CSCI 561: Foundations of Artificial Intelligence
Instructor: Prof. Laurent Itti
Homework #4: Game Playing
Due on May 1 at 11:59pm, Los Angeles time, 2013

Suppose Alice is playing English Draughts (or Checkers) against Bill. Given a current board configuration, you will help Alice determine the next move that will maximize her chance of winning. You will use minimax algorithm with alpha-beta pruning to calculate the optimal move.

English Draughts: http://en.wikipedia.org/wiki/English_draughts

It is played by two opponents, alternating moves on opposite sides of an 8×8 square board. The pieces are traditionally black, red (or white). The pieces move diagonally and capture enemy pieces by jumping over them. The pieces may only move forward until they reach the opposite end of the board, when they are crowned (become "kings") and may thereafter move and capture both backward and forward. A player wins by capturing all of the opponent's pieces or by leaving the opponent with no legal move. The game ends in a draw if neither side can force a win. Specifically, there are two different moves in English draughts:

1. A **simple move** consists of sliding a piece one square diagonally to an adjacent unoccupied dark square. Uncrowned pieces may move only diagonally forward; kings may move in any diagonal direction.
2. A **jump** is a move from a square diagonally adjacent to an opponent's piece to an empty square immediately beyond it, in the same line. (Thus, **jumping over** the square containing the opponent's piece.) Uncrowned pieces may jump only diagonally forward. Kings may jump in any diagonal direction. A jumped piece is considered "captured" and removed from the game. Any piece, whether crowned or not, may jump a king. Jumping is always **mandatory**: if a player has the option to jump, he must take it, even if doing so results in disadvantage for the jumping player. For example, a single jump might set up a player such that the opponent has a multi-jump move in reply. If several jumping options are available, the player should choose which piece to jump and what direction to jump. In a real game, **Multiple jumps** are possible, if after one jump, another piece is immediately eligible to be jumped even if that jump is in a different diagonal direction. **For simplicity, however, we will only allow single jumps in this assignment, and therefore multiple jumps are illegal.**

Programming Task

Alice and Bill are playing the game, and it is Alice's turn to take next move. You will write a program to determine Alice's next move that is optimal. Your program will take the current board configuration as input. For example (case 5 in input.txt):

```
+B+B+B+B
O+O+O+O+
+O+O+O+O
O+O+O+O+
+B+B+K+O
O+A+O+O+
+O+O+O+O
A+A+O+A+
```

The interpretations are

- The configuration is denoted as a 8×8 matrix, where `(0,0)` denotes the top left corner of the board, and `(7,7)` denotes the bottom right corner.
- Alice's pieces move upwards, and Bill's pieces move downwards.
- 'O' denotes an empty dark square where a piece can be placed.
- '+' denotes a light-shaded square where **no** piece can be placed.
- 'A' denotes Alice's piece.
- 'B' denotes Bill's piece.
- 'k' denotes Alice's king piece.
- 'K' denotes Bill's king piece

Your program will use **minimax** algorithm with **alpha-beta pruning** to determine Alice's next move that yields the best utility for her. **For simplicity, you do not need to search all possible states until the end of the game. In this assignment, you only need to consider all possible states for the next 4 moves (2 alternating moves for each player).**

We use heuristic values to measure the utility of each end state after the next 4 moves. The heuristic value is defined as the number of Alice's pieces left in the board subtracted by the number of player Bill's pieces. Because kings have more powerful ability than uncrowned pieces, our heuristic counts each king as **two** normal pieces. If Alice wins the game in a configuration, the heuristic value is $+\infty$ and $-\infty$ if Alice loses. You will help Alice choose a move that maximizes this heuristic value. If there are several moves that result in the same most utility, just output one of them. We assume Bill is rational and always choose a move that minimizes this heuristic value.

The output of your program will be the next optimal move for Alice. It is achieved by traversing all possible search states for the next 4 moves using minimax algorithm with alpha-beta pruning. If the game will ends in less than 4 moves, then just output the next few moves before the game end. Below is a sample output, but it is ***not*** necessarily correct:

```
A1: (5,4)=>(3,2).
|-B2: (2,1)=>(4,3).
|-|-A3: (6,7)=>(5,8).
|-|-|-B4: (4,7)=>(5,6); h=0.
|-|-|-B4: (4,5)=>(5,6); h=3.
|-|-A3: (6,5)=>(4,6)
|-|-|-B4: (2,3)=>(4,1); h=5.
|-|-A3: pruning (7,4)=>(6,5), (7,6)=>(6,5); alpha=1, beta=1.
|-B2: (2,3)=>(4,5)
A1: (6,7)=>(5,8).
...
first move: (5,4)=>(3,2).
```

The interpretations are:

1. The output will be the traversal of possible search states, but not the optimal search path. Each move could be followed by several possible moves, we use "`|-`" to represent the traversal relationships of the two consecutive moves.
2. Each line is in the format of "`Pi: (x1,y1)=>(x2,y2).`", where `P` could be `A` (Alice) or `B` (Bill) , `i=1, 2, 3` or `4` denoting the following `i`th move, `(x1,y1)` is the initial position of the piece to be moved and **(x2,y2)** points to its destination.
3. If it is the `4`th move, output heuristic value after (e.g., `B4: (2,3)=>(4,1); h=5.)`.
4. If the move involves alpha-beta pruning, please output all branch moves that should be pruned and the values of alpha and beta, e.g., `pruning (7,4)=>(6,5), (7,6)=>(6,5); alpha=1, beta=1.`
5. After traversing all possible states, output the answer of Alice's first move, e.g., `first move: (5,4)=>(3,2).`

Suggestions:

Your program should first check if there exists a jumping move starting from the current configuration since jumping is mandatory. If there is none, then expand the simple moves. Your program should also do alpha-beta pruning while expanding the search to reduce the complexity.

Grading:

The grader will compile your program using the following command, for examples:

```
C++:    g++ *.cpp -o hw4
Java:   javac *.java
```

And execute your program using, for examples for each experiment:

```
C++:    ./hw4 input.txt output.txt
Java:   java hw4 input.txt output.txt
```

Your program will be evaluated based on 6 given test cases in input.txt and 3 unseen test cases. Please check boundary situations when implementing your program. Each test case in input.txt begins with a line case i (i = 1, 2, … 5) and continues with the current configuration. You need to generate output.txt that results from input.txt using your program. The output of each test case will also begin with case i, and continue with the traversal of states and the answer of Alice's first move.

Deliverables:

You are required to hand in all code that you wrote to complete this assignment; implementation language not important. However, if you code in C++, C#, or Java, the TA will be better able to assist you ☺. In addition, you also need to hand in output.txt which results from input.txt using your program. Please include a readme.txt to write any comments you may have.

The deadline for this assignment is **Wednesday May 1, at 11:59pm** Los Angeles time. Please turn in all materials as a .zip file via Blackboard, with the title format [firstname]_[lastname]_HW4.zip (e.g., Harry_Potter_HW4.zip).