**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Трикош І. В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М. М.*

Київ 2023

Зміст

[1 Мета лабораторної роботи 3](#_Toc86770239)

[2 Завдання 4](#_Toc86770240)

[3 Виконання 7](#_Toc86770241)

[3.1 Програмна реалізація алгоритму 7](#_Toc86770242)

[3.1.1 Вихідний код 7](#_Toc86770243)

[3.1.2 Приклади роботи 29](#_Toc86770244)

[Висновок 31](#_Toc86770245)

[Критерії оцінювання 32](#_Toc86770246)

# 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 |
| 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

## Програмна реалізація алгоритму

### Вихідний код

***game.py***

import random

import time

import constants as const

class Algorithm():

"""

Клас з алгоритмом альфа-бета відсікань

:param game: об'єкт класу Game

:param max\_score: максимальний результат, що може отримати гравець

:param min\_score: мінімальний результат, що може отримати гравець

"""

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

"""Конструктор"""

self.game = game

self.max\_score = self.game.size \*\* 2

self.min\_score = -self.max\_score

return

def find\_best\_move(self, field, index\_player):

"""Знаходження найкращого ходу"""

scores = {}

is\_max = self.game.players[index\_player].is\_max

indexes = [[i, j] for i in range(self.game.size) for j in range(self.game.size)]

random.shuffle(indexes)

for i, j in indexes:

if field[i][j] == '0':

if self.game.is\_move(field, index\_player, [i, j]):

field[i][j] = self.game.players[index\_player].color

move\_score = self.alpha\_beta\_prunning(field, 0, not is\_max, -float('inf'), float('inf'))

field[i][j] = '0'

scores[i, j] = move\_score

move = None

score = None

if self.game.game\_mode == 'easy':

if is\_max:

score = float('inf')

for i in scores.keys():

if scores[i] < score:

score = scores[i]

move = i

else:

score = -float('inf')

for i in scores.keys():

if scores[i] > score:

score = scores[i]

move = i

elif self.game.game\_mode == 'medium':

values\_sum = 0

for i in scores.values():

values\_sum += i

average\_value = values\_sum // len(scores.values())

difference = float('inf')

for i in scores.keys():

if abs(scores[i] - average\_value) < difference:

difference = abs(scores[i] - average\_value)

move = i

score = scores[i]

if not move:

move = random.choice(scores.keys())

score = scores[move]

elif self.game.game\_mode == 'hard':

if is\_max:

score = -float('inf')

for i in scores.keys():

if scores[i] > score:

score = scores[i]

move = i

else:

score = float('inf')

for i in scores.keys():

if scores[i] < score:

score = scores[i]

move = i

return score, move

def alpha\_beta\_prunning(self, field, depth, is\_max, alpha, beta):

"""Альфа-бета відсікання"""

index\_player = 0

if self.game.players[index\_player].is\_max != is\_max:

index\_player = 1

result = self.game.is\_looser(field, index\_player)

if result:

if is\_max:

return self.min\_score + depth

else:

return self.max\_score - depth

if is\_max:

best\_score = -float('inf')

indexes = [[i, j] for i in range(self.game.size) for j in range(self.game.size)]

random.shuffle(indexes)

for i, j in indexes:

if field[i][j] == '0':

if self.game.is\_move(field, index\_player, [i, j]):

field[i][j] = self.game.players[index\_player].color

best\_score = max(best\_score, self.alpha\_beta\_prunning(field, depth + 1, not is\_max, alpha, beta))

field[i][j] = '0'

alpha = max(alpha, best\_score)

if beta <= alpha:

break

return best\_score

else:

best\_score = float('inf')

indexes = [[i, j] for i in range(self.game.size) for j in range(self.game.size)]

random.shuffle(indexes)

for i, j in indexes:

if field[i][j] == '0':

if self.game.is\_move(field, index\_player, [i, j]):

field[i][j] = self.game.players[index\_player].color

best\_score = min(best\_score, self.alpha\_beta\_prunning(field, depth + 1, not is\_max, alpha, beta))

field[i][j] = '0'

beta = min(beta, best\_score)

if beta <= alpha:

break

return best\_score

return

class Player():

"""

Гравець

:param name: ім'я гравця

:param color: колір гравця

:param is\_max: чи є гравець max

"""

def \_\_init\_\_(self, name, color, is\_max):

"""Створення гравця"""

self.name = name

self.color = color

self.is\_max = is\_max

return

def set\_is\_max(self, is\_max):

"""Встановлення типу гравця"""

self.is\_max = is\_max

return

def make\_move(self, game, is\_PC, \*args):

"""Зробити хід"""

if is\_PC:

algorithm = Algorithm(game)

best\_score, best\_move = algorithm.find\_best\_move(game.game\_field, game.index\_current\_player)

return best\_move

return

class Game():

"""

Гра

:param size: розмір ігрового поля

:param game\_field: ігрове поле

:param colors: кольори

:param players: гравці

:param players\_number: кількість гравців

:param index\_current\_player: індекс поточного гравця (гравця, яких зараз ходить)

:param winner: переможець гри

:param game\_mode: режим гри

"""

def \_\_init\_\_(self, Game\_Field, COLORS, GAME\_MODE):

"""Створення гри"""

self.size = len(Game\_Field)

self.game\_field = Game\_Field

self.colors = COLORS

random.shuffle(self.colors)

self.players = [Player('Player 1 (PC)', self.colors[0], False), Player('Player 2 (user)', self.colors[1], False)]

self.players\_number = len(self.players)

self.index\_current\_player = self.set\_first\_player()

self.players[self.index\_current\_player].set\_is\_max(True)

self.winner = None

self.game\_mode = GAME\_MODE

return

def set\_first\_player(self):

"""Визначити першого гравця"""

return random.randint(0, self.players\_number - 1)

def set\_next\_player(self):

"""Визначити наступного гравця"""

self.index\_current\_player = (self.index\_current\_player + 1) % self.players\_number

return self.index\_current\_player

def set\_cell(self, cell, color):

"""Зафарбувати комірку у заданий колір"""

if self.game\_field[cell[0]][cell[1]] == '0':

self.game\_field[cell[0]][cell[1]] = color

return

def output(self):

"""Вивести ігрове поле поля"""

print('\nThe game\'s field:')

for i in self.game\_field:

for j in i:

color = None

if j[0] == '-':

color = '\033[40m'

elif j[0] == '0':

color = '\033[47m'

elif j[0] == 'r':

color = '\033[41m'

else:

color = '\033[44m'

print(color, j[0], '\033[00m', sep='', end='')

print()

return

def is\_move(self, field, index\_player, cell):

"""Перевірити, чи може гравець походити у цю клітинку (cell - список або множина у вигляді [i, j] або (i, j))"""

color\_opp = self.colors[0] if self.players[index\_player].color == self.colors[1] else self.colors[1]

if field[cell[0]][cell[1]] == '0':

if cell[0] - 1 != -1:

if field[cell[0] - 1][cell[1]] == color\_opp:

return False

if cell[0] + 1 != self.size:

if field[cell[0] + 1][cell[1]] == color\_opp:

return False

if cell[1] - 1 != -1:

if field[cell[0]][cell[1] - 1] == color\_opp:

return False

if cell[1] + 1 != self.size:

if field[cell[0]][cell[1] + 1] == color\_opp:

return False

return True

else:

return False

return

def is\_looser(self, field, index\_player):

"""Перевірити, чи програв гравець"""

for i in range(self.size):

for j in range(self.size):

if not field[i][j] == '-':

if field[i][j] == '0':

if self.is\_move(field, index\_player, [i, j]):

return False

return True

def is\_winner(self, field, index\_player):

"""Перевірити, чи виграв гравець"""

index\_opp\_player = (index\_player + 1) % self.players\_number

return self.is\_looser(field, index\_opp\_player)

def main():

"""Початок консольної гри"""

game\_field = []

for i in range(len(const.GAME\_FIELD)):

temp = []

for j in range(len(const.GAME\_FIELD[i])):

temp += [const.GAME\_FIELD[i][j]]

game\_field += [temp]

game = Game(game\_field, const.COLORS, const.GAME\_MODES[2])

while True:

print('-' \* 100)

print('\033[36m', game.players[game.index\_current\_player].name, ' : ',

game.players[game.index\_current\_player].color, '\033[00m', sep='')

game.output()

if game.index\_current\_player:

numb = int(input('\033[93mInput the cell\'s number: \033[00m'))

i, j = numb // game.size, numb % game.size

if not (0 <= numb < game.size \*\* 2) or game.game\_field[i][j] != '0' or not game.is\_move(game.game\_field, game.index\_current\_player, [i, j]):

print('\033[91mThe number is wrong\033[00m')

continue

else:

game.set\_cell([i, j], game.players[game.index\_current\_player].color)

if game.is\_winner(game.game\_field, game.index\_current\_player):

game.winner = game.players[game.index\_current\_player]

game.output()

print('\033[47m\033[90m', 'The winnner is: ', game.winner.name, ' (color: ', game.winner.color, ')', '\033[00m', sep='')

break

game.set\_next\_player()

else:

algorithm = Algorithm(game)

time\_start = time.time\_ns()

best\_score, best\_move = algorithm.find\_best\_move(game.game\_field, game.index\_current\_player)

time\_end = time.time\_ns()

print('Score for PC: ', best\_score, ' Time in miliseconds:', (time\_end-time\_start) / 1000 / 1000)

game.set\_cell([best\_move[0], best\_move[1]], game.players[game.index\_current\_player].color)

if game.is\_winner(game.game\_field, game.index\_current\_player):

game.winner = game.players[game.index\_current\_player]

game.output()

print('\033[47m\033[90m', 'The winnner is: ', game.winner.name, ' (color: ', game.winner.color, ')', '\033[00m', sep='')

break

game.set\_next\_player()

return

if \_\_name\_\_ == "\_\_main\_\_":

main()

***constants.py***

# Ігрові константи

GAME\_FIELD = [['-', '0', '0', '-'],

['0', '0', '0', '0'],

['0', '0', '0', '0'],

['-', '0', '0', '-']]

COLORS = ['red', 'blue']

GAME\_MODES = ['easy', 'medium', 'hard']

***interface.py***

import tkinter as tk

import tkinter.messagebox

import game as Game

from functools import partial

import constants as const

class Labels\_and\_Moves():

"""

Представлення ігрового поля та відтворення ходів гравців

:param master: вікно з ігровим полем

:param mainwindow: стартове вікно

:param Labels: ігрові клітинки

:param game: об'єкт класу Game.Game

:param label: поле з ім'ям гравця, що зараз ходить

"""

def \_\_init\_\_(self, master, mainwindow, x\_s, y\_s, game, label):

"""Створити ігрові клітинки"""

self.master = master

self.mainwindow = mainwindow

self.Labels = []

self.create\_labels(x\_s, y\_s)

self.game = game

self.label = label

return

def make\_move\_user(self, row, column):

"""Хід користувача"""

index = row \* self.game.size + column

is\_move = False

if not self.game.index\_current\_player:

tkinter.messagebox.showerror(title='Snort', message='Зараз ходить ' + self.game.players[

self.game.index\_current\_player].name + '!')

elif self.game.is\_move(self.game.game\_field, self.game.index\_current\_player, [row, column]):

self.set\_color(row, column, self.game.players[self.game.index\_current\_player].color)

self.game.set\_cell([row, column], self.game.players[self.game.index\_current\_player].color)

is\_move = True

if self.is\_winner():

is\_move = False

return is\_move

self.game.set\_next\_player()

self.label.config(text=self.game.players[self.game.index\_current\_player].name,

foreground=self.game.players[self.game.index\_current\_player].color)

else:

if self.game.game\_field[row][column] != '-':

tkinter.messagebox.showerror(title='Snort', message='У цю клітинку походити неможливо!')

return is\_move

def make\_move\_pc(self):

"""Хід комп'ютера"""

move = self.game.players[self.game.index\_current\_player].make\_move(self.game, True if not self.game.index\_current\_player else False)

if move:

self.set\_color(move[0], move[1], self.game.players[self.game.index\_current\_player].color)

self.game.set\_cell([move[0], move[1]], self.game.players[self.game.index\_current\_player].color)

if self.is\_winner():

return

self.game.set\_next\_player()

self.label.config(text=self.game.players[self.game.index\_current\_player].name,

foreground=self.game.players[self.game.index\_current\_player].color)

else:

return

return

def change\_color(self, row, column, \*args):

"""Змінити колір поля Label (зробити ходи користувача та комп'ютера)"""

is\_move = self.make\_move\_user(row, column)

if is\_move:

self.make\_move\_pc()

if not self.game.winner:

self.paint\_unmoved\_cells()

return

def paint\_unmoved\_cells(self):

"""Зафарбувати клітинки, в які неможна походити"""

for i in range(self.game.size):

for j in range(self.game.size):

if self.game.game\_field[i][j] == '0' and not self.game.is\_move(self.game.game\_field, self.game.index\_current\_player, [i, j]):

self.set\_color(i, j, '#B3B1B2')

return

def is\_winner(self):

"""Якщо є переможець, то вивести повідомлення та закрити вікно"""

if self.game.is\_winner(self.game.game\_field, self.game.index\_current\_player):

self.game.winner = self.game.players[self.game.index\_current\_player]

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

self.Labels[i].bind('<Button-1>', partial(self.nothing))

tkinter.messagebox.showinfo(title='Snort', message='Переможець - ' + self.game.winner.name + '; колір - ' + self.game.winner.color)

self.mainwindow.deiconify()

self.master.destroy()

return True

return False

def create\_labels(self, x\_s, y\_s):

"""Створити поле гри"""

row = 0

column = 0

for y in y\_s:

for x in x\_s:

color = 'black' if (x == x\_s[0] or x == x\_s[-1]) and (y == y\_s[0] or y == y\_s[-1]) else 'white'

label\_frame = tk.Frame(self.master, width=48, height=48, background=color)

label\_frame.place(x=x, y=y)

label\_frame.bind('<Button-1>', partial(self.change\_color, row, column))

self.Labels += [label\_frame]

column += 1

row += 1

column = 0

return

def set\_color(self, i, j, color):

"""Змінити колір клітинки"""

index = i \* self.game.size + j

self.Labels[index].config(bg=color)

return

def nothing(self, \*args):

"""Пустий метод (для того, щоб зміна кольору не відбувалась після завершення гри"""

return

class BoardWindow():

"""

Вікно з ігровим полем

:param master: вікно з ігровим полем

:param mainwindow: початкове вікно

:param game\_mode: режим гри

:param game\_field: ігрове поле

:param GAME: об'єкт класу Game.Game

:param LABEL: поле з ім'ям поточного гравця

:param Labels: ігрове поле

"""

def \_\_init\_\_(self, master, mainwindow, game\_mode):

"""Створити вікно"""

self.master = master

self.master.title('Snort')

self.master.geometry('300x300+475+190')

self.master.resizable(width=False, height=False)

self.master.iconbitmap("icons/icon.ico")

self.mainwindow = mainwindow

self.game\_mode = game\_mode

self.game\_field = []

for i in range(len(const.GAME\_FIELD)):

temp = []

for j in range(len(const.GAME\_FIELD[i])):

temp += [const.GAME\_FIELD[i][j]]

self.game\_field += [temp]

self.GAME = Game.Game(self.game\_field, const.COLORS, self.game\_mode)

self.create\_lines()

self.LABEL = tk.Label(self.master, text=self.GAME.players[self.GAME.index\_current\_player].name, foreground=self.GAME.players[self.GAME.index\_current\_player].color, font=("Helvetica", 18))

self.LABEL.place(relx=0.5, y=25, anchor='center')

self.Labels = None

self.create\_labels()

self.Labels.make\_move\_pc()

if not self.GAME.winner:

self.Labels.paint\_unmoved\_cells()

self.master.protocol("WM\_DELETE\_WINDOW", self.exit)

return

def create\_lines(self):

"""Створення ліній"""

canvas = tk.Canvas(self.master)

canvas.create\_line(46.5, 107.5, 253.5, 107.5, width=5)

canvas.create\_line(46.5, 160.5, 253.5, 160.5, width=5)

canvas.create\_line(46.5, 213.5, 253.5, 213.5, width=5)

canvas.create\_line(96.5, 57.5, 96.5, 264.5, width=5)

canvas.create\_line(149.5, 57.5, 149.5, 264.5, width=5)

canvas.create\_line(202.5, 57.5, 202.5, 264.5, width=5)

canvas.pack()

return

def create\_labels(self):

"""Створення клітинок"""

x\_s = [46.5, 99.5, 152.5, 205.5]

y\_s = [57.5, 110.5, 163.5, 216.5]

self.Labels = Labels\_and\_Moves(self.master, self.mainwindow, x\_s, y\_s, self.GAME, self.LABEL)

return

def exit(self):

"""Закрити вікно"""

self.mainwindow.deiconify()

self.master.destroy()

return

class Game\_modes():

"""

Вікно з вибором ігрового режиму

:param master: поточне вікно

:param mainwindow: початкове вікно

:param app: вікно з ігровим полем

:param game\_mode: обраний режим гри

"""

def \_\_init\_\_(self, master, mainwindow):

"""Створення вікна та кнопок"""

self.master = master

self.mainwindow = mainwindow

self.app = None

self.master.title('Snort')

self.master.geometry('300x300+475+190')

self.master.resizable(width=False, height=False)

self.master.iconbitmap("icons/icon.ico")

self.master.config(background='#616161')

self.game\_mode = None

self.label = tk.Label(self.master, text='ОБЕРІТЬ РЕЖИМ ГРИ', background='#d3d3d3', font=('Times, 15'))

self.label.place(x=40, y=20, width=220, height=30)

self.button1 = tk.Button(self.master, text='ЛЕГКИЙ', font=('Times, 15'), background='#d3d3d3', command=lambda: self.set\_game\_mode(0))

self.button1.place(x=90, y=70, width=120, height=40)

self.button2 = tk.Button(self.master, text='СЕРЕДНІЙ', font=('Times, 15'), background='#d3d3d3', command=lambda: self.set\_game\_mode(1))

self.button2.place(x=90, y=120, width=120, height=40)

self.button3 = tk.Button(self.master, text='ВАЖКИЙ', font=('Times, 15'), background='#d3d3d3', command=lambda: self.set\_game\_mode(2))

self.button3.place(x=90, y=170, width=120, height=40)

self.button4 = tk.Button(self.master, text='НАЗАД', font=('Times, 15'), background='#d3d3d3', command=self.close\_window)

self.button4.place(x=90, y=240, width=120, height=40)

self.master.protocol("WM\_DELETE\_WINDOW", self.close\_window)

return

def set\_game\_mode(self, index, \*args):

"""Встановлення ігрового режиму"""

if 0 <= index < len(const.GAME\_MODES):

self.game\_mode = const.GAME\_MODES[index]

self.start\_game()

return

def start\_game(self):

"""Початок гри (запуск вікна з ігровим полем)"""

if self.game\_mode:

self.master.withdraw()

self.newWindow = tk.Toplevel(self.master)

self.app = BoardWindow(self.newWindow, self.mainwindow, self.game\_mode)

return

def close\_window(self):

"""Закрити вікно"""

self.mainwindow.deiconify()

self.master.destroy()

return

class MainMenu():

"""

Головне меню

:param master: поточне вікно

:param app: вікно з вибором ігрового режиму

"""

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

"""Створення вікна"""

self.master = master

self.app = None

self.master.title('Snort')

self.master.geometry('300x300+475+190')

self.master.resizable(width=False, height=False)

self.master.iconbitmap("icons/icon.ico")

self.master.config(background='#616161')

self.label = tk.Label(self.master, text='SNORT', background='#d3d3d3', font=('Arial, 20'))

self.label.place(x=85, y=30, width=130, height=60)

self.button1 = tk.Button(self.master, text='ГРАТИ', font=('Arial, 15'), background='#d3d3d3', command=self.choose\_game\_mode)

self.button1.place(x=85, y=130, width=130, height=60)

self.button2 = tk.Button(self.master, text='ВИЙТИ', font=('Arial, 15'), background='#d3d3d3', command=self.exit)

self.button2.place(x=85, y=210, width=130, height=60)

self.master.protocol("WM\_DELETE\_WINDOW", self.exit)

return

def choose\_game\_mode(self):

"""Створення вікна з вибором ігрового режиму"""

self.master.withdraw()

self.newWindow = tk.Toplevel(self.master)

self.app = Game\_modes(self.newWindow, self.master)

return

def exit(self):

"""Закрити вікно"""

self.master.destroy()

return

def main():

"""Початок програми"""

window = tk.Tk()

app = MainMenu(window)

window.mainloop()

return

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

main()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

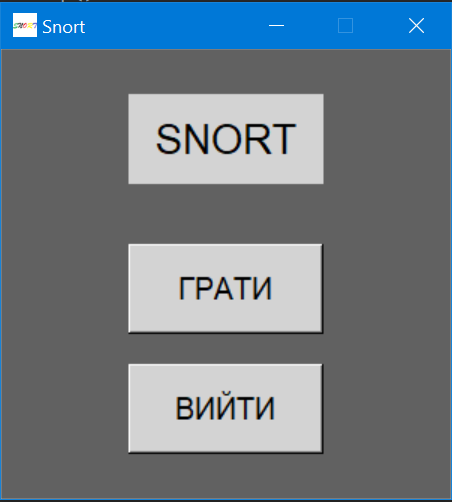


Рисунок 3.1 – Стартове вікно (тиснемо «Грати»)

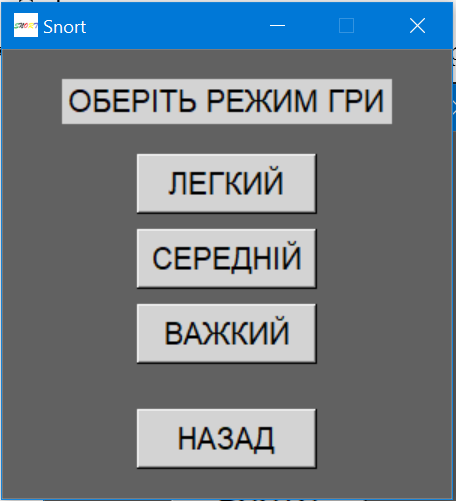


Рисунок 3.2 – Вікно вибору режиму гри (тиснемо «Середній»)

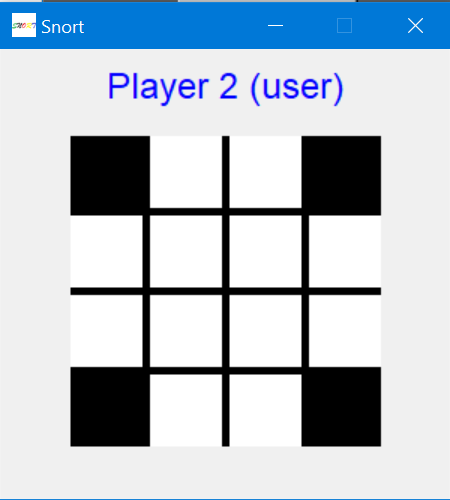


Рисунок 3.3 – Початок гри

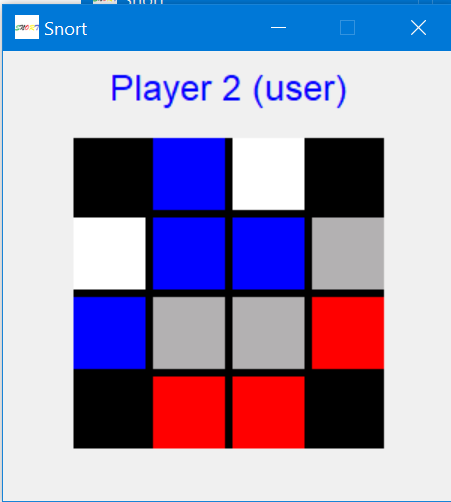


Рисунок 3.4 – Кінець гри (користувач переміг)

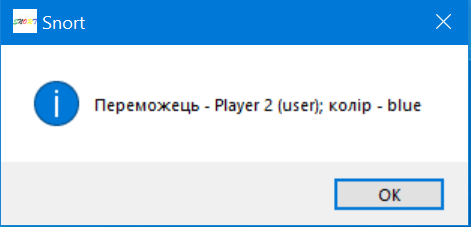


Рисунок 3.5 – Повідомлення про перемогу користувача

Висновок

В рамках даної лабораторної роботи я вивчив основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Я ознайомив з підходами до програмування алгоритмів штучного інтелекту в іграх з повною інформацією.

Я розробив алгоритм альфа-бета відсікань для гри Snort. Цей алгоритм працює наступним чином:

Нехай зараз ходить гравець max. Алгоритм проходиться по всьому доступному ігровому полю, вибираючи кожну клітинку поля і ходячи у неї. Далі алгоритм імітує хід гравця min, далі знову max і так до термінального вузла. Термінальний вузол повертає кількість очок, які заробить гравець. І так по всім термінальним вузлам, максимізуючи результат для гравця max та мінімізуючи результат для гравця min. Якщо на певному етапі для гравця один з попередніх вузлів кращий за поточний, то гравець його не обере, тому що це не оптимально, і, як наслідок, цей вузол буде відсічений.

Для інтерфейсної частини програми я використав Python Tkinter.

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).