**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-14 Радзівіло Валерія Артемівна*

**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

Main.java

import boad\_view.\*;  
  
  
public class Main {  
 public static void main(String[] args){  
  
 board board = new board();  
 board.setup\_board(0);  
 }  
  
}

board.java

package boad\_view;  
  
import ArrWork.Figure;  
import ArrWork.Node;  
import ArrWork.Tree;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
import java.util.ArrayList;  
  
import static javax.swing.WindowConstants.*EXIT\_ON\_CLOSE*;  
  
  
public class board {  
  
 Tree tree;  
 JFrame game\_frame;  
  
 int computerWon= 0; // 0 - game in process 1 - comp won 2 - user won  
  
 public void setTree(Tree tree) {  
 this.tree = tree;  
 }  
  
 public void setGame\_frame(JFrame game\_frame) {  
 this.game\_frame = game\_frame;  
 }  
 // 0-easy, 1-medium, 2 - hard  
 public void setup\_board(int level\_option){  
  
  
 int rabbit\_start\_x = 6;  
 int rabbit\_start\_y =7;  
 JButton easy\_level=new JButton("Easy");  
 JButton medium\_level=new JButton("Medium");  
 JButton hard\_level=new JButton("Hard");  
  
  
  
  
 ArrayList<Figure> rabbits\_options = new ArrayList<>();  
 rabbits\_options.add(new Figure(rabbit\_start\_x,rabbit\_start\_y,true));  
  
 Tree tree\_m = new Tree(new Node(rabbits\_options,0));  
  
 ArrayList<ArrayList<Figure>> prev\_wolves = new ArrayList<>();  
 ArrayList<Figure> start\_wolves = new ArrayList<>();  
 for(int l =0; l<4;l++)  
 {  
 start\_wolves.add(new Figure(l\*2+1,0,false));  
 }  
 prev\_wolves.add(start\_wolves);  
 tree\_m.getRoot().addChild(new Node(start\_wolves,1));  
 tree\_m.make\_tree(tree\_m.getRoot().getChildren().get(0),prev\_wolves,rabbits\_options,1);  
 setTree(tree\_m);  
  
  
 JFrame f=new JFrame("Chess Game");  
 JPanel pane = new JPanel(new GridBagLayout()); //Create a pane to house all content, and give it a GridBagLayout  
 f.setContentPane(pane);  
  
 GridBagConstraints c = new GridBagConstraints();  
 c.fill = GridBagConstraints.*BOTH*;  
 c.insets.right = 0;  
 c.insets.left = 0;  
 c.weightx = 1;  
 c.weighty = 1;  
  
 JButton rabbit\_button = rabbit\_but();  
 c.gridx = rabbit\_start\_x;  
 c.gridwidth = 1;  
 c.gridy = rabbit\_start\_y;  
 pane.add(rabbit\_button, c);  
  
  
 ArrayList<ArrayList<Integer>> rabbit\_pos = new ArrayList<>();  
 ArrayList<Integer> cur\_rabbit\_pos = new ArrayList<>();  
 cur\_rabbit\_pos.add(rabbit\_start\_x);  
 cur\_rabbit\_pos.add(rabbit\_start\_y);  
 rabbit\_pos.add(cur\_rabbit\_pos);  
 ArrayList<Figure> rabbit\_movements = new ArrayList<>();  
 rabbit\_movements.add(new Figure(rabbit\_start\_x,rabbit\_start\_y,true));  
  
  
 ArrayList<JButton> wolves = add\_wolves();  
 int iter =0;  
 for(JButton w: wolves) {  
 w.setBorderPainted( false );  
 w.setFocusPainted( false );  
 c.gridx = iter\*2+1;  
 c.gridwidth = 1;  
 c.gridy = 0;  
 pane.add(w,c);  
 iter++;  
 }  
  
 easy\_level.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 f.dispose();  
 setup\_board(0);  
  
 }  
 });  
 medium\_level.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 f.dispose();  
 setup\_board(1);  
 }  
 });  
 hard\_level.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 f.dispose();  
 setup\_board(2);  
 }  
 });  
  
 ArrayList<ArrayList<JButton>> board\_buttons = make\_board(rabbit\_button,pane, c,rabbit\_pos,rabbit\_movements,wolves, level\_option);  
  
  
  
 int i=0;  
 int j =0;  
 for(ArrayList<JButton> b:board\_buttons) {  
 j=0;  
 for (JButton but : b ){  
 c.gridx=i;  
 c.gridwidth = 1;  
 c.gridy=j;  
 pane.add(but,c);  
 j++;  
 }  
  
 i++;  
 }  
  
  
 c.gridx = 8;  
 c.gridy = 0;  
 pane.add(easy\_level,c);  
 c.gridy+=1;  
 pane.add(medium\_level,c);  
 c.gridy+=1;  
 pane.add(hard\_level,c);  
  
 f.setDefaultCloseOperation(*EXIT\_ON\_CLOSE* );  
 f.pack();  
 f.setLocationRelativeTo( null );  
 f.setVisible(true);  
 f.setSize(900,700);  
  
 f.setResizable(false);  
 setGame\_frame(f);  
  
 }  
  
  
 ArrayList<JButton> add\_wolves()  
 {  
 ArrayList<JButton> wolves = new ArrayList<>();  
 for(int i =0; i<4;i++) {  
 Icon icon = new ImageIcon("C:\\Users\\leraz\\IdeaProjects\\algorithms\_lab6\\src\\wolf.png");  
 Image img = ((ImageIcon) icon).getImage();  
 Image newimg = img.getScaledInstance( 40, 40, java.awt.Image.*SCALE\_SMOOTH* ) ;  
 icon = new ImageIcon( newimg );  
 JButton button7 = new JButton(icon);  
 button7.setBackground(Color.*red*);  
 button7.setPreferredSize(new Dimension(40, 40));  
 wolves.add(button7);  
 }  
 return wolves;  
 }  
  
 JButton rabbit\_but()  
 {  
 Icon icon = new ImageIcon("C:\\Users\\leraz\\IdeaProjects\\algorithms\_lab6\\src\\rabbit.png");  
 Image img = ((ImageIcon) icon).getImage() ;  
 Image newimg = img.getScaledInstance( 40, 40, java.awt.Image.*SCALE\_SMOOTH* ) ;  
 icon = new ImageIcon( newimg );  
 JButton button7 = new JButton(icon);  
 button7.setBackground(Color.*green*);  
 button7.setPreferredSize(new Dimension(40, 40));  
  
 return button7;  
 }  
  
 ArrayList<ArrayList<JButton>> make\_board(JButton rabbit, JPanel panel, GridBagConstraints c, ArrayList<ArrayList<Integer>>rabbit\_pos, ArrayList<Figure>rabbit\_movement  
 ,ArrayList<JButton>wolves\_buttons, int level\_option)  
 {  
  
 ArrayList<ArrayList<JButton>>board\_buttons = new ArrayList<>();  
 for(int i =0; i<8;i++) {  
 ArrayList<JButton> butts = new ArrayList<>();  
 for(int j =0;j<8;j++) {  
  
 JButton b = new JButton();  
 b.setPreferredSize(new Dimension(40, 40));  
 int finalJ = j;  
 int finalI = i;  
 if((finalJ+finalI) % 2 ==0)  
 b.setBackground(Color.*white*);  
 else  
 b.setBackground(Color.*black*);  
 b.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 System.*out*.println("Button clicked " + " X: " + finalI + " Y:" + finalJ );  
 int checker = move\_rabbit(rabbit, finalI, finalJ,panel, c, board\_buttons,rabbit\_pos,rabbit\_movement);  
 ArrayList<Figure> last\_wolves = new ArrayList<>();  
 if(checker ==1) {  
 if(level\_option==0) last\_wolves = new ArrayList<>(move\_wolves\_easy(rabbit\_movement, wolves\_buttons, panel, c, board\_buttons));  
 if(level\_option == 1) last\_wolves = new ArrayList<>(move\_wolves\_medium(rabbit\_movement, wolves\_buttons, panel, c, board\_buttons));  
 if(level\_option == 2) last\_wolves = new ArrayList<>(move\_wolves\_hard(rabbit\_movement, wolves\_buttons, panel, c, board\_buttons));  
 check\_if\_comp\_won(rabbit\_movement.get(rabbit\_movement.size()-1).getX(),rabbit\_movement.get(rabbit\_movement.size()-1).getY(),last\_wolves);  
  
 }  
  
  
 }  
 });  
 butts.add(b);  
 }  
 board\_buttons.add(butts);  
 }  
 return board\_buttons;  
 }  
  
 void paint\_board(ArrayList<ArrayList<JButton>> board)  
 {  
 for(int i =0; i<8;i++)  
 {  
 for(int j =0;j <8;j++)  
 {  
 JButton b = board.get(i).get(j);  
 if((j+i) % 2 ==0)  
 b.setBackground(Color.*white*);  
 else  
 b.setBackground(Color.*black*);  
 }  
 }  
 }  
  
 int move\_rabbit(JButton rabbit, int move\_x, int move\_y, JPanel panel, GridBagConstraints c, ArrayList<ArrayList<JButton>> board\_buttons, ArrayList<ArrayList<Integer>>rabbit\_pos,  
 ArrayList<Figure>rabbit\_movements)  
 {  
  
 int rabbit\_x = rabbit\_pos.get(rabbit\_pos.size()-1).get(0);  
 int rabbit\_y = rabbit\_pos.get(rabbit\_pos.size()-1).get(1);  
 System.*out*.println("Rabbit: "+rabbit\_x+ " "+rabbit\_y);  
 System.*out*.println("Where: "+move\_x+ " "+move\_y);  
  
 if((move\_x == rabbit\_x-1 && move\_y == rabbit\_y-1) ||(move\_x == rabbit\_x+1 && move\_y == rabbit\_y+1)  
 || (move\_x == rabbit\_x-1 && move\_y == rabbit\_y+1)||(move\_x == rabbit\_x+1 && move\_y == rabbit\_y-1))  
 {  
 System.*out*.println("Moving");  
  
 panel.remove(rabbit);  
 panel.remove(board\_buttons.get(move\_x).get(move\_y));  
  
 c.gridx = move\_x;  
 c.gridwidth = 1;  
 c.gridy = move\_y;  
 panel.repaint();  
 panel.revalidate();  
 panel.add(rabbit,c);  
 rabbit\_movements.add(new Figure(move\_x,move\_y,true));  
  
 if(rabbit\_pos.size()>1) {  
 c.gridx = rabbit\_x;  
 c.gridwidth = 1;  
 c.gridy = rabbit\_y;  
 JButton old\_place = board\_buttons.get(rabbit\_x).get(rabbit\_y);  
 if(rabbit\_x+rabbit\_y%2==0)  
 old\_place.setBackground(Color.*white*);  
 else  
 old\_place.setBackground(Color.*black*);  
 panel.add(old\_place, c);  
 System.*out*.println("Added board button to "+ rabbit\_x+" "+rabbit\_y);  
 }  
  
  
 ArrayList<Integer> new\_position = new ArrayList<>();  
 new\_position.add(move\_x);  
 new\_position.add(move\_y);  
 rabbit\_pos.add(new\_position);  
 System.*out*.println("Rabbit moved");  
 if(move\_y == 0) {  
 JOptionPane.*showMessageDialog*(game\_frame, "You won!");  
 computerWon = 2;  
 game\_frame.dispose();  
 }  
 return 1;  
  
 }  
 return 0;  
  
 }  
  
 void setupBoardButtons(ArrayList<ArrayList<JButton>> board\_buttons, JPanel panel, GridBagConstraints c)  
 {  
 for(int i =0; i<8;i++)  
 {  
 for(int j =0; j<8;j++)  
 {  
 c.gridx=i;  
 c.gridy=j;  
 panel.add(board\_buttons.get(i).get(j),c);  
 }  
 }  
 }  
  
 ArrayList<Figure> move\_wolves\_easy( ArrayList<Figure>rabbit\_movements, ArrayList<JButton> wolf\_buttons, JPanel panel, GridBagConstraints c, ArrayList<ArrayList<JButton>> board\_button)  
 {  
 ArrayList<Figure> move\_w = tree.possible\_options\_for\_wolves(rabbit\_movements.size()\*2-1,rabbit\_movements.get(rabbit\_movements.size()-1),tree.getRoot(),rabbit\_movements,1);  
  
 System.*out*.println(move\_w.size());  
 for(Figure f: move\_w)  
 System.*out*.println("W: " +f.getX()+" "+ f.getY());  
 for(JButton w: wolf\_buttons)  
 {  
 panel.remove(w);  
 }  
 for(ArrayList<JButton> board: board\_button)  
 {  
 for(JButton b: board)  
 panel.remove(b);  
 }  
 setupBoardButtons(board\_button, panel,c);  
 System.*out*.println("Wolves deleted");  
 panel.repaint();  
 panel.revalidate();  
 for(int i =0; i<move\_w.size();i++)  
 {  
 c.gridx = move\_w.get(i).getX();  
 c.gridy = move\_w.get(i).getY();  
  
 panel.remove(board\_button.get(move\_w.get(i).getX()).get(move\_w.get(i).getY()));  
 panel.repaint();  
 panel.revalidate();  
 panel.add(wolf\_buttons.get(i), c);  
  
 }  
 System.*out*.println("Wolves moved");  
 if(rabbit\_movements.size()==3)  
 {  
 rabbit\_movements.remove(0);  
 rabbit\_movements.remove(0);  
 Figure new\_rabbit = new Figure(rabbit\_movements.get(0).getX(),rabbit\_movements.get(0).getY(),true);  
 ArrayList<Figure> rab = new ArrayList<>();  
 rab.add(new\_rabbit);  
  
 Tree tree\_m = new Tree(new Node(rab,0));  
 tree\_m.getRoot().addChild(new Node(move\_w,1));  
 ArrayList<ArrayList<Figure>> previous\_wolf = new ArrayList<>();  
 previous\_wolf.add(move\_w);  
 tree\_m.make\_tree(tree\_m.getRoot().getChildren().get(0),previous\_wolf,rab,1);  
 setTree(tree\_m);  
 }  
  
 return move\_w;  
 }  
  
 ArrayList<Figure> move\_wolves\_medium( ArrayList<Figure>rabbit\_movements, ArrayList<JButton> wolf\_buttons, JPanel panel, GridBagConstraints c, ArrayList<ArrayList<JButton>> board\_button)  
 {  
 ArrayList<Figure> move\_w = tree.alpha\_beta\_pruning(rabbit\_movements.size()\*2-1,rabbit\_movements.get(rabbit\_movements.size()-1),tree.getRoot(),rabbit\_movements,1);  
 ArrayList<Figure> move\_w\_plus\_rab = new ArrayList<>(move\_w);  
 move\_w\_plus\_rab.add(rabbit\_movements.get(rabbit\_movements.size()-1));  
 while(!Figure.*check\_for\_duplicates*(move\_w\_plus\_rab,null))  
 {  
 move\_w = tree.alpha\_beta\_pruning(rabbit\_movements.size()\*2-1,rabbit\_movements.get(rabbit\_movements.size()-1),tree.getRoot(),rabbit\_movements,1);  
 move\_w\_plus\_rab = new ArrayList<>(move\_w);  
 move\_w\_plus\_rab.add(rabbit\_movements.get(rabbit\_movements.size()-1));  
 }  
  
 System.*out*.println(move\_w.size());  
 for(Figure f: move\_w)  
 System.*out*.println("W: " +f.getX()+" "+ f.getY());  
 for(JButton w: wolf\_buttons)  
 {  
 panel.remove(w);  
 }  
 for(ArrayList<JButton> board: board\_button)  
 {  
 for(JButton b: board)  
 panel.remove(b);  
 }  
 setupBoardButtons(board\_button, panel,c);  
 System.*out*.println("Wolves deleted");  
 panel.repaint();  
 panel.revalidate();  
 for(int i =0; i<move\_w.size();i++)  
 {  
 c.gridx = move\_w.get(i).getX();  
 c.gridy = move\_w.get(i).getY();  
  
 panel.remove(board\_button.get(move\_w.get(i).getX()).get(move\_w.get(i).getY()));  
 panel.repaint();  
 panel.revalidate();  
 panel.add(wolf\_buttons.get(i), c);  
  
 }  
 System.*out*.println("Wolves moved");  
 if(rabbit\_movements.size()==3)  
 {  
 rabbit\_movements.remove(0);  
 rabbit\_movements.remove(0);  
 Figure new\_rabbit = new Figure(rabbit\_movements.get(0).getX(),rabbit\_movements.get(0).getY(),true);  
 ArrayList<Figure> rab = new ArrayList<>();  
 rab.add(new\_rabbit);  
  
 Tree tree\_m = new Tree(new Node(rab,0));  
 tree\_m.getRoot().addChild(new Node(move\_w,1));  
 ArrayList<ArrayList<Figure>> previous\_wolf = new ArrayList<>();  
 previous\_wolf.add(move\_w);  
 tree\_m.make\_tree(tree\_m.getRoot().getChildren().get(0),previous\_wolf,rab,1);  
 setTree(tree\_m);  
 }  
 return move\_w;  
  
  
 }  
  
 void check\_if\_comp\_won(int rabbit\_x, int rabbit\_y, ArrayList<Figure> wolves)  
 {  
 // check if all are surrounding  
 int checker\_1 = 0;  
 for(Figure f:wolves)  
 {  
 if((f.getX()==rabbit\_x+1 &&f.getY()==rabbit\_y+1) ||(f.getX()==rabbit\_x-1 &&f.getY()==rabbit\_y-1)||(f.getX()==rabbit\_x+1 &&f.getY()==rabbit\_y-1)||(f.getX()==rabbit\_x-1 &&f.getY()==rabbit\_y+1))  
 checker\_1++;  
 }  
 if(checker\_1==4)  
 {  
 computerWon=1;  
 JOptionPane.*showMessageDialog*(game\_frame, "Computer won");  
 game\_frame.dispose();  
 }  
 else{  
 // check if rabbit has no moves and not in 0  
 int checker = 0;  
 for(Figure w: wolves)  
 {  
 if((rabbit\_x+1== w.getX() && rabbit\_y+1==w.getY())||(rabbit\_x-1 == w.getX() && rabbit\_y-1 == w.getY())||(rabbit\_x-1==w.getX()&&rabbit\_y+1==w.getY())||(rabbit\_x+1==w.getX()&&rabbit\_y-1==w.getY()))  
 checker++;  
 }  
 if(checker>=3)  
 {  
 computerWon=1;  
 JOptionPane.*showMessageDialog*(game\_frame, "Computer won");  
 game\_frame.dispose();  
 }  
 }  
  
 }  
  
 ArrayList<Figure> move\_wolves\_hard( ArrayList<Figure>rabbit\_movements, ArrayList<JButton> wolf\_buttons, JPanel panel, GridBagConstraints c, ArrayList<ArrayList<JButton>> board\_button)  
 {  
 ArrayList<Figure> move\_w = tree.alpha\_beta\_pruning(rabbit\_movements.size()\*2-1,rabbit\_movements.get(rabbit\_movements.size()-1),tree.getRoot(),rabbit\_movements,1);  
  
 System.*out*.println(move\_w.size());  
 for(Figure f: move\_w)  
 System.*out*.println("W: " +f.getX()+" "+ f.getY());  
 for(JButton w: wolf\_buttons)  
 {  
 panel.remove(w);  
 }  
 for(ArrayList<JButton> board: board\_button)  
 {  
 for(JButton b: board)  
 panel.remove(b);  
 }  
 setupBoardButtons(board\_button, panel,c);  
 System.*out*.println("Wolves deleted");  
 panel.repaint();  
 panel.revalidate();  
 for(int i =0; i<move\_w.size();i++)  
 {  
 c.gridx = move\_w.get(i).getX();  
 c.gridy = move\_w.get(i).getY();  
  
 panel.remove(board\_button.get(move\_w.get(i).getX()).get(move\_w.get(i).getY()));  
 panel.repaint();  
 panel.revalidate();  
 panel.add(wolf\_buttons.get(i), c);  
  
 }  
 System.*out*.println("Wolves moved");  
 if(rabbit\_movements.size()==3)  
 {  
 rabbit\_movements.remove(0);  
 rabbit\_movements.remove(0);  
 Figure new\_rabbit = new Figure(rabbit\_movements.get(0).getX(),rabbit\_movements.get(0).getY(),true);  
 ArrayList<Figure> rab = new ArrayList<>();  
 rab.add(new\_rabbit);  
  
 Tree tree\_m = new Tree(new Node(rab,0));  
 tree\_m.getRoot().addChild(new Node(move\_w,1));  
 ArrayList<ArrayList<Figure>> previous\_wolf = new ArrayList<>();  
 previous\_wolf.add(move\_w);  
 tree\_m.make\_tree(tree\_m.getRoot().getChildren().get(0),previous\_wolf,rab,1);  
 setTree(tree\_m);  
 }  
  
 return move\_w;  
 }  
}

Node.java

package ArrWork;  
  
import java.util.ArrayList;  
  
public class Node {  
 private final ArrayList<Figure> figure;  
 private final int depth;  
 private final ArrayList<Node> children ;  
 int value;  
  
 public Node(ArrayList<Figure> figure, int depth){  
 this.figure = figure;  
 this.children = new ArrayList<>();  
 this.depth = depth;  
  
 }  
  
  
  
 public int getValue ()  
 {  
 return this.value;  
 }  
  
 public void setValue(int value) {  
 this.value = value;  
 }  
  
 public ArrayList<Figure> get\_figure(){  
 return figure;  
 }  
  
  
  
 public void addChild(Node child){  
 children.add(child);  
 }  
 public ArrayList<Node> getChildren() {  
 return children;  
 }  
  
 public int getDepth() {  
 return depth;  
 }  
}

Figure.java

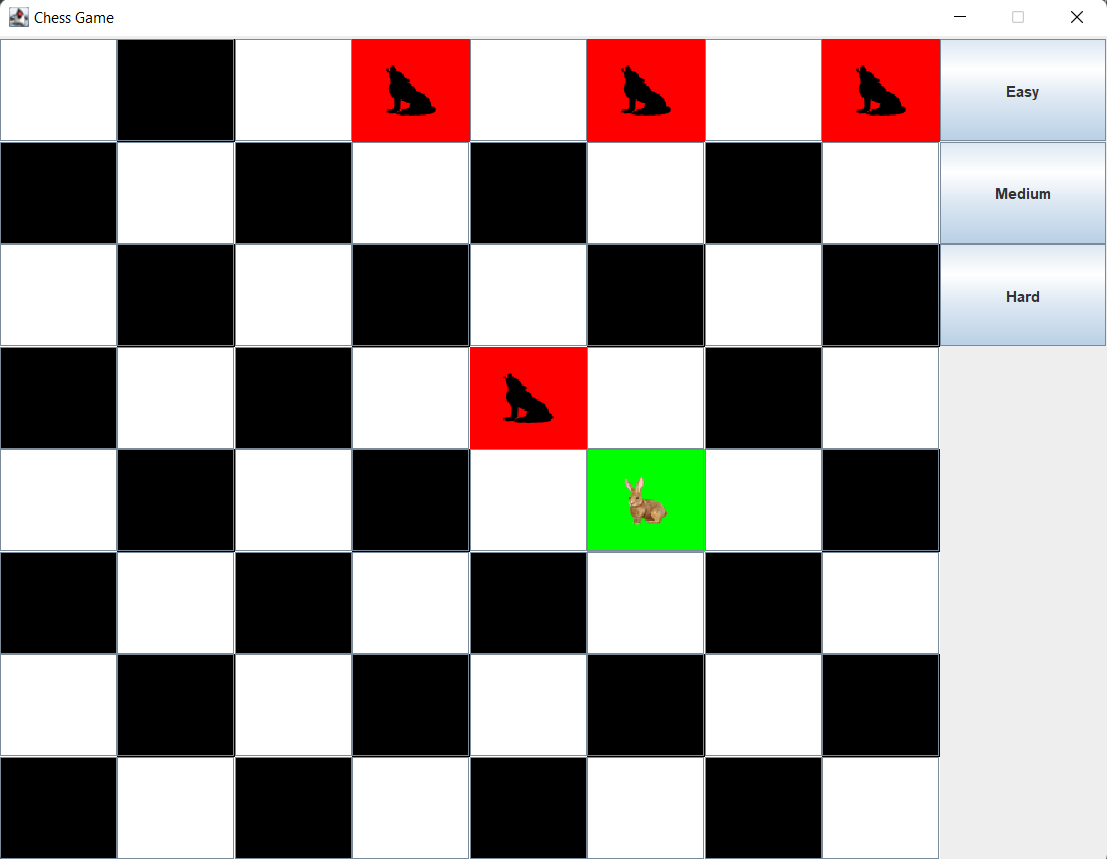
package ArrWork;  
  
import java.util.ArrayList;  
  
public class Figure {  
 int x;  
 int y;  
 boolean isRabbit;  
  
  
 public Figure(int x, int y, boolean isRabbit)  
 {  
 this.x = x;  
 this.y = y;  
 this.isRabbit = isRabbit;  
 }  
  
 public int getX()  
 {  
 return x;  
 }  
  
 public int getY()  
 {  
 return y;  
 }  
  
 public ArrayList<Integer> getCoords()  
 {  
 ArrayList<Integer> x\_y = new ArrayList<>();  
 x\_y.add(x);  
 x\_y.add(y);  
 return x\_y;  
 }  
  
 public static boolean check\_for\_duplicates(ArrayList<Figure> wolves, Figure needed\_to\_add)  
 {  
 for(int i =0;i<wolves.size();i++)  
 {  
 for(int j =0; j< wolves.size();j++)  
 {  
 if(i!=j && wolves.get(i).getX()==wolves.get(j).getX()&& wolves.get(i).getY()==wolves.get(j).getY())  
 return false;  
 }  
 }  
  
 return true;  
 }  
}

Tree.java

package ArrWork;  
import java.util.ArrayList;  
import java.util.Random;  
  
public class Tree{  
 private final Node root;  
 public Node getRoot(){  
 return root;  
 }  
 public Tree(Node root){  
 this.root = root;  
 }  
  
 public void make\_tree(Node current\_node, ArrayList<ArrayList<Figure>> previous\_wolves, ArrayList<Figure> prev\_rabbits, int depth)  
 {  
  
 if(depth<4) {  
  
 depth += 1;  
 for (Figure prev\_rabbit : prev\_rabbits) {  
 ArrayList<Figure> new\_rabbits = new ArrayList<>();  
 int rabbit\_x = prev\_rabbit.x;  
 int rabbit\_y = prev\_rabbit.y;  
 if (rabbit\_x + 1 < 8 && rabbit\_y + 1 < 8)  
 new\_rabbits.add(new Figure(rabbit\_x + 1, rabbit\_y + 1, true));  
 if (rabbit\_x - 1 >= 0 && rabbit\_y - 1 >= 0)  
 new\_rabbits.add(new Figure(rabbit\_x - 1, rabbit\_y - 1, true));  
 if (rabbit\_x - 1 >= 0 && rabbit\_y + 1 < 8)  
 new\_rabbits.add(new Figure(rabbit\_x - 1, rabbit\_y + 1, true));  
 if (rabbit\_x + 1 < 8 && rabbit\_y - 1 >= 0)  
 new\_rabbits.add(new Figure(rabbit\_x + 1, rabbit\_y - 1, true));  
  
 for(Figure f: new\_rabbits) {  
  
 ArrayList<Figure> f\_list = new ArrayList<>();  
 f\_list.add(f);  
 Node children\_node = new Node(f\_list, depth);  
 current\_node.addChild(children\_node);  
  
  
 for (ArrayList<Figure> old\_wolves : previous\_wolves) {  
 ArrayList<ArrayList<Figure>> new\_wolves = new ArrayList<>();  
 for(int l =0; l< old\_wolves.size();l++)  
 {  
 if(old\_wolves.get(l).getX()+1<8 && old\_wolves.get(l).getY()+1<8 && old\_wolves.get(l).getX()!=f.getX()  
 &&old\_wolves.get(l).getY()!=f.getY()  
 ) {  
 ArrayList<Figure> new\_old\_wolves = new ArrayList<>(old\_wolves);  
 if(Figure.*check\_for\_duplicates*(new\_old\_wolves, f)) {  
 new\_old\_wolves.set(l, new Figure(old\_wolves.get(l).getX() + 1, old\_wolves.get(l).getY() + 1, false));  
 new\_wolves.add(new\_old\_wolves);  
 }  
  
 }  
 if(old\_wolves.get(l).getX()-1>0 && old\_wolves.get(l).getY()+1<8 && old\_wolves.get(l).getX()!=f.getX()  
 &&old\_wolves.get(l).getY()!=f.getY() ) {  
 ArrayList<Figure> new\_old\_wolves = new ArrayList<>(old\_wolves);  
 if(Figure.*check\_for\_duplicates*(new\_old\_wolves, f)) {  
 new\_old\_wolves.set(l, new Figure(old\_wolves.get(l).getX() - 1, old\_wolves.get(l).getY() + 1, false));  
 new\_wolves.add(new\_old\_wolves);  
 }  
 }  
 }  
 if(new\_wolves.size()!=0) {  
 for (ArrayList<Figure> newie : new\_wolves) {  
 if(Figure.*check\_for\_duplicates*(newie, null)) {  
 Node new\_node = new Node(newie, depth + 1);  
 int val = 0;  
 for (Figure wolf\_f : newie) {  
 if (wolf\_f.y != 0)  
 val += 1;  
 val += Math.*abs*(wolf\_f.y - f.y) + Math.*abs*(wolf\_f.x - f.x);  
 }  
 new\_node.setValue(val);  
 children\_node.addChild(new\_node);  
 ArrayList<ArrayList<Figure>> newie\_arr\_list = new ArrayList<>();  
 newie\_arr\_list.add(newie);  
  
 make\_tree(new\_node, newie\_arr\_list, f\_list, depth + 1);  
 }  
 }  
 }  
  
 }  
  
 }  
  
  
  
 }  
  
 }  
  
  
  
 }  
  
  
  
 public ArrayList<Figure> possible\_options\_for\_wolves(int depth, Figure last\_rabbit, Node current\_node, ArrayList<Figure>rabbit\_movements, int iter\_movements)  
 {  
 ArrayList<Figure> answer = new ArrayList<>();  
 if(current\_node.getDepth()==depth-2 ) {  
 ArrayList<Figure> last\_rabbit\_list = new ArrayList<>();  
 last\_rabbit\_list.add(last\_rabbit);  
 ArrayList<Node> cur\_node\_children = current\_node.getChildren();  
 for(Node find: cur\_node\_children)  
 {  
 if(find.get\_figure().get(0).getX()==last\_rabbit\_list.get(0).getX() && find.get\_figure().get(0).getY()==last\_rabbit\_list.get(0).getY())  
 {  
 answer = new ArrayList<>( find.getChildren().get(0).get\_figure());  
 return answer;  
 }  
 }  
 }  
 else if(current\_node.getDepth()<depth-2) {  
 if(current\_node.getDepth()%2!=0) {  
 for (Node child : current\_node.getChildren()) {  
 if(child.get\_figure().get(0).getX()==rabbit\_movements.get(iter\_movements).getX() && child.get\_figure().get(0).getY()==rabbit\_movements.get(iter\_movements).getY())  
 {  
 iter\_movements+=1;  
 return possible\_options\_for\_wolves(depth, last\_rabbit, child,rabbit\_movements,iter\_movements);  
 }  
 }  
  
 }  
 else  
 return possible\_options\_for\_wolves(depth, last\_rabbit, current\_node.getChildren().get(0),rabbit\_movements,iter\_movements);  
 }  
 return answer;  
 }  
  
 public ArrayList<Figure> alpha\_beta\_pruning(int depth, Figure last\_rabbit, Node current\_node, ArrayList<Figure>rabbit\_movements, int iter\_movements)  
 {  
 ArrayList<Figure> answer = new ArrayList<>();  
 if(current\_node.getDepth()==depth-2 ) {  
 ArrayList<Figure> last\_rabbit\_list = new ArrayList<>();  
 last\_rabbit\_list.add(last\_rabbit);  
 ArrayList<Node> cur\_node\_children = current\_node.getChildren();  
 for(Node find: cur\_node\_children)  
 {  
 if(find.get\_figure().get(0).getX()==last\_rabbit\_list.get(0).getX() && find.get\_figure().get(0).getY()==last\_rabbit\_list.get(0).getY())  
 {  
 int max = 0;  
 Node answ = null;  
 for(Node n: find.getChildren())  
 {  
 if(n.getValue()>max)  
 {  
 max = n.getValue();  
 answ = n;  
 }  
 }  
 assert answ != null;  
 return answ.get\_figure();  
 }  
 }  
 }  
 else if(current\_node.getDepth()<depth-2) {  
 if(current\_node.getDepth()%2!=0) {  
 for (Node child : current\_node.getChildren()) {  
 if(child.get\_figure().get(0).getX()==rabbit\_movements.get(iter\_movements).getX() && child.get\_figure().get(0).getY()==rabbit\_movements.get(iter\_movements).getY())  
 {  
 iter\_movements+=1;  
 return alpha\_beta\_pruning(depth, last\_rabbit, child,rabbit\_movements,iter\_movements);  
 }  
 }  
  
 }  
 else  
 return alpha\_beta\_pruning(depth, last\_rabbit, current\_node.getChildren().get(0),rabbit\_movements,iter\_movements);  
 }  
 return answer;  
 }  
  
 public ArrayList<Figure> possible\_options\_for\_wolves\_med(int depth, Figure last\_rabbit, Node current\_node, ArrayList<Figure>rabbit\_movements, int iter\_movements)  
 {  
 ArrayList<Figure> answer = new ArrayList<>();  
 if(current\_node.getDepth()==depth-2 ) {  
 ArrayList<Figure> last\_rabbit\_list = new ArrayList<>();  
 last\_rabbit\_list.add(last\_rabbit);  
 ArrayList<Node> cur\_node\_children = current\_node.getChildren();  
 for(Node find: cur\_node\_children)  
 {  
 if(find.get\_figure().get(0).getX()==last\_rabbit\_list.get(0).getX() && find.get\_figure().get(0).getY()==last\_rabbit\_list.get(0).getY())  
 {  
 Random random = new Random();  
 int pos = random.nextInt(find.getChildren().size()-1);  
 answer = new ArrayList<>( find.getChildren().get(pos).get\_figure());  
 return answer;  
 }  
 }  
 }  
 else if(current\_node.getDepth()<depth-2) {  
 if(current\_node.getDepth()%2!=0) {  
 for (Node child : current\_node.getChildren()) {  
 if(child.get\_figure().get(0).getX()==rabbit\_movements.get(iter\_movements).getX() && child.get\_figure().get(0).getY()==rabbit\_movements.get(iter\_movements).getY())  
 {  
 iter\_movements+=1;  
 return possible\_options\_for\_wolves(depth, last\_rabbit, child,rabbit\_movements,iter\_movements);  
 }  
 }  
  
 }  
 else  
 return possible\_options\_for\_wolves(depth, last\_rabbit, current\_node.getChildren().get(0),rabbit\_movements,iter\_movements);  
 }  
 return answer;  
 }  
  
  
  
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.



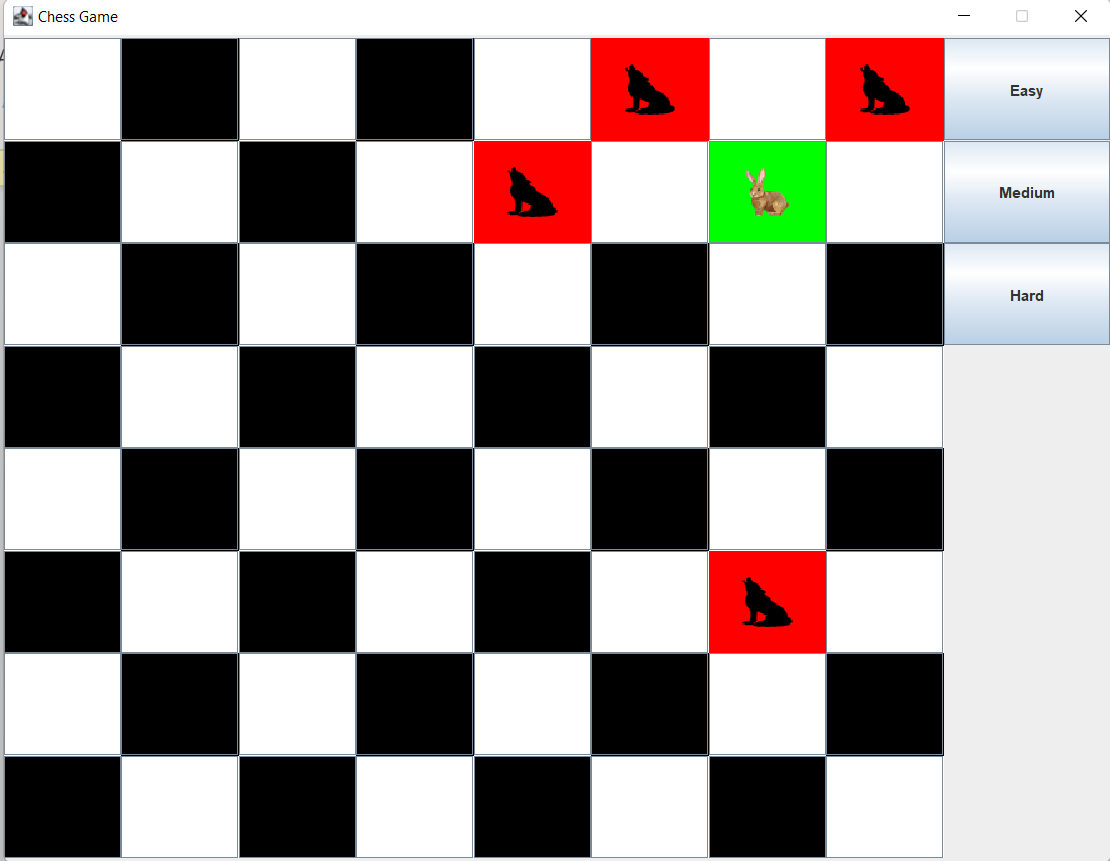


Рисунок 3.1, 3.2 – Звичайна гра гравець керує зайцем

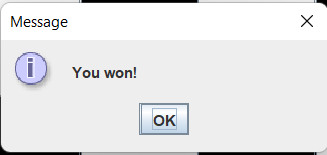


Рисунок 3.3 – Повідомлення яке видається якщо виграє гравець

Висновок

В рамках даної лабораторної роботи було вивчено основні підходи до формалізації алгоритмів знаходження рішень задач в умовах протидії. Була створена гра в якій один з компонентів керується гравцем, а інші комп’ютером. Алгоритм підбирає найкращий для виграшу комп’ютера варіант, який базується на тому чи оточений гравець з усіх сторін чи ні, якщо ні то наскільки близько персонаж комп’ютеру знаходиться до гравця. Будується дерево різних варіантів розвитку подій, на основі нього і працює алгоритм альфа-бета відсікань, який не бере до уваги гілки значення яких вже перевищують значення знайденої мінімальної гілки. Було також ознайомлено з підходами до програмування алгоритмів штучного інтелекту в іграх з повною інформацією, іграх з елементами випадковості та в іграх з неповною інформацією. Створена була саме гра з повною інформацією.

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).