



**人工智能实验报告**

**课程名称\_\_\_\_\_人工智能实验 \_**

**学生学院\_\_ 计算机学院\_\_\_\_\_\_**

**专业班级\_\_\_ 16级计科3班**

**学 号\_\_\_\_ 3116004560 \_\_\_**

**学生姓名\_\_\_\_ 刘付彩文 \_ \_**

**2018 年 11 月 17 日**

**α -β剪枝法实现井字棋**

1. **实验目的**

熟悉和掌握博弈树算法过程，实现从极大极小值算法过渡到利用α -β剪枝算法实现人机之间井字棋对弈。

1. **实验环境**
2. 硬件环境：微型计算机。
3. 软件环境：Windows操作系统，Java语言，eclipse开发工具。
4. **实验原理**

α -β 剪枝技术的基本思想或算法是，边生成博弈树边计算评估各节点的倒推值，并且根据评估出的倒推值范围，及时停止扩展那些已无必要再扩展的子节点，即相当于剪去了博弈树上的一些分枝，从而节约了机器开销，提高了搜索效率。

具体的剪枝方法如下：

(1) 对于一个与节点 MIN，若能估计出其倒推值的上确界 β，并且这个 β 值不大于 MIN 的父节点(一定是或节点)的估计倒推值的下确界 α，即 α≥β，则就不必再扩展该MIN 节点的其余子节点了(因为这些节点的估值对 MIN 父节点的倒推值已无任何影响了)。这一过程称为 α 剪枝。

(2) 对于一个或节点 MAX，若能估计出其倒推值的下确界 α，并且这个 α 值不小于 MAX 的父节点(一定是与节点)的估计倒推值的上确界 β，即 α≥β，则就不必再扩展该 MAX 节点的其余子节点了(因为这些节点的估值对 MAX 父节点的倒推值已无任何影响 了)。这一过程称为 β 剪枝。

从算法中看到：

(1) MAX 节点(包括起始节点)的 α 值永不减少；

(2) MIN 节点(包括起始节点)的 β 值永不增加。

在搜索期间，α 和 β 值的计算如下：

(1) 一个 MAX 节点的 α 值等于其后继节点当前最大的最终倒推值。

(2) 一个 MIN 节点的 β 值等于其后继节点当前最小的最终倒推值。

1. **设计实现**
2. 类定义：

Start类：main方法，启动井字棋

ChessBoard类：该类封装是棋盘的界面以及α -β剪枝算法的操作方

1. 程序流程图：

N

比赛结束?

棋盘

计算落子位置

选择先手

Y

玩家下

电脑下

比赛结束

N N

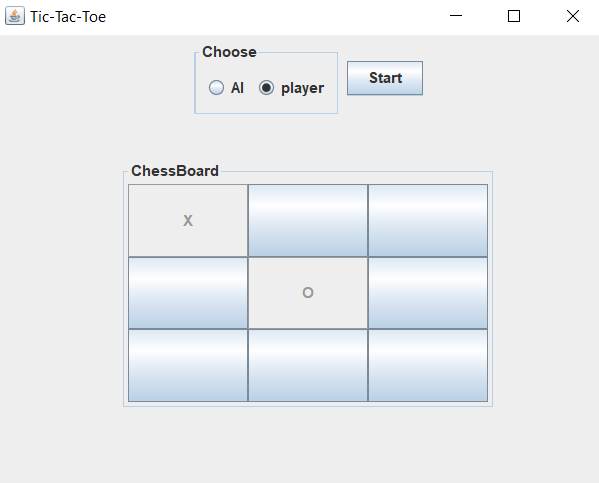
Y

比赛结束?

Y

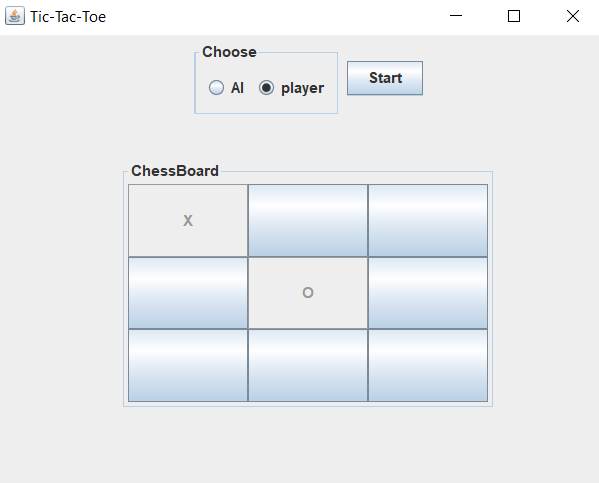
计算落子位置

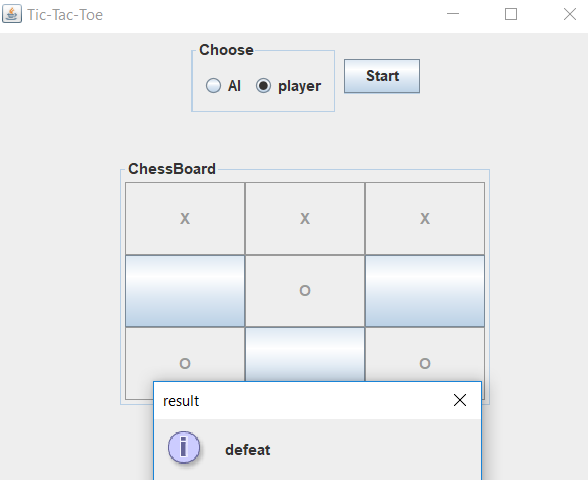
1. **使用说明**

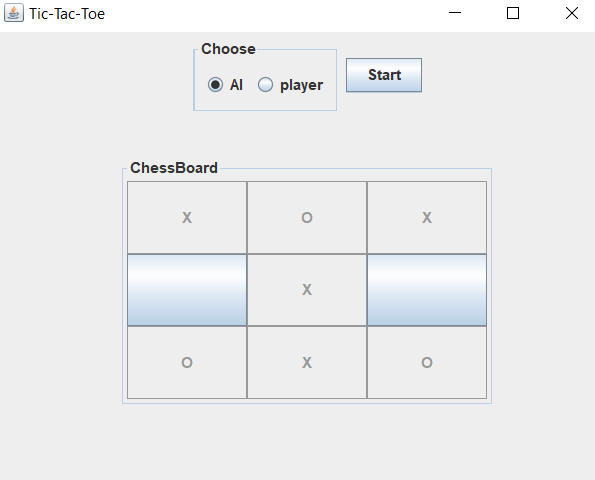


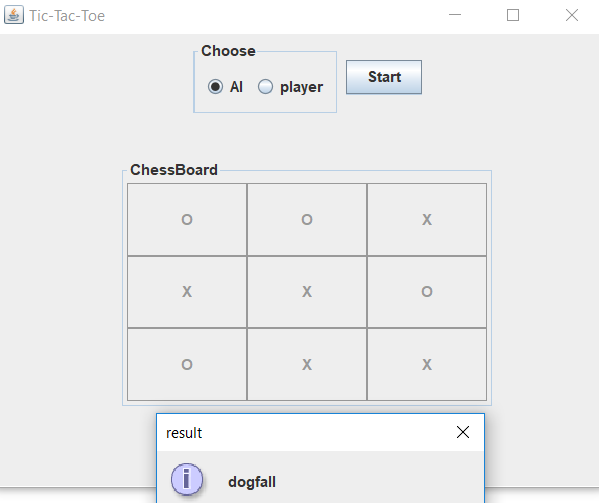
* + - 1. **选择先手，机器或玩家**
      2. **Start之前无法进行落子**
      3. **同一格子只能下子一次**
      4. **玩家为“0”，机器为“x”**

1. **运行与测试**









1. **心得体会**

本实验难度不算大，主要是掌握极大极小以及AlphaBeta算法的实现。在实验中的主要问题是进行剪枝的判断，而在这过程中也进行了较长时间的调试。但完成该实验也是十分有趣的，运用人工智能课程学到的知识，完成一个能够真正会下井字棋的智能玩家。

1. **源代码**

public class ChessBoard extends JFrame {

JButton chessButton[] = new JButton[9];

int chessBoard[] = {1, 1, 1, 1, 1, 1, 1, 1, 1};

int chessValue;

String playerMan;

int chessPosition;

public ChessBoard() {

setSize(500, 400);

setTitle("Tic-Tac-Toe");

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

JPanel upPanel = new JPanel();

JPanel checkPanel = new JPanel();

JPanel downPanel = new JPanel();

//JRadioButton

JRadioButton rb\_AI = new JRadioButton("AI");

rb\_AI.setSelected(true);

JRadioButton rb\_player = new JRadioButton("player");

ButtonGroup checkGroup = new ButtonGroup();

checkGroup.add(rb\_AI);

checkGroup.add(rb\_player);

checkPanel.add(rb\_AI);

checkPanel.add(rb\_player);

//set border to choose

checkPanel.setBorder(new TitledBorder(null, "Choose", TitledBorder.LEADING, TitledBorder.TOP, null,null));

//TextField

//TextField depthText = new TextField("2", 10);

//JButton

JButton startButton = new JButton("Start");

upPanel.add(checkPanel);

//upPanel.add(depthText);

upPanel.add(startButton);

setLayout(null);

upPanel.setBounds(0, 0, 500, 100);

//only can use 'setBounds' when the frame's layout is null

add(upPanel);

startButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

//----clear chessBoard

playerMan = "";

chessPosition = -1;

for(int i = 0; i < 9; i++) {

chessBoard[i] = 1;

}

for(int i = 0; i < 9; i++) {

chessButton[i].setText("");

chessButton[i].setEnabled(true);

}

//----clear chessBoard

//----the first chess

if(rb\_AI.isSelected()) {

findPosition();

playerMan = "player";

}else {

playerMan = "player";

}

//----the first chess

}

});

//downPanel button

downPanel.setLayout(new GridLayout(3, 3));

for(int i = 0; i < 9; i++) {

chessButton[i] = new JButton();

chessButton[i].setEnabled(false);

downPanel.add(chessButton[i]);

}

downPanel.setBounds(100, 100, 300, 200);

//set border to chessBoard

downPanel.setBorder(new TitledBorder(null, "ChessBoard", TitledBorder.LEADING, TitledBorder.TOP, null,null));

add(downPanel);

for(int i = 0; i < 9; i++) {

chessButton[i].addActionListener(new ChessAction());

}

}

}

//set button to unable

void setChessButtonUnable() {

for(int i = 0; i < 9; i++) {

chessButton[i].setText("");

chessButton[i].setEnabled(false);

}

}

int chessAnalysis() {

int victoryTerm[][] = {{0, 1, 2}, {0, 4, 8}, {0, 3, 6}, {1, 4, 7}, {2, 5, 8}, {2, 4, 6}, {3, 4, 5}, {6, 7, 8}};

//eight victory term

for(int[] a : victoryTerm) {

if(isChessLine(chessButton[a[0]], chessButton[a[1]], chessButton[a[2]])) {

if("O".equals(chessButton[a[0]].getText()))return 1;

else if("X".equals(chessButton[a[0]].getText()))return 2;

}

}//judge chessResult

return 3;

}

boolean isChessLine(JButton button\_1, JButton button\_2, JButton button\_3) {

if(button\_1.getText().equals(button\_2.getText()) && button\_2.getText().equals(button\_3.getText())) {

return true;

}else {

return false;

}

}

class ChessAction implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

//ButtonListener

for(int i = 0; i < 9; i++) {

if(e.getSource() == chessButton[i]) {

if("player".equals(playerMan)) {

//playerPlaying

chessButton[i].setText("O");

chessButton[i].setEnabled(false);

chessBoard[i] = 0;

playerMan = "AI";

break;

}else {

//AIPlaying

chessButton[i].setEnabled(false);

chessButton[i].setText("X");

chessBoard[i] = 0;

playerMan = "player";

break;

}

}

}

switch(chessAnalysis()) {

case 1:

JOptionPane.showMessageDialog(null, "victory", "result", JOptionPane.INFORMATION\_MESSAGE);

//set button to unable

setChessButtonUnable();

break;

case 2:

JOptionPane.showMessageDialog(null, "defeat", "result", JOptionPane.INFORMATION\_MESSAGE);

//set button to unable

setChessButtonUnable();

break;

case 3:

if(isChessBoardFull()) {

JOptionPane.showMessageDialog(null, "dogfall", "result", JOptionPane.INFORMATION\_MESSAGE);

//set button to unable

setChessButtonUnable();

}else {

//playing when player is AI

if("AI".equals(playerMan)) {

findPosition();

playerMan = "player";

}

}

break;

}

}

}

void findPosition() {

int analysis = -100;

int value = 100;

int topValue = -100;

int position = 0;

int testBoard\_1[] = chessBoard;

String[] testChess\_1 = new String[9];

for(int t = 0; t < 9; t++) {

if("X".equals(chessButton[t].getText())) testChess\_1[t] = "X";

else if("O".equals(chessButton[t].getText())) testChess\_1[t] = "O";

else testChess\_1[t] = "1";

}

for(int i = 0; i < 9; i++) {

if(testBoard\_1[i] == 0) continue;

value = 100;

testBoard\_1[i] = 0;

testChess\_1[i] = "X";

if(judge(testChess\_1, "X")) value = 100;

else {

for(int j = 0; j < 9; j++) {

if(testBoard\_1[j] == 0)continue;

int Xcount = 0, Ocount = 0;

int tempValue;

testBoard\_1[j] = 0;

testChess\_1[j] = "O";

if(judge(testChess\_1, "O")) value = -100;

else {

Xcount = countLine(testBoard\_1, testChess\_1, "X");

Ocount = countLine(testBoard\_1, testChess\_1, "O");

tempValue = Xcount - Ocount;

if(tempValue <= value) value = tempValue;

}

testBoard\_1[j] = 1;

testChess\_1[j] = "1";

}

}

testBoard\_1[i] = 1;

testChess\_1[i] = "1";

if(analysis < value) {

analysis = value;

position = i;

}

if(analysis > topValue) {

topValue = analysis;

}

}

System.out.println(position);

AIPlaying(position);

}

boolean judge(String[] testChess, String index){

int victoryTerm[][] = {{0, 1, 2}, {0, 4, 8}, {0, 3, 6}, {1, 4, 7}, {2, 5, 8}, {2, 4, 6}, {3, 4, 5}, {6, 7, 8}};

//eight victory term

for(int[] a : victoryTerm) {

if(testChess[a[0]].equals(testChess[a[1]]) && testChess[a[1]].equals(testChess[a[2]])) {

if(index.equals(testChess[a[0]])) return true;

}

}//judge chessResult

return false;

}

int countLine(int[] testBoard, String[] testChess, String index) {

int count = 0;

for(int i = 0; i < 9; i++) {

if(testBoard[i] == 1) testChess[i] = index;

}

int victoryTerm[][] = {{0, 1, 2}, {0, 4, 8}, {0, 3, 6}, {1, 4, 7}, {2, 5, 8}, {2, 4, 6}, {3, 4, 5}, {6, 7, 8}};

//eight victory term

for(int[] a : victoryTerm) {

if(testChess[a[0]].equals(testChess[a[1]]) && testChess[a[1]].equals(testChess[a[2]])) {

if(index.equals(testChess[a[0]])) {

count++;

}

}

}

for(int i = 0; i < 9; i++) {

if(testBoard[i] == 1) testChess[i] = "1";

}

return count;

}

}