CS 188 Spring 2019 Introduction to Artificial Intelligence Written HW 2

Due: Monday 2/11/2019 at 11:59pm (submit via Gradescope).

Leave self assessment boxes blank for this due date.

Self assessment due: Monday 2/18/2019 at 11:59pm (submit via Gradescope)

For the self assessment, **fill in the self assessment boxes in your original submission** (you can download a PDF copy of your submission from Gradescope). For each subpart where your original answer was correct, write "correct." Otherwise, write and explain the correct answer.

Policy: Can be solved in groups (acknowledge collaborators) but must be written up individually

Submission: Your submission should be a PDF that matches this template. Each page of the PDF should align with the corresponding page of the template (page 1 has name/collaborators, question 1 begins on page 2, etc.). **Do not reorder, split, combine, or add extra pages.** The intention is that you print out the template, write on the page in pen/pencil, and then scan or take pictures of the pages to make your submission. You may also fill out this template digitally (e.g. using a tablet.)

First name	Gregory
Last name	Uezono
SID	3031967186
Collaborators	

Q1. One Wish Pacman

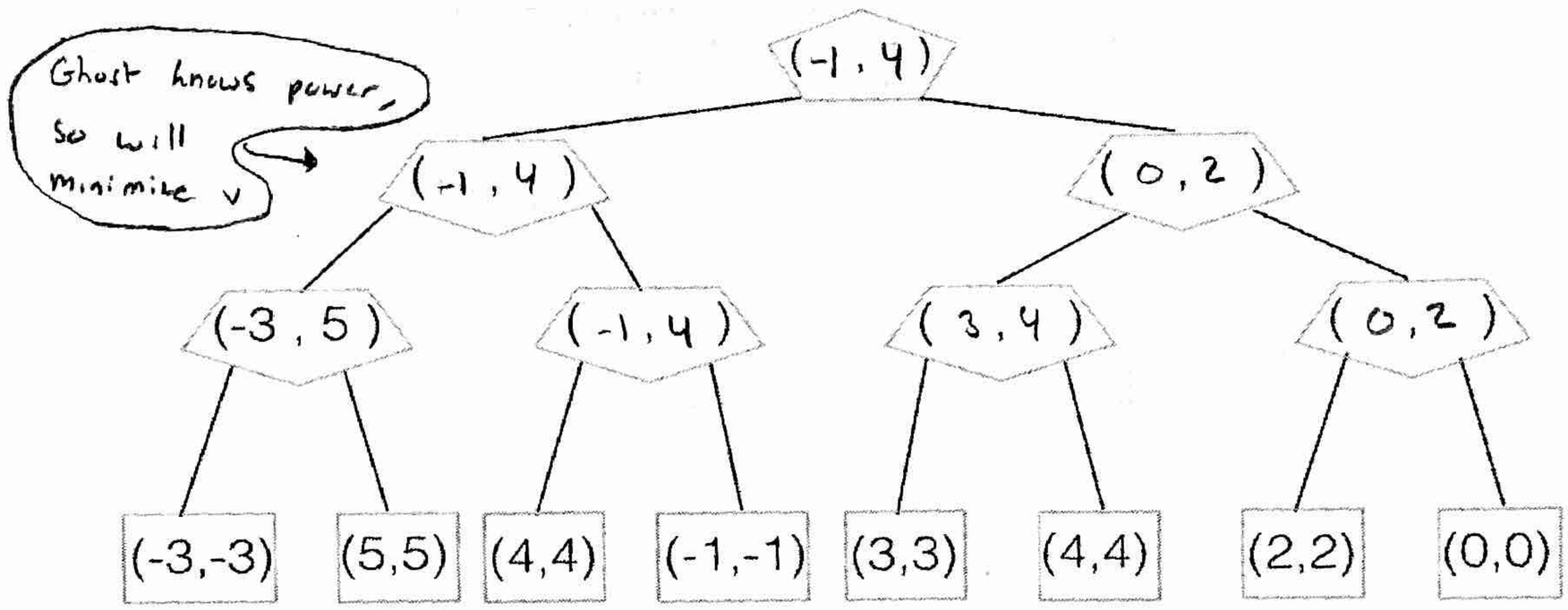
(a) Power Search. Pacman has a special power: *once* in the entire game when a ghost is selecting an action, Pacman can make the ghost choose any desired action instead of the min-action which the ghost would normally take. *The ghosts know about this special power and act accordingly.*

(i) Similar to the minimax algorithm, where the value of each node is determined by the game subtree hanging from that node, we define a value pair (u, v) for each node: u is the value of the subtree if the power is not used in that subtree; v is the value of the subtree if the power is used once in that subtree. For example, in the below subtree with values (-3, 5), if Pacman does not use the power, the ghost acting as a minimizer would choose -3; however, with the special power, Pacman can make the ghost choose the value more desirable to Pacman, in this case 5.

Reminder: Being allowed to use the power once during the game is different from being allowed to use the power in only one node in the game tree below. For example, if Pacman's strategy was to always use the special power on the second ghost then that would only use the power once during execution of the game, but the power would be used in four possible different nodes in the game tree.

For the terminal states we set $u = v = \text{UTILITY}(State)$.

Fill in the (u, v) values in the modified minimax tree below. Pacman is the root and there are two ghosts.



(ii) Complete the algorithm below, which is a modification of the minimax algorithm, to work in the general case: Pacman can use the power at most once in the game but Pacman and ghosts can have multiple turns in the game.

```
function VALUE(state)
    if state is leaf then
        u ← UTILITY(state)
        v ← UTILITY(state)
        return (u, v)
    end if
    if state is Max-Node then
        return MAX-VALUE(state)
    else
        return MIN-VALUE(state)
    end if
end function
```

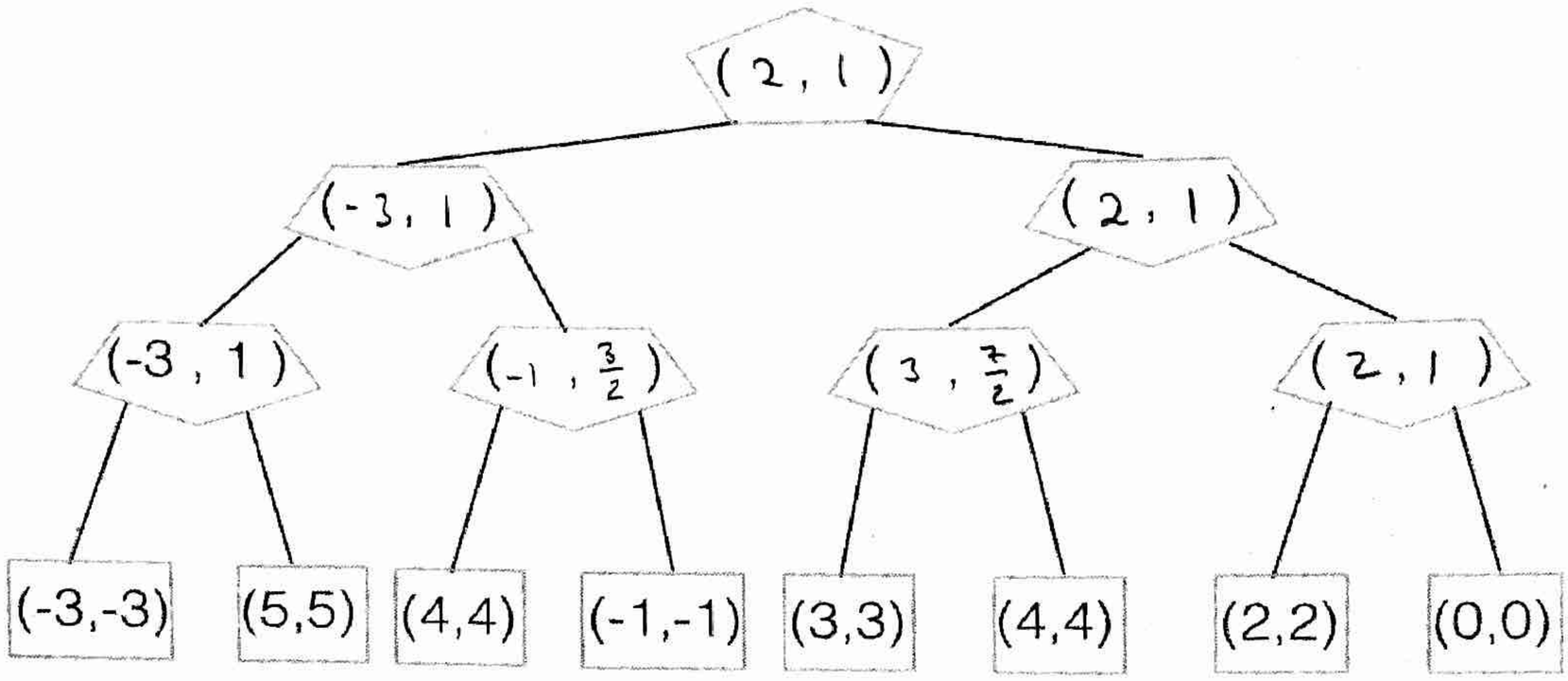
```
function MAX-VALUE(state)
    uList ← [ ], vList ← [ ]
    for successor in SUCCESSORS(state) do
        (u', v') ← VALUE(successor)
        uList.append(u')
        vList.append(v')
    end for
    u ← max(uList)
    v ← max(vList)
    return (u, v)
end function
```

```
function MIN-VALUE(state)
    uList ← [ ], vList ← [ ]
    for successor in SUCCESSORS(state) do
        (u', v') ← VALUE(successor)
        uList.append(u')
        vList.append(v')
    end for
    u ← min(uList)
    v ← max(vList)
    return (u, v)
end function
```

(b) Weak-Power Search. Now, rather than giving Pacman control over a ghost move once in the game, the special power allows Pacman to once make a ghost act randomly. The ghosts know about Pacman's power and act accordingly.

(i) The propagated values (u, v) are defined similarly as in the preceding question: u is the value of the subtree if the power is not used in that subtree: v is the value of the subtree if the power is used once in that subtree.

Fill in the (u, v) values in the modified minimax tree below, where there are two ghosts.



(ii) Complete the algorithm below, which is a modification of the minimax algorithm, to work in the general case: Pacman can use the weak power at most once in the game but Pacman and ghosts can have multiple turns in the game.

Hint: you can make use of a min, max, and average function

```
function VALUE(state)
    if state is leaf then
        u ← UTILITY(state)
        v ← UTILITY(state)
        return (u, v)
    end if
    if state is Max-Node then
        return MAX-VALUE(state)
    else
        return MIN-VALUE(state)
    end if
end function
```

```
function MAX-VALUE(state)
    uList ← [ ], vList ← [ ]
    for successor in SUCCESSORS(state) do
        (u', v') ← VALUE(successor)
        uList.append(u')
        vList.append(v')
    end for
    u ← max(uList)
    v ← max(vList)
    return (u, v)
end function
```

```
function MIN-VALUE(state)
    uList ← [ ], vList ← [ ]
    for successor in SUCCESSORS(state) do
        (u', v') ← VALUE(successor)
        uList.append(u')
        vList.append(v')
    end for
    u ← min (uList)
    v ← average (vList)
    return (u, v)
end function
```

(c) **Power search without changing the algorithm** Now consider that we're in the original power search setting mentioned in Part (a). How would you solve power search within this model by only redefining the state and transition function appropriately? You're not allowed to change the algorithm, and you're only allowed to use scalar values.

Algorithm: minimax Search Tree (cannot change)

States: boolean powerUsed to denote if PM has used the power.
position of PM
position of ghosts

State Transition Function (takes in actions and returns a new state):

Actions: all possible moves pacman can move to + moves if pacman uses the power on each ghost

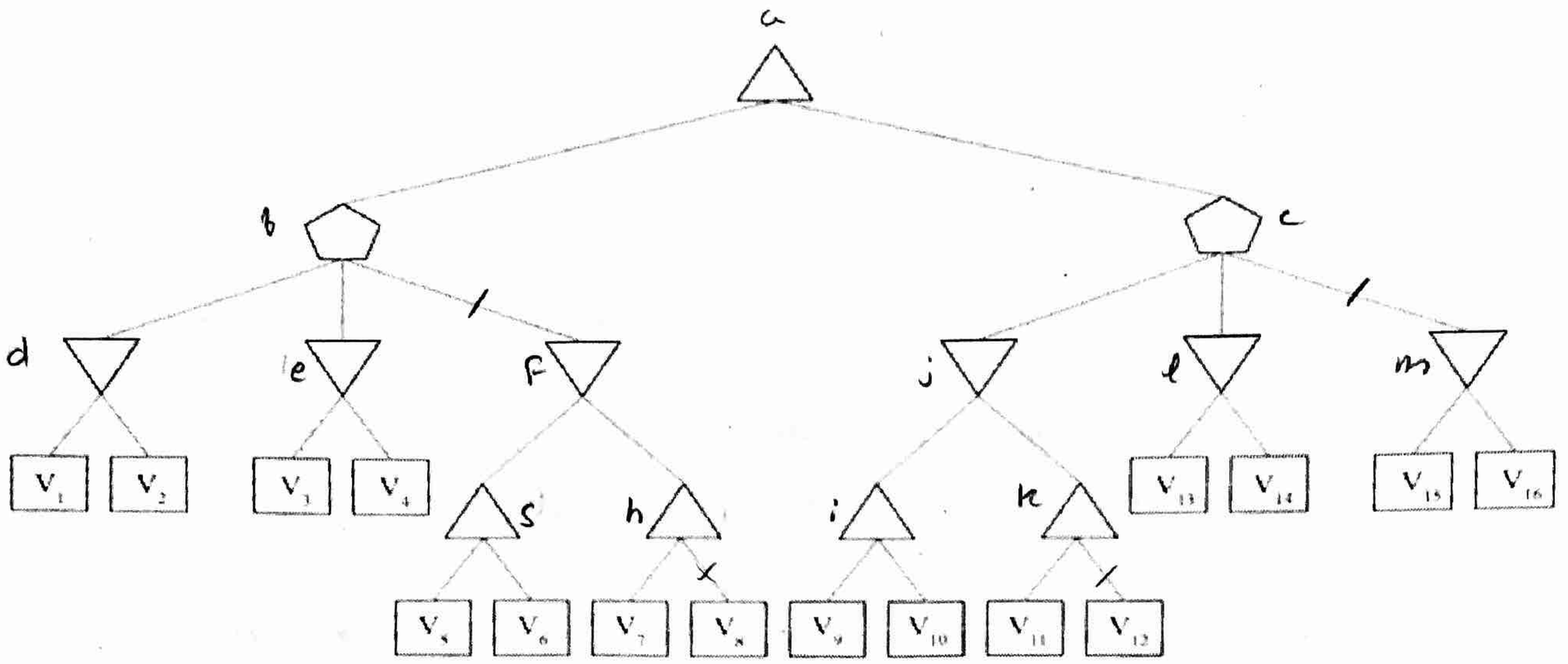
Function: given all these states calculate utility of each state (takes into account pellets, ghost proximity, and powerUsed boolean)

returns state w/ highest utility

Goal State: All pellets eaten

Q2. MedianMiniMax

You're living in utopia! Despite living in utopia, you still believe that you need to maximize your utility in life; but other people want to minimize your utility, and the world is a zero-sum game. But because you live in utopia, a benevolent social planner occasionally steps in and chooses an option that is a compromise. Essentially, the social planner (represented as the pentagon) is a median node that chooses the successor with median utility. Your struggle with your fellow citizens can be modelled as follows:



There are some nodes that we are sometimes able to prune. List all of the terminal nodes (from V_1 to V_{16}) such that **there exists a possible situation** for which the node **can be pruned**. In other words, you must consider **all** possible pruning situations. Assume that evaluation order is **left to right** and all V_i's are **distinct.**

Note that as long as there exists ANY pruning situation (does not have to be the same situation for every node), you should list the node as prunable. Although the details are different, the same principle underlying alpha-beta pruning applies here too, simply prune a sub-tree when you can reason that its value will not affect your final decision.

V_5, V_6, V_7, V_8 IF node d = node e $\Longrightarrow$ node b (median) = d = e

V_{12} IF node i < V_{11}

V_{15}, V_{16} IF node j = l $\Longrightarrow$ node c (median) = j = l