《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 On-Policy TD(0) 算法 **31**

10 遗传算法 **39**

[*]本系列文档属于讲义性质，仅用于学习目的。Last updated on: December 3, 2018。

11 参考文献 43

1 Array 类

```python
# -*- 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

    def print(self):
        for index in range(len(self)):
            print(self.elements[index], end=' ')

    def __len__(self):
        return self.size

    def __getitem__(self, index):
        assert index >= 0 and index < len(self), \
               'Array subscript out of range'
        return self.elements[index]

    def __setitem__(self, index, value):
        assert index >= 0 and index < len(self), \
               'Array subscript out of range'
        self.elements[index] = value

    def clear(self, value):
        for i in range(len(self)):
            self.elements[i] = value

    def __iter__(self):
        return ArrayIterator(self.elements)

class ArrayIterator:
    def __init__(self, theArray):
        self.arrayRef = theArray
        self.curNdx = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.curNdx < len(self.arrayRef):
            entry = self.arrayRef[self.curNdx]
            self.curNdx = self.curNdx + 1
            return entry
        else:
            raise StopIteration

def main():
    a = Array(10)
    for i in range(len(a)):
        a[i] = random.random()
    a.print()

if __name__ == '__main__':
    main()
```

2 Array2D 类

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 9 20:25:08 2018

@author: duxiaoqin

Functions:
    (1) Array2D class;
"""

import random
```

```python
from myarray import Array

class Array2D:
    def __init__(self, numRows, numCols):
        self.theRows = Array(numRows)

        for i in range(numRows):
            self.theRows[i] = Array(numCols)

    def clone(self):
        newa2d = Array2D(self.numRows(), self.numCols())
        for row in range(self.numRows()):
            for col in range(self.numCols()):
                newa2d.theRows[row][col] = self.theRows[row][col]
        return newa2d

    def print(self):
        for i in range(self.numRows()):
            self.theRows[i].print()
            print()

    def numRows(self):
        return len(self.theRows)

    def numCols(self):
        return len(self.theRows[0])

    def clear(self, value):
        for row in range(self.numRows()):
            self.theRows[row].clear(value)

    def __getitem__(self, ndxTuple):
        assert len(ndxTuple) == 2, 'Invalid number of array subscripts.'
        row = ndxTuple[0]
        col = ndxTuple[1]
        assert row >= 0 and row < self.numRows() and \
               col >= 0 and col < self.numCols(), \
               "Array subscript out of range."
        the1dArray = self.theRows[row]
        return the1dArray[col]

    def __setitem__(self, ndxTuple, value):
         assert len(ndxTuple) == 2, 'Invalid number of array subscripts.'
         row = ndxTuple[0]
         col = ndxTuple[1]
         assert row >= 0 and row < self.numRows() and \
                col >= 0 and col < self.numCols(), \
                'Array subscript out of range.'
         the1dArray = self.theRows[row]
         the1dArray[col] = value

def main():
    a = Array2D(10, 5)
    for r in range(a.numRows()):
        for c in range(a.numCols()):
            a[r, c] = random.random()

    a.print()

if __name__ == '__main__':
    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：

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

@author: duxiaoqin

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

```python

from myarray2d import Array2D

class TicTacToe:

    BLACK = True
    WHITE = False
    EMPTY = None

    BLACKWIN = 1
    WHITEWIN = -1
    DRAW = 0

    def __init__(self):
        self.board = Array2D(3, 3)
        self.player = TicTacToe.BLACK
        self.black = []
        self.white = []

        self.magic = Array2D(3, 3)
        self.magic[0, 0] = 2
        self.magic[0, 1] = 9
        self.magic[0, 2] = 4

        self.magic[1, 0] = 7
        self.magic[1, 1] = 5
        self.magic[1, 2] = 3

        self.magic[2, 0] = 6
        self.magic[2, 1] = 1
        self.magic[2, 2] = 8

    def reset(self):
        self.board.clear(None)
        self.player = TicTacToe.BLACK
        self.black = []
        self.white = []

    def clone(self):
        newttt = TicTacToe()
        for row in range(3):
            for col in range(3):
                newttt.board[row, col] = self.board[row, col]
        newttt.player = self.player
        newttt.black = self.black[:]
        newttt.white = self.white[:]

        return newttt

    def ToString(self):
        l = []
        for row in range(3):
            for col in range(3):
                if self.board[row, col] == TicTacToe.BLACK:
                    l.append('X')
                elif self.board[row, col] == TicTacToe.WHITE:
                    l.append('O')
                else:
                    l.append('_')
        return ''.join(l)

    def print(self):
        for row in range(3):
            for col in range(3):
                if self.board[row, col] == TicTacToe.BLACK:
                    print('X', end=' ')
                elif self.board[row, col] == TicTacToe.WHITE:
                    print('O', end=' ')
                else:
                    print('_', end=' ')
            print()

    def play(self, row, col):
        self.board[row, col] = self.player
        if self.player == TicTacToe.BLACK:
            self.black.append(self.magic[row, col])
        else:
            self.white.append(self.magic[row, col])
        self.player = not self.player

    def getPlayer(self):
        return self.player

    def getAllMoves(self):
        return [(row, col) for row in range(3) \
                                for col in range(3) \
                                    if self.board[row, col] == TicTacToe.EMPTY]

    def isWin(self, n, goal, moves):
        moves_clone = moves[:]
        if n == 0:
            return goal == 0
        elif goal <= 0:
            return False
        elif len(moves_clone) == 0:
            return False
        else:
            item = moves_clone.pop(0)
            if self.isWin(n-1, goal-item, moves_clone[:]):
                return True
            elif self.isWin(n, goal, moves_clone[:]):
                return True
        return False

    def isGameOver(self):
        if self.isWin(3, 15, self.black):
            return TicTacToe.BLACKWIN
        elif self.isWin(3, 15, self.white):
            return TicTacToe.WHITEWIN
        elif len(self.black)+len(self.white) == 9:
            return TicTacToe.DRAW
        else:
            return None

def main():
    ttt = TicTacToe()
    ttt.play(1, 1)
    ttt.play(0, 0)
    ttt.play(2, 0)
    ttt.play(0, 1)
    ttt.play(0, 2)
    ttt.print()
    print(ttt.isGameOver())
    print(ttt.ToString())
```

```python

if __name__ == '__main__':
    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：

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 11 15:16:17 2018

@author: duxiaoqin

Functions:
    (1) TTTDraw class;
"""

from graphics import *
from tictactoe import *
from tttinput import *

class TTTDraw:
    WIDTH = 5.0
    HEIGHT = 5.0
    START = 1.0
    END = 4.0

    def __init__(self, win):
        self.win = win
        self.win.setCoords(0.0, 0.0, TTTDraw.WIDTH, TTTDraw.HEIGHT)

        self.lines = []
        for offset in range(4):
            l = Line(Point(TTTDraw.START, TTTDraw.START+offset), \
                     Point(TTTDraw.END, TTTDraw.START+offset))
            l.setWidth(3)
            self.lines.append(l)
            l = Line(Point(TTTDraw.START+offset, TTTDraw.START), \
                     Point(TTTDraw.START+offset, TTTDraw.END))
            l.setWidth(3)
            self.lines.append(l)

        self.ximg = Image(Point(0, 0), 'x.gif')
        self.oimg = Image(Point(0, 0), 'o.gif')

        self.ximgs = Array2D(3, 3)
        for row in range(3):
            for col in range(3):
                newximg = self.ximg.clone()
                newximg.move(TTTDraw.START+1/2+col, TTTDraw.END-1/2-row)
                self.ximgs[row, col] = newximg
        self.oimgs = Array2D(3, 3)
        for row in range(3):
            for col in range(3):
                newoimg = self.oimg.clone()
                newoimg.move(TTTDraw.START+1/2+col, TTTDraw.END-1/2-row)
                self.oimgs[row, col] = newoimg

        self.text = Text(Point(2.5, 0.5), '')
        self.text.setTextColor('red')

    def draw_lines(self):
        for l in self.lines:
            l.undraw()
        for l in self.lines:
            l.draw(self.win)

    def draw_ttt(self, ttt):
        self.text.undraw()
        if ttt.isGameOver() == TicTacToe.BLACKWIN:
            self.text.setText('X Win')
        elif ttt.isGameOver() == TicTacToe.WHITEWIN:
            self.text.setText('O Win')
        elif ttt.isGameOver() == TicTacToe.DRAW:
            self.text.setText('X/O Draw')
        elif ttt.getPlayer() == TicTacToe.BLACK:
            self.text.setText('X to play')
        elif ttt.getPlayer() == TicTacToe.WHITE:
            self.text.setText('O to play')
        self.text.draw(self.win)

        for row in range(3):
            for col in range(3):
                self.ximgs[row, col].undraw()
                self.oimgs[row, col].undraw()
```

```python

        for row in range(3):
            for col in range(3):
                if ttt.board[row, col] == TicTacToe.BLACK:
                    self.ximgs[row, col].draw(self.win)
                elif ttt.board[row, col] == TicTacToe.WHITE:
                    self.oimgs[row, col].draw(self.win)

    def draw(self, ttt):
        self.draw_lines()
        self.draw_ttt(ttt)
        self.win.update()

def main():
    win = GraphWin('TTTDraw', 600, 600, autoflush=False)
    ttt = TicTacToe()
    tttdraw = TTTDraw(win)
    tttinput = TTTInput(win)

    while win.checkKey() != 'Escape':
        tttinput.input(ttt)
        tttdraw.draw(ttt)
        if ttt.isGameOver() != None:
            ttt.reset()
            win.getMouse()
    win.close()

if __name__ == '__main__':
    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：

```python
# -*- 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()
        if mpos == None:
            return False
        moves = ttt.getAllMoves()
        row, col = 4-int(mpos.getY())-1, int(mpos.getX())-1
        if (row, col) not in moves:
            return False
        ttt.play(row, col)
        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：

```python
# -*- coding: utf-8 -*-
"""
```

```
Created on Fri Oct 26 14:41:12 2018

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

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

def Minimax(node, depth):
    result = node.isGameOver()
    if result != None:
        return result, (), depth
    if node.getPlayer() == TicTacToe.BLACK:
        bestValue = -sys.maxsize
        bestMove = ()
        bestDepth = sys.maxsize
        moves = node.getAllMoves()
        for move in moves:
            child = node.clone()
            child.play(*move)
            v, _, leafDepth = Minimax(child, depth+1)
            if bestValue == v and bestDepth > leafDepth:
                bestValue = v
                bestMove = move
                bestDepth = leafDepth
            if bestValue < v:
                bestValue = v
                bestMove = move
        return bestValue, bestMove, bestDepth
    else:
        bestValue = sys.maxsize
        bestMove = ()
        bestDepth = sys.maxsize
        moves = node.getAllMoves()
        for move in moves:
            child = node.clone()
            child.play(*move)
            v, _, leafDepth = Minimax(child, depth+1)
            if bestValue == v and bestDepth > leafDepth:
                bestValue = v
                bestMove = move
                bestDepth = leafDepth
            if bestValue > v:
                bestValue = v
                bestMove = move
        return bestValue, bestMove, bestDepth

def main():
    win = GraphWin('Minimax for TicTacToe', 600, 600, autoflush=False)
    ttt = TicTacToe()
    tttdraw = TTTDraw(win)
    tttinput = TTTInput(win)
    tttdraw.draw(ttt)

    while win.checkKey() != 'Escape':
        if ttt.getPlayer() == TicTacToe.WHITE:
            v, move, _ = Minimax(ttt, 0)
```

```
                if move != ():
                    ttt.play(*move)
            tttinput.input(ttt)
            tttdraw.draw(ttt)
            if ttt.isGameOver() != None:
                time.sleep(1)
                ttt.reset()
                tttdraw.draw(ttt)
                #win.getMouse()
        win.close()

if __name__ == '__main__':
    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 = ()
        bestDepth = sys.maxsize
        moves = node.getAllMoves()
        for move in moves:
            child = node.clone()
            child.play(*move)
            v, _, leafDepth = AlphaBeta(child, depth+1, alpha, beta)
            if bestValue == v and bestDepth > leafDepth:
                bestValue = v
                bestMove = move
                bestDepth = leafDepth
            if bestValue < v:
                bestValue = v
                bestMove = move
            alpha = max(alpha, bestValue)
            if beta <= alpha:
                break #beta pruning
        return bestValue, bestMove, bestDepth
    else:
        bestValue = sys.maxsize
        bestMove = ()
        bestDepth = sys.maxsize
        moves = node.getAllMoves()
        for move in moves:
            child = node.clone()
            child.play(*move)
            v, _, leafDepth = AlphaBeta(child, depth+1, alpha, beta)
            if bestValue == v and bestDepth > leafDepth:
                bestValue = v
                bestMove = move
                bestDepth = leafDepth
            if bestValue > v:
                bestValue = v
                bestMove = move
                beta = min(beta, bestValue)
                if beta <= alpha:
                    break #alpha pruning
        return bestValue, bestMove, bestDepth

def main():
    win = GraphWin('Minimax for TicTacToe', 600, 600, autoflush=False)
    ttt = TicTacToe()
    tttdraw = TTTDraw(win)
    tttinput = TTTInput(win)
    tttdraw.draw(ttt)

    while win.checkKey() != 'Escape':
        if ttt.getPlayer() == TicTacToe.WHITE:
            v, move, _ = AlphaBeta(ttt, 0, -sys.maxsize, sys.maxsize)
            if move != ():
                ttt.play(*move)
        tttinput.input(ttt)
        tttdraw.draw(ttt)
        if ttt.isGameOver() != None:
            time.sleep(1)
            ttt.reset()
            tttdraw.draw(ttt)
            #win.getMouse()
    win.close()

if __name__ == '__main__':
    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:

```python
# -*- 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
        self.visit = 0
        self.win = 0
```

```python

def MCTS(root, nodes_map):
    def Select(node):
        node_key = node.ToString()
        path.append(node_key)
        node_info = nodes_map.get(node_key)
        if node_info == None:
            node_info = NodeInfo()
            node_info.player = node.getPlayer()
            nodes_map[node_key] = node_info

        while node.isGameOver() == None and isFullyExpanded(node):
            node = BestUCTChild(node)
            child_key = node.ToString()
            path.append(child_key)
            child_info = nodes_map.get(child_key)
            if child_info == None:
                child_info = NodeInfo()
                child_info.player = node.getPlayer()
                nodes_map[child_key] = child_info

        return node

    def Expand(node):
        node_key = node.ToString()
        path.append(node_key)
        node_info = nodes_map.get(node_key)
        if node_info == None:
            node_info = NodeInfo()
            node_info.player = node.getPlayer()
            nodes_map[node_key] = node_info

        if node.isGameOver() == None:
            node = RandomChild(node)
            child_key = node.ToString()
            path.append(child_key)
            child_info = nodes_map.get(child_key)
            if child_info == None:
                child_info = NodeInfo()
                child_info.player = node.getPlayer()
                nodes_map[child_key] = child_info
            return node
        else:
            return node

    def Simulate(node):
        result = node.isGameOver()
        while result == None:
            node = RandomChild(node)
            result = node.isGameOver()
        return result

    def Backpropagate(result):
        for node_key in path:
            UpdateStatistics(node_key, result)

    def MaxVisitChild(node):
        max_visit_num = -sys.maxsize
        max_visit_child = ()
        moves = node.getAllMoves()
        for move in moves:
            tmp_node = node.clone()
            tmp_node.play(*move)
            child_info = nodes_map.get(tmp_node.ToString())
            if child_info == None:
                continue
            if max_visit_num < child_info.visit:
                max_visit_num = child_info.visit
                max_visit_child = move
        return max_visit_child

    def isFullyExpanded(node):
        moves = node.getAllMoves()
        for move in moves:
            tmp_node = node.clone()
            tmp_node.play(*move)
            child_info = nodes_map.get(tmp_node.ToString())
            if child_info == None:
                return False
        return True

    def BestUCTChild(node):
        c = 1.4142135623730951
        best_uct = -sys.maxsize
        best_uct_child = None
        node_info = nodes_map[node.ToString()]
        moves = node.getAllMoves()
        for move in moves:
            tmp_node = node.clone()
            tmp_node.play(*move)
            child_key = tmp_node.ToString()
            child_info = nodes_map[child_key]
            ucb1 = child_info.win / child_info.visit + \
```

```python
                   c * math.sqrt(math.log(node_info.visit) /
                    ↪ child_info.visit)
            if best_uct < ucb1:
                best_uct = ucb1
                best_uct_child = move
        if best_uct_child != None:
            node.play(*best_uct_child)
        return node

    def RandomChild(node):
        moves = node.getAllMoves()
        node.play(*moves[randint(0, len(moves)-1)])
        return node

    def UpdateStatistics(node_key, result):
        node_info = nodes_map[node_key]
        node_info.visit += 1
        if node_info.player == TicTacToe.BLACK:
            if result == TicTacToe.WHITEWIN:
                node_info.win += 1
            elif result == TicTacToe.DRAW:
                node_info.win += 0.5
        else:
            if result == TicTacToe.BLACKWIN:
                node_info.win += 1
            elif result == TicTacToe.DRAW:
                node_info.win += 0.5

    decision_time = 500
    for time in range(decision_time):
        node = root.clone()
        path = []
        node = Select(node)
        simulation_node = Expand(node)
        simulation_result = Simulate(simulation_node)
        Backpropagate(simulation_result)
    return MaxVisitChild(root)

def main():
    seed()
    win = GraphWin('MCTS for TicTacToe', 600, 600, autoflush=False)
    ttt = TicTacToe()
    tttdraw = TTTDraw(win)
    tttinput = TTTInput(win)
    tttdraw.draw(ttt)

    nodes_map = {}
    while win.checkKey() != 'Escape':
        if ttt.getPlayer() == TicTacToe.WHITE:
            move = MCTS(ttt, nodes_map)
            if move != ():
                ttt.play(*move)
        tttinput.input(ttt)
        tttdraw.draw(ttt)
        if ttt.isGameOver() != None:
            time.sleep(1)
            ttt.reset()
            tttdraw.draw(ttt)
            #win.getMouse()
    win.close()

if __name__ == '__main__':
    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：

```python
# -*- 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
"""

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

```python

def TD0(root, V, alpha, epsilon, learning_time):
    def RandomMove(node):
        moves = node.getAllMoves()
        return moves[randint(0, len(moves)-1)]

    def UpdateTerminalNode(node, result):
        key = node.ToString()
        if V.get(key) != None:
            return
        if result == TicTacToe.BLACKWIN:
            V[key] = 1
        elif result == TicTacToe.WHITEWIN:
            V[key] = 0
        else:
            V[key] = 0.5

    def UpdateValueFunction(node1, node2, alpha):
        key1 = node1.ToString()
        key2 = node2.ToString()
        if V.get(key1) == None:
            V[key1] = 0.5
        if V.get(key2) == None:
            V[key2] = 0.5
        V[key1] += alpha * (V[key2] - V[key1])

    def BestMove(node):
        if node.getPlayer() == TicTacToe.BLACK:
            best_value = -sys.maxsize
            best_move = ()
            moves = node.getAllMoves()
            for move in moves:
                tmp_node = node.clone()
                tmp_node.play(*move)
                key = tmp_node.ToString()
                if V.get(key) == None:
                    continue
                if best_value < V[key]:
                    best_value = V[key]
                    best_move = move
            if best_move == () and len(moves) != 0:
                best_move = RandomMove(node)
            return best_move
        else:
            best_value = sys.maxsize
            best_move = ()
            moves = node.getAllMoves()
            for move in moves:
                tmp_node = node.clone()
                tmp_node.play(*move)
                key = tmp_node.ToString()
                if V.get(key) == None:
                    continue
                if best_value > V[key]:
                    best_value = V[key]
                    best_move = move
            if best_move == () and len(moves) != 0:
                best_move = RandomMove(node)
            return best_move

    for i in range(learning_time):
        node = root.clone()
        parent = None
        result = node.isGameOver()
        while result == None:
            if random() < epsilon:
                move = RandomMove(node)
                node.play(*move)
                result = node.isGameOver()
                if result != None:
                    UpdateTerminalNode(node, result)
                    if parent != None:
                        UpdateValueFunction(parent, node, alpha)
                    parent = None
                else:
                    parent = node.clone()
            else:
                move = BestMove(node)
                node.play(*move)
                result = node.isGameOver()
                if result != None:
                    UpdateTerminalNode(node, result)
                if parent != None:
                    UpdateValueFunction(parent, node, alpha)
                if result != None:
                    parent = None
                else:
                    parent = node.clone()
            if result == None:
                move = RandomMove(node)
                node.play(*move)
                result = node.isGameOver()
        UpdateTerminalNode(node, result)
```

```python
        if parent != None:
            UpdateValueFunction(parent, node, alpha)
    return BestMove(root)

def main():
    seed()
    win = GraphWin('TD-Learning(0) for TicTacToe', 600, 600,
    ↪   autoflush=False)
    ttt = TicTacToe()
    tttdraw = TTTDraw(win)
    tttinput = TTTInput(win)
    tttdraw.draw(ttt)

    try:
        vfile = open('ValueFunction.dat', 'rb')
        V = pickle.load(vfile)
        vfile.close()
    except FileNotFoundError:
        V = {}

        #Start to self-play
        self_play = 100
        for i in range(self_play):
            tmp_root = ttt.clone()
            tttdraw.draw(tmp_root)
            result = tmp_root.isGameOver()
            while result == None:
                move = TD0(tmp_root, V, 0.5, 0.1, 2000)
                #moves = tmp_root.getAllMoves()
                #move = moves[randint(0, len(moves)-1)]
                if move != ():
                    tmp_root.play(*move)
                tttdraw.draw(tmp_root)
                result = tmp_root.isGameOver()
                if result == None:
                    move = TD0(tmp_root, V, 0.5, 0.1, 2000)
                    if move != ():
                        tmp_root.play(*move)
                tttdraw.draw(tmp_root)
                result = tmp_root.isGameOver()
                if result != None:
                    time.sleep(0.5)
        #Save V to file
        vfile = open('ValueFunction.dat', 'wb')
        pickle.dump(V, vfile)
        vfile.close()

    while win.checkKey() != 'Escape':
        if ttt.getPlayer() == TicTacToe.WHITE:
            move = TD0(ttt, V, 0.2, 0.1, 500)
            if move != ():
                ttt.play(*move)
        tttinput.input(ttt)
        tttdraw.draw(ttt)
        if ttt.isGameOver() != None:
            time.sleep(1)
            ttt.reset()
            tttdraw.draw(ttt)
            #win.getMouse()
    win.close()

if __name__ == '__main__':
    main()
```

10 遗传算法

TicTacToe 的遗传算法如下：

```
Input:
    None
Output:
    the best solution
def GA():
    generation_num = 500
    population_num = 500
    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
```

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.