

《人工智能》课程系列

TicTacToe 实验平台的设计与实现*

武汉纺织大学数学与计算机学院

杜小勤

2018/09/09

Contents

1	Array 类	2
2	Array2D 类	4
3	TicTacToe 类	7
4	TTTDraw 类	12
5	TTTInput 类	16
6	Minimax 算法	17
7	$\alpha - \beta$ 算法	20
8	Monte Carlo 树搜索算法	24
9	参考文献	31

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: November 9, 2019。

1 Array 类

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Sep 9 19:25:08 2018
4
5  @author: duxiaoqin
6
7  Functions:
8      (1) Array class;
9  """
10
11 import random
12 import ctypes
13
14 class Array:
15     def __init__(self, size):
16         assert size > 0, 'Array size must be > 0'
17         self.size = size
18         PyArrayType = ctypes.py_object * size
19         self.elements = PyArrayType()
20         self.clear(None)
21
22     def clone(self):
23         newa = Array(len(self))
24         for index in range(len(self)):
25             newa[index] = self[index]
26         return newa
27
28     def print(self):
29         for index in range(len(self)):
```

```
30         print(self.elements[index], end=' ')
31
32     def __len__(self):
33         return self.size
34
35     def __getitem__(self, index):
36         assert index >= 0 and index < len(self), \
37             'Array subscript out of range'
38         return self.elements[index]
39
40     def __setitem__(self, index, value):
41         assert index >= 0 and index < len(self), \
42             'Array subscript out of range'
43         self.elements[index] = value
44
45     def clear(self, value):
46         for i in range(len(self)):
47             self.elements[i] = value
48
49     def __iter__(self):
50         return ArrayIterator(self.elements)
51
52     class ArrayIterator:
53         def __init__(self, theArray):
54             self.arrayRef = theArray
55             self.curNdx = 0
56
57         def __iter__(self):
58             return self
59
60         def __next__(self):
```

```
61         if self.curNdx < len(self.arrayRef):
62             entry = self.arrayRef[self.curNdx]
63             self.curNdx = self.curNdx + 1
64             return entry
65         else:
66             raise StopIteration
67
68 def main():
69     a = Array(10)
70     for i in range(len(a)):
71         a[i] = random.random()
72     a.print()
73
74 if __name__ == '__main__':
75     main()
```

2 Array2D 类

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Sep 9 20:25:08 2018
4
5  @author: duxiaoqin
6
7  Functions:
8      (1) Array2D class;
9  """
10
11 import random
12 from myarray import Array
13
```

```
14 class Array2D:
15     def __init__(self, numRows, numCols):
16         self.theRows = Array(numRows)
17
18         for i in range(numRows):
19             self.theRows[i] = Array(numCols)
20
21     def clone(self):
22         newa2d = Array2D(self.numRows(), self.numCols())
23         for row in range(self.numRows()):
24             for col in range(self.numCols()):
25                 newa2d.theRows[row][col] = self.theRows[row][col]
26         return newa2d
27
28     def print(self):
29         for i in range(self.numRows()):
30             self.theRows[i].print()
31             print()
32
33     def numRows(self):
34         return len(self.theRows)
35
36     def numCols(self):
37         return len(self.theRows[0])
38
39     def clear(self, value):
40         for row in range(self.numRows()):
41             self.theRows[row].clear(value)
42
43     def __getitem__(self, ndxTuple):
44         assert len(ndxTuple) == 2, 'Invalid number of array subscripts.'
```

```
45         row = ndxTuple[0]
46         col = ndxTuple[1]
47         assert row >= 0 and row < self.numRows() and \
48             col >= 0 and col < self.numCols(), \
49             "Array subscript out of range."
50         the1dArray = self.theRows[row]
51         return the1dArray[col]
52
53     def __setitem__(self, ndxTuple, value):
54         assert len(ndxTuple) == 2, 'Invalid number of array subscripts.'
55         row = ndxTuple[0]
56         col = ndxTuple[1]
57         assert row >= 0 and row < self.numRows() and \
58             col >= 0 and col < self.numCols(), \
59             'Array subscript out of range.'
60         the1dArray = self.theRows[row]
61         the1dArray[col] = value
62
63     def main():
64         a = Array2D(10, 5)
65         for r in range(a.numRows()):
66             for c in range(a.numCols()):
67                 a[r, c] = random.random()
68
69         a.print()
70
71     if __name__ == '__main__':
72         main()
```

3 TicTacToe 类

TicTacToe 类实现棋盘的管理，具体的功能有：棋盘初始化、棋盘状态的更新、棋手管理、胜负判断等。

下面定义 TicTacToe 的 ADT：

- TicTacToe()

创建一个 TicTacToe 对象，初始化棋盘为空（所有棋盘格均为 None）；

- clone()

克隆当前的 TicTacToe 对象，生成一个新对象并返回该对象；

- play(row, col)

当前棋手在 (row, col) 处落子，并出让落子权给对方。棋手与棋盘都被改变；

- getPlayer()

返回当前棋手：True-黑方、False-白方；

- getAllMoves()

返回当前棋局的所有可落子位置（元组列表）；

- isGameOver()

判断棋局是否结束：None-未结束、1-黑方胜、-1-白方胜、0-平局；

下面给出 TicTacToe 类的 ADT 实现：

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Sep 10 22:25:08 2018
4
5  @author: duxiaoqin
6
7  Functions:
8      (1) TicTacToe class;
9  """
```

```
10
11 from myarray2d import Array2D
12
13 class TicTacToe:
14
15     BLACK = True
16     WHITE = False
17     EMPTY = None
18
19     BLACKWIN = 1
20     WHITEWIN = -1
21     DRAW = 0
22
23     def __init__(self):
24         self.board = Array2D(3, 3)
25         self.player = TicTacToe.BLACK
26         self.black = []
27         self.white = []
28
29         self.magic = Array2D(3, 3)
30         self.magic[0, 0] = 2
31         self.magic[0, 1] = 9
32         self.magic[0, 2] = 4
33
34         self.magic[1, 0] = 7
35         self.magic[1, 1] = 5
36         self.magic[1, 2] = 3
37
38         self.magic[2, 0] = 6
39         self.magic[2, 1] = 1
40         self.magic[2, 2] = 8
```



```
41
42     def reset(self):
43         self.board.clear(None)
44         self.player = TicTacToe.BLACK
45         self.black = []
46         self.white = []
47
48     def clone(self):
49         newttt = TicTacToe()
50         for row in range(3):
51             for col in range(3):
52                 newttt.board[row, col] = self.board[row, col]
53         newttt.player = self.player
54         newttt.black = self.black[:]
55         newttt.white = self.white[:]
56
57         return newttt
58
59     def ToString(self):
60         l = []
61         for row in range(3):
62             for col in range(3):
63                 if self.board[row, col] == TicTacToe.BLACK:
64                     l.append('X')
65                 elif self.board[row, col] == TicTacToe.WHITE:
66                     l.append('O')
67                 else:
68                     l.append('_')
69         return ''.join(l)
70
71     def print(self):
```

```
72         for row in range(3):
73             for col in range(3):
74                 if self.board[row, col] == TicTacToe.BLACK:
75                     print('X', end=' ')
76                 elif self.board[row, col] == TicTacToe.WHITE:
77                     print('O', end=' ')
78                 else:
79                     print('_', end=' ')
80             print()
81
82     def play(self, row, col):
83         self.board[row, col] = self.player
84         if self.player == TicTacToe.BLACK:
85             self.black.append(self.magic[row, col])
86         else:
87             self.white.append(self.magic[row, col])
88         self.player = not self.player
89
90     def getPlayer(self):
91         return self.player
92
93     def getAllMoves(self):
94         return [(row, col) for row in range(3) \
95                 for col in range(3) \
96                 if self.board[row, col] == TicTacToe.EMPTY]
97
98     def isWin(self, n, goal, moves):
99         moves_clone = moves[:]
100         if n == 0:
101             return goal == 0
102         elif goal <= 0:
```

```
103         return False
104     elif len(moves_clone) == 0:
105         return False
106     else:
107         item = moves_clone.pop(0)
108         if self.isWin(n-1, goal-item, moves_clone[:]):
109             return True
110         elif self.isWin(n, goal, moves_clone[:]):
111             return True
112     return False
113
114 def isGameOver(self):
115     if self.isWin(3, 15, self.black):
116         return TicTacToe.BLACKWIN
117     elif self.isWin(3, 15, self.white):
118         return TicTacToe.WHITEWIN
119     elif len(self.black)+len(self.white) == 9:
120         return TicTacToe.DRAW
121     else:
122         return None
123
124 def main():
125     ttt = TicTacToe()
126     ttt.play(1, 1)
127     ttt.play(0, 0)
128     ttt.play(2, 0)
129     ttt.play(0, 1)
130     ttt.play(0, 2)
131     ttt.print()
132     print(ttt.isGameOver())
133     print(ttt.ToString())
```

```
134
135 if __name__ == '__main__':
136     main()
```

4 TTTDraw 类

TTTDraw 类实现棋盘的绘制功能。下面是 TTTDraw 类的 ADT 定义：

- TTTDraw(gui)

创建一个 TTTDraw 对象，参数 gui 为图形接口；

- draw(ttt)

依据参数 ttt 绘制棋盘，棋盘格有三种状态：空白、黑方与白方。参数 ttt 是 TicTacToe 类的实例；

下面给出 TTTDraw 类的 ADT 实现：

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Sep 11 15:16:17 2018
4
5  @author: duxiaoqin
6
7  Functions:
8      (1) TTTDraw class;
9  """
10
11 from graphics import *
12 from tictactoe import *
13 from ttinput import *
14
15 class TTTDraw:
16     WIDTH = 5.0
```

```
17     HEIGHT = 5.0
18     START = 1.0
19     END = 4.0
20
21     def __init__(self, win):
22         self.win = win
23         self.win.setCoords(0.0, 0.0, TTDraw.WIDTH, TTDraw.HEIGHT)
24
25         self.lines = []
26         for offset in range(4):
27             l = Line(Point(TTDraw.START, TTDraw.START+offset), \
28                     Point(TTDraw.END, TTDraw.START+offset))
29             l.setWidth(3)
30             self.lines.append(l)
31             l = Line(Point(TTDraw.START+offset, TTDraw.START), \
32                     Point(TTDraw.START+offset, TTDraw.END))
33             l.setWidth(3)
34             self.lines.append(l)
35
36         self.ximg = Image(Point(0, 0), 'x.gif')
37         self.oimg = Image(Point(0, 0), 'o.gif')
38
39         self.ximgs = Array2D(3, 3)
40         for row in range(3):
41             for col in range(3):
42                 newximg = self.ximg.clone()
43                 newximg.move(TTDraw.START+1/2+col, TTDraw.END-1/2-row)
44                 self.ximgs[row, col] = newximg
45         self.oimgs = Array2D(3, 3)
46         for row in range(3):
47             for col in range(3):
```

```
48         newoimg = self.oimg.clone()
49         newoimg.move(TTTDraw.START+1/2+col, TTTDraw.END-1/2-row)
50         self.oimgs[row, col] = newoimg
51
52     self.text = Text(Point(2.5, 0.5), '')
53     self.text.setTextColor('red')
54
55     def draw_lines(self):
56         for l in self.lines:
57             l.undraw()
58         for l in self.lines:
59             l.draw(self.win)
60
61     def draw_ttt(self, ttt):
62         self.text.undraw()
63         if ttt.isGameOver() == TicTacToe.BLACKWIN:
64             self.text.setText('X Win')
65         elif ttt.isGameOver() == TicTacToe.WHITEWIN:
66             self.text.setText('O Win')
67         elif ttt.isGameOver() == TicTacToe.DRAW:
68             self.text.setText('X/O Draw')
69         elif ttt.getPlayer() == TicTacToe.BLACK:
70             self.text.setText('X to play')
71         elif ttt.getPlayer() == TicTacToe.WHITE:
72             self.text.setText('O to play')
73         self.text.draw(self.win)
74
75     for row in range(3):
76         for col in range(3):
77             self.ximgs[row, col].undraw()
78             self.oimgs[row, col].undraw()
```

```
79
80     for row in range(3):
81         for col in range(3):
82             if ttt.board[row, col] == TicTacToe.BLACK:
83                 self.ximgs[row, col].draw(self.win)
84             elif ttt.board[row, col] == TicTacToe.WHITE:
85                 self.oimgs[row, col].draw(self.win)
86
87     def draw(self, ttt):
88         self.draw_lines()
89         self.draw_ttt(ttt)
90         self.win.update()
91
92     def main():
93         win = GraphWin('TTTDraw', 600, 600, autoflush=False)
94         ttt = TicTacToe()
95         tttdraw = TTTDraw(win)
96         tttinput = TTTInput(win)
97
98         while win.checkKey() != 'Escape':
99             tttinput.input(ttt)
100             tttdraw.draw(ttt)
101             if ttt.isGameOver() != None:
102                 ttt.reset()
103                 win.getMouse()
104         win.close()
105
106     if __name__ == '__main__':
107         main()
```

5 TTTInput 类

TTTInput 类实现棋盘的输入功能：控制鼠标落子。下面是 TTTInput 类的 ADT 定义：

- TTTInput(gui)

创建一个 TTTInput 对象，参数 gui 为图形接口；

- Input(ttt)

控制鼠标在空白棋盘格处落子 (依据参数 ttt 获取空白棋盘格的位置)，ttt 被改变。落子成功，返回 True；否则，返回 False；

下面给出 TTTInput 类的 ADT 实现：

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Sep 11 19:13:37 2018
4
5  @author: duxiaoqin
6
7  Functions:
8      (1) TTTInput class;
9  """
10
11  from graphics import *
12  from tictactoe import *
13
14  class TTTInput:
15      def __init__(self, win):
16          self.win = win
17
18      def input(self, ttt):
19          mpos = self.win.checkMouse()
```



```

20         if mpos == None:
21             return False
22         moves = ttt.getAllMoves()
23         row, col = 4-int(mpos.getY())-1, int(mpos.getX())-1
24         if (row, col) not in moves:
25             return False
26         ttt.play(row, col)
27         return True

```

6 Minimax 算法

Minimax 算法如下:

```

def Minimax(node, depth, player):
    if depth == 0 or node is a terminal node:
        return the heuristic value of node
    if player == True:
        bestValue = -∞
        for each child of node:
            v = Minimax(child, depth-1, False)
            bestValue = max(bestValue, v)
        return bestValue
    else:
        bestValue = +∞
        for each child of node:
            v = Minimax(child, depth-1, True)
            bestValue = min(bestValue, v)
        return bestValue

```

TicTacToe 的 Minimax 对弈程序如下:

```

1  # -*- coding: utf-8 -*-
2  """

```

```
3  Created on Fri Oct 26 14:41:12 2018
4
5  @author: duxiaoqin
6  Functions:
7      (1) Minimax Algorithm for TicTacToe
8  """
9
10 from graphics import *
11 from tictactoe import *
12 from tttdraw import *
13 from ttinput import *
14 import sys
15
16 def Minimax(node, depth):
17     result = node.isGameOver()
18     if result != None:
19         return result, (), depth
20     if node.getPlayer() == TicTacToe.BLACK:
21         bestValue = -sys.maxsize
22         bestMove = ()
23         bestDepth = sys.maxsize
24         moves = node.getAllMoves()
25         for move in moves:
26             child = node.clone()
27             child.play(*move)
28             v, _, leafDepth = Minimax(child, depth+1)
29             if bestValue == v and bestDepth > leafDepth:
30                 bestValue = v
31                 bestMove = move
32                 bestDepth = leafDepth
33         if bestValue < v:
```

```
34         bestValue = v
35         bestMove = move
36         return bestValue, bestMove, bestDepth
37     else:
38         bestValue = sys.maxsize
39         bestMove = ()
40         bestDepth = sys.maxsize
41         moves = node.getAllMoves()
42         for move in moves:
43             child = node.clone()
44             child.play(*move)
45             v, _, leafDepth = Minimax(child, depth+1)
46             if bestValue == v and bestDepth > leafDepth:
47                 bestValue = v
48                 bestMove = move
49                 bestDepth = leafDepth
50             if bestValue > v:
51                 bestValue = v
52                 bestMove = move
53         return bestValue, bestMove, bestDepth
54
55 def main():
56     win = GraphWin('Minimax for TicTacToe', 600, 600, autoflush=False)
57     ttt = TicTacToe()
58     tttdraw = TTDraw(win)
59     tttinput = TTTInput(win)
60     tttdraw.draw(ttt)
61
62     while win.checkKey() != 'Escape':
63         if ttt.getPlayer() == TicTacToe.WHITE:
64             v, move, _ = Minimax(ttt, 0)
```

```

65         if move != ():
66             ttt.play(*move)
67         tttinput.input(ttt)
68         tttdraw.draw(ttt)
69         if ttt.isGameOver() != None:
70             time.sleep(1)
71             ttt.reset()
72             tttdraw.draw(ttt)
73             #win.getMouse()
74     win.close()
75
76 if __name__ == '__main__':
77     main()

```

7 $\alpha - \beta$ 算法

$\alpha - \beta$ 算法如下:

```

def alpha-beta(node, depth, alpha, beta, player):
    if depth == 0 or node is a terminal node:
        return the heuristic value of node
    if player:
        v = -∞
        for each child of node:
            v = max(v, alpha-beta(child, depth-1, alpha, beta, False))
            alpha = max(alpha, v)
            if beta <= alpha:
                break #beta pruning
        return v
    else:
        v = +∞
        for each child of node:

```

```

        v = min(v, alphabeta(child, depth-1, alpha, beta, True))
        beta = min(beta, v)
        if beta <= alpha:
            break #alpha pruning
    return v

```

TicTacToe 的 $\alpha - \beta$ 对弈程序如下:

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Fri Oct 26 20:53:08 2018
4
5  @author: duxiaoqin
6  Functions:
7      (1) Alpha-Beta Algorithm for TicTacToe
8  """
9
10 from graphics import *
11 from tictactoe import *
12 from tttdraw import *
13 from ttinput import *
14 import sys
15 import time
16
17 def AlphaBeta(node, depth, alpha, beta):
18     result = node.isGameOver()
19     if result != None:
20         return result, (), depth
21     if node.getPlayer() == TicTacToe.BLACK:
22         bestValue = -sys.maxsize
23         bestMove = ()
24         bestDepth = sys.maxsize
25         moves = node.getAllMoves()

```

```
26         for move in moves:
27             child = node.clone()
28             child.play(*move)
29             v, _, leafDepth = AlphaBeta(child, depth+1, alpha, beta)
30             if bestValue == v and bestDepth > leafDepth:
31                 bestValue = v
32                 bestMove = move
33                 bestDepth = leafDepth
34             if bestValue < v:
35                 bestValue = v
36                 bestMove = move
37             alpha = max(alpha, bestValue)
38             if beta <= alpha:
39                 break #beta pruning
40         return bestValue, bestMove, bestDepth
41     else:
42         bestValue = sys.maxsize
43         bestMove = ()
44         bestDepth = sys.maxsize
45         moves = node.getAllMoves()
46         for move in moves:
47             child = node.clone()
48             child.play(*move)
49             v, _, leafDepth = AlphaBeta(child, depth+1, alpha, beta)
50             if bestValue == v and bestDepth > leafDepth:
51                 bestValue = v
52                 bestMove = move
53                 bestDepth = leafDepth
54             if bestValue > v:
55                 bestValue = v
56                 bestMove = move
```

```
57         beta = min(beta, bestValue)
58         if beta <= alpha:
59             break #alpha pruning
60         return bestValue, bestMove, bestDepth
61
62 def main():
63     win = GraphWin('Minimax for TicTacToe', 600, 600, autoflush=False)
64     ttt = TicTacToe()
65     tttdraw = TTDraw(win)
66     tttinput = TTTInput(win)
67     tttdraw.draw(ttt)
68
69     while win.checkKey() != 'Escape':
70         if ttt.getPlayer() == TicTacToe.WHITE:
71             v, move, _ = AlphaBeta(ttt, 0, -sys.maxsize, sys.maxsize)
72             if move != ():
73                 ttt.play(*move)
74             tttinput.input(ttt)
75             tttdraw.draw(ttt)
76             if ttt.isGameOver() != None:
77                 time.sleep(1)
78                 ttt.reset()
79                 tttdraw.draw(ttt)
80                 #win.getMouse()
81     win.close()
82
83 if __name__ == '__main__':
84     main()
```

8 Monte Carlo 树搜索算法

MCTS 算法如下:

```
def MCTS(root):  
    decision_time = MAX_TIME  
    for time in range(decision_time):  
        path = [] #for backpropagation  
        node = Select(root)  
        simulation_node = Expand(node)  
        simulation_result = Simulate(simulation_node)  
        Backpropagate(simulation_result)  
    retrun a child of root, with highest number of visits  
  
def Select(node):  
    path.append(node)  
    while node is nonterminal and node is fully expanded:  
        node = a best UCT child of node  
        path.append(node)  
    return node  
  
def Expand(node):  
    path.append(node)  
    if node is nonterminal:  
        child = a random child of node  
        path.append(child)  
        return child  
    else:  
        return node  
  
def Simulate(node):  
    while node is nonterminal:
```



```
        node = a random child of node
    return result(node)
```

```
def Backpropagate(result):
    for node in path:
        update node's statistics with result
```

程序如下:

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Nov 12 19:55:03 2018
4
5  @author: duxiaoqin
6  Functions:
7      (1) MCTS Algorithm for TicTacToe
8  """
9
10 from graphics import *
11 from tictactoe import *
12 from tttdraw import *
13 from ttinput import *
14 import sys
15 import time
16 import math
17 from random import *
18
19 class NodeInfo:
20     def __init__(self):
21         self.player = None
22         self.visit = 0
23         self.win = 0
24
```

```
25 def MCTS(root, nodes_map):
26     def Select(node):
27         node_key = node.ToString()
28         path.append(node_key)
29         node_info = nodes_map.get(node_key)
30         if node_info == None:
31             node_info = NodeInfo()
32             node_info.player = node.getPlayer()
33             nodes_map[node_key] = node_info
34
35         while node.isGameOver() == None and isFullyExpanded(node):
36             node = BestUCTChild(node)
37             child_key = node.ToString()
38             path.append(child_key)
39             child_info = nodes_map.get(child_key)
40             if child_info == None:
41                 child_info = NodeInfo()
42                 child_info.player = node.getPlayer()
43                 nodes_map[child_key] = child_info
44
45         return node
46
47     def Expand(node):
48         node_key = node.ToString()
49         path.append(node_key)
50         node_info = nodes_map.get(node_key)
51         if node_info == None:
52             node_info = NodeInfo()
53             node_info.player = node.getPlayer()
54             nodes_map[node_key] = node_info
55
```

```
56         if node.isGameOver() == None:
57             node = RandomUnvisitedChild(node)
58             child_key = node.ToString()
59             path.append(child_key)
60             child_info = nodes_map.get(child_key)
61             if child_info == None:
62                 child_info = NodeInfo()
63                 child_info.player = node.getPlayer()
64                 nodes_map[child_key] = child_info
65             return node
66         else:
67             return node
68
69     def Simulate(node):
70         result = node.isGameOver()
71         while result == None:
72             node = RandomChild(node)
73             result = node.isGameOver()
74         return result
75
76     def Backpropagate(result):
77         for node_key in path:
78             UpdateStatistics(node_key, result)
79
80     def MaxVisitChild(node):
81         max_visit_num = -sys.maxsize
82         max_visit_child = ()
83         moves = node.getAllMoves()
84         for move in moves:
85             tmp_node = node.clone()
86             tmp_node.play(*move)
```

```
87         child_info = nodes_map.get(tmp_node.ToString())
88         if child_info == None:
89             continue
90         if max_visit_num < child_info.visit:
91             max_visit_num = child_info.visit
92             max_visit_child = move
93     return max_visit_child
94
95 def isFullyExpanded(node):
96     moves = node.getAllMoves()
97     for move in moves:
98         tmp_node = node.clone()
99         tmp_node.play(*move)
100         child_info = nodes_map.get(tmp_node.ToString())
101         if child_info == None:
102             return False
103     return True
104
105 def BestUCTChild(node):
106     c = 1.4142135623730951
107     best_uct = -sys.maxsize
108     best_uct_child = None
109     node_info = nodes_map[node.ToString()]
110     moves = node.getAllMoves()
111     for move in moves:
112         tmp_node = node.clone()
113         tmp_node.play(*move)
114         child_key = tmp_node.ToString()
115         child_info = nodes_map[child_key]
116         ucb1 = child_info.win / child_info.visit + \
```

```
117         c * math.sqrt(math.log(node_info.visit) /
118             ↪ child_info.visit)
119
120     if best_uct < ucb1:
121         best_uct = ucb1
122         best_uct_child = move
123     if best_uct_child != None:
124         node.play(*best_uct_child)
125     return node
126
127 def RandomChild(node):
128     moves = node.getAllMoves()
129     node.play(*moves[randint(0, len(moves) - 1)])
130     return node
131
132 def RandomUnvisitedChild(node):
133     moves = node.getAllMoves()
134     while True:
135         tmp_node = node.clone()
136         move = moves[randint(0, len(moves) - 1)]
137         tmp_node.play(*move)
138         child_info = nodes_map.get(tmp_node.ToString())
139         if child_info == None:
140             return tmp_node
141
142 def UpdateStatistics(node_key, result):
143     node_info = nodes_map[node_key]
144     node_info.visit += 1
145     if node_info.player == TicTacToe.BLACK:
146         if result == -1:
147             node_info.win += 1
148         elif result == 0:
```

```
147         node_info.win += 0.5
148     else:
149         if result == 1:
150             node_info.win += 1
151         elif result == 0:
152             node_info.win += 0.5
153
154     decision_time = 500
155     for time in range(decision_time):
156         node = root.clone()
157         path = []
158         node = Select(node)
159         simulation_node = Expand(node)
160         simulation_result = Simulate(simulation_node)
161         Backpropagate(simulation_result)
162     return MaxVisitChild(root)
163
164 def main():
165     win = GraphWin('MCTS for TicTacToe', 600, 600, autoflush=False)
166     ttt = TicTacToe()
167     tttdraw = TTDraw(win)
168     tttinput = TTTInput(win)
169     tttdraw.draw(ttt)
170
171     nodes_map = {}
172     while win.checkKey() != 'Escape':
173         if ttt.getPlayer() == TicTacToe.WHITE:
174             move = MCTS(ttt, nodes_map)
175             if move != ():
176                 ttt.play(*move)
177             tttinput.input(ttt)
```

```
178         tttdraw.draw(ttt)
179         if ttt.isGameOver() != None:
180             time.sleep(1)
181             ttt.reset()
182             tttdraw.draw(ttt)
183             #win.getMouse()
184     win.close()
185
186 if __name__ == '__main__':
187     main()
```

9 参考文献

1. 杜小勤。《人工智能》课程系列，Part I: Python 程序设计基础，2018/06/13。
2. 杜小勤。《人工智能》课程系列，Part II: Python 算法基础，2018/07/31。
3. 杜小勤。《人工智能》课程系列，Chapter 5: 博弈树搜索技术，2018/10/23。