**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 № 5 з дисципліни

«Проектування алгоритмів»

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

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІТ-04 Стрільчук Михайло*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc52291748)

[2 Завдання 4](#_Toc52291749)

[3 Виконання 6](#_Toc52291750)

[3.1 Програмна реалізація алгоритму 6](#_Toc52291752)

[3.1.1 Вихідний код 6](#_Toc52291753)

[3.1.2 Приклади роботи 6](#_Toc52291754)

[3.3 Тестування алгоритму 6](#_Toc52291755)

[Висновок 7](#_Toc52291756)

[Критерії оцінювання 8](#_Toc52291757)

# 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.

# Завдання

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) + 1балл.

Зробити узагальнений висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти

|  |  |
| --- | --- |
| **№** | **Варіант** |
| 1 | Баше https://ru.wikipedia.org/wiki/Баше\_(игра) |
| 2 | Нейтріко http://www.iggamecenter.com/info/ru/neutreeko.html |
| 3 | Точки https://ru.wikipedia.org/wiki/Точки\_(игра) |
| 4 | Dots and Boxes https://ru.wikipedia.org/wiki/Палочки\_(игра) |
| 5 | Алемунгула http://www.iggamecenter.com/info/ru/alemungula.html |
| 6 | Snakes http://www.papg.com/show?3AE4 |
| 7 | Cram https://en.wikipedia.org/wiki/Cram\_(game) |
| 8 | Тіко http://www.iggamecenter.com/info/ru/teeko.html |
| 9 | Obstruction 8х8 http://www.papg.com/show?2XMX |
| 10 | Gale http://www.papg.com/show?1TPI |
| 11 | Гомоку https://ru.wikipedia.org/wiki/Гомоку |
| 12 | Нім https://ru.wikipedia.org/wiki/Ним\_(игра) |
| 13 | Клоббер http://www.iggamecenter.com/info/ru/clobber.html |
| 14 | Hackenbush http://www.papg.com/show?1TMP |
| 15 | Лінкідж (більше) http://www.iggamecenter.com/info/ru/linkage.html |
| 16 | Заєць и Вовки (за Зайця) http://www.iggamecenter.com/info/ru/foxh.html |
| 17 | 3D Noughts and Crosses 4 x 4 x 4 http://www.papg.com/show?1TND |
| 18 | Domineering 8х8 http://www.papg.com/show?1TX6 |
| 19 | Транспозиція http://www.iggamecenter.com/info/ru/transposition.html |
| 20 | Заєць и Вовки (за Вовків) http://www.iggamecenter.com/info/ru/foxh.html |
| 21 | Вари http://www.iggamecenter.com/info/ru/oware.html |
| 22 | Калах http://www.iggamecenter.com/info/ru/kalah.html |
| 23 | Реверсі https://ru.wikipedia.org/wiki/Реверси |
| 24 | Лінкідж (менше) http://www.iggamecenter.com/info/ru/linkage.html |
| 25 | Три мушкетери (за мушкетерів) http://www.iggamecenter.com/info/ru/threemusk.html |
| 26 | Сим https://ru.wikipedia.org/wiki/Сим\_(игра) |
| 27 | Col http://www.papg.com/show?2XLY |
| 28 | Snort http://www.papg.com/show?2XM1 |
| 29 | Chomp http://www.papg.com/show?3AEA |
| 30 | Три мушкетери (за гвардейців) http://www.iggamecenter.com/info/ru/threemusk.html |

# Виконання

## Програмна реалізація алгоритму

https://github.com/Mishanych/Algorithms/tree/main/laba5

### Вихідний код

import pygame  
from copy import deepcopy  
import time  
  
# constants  
WIDTH, HEIGHT = 800, 800  
ROWS, COLS = 8, 8  
SQUARE\_SIZE = HEIGHT//ROWS  
  
# colours  
RED = (255, 0, 0)  
WHITE = (255, 255, 255)  
BLACK = (0, 0, 0)  
BLUE = (0, 0, 255)  
BG = (75, 100, 130)  
GOLD = (255, 215, 0)  
  
# main window  
window = pygame.display.set\_mode((WIDTH, HEIGHT))  
pygame.display.set\_caption('Hare and Wolfs')  
  
  
class Board:  
 def \_\_init\_\_(self):  
 self.board = []  
 self.board\_debug = []  
 self.create\_board()  
 self.harePos = (7, 0)  
 self.prevHarePos = (0, 0)  
  
 # draw table  
 def draw\_board(self, window):  
 window.fill(BLACK)  
 for row in range(ROWS):  
 for col in range(row % 2, COLS, 2):  
 pygame.draw.rect(window, WHITE, (row \* SQUARE\_SIZE, col \* SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE))  
  
 def evaluate(self):  
 if len(self.get\_valid\_moves(self.get\_piece(self.harePos[0], self.harePos[1]))) == 0:  
 return 10000000  
 return -(7 - self.harePos[0]) \* 10 - len(self.get\_valid\_moves(self.get\_piece(self.harePos[0], self.harePos[1]))) \* 4 - self.distance\_to\_hare() \* 8 - self.distance\_between\_wolfs() \* 6 - self.will\_win()  
  
 # hare evaluation depends on height, distance from wolfs, number of moves  
 def alternate\_evaluate(self):  
 return 8 \*\* abs(self.harePos[0] - 7) + self.distance\_to\_hare() \* 2 + 4 \*\* abs(4 - self.harePos[1]) - self.evaluate()  
  
 def get\_all\_pieces(self, color):  
 pieces = []  
 for row in self.board:  
 for piece in row:  
 if piece != 0 and piece.color == color:  
 pieces.append(piece)  
 # print(pieces)  
 return pieces  
  
 def distance\_to\_hare(self):  
 distance = 0  
 for piece in self.get\_all\_pieces(BLACK):  
 distance += abs(self.harePos[0] - piece.row) + abs(self.harePos[1] - piece.col)  
 return distance  
  
 def distance\_between\_wolfs(self):  
 minX = 8  
 maxX = 0  
 minY = 8  
 maxY = 0  
 for piece in self.get\_all\_pieces(BLACK):  
 if piece.row > maxY:  
 maxY = piece.row  
 if piece.col > maxX:  
 maxX = piece.col  
 if piece.row < minY:  
 minY = piece.row  
 if piece.col < minX:  
 minX = piece.row  
 return (maxX - minX) \* 2 + (maxY - minY) \* 3 + maxX \* 2  
  
 # check if hare can win  
 def will\_win(self):  
 for move in self.get\_valid\_moves(self.get\_piece(self.harePos[0], self.harePos[1])):  
 if move[0] == 0:  
 return 1000000  
 return 0  
  
 def move(self, piece, row, col):  
 self.board[piece.row][piece.col], self.board[row][col] = self.board[row][col], self.board[piece.row][piece.col]  
 self.board\_debug[piece.row][piece.col], self.board\_debug[row][col] = self.board\_debug[row][col], self.board\_debug[piece.row][piece.col]  
 if piece.color == RED:  
 self.harePos = (row, col)  
 self.prevHarePos = (piece.row, piece.col)  
 piece.move(row, col)  
  
 # check end state  
 def winner(self):  
 piece = self.get\_piece(self.harePos[0], self.harePos[1])  
 if self.harePos[0] == 0:  
 return 'Hare'  
 elif not self.get\_valid\_moves(piece):  
 return 'Wolfs'  
 else:  
 counter = 0  
 for piece in self.get\_all\_pieces(BLACK):  
 if len(self.get\_valid\_moves(piece)) == 0:  
 counter += 1  
 if counter == 4:  
 return 'Hare'  
 return None  
  
 def get\_piece(self, row, col):  
 return self.board[row][col]  
  
 # 0 - none of pieces  
 def create\_board(self):  
 for row in range(ROWS):  
 self.board.append([])  
 for col in range(COLS):  
 if col % 2 == ((row + 1) % 2):  
 if row == 0:  
 self.board[row].append(Piece(row, col, BLACK))  
 elif row == 7 and col == 0:  
 self.board[row].append(Piece(row, col, RED))  
 else:  
 self.board[row].append(0)  
 else:  
 self.board[row].append(0)  
 # print(self.board[row][0].\_\_repr\_\_)  
  
 def \_init\_board\_debug(self):  
 for row in range(ROWS):  
 self.board\_debug.append([])  
 for col in range(COLS):  
 if isinstance(self.board[row][col], Piece):  
 if str(self.board[row][col]) == str(RED):  
 self.board\_debug[row].append("Hare")  
 else:  
 self.board\_debug[row].append("Wolf")  
 else:  
 if row % 2 == 0:  
 if col % 2 == 1:  
 self.board\_debug[row].append("BLACK")  
 else:  
 self.board\_debug[row].append("WHITE")  
 else:  
 if col%2 == 0:  
 self.board\_debug[row].append("BLACK")  
 else:  
 self.board\_debug[row].append("WHITE")  
 return self.board\_debug  
  
 def debug\_print(self):  
 if self.board\_debug == []:  
 self.\_init\_board\_debug()  
 for row in range(ROWS):  
 print(self.board\_debug[row])  
  
 # draw pieces  
 def draw(self, window):  
 self.draw\_board(window)  
 for row in range(ROWS):  
 for col in range(COLS):  
 piece = self.board[row][col]  
 if piece != 0:  
 piece.draw\_piece(window)  
  
 def get\_valid\_moves(self, piece):  
 moves = {}  
 left = piece.col - 1  
 right = piece.col + 1  
 row = piece.row  
  
 if piece.color == RED:  
 moves.update(self.\_traverse\_left(row - 1, max(row - 3, -1), -1, left))  
 moves.update(self.\_traverse\_right(row - 1, max(row - 3, -1), -1, right))  
 moves.update(self.\_traverse\_left(row + 1, min(row + 3, ROWS), 1, left))  
 moves.update(self.\_traverse\_right(row + 1, min(row + 3, ROWS), 1, right))  
  
 if piece.color == BLACK:  
 moves.update(self.\_traverse\_left(row + 1, min(row + 3, ROWS), 1, left))  
 moves.update(self.\_traverse\_right(row + 1, min(row + 3, ROWS), 1, right))  
 return moves  
  
 # diagonal move  
 def \_traverse\_left(self, start, stop, step, left, skipped=[]):  
 moves = {}  
 last = []  
 for r in range(start, stop, step):  
 if left < 0:  
 break  
  
 current = self.board[r][left]  
 if current == 0:  
 if skipped and not last:  
 break  
 else:  
 moves[(r, left)] = last  
 break  
 elif current.color == RED or current.color == BLACK:  
 break  
 else:  
 last = [current]  
  
 left -= 1  
  
 return moves  
  
 def \_traverse\_right(self, start, stop, step, right, skipped=[]):  
 moves = {}  
 last = []  
 for r in range(start, stop, step):  
 if right >= COLS:  
 break  
  
 current = self.board[r][right]  
 if current == 0:  
 if skipped and not last:  
 break  
 else:  
 moves[(r, right)] = last  
 break  
 elif current.color == RED or current.color == BLACK:  
 break  
 else:  
 last = [current]  
  
 right += 1  
  
 return moves  
  
 def \_\_repr\_\_(self):  
 return self.board  
  
  
class Piece:  
 PADDING = 10  
 OUTLINE = 2  
  
 def \_\_init\_\_(self, row, col, color):  
 self.row = row  
 self.col = col  
 self.color = color  
  
 self.x = 0  
 self.y = 0  
 self.pos()  
  
 def pos(self):  
 self.x = SQUARE\_SIZE \* self.col + SQUARE\_SIZE // 2  
 self.y = SQUARE\_SIZE \* self.row + SQUARE\_SIZE // 2  
  
 def move(self, row, col):  
 self.row = row  
 self.col = col  
 self.pos()  
  
 def draw\_piece(self, window):  
 radius = SQUARE\_SIZE//2 - self.PADDING  
 pygame.draw.circle(window, BG, (self.x, self.y), radius + self.OUTLINE)  
 pygame.draw.circle(window, self.color, (self.x, self.y), radius)  
  
 def \_\_repr\_\_(self):  
 return str(self.color)  
  
  
class Game:  
 def \_\_init\_\_(self, window):  
 self.selected = None  
 self.\_init()  
 self.window = window  
  
 # update board and check win state  
 def update(self):  
 self.board.draw(self.window)  
 if self.selected:  
 self.draw\_valid\_moves(self.valid\_moves)  
 if self.board.winner() == 'Hare':  
 pygame.draw.circle(self.window, GOLD, (self.board.harePos[1] \* SQUARE\_SIZE + SQUARE\_SIZE // 2, self.board.harePos[0] \* SQUARE\_SIZE + SQUARE\_SIZE // 2), 50)  
 end\_screen(RED, 'HARE')  
 elif self.board.winner() == 'Wolfs':  
 for piece in self.board.get\_all\_pieces(BLACK):  
 pygame.draw.circle(self.window, GOLD, (piece.col \* SQUARE\_SIZE + SQUARE\_SIZE // 2, piece.row \* SQUARE\_SIZE + SQUARE\_SIZE // 2), 50)  
 end\_screen(BG, 'WOLFS')  
 else:  
 pass  
 pygame.display.update()  
  
 def \_init(self):  
 self.board = Board()  
 self.turn = RED  
 self.valid\_moves = {}  
 self.board.debug\_print()  
  
 # show piece and its valid moves  
 def select(self, row, col):  
 if self.selected:  
 result = self.move(row, col)  
 # invalid move  
 if not result:  
 self.selected = None  
 self.select(row, col)  
  
 piece = self.board.get\_piece(row, col)  
  
 if piece != 0 and piece.color == self.turn:  
 self.selected = piece  
 self.valid\_moves = self.board.get\_valid\_moves(piece)  
 return True  
 return False  
  
 def move(self, row, col):  
 piece = self.board.get\_piece(row, col)  
 if self.selected and piece == 0 and (row, col) in self.valid\_moves:  
 self.board.move(self.selected, row, col)  
 self.board.debug\_print()  
 self.change\_turn()  
 else:  
 return False  
 return True  
  
 # show valid moves  
 def draw\_valid\_moves(self, moves):  
 for move in moves:  
 row, col = move  
 pygame.draw.circle(self.window, BLUE, (col \* SQUARE\_SIZE + SQUARE\_SIZE//2, row \* SQUARE\_SIZE + SQUARE\_SIZE//2), 20)  
  
 def change\_turn(self):  
 self.valid\_moves = []  
 if self.turn == RED:  
 self.turn = BLACK  
 print("Wolfs to move")  
 else:  
 print("Hare to move")  
 self.turn = RED  
  
 def get\_board(self):  
 return self.board  
  
 def ai\_move(self, board):  
 self.board = board  
 self.change\_turn()  
  
  
def end\_screen(colour, label):  
 time.sleep(2)  
 pygame.draw.rect(window, colour, (0, 0, 800, 800))  
 pygame.init()  
 font = pygame.font.Font('freesansbold.ttf', 64)  
 text = font.render('!!! ' + label + ' WON !!!', True, GOLD)  
 window.blit(text, (0, 350))  
 pygame.display.update()  
 time.sleep(2)  
  
  
  
def minimax(position, depth, max\_player):  
 if depth == 0 or position.winner() != None:  
 return position.evaluate(), position  
  
 if max\_player:  
 maxEval = float('-inf')  
 best\_move = None  
 for move in get\_all\_moves(position, BLACK):  
 evaluation = minimax(move, depth - 1, False)[0]  
 maxEval = max(maxEval, evaluation)  
 if maxEval == evaluation:  
 best\_move = move  
  
 return maxEval, best\_move  
 else:  
 minEval = float('inf')  
 best\_move = None  
 for move in get\_all\_moves(position, RED):  
 evaluation = minimax(move, depth - 1, True)[0]  
 minEval = min(minEval, evaluation)  
 if minEval == evaluation:  
 best\_move = move  
  
 return minEval, best\_move  
  
  
def minimax\_hare(position, depth, max\_player):  
 if depth == 0 or position.winner() != None:  
 return position.alternate\_evaluate(), position  
  
 if max\_player:  
 maxEval = float('-inf')  
 best\_move = None  
 for move in get\_all\_moves(position, RED):  
 evaluation = minimax\_hare(move, depth - 1, False)[0]  
 maxEval = max(maxEval, evaluation)  
 if maxEval == evaluation:  
 best\_move = move  
  
 return maxEval, best\_move  
 else:  
 minEval = float('inf')  
 best\_move = None  
 for move in get\_all\_moves(position, BLACK):  
 evaluation = minimax\_hare(move, depth - 1, True)[0]  
 minEval = min(minEval, evaluation)  
 if minEval == evaluation:  
 best\_move = move  
  
 return minEval, best\_move  
  
  
# function helper for minimax  
def simulate\_move(piece, move, board):  
 board.move(piece, move[0], move[1])  
 return board  
  
  
# all possible moves  
def get\_all\_moves(board, color):  
 moves = []  
 for piece in board.get\_all\_pieces(color):  
 valid\_moves = board.get\_valid\_moves(piece)  
 for move in valid\_moves.keys():  
 temp\_board = deepcopy(board)  
 temp\_piece = temp\_board.get\_piece(piece.row, piece.col)  
 new\_board = simulate\_move(temp\_piece, move, temp\_board)  
 moves.append(new\_board)  
  
 return moves  
  
  
# mouse pos on board  
def get\_pos\_mouse(pos):  
 x, y = pos  
 row = y // SQUARE\_SIZE  
 col = x // SQUARE\_SIZE  
 return row, col  
  
  
# game mode menu  
def game\_intro():  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 window.fill(BG)  
  
 title = pygame.image.load("title.png")  
 titleText = window.blit(title, title.get\_rect()) # title is an image  
 titleText.center = ((WIDTH / 2), (HEIGHT / 2))  
  
 wolfsVSai = pygame.image.load("wolfsVSai.png")  
 playerVSplayer = pygame.image.load("playerVSplayer.png")  
 hareVSai = pygame.image.load("hareVsai.png")  
 aiVSai = pygame.image.load("aiVSai.png")  
  
 button(300, 200, 195, 100, playerVSplayer, two\_players)  
 button(300, 320, 195, 100, hareVSai, depth\_selector\_hare\_ai)  
 button(300, 440, 195, 100, wolfsVSai, depth\_selector\_ai\_wolfs)  
 button(300, 560, 195, 100, aiVSai, depth\_selector\_ai\_ai)  
  
 pygame.display.update()  
  
  
# button logic  
def button(x, y, w, h, inactive, action=None):  
 mouse = pygame.mouse.get\_pos()  
 click = pygame.mouse.get\_pressed()  
  
 if x + w > mouse[0] > x and y + h > mouse[1] > y:  
 window.blit(inactive, (x, y))  
 if click[0] == 1 and action is not None:  
 action()  
 else:  
 window.blit(inactive, (x, y))  
  
  
# choose depth of level: player vs ai  
def depth\_selector\_hare\_ai():  
 pygame.init()  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 window.fill(BG)  
  
 depth3 = pygame.image.load("Easy.png")  
 depth5 = pygame.image.load("Medium.png")  
 depth7 = pygame.image.load("Hard.png")  
  
 button(300, 200, 195, 100, depth3, hare\_vs\_ai\_depth3)  
 button(300, 315, 195, 100, depth5, hare\_vs\_ai\_depth5)  
 button(300, 440, 195, 100, depth7, hare\_vs\_ai\_depth7)  
  
 pygame.display.update()  
  
  
def depth\_selector\_ai\_wolfs():  
 pygame.init()  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 window.fill(BG)  
  
 depth3 = pygame.image.load("Easy.png")  
 depth5 = pygame.image.load("Medium.png")  
 depth7 = pygame.image.load("Hard.png")  
  
 button(300, 200, 195, 100, depth3, wolfs\_vs\_ai\_depth3)  
 button(300, 315, 195, 100, depth5, wolfs\_vs\_ai\_depth5)  
 button(300, 440, 195, 100, depth7, wolfs\_vs\_ai\_depth7)  
  
 pygame.display.update()  
  
  
# choose depth of level: ai vs ai  
def depth\_selector\_ai\_ai():  
 pygame.init()  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 window.fill(BG)  
  
 depth3 = pygame.image.load("Easy.png")  
 depth5 = pygame.image.load("Medium.png")  
 depth7 = pygame.image.load("Hard.png")  
  
 button(300, 200, 195, 100, depth3, ai\_vs\_ai\_depth3)  
 button(300, 315, 195, 100, depth5, ai\_vs\_ai\_depth5)  
 button(300, 440, 195, 100, depth7, ai\_vs\_ai\_depth7)  
  
 pygame.display.update()  
  
  
# 2 players  
def two\_players():  
 game = Game(window)  
 pygame.init()  
  
 while True:  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 pos = pygame.mouse.get\_pos()  
 row, col = get\_pos\_mouse(pos)  
 game.select(row, col)  
  
 game.update()  
  
 pygame.quit()  
  
  
# hare vs ai  
def hare\_vs\_ai(depth):  
 running = True  
 game = Game(window)  
 pygame.init()  
  
 while running:  
  
 if game.turn == BLACK:  
 value, new\_board = minimax(game.get\_board(), depth, True)  
 game.ai\_move(new\_board)  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 pos = pygame.mouse.get\_pos()  
 row, col = get\_pos\_mouse(pos)  
 game.select(row, col)  
 game.update()  
 pygame.quit()  
  
  
def hare\_vs\_ai\_depth3():  
 hare\_vs\_ai(3)  
  
  
def hare\_vs\_ai\_depth5():  
 hare\_vs\_ai(5)  
  
  
def hare\_vs\_ai\_depth7():  
 hare\_vs\_ai(7)  
  
  
def wolfs\_vs\_ai(depth):  
 running = True  
 game = Game(window)  
 pygame.init()  
  
 while running:  
  
 if game.turn == RED:  
 value, new\_board = minimax(game.get\_board(), depth, False)  
 game.ai\_move(new\_board)  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
 if event.type == pygame.MOUSEBUTTONDOWN:  
 pos = pygame.mouse.get\_pos()  
 row, col = get\_pos\_mouse(pos)  
 game.select(row, col)  
 game.update()  
 pygame.quit()  
  
  
def wolfs\_vs\_ai\_depth3():  
 wolfs\_vs\_ai(3)  
  
  
def wolfs\_vs\_ai\_depth5():  
 wolfs\_vs\_ai(5)  
  
  
def wolfs\_vs\_ai\_depth7():  
 wolfs\_vs\_ai(7)  
  
  
# ai vs ai  
def ai\_vs\_ai(depth):  
 running = True  
 game = Game(window)  
 pygame.init()  
 while running:  
  
 if game.turn == BLACK:  
 value, new\_board = minimax(game.get\_board(), depth, True)  
 game.ai\_move(new\_board)  
 else:  
 value, new\_board = minimax\_hare(game.get\_board(), depth, True)  
 game.ai\_move(new\_board)  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
 game.update()  
 pygame.quit()  
  
  
def ai\_vs\_ai\_depth3():  
 ai\_vs\_ai(3)  
  
  
def ai\_vs\_ai\_depth5():  
 ai\_vs\_ai(5)  
  
  
def ai\_vs\_ai\_depth7():  
 ai\_vs\_ai(7)  
  
  
# run  
game\_intro()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.



Рисунок 3.1

*При виборі режимів «Hare vs AI», «Wolfs vs AI» або «AI vs AI» є можливість вибрати рівень складності:*

**

Рисунок 3.2

*Процес гри*

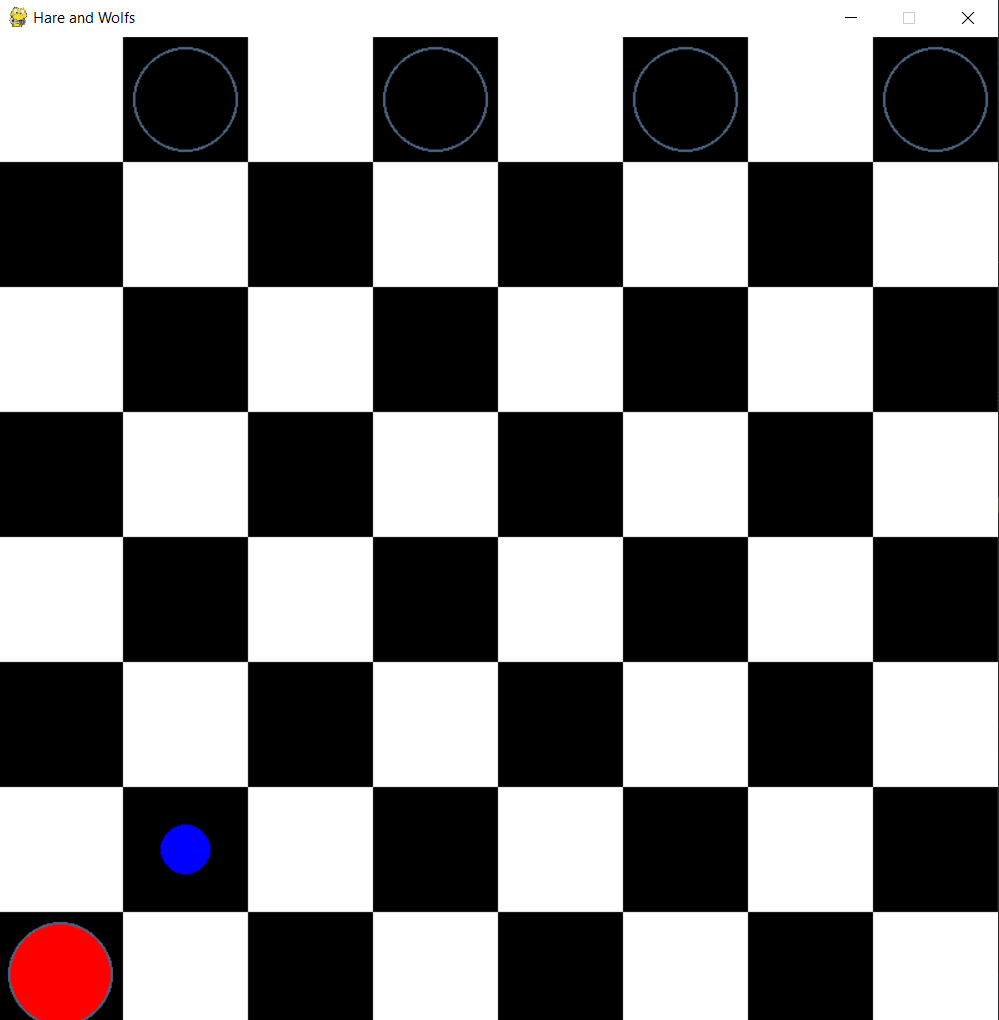


Рисунок 3.3

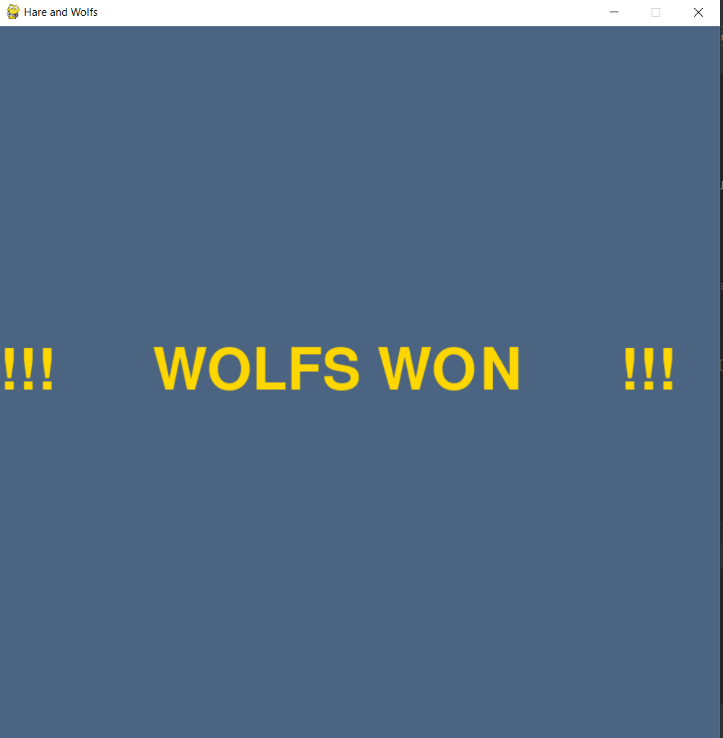
*Кінцеві результати:*

Рисунок 3.4

Висновок

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

Критерії оцінювання

При здачі лабораторної роботи до 10.12.2020 включно максимальний бал дорівнює – 5. Після 10.12.2020 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* програмна реалізація алгоритму – 95%;
* висновок – 5%.

+1 додатковий бал можна отримати за реалізацію рівнів складності.