Johdatus tekoälyyn, syksy 2011, Teemu Roos **Harjoitus 2**

Huom: Voit saada tästä harjoituskerrasta max. 8 pistettä.

Tehtävä 1. Minimax ja alpha-beta. (1 piste)

a ja b) Ks. kuva 1 alla.

c) Katso video kurssin sivulla.

Muista, että α-arvon voi tulkita MIN-solmun yläpuolelle olevan MAX-solmun saamaksi toistaiseksi parhaaksi ”tarjoukseksi”. Jos MIN-solmussa havaitaan, että solmun arvoksi on tulossa α-arvoa pienempi luku, voidaan lopettaa, koska yläpuolella olevasta MAX-solmussa ei koskaan pelata siirtoa, joka päätyisi nykyiseen MIN-solmuun. Myös β arvolla on vastaava tulkinta.

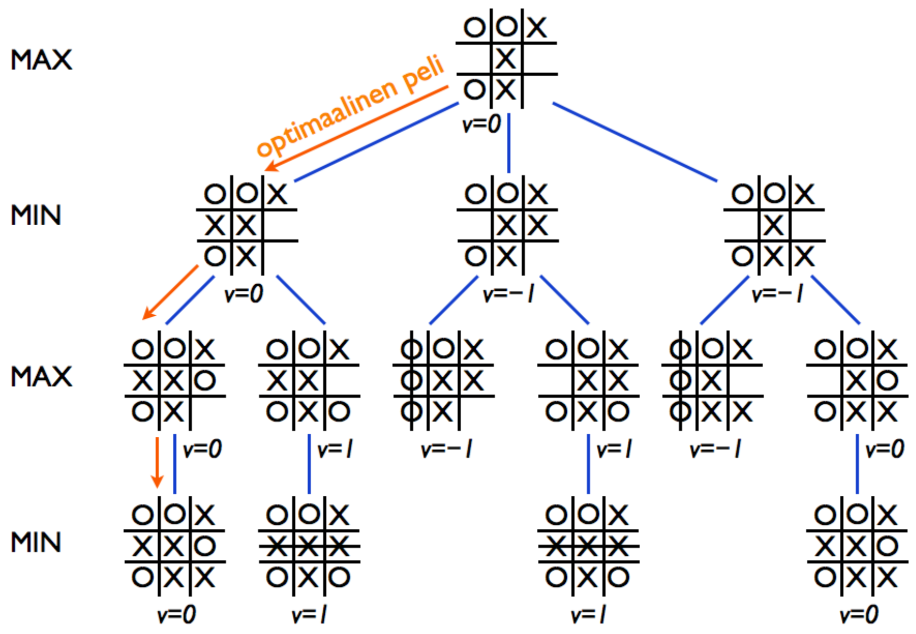

Figure 1: Tehtävän 1 pelipuu. Minimax-algoritmin tuottamat arvot on merkitty kunkin solmun alle ($v = -1/0/1$).

Tehtävä 2. Syvyysrajoitettu alpha-beta. (1-2 pistettä)

```
#!/usr/bin/python

from array import array
import random

Empty = ' '

maxDepth = 10

def checkWin(board):
    col1 = board.b[0][0] + board.b[1][0] + board.b[2][0]
    col2 = board.b[0][1] + board.b[1][1] + board.b[2][1]
    col3 = board.b[0][2] + board.b[1][2] + board.b[2][2]
    row1 = board.b[0][0] + board.b[0][1] + board.b[0][2]
    row2 = board.b[1][0] + board.b[1][1] + board.b[1][2]
    row3 = board.b[2][0] + board.b[2][1] + board.b[2][2]
    diag1 = board.b[0][0] + board.b[1][1] + board.b[2][2]
    diag2 = board.b[2][0] + board.b[1][1] + board.b[0][2]

    if ("XXX" in [row1,row2,row3,col1,col2,col3,diag1,diag2]):
        return 'X'
    elif ("OOO" in [row1,row2,row3,col1,col2,col3,diag1,diag2]):
        return 'O'
    else:
        return None

def applyMove(board,r,c,turn):
    # returns a new position with 'turn'  (X or O) added at (r,c)
    newBoard = Board(board)
    newBoard.b[r][c] = turn
    newBoard.winner = checkWin(newBoard)
    newBoard.turn = 'X' if board.turn == 'O' else 'O'
    return newBoard

def copyPieces(b):
    return [array('c',row) for row in b]

class Board:

    def __init__(self,board=None):
        self.value = 0
        self.winner = None
        if board:
            self.b = copyPieces(board.b)
            self.turn = board.turn
            self.value = board.value
            self.winner = board.winner

    def initialState(self):
        b=['OOX', ' X ', 'OX ']
#        b=['   ', '   ', '   ']
        self.b = [array('c',row) for row in b]
        self.turn = 'X'
        self.winner = None

    def __str__(self):
        return "\n".join([row.tostring() for row in self.b])+"\n"

    def validMoves(self):
        if not self.winner == None:
            return []
        moves = []

        for (r,c) in [(r,c) for r in range(3) for c in range(3)]:
            if self.b[r][c] == ' ':
                moves.append(applyMove(self,r,c,self.turn))

        return moves

    def evaluate(self):
        if self.turn == 'X':
            self.value = self.maxValue(-float('inf'), float('inf'), 1)
        else:
            self.value = self.minValue(-float('inf'), float('inf'), 1)

    def maxValue(self, alpha, beta, depth):
        if self.winner == 'X': return 1
        elif self.winner == 'O':return -1
        if depth >= maxDepth: return self.evaluateHeuristic()
        v = -float('inf')
        moves = self.validMoves()
        if moves == []: return 0
        for move in moves:
            move.value = move.minValue(alpha,beta,depth+1)
            v = max(v, move.value)
            if v >= beta: return v
            alpha = max(alpha, v)
        return v

    def minValue(self, alpha, beta, depth):
        if self.winner == 'X': return 1
        elif self.winner == 'O': return -1
        if depth >= maxDepth: return self.evaluateHeuristic()
        moves = self.validMoves()
        if moves == []: return 0
        v = float('inf')
        for move in moves:
            move.value = move.maxValue(alpha,beta,depth+1)
            v = min(v, move.value)
            if v <= alpha: return v
            beta = min(beta, v)
        return v

    def evaluateHeuristic(self):
        if self.winner == 'X': return 1
        elif self.winner == 'O': return -1

        col1 = self.b[0][0] + self.b[1][0] + self.b[2][0]
        col2 = self.b[0][1] + self.b[1][1] + self.b[2][1]
        col3 = self.b[0][2] + self.b[1][2] + self.b[2][2]
        row1 = self.b[0][0] + self.b[0][1] + self.b[0][2]
        row2 = self.b[1][0] + self.b[1][1] + self.b[1][2]
        row3 = self.b[2][0] + self.b[2][1] + self.b[2][2]
        diag1 = self.b[0][0] + self.b[1][1] + self.b[2][2]
        diag2 = self.b[2][0] + self.b[1][1] + self.b[0][2]
        all = [col1,col2,col3,row1,row2,row3,diag1,diag2]

        # count how many lines there are where X has one or two
        # pieces and O none, and vice versa, take the difference
        # and divide by the total number of lines (=8)

        heuristic = (len([x for x in all if x in
                          ['XX ','X X',' XX','X  ',' X  ','X  ']]) \
                         -len([x for x in all if x in
                               ['OO ','O O',' OO','O  ',' O ','  O']]))\
                         / 8.
        return heuristic

board = Board()

board.initialState()
print board

while True:

    print (board.turn + ' moves')
    bestMove = None
    bestValue = float('-Inf') if board.turn == 'X' else float('Inf')
    for move in board.validMoves():
        move.evaluate()
        if (board.turn == 'X' and move.value > bestValue) or \
                (board.turn == 'O' and move.value < bestValue):
            bestMove = move
            bestValue = move.value

    if bestMove == None:
        print 'No moves!'
        break
    board = bestMove
    print board
    if not board.winner == None:
        print (board.winner + ' wins!')
        break

if board.winner == None:
    print 'A draw!'
```

Esimerkkiajo:

```
$ ./tictactoe.py
OOX
 X
OX

X moves
OOX
XX
OX

O moves
OOX
XXO
OX

X moves
OOX
XXO
OXX

O moves
No moves!
A draw!
```

Tehtävä 3. Reittiopas. (3-6 pistettä)

Java-esimerkki (kiitokset Kalle Viirille):

`Node.java.`

```
package reittiopas;

import java.io.PrintStream;
import java.util.List;
import java.util.ArrayList;

/**
 * @author Kalle Viiri
 */
class Node implements Comparable {
    private int time;   //minutes from start
    private Stop location;      //stop we're at
    private Node parent;        //We came from where?
    private Transportation trans;
    public static Stop goal;


    public Node(int time, Stop location, Node parent, Transportation trans) {
        this.time = time;
        this.location = location;
        this.parent = parent;
        this.trans = trans;

    }

    public boolean isGoalState(int number) {
        if(number == location.number) return true;
        else return false;
    }



    public List<Node> getNeighbors(List<Stop> stopList) {
        List<Node> neighbors = new ArrayList<Node>();
        Stop currentLocation = this.getLocation();
        for(Transportation t : currentLocation.transportations) {
            //Finding the right stop and the time associated
            int wait = this.getTime() % 10;
            int stopIndex = 999;        //We want to know the stop index
            for(int i = 0; i < t.timeTable.size(); i++) {
                if(t.timeTable.get(i)[0] == currentLocation.number) {
                    stopIndex = i;
                    wait = t.timeTable.get(i)[1] % 10 - wait;
                    break;
                }
            }
            if(wait < 0)
                wait+= 10;
            if(stopIndex + 1 < t.timeTable.size()) {    //If this isn’t the last stop
                for(Stop s : stopList) {
                    if(s.number == t.timeTable.get(stopIndex + 1)[0]) {
                        //Find the next stop
                        int tripCost = t.timeTable.get(stopIndex + 1)[1]
                              - t.timeTable.get(stopIndex)[1];
                        neighbors.add(new Node(this.getTime() + wait
                              + tripCost, s, this, t));
                    }
                }
            }
        }
        return neighbors;
    }

    public void printState(PrintStream ps) {
        ps.print(location.name + " " + (trans.name != null ? trans.name : "")
               + " Aika: " + time + "\n\n");
    }

    public int calculateDistance(Stop target) {
        return (int) Math.sqrt(Math.pow(Math.abs((this.getLocation().x - target.x)), 2)
               + Math.pow(Math.abs((this.getLocation().y - target.y)), 2));
    }

    public int estimateCost(Stop goal) {
        return calculateDistance(goal) / 1000;
    }

    public int totalCost(Stop goal) {
        return time + estimateCost(goal);
    }

    public Stop getLocation() {
        return location;
    }

    public void setLocation(Stop location) {
        this.location = location;
    }

    public Node getParent() {
        return parent;
    }

    public void setParent(Node parent) {
        this.parent = parent;
    }

    public int getTime() {
        return time;
    }

    public void setTime(int time) {
        this.time = time;
    }

    @Override
    public int compareTo(Object t) {
        Node n = (Node) t;
        if(this.estimateCost(goal) + getTime() < n.estimateCost(goal) + n.getTime())
            return -1;
        else if(this.estimateCost(goal) + getTime() > n.estimateCost(goal) + n.getTime())
            return 1;
        else return 0;
    }
}
```

AStarReittiopas.java.

```
package reittiopas;

import java.util.LinkedList;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Vector;
import reittiopas.Stop;

/**
 * @author Kalle Viiri
 */
public class AStarReittiopas {
    public static int goalNodeNumber = 2;
    public static int startNodeNumber = 1;
    public static int startTime = 1;

    public static void main(String[] args) {
        PriorityQueue<Node> pq = new PriorityQueue<Node>();
        List<Node> alreadyVisited = new LinkedList<Node>();
        FileOperations.readDefaultFiles();
        Vector<Stop> stops = FileOperations.stops;
        Node goalNode = null;
        boolean goalFound = false;

        Node n = null;
        for(Stop s : stops) {
            if(s.number == startNodeNumber) {
                n = new Node(startTime, s, null, null);
            }
            else if (s.number == goalNodeNumber) {
                Node.goal = s;
            }
        }

        pq.add(n);
        while(!pq.isEmpty() && !goalFound) {
            alreadyVisited.add(n);
            for (Node newNode : pq.poll().getNeighbors(stops)) {
                if(newNode.isGoalState(goalNodeNumber)) {
                    goalFound = true;
                    goalNode = newNode;
                    break;
                }
                if(!pq.contains(newNode) && !alreadyVisited.contains(newNode)) {
                    pq.add(newNode);
                }
            }
        }
        int totalTripTime = goalNode.getTime() - startTime;
        while(goalFound && goalNode.getParent() != null) {
            goalNode.printState(System.out);
            goalNode = goalNode.getParent();
        }
        System.out.println(n.getLocation().name + " Aika: " + startTime + "\n");
        System.out.println("Total trip time: " + totalTripTime + " minutes.");
    }
}
```