350), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'старші за значенням (ЛКМ по карті бажаного значення на столі)', (460, 350, 575, 390), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'У кінці ходу гравець добирає карти з колоди до чотирьох.', (350, 390, 550, 430), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'Гра закінчується, коли в колоді та в руках гравців більше немає карт.', (413, 430, 600, 470), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'Бали нараховуються так:', (155, 470, 425, 510), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'Кожен набір по 3 карти - 1 бал', (218, 510, 420, 550), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'Кожен набір по 4 карти - 2 бали', (225, 550, 425, 590), (255, 255, 255), -1, 20))

self.text.append(TextBox.TextBox(self.scr, 'Виграє той, хто має більше балів.', (233, 590, 430, 630), (255, 255, 255), -1, 20))

self.closeButton = Button.Button(self.mouse, self.scr, 'Назад', (840, 660, 1015, 730), (255, 255, 255), 2, 28)

def update(self):

self.title.update()

for line in self.text:

line.update()

self.closeButton.update()

if self.closeButton.isPressed:

return 1

else:

return 0

Button.py

import TextBox

class Button(TextBox.TextBox):

"""Кнопка в меню, яку можна натиснути"""

def \_\_init\_\_(self, mouse, scr, text, coords, color, borderWidth, textSize):

super().\_\_init\_\_(scr, text, coords, color, borderWidth, textSize)

self.isPressed = False

self.isHovered = False

self.mouse = mouse

def update(self, isSelected = False):

if self.mouse.intersects(self.coords):

if self.mouse.isDown:

self.isPressed = True

self.isHovered = False

else:

self.isPressed = False

self.isHovered = True

else:

self.isPressed = False

self.isHovered = False

if self.isPressed:

super().update((140, 255, 251))

elif self.isHovered:

super().update((161, 209, 210))

elif isSelected:

super().update((255, 248, 125))

else:

super().update()

TextBox.py

import pygame

class TextBox(object):

"""Поле з текстом"""

def \_\_init\_\_(self, scr, text, coords, color, borderWidth, textSize):

self.scr = scr

self.text = text

self.coords = coords

self.defaultColor = color

self.color = color

self.borderWidth = borderWidth

self.textSize = textSize

def update(self, color = None, newText = None):

if newText != None:

self.text = newText

if color != None:

self.color = color

else:

self.color = self.defaultColor

pygame.draw.rect(self.scr, self.color, (self.coords[0], self.coords[1], self.coords[2]-self.coords[0], self.coords[3]-self.coords[1]), self.borderWidth)

textObj, textRect = self.makeText(self.text, ((self.coords[0]+self.coords[2])//2, (self.coords[1]+self.coords[3])//2), self.textSize, self.color)

self.scr.blit(textObj, textRect)

@staticmethod

def makeText(text, coords, size, color):

fontObj = pygame.font.SysFont('bookantiquaполужирный', size)

text = str(text)

textObj = fontObj.render(text, True, color)

textRect = textObj.get\_rect()

textRect.center = coords

return textObj, textRect

InGame.py

import random

import Human

import Computer

import Deck

import Action

import random

import Table

import EndScreen

import pygame

class InGame(object):

"""Ігрове поле: колода, стіл, гравці"""

def \_\_init\_\_(self, scr, mouse, p2difficulty, p3difficulty, p4difficulty):

self.scr = scr

self.mouse = mouse

self.deck = Deck.Deck(self.scr)

self.table = Table.Table(self.scr, self.mouse)

computerNames = [['Тобі', 'Міг', 'Ларрі'], ['Альфа', 'Макс', 'Корвус'], ['Бенбен', 'Аксель', 'Текет']]

computerOriginCoords = [[(375, 8)], [(200, 8), (575, 8)], [(25, 8), (375, 8), (725, 8)]]

computersCount = 0

for item in (p2difficulty, p3difficulty, p4difficulty):

if item is not None:

computersCount += 1

computerOriginCoords = computerOriginCoords[computersCount - 1]

p2NameInd = random.randint(0, 2)

self.players = [Human.Human(self.scr, self.mouse), Computer.Computer(self.scr, computerOriginCoords[0], p2difficulty, computerNames[p2difficulty][p2NameInd])]

if p3difficulty is not None:

p3NameInd = random.randint(0, 2)

if p3difficulty == p2difficulty:

while p3NameInd == p2NameInd:

p3NameInd = random.randint(0, 2)

self.players.append(Computer.Computer(self.scr, computerOriginCoords[1], p3difficulty, computerNames[p3difficulty][p3NameInd]))

if p4difficulty is not None:

p4NameInd = random.randint(0, 2)

if p4difficulty == p2difficulty and p4difficulty == p3difficulty:

while p4NameInd == p2NameInd or p4NameInd == p3NameInd:

p4NameInd = random.randint(0, 2)

elif p4difficulty == p2difficulty:

while p4NameInd == p2NameInd:

p4NameInd = random.randint(0, 2)

elif p4difficulty == p3difficulty:

while p4NameInd == p3NameInd:

p4NameInd = random.randint(0, 2)

self.players.append(Computer.Computer(self.scr, computerOriginCoords[-1], p4difficulty, computerNames[p4difficulty][p4NameInd]))

self.turn = random.randrange(0, len(self.players))

# Action.ActionType:

# 0 - Draw a card from the deck

# 1 - Put a card on the table

# 2 - Grab cards from the table

# 3 - Put cards on the table

# Action.player:

# the index in InGame.players

self.actionsQueue = []

for i in range(len(self.players)): #each player draws 4 cards

for j in range(4):

self.actionsQueue.append(Action.Action(0, None, i))

self.endScreen = None

def allEmpty(self):

if len(self.actionsQueue) > 0:

return False

if len(self.deck.cards) > 0:

return False

for player in self.players:

if len(player.hand) > 2:

return False

return True

def update(self):

"""Оновлення ігрового поля, інтерпретування об'єктів Action"""

self.deck.update()

for player in self.players:

player.update()

self.table.update()

if self.endScreen is not None:

endScreenUpdate = self.endScreen.update()

if endScreenUpdate == 0:

return 0

elif endScreenUpdate == 1:

self.endScreen = None

return 1

else:

if self.allEmpty():

computerNames = []

scores = []

for player in self.players:

if isinstance(player, Computer.Computer):

computerNames.append(player.name)

scores.append(player.getScore())

self.endScreen = EndScreen.EndScreen(self.scr, self.mouse, computerNames, scores)

else:

if len(self.actionsQueue) > 0:

if self.actionsQueue[0].step == 0:

if self.actionsQueue[0].actionType == 0:

cardsInAction = [self.deck.draw()]

elif self.actionsQueue[0].actionType == 1:

source = self.actionsQueue[0].source

playerInd = source[0]

searchedValue = source[1]

searchedSuit = source[2]

searchedCardInd = 0

while searchedValue != self.players[playerInd].hand[searchedCardInd].value or searchedSuit != self.players[playerInd].hand[searchedCardInd].suit:

searchedCardInd += 1

cardsInAction = [self.players[source[0]].giveCard(searchedCardInd)]

elif self.actionsQueue[0].actionType == 2:

cardsInAction = self.table.giveStack(self.actionsQueue[0].source, stackOf3 = (self.actionsQueue[0].cardsAmount == 3))

elif self.actionsQueue[0].actionType == 3:

playerInd = self.actionsQueue[0].source[0]

searchedCards = self.actionsQueue[0].source[1]

cardsInAction = []

for card in searchedCards:

for i in range(len(self.players[playerInd].hand) - 1, -1, -1):

if card == (self.players[playerInd].hand[i].value, self.players[playerInd].hand[i].suit):

cardsInAction.append(self.players[playerInd].giveCard(i))

if cardsInAction[0] is None:

\_ = self.actionsQueue.pop(0)

#decide on the next turn

if len(self.actionsQueue) == 0:

self.turn += 1

if self.turn == len(self.players):

self.turn = 0

while len(self.players[self.turn].hand) == 0:

self.turn += 1

if self.turn == len(self.players):

self.turn = 0

else:

self.actionsQueue[0].setPaths(cardsInAction, self.players, self.table)

if len(self.actionsQueue) > 0:

if self.actionsQueue[0].movedCards is not None:

actionResult = self.actionsQueue[0].update()

if actionResult != 0:

if self.actionsQueue[0].actionType == 0:

self.players[self.actionsQueue[0].destination].addCard(actionResult[0])

elif self.actionsQueue[0].actionType == 1:

self.table.addCard(actionResult[0])

elif self.actionsQueue[0].actionType == 2:

for card in actionResult:

self.players[self.actionsQueue[0].destination].addCard(card)

if self.actionsQueue[0].cardsAmount == 3:

self.players[self.actionsQueue[0].destination].addTo3()

if self.actionsQueue[0].cardsAmount == 4:

self.players[self.actionsQueue[0].destination].addTo4()

elif self.actionsQueue[0].actionType == 3:

for i in range(len(actionResult)):

self.table.addCard(actionResult[i])

\_ = self.actionsQueue.pop(0)

#decide on the next turn

if len(self.actionsQueue) == 0:

self.turn = (self.turn + 1) % len(self.players)

while len(self.players[self.turn].hand) == 0:

self.turn = (self.turn + 1) % len(self.players)

else:

if self.turn == 0:

clickedCard = self.players[0].clickedCard()

if clickedCard is not None:

#add an action of type 1 and an action of type 0 if deck isn't empty

clickedCardValue, clickedCardSuit = clickedCard[0], clickedCard[1]

self.actionsQueue.append(Action.Action(1, (0, clickedCardValue, clickedCardSuit), None))

if not self.deck.isEmpty():

self.actionsQueue.append(Action.Action(0, None, 0))

else:

clickedStack = self.table.clickedStack()

if clickedStack is not None:

#add an action of type 2 and type 3 and as many type 0 as needed if possible

stackSize = clickedStack[1]

clickedStackValue = clickedStack[0]

if stackSize == 3:

stackInHandValue = self.players[0].has3Stack()

if stackInHandValue != None:

if self.table.cardKeys.index(stackInHandValue) > self.table.cardKeys.index(clickedStackValue):

#find the names of the 3 cards

cardsToGive = [(self.players[0].hand[i].value, self.players[0].hand[i].suit) for i in range(len(self.players[0].hand))]

if len(self.players[0].hand) == 4:

wrongCard = 0

while cardsToGive[wrongCard][0] == stackInHandValue and wrongCard < 3:

wrongCard += 1

cardsToGive.pop(wrongCard)

self.actionsQueue.append(Action.Action(3, (0, cardsToGive), None, cardsAmount = 3))

self.actionsQueue.append(Action.Action(2, clickedStackValue, 0, cardsAmount = 3))

elif stackSize == 4:

stackInHandValue = self.players[0].has4Stack()

if stackInHandValue != None:

if self.table.cardKeys.index(stackInHandValue) > self.table.cardKeys.index(clickedStackValue):

#find the names of the 4 cards

cardsToGive = [(self.players[0].hand[i].value, self.players[0].hand[i].suit) for i in range(4)]

self.actionsQueue.append(Action.Action(3, (0, cardsToGive), None, cardsAmount = 4))

self.actionsQueue.append(Action.Action(2, clickedStackValue, 0, cardsAmount = 4))

else:

stackInHandValue = self.players[0].has3Stack()

if stackInHandValue != None:

if self.table.cardKeys.index(stackInHandValue) > self.table.cardKeys.index(clickedStackValue):

#find the names of the 3 cards

cardsToGive = [(self.players[0].hand[i].value, self.players[0].hand[i].suit) for i in range(len(self.players[0].hand))]

if len(self.players[0].hand) == 4:

wrongCard = 0

while cardsToGive[wrongCard][0] == stackInHandValue and wrongCard < 3:

wrongCard += 1

cardsToGive.pop(wrongCard)

self.actionsQueue.append(Action.Action(3, (0, cardsToGive), None, cardsAmount = 3))

self.actionsQueue.append(Action.Action(2, clickedStackValue, 0, cardsAmount = 3))

else:

pygame.display.update()

actions = self.players[self.turn].decide(self.players, self.deck, self.table, self.turn)

if len(actions) > 0:

for action in actions:

self.actionsQueue.append(action)

else:

self.turn = (self.turn + 1) % len(self.players)

return 0

Player.py

import Card

import pygame

class Player(object):

"""Клас-шаблон для створення графців"""

def \_\_init\_\_(self, scr):

self.scr = scr

self.hand = []

self.stacksOf3 = 0

self.stacksOf3Image = pygame.image.load('assets/3\_stack.png')

self.stacksOf4 = 0

self.stacksOf4Image = pygame.image.load('assets/4\_stack.png')

def giveCard(self, ind):

return self.hand.pop(ind)

def handIsFull(self):

if len(self.hand) == 4:

return True

else:

return False

def has3Stack(self):

if len(self.hand) == 3:

value0 = self.hand[0].value

for card in self.hand:

if not card.value == value0:

return None

return value0

elif self.handIsFull():

value0 = self.hand[0].value

wrongCards = 0

for card in self.hand:

if not card.value == value0:

wrongCards += 1

if wrongCards <= 1:

return value0

elif self.hand[1].value == self.hand[2].value and self.hand[3].value == self.hand[2].value:

return self.hand[1].value

return None

def has4Stack(self):

if self.handIsFull():

value0 = self.hand[0].value

for card in self.hand:

if not card.value == value0:

return None

return value0

return None

def addCard(self, card):

self.hand.append(card)

def addTo3(self):

self.stacksOf3 += 1

def addTo4(self):

self.stacksOf4 += 1

def getScore(self):

return self.stacksOf3 + self.stacksOf4 \* 2

Human.py

import Player

import Card

import TextBox

class Human(Player.Player):

"""Гравець, яким керує людина"""

def \_\_init\_\_(self, scr, mouse):

super().\_\_init\_\_(scr)

self.mouse = mouse

self.stacksOf4Coords = (50, 600)

self.stacksOf4Number = TextBox.TextBox(self.scr, self.stacksOf4, (50, 600, 200, 700), (255, 255, 255), -1, 40)

self.stacksOf3Coords = (225, 600)

self.stacksOf3Number = TextBox.TextBox(self.scr, self.stacksOf3, (225, 600, 350, 700), (255, 255, 255), -1, 40)

self.cardCoords = [

None,

[(650, 600)],

[(600, 600), (700, 600)],

[(550, 600), (650, 600), (750, 600)],

[(500, 600), (600, 600), (700, 600), (800, 600)]

]

def clickedCard(self):

"""Знайти значення карти, по якій клікнув гравець-людина"""

if self.mouse.isDown:

for i in range(len(self.hand)):

if self.mouse.intersects((self.cardCoords[len(self.hand)][i][0] + 1, self.cardCoords[len(self.hand)][i][1] + 1, self.cardCoords[len(self.hand)][i][0] + 99, self.cardCoords[len(self.hand)][i][1] + 138)):

return (self.hand[i].value, self.hand[i].suit)

return None

def update(self):

self.renderCards()

self.scr.blit(self.stacksOf4Image, self.stacksOf4Coords)

self.scr.blit(self.stacksOf3Image, self.stacksOf3Coords)

self.stacksOf3Number.update(None, self.stacksOf3)

self.stacksOf4Number.update(None, self.stacksOf4)

def renderCards(self):

handSize = len(self.hand)

for i in range(len(self.hand)):

self.hand[i].update(self.cardCoords[handSize][i], 2)

Computer.py

import Player

import TextBox

import Card

import GameState

import Action

class Computer(Player.Player):

"""Клас гравця, за якого грає комп'ютер"""

def \_\_init\_\_(self, scr, originCoords, difficulty, name):

super().\_\_init\_\_(scr)

self.x0 = originCoords[0]

self.y0 = originCoords[1]

self.difficulty = difficulty

self.stacksOf4Coords = (self.x0, self.y0+125)

self.stacksOf4Number = TextBox.TextBox(self.scr, self.stacksOf4, (self.x0, self.y0+125, self.x0+150, self.y0+225), (255, 255, 255), -1, 40)

self.stacksOf3Coords = (self.x0+175, self.y0+125)

self.stacksOf3Number = TextBox.TextBox(self.scr, self.stacksOf3, (self.x0+175, self.y0+125, self.x0+300, self.y0+225), (255, 255, 255), -1, 40)

self.cardCoords = [

None,

[(self.x0 + 112, self.y0)],

[(self.x0 + 75, self.y0), (self.x0 + 150, self.y0)],

[(self.x0 + 37, self.y0), (self.x0 + 112, self.y0), (self.x0 + 187, self.y0)],

[(self.x0, self.y0), (self.x0 + 75, self.y0), (self.x0 + 150, self.y0), (self.x0 + 225, self.y0)]

]

self.name = name

self.nameBox = TextBox.TextBox(self.scr, self.name, (self.x0, self.y0 + 225, self.x0 + 300, self.y0 + 275), (255, 255, 255), -1, 28)

def update(self):

self.renderCards()

self.scr.blit(self.stacksOf4Image, self.stacksOf4Coords)

self.scr.blit(self.stacksOf3Image, self.stacksOf3Coords)

self.stacksOf3Number.update(None, self.stacksOf3)

self.stacksOf4Number.update(None, self.stacksOf4)

self.nameBox.update()

def renderCards(self):

handSize = len(self.hand)

for i in range(len(self.hand)):

self.hand[i].update(self.cardCoords[handSize][i], 1)

def decide(self, players, deck, table, thisPlayerInd):

"""Обрати наступну дію для комп'ютера, використовуючи мінімакс із альфа-бета відсіканнями"""

actions = []

simpleDeck = []

for card in deck.cards:

simpleDeck.append((card.value, card.suit))

simpleHands = []

for player in players:

simpleHands.append([])

for card in player.hand:

simpleHands[-1].append((card.value, card.suit))

simpleTable = {'2' : None,

'3' : None,

'4' : None,

'5' : None,

'6' : None,

'7' : None,

'8' : None,

'9' : None,

'10' : None,

'J' : None,

'Q' : None,

'K' : None,

'A' : None}

for value in simpleTable:

simpleTable[value] = [(card.value, card.suit) for card in table.cards[value]]

playerScores = [player.getScore() for player in players]

simpleState = GameState.GameState(

simpleDeck,

simpleHands,

simpleTable,

playerScores,

thisPlayerInd,

height = 6 + self.difficulty

)

if simpleState.bestMoveInd is None:

return actions

move = simpleState.possibleMoves[simpleState.bestMoveInd]

###################

#Вивести всі можливі ходи, які розглядав комп'ютер, та їх ціну

#print(f'comp: {simpleState.handNames[1]}')

#print(f'deck: {simpleState.deckNames[:min(len(simpleState.deckNames), 4)]}...\n')

#for i in range(len(simpleState.possibleMoves)):

# print(simpleState.possibleMoves[i], simpleState.childStates[i].stateScore)

# print(f'comp: {simpleState.childStates[i].handNames[1]}')

# print(f'deck: {simpleState.childStates[i].deckNames[:min(len(simpleState.deckNames), 4)]}...\n')

###################

if move[0] == 0:

handCardValue = move[1][0]

handCardSuit = move[1][1]

deckEmpty = move[2]

actions.append(Action.Action(1, (thisPlayerInd, handCardValue, handCardSuit), None, cardsAmount = 1))

if not deckEmpty:

actions.append(Action.Action(0, None, thisPlayerInd, cardsAmount = 1))

elif move[0] == 1:

handStackNames = move[1][0]

tableStackValue = move[2][0]

stackSize = move[3]

cardsToGive = []

for i in range(stackSize):

cardsToGive.append(handStackNames.pop(0))

actions.append(Action.Action(3, (thisPlayerInd, cardsToGive), None, cardsAmount = stackSize))

actions.append(Action.Action(2, tableStackValue, thisPlayerInd, cardsAmount = stackSize))

return actions

Card.py

import pygame

class Card(object):

"""Карта, її значення, розташування, зображення тощо"""

def \_\_init\_\_(self, scr, imgName):

self.scr = scr

self.coords = [0, 0]

#Card.size:

#0 - in deck

#1 - everywhere else

#2 - in human's hand

self.size = 0

cardName = imgName.split('.')[0].split('\_')

self.value = cardName[0]

self.suit = cardName[1]

self.sourceImage = pygame.image.load(f'assets/cards/{imgName}')

self.image = pygame.transform.scale(self.sourceImage, (48, 64))

self.rect = self.image.get\_rect()

self.rect.topleft = self.coords

def update(self, coords = None, size = None):

if coords is not None:

self.coords = coords

if size is not None:

self.size = size

if self.size == 1:

self.image = self.sourceImage

self.rect = self.image.get\_rect()

self.rect.topleft = self.coords

if self.size == 2:

self.image = pygame.transform.scale(self.sourceImage, (100, 139))

self.rect = self.image.get\_rect()

self.rect.topleft = self.coords

self.rect.topleft = self.coords

self.scr.blit(self.image, self.rect)

Table.py

import Card

class Table(object):

"""Ігровий стіл, на який викладають карти"""

def \_\_init\_\_(self, scr, mouse):

self.scr = scr

self.mouse = mouse

self.x0 = 10

self.y0 = 300

self.xmax = 1050

self.ymax = 525

self.cardKeys = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']

self.cards = {'2' : [], '3' : [], '4' : [], '5' : [], '6' : [], '7' : [], '8' : [], '9' : [], '10' : [], 'J' : [], 'Q' : [], 'K' : [], 'A' : []}

def update(self):

currTopLeft = [self.x0, self.y0]

for keyInd in range(len(self.cardKeys)):

if len(self.cards[self.cardKeys[keyInd]]) > 0:

for card in self.cards[self.cardKeys[keyInd]]:

card.update(currTopLeft, 1)

currTopLeft[0] += 25

currTopLeft[0] += 56

if keyInd < len(self.cardKeys) - 1:

if currTopLeft[0] + 25 \* len(self.cards[self.cardKeys[keyInd+1]]) + 50 > self.xmax:

currTopLeft = [self.x0, currTopLeft[1]+110]

def clickedStack(self):

"""Повертає значення та розмір стопки з 3 або 4 карт, по якій клікнув гравець"""

if self.mouse.isDown:

for key in self.cardKeys:

stackSize = len(self.cards[key])

if stackSize > 2:

for i in range(stackSize):

coords = self.cards[key][i].rect.topleft

if self.mouse.intersects((coords[0], coords[1], coords[0] + 75, coords[1] + 104)):

return (key, stackSize)

return None

def addCard(self, card):

"""Додати карту на стіл"""

self.cards[card.value].append(card)

def giveStack(self, value, stackOf3):

"""Видати стопку карт"""

if stackOf3 == True and len(self.cards[value]) == 4:

stack = self.cards[value][1:]

self.cards[value] = [self.cards[value][0]]

return stack

else:

stack = self.cards[value]

self.cards[value] = []

return stack

TextBox.py

import Card

import random

class Deck(object):

"""Колода, що містить усі карти"""

def \_\_init\_\_(self, scr):

self.scr = scr

self.cards = []

for value in ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']:

for suits in ['c', 'd', 'h', 's']:

self.cards.append(Card.Card(self.scr, f'{value}\_{suits}.png'))

random.shuffle(self.cards)

for i in range(len(self.cards)):

self.cards[i].coords = (1066 + 58 \* (i // 13), 9 + 67 \* (i % 13))

self.needToRefresh = False

def isEmpty(self):

return len(self.cards) == 0

def update(self):

if self.needToRefresh:

self.needToRefresh = False

for i in range(len(self.cards)):

self.cards[i].update((1064 + 58 \* (i // 13), 678 - 56 \* (i % 13)))

for i in range(len(self.cards)-1, -1, -1):

self.cards[i].update()

def draw(self):

"""Витягти карту з колоди"""

self.needToRefresh = True

if len(self.cards) > 0:

return self.cards.pop(0)

else:

return None

GameState.py

import copy

import time

class GameState(object):

"""Дані про можливий стан гри: карти на столі, в руках і колоді, та оцінка стану через мінімакс"""

def \_\_init\_\_(self, deckNames, handNames, tableNames, playerScores, currTurn, height):

self.deckNames = deckNames #for example [('3', 'c'), ('10', 's'), ('J', 'h'), ...]

self.handNames = handNames #for example [[('3', 'c'), ('10', 's'), ('J', 'h')], [...], ...]

self.tableNames = tableNames #dict like Table.cards

self.playerScores = playerScores #for example [3, 7, 0]

self.turn = currTurn #ind of current player

self.height = height #how many layers there are until we evaluate the static position

self.possibleMoves = None

self.childStates = None

self.bestMoveInd = None

self.stateScore = self.minimax(self.height, True)

def allEmpty(self):

if len(self.deckNames) == 0:

for hand in self.handNames:

if len(hand) > 0:

return False

return True

return False

@staticmethod

def handScore(hand, table): #for example [('3', 'c'), ('10', 's'), ('J', 'h')]

"""Статична оцінка руки"""

#we want better score if:

#the player has big value cards

#the player has multiple of them

order = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']

if len(hand) == 0:

return 0

values = []

valuesCount = []

for card in hand:

if len(table[card[0]]) < 2:

if card[0] in values:

valuesCount[values.index(card[0])] += 1

else:

values.append(card[0])

valuesCount.append(1)

for i in range(len(valuesCount)):

valuesCount[i] = ((valuesCount[i]-0.5) \*\* 1.2) \* (order.index(values[i]) \*\* 0.5)

return sum(valuesCount)

@staticmethod

def stacksScore(playerScore):

"""Статична оцінка балів, які гравець уже має"""

return playerScore \* 10

def evaluatePosition(self): #works only for 2 players

"""Повна статична оцінка стану (максимізуючий гравець - комп'ютер)"""

compScore = self.handScore(self.handNames[1], self.tableNames) + self.stacksScore(self.playerScores[1])

humanScore = self.handScore(self.handNames[0], self.tableNames) + self.stacksScore(self.playerScores[0])

return compScore - humanScore

def minimax(self, height, maximizingPlayer, alpha = -9999, beta = 9999): #works only for 2 players

"""Мінімакс із альфа-бета відсіканнями"""

if height == 0 or self.allEmpty():

return self.evaluatePosition()

self.possibleMoves = self.getPossibleMoves()

self.childStates = []

for move in self.possibleMoves:

self.childStates.append(self.getChildState(move))

if maximizingPlayer:

maxEval = -9999

for i in range(len(self.childStates)):

eval = self.childStates[i].stateScore

if eval > maxEval:

maxEval = eval

self.bestMoveInd = i

alpha = max(alpha, eval)

if beta <= alpha:

break

return maxEval

else:

minEval = 9999

for i in range(len(self.childStates)):

eval = self.childStates[i].stateScore

if eval < minEval:

minEval = eval

self.bestMoveInd = i

beta = min(beta, eval)

if beta <= alpha:

break

return minEval

def stackInHand(self):

"""Знайти, чи має гравець карти для того, щоб узяти карти зі столу"""

if len(self.handNames[self.turn]) == 3:

namesInStack = self.handNames[self.turn]

for card in namesInStack:

if card[0] != namesInStack[0][0]:

return (None, None)

return (namesInStack, 3)

elif len(self.handNames[self.turn]) == 4:

namesInStack = self.handNames[self.turn]

wrongCards = 0

for card in namesInStack:

if card[0] != namesInStack[0][0]:

wrongCards += 1

if wrongCards == 1:

#find index of wrongcard

for i in range(len(namesInStack)):

if namesInStack[i][0] != namesInStack[0][0]:

wrongInd = i

return (namesInStack[:wrongInd]+namesInStack[wrongInd+1:], 3)

elif wrongCards == 0:

return (namesInStack, 4)

elif namesInStack[1][0] == namesInStack[2][0] and namesInStack[3][0] == namesInStack[2][0]:

return (namesInStack[1:], 3)

return (None, None)

def stacksOnTable(self, maxValue):

"""Знайти найбільші стопки з трьох та чотирьох карт, які гравець може взяти"""

smallStackValues = []

bigStackValues = []

for key in self.tableNames:

if key == maxValue:

break

if len(self.tableNames[key]) == 3:

smallStackValues.append((key, 3))

elif len(self.tableNames[key]) == 4:

bigStackValues.append((key, 4))

stackValues = []

if len(bigStackValues) > 0:

stackValues.append(bigStackValues[-1])

if len(smallStackValues) > 0:

stackValues.append(smallStackValues[-1])

return stackValues

def getPossibleMoves(self):

"""Знайти можливі ходи, які будуть використані для створення об'єктів Action"""

#MoveType:

#0 - card from hand to table

# cardName = (value, suit) - value and suit of the card in hand

# deckEmpty : bool - whether the deck is empty

#1 - change cards for points

# handStack(cardNames = (value, suit), stackSize) - value of the stack that the player can get from the table

# tableStack(stackValue, stackSize)

# stackSize = min(tableStackSize, handStackSize)

moves = []

deckEmpty = (len(self.deckNames) == 0)

valuesToGive = []

for cardName in self.handNames[self.turn]:

if not cardName[0] in valuesToGive:

valuesToGive.append(cardName[0])

moves.append((0, cardName, deckEmpty))

handStack = self.stackInHand()

if handStack[0] is not None:

tableStacks = self.stacksOnTable(handStack[0][0][0])

for tableStack in tableStacks:

stackSize = min(handStack[1], tableStack[1])

moves.append((1, handStack, tableStack, stackSize))

return moves

def getChildState(self, move):

"""Створення стану, в який перейде гра, якщо зробити хід move"""

if move[0] == 0:

handCardValue = move[1][0]

handCardSuit = move[1][1]

deckEmpty = move[2]

newTable = copy.deepcopy(self.tableNames)

newHands = copy.deepcopy(self.handNames)

newTable[handCardValue].append(newHands[self.turn].pop(newHands[self.turn].index((handCardValue, handCardSuit))))

if not deckEmpty:

newHands[self.turn].append(self.deckNames[0])

newDeck = self.deckNames[1:]

else:

newDeck = self.deckNames

return GameState(newDeck, newHands, newTable, self.playerScores, (self.turn+1) % len(self.handNames), self.height-1)

elif move[0] == 1:

handStackNames = move[1][0]

tableStackValue = move[2][0]

stackSize = move[3]

newTable = copy.deepcopy(self.tableNames)

cardsFromTable = []

for i in range(stackSize):

cardsFromTable.append(newTable[tableStackValue].pop(0))

newHands = copy.deepcopy(self.handNames)

cardsFromHand = []

for i in range(stackSize):

cardsFromHand.append(newHands[self.turn].pop(newHands[self.turn].index(handStackNames[i])))

newTable[handStackNames[0][0]] += cardsFromHand

newHands[self.turn] += cardsFromTable

newPlayerScores = self.playerScores[:]

newPlayerScores[self.turn] += stackSize - 2

return GameState(self.deckNames, newHands, newTable, newPlayerScores, (self.turn+1) % len(self.handNames), self.height-1)

Action.py

class Action(object):

"""Клас, що описує дії, виконувані гравцями, у вигляді інструкцій для програми)"""

def \_\_init\_\_(self, actionType, source, destination, steps = 10, cardsAmount = 1):

#0 - card from deck

#1 - card to table

#2 - cards from table to hand

#3 - cards from hand to table (to allow action of type 2)

self.actionType = actionType

self.source = source

self.destination = destination

self.steps = steps

self.cardsAmount = cardsAmount

#if actionType == 0:

#source = None (deck)

#destination = playerInd

#cardsAmount = 1

#if actionType == 1:

#source = (playerInd, chosenCardValue, chosenCardSuit)

#destination = None (table)

#cardsAmount = 1

#if actionType == 2:

#source = chosenStackValue

#destination = playerInd

#cardsAmount = 3 or 4

#if actionType == 3:

#source = (playerInd, [(card.value, card.suit) for card in movedCards])

#destination = None (table)

#cardsAmount = 3 or 4

self.cardsAmount = cardsAmount

self.paths = None

self.step = 0

self.movedCards = None

def setPaths(self, cards, players = None, table = None):

"""Встановлення координат, якими рухатимуться карти"""

self.movedCards = cards

if self.actionType == 0:

#players needed

startCoords = [cards[0].coords]

endCoords = [players[self.destination].cardCoords[len(players[self.destination].hand)+1][-1]]

elif self.actionType == 1:

#table needed

startCoords = [cards[0].coords]

tableSection = cards[0].value

if len(table.cards[tableSection]) > 0:

endCoords = [(table.cards[tableSection][-1].rect.topleft[0] + 25, table.cards[tableSection][-1].rect.topleft[1])]

else:

currInd = table.cardKeys.index(tableSection)

while len(table.cards[table.cardKeys[currInd-1]]) == 0 and currInd > 0:

currInd -= 1

if currInd == 0:

endCoords = [(table.x0, table.y0)]

else:

endCoords = [(table.cards[table.cardKeys[currInd-1]][-1].rect.topleft[0]+100, table.cards[table.cardKeys[currInd-1]][-1].rect.topleft[1])]

elif self.actionType == 2: #now it's giving the cards to the player stacks, change to put them into the hand

#players needed

startCoords = [cards[i].coords for i in range(self.cardsAmount)]

if self.cardsAmount == 3:

endCoords = [players[self.destination].cardCoords[self.cardsAmount][i] for i in range(3)]

else:

endCoords = [players[self.destination].cardCoords[self.cardsAmount][i] for i in range(4)]

elif self.actionType == 3:

#table needed

startCoords = [cards[i].coords for i in range(self.cardsAmount)]

tableSection = cards[0].value

if len(table.cards[tableSection]) > 0:

endCoords = [(table.cards[tableSection][-1].rect.topleft[0] + 25\*(i+1), table.cards[tableSection][-1].rect.topleft[1]) for i in range(self.cardsAmount)]

else:

currInd = table.cardKeys.index(tableSection)

while len(table.cards[table.cardKeys[currInd-1]]) == 0:

currInd -= 1

endCoords = [(

table.cards[table.cardKeys[currInd-1]][-1].rect.topleft[0]+100+(i)\*25, # i-self.cardsAmount

table.cards[table.cardKeys[currInd-1]][-1].rect.topleft[1]

) for i in range(self.cardsAmount) ]

start = [(startCoords[i][0], startCoords[i][1]) for i in range(self.cardsAmount)]

end = [(endCoords[i][0], endCoords[i][1]) for i in range(self.cardsAmount)]

self.paths = []

for step in range(self.steps):

self.paths.append([])

for cardInd in range(self.cardsAmount):

self.paths[step].append((

start[cardInd][0]+(end[cardInd][0]-start[cardInd][0])\*step/(self.steps-1),

start[cardInd][1]+(end[cardInd][1]-start[cardInd][1])\*step/(self.steps-1)

))

def update(self):

"""Пересунути карти на крок"""

if self.step > self.steps - 1:

return self.movedCards

else:

for i in range(self.cardsAmount):

self.movedCards[i].update(self.paths[self.step][i], 1)

self.step += 1

return 0

EndScreen.py

import TextBox

import Button

import pygame

class EndScreen(object):

"""Вікно, що з'являється після кінця гри, з таблицею балів"""

def \_\_init\_\_(self, scr, mouse, computerNames, scores):

self.scr = scr

self.mouse = mouse

self.background = pygame.image.load('assets/endScreenBackground.png')

scoresOrder = ['Ви']

scoresOrder.extend(computerNames)

for i in range(len(scores)-1):

for j in range(i+1, len(scores)):

if scores[i] < scores[j]:

scores[i], scores[j] = scores[j], scores[i]

scoresOrder[i], scoresOrder[j] = scoresOrder[j], scoresOrder[i]

if scores[0] == scores[1]:

winnerMessage = 'Нічия!'

elif scoresOrder [0] == 'Ви':

winnerMessage = 'Ви перемогли!'

else:

winnerMessage = f'{scoresOrder[0]} переміг!'

self.title = TextBox.TextBox(self.scr, winnerMessage, (350, 125, 700, 225), (255, 255, 255), -1, 28)

self.tableHeadNames = TextBox.TextBox(self.scr, "Гравець", (350, 250, 600, 300), (255, 255, 255), 1, 28)

self.tableHeadScores = TextBox.TextBox(self.scr, "Бали", (600, 250, 700, 300), (255, 255, 255), 1, 28)

self.scoreBoxes = []

self.nameBoxes = []

for i in range(len(scores)):

self.nameBoxes.append(TextBox.TextBox(self.scr, scoresOrder[i], (350, 300+i\*50, 600, 350+i\*50), (255, 255, 255), 1, 28))

self.scoreBoxes.append(TextBox.TextBox(self.scr, scores[i], (600, 300+i\*50, 700, 350+i\*50), (255, 255, 255), 1, 28))

self.menuButton = Button.Button(self.mouse, self.scr, 'До головного меню', (350, 525, 700, 625), (255, 255, 255), 2, 28)

def update(self):

self.scr.blit(self.background, (300, 100))

self.title.update()

self.tableHeadNames.update()

self.tableHeadScores.update()

for i in range(len(self.scoreBoxes)):

self.nameBoxes[i].update()

self.scoreBoxes[i].update()

self.menuButton.update()

if self.menuButton.isPressed:

return 1

else:

return 0

### Приклади роботи

На рисунках 3.1-3.3 показані приклади роботи програми.

Рисунок 3.1 – Головне меню гри

Рисунок 3.2 – Ігрове поле



Рисунок 3.3 – Результати гри

Висновок

В рамках даної лабораторної роботи було розглянуто роботу алгоритму мінімакс на прикладі карткової гри “Ремінь”. Досліджено переваги та недоліки алгоритму, а також оптимізовано його з використанням альфа-бета відсікань.

Розроблено програмне забезпечення для роботи гри проти комп’ютера, керованого алгоритмом мінімакс, з допомогою мови програмування Python із використанням модуля pygame. Створено три рівні складності комп’ютера.

Проведено аналіз роботи програми за різних умов та оптимізовано роботу рекурсивного алгоритму.

Критерії оцінювання

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