CS 188 Spring 2019 | Introduction to Artificial Intelligence | Written HW 2 Sol.

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

**Instructions for self-assessment:** Take your original submission and annotate any differences from the provided solutions. For **each subpart** where your original answer was correct, **write "correct"** to demonstrate that you have checked your work. For each subpart where your original answer was incorrect, write out the correct answer and comment on the difference between your answer and the explanation provided in the solutions. You should complete your self-assessment using a **different color** of ink from your original work. If you need to, you can download a PDF copy of your submission from Gradescope.

Your submission must be a PDF that follows the template (page 1 has name/collaborators, question 1 begins on page 2, etc.). Do not reorder, split, combine, or add extra pages. If your original homework submission did not follow the correct format, **you must fix the format to receive credit on your self-assessment**.

If you did not complete some questions in your original submission, first complete those questions without consulting the solutions and then use a different color of ink to conduct a self-assessment.

# Q1. One Wish Pacman

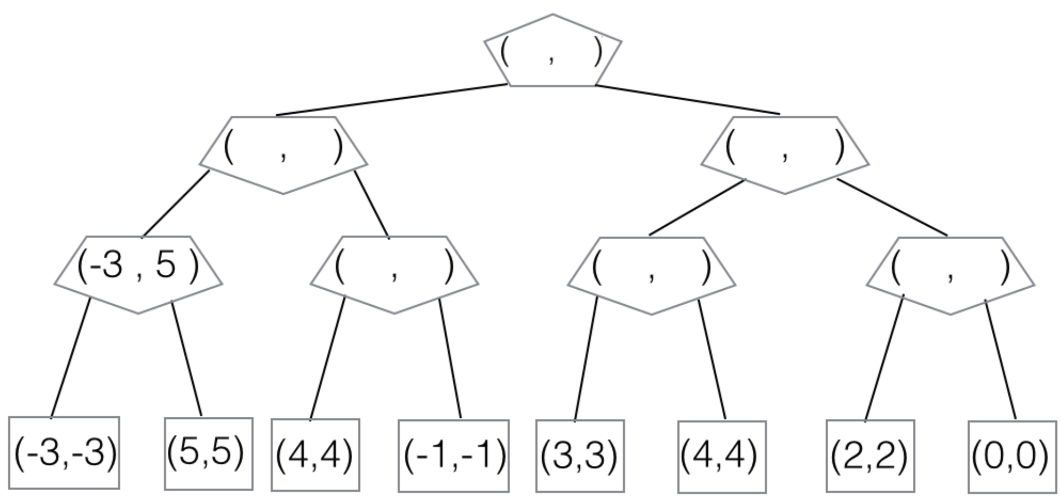
**(a) Power Search.** Pacman has a special power: *once* in the entire game when a ghost is selecting an action, Pacman can make the ghost choose any desired action instead of the min-action which the ghost would normally take. *The ghosts know about this special power and act accordingly.*

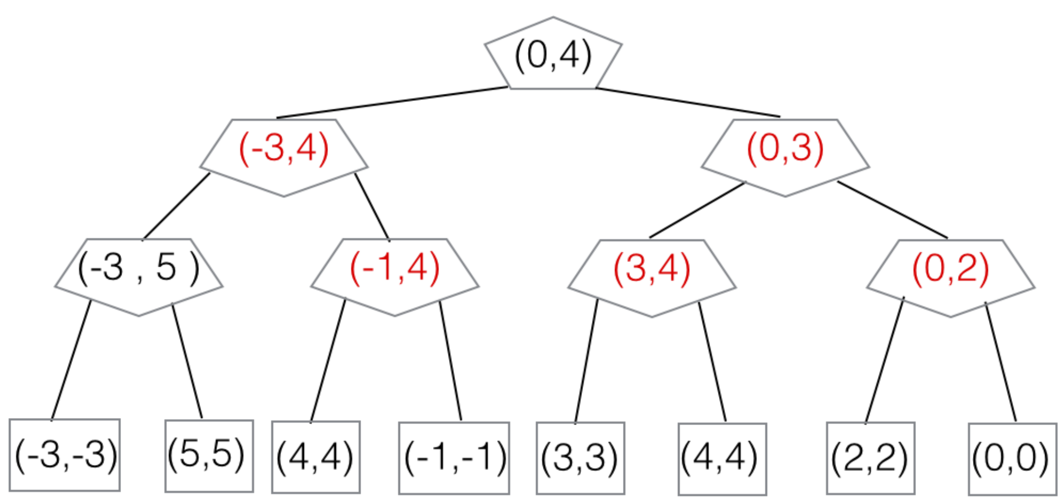
**(i)** Similar to the minimax algorithm, where the value of each node is determined by the game subtree hanging from that node, we define a value pair $(u, v)$ for each node: $u$ is the value of the subtree if the power is not used in that subtree; $v$ is the value of the subtree if the power is used 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 $u = v = \textsc{Utility}(State)$.

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




Please see the solution of the general algorithm in the next part to see how the $u, v$ values get propagated up the game tree.

**(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(max(uList), min(vList))

    return (u, v)
end function
```

The $u$ value of a min-node corresponds to the case if Pacman does not use his power in the game subtree hanging from the current min-node. Therefore, it is equal to the minimum of the $u$ values of the children of the node.

The $v$ value of the min-node corresponds to the case when pacman uses his power once in the subtree. Pacman has two choices here - a) To use the power on the current node, or b) To use the power further down in the subtree.

In case a), the value of the node corresponds to choosing the best among the children's $u$ values = $\max(uList)$ (we consider $u$ values of children as Pacman is using his power on this node and therefore, cannot use it in the subtrees of the node's children).

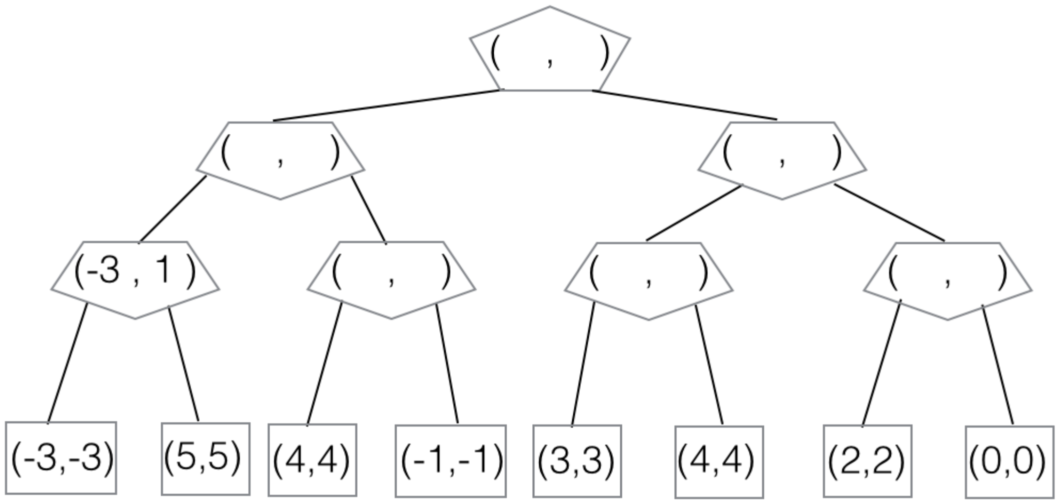
In case b), Pacman uses his power in one of the child subtrees so we consider the $v$ values of the children, Since Pacman is not using his power on this node, the current node acts as a minimizer, making the value in case b) = $\min(vList)$

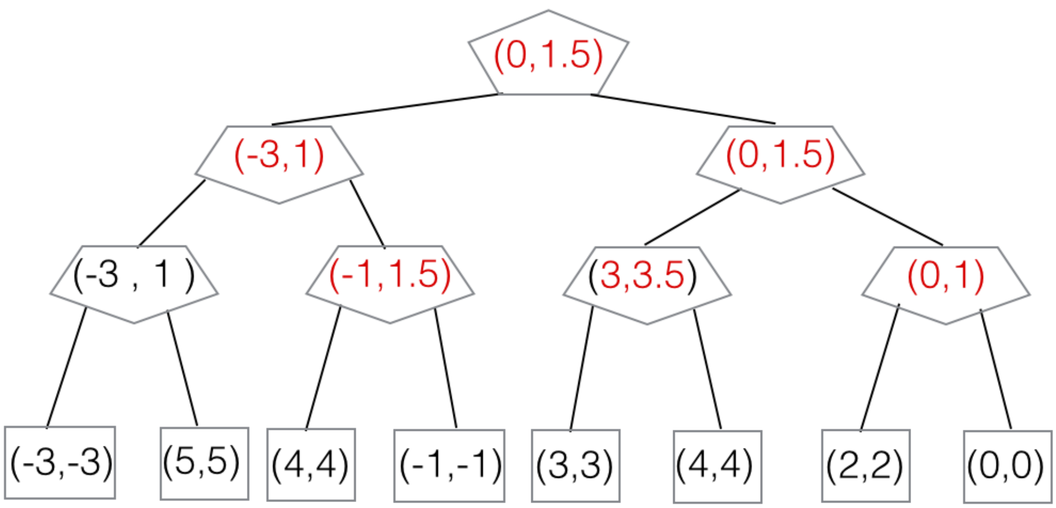
The $v$ value at the current node is the best of the above two cases.

**(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.




Please see the solution of the general algorithm in the next part to see how the $u, v$ values get propagated up the game tree.

**(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**
    $u \leftarrow$ UTILITY(*state*)
    $v \leftarrow$ 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*)
  $uList \leftarrow [\,], vList \leftarrow [\,]$
  **for** *successor* in SUCCESSORS(*state*) **do**
    $(u', v') \leftarrow$ VALUE(*successor*)
    $uList$.append($u'$)
    $vList$.append($v'$)
  **end for**
  $u \leftarrow \max(uList)$
  $v \leftarrow \max(vList)$
  **return** $(u, v)$
**end function**

**function** MIN-VALUE(*state*)
  $uList \leftarrow [\,], vList \leftarrow [\,]$
  **for** *successor* in SUCCESSORS(*state*) **do**
    $(u', v') \leftarrow$ VALUE(*successor*)
    $uList$.append($u'$)
    $vList$.append($v'$)
  **end for**

  $u \leftarrow$ ___ $\min(uList)$ ___

  $v \leftarrow$ ___ $\max(\text{avg}(uList), \min(vList))$ ___

  **return** $(u, v)$
**end function**

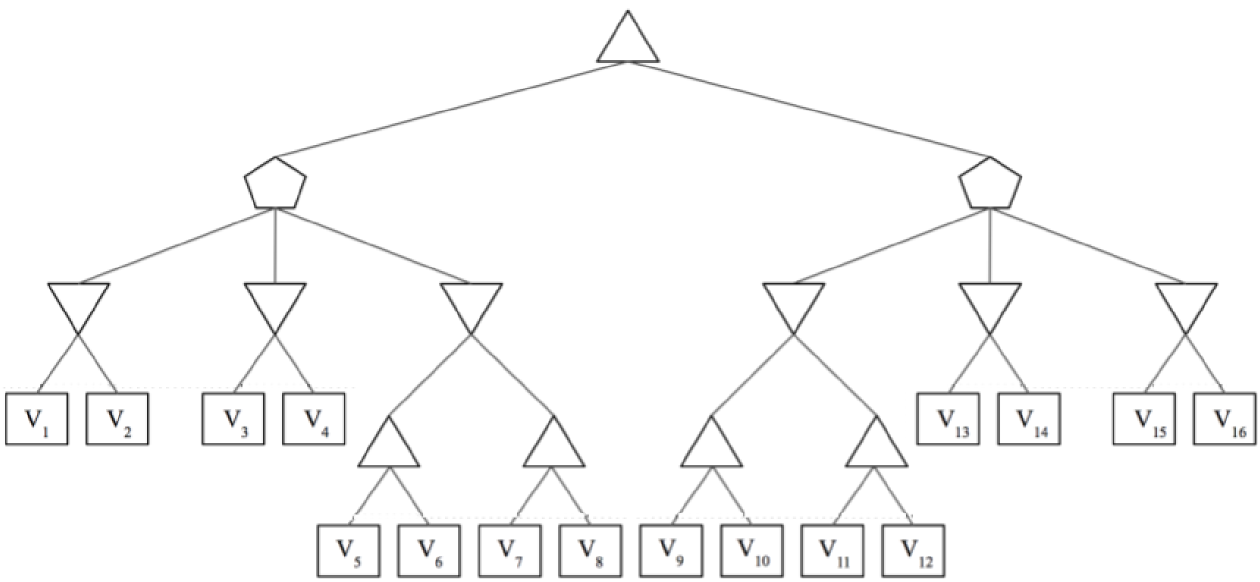
The solution to this scenario is same as before, except that when considering case a) for the $v$ value of a min-node, the value of the node corresponds to choosing the average of the children's $u$ values = $avg(uList)$

**(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.

Give Pacman an extra category of moves, called ghost-fixing moves. When one such move is executed, it updates the ghost state (based on Pacman's choice), and leaves it as Pacman's turn to move again. The state can be augmented to include a flag that tracks whether the ghost-fixing move has been moved or not. Once the flag is set, Pacman cannot make the ghost fixing move again.

# 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:

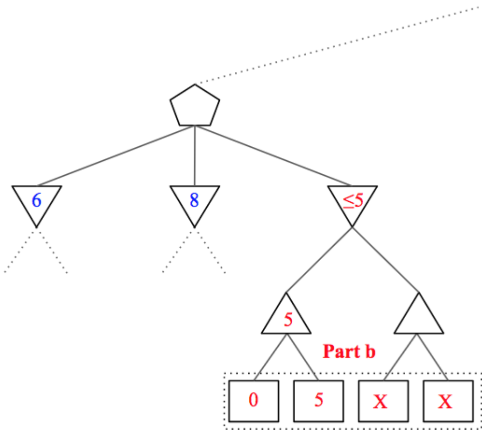



There are some nodes that we are sometimes able to prune. List all of the terminal nodes (from $V_1$ to $V_{16}$) such that **there exists a possible situation** for which the node **can be pruned**. In other words, you must consider **all** possible pruning situations. Assume that evaluation order is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

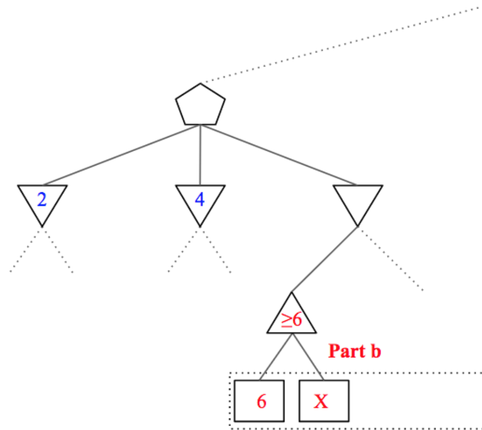
Nodes that can be pruned for at least one possible situation are $V_6$, $V_7$, $V_8$, $V_{11}$, $V_{12}$, $V_{14}$, $V_{15}$, $V_{16}$. The solutions for these are presented below in four parts: part (a) looks at nodes from $V_1$ to $V_4$, part (b) looks at $V_5$ to $V_8$, and so on.

**Part a:**
For the left median node with three children, at least two of the childrens' values must be known since one of them will be guaranteed to be the value of the median node passed up to the final maximizer. For this reason, none of the nodes in part a can be pruned.




The value of this subtree will only get smaller.

The median node will **NOT** choose the value of this subtree. 6 is the median.




The value of this subtree will only get bigger.

*If the value of this subtree is chosen by the minimizer****, it will **NOT** be chosen by the median node.

****It is possible that the median is the value of the subtree to the right that we haven't looked at yet*

**Part b (pruning $V_7, V_8$ ):**
Let $min_1, min_2, min_3$ be the values of the three minimizer nodes in this subtree.

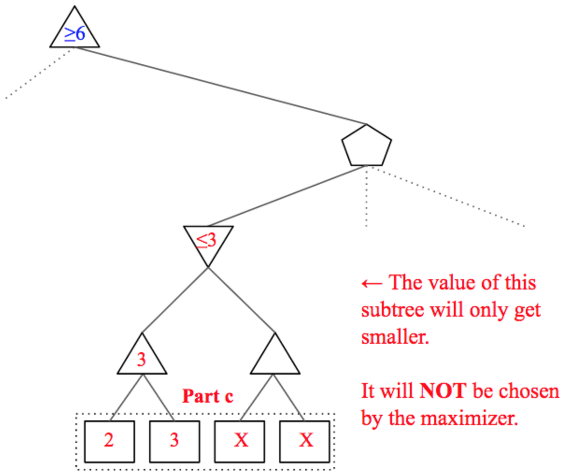
In this case, we may not need to know the final value $min_3$. The reason for this is that we may be able to put a bound on its value after exploring only partially, and determine the value of the median node as either $min_1$ or $min_2$ if $min_3 \leq \min(min_1, min_2)$ or $min_3 \geq \max(min_1, min_2)$.

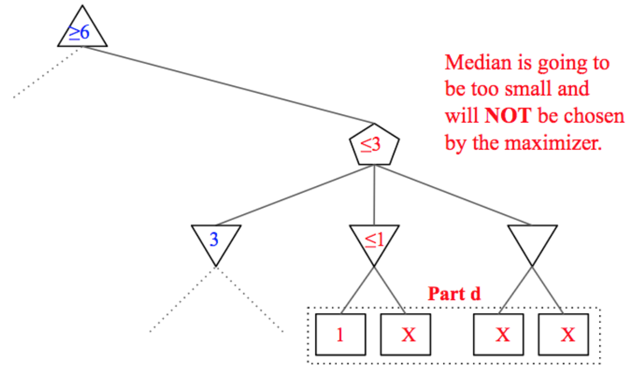
We can put an upper bound on $min_3$ by exploring the left subtree $V_5, V_6$ and if $\max(V_5, V_6)$ is lower than both $min_1$ and $min_2$, the median node's value is set as the smaller of $min_1, min_2$ and we don't have to explore $V_7, V_8$ in Figure 1.

**Part b (pruning $V_6$):**
It's possible for us to put a lower bound on $min_3$. If $V_5$ is larger than both $min_1$ and $min_2$, we do not need to explore $V_6$.

The reason for this is subtle, but if the minimizer chooses the left subtree, we know that $min_3 \geq V_5 \geq \max(min_1, min_2)$ and we don't need $V_6$ to get the correct value for the median node which will be the larger of $min_1, min_2$.

If the minimizer chooses the value of the right subtree, the value at $V_6$ is unnecessary again since the minimizer never chose its subtree.




**Part c (pruning $V_{11}, V_{12}$ ):**
Assume the highest maximizer node has a current value $max_1 \geq Z$ set by the left subtree and the three minimizers on this right subtree have value $min_1, min_2, min_3$.

In this part, if $min_1 \leq \max(V_9, V_{10}) \leq Z$, we do not have to explore $V_{11}, V_{12}$. Once again, the reasoning is subtle, but we can now realize if either $min_2 \leq Z$ or $min_3 \leq Z$ then the value of the right median node is for sure $\leq Z$ and is useless.

Only if both $min_2, min_3 \geq Z$ will the whole right subtree have an effect on the highest maximizer, but in this case the exact value of $min_1$ is not needed, just the information that it is $\leq Z$. Clearly in both cases, $V_{11}, V_{12}$ are not needed since an exact value of $min_1$ is not needed.

We will also take the time to note that if $V_9 \geq Z$ we do have to continue the exploring as $V_{10}$ could be even greater and the final value of the top maximizer, so $V_{10}$ can't really be pruned.

**Part d (pruning $V_{14}, V_{15}, V_{16}$ ):**
Continuing from part c, if we find that $min_1 \leq Z$ and $min_2 \leq Z$ we can stop.

We can realize this as soon we explore $V_{13}$. Once we figure this out, we know that our median node's value must be one of these two values, and neither will replace $Z$ so we can stop.