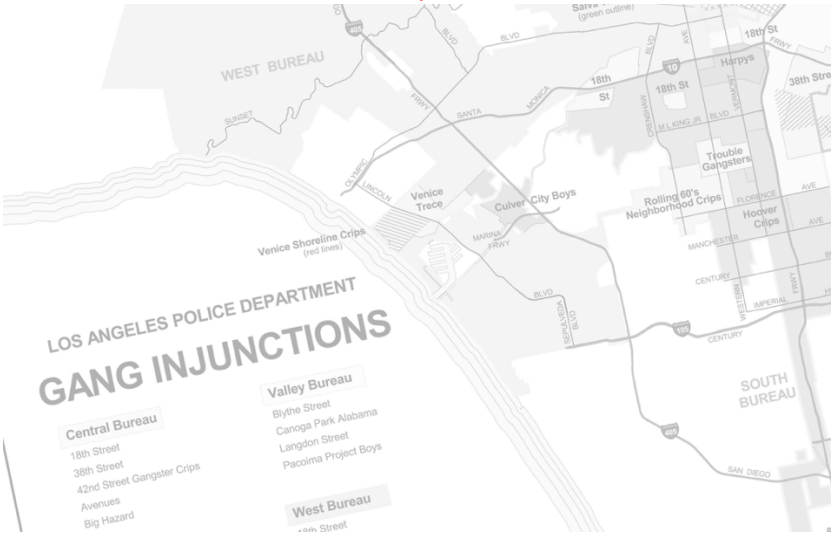
CSCI-561 - Fall 2016 - Foundations of Artificial Intelligence
Homework 2

Due October 17, 2016 23:59:59



Guidelines

This is a programming assignment. You will be provided sample inputs and outputs (see below). Please understand that the goal of the samples is to check that you can correctly parse the problem definitions, and generate a correctly formatted output. The samples are very simple and it should not be assumed that if your program works on the samples it will work on all test cases. There will be more complex test cases and it is your task to make sure that your program will work correctly on any valid input. You are encouraged to try your own test cases to check how your program would behave in some complex special case that you might think of. Since **each homework is checked via an automated A.I. script**, your output should match the example format *exactly*. Failure to do so will most certainly cost some points. The output format is simple and examples are provided. You should upload and test your code on vocareum.com, and you will submit it there. You may use any of the programming languages provided by vocareum.com.

Grading

Your code will be tested as follows: Your program should take no command-line arguments. It should read a text file called "input.txt" in the current directory that contains a problem definition. It should write a file "output.txt" with your solution. Format for files input.txt and output.txt is specified below. End-of-line convention is Unix (since vocareum is a Unix system).

The grading A.I. script will, 50 times:

- Create an input.txt file, delete any old output.txt file.
- Run your code.
- Compare output.txt created by your program with the correct one.
- If your outputs for all 50 test cases are correct, you get 100 points.
- If one or more test case fails, you get 50 – N points where N is the number of failed test cases.

Note that if your code does not compile, or somehow fails to load and parse input.txt, or writes an incorrectly formatted output.txt, or no output.txt at all, or OuTpUt.TxT, **you will get zero points**. Please test your program with the provided sample files to avoid this. You can submit code as many times as you wish on vocareum, and the last submitted version will be used for grading.

Academic Honesty and Integrity

All homework material is checked vigorously for dishonesty using several methods. All detected violations of academic honesty are forwarded to the Office of Student Judicial Affairs. To be safe you are urged to err on the side of caution. Do not copy work from another student or off the web. Keep in mind that sanctions for dishonesty are reflected in *your permanent record* and can negatively impact your future success. As a general guide:

Do not copy code or written material from another student. Even single lines of code should not be copied.
Do not collaborate on this assignment. The assignment is to be solved individually.
Do not copy code off the web. This is easier to detect than you may think.
Do not share any custom test cases you may create to check your program's behavior in more complex scenarios than the simplistic ones considered below.
Do not copy code from past students. We keep copies of past work to check for this.
Do ask the professor or TA if you are unsure about whether certain actions constitute dishonesty. It is better to be safe than sorry.

Project description

Once upon a time, Los Angeles was a Never Never Land. People were blissfully happy, Conan O'Brien was still on late-night, and no one knew who "the Real Housewives of New Jersey" were. Until the day that two violent, money-thirsty gangs decided to put their dirty hands onto Los Angeles.

One gang was from the north, and the other gang was from the south. Both wanted to take over the entire city. Therefore, war was unavoidable. To win the war, the leader of the south gang knew that he needed not mere foot troops, but admirals. Therefore, he built an institute to train his future admirals. He named his training institute CSCI-561, and asked his recruits to create artificial agents to engage in a combat simulation, which he called a 'game', meant to imitate the upcoming gang war.
(Once the members are finished, the leader will classify their research and pull them from the institute, leaving them fractured and bitter human beings. However, as they do not know this yet, they are full of excitement and zeal for the project.)

The game is a simulation of ground warfare and has the following rules:

1. The game board is an NxN grid representing the territory your forces will trample (N=5 in the figures below). Columns are named A, B, C, … starting from the left, and rows are named 1, 2, 3, … from top.

2. Each player takes turns as in chess or tic-tac-toe. That is, player X takes a move, then player O, then back to player X, and so forth.

3. Each square has a fixed point value between 1 and 99, based upon its computed strategic and resource value.

4. The object of the game for each player is to score the most points, where score is the sum of all point values of all his or her occupied squares minus the sum of all points in the squares occupied by the other player. Thus, one wants to capture the squares worth the most points while preventing the other player to do so.
5. The game ends when all the squares are occupied by the players since no more moves are left.
6. Players cannot pass their move, i.e., they must make a valid move if one exists (game is not over).
7. Movement and adjacency relations are always vertical and horizontal but never diagonal.
8. The values of the squares can be changed for each game, but remain constant within a game.
9. **Game score** is computed as the difference between (a) the sum of the values of all squares occupied by your player and (b) the sum of the values of all squares occupied by the other player. This applies both to terminal (game over, terminal utility function) and non-terminal states (evaluation function). This is required to ensure that your program produces the same results as the grading script.
10. On each turn, a player can make one of two moves:

Stake – You can take any open space on the board. This will create a new piece on the board. This move can be made as many times as one wants to during the game, but only once per turn. However, a Stake *cannot* conquer any pieces. It simply allows one to arbitrarily place a piece anywhere unoccupied on the board. Once you have done a Stake, your turn is complete.

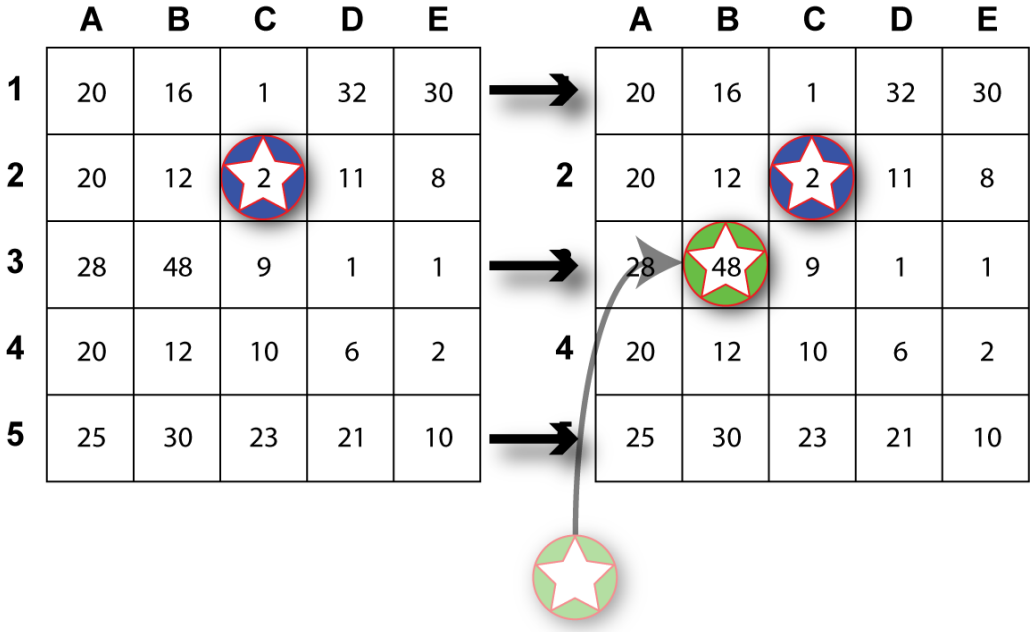


Figure 1. This is a Stake. In this example, green drops a new piece on square [B,3]**.** This square is worth 48, which is a higher number, meaning that it contains some juicy oil wells or other important resources. After that, the score is green 48 : blue 2, i.e., GameScore = 46 for green. A Stake could have been carried out on any squares except for [C,2] since blue already occupies it.

Raid – From any space you occupy on the board, you can take the one next to it (up, down, left, right, but not diagonally) if it is unoccupied. The space you originally held is still occupied. Thus, you get to create a new piece in the raided square. Any enemy touching the square you have taken is conquered and that square is turned to your side (you turn its piece to your side). A Raid can be done even if it will not conquer another piece. Once you have made this move, your turn is over.

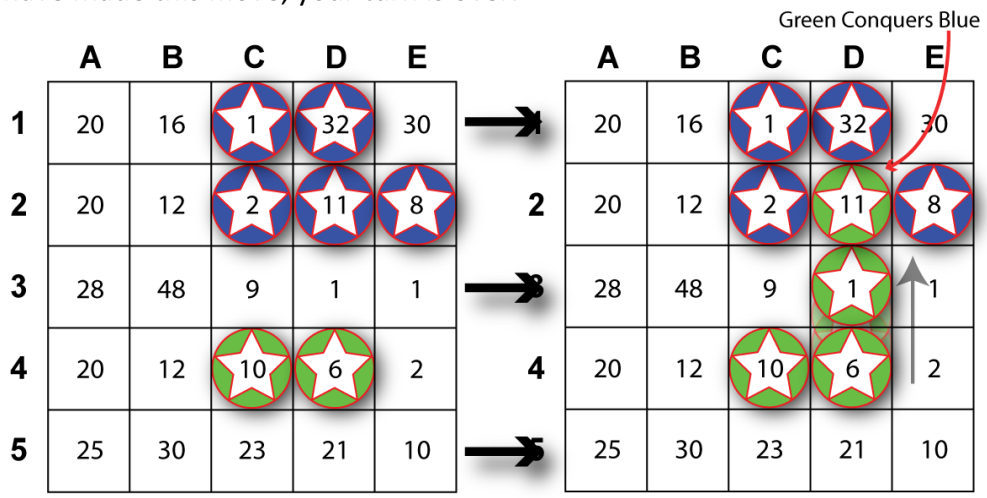


Figure 2. This is a Raid. Green raids the piece in [D,4] to [D,3]. This conquers the blue piece in [D,2] since it is touching the new green piece in [D,3]. A raid always creates a new piece adjacent to an existing one, but it does not conquer another piece unless it is touching it. Thus, another valid move might have been for [D,4] to have raided [E,4]. Then the green player would own [D,4] and [E,4] but would have conquered none of blue's pieces. Note, the score before the raid was green 16 : blue 54, i.e., GameScore = -38 for green, and afterwards is green 28 : blue 43, i.e., GameScore = -15 for green.

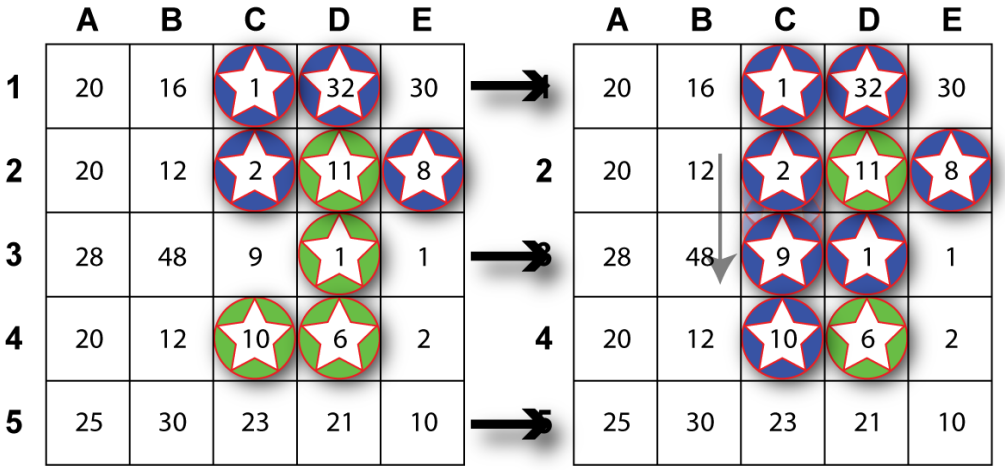


Figure 3. Here blue raids [C,3] from [C,2]. In the process green's pieces at [D,3] and [C,4] are conquered since they touch [C,3]. Notice that in its next move, green will not be able to conquer any of blue's pieces and only the piece at [D,4] would be able to execute a Raid since [D,2] has no neighboring unoccupied squares.

Your assignment is:

Base homework (required): Create a program to play the game, using, depending on specification in the input file, either the plain **Minimax** algorithm with depth limit, or **Alpha-Beta Pruning**.

Competition mode (optional): Your algorithm will play against the other students' algorithms in a tournament, and the evaluation function is now up to you. You can use any algorithm, heuristic, and trick you wish. Your CPU time will be limited, however. Please see the additional instructions for the competition mode in a separate file.

Format for input.txt:

```
<N>
<MODE>
<YOUPLAY>
<DEPTH>
<… CELL VALUES …>
<… BOARD STATE …>
```

where

<N> is the board width and height, e.g., N=5 for the 5x5 board shown in the figures above. N is an integer strictly greater than 0 and smaller than or equal to 26.

<MODE> is "MINIMAX" or "ALPHABETA" or "COMPETITION".

<YOUPLAY> is either "X" or "O" and is the player which you will play on this turn.

<DEPTH> is the depth of your search. By convention, the root of the search tree is at depth 0. DEPTH will always be larger than or equal to 1.

<… CELL VALUES …> contains N lines with, in each line, N positive integer numbers each separated by a single space. These numbers represent the value of each location.

<… BOARD STATE …> contains N lines, each with N characters "X" or "O" or "." to represent the state of each cell as occupied by X, occupied by O, or free.

Format for output.txt:

```
<MOVE> <MOVETYPE>
<… NEXT BOARD STATE …>
```

where

<MOVE> is your move. As in the figures above, we use capital letters for column and numbers for rows. An example move is "F22" (remember that N is from 1 to 26, see above).

<MOVETYPE> is “Stake” or “Raid” and is the type of move that your <MOVE> is.
<… NEXT BOARD STATE …> a description of the new board state after you have played your move. Same format as <… BOARD STATE …> in input.txt above.

Notes and hints:

- Please name your program “**homework.xxx**” where ‘xxx’ is the extension for the programming language you choose. (“**py**” for python, “**cpp**” for C++, and “**java**” for Java). If you are using C++11, then the name of your file should be “**homework11.cpp**” and if you are using python3.4 then the name of your file should be “**homework3.py**”.
- We will guarantee that at least one valid move exists for your player for any given input.txt file.
- Remember to use Game Score as defined above, both for the terminal utility computation, and for the evaluation function of non-terminal nodes.

Pseudo code:
To ensure that your outputs will match those of the grading program, please use the following pseudo code to program your algorithm:

Minimax: AIMA Figure 5.3 (Minimax without cut-off) and section 5.4.2 (Explanation of Cutting off search)

Alpha-Beta: AIMA Figure 5.3 (Alpha-Beta without cut-off) and section 5.4.2 (Explanation of Cutting off search)

Example 1:

For this input.txt:

```
3
MINIMAX
O
2
1 8 23
5 42 12
26 30 9
X..
...
...
```

your output.txt should be:

```
B3 Stake
X..
...
.O.
```

Example 2: using the board state from Figure 3 (left) as the starting configuration for input.txt with minimax search depth = 1,

```
5
MINIMAX
X
1
20 16 1 32 30
20 12 2 11 8
28 48 9 1 1
20 12 10 6 2
25 30 23 21 10
..XX.
..XOX
...O.
..OO.
.....
```

The output produced is:

```
B3 Stake
..XX.
..XOX
.X.O.
..OO.
.....
```

Which is the correct move given the specified algorithm and search depth, but is not the same move as shown in Figure 3 (right). With search depth of 1, the algorithm just went for the high-value cell in B3.

Example 3: changing the search parameters so as to use alpha beta pruning with search depth=4 on the otherwise same input.txt as in example 2 results in the following output.txt:

```
C3 Raid
..XX.
..XOX
..XX.
..XO.
.....
```

which matches the solution depicted in Figure 3 (right).