**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-12 Йолкін Даніїл Сергійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc86770239)

[2 Завдання 4](#_Toc86770240)

[3 Виконання 6](#_Toc86770241)

[3.1 Програмна реалізація алгоритму 6](#_Toc86770242)

[3.1.1 Вихідний код 6](#_Toc86770243)

[3.1.2 Приклади роботи 6](#_Toc86770244)

[Висновок 7](#_Toc86770245)

[Критерії оцінювання 8](#_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

import random  
import copy  
import pygame  
import sys  
from pygame.color import THECOLORS  
  
class Node:  
 def \_\_init\_\_(self, adjacent\_nodes, value, move, player\_name, current\_game):  
 self.value = value  
 self.adjacent\_nodes = list(adjacent\_nodes)  
 self.move = move  
 self.player\_name = player\_name  
 self.current\_game = current\_game  
  
class Algorithm:  
 def \_\_init\_\_(self, game, mode):  
 self.game = game  
 self.mode = mode  
 self.decision\_tree = Node(list(), None, None, 'MAX', copy.deepcopy(game))  
 self.build\_decision\_tree()  
  
 # Function for making decision  
 def make\_decision(self):  
 # For 'easy' mode computer makes fully random decisions  
 if self.mode == 'easy':  
 return self.random\_decision()  
  
 # For 'medium' mode computer makes either random or alpha beta pruning decision with equal chances  
 elif self.mode == 'medium':  
 random\_choice = random.randint(0, 1)  
 if random\_choice:  
 return self.alpha\_beta\_pruning\_decision()  
 else:  
 return self.random\_decision()  
  
 # For 'difficult' mode computer makes fully alpha beta pruning decisions  
 else:  
 return self.alpha\_beta\_pruning\_decision()  
  
 # Function for taking random decision  
 def random\_decision(self):  
 random\_index = random.randint(0, len(self.game.available\_computer\_connections)-1)  
 return self.game.available\_computer\_connections[random\_index]  
  
 # Function for alpha beta pruning  
 def alpha\_beta\_pruning\_decision(self):  
 for node in self.decision\_tree.adjacent\_nodes:  
 if node.value == self.decision\_tree.value:  
 return node.move  
  
 def build\_decision\_tree(self):  
 decision\_tree\_queue = list()  
 decision\_tree\_queue.append(self.decision\_tree)  
 while decision\_tree\_queue:  
 current\_node = decision\_tree\_queue.pop(0)  
 if current\_node.player\_name == 'MAX':  
 for move in current\_node.current\_game.available\_computer\_connections:  
 possible\_game = copy.deepcopy(current\_node.current\_game)  
 possible\_game.move\_order.append(move)  
 possible\_game.delete\_connection(move)  
 if move in possible\_game.available\_computer\_connections:  
 possible\_game.available\_computer\_connections.remove(move)  
 possible\_game.refresh\_connections()  
 possible\_game.is\_Players\_turn = True  
 node = Node(list(), None, move, 'MIN', possible\_game)  
 decision\_tree\_queue.append(node)  
 current\_node.adjacent\_nodes.append(node)  
 else:  
 for move in current\_node.current\_game.available\_player\_connections:  
 possible\_game = copy.deepcopy(current\_node.current\_game)  
 possible\_game.move\_order.append(move)  
 possible\_game.delete\_connection(move)  
 if move in possible\_game.available\_player\_connections:  
 possible\_game.available\_player\_connections.remove(move)  
 possible\_game.refresh\_connections()  
 possible\_game.is\_Players\_turn = False  
 node = Node(list(), None, move, 'MAX', possible\_game)  
 decision\_tree\_queue.append(node)  
 current\_node.adjacent\_nodes.append(node)  
 if current\_node.current\_game.is\_game\_ended():  
 if current\_node.current\_game.is\_Players\_turn:  
 current\_node.value = 1  
 else:  
 current\_node.value = -1  
 self.refresh\_values()  
  
 def refresh\_values(self):  
 decision\_tree\_queue = list()  
 decision\_tree\_queue.append(self.decision\_tree)  
 visited\_nodes = []  
 while decision\_tree\_queue:  
 current\_node = decision\_tree\_queue.pop(0)  
 visited\_nodes.append(current\_node)  
 for node in current\_node.adjacent\_nodes:  
 decision\_tree\_queue.append(node)  
 while visited\_nodes:  
 current\_node = visited\_nodes.pop(len(visited\_nodes)-1)  
 if not current\_node.value:  
 if current\_node.player\_name == 'MAX':  
 current\_min\_node\_value = -2  
 for node in current\_node.adjacent\_nodes:  
 if node.value == 1:  
 current\_min\_node\_value = 1  
 break  
 elif node.value is None:  
 current\_min\_node\_value = None  
 break  
 elif node.value > current\_min\_node\_value:  
 current\_min\_node\_value = node.value  
 current\_node.value = current\_min\_node\_value  
 else:  
 current\_min\_node\_value = 2  
 for node in current\_node.adjacent\_nodes:  
 if node.value == -1:  
 current\_min\_node\_value = -1  
 break  
 elif node.value is None:  
 current\_min\_node\_value = None  
 break  
 elif node.value < current\_min\_node\_value:  
 current\_min\_node\_value = node.value  
 current\_node.value = current\_min\_node\_value  
  
  
class Game:  
 def \_\_init\_\_(self):  
 self.connections\_matrix = Game.generate\_game\_matrix()  
 self.current\_matrix = list(self.connections\_matrix)  
 self.move\_order = []  
 self.is\_Players\_turn = True  
 self.is\_Game\_ended = False  
 self.victory = None  
 self.available\_player\_connections = ['0-2', '2-4', '3-5', '5-7', '3-6']  
 self.available\_computer\_connections = ['1-2', '2-5', '2-3', '4-5', '5-8']  
 self.cleared\_nodes = [0, 1]  
 self.current\_lowest\_node = -1  
  
 # Function to generate game matrix  
 @staticmethod  
 def generate\_game\_matrix():  
 # Default starting combination  
 return [  
 ['x',0,1,0,0,0,0,0,0],  
 [0,'x',2,0,0,0,0,0,0],  
 [1,2,'x',2,1,2,0,0,0],  
 [0,0,2,'x',0,1,1,0,0],  
 [0,0,1,0,'x',2,0,0,0],  
 [0,0,2,1,2,'x',0,1,2],  
 [0,0,0,1,0,0,'x',0,0],  
 [0,0,0,0,0,1,0,'x',0],  
 [0,0,0,0,0,2,0,0,'x']  
 ]  
  
 # Function to determine whether game ended  
 def is\_game\_ended(self):  
 # Check whether there are any connection for one of the players  
 for i in range(0, len(self.current\_matrix[0])):  
 for j in range(0, len(self.current\_matrix[0])):  
 if self.is\_Players\_turn:  
 if self.current\_matrix[i][j] == 1:  
 self.is\_Game\_ended = False  
 return False  
 else:  
 if self.current\_matrix[i][j] == 2:  
 self.is\_Game\_ended = False  
 return False  
  
 # Check if bottom nodes are connected  
 for i in range(0, len(self.current\_matrix)):  
 if(self.current\_matrix[0][i] == 1  
 or self.current\_matrix[0][i] == 2  
 or self.current\_matrix[1][i] == 1  
 or self.current\_matrix[1][i] == 2):  
 self.is\_Game\_ended = False  
 return False  
 # Check if any available moves are  
 if len(self.available\_player\_connections) and len(self.available\_computer\_connections):  
 return False  
 return True  
  
 # Function to delete connection  
 def delete\_connection(self, connection: str):  
 x, y = [int(value) for value in connection.split('-')]  
 self.current\_matrix[x][y] = 0  
 self.current\_matrix[y][x] = 0  
 self.refresh\_connections()  
  
 # Function to refresh the list of cleared nodes  
 def refresh\_cleared\_nodes(self):  
 for i in range(2, len(self.current\_matrix[0])):  
 found = False  
 for j in range(0, len(self.current\_matrix[0])):  
 if self.current\_matrix[i][j] == 1 or self.current\_matrix[i][j] == 2:  
 found = True  
 break  
 if not found and i not in self.cleared\_nodes:  
 self.cleared\_nodes.append(i)  
 for connection in list(self.available\_player\_connections):  
 x, y = [int(value) for value in connection.split('-')]  
 if x in self.cleared\_nodes and y in self.cleared\_nodes:  
 self.available\_player\_connections.remove(connection)  
 for connection in list(self.available\_computer\_connections):  
 x, y = [int(value) for value in connection.split('-')]  
 if x in self.cleared\_nodes and y in self.cleared\_nodes:  
 self.available\_computer\_connections.remove(connection)  
 self.refresh\_cleared\_nodes()  
  
 # Function to clear node  
 def clear\_node(self, node):  
 for i in range(0, len(self.current\_matrix[0])):  
 if node == i:  
 self.current\_matrix[node][i] = 'x'  
 else:  
 self.current\_matrix[node][i] = 0  
 self.current\_matrix[i][node] = 0  
 self.cleared\_nodes.append(node)  
  
 # Function to refresh connections  
 def refresh\_connections(self):  
 self.refresh\_cleared\_nodes()  
 for i in range(2, len(self.current\_matrix[0])):  
 if i not in self.cleared\_nodes:  
 self.current\_lowest\_node = i  
 self.update\_lowest\_connected\_node(i, i, [])  
 lowest\_node = self.current\_lowest\_node  
 # print("Lowest node for " + str(i) + " is " + str(lowest\_node))  
 if lowest\_node == i:  
 self.clear\_node(i)  
 self.refresh\_connections()  
 break  
  
 # Function to find the lowest connected node to the given node  
 def update\_lowest\_connected\_node(self, node, initial\_node, checked\_nodes: list):  
 if node not in checked\_nodes:  
 checked\_nodes.append(node)  
 if self.current\_lowest\_node < initial\_node:  
 return  
 node\_connections = [i for i in range(0, len(self.current\_matrix[0])) if (self.current\_matrix[node][i] != 0 and self.current\_matrix[node][i] != 'x')]  
 if not len(node\_connections) or node == 0 or node == 1:  
 if node < self.current\_lowest\_node:  
 self.current\_lowest\_node = node  
 return  
 for adjacent\_node in node\_connections:  
 self.update\_lowest\_connected\_node(adjacent\_node, initial\_node, checked\_nodes)  
 else:  
 return  
  
 # Function to launch the game  
 def launch\_game(self):  
 while not self.is\_game\_ended():  
 move = None  
 if self.is\_Players\_turn:  
 print("Player's move: ")  
 print(self.available\_player\_connections)  
 while move not in self.available\_player\_connections:  
 move = input()  
 self.move\_order.append(move)  
 self.delete\_connection(move)  
 if move in self.available\_player\_connections:  
 self.available\_player\_connections.remove(move)  
 self.refresh\_connections()  
 print("Move is successful!")  
 print(self.cleared\_nodes)  
 self.is\_Players\_turn = False  
 else:  
 print("Computer's move: ")  
 print(self.available\_computer\_connections)  
 computer = Algorithm(self, 'difficult')  
 move = computer.make\_decision()  
 self.move\_order.append(move)  
 self.delete\_connection(move)  
 if move in self.available\_computer\_connections:  
 self.available\_computer\_connections.remove(move)  
 self.refresh\_connections()  
 print("Move is successful!")  
 print(move)  
 print(self.cleared\_nodes)  
 self.is\_Players\_turn = True  
 if self.is\_Players\_turn:  
 print("Computer won!")  
 else:  
 print("Player won!")  
  
  
def print\_hi(name):  
 # Use a breakpoint in the code line below to debug your script.  
 print(f'Hi, {name}') # Press ⌘F8 to toggle the breakpoint.  
  
  
# Press the green button in the gutter to run the script.  
if \_\_name\_\_ == '\_\_main\_\_':  
 pygame.init()  
 font = pygame.font.Font('freesansbold.ttf', 32)  
 text = font.render('Hackenbush', True, THECOLORS['black'])  
 textRect = text.get\_rect()  
 textRect.center = (600, 25)  
 print('Enter computer difficulty mode(easy, medium, difficult): ')  
 computer\_difficulty\_mode = input()  
 screen = pygame.display.set\_mode((1200, 800))  
 screen.fill(THECOLORS['white'])  
 pygame.display.set\_caption('Hackenbush')  
 game = Game()  
 bottom\_line = pygame.draw.line(screen, THECOLORS['black'], [50, 750], [600, 750], 4)  
 all\_connections = [['5-7', [175, 100], [275, 250], pygame.draw.line(screen, THECOLORS['blue'], [175, 100], [275, 250], 4)],  
 ['5-8', [375, 100], [275, 250], pygame.draw.line(screen, THECOLORS['red'], [375, 100], [275, 250], 4)],  
 ['4-5', [275, 250], [175, 400], pygame.draw.line(screen, THECOLORS['red'], [275, 250], [175, 400], 4)],  
 ['3-5', [275, 250], [375, 400], pygame.draw.line(screen, THECOLORS['blue'], [275, 250], [375, 400], 4)],  
 ['2-4', [175, 400], [275, 600], pygame.draw.line(screen, THECOLORS['blue'], [175, 400], [275, 600], 4)],  
 ['2-5', [275, 600], [275, 250], pygame.draw.line(screen, THECOLORS['red'], [275, 600], [275, 250], 4)],  
 ['2-3', [275, 600], [375, 400], pygame.draw.line(screen, THECOLORS['red'], [275, 600], [375, 400], 4)],  
 ['3-6', [375, 400], [500, 250], pygame.draw.line(screen, THECOLORS['blue'], [375, 400], [500, 250], 4)],  
 ['0-2', [175, 750], [275, 600], pygame.draw.line(screen, THECOLORS['blue'], [175, 750], [275, 600], 4)],  
 ['1-2', [375, 750], [275, 600], pygame.draw.line(screen, THECOLORS['red'], [375, 750], [275, 600], 4)]]  
 all\_nodes = [[[175, 750], pygame.draw.circle(screen, THECOLORS['black'], [175, 750], radius=6, width=0)],  
 [[375, 750], pygame.draw.circle(screen, THECOLORS['black'], [375, 750], radius=6, width=0)],  
 [[275, 600], pygame.draw.circle(screen, THECOLORS['black'], [275, 600], radius=6, width=0)],  
 [[375, 400], pygame.draw.circle(screen, THECOLORS['black'], [375, 400], radius=6, width=0)],  
 [[175, 400], pygame.draw.circle(screen, THECOLORS['black'], [175, 400], radius=6, width=0)],  
 [[275, 250], pygame.draw.circle(screen, THECOLORS['black'], [275, 250], radius=6, width=0)],  
 [[500, 250], pygame.draw.circle(screen, THECOLORS['black'], [500, 250], radius=6, width=0)],  
 [[175, 100], pygame.draw.circle(screen, THECOLORS['black'], [175, 100], radius=6, width=0)],  
 [[375, 100], pygame.draw.circle(screen, THECOLORS['black'], [375, 100], radius=6, width=0)]]

print('Player plays using blue connections, computer - red')

print('Click on connection to remove it.')  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
 screen.blit(text, textRect)  
 if pygame.mouse.get\_pressed()[0] and game.is\_Players\_turn and not game.is\_game\_ended():  
 position = pygame.mouse.get\_pos()  
 for connection in list(all\_connections):  
 if connection[3].collidepoint(position) and connection[0] in game.available\_player\_connections:  
 game.move\_order.append(connection[0])  
 game.delete\_connection(connection[0])  
 game.refresh\_connections()  
 game.is\_Players\_turn = False  
 print("Player's move: " + connection[0])  
 if connection[0] in game.available\_player\_connections:  
 game.available\_player\_connections.remove(connection[0])  
 for move in list(all\_connections):  
 if move[0] not in game.available\_player\_connections and move[0] not in game.available\_computer\_connections:  
 pygame.draw.line(screen, THECOLORS['white'], move[1], move[2], 4)  
 all\_connections.remove(move)  
 if not game.is\_Players\_turn and len(game.available\_computer\_connections) > 0:  
 pygame.time.wait(500)  
 computer = Algorithm(game, computer\_difficulty\_mode)  
 move = computer.make\_decision()  
 game.move\_order.append(move)  
 game.delete\_connection(move)  
 game.refresh\_connections()  
 game.is\_Players\_turn = True  
 print("Computer's move: " + move)  
 if move in game.available\_computer\_connections:  
 game.available\_computer\_connections.remove(move)  
 for move in list(all\_connections):  
 if move[0] not in game.available\_player\_connections and move[0] not in game.available\_computer\_connections:  
 pygame.draw.line(screen, THECOLORS['white'], move[1], move[2], 4)  
 all\_connections.remove(move)  
 for node in list(all\_nodes):  
 if all\_nodes.index(node) in game.cleared\_nodes and all\_nodes.index(node) > 1:  
 pygame.draw.circle(screen, THECOLORS['white'], node[0], radius=6, width=0)  
 else:  
 pygame.draw.circle(screen, THECOLORS['black'], node[0], radius=6, width=0)  
 if game.is\_game\_ended():  
 for node in list(all\_nodes):  
 if all\_nodes.index(node) in game.cleared\_nodes:  
 pygame.draw.circle(screen, THECOLORS['white'], node[0], radius=6, width=0)  
 else:  
 pygame.draw.circle(screen, THECOLORS['black'], node[0], radius=6, width=0)  
 bottom\_line = pygame.draw.line(screen, THECOLORS['black'], [50, 750], [600, 750], 4)  
 if game.is\_Players\_turn:  
 text\_2 = font.render('Computer won!', True, THECOLORS['black'])  
 textRect\_2 = text\_2.get\_rect()  
 textRect\_2.center = (1000, 25)  
 print("Computer won!")  
 else:  
 text\_2 = font.render('Player won!', True, THECOLORS['black'])  
 textRect\_2 = text\_2.get\_rect()  
 textRect\_2.center = (1000, 25)  
 print("Player won!")  
 screen.blit(text\_2, textRect\_2)  
 elif game.is\_game\_ended():  
 pygame.display.flip()  
 pygame.display.update()  
 pygame.display.flip()  
 pygame.display.update()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

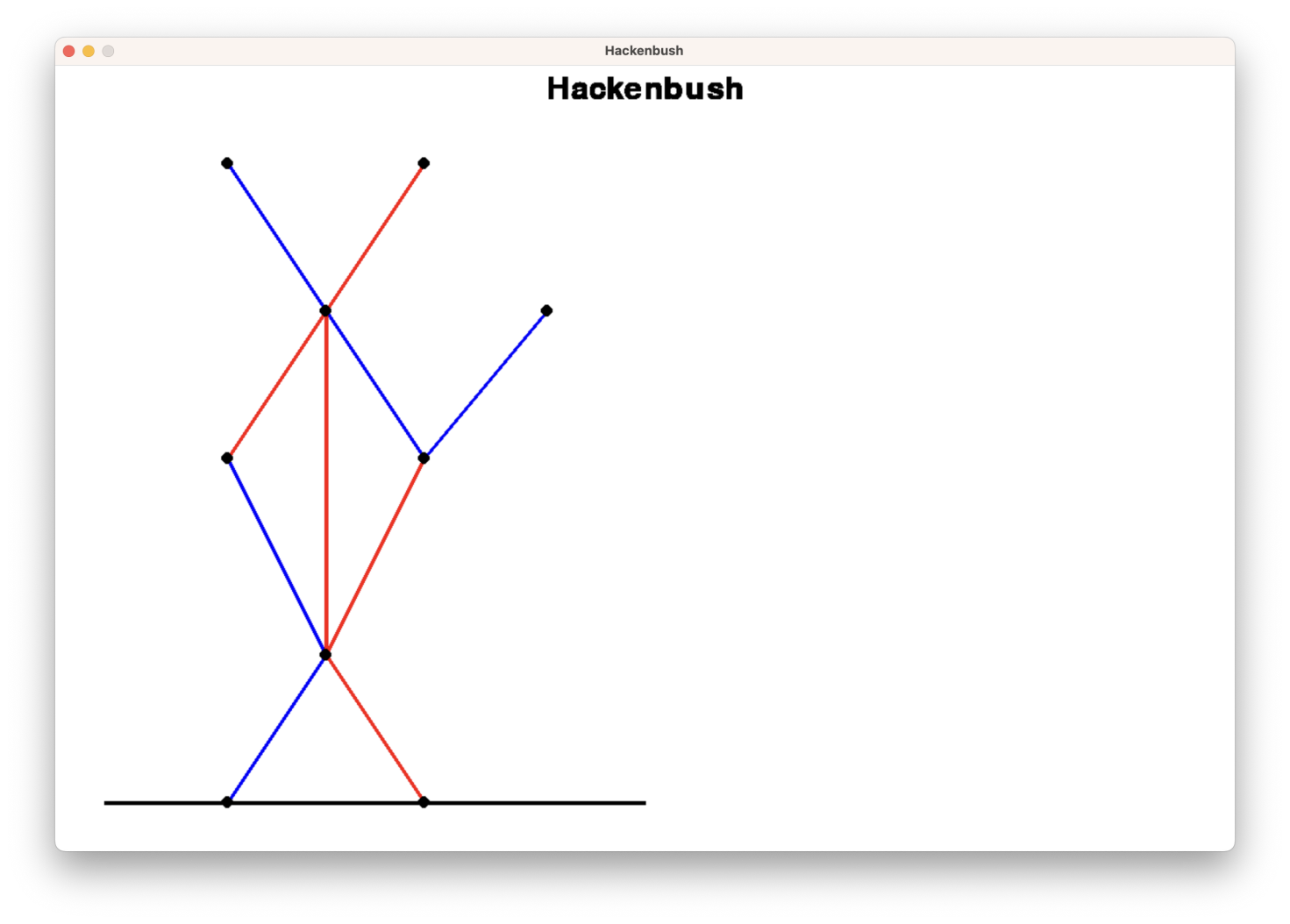


Рисунок 3.1 – Початкове положення гри

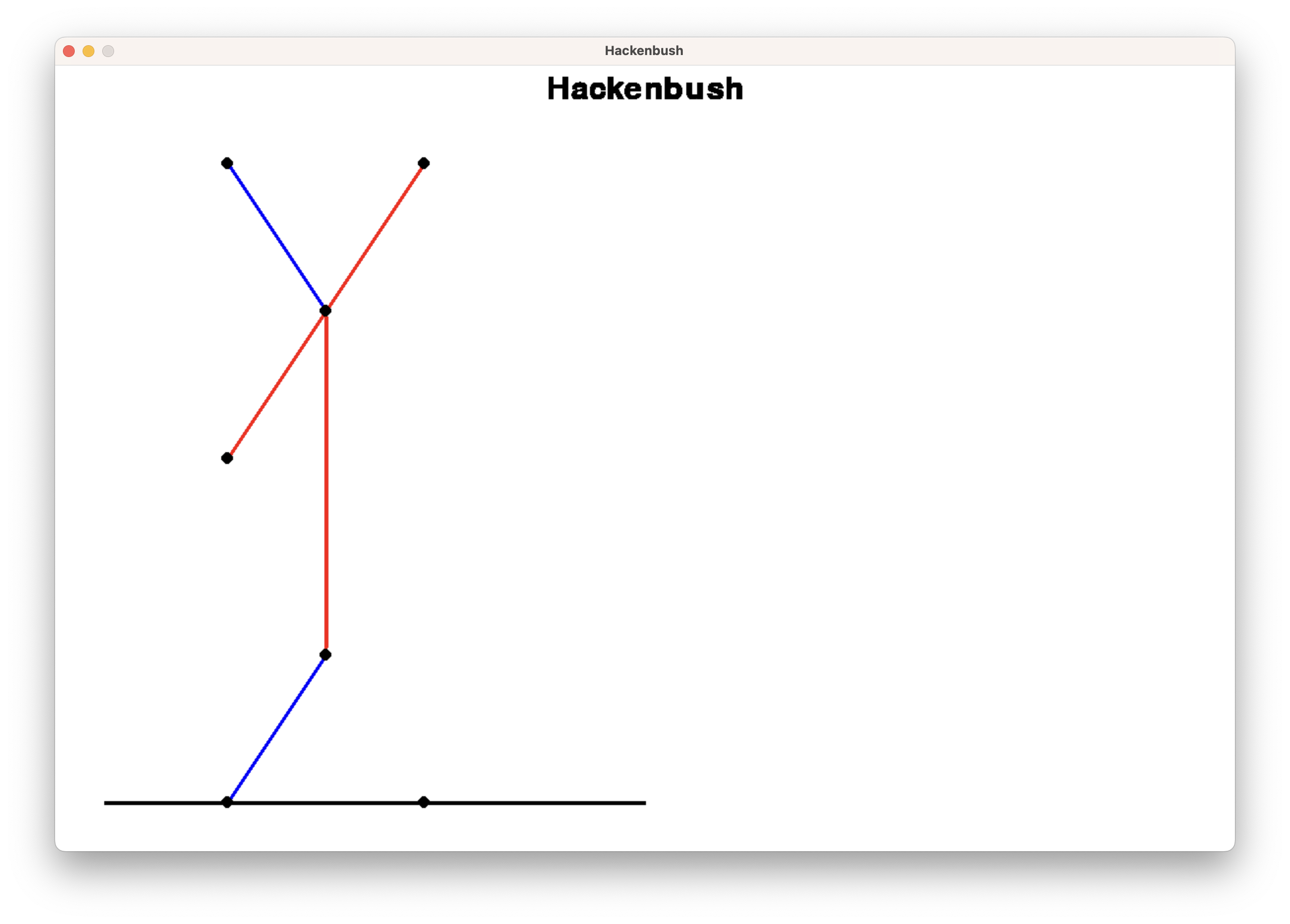


Рисунок 3.2 – Гра після декількох ходів

Висновок

В рамках даної лабораторної роботи було розроблено гру та штучний інтелект для гри в Hackenbush. Штучний інтелект має 3 складності та реалізований за допомогою алгоритма Alpha Beta Pruning.

На початку гри в консолі зображаються підказки та вибирається рівень складності штучного інтелекту.

Також було розроблено графічне меню користувача для представлення гри. Гравець повинен видаляти сині зв’язки, а комп’ютер червоні. Для видалення зв’язку гравець повинен натиснути на зв’язок. По закінченню гри, на екрані з’явиться, хто переміг.

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).