《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

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: November 13, 2019。

1 Array 类

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 9 19:25:08 2018

@author: duxiaoqin

Functions:
    (1) Array class;
"""

import random
import ctypes

class Array:
    def __init__(self, size):
        assert size > 0, 'Array size must be > 0'
        self.size = size
        PyArrayType = ctypes.py_object * size
        self.elements = PyArrayType()
        self.clear(None)

    def clone(self):
        newa = Array(len(self))
        for index in range(len(self)):
            newa[index] = self[index]
        return newa

```

```
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：

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 10 22:25:08 2018

@author: duxiaoqin

Functions:
    (1) TicTacToe class;
"""
```

```
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 tttinput 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, TTTDraw.WIDTH, TTTDraw.HEIGHT)
24
25          self.lines = []
26          for offset in range(4):
27              l = Line(Point(TTTDraw.START, TTTDraw.START+offset), \
28                       Point(TTTDraw.END, TTTDraw.START+offset))
29              l.setWidth(3)
30              self.lines.append(l)
31              l = Line(Point(TTTDraw.START+offset, TTTDraw.START), \
32                       Point(TTTDraw.START+offset, TTTDraw.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(TTTDraw.START+1/2+col, TTTDraw.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：

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 11 19:13:37 2018

@author: duxiaoqin

Functions:
    (1) TTTInput class;
"""

from graphics import *
from tictactoe import *

class TTTInput:
    def __init__(self, win):
        self.win = win

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

```
65          if ttt.getPlayer() == TicTacToe.WHITE:
66              v, move, _ = Minimax(ttt, 0)
67              if move != ():
68                  ttt.play(*move)
69          tttinput.input(ttt)
70          tttdraw.draw(ttt)
71          if ttt.isGameOver() != None:
72              time.sleep(1)
73              ttt.reset()
74              tttdraw.draw(ttt)
75              #win.getMouse()
76      win.close()
77
78  if __name__ == '__main__':
79      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, alpah-beta(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：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 26 20:53:08 2018

@author: duxiaoqin
Functions:
    (1) Alpha-Beta Algorithm for TicTacToe
"""

from graphics import *
from tictactoe import *
from tttdraw import *
from tttinput import *
import sys
import time

def AlphaBeta(node, depth, alpha, beta):
    result = node.isGameOver()
    if result != None:
        return result, (), depth
    if node.getPlayer() == TicTacToe.BLACK:
        bestValue = -sys.maxsize
        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                  bestDepth = leafDepth
38              alpha = max(alpha, bestValue)
39              if beta <= alpha:
40                  break #beta pruning
41          return bestValue, bestMove, bestDepth
42      else:
43          bestValue = sys.maxsize
44          bestMove = ()
45          bestDepth = sys.maxsize
46          moves = node.getAllMoves()
47          for move in moves:
48              child = node.clone()
49              child.play(*move)
50              v, _, leafDepth = AlphaBeta(child, depth+1, alpha, beta)
51              if bestValue == v and bestDepth > leafDepth:
52                  bestValue = v
53                  bestMove = move
54                  bestDepth = leafDepth
```

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

```
86      main()
```

8 Monte Carlo 树搜索算法

MCTS 算法如下:

```
def MCTS(root):
    seed()
    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：

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 12 19:55:03 2018

@author: duxiaoqin
Functions:
    (1) MCTS Algorithm for TicTacToe
"""

from graphics import *
from tictactoe import *
from tttdraw import *
from tttinput import *
import sys
import time
import math
from random import *

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

```
145                 node_info.win += 1
146             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 = TTTDraw(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 On-Policy TD(0) 算法

下面给出 Tic-Tac-Toe 的 On-Policy TD(0)（ϵ-greedy 控制策略）算法：

```
Input:
    root: the root game state;
    V: all V(s) are initialized with 0.5;
    alpha: 0.5
    epsilon: 0.1
    learning_time: 10000
Output:
    best move (with V changed)
def TDLearning(root, V, alpha, epsilon, learning_time)
    seed()
    for i in range(learning_time):
        node = root.clone()
        parent = None
        result = node.isGameOver()
        while result == None:#node is nonterminal
```

```
        if random() < epsilon:
            Choose a move of node randomly
            node.play(move)
            result = node.isGameOver()
            if result != None:#node is terminal
                UpdateTerminalNode(node, result)
                if parent != None:
                    Update V(parent) with V(node) & alpha, by
                    ↪ TD(0) learning rule
                parent = None
            else:
                parent = node.clone()
        else:
            move = BestMove(node)
            node.play(move)
            result = node.isGameOver()
            if result != None:#node is terminal
                UpdateTerminalNode(node, result)
            if parent != None:
                Update V(parent) with V(node) & alpha, by TD(0)
                ↪ learning rule
            if result != None:#node is terminal
                parent = None
            else:
                parent = node.clone()
        if result == None:#node is nonterminal:
            Choose a move randomly for the opponent
            node.play(move)
            result = node.isGameOver()
    UpdateTerminalNode(node, result)
    if parent != None:
```

```
            Update V(parent) with V(node) & alpha, by TD(0) learning
            ↪ rule
    return BestMove(root)

    def UpdateTerminalNode(node, result):
        if V(node) exists:
            return
        if result is X win:
            V(node) = 1
        elif result is O win:
            V(node) = 0
        elif result is draw:
            V(node) = 0.5

    #Exploring Start: in the beginning, all children should be
    ↪ selected uniformly, because each V(child) is 0.5
    def BestMove(node):
        if node is X player:
            return the move to the child of node with the MAX
            ↪ V(child)
        else:
            return the move to the child of node with the MIN
            ↪ V(child)
```

实现程序如下：

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 21 14:49:32 2018

@author: duxiaoqin
Functions:
    (1) On-Policy TD(0) for TicTacToe, epsilon-greedy control
```

```
8  """
9
10 from graphics import *
11 from tictactoe import *
12 from tttdraw import *
13 from tttinput import *
14 import sys
15 import time
16 import math
17 from random import *
18 import pickle
19
20 def TDO(root, V, alpha, epsilon, learning_time):
21     def RandomMove(node):
22         moves = node.getAllMoves()
23         return moves[randint(0, len(moves)-1)]
24
25     def UpdateTerminalNode(node, result):
26         key = node.ToString()
27         if V.get(key) != None:
28             return
29         if result == TicTacToe.BLACKWIN:
30             V[key] = 1
31         elif result == TicTacToe.WHITEWIN:
32             V[key] = 0
33         else:
34             V[key] = 0.5
35
36     def UpdateValueFunction(node1, node2, alpha):
37         key1 = node1.ToString()
38         key2 = node2.ToString()
```

```
39          if V.get(key1) == None:
40              V[key1] = 0.5
41          if V.get(key2) == None:
42              V[key2] = 0.5
43          V[key1] += alpha * (V[key2] - V[key1])
44
45      def BestMove(node):
46          if node.getPlayer() == TicTacToe.BLACK:
47              best_value = -sys.maxsize
48              best_move = ()
49              moves = node.getAllMoves()
50              for move in moves:
51                  tmp_node = node.clone()
52                  tmp_node.play(*move)
53                  key = tmp_node.ToString()
54                  if V.get(key) == None:
55                      continue
56                  if best_value < V[key]:
57                      best_value = V[key]
58                      best_move = move
59              if best_move == () and len(moves) != 0:
60                  best_move = RandomMove(node)
61              return best_move
62          else:
63              best_value = sys.maxsize
64              best_move = ()
65              moves = node.getAllMoves()
66              for move in moves:
67                  tmp_node = node.clone()
68                  tmp_node.play(*move)
69                  key = tmp_node.ToString()
```

```
70                  if V.get(key) == None:
71                      continue
72                  if best_value > V[key]:
73                      best_value = V[key]
74                      best_move = move
75              if best_move == () and len(moves) != 0:
76                  best_move = RandomMove(node)
77              return best_move
78
79      for i in range(learning_time):
80          node = root.clone()
81          parent = None
82          result = node.isGameOver()
83          while result == None:
84              if random() < epsilon:
85                  move = RandomMove(node)
86                  node.play(*move)
87                  result = node.isGameOver()
88                  if result != None:
89                      UpdateTerminalNode(node, result)
90                      if parent != None:
91                          UpdateValueFunction(parent, node, alpha)
92                      parent = None
93                  else:
94                      parent = node.clone()
95              else:
96                  move = BestMove(node)
97                  node.play(*move)
98                  result = node.isGameOver()
99                  if result != None:
100                     UpdateTerminalNode(node, result)
```

```
101                 if parent != None:
102                     UpdateValueFunction(parent, node, alpha)
103                 if result != None:
104                     parent = None
105                 else:
106                     parent = node.clone()
107             if result == None:
108                 move = RandomMove(node)
109                 node.play(*move)
110                 result = node.isGameOver()
111         UpdateTerminalNode(node, result)
112         if parent != None:
113             UpdateValueFunction(parent, node, alpha)
114     return BestMove(root)
115 
116 def main():
117     seed()
118     win = GraphWin('TD-Learning(0) for TicTacToe', 600, 600,
    ↪  autoflush=False)
119     ttt = TicTacToe()
120     tttdraw = TTTDraw(win)
121     tttinput = TTTInput(win)
122     tttdraw.draw(ttt)
123 
124     try:
125         vfile = open('ValueFunction.dat', 'rb')
126         V = pickle.load(vfile)
127         vfile.close()
128     except FileNotFoundError:
129         V = {}
130 
```

```
131         #Start to self-play
132         self_play = 100
133         for i in range(self_play):
134             tmp_root = ttt.clone()
135             tttdraw.draw(tmp_root)
136             result = tmp_root.isGameOver()
137             while result == None:
138                 move = TD0(tmp_root, V, 0.5, 0.1, 2000)
139                 #moves = tmp_root.getAllMoves()
140                 #move = moves[randint(0, len(moves)-1)]
141                 if move != ():
142                     tmp_root.play(*move)
143                 tttdraw.draw(tmp_root)
144                 result = tmp_root.isGameOver()
145                 if result == None:
146                     move = TD0(tmp_root, V, 0.5, 0.1, 2000)
147                     if move != ():
148                         tmp_root.play(*move)
149                 tttdraw.draw(tmp_root)
150                 result = tmp_root.isGameOver()
151                 if result != None:
152                     time.sleep(0.5)
153         #Save V to file
154         vfile = open('ValueFunction.dat', 'wb')
155         pickle.dump(V, vfile)
156         vfile.close()
157 
158     while win.checkKey() != 'Escape':
159         if ttt.getPlayer() == TicTacToe.WHITE:
160             move = TD0(ttt, V, 0.2, 0.1, 500)
161             if move != ():
```

```
162                 ttt.play(*move)
163         tttinput.input(ttt)
164         tttdraw.draw(ttt)
165         if ttt.isGameOver() != None:
166             time.sleep(1)
167             ttt.reset()
168             tttdraw.draw(ttt)
169             #win.getMouse()
170     win.close()
171
172 if __name__ == '__main__':
173     main()
```

10 遗传算法

TicTacToe 的遗传算法如下：

```
Input:
    None
Output:
    the best solution
def GA():
    generation_num = 3000
    population_num = 800
    prob_crossover = 0.15
    prob_replicate = 0.10
    prob_mutation = 0.001
    INDIVIDUAL_TEMPLATE = {}
    STATE = {}
    POPULATION = []
    FITNESS = [0]*population_num
    PROB = [0]*population_num
```

```
Init()
for t in range(generation_num):
    P_TMP = copy of POPULATION
    for i in range(population_num):
        seed()
        if random() <= prob_replicate:
            POPULATION[i] = Select(P_TMP)
        else:
            d1 = Select(P_TMP)
            d2 = Select(P_TMP)
            d = Crossover(d1, d2)
            Mutate(d)
            POPULATION[i] = d

    fitness_sum = CalculateFitness()

    #Update the statistics of population
    PROB[0] = FITNESS[0]/fitness_sum
    for i in range(1, len(FITNESS)):
        PROB[i] = PROB[i-1]+FITNESS[i]/fitness_sum

return the individual with MAX_FITNESS of POPULATION

def Init():
    ttt = TicTacToe()
    GenerateIndividualTemplate(ttt)
    items = INDIVIDUAL_TEMPLATE.items()
    for i in range(population_num):
        individual = GenRandomIndividual(items)
        POPULATION.append(individual)
    fitness_sum = CalculateFitness()
```

```
        PROB[0] = FITNESS[0]/fitness_sum
        for i in range(1, len(FITNESS)):
            PROB[i] = PROB[i-1]+FITNESS[i]/fitness_sum

    def GenerateIndividualTemplate(ttt):
        if ttt.isGameOver() != None:
            return

        moves = ttt.getAllMoves()
        ttt_str = ttt.ToString()
        if STATE.get(ttt_str) == None:
            for equ_str in GenEquivalent(ttt_str):
                STATE[equ_str] = ttt_str #base state
            INDIVIDUAL_TEMPLATE[ttt_str] = moves
        for move in moves:
            node = ttt.clone()
            node.play(move)
            GenerateIndividualTemplate(node)

    def GenRandomIndividual(items):
        seed()
        individual = {}
        for ttt_str, moves in items:
            individual[ttt_str] = Random(moves)
        return individual

    def Select(population):
        r = random()
        for i in range(len(PROB)):
            if r <= PROB[i]:
                return copy of population[i]
```

```
#d1, d2: two individuals
def Crossover(d1, d2):
    d = {}
    for key in d1.keys():
        r = random()
        if r <= prob_crossover:
            d[key] = d1[key]
        else:
            d[key] = d2[key]
    return d

#d: individual
#d[i][0]: encode of state i
#d[i][1]: move of state i
def Mutate(d):
    for key in d.keys():
        if random() <= prob_mutation:
            moves = INDIVIDUAL_TEMPLATE[key]
            d[key] = Random(moves)

def CalculateFitness():
    PLAY_NUM = [0]*population_num
    LOST_NUM = [0]*population_num
    for i in range(population_num):
        ttt = TicTacToe()
        lost_num, play_num = PlayGameAsFirst(ttt,
        ↪ POPULATION[i])#from ttt to all states, as the first
        ↪ player
        LOST_NUM[i] += lost_num
        PLAY_NUM[i] += play_num
```

```
            ttt = TicTacToe()
            lost_num, play_num = PlayGameAsSecond(ttt,
            ↪  POPULATION[i])#from ttt to all states, as the second
            ↪  player
            LOST_NUM[i] += lost_num
            PLAY_NUM[i] += play_num
        fitness_sum = 0
        for i in range(population_num):
            FITNESS[i] = 1 - LOST_NUM[i]/PLAY_NUM[i]
            fitness_sum += FITNESS[i]
        return fitness_sum
```

程序如下：

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 28 21:23:04 2018

@author: duxiaoqin
Functions:
    (1) GA for TicTacToe, evolving a perfect strategy which never
    ↪  loses a game.
"""

from graphics import *
from tictactoe import *
from tttdraw import *
from tttinput import *
import sys
import time
from random import *
import numpy as np
```

```
18 import matplotlib.pyplot as plt
19 import pickle
20 import copy
21
22 EQUIVALENT = [
23               [0,1,2,3,4,5,6,7,8],
24               [6,3,0,7,4,1,8,5,2],
25               [8,7,6,5,4,3,2,1,0],
26               [2,5,8,1,4,7,0,3,6],
27               [6,7,8,3,4,5,0,1,2],
28               [8,5,2,7,4,1,6,3,0],
29               [2,1,0,5,4,3,8,7,6],
30               [0,3,6,1,4,7,2,5,8]]
31 generation_num = 3000
32 population_num = 800
33 prob_crossover = 0.15
34 prob_replicate = 0.10
35 prob_mutation = 0.001
36 INDIVIDUAL_TEMPLATE = {}
37 STATE = {}
38 POPULATION = []
39 FITNESS = [0]*population_num
40 PROB = [0]*population_num
41 T = range(generation_num)
42 BEST_FITNESS = [1]*generation_num
43 MAX_FITNESS = [0]*generation_num
44 AVERAGE_FITNESS = [0]*generation_num
45 MAX_INDIVIDUAL = [0]*generation_num
46
47 def GenEquivalent(ttt_str):
48     TTT_STR = []
```

```
49      for index in EQUIVALENT:
50          TTT_STR.append(''.join([ttt_str[i] for i in index]))
51      return TTT_STR
52
53  def GenEquivalentMove(base_str, base_move, equ_str):
54      TTT_STR = GenEquivalent(base_str)
55      move = base_move[0]*3+base_move[1]
56      equ_index = TTT_STR.index(equ_str)
57      move_index = EQUIVALENT[equ_index].index(move)
58      return (move_index // 3, move_index % 3)
59
60  def Init():
61      global INDIVIDUAL_TEMPLATE, STATE
62      try:
63          template_file = open('IndividualTemplate.dat', 'rb')
64          INDIVIDUAL_TEMPLATE = pickle.load(template_file)
65          template_file.close()
66
67          state_file = open('State.dat', 'rb')
68          STATE = pickle.load(state_file)
69          state_file.close()
70      except FileNotFoundError:
71          INDIVIDUAL_TEMPLATE = {}
72          STATE = {}
73          ttt = TicTacToe()
74          GenerateIndividualTemplate(ttt)
75
76          template_file = open('IndividualTemplate.dat', 'wb')
77          pickle.dump(INDIVIDUAL_TEMPLATE, template_file)
78          template_file.close()
79
```

```
80          state_file = open('State.dat', 'wb')
81          pickle.dump(STATE, state_file)
82          state_file.close()
83
84      items = INDIVIDUAL_TEMPLATE.items()
85      print(len(items))
86      for i in range(population_num):
87          individual = GenRandomIndividual(items)
88          POPULATION.append(individual)
89      fitness_sum = CalculateFitness()
90      PROB[0] = FITNESS[0]/fitness_sum
91      for i in range(1, len(FITNESS)):
92          PROB[i] = PROB[i-1]+FITNESS[i]/fitness_sum
93
94  def GenerateIndividualTemplate(ttt):
95      if ttt.isGameOver() != None:
96          return
97
98      moves = ttt.getAllMoves()
99      ttt_str = ttt.ToString()
100     if STATE.get(ttt_str) == None:
101         for equ_str in GenEquivalent(ttt_str):
102             STATE[equ_str] = ttt_str #base state
103         INDIVIDUAL_TEMPLATE[ttt_str] = moves
104     for move in moves:
105         node = ttt.clone()
106         node.play(*move)
107         GenerateIndividualTemplate(node)
108
109 def GenRandomIndividual(items):
110     seed()
```

```
111     individual = {}
112     for ttt_str, moves in items:
113         individual[ttt_str] = moves[randint(0, len(moves)-1)]
114     return individual
115 
116 def Select(population):
117     r = random()
118     for i in range(len(PROB)):
119         if r <= PROB[i]:
120             return copy.deepcopy(population[i])
121 
122 #d1, d2: two individuals
123 def Crossover(d1, d2):
124     d = {}
125     for key in d1.keys():
126         r = random()
127         if r <= prob_crossover:
128             d[key] = d1[key]
129         else:
130             d[key] = d2[key]
131     return d
132 
133 #d: individual
134 #d[i][0]: encode of state i
135 #d[i][1]: move of state i
136 def Mutate(d):
137     for key in d.keys():
138         if random() <= prob_mutation:
139             moves = INDIVIDUAL_TEMPLATE[key]
140             d[key] = moves[randint(0, len(moves)-1)]
141 
```

```
142 def CalculateFitness():
143     PLAY_NUM = [0]*population_num
144     LOST_NUM = [0]*population_num
145     for i in range(population_num):
146         ttt = TicTacToe()
147         lost_num, play_num = PlayGameAsFirst(ttt, POPULATION[i])
148         LOST_NUM[i] += lost_num
149         PLAY_NUM[i] += play_num
150         ttt = TicTacToe()
151         lost_num, play_num = PlayGameAsSecond(ttt, POPULATION[i])
152         LOST_NUM[i] += lost_num
153         PLAY_NUM[i] += play_num
154     fitness_sum = 0
155     for i in range(population_num):
156         FITNESS[i] = 1 - LOST_NUM[i]/PLAY_NUM[i]
157         fitness_sum += FITNESS[i]
158     return fitness_sum
159 
160 def PlayGameAsFirst(ttt, d):
161     all_lost_num = 0
162     all_play_num = 0
163     result = ttt.isGameOver()
164     if result != None:
165         if result == TicTacToe.WHITEWIN:
166             return 1, 1
167         else:
168             return 0, 1
169     ttt_str = ttt.ToString()
170     base_str = STATE[ttt_str]
171     move = GenEquivalentMove(base_str, d[base_str], ttt_str)
172     ttt.play(*move)
```

```
173     result = ttt.isGameOver()
174     if result != None:
175         if result == TicTacToe.WHITEWIN:
176             return 1, 1
177         else:
178             return 0, 1
179 
180     moves = ttt.getAllMoves()
181     for move in moves:
182         node = ttt.clone()
183         node.play(*move)
184         result = node.isGameOver()
185         if result != None:
186             if result == TicTacToe.WHITEWIN:
187                 all_lost_num += 1
188             all_play_num += 1
189         else:
190             lost_num, play_num = PlayGameAsFirst(node, d)
191             all_lost_num += lost_num
192             all_play_num += play_num
193 
194     return all_lost_num, all_play_num
195 
196 def PlayGameAsSecond(ttt, d):
197     all_lost_num = 0
198     all_play_num = 0
199     result = ttt.isGameOver()
200     if result != None:
201         if result == TicTacToe.BLACKWIN:
202             return 1, 1
203         else:
```

```
204             return 0, 1
205
206     moves = ttt.getAllMoves()
207     for move in moves:
208         node = ttt.clone()
209         node.play(*move)
210         result = node.isGameOver()
211         if result != None:
212             if result == TicTacToe.BLACKWIN:
213                 all_lost_num += 1
214             all_play_num += 1
215         else:
216             ttt_str = node.ToString()
217             base_str = STATE[ttt_str]
218             move = GenEquivalentMove(base_str, d[base_str], ttt_str)
219             node.play(*move)
220             result = node.isGameOver()
221             if result != None:
222                 if result == TicTacToe.BLACKWIN:
223                     all_lost_num += 1
224                 all_play_num += 1
225             else:
226                 lost_num, play_num = PlayGameAsSecond(node, d)
227                 all_lost_num += lost_num
228                 all_play_num += play_num
229
230     return all_lost_num, all_play_num
231
232 def GetBestIndividual():
233     max_fitness = -sys.maxsize
234     max_individual = None
```

```
235     for i in range(population_num):
236         if max_fitness < FITNESS[i]:
237             max_fitness = FITNESS[i]
238             max_individual = copy.deepcopy(POPULATION[i])
239     return max_individual
240 
241 def main():
242     global INDIVIDUAL_TEMPLATE, STATE, MAX_FITNESS, AVERAGE_FITNESS,
        ↪ MAX_INDIVIDUAL
243     try:
244         best_file = open('BestIndividual.dat', 'rb')
245         best_individual = pickle.load(best_file)
246         best_file.close()
247 
248         template_file = open('IndividualTemplate.dat', 'rb')
249         INDIVIDUAL_TEMPLATE = pickle.load(template_file)
250         template_file.close()
251 
252         state_file = open('State.dat', 'rb')
253         STATE = pickle.load(state_file)
254         state_file.close()
255 
256         maxfitness_file = open('MaxFitness.dat', 'rb')
257         MAX_FITNESS = pickle.load(maxfitness_file)
258         maxfitness_file.close()
259 
260         avgfitness_file = open('AverageFitness.dat', 'rb')
261         AVERAGE_FITNESS = pickle.load(avgfitness_file)
262         avgfitness_file.close()
263 
264         maxindividual_file = open('MaxIndividual.dat', 'rb')
```

```
265         MAX_INDIVIDUAL = pickle.load(maxindividual_file)
266         maxindividual_file.close()
267 
268     except FileNotFoundError:
269         Init()
270         for t in range(generation_num):
271             P_TMP = copy.deepcopy(POPULATION)
272             for i in range(population_num):
273                 seed()
274                 if random() <= prob_replicate:
275                     POPULATION[i] = Select(P_TMP)
276                 else:
277                     d1 = Select(P_TMP)
278                     d2 = Select(P_TMP)
279                     d = Crossover(d1, d2)
280                     Mutate(d)
281                     POPULATION[i] = d
282 
283             fitness_sum = CalculateFitness()
284 
285             #Update the statistics of population
286             PROB[0] = FITNESS[0]/fitness_sum
287             for i in range(1, len(FITNESS)):
288                 PROB[i] = PROB[i-1]+FITNESS[i]/fitness_sum
289 
290             MAX_FITNESS[t] = max(FITNESS)
291             AVERAGE_FITNESS[t] = fitness_sum/population_num
292             MAX_INDIVIDUAL[t] = GetBestIndividual()
293             print('t = ', t, ' Average Fitness = ',
                ↪  AVERAGE_FITNESS[t], \
294                   ' Max Fitness = ', MAX_FITNESS[t])
```

```
295             if MAX_FITNESS[t] == 1.0:
296                 break
297 
298         best_individual = GetBestIndividual()
299 
300         best_file = open('BestIndividual.dat', 'wb')
301         pickle.dump(best_individual, best_file)
302         best_file.close()
303 
304         maxfitness_file = open('MaxFitness.dat', 'wb')
305         pickle.dump(MAX_FITNESS, maxfitness_file)
306         maxfitness_file.close()
307 
308         avgfitness_file = open('AverageFitness.dat', 'wb')
309         pickle.dump(AVERAGE_FITNESS, avgfitness_file)
310         avgfitness_file.close()
311 
312         maxindividual_file = open('MaxIndividual.dat', 'wb')
313         pickle.dump(MAX_INDIVIDUAL, maxindividual_file)
314         maxindividual_file.close()
315 
316         plt.plot(T, BEST_FITNESS)
317         plt.plot(T, MAX_FITNESS)
318         plt.plot(T, AVERAGE_FITNESS)
319         plt.show()
320 
321     win = GraphWin('GA for TicTacToe', 600, 600, autoflush=False)
322     ttt = TicTacToe()
323     tttdraw = TTTDraw(win)
324     tttinput = TTTInput(win)
325     tttdraw.draw(ttt)
```

```
326
327     while win.checkKey() != 'Escape':
328         if ttt.getPlayer() == TicTacToe.WHITE:
329             ttt_str = ttt.ToString()
330             base_str = STATE[ttt_str]
331             move = GenEquivalentMove(base_str,
                ↪  best_individual[base_str], ttt_str)
332             if move != ():
333                 ttt.play(*move)
334         tttinput.input(ttt)
335         tttdraw.draw(ttt)
336         if ttt.isGameOver() != None:
337             time.sleep(1)
338             ttt.reset()
339             tttdraw.draw(ttt)
340             #win.getMouse()
341     win.close()
342
343 if __name__ == '__main__':
344     main()
```

程序在迭代到 1516 次时，最大适应度收敛到 1，表明策略可以保证算法不会输棋（因为胜与平的得分都是 1，而输的得分是 0），遗传迭代曲线如图10-1所示。

11 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。
4. Gregor Hochmuth. On the Genetic Evolution of a Perfect Tic-Tac-Toe Strategy. Stanford University.

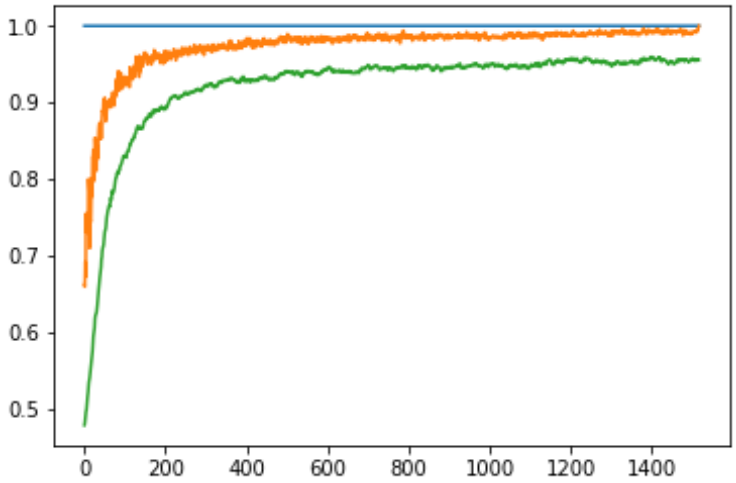

图 10-1: TicTacToe 遗传程序的迭代曲线