Self-assessment due: Tuesday 7/9/2019 at 11:59pm (submit via Gradescope)

Please grade your own work honestly as we will manually inspect a small subset of the submissions to verify that they are done accurately.
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.

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.
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(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 $= \text{max}(uList)$ (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) $= \text{min}(vList)$

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.

(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 \textsc{Value}(state)
  if state is leaf then
    \( u \leftarrow \text{Utility}(state) \)
    \( v \leftarrow \text{Utility}(state) \)
    return \((u, v)\)
  end if
  if state is Max-Node then
    return \textsc{Max-Value}(state)
  else
    return \textsc{Min-Value}(state)
  end if
end function

function \textsc{Max-Value}(state)
  \( uList \leftarrow [\,], vList \leftarrow [\,] \)
  for successor in \textsc{Successors}(state) do
    \((u', v') \leftarrow \text{Value}(successor)\)
    \( uList.\text{append}(u') \)
    \( vList.\text{append}(v') \)
  end for
  \( u \leftarrow \max(uList) \)
  \( v \leftarrow \max(\text{avg}(uList), \min(vList)) \)
  return \((u, v)\)
end function

function \textsc{Min-Value}(state)
  \( uList \leftarrow [\,], vList \leftarrow [\,] \)
  for successor in \textsc{Successors}(state) do
    \((u', v') \leftarrow \text{Value}(successor)\)
    \( uList.\text{append}(u') \)
    \( vList.\text{append}(v') \)
  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 = \( \text{avg}(uList) \)
Q2. MedianMiniMax

You’re living in utopia! Despite living in utopia, you still believe that you need to maximize your utility in life, other people want to minimize your utility, and the world is a 0 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. In each part, mark all of the terminal nodes 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 mark the node as prunable. Also, alpha-beta pruning does not apply here, simply prune a sub-tree when you can reason that its value will not affect your final utility.

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 children’s 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.

**Part b (pruning V₇, V₈):**
Let \( \text{min}_1, \text{min}_2, \text{min}_3 \) be the values of the three minimizer nodes in this subtree.

In this case, we may not need to know the final value \( \text{min}_3 \). 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 \( \text{min}_1 \) or \( \text{min}_2 \) if \( \text{min}_3 \leq \text{min}(\text{min}_1, \text{min}_2) \) or \( \text{min}_3 \geq \text{max}(\text{min}_1, \text{min}_2) \).

We can put an upper bound on \( \text{min}_3 \) by exploring the left subtree \( V_5, V_6 \) and if \( \text{max}(V_5, V_6) \) is lower than both \( \text{min}_1 \) and \( \text{min}_2 \), the median node’s value is set as the smaller of \( \text{min}_1, \text{min}_2 \) and we don’t have to explore \( V_7, V_8 \) in Figure 1.

**Part b (pruning V₆):**
It’s possible for us to put a lower bound on \( \text{min}_3 \). If \( V_5 \) is larger than both \( \text{min}_1 \) and \( \text{min}_2 \), 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 \( \text{min}_3 \geq V_5 \geq \text{max}(\text{min}_1, \text{min}_2) \) and we don’t need \( V_6 \) to get the correct value for the median node which will be the larger of \( \text{min}_1, \text{min}_2 \).

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.

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 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.