CSCI 561: Foundations of Artificial Intelligence
Spring 2016, Homework #1
Due on Feb 8, 2016 at 11:59 PM PST

Question

The campus of USC is home to two large families of squirrels, the Leavey Ninja Squirrels from the north, and the Viterbi Fluffy Hackers from the west. They are constantly battling for territories with the highest yield of pine nuts (or whatever nuts they desire). The family patriarch of Viterbi Fluffy Hackers, Master Yoda, after "attending" many years of CSCI-561 outside the classroom window, finally realized this is the key to the glory of his family. He requested you, the Chosen One (who apparently knows how to communicate with a squirrel), to be his strategy advisor. As a member of greater Viterbi family, you feel obliged to help your fluffy brethren, using the power of the Force (a.k.a. Artificial Intelligence).

As the wise Master Yoda has told you, the squirrel war on USC campus can be simulated as a game with the following rules:

1. The game board is a 5x5 grid representing territories the squirrel warriors will trample.

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

3. Each square has a fixed point value between 1 and 99, based upon its yield of nuts.

4. The values of the squares can be changed for each game, but remain constant within a game.

5. The objective of the game for each player is to score the most points, i.e. the total point value of all his or her occupied squares. Thus, one wants to capture the squares worth the most points.

6. The game ends when all the squares are occupied, because no more moves are left.

7. On each turn, a player can make one of two moves:

Raid – You can take over any unoccupied square that is adjacent to one of your current pieces (horizontally or vertically, but not diagonally). You place a new piece in the taken over square. Also, any enemy pieces adjacent to your new piece (horizontally or vertically, but not diagonally) are conquered and replaced by your own pieces. You can Raid a square even if there are no enemy pieces adjacent to it to be conquered. Once you have made this move, your turn is over.
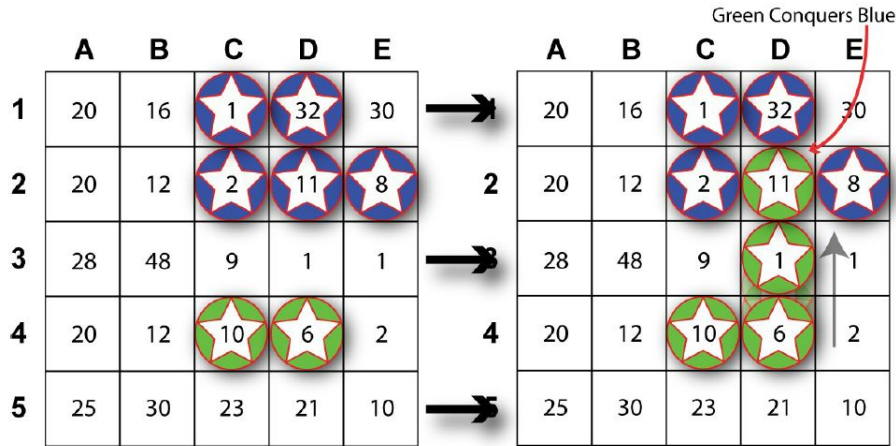
Figure 1. This is a Raid. Green raids D3 and conquers the blue piece in D2 since it is touching the new green piece in D3. A Raid always creates at least one new piece (in the square being raided), but it may not always conquer any of the other player's pieces. Thus, another valid move might have been to have raided E4. Then the green player would own D4 and E4 but would have conquered none of blue's pieces. Note, the total value of each side before the raid was green 16 : blue 54, but afterwards is green 28 : blue 43. These values will be used in the evaluation function as explained later.
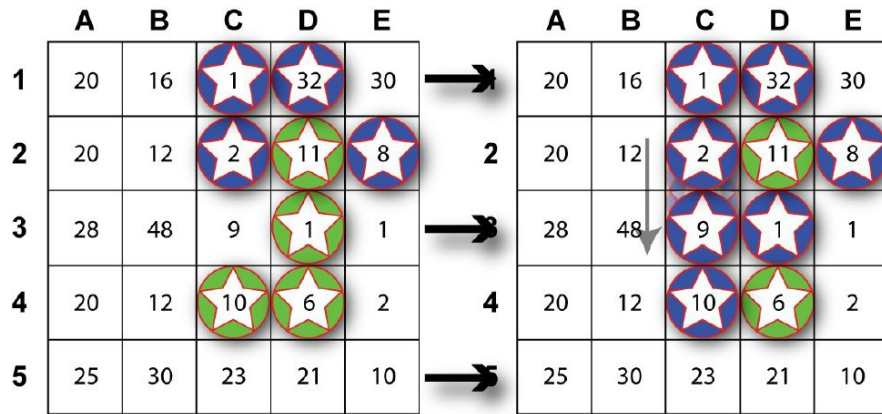

Figure 2. Here blue raids C3. In the process green's pieces at D3 and D4 are conquered because they touch C3. Notice that in its next move, green will not be able to conquer any of blue's pieces, because it can raid only D5 and E4.

Sneak – You can take any unoccupied square on the board that is not next to your existing pieces. This will create a new piece on the board. Unlike Raid which is an aggressive move, Sneak is a covert operation, so it won't conquer any enemy pieces. It simply allows one to place a piece at an unoccupied square that is not reachable by Raid.
Notice that a space that can be Raided cannot be Sneaked (your squirrel warriors are always more aggressive when near home territory). Once you have done a Sneak, your turn is complete.
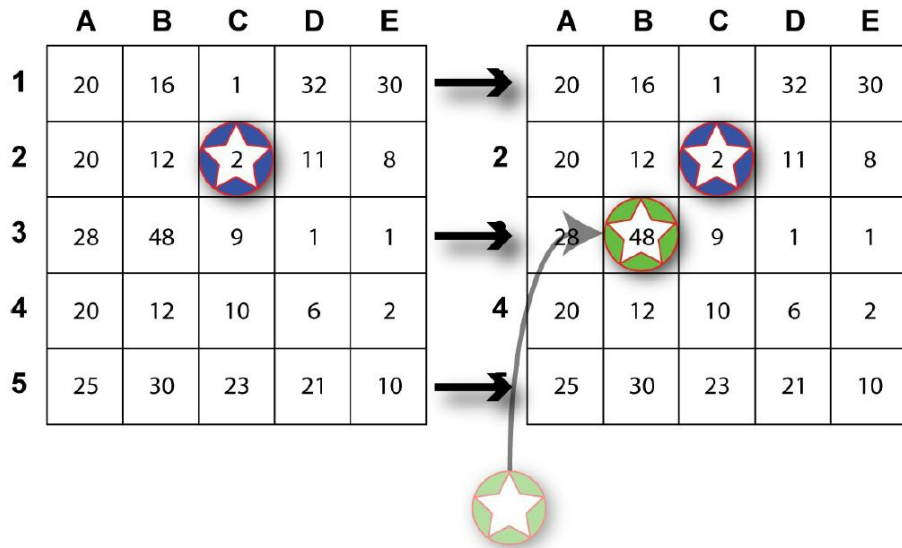
Figure 3. This is a Sneak. In this case, green drops a new piece on square B3. This square is worth 48, which is a higher number, meaning that it contains some important resources (e.g. a large pine tree). A Sneak could have been carried out on any squares except for C2 since blue already occupies it. In the next move, blue can Raid B2, C1, C3 or D2, or it can Sneak any other square, except for B3 and C2, which are already occupied.

8. Again, the Raid operation has two effects: (1) A new piece is created in the target square, and (2) any enemy pieces adjacent to the target square are turned to the player's side. On the other hand, Sneak has only effect (1).
9. Any unoccupied square can be taken with either Raid or Sneak, but they are mutually-exclusive. If the square is horizontally or vertically adjacent to an existing self-owned piece, it's a Raid. Otherwise it's a Sneak.
10. Anytime adjacency is checked (e.g. Raid validity, conquering enemy pieces), it's always checking vertical and horizontal neighbors, but never diagonal. In other words, a diagonal neighbor is never considered adjacent.

Assignment

PART 1: You will write a program to determine the next move by implementing the following algorithms:

- Greedy Best-first Search (30%);
- Minimax (30%);
- Alpha-Beta Pruning (30%).

PART 2:

You will simulate several battles by applying the above algorithms as two players (10%).

Evaluation Function

As introduced, each grid on the game board has different strategic value. The evaluation function of a game board state can be computed by:

$$\text{E(s) = Total\_Value\_Player - Total\_Value\_Opponent}$$

For example, the board on the left side of Figure 2 for the blue player has an evaluated value of:

$$E(s) = (1+32+2+8)-(11+1+10+6) = 15$$

Note: The leaf node values are always calculated from this evaluation function. Although there might be a better evaluation function, you should comply with this rule for simplicity.

Expand Order and Tie Breaking

The positional order on *Figure 4(b)* decides the next move among multiple candidates with the same evaluated value. For example, if some legal moves (B4, C3, C5, D3, E3, E4) have the same evaluated values, the program must pick C3 according to the tie breaker rule.

Your traverse and expand order must be in the positional order as well. For example, your program will traverse on C3, D3, E3, B4, E4, C5 branch in order.

(The wise Master Yoda explained why this positional order is passed on from a long time ago: When choosing territories with the same yield of nuts, the brave Viterbi Fluffy Hackers always prioritize those closer to their enemy's home - North / Up - for attack, then consider those closer to their own home - West / Left - for defence. The Leavey Ninja Squirrels follow the same two rules, but they are more conservative and prefer defence over attack, thus they end up with the same positional priority order.)
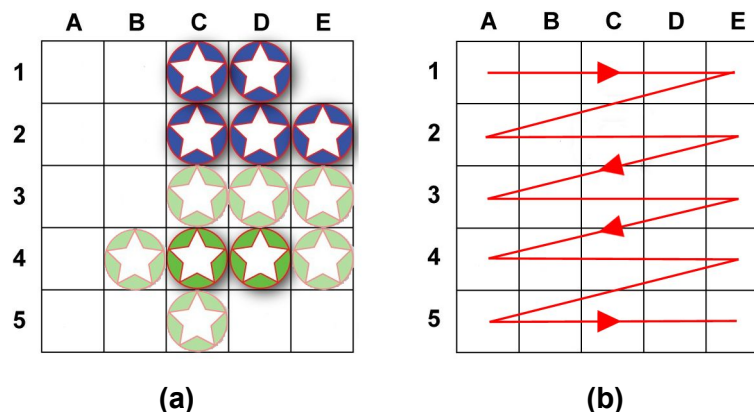

(a)   (b)

Figure 4. Illustration of tie breaking positional order for traverse and expand.

Your expand and traverse orders don't need to differentiate between Raid or Sneak, because each square is valid for only one type of move. When expanding a node for a square, you must perform the correct move to compute the next board state.

Pseudocode

1. Greedy Best-first Search: AIMA section 3.5.1 (Greedy best-first search) The cut-off depth is always 1. Thus, the algorithm is very simple. You only need to pick the action which has the highest evaluation value.
2. Minimax: AIMA Figure 5.3 (Minimax without cut-off) and section 5.4.2 (Explanation of Cutting off search)
3. Alpha-Beta Pruning: AIMA Figure 5.7 (Alpha-Beta without cut-off) and section 5.4.2 (Explanation of Cutting off search)

PART 1

Drawing upon the Force, you must skillfully manipulate three algorithms as powers to guide your squirrel warriors' next move.

Input:

For each test case, you are provided with an input file that describes the current state of the game. In the input and output files, the two sides will be represented as X and O.

<task#>

Greedy Best-first Search = 1, MiniMax = 2, Alpha-beta Pruning = 3

<your player> **X or O**

<cutting off depth>

Cut-off depth started from the root.

<board grid value>

Positive integers from 1 - 99

5 in each row separated with a space, 5 total rows

<current board state>

*: Unoccupied

X: Player 1

O: Player 2

5 in each row, no space in between, 5 total rows

The ordering corresponds with the board values.

Input Example (board state as *Figure 5*):

2
X
2
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*
*****

Figure 5

NOTES:
- The input file is a text file ending with .txt extension.
- The input file given to your program will not contain any formatting errors, so it is not necessary to check for those.

Output:

For each test case, your program should output a file named "**next_state.txt**" showing the next state of the board after the move. **For Minimax and Alpha-Beta Pruning**, your program should output another file named "**traverse_log.txt**" showing the traverse log of your program in the following format. **There is no need to output "traverse_log.txt" for Greedy Best-first Search.**

The format of "next_state.txt" should be:

<next state>

*: Unoccupied
X: Player 1
O: Player 2
5 in each row, no space in between, 5 total rows

For example:

**XX*
**XOX
*X*O*
**OO*
*****

The format of "traverse_log.txt" for Minimax traverse log requires 3 columns. Each column is separated by "," (a single comma). Three columns are **Node**, **Depth** and **Value**. Everything shown here is case sensitive.

For example:
Node,Depth,Value
root,0,-Infinity
A1,1,Infinity
B1,2,19
A1,1,19
E1,2,5
A1,1,5
A2,2,15
A1,1,5
B2,2,23
A1,1,5
A3,2,7
A1,1,5
B3,2,-13
A1,1,-13
C3,2,22
…...

"**Node**": is the node name which refers to the move that is made by the agent. For example, the blue player places a piece at the position "D3". The node name is D3 and has depth 1. Then, the green player places a piece at the position "C3" and has depth 2.
There is special node named "root" which is the name for the root node.
"**Depth**": is the depth of the node. The root node has depth zero.
"**Value**": is the value of the node. The value is initialized to "-Infinity" for the max node (your agent is always max) and "Infinity" for the min (your agent's opponent) node. The value will be updated when its children return the value to the node. The value of leaf nodes is the evaluated value, for example, C3,2,-3
The algorithm traverses from the root node. The log should show both when:
1. The algorithm traverses down to the node.
2. The value of the node is updated from its children.
For example, the log shows the value of the node "D3" when traversing from the root. The log shows the node "D3" again when the node is updated from its children "C3", "E3" and so on. The leaf nodes have no children. Thus, the log shows only once when the algorithm traverses a leaf node and the value is the evaluated value, for example, C3,2,-3. where -3 is the evaluated value.

The format of "traverse_log.txt" for Alpha-Beta traverse log requires 5 columns. Each column is separated by "," (a single comma). The five columns are node, depth, value, alpha, and beta. The description is same as with the MiniMax log. However, you need to show the alpha and beta values in the Alpha-Beta traverse log.

For example:
Node,Depth,Value,Alpha,Beta
root,0,-Infinity,-Infinity,Infinity
A1,1,Infinity,-Infinity,Infinity
B1,2,19,-Infinity,Infinity
A1,1,19,-Infinity,19
E1,2,5,-Infinity,19
A1,1,5,-Infinity,5
A2,2,15,-Infinity,5
A1,1,5,-Infinity,5
B2,2,23,-Infinity,5
A1,1,5,-Infinity,5
A3,2,7,-Infinity,5
A1,1,5,-Infinity,5
B3,2,-13,-Infinity,5
A1,1,-13,-Infinity,-13
C3,2,22,-Infinity,-13
…...

NOTES:
● The output examples above are based on input as in figure 5. They are provided here to demonstrate file format specification, and they only show a small part of the full traverse log (the full log can be found in the samples provided).
● Your program should stop searching when there's no valid move, even if the preset cut-off depth is not reached yet.
● For any test case, it will be marked as correct only if both next state and traverse log are correct. **(For Greedy Best-first Search only next state will be checked)**.
● With the given description, we don't believe that multiple outputs are possible for any test case. If you are able to think of any such case, please let us know and we will make the necessary changes in the grading guidelines.
● The final test cases will be different from the sample test cases provided. Your assignment will be graded based on the performance on the final test cases only.

PART 2

Masterfully controlling the three algorithm powers, you are now ready to wage a full-scale squirrel war. To the wonderland of nuts!!

Input

You are provided with an input file that describes the current state of the game. Your agent should be able to identify one more task.

<task#>

Battle simulation = 4

<first player> **X or O. The first player always moves first**

<first player algorithm>

Greedy Best-first Search = 1, MiniMax = 2, Alpha-beta Pruning = 3

<first player cutting off depth>

<second player> **X or O**

<second player algorithm>

Greedy Best-first Search = 1, MiniMax = 2, Alpha-beta Pruning = 3

<second player cutting off depth>

<board grid value>

Positive integers from 1 - 99

5 in each row separated with a space, 5 total rows

<current board state>

*: Unoccupied

X: Player 1

O: Player 2

5 in each row, no space in between, 5 total rows

The ordering corresponds with the board values.

For example:

Greedy Best-first Search(cut-off depth is always 1) vs. Minimax(with cut-off depth: 2)

4

X

1

1

O

2

2

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*

*****

NOTES:

- **The first player always moves first**.
- The input file is a text file ending with .txt extension.
- The input file given to your program will not contain any formatting errors, so it is not necessary to check for those.

Output:

For each test case, your program should output a file named "**trace_state.txt**" tracing every state of the game until it ends. The game ends when all the squares are occupied (See Rule 6 on Page 1).

The format of "trace_state.txt" should be:

<next state>

<next state>

<next state>

...

For example:

**XX*

**X*X

*****

**OO*

*****

**XX*

**XXX

*****

**OO*

*****

**XX*

**XOX

***O*

**OO*

*****

…

NOTES:

- Your program should stop searching when there's no valid move, even if the preset cut-off depth is not reached yet.

Grading Notice:

- **Please follow the instructions carefully. Any deviations from the instructions will lead your grade to be zero for the assignment**. If you have any doubts, please use the discussion board. Do not assume anything that is not explicitly stated.
- You must use **PYTHON** (Python 2.7) to implement your code.
- You need to create a file named "hw1cs561s16.py". The command to run your program would be as follows: (When you submit the homework on labs.vocareum.com, the following commands will be executed.)

    python hw1cs561s16.py –i *inputFile*

- You will use labs.vocareum.com to submit your code. Please refer to http://help.vocareum.com/article/30-getting-started-students to get started with the system. Please only upload your code to the "/work" directory. Don't create any subfolder or upload any other files.
- If we are unable to execute your code successfully, you will not receive any credits. You are allowed to use standard libraries only. You have to implement any other functions or methods by yourself.
- You will get partial credit based on the percentage of test cases that your program gets right for each task.
- Your program should handle all test cases within a reasonable time (not more than a few seconds for each sample test case). The complexity of test cases is similar to, but not necessarily the same as, the ones provided in the homework.
- The deadline for this assignment is Feb 8, 2016 at 11:59 PM PST. **No late homework will be accepted.** Any late submission will not be graded. Any email for late submission will be ignored.