CS 188 Spring 2015 Introduction to Artificial Intelligence Midterm 1

- You have approximately 2 hours and 50 minutes.
- The exam is closed book, closed calculator, and closed notes except your one-page crib sheet.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation. All short answer sections can be successfully answered in a few sentences AT MOST.

First name	
Last name	
SID	
edX username	

First and last name of student to your left	
First and last name of student to your right	

For staff use only:

Q1.	Pacman's Tour of San Francisco	/12
Q2.	Missing Heuristic Values	/6
Q3.	PAC-CORP Assignments	/10
Q4.	k-CSPs	/5
Q5.	One Wish Pacman	/12
Q6.	AlphaBetaExpinimax	/9
Q7.	Lotteries in Ghost Kingdom	/11
Q8.	Indecisive Pacman	/13
Q9.	Reinforcement Learning	/8
Q10.	Potpourri	/14
	Total	/100

THIS PAGE IS INTENTIONALLY LEFT BLANK

Q1. [12 pts] Pacman's Tour of San Francisco

Pacman is visiting San Francisco and decides to visit N different landmarks $\{L_1, L_2, \ldots, L_N\}$. Pacman starts at L_1, which can be considered visited, and it takes t_{ij} minutes to travel from L_i to L_j.

(a) [2 pts] Pacman would like to find a route that visits all landmarks while minimizing the total travel time. Formulating this as a search problem, what is the minimal state representation?

minimal state representation:
The minimal state representation is Pacman's current location and which landmarks he has visited (boolean indicators or a set or an unordered list) OR the inverse, which landmarks Pacman has not visited.

(b) [2 pts] Ghosts have invaded San Francisco! If Pacman travels from L_i to L_j, he will encounter g_{ij} ghosts. Pacman wants to find a route which minimizes total travel time *without encountering more than G_{max} ghosts* (while still visiting all landmarks). What is the minimal state representation?

minimal state representation:
The minimal state representation is the visited (or unvisited) landmarks (as above), Pacman's current landmark (as above), and the number of ghosts encountered thus far (or the inverse, the number of ghosts that can be encountered in the future).

(c) [4 pts] The ghosts are gone, but now Pacman has brought all of his friends to take pictures of all the landmarks. Pacman would like to find routes for him and each of his $k-1$ friends such that *all landmarks are visited by at least one individual*, while minimizing the **sum of the tour times** of all individuals. You may assume that Pacman and all his friends start at landmark L_1 and each travel independently at the same speed. Formulate this as a search problem and fill in the following:

minimal state representation:
The state representation is the landmarks visited by at least one agent, stored as N boolean indicators, a set, or an unordered list (or the inverse – the unvisited landmarks) and the current location of each of the k agents.

actions between states:
An action can be a single agent moving from one landmark to another or a number of the agents moving from their respective landmarks to new landmarks. The new landmarks need not be unvisited or unoccupied.

cost function $c(s, s')$ between neighboring states:
The cost function is the sum of the time traveled by each agent between state s and s' (or just a single term if only one agent moves per action)

(d) [4 pts] Pacman would now like to find routes for him and each of his $k-1$ friends such that all landmarks are still visited by at least one individual, but now minimizing the **maximum tour time** of any individual. Formulate this as a search problem and fill in the following:

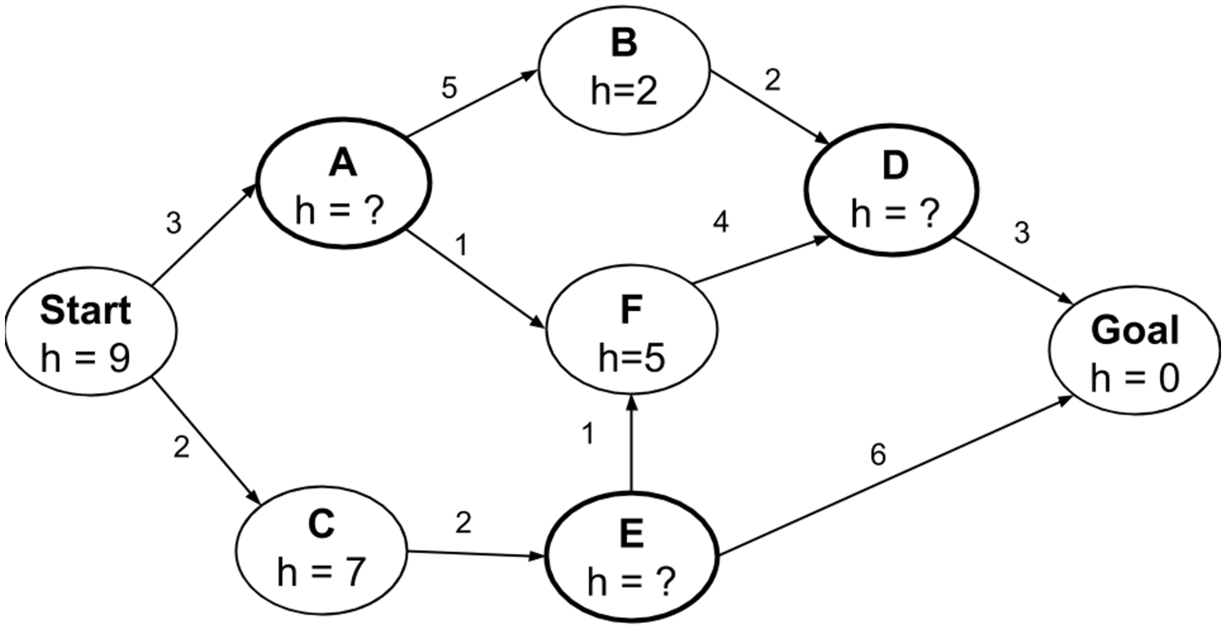
minimal state representation:
The state representation is the landmarks visited and the current location of each agent (both as above), as well as the tour time for each agent.

actions between states:
An action is when a single agents moves from one landmark to another or some number of the agents move from their current landmark to another landmark.

cost function $c(s, s')$ between neighboring states:
Let t_{max} be the maximum tour time of any individual at state s and t'_{max} be the equivalent at state s'. Then the cost between s and s' is $t'_{max} - t_{max}$.

Q2. [6 pts] Missing Heuristic Values

Consider the state space graph shown below in which some of the states are missing a heuristic value. Determine the possible range for each missing heuristic value so that the heuristic is admissible and consistent. If this isn't possible, write so.



State	Range for h(s)
A	$6 \leq h(A) \leq 6$
D	$1 \leq h(D) \leq 3$
E	$5 \leq h(E) \leq 6$

We only need to check for consistency since admissibility is implied by consistency. A consistent heuristic is one such that $h(s) \leq c(s, s') + h(s')$. For the search graph above, this means that:
For $s = A$:

$$\begin{aligned} h(Start) \leq c(Start, A) + h(A) &\implies 6 \leq h(A) \\ h(A) \leq c(A, B) + h(B) &\implies h(A) \leq 7 \\ h(A) \leq c(A, G) + h(F) &\implies h(A) \leq 6 \text{ to be consistent} \end{aligned}$$

For $s = D$:

$$\begin{aligned} h(D) \leq c(D, G) + h(G) &\implies h(D) \leq 3 \\ h(B) \leq c(B, D) + h(D) &\implies 0 \leq h(D) \\ h(F) \leq c(F, D) + h(D) &\implies 1 \leq h(D) \text{ to be consistent} \end{aligned}$$

For $s = E$:

$$\begin{aligned} h(E) \leq c(E, G) + h(G) &\implies h(E) \leq 6 \\ h(C) \leq c(C, E) + h(E) &\implies 5 \leq h(E) \\ h(E) \leq c(E, F) + h(F) &\implies h(E) \leq 6 \text{ to be consistent} \end{aligned}$$

Q3. [10 pts] PAC-CORP Assignments

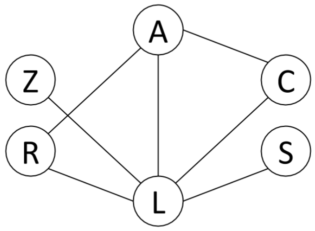
Your CS188 TAs have all secured jobs at PAC-CORP. Now, PAC-CORP must assign each person to exactly one team. The TAs are Alvin (A), Chelsea (C), Lisa (L), Rohin (R), Sandy (S), and Zoe (Z). We would like to formulate this as a CSP using one variable for each TA. The teams to choose from are:

Team 1:	Ghostbusting
Team 2:	Pellet Detection
Team 3:	Capsule Vision
Team 4:	Fruit Processing
Team 5:	R&D
Team 6:	Mobile

The TAs have the following preferences. Note that some of the teams may not receive a TA and some of the teams may receive more than one TA.

- Alvin (A) and Chelsea (C) must be on the same team.
- Sandy (S) must be on an even team (2, 4, or 6).
- Lisa (L) must be on one of the last 3 teams.
- Alvin (A) and Rohin (R) must be on different teams.
- Zoe (Z) must be on Team 1 Ghostbusting or Team 2 Pellet Detection.
- Chelsea's (C) team number must be greater than than Lisa's (L) team number.
- Lisa (L) cannot be on a team with any other TAs.

(a) [3 pts] Complete the constraint graph for this CSP.



Binary constraints are 1, 4, 6, and 7. Constraint 1 connects A and C. Constraint 4 connects A and R. Constraint 6 connects C and L. Constraint 7 connects L with all other variables.

(b) [2 pts] On the grid below, cross out domains that are removed after enforcing all unary constraints. (The second grid is a backup in case you mess up on the first one. Clearly cross out the first grid if it should not be graded.)

A	1	2	3	4	5	6
C	1	2	3	4	5	6
S	~~1~~	2	~~3~~	4	~~5~~	6
L	~~1~~	~~2~~	~~3~~	4	5	6
R	1	2	3	4	5	6
Z	1	2	~~3~~	~~4~~	~~5~~	~~6~~

A	1	2	3	4	5	6
C	1	2	3	4	5	6
S	~~1~~	2	~~3~~	4	~~5~~	6
L	~~1~~	~~2~~	~~3~~	4	5	6
R	1	2	3	4	5	6
Z	1	2	~~3~~	~~4~~	~~5~~	~~6~~

Unary constraints are constraint 2, 3, and 5. Constraint 2 removes 1, 3, and 5 from the domain of S. Constraint 3 removes 1, 2, and 3 from the domain of L. Constraint 5 removes 3, 4, and 5 from the domain of Z.

(c) [2 pts] When using Minimum Remaining Values (MRV) to select the next variable, which variable should be assigned after unary constraints are enforced?

○ A ○ C ○ S ○ L ○ R ● Z

Z has the fewest values remaining in it's domain, so it will be assigned first by MRV.

(d) [3 pts] Assume that the current state of the CSP is shown below. Cross off the values that are eliminated after enforcing arc consistency at this stage. You should only enforce binary constraints. (The second grid is back-up. Clearly cross out the first grid if it should not be graded.)

A	~~1~~	~~2~~	~~3~~	4	~~5~~	~~6~~
C	~~1~~			4		
S				4		
L		2	3	~~4~~	~~5~~	~~6~~
R	1			~~4~~	5	6
Z	1	2	3	4	5	6

A	~~1~~	~~2~~	~~3~~	4	~~5~~	~~6~~
C	~~1~~			4		
S				4		
L		2	3	~~4~~	~~5~~	~~6~~
R	1			~~4~~	5	6
Z	1	2	3	4	5	6

The arcs that prune values are below.

1. C-L: Constraint 6 removes 1 from C.
2. A-C: Constraint 1 removes 1, 2, 3, 5 and 6 from A.
3. R-A: Constraint 4 removes 4 from R.
4. L-C: Constraint 1 removes 4, 5, and 6 from C.

Q4. [5 pts] k-CSPs

Let a k-CSP be a CSP where the solution is allowed to have k variables violate constraints. We would like to modify the classic CSP algorithm to solve k-CSPs. The classic backtracking algorithm is shown below. To modify it to solve k-CSPs, we need to change line 15. Note that k is used to denote the number of allowable inconsistent variables and i is the number of inconsistent variables in the current assignment.

```
 1: function K-CSP-BACKTRACKING(csp, $k$)
 2:     return Recursive-Backtracking({}, csp, 0, $k$)
 3: end function
```

```
 1: function RECURSIVE-BACKTRACKING(assignment, csp, $i$, $k$)
 2:     if assignment is complete then
 3:         return assignment
 4:     end if
 5:     var ← Select-Unassigned-Variable(Variables[csp], assignment, csp)
 6:     for each value in Order-Domain-Values(var, assignment, csp) do
 7:         if value is consistent with assignment given Constraints(csp) then
 8:             add {var = value} to assignment
 9:             result ← Recursive-Backtracking(assignment, csp, $i$, $k$)
10:             if result $\neq$ failure then
11:                 return result
12:             end if
13:             remove {var = value} from assignment
14:         else
15:             continue
16:         end if
17:     end for
18:     return failure
19: end function
```

If each of the following blocks of code were to replace line 15, which code block(s) would yield a correct algorithm for solving k-CSPS?

For this formulation, we want to only backtrack when more than k variables have violated a constraint. If there are fewer than k, we can assign the current variable to a constrained value, increment i, and continue the algorithm. The first code block provides the correct implementation. The second option using a tree structured CSP algorithm will incorrectly return failure if there is no solution with only $i + 1$ constraints violated (where $i + 1$ may be less than k). The third and fourth option filter the domains of the remaining variables. If a variable's domain is filtered, then the recursive-backtracking call will only consider values in the filtered domain rather than values which violate a constraint, potentially causing the code to miss the solution.

●

```
if $i < k$ then
    add {var = value} to assignment
    result ← Recursive-Backtracking(
                    assignment, csp, $i + 1$, $k$)
    if result $\neq$ failure then
        return result
    end if
    remove {var = value} from assignment
end if
```

○

```
if $i < k$ then
    add {var = value} to assignment
    if Is-Tree(Unassigned-Variables[csp]) then
        result ← Tree-Structured-CSP-Algorithm(
                    assignment, csp)
    else
        result ← Recursive-Backtracking(
                    assignment, csp, $i + 1$, $k$)
    end if
    if result $\neq$ failure then
        return result
    end if
    remove {var = value} from assignment
end if
```

○

```
if $i < k$ then
    add {var = value} to assignment
    Filter-Domains-with-Forward-Checking()
    result ← Recursive-Backtracking(
                    assignment, csp, $i + 1$, $k$)
    if result ≠ failure then
        return result
    end if
    Undo-Filter-Domains-with-Fwd-Checking()
    remove {var = value} from assignment
end if
```

○

```
if $i < k$ then
    add {var = value} to assignment
    Filter-Domains-with-Arc-Consistency()
    result ← Recursive-Backtracking(
                    assignment, csp, $i + 1$, $k$)
    if result ≠ failure then
        return result
    end if
    Undo-Filter-Domains-with-Arc-Consistency()
    remove {var = value} from assignment
end if
```

○ None of the code blocks

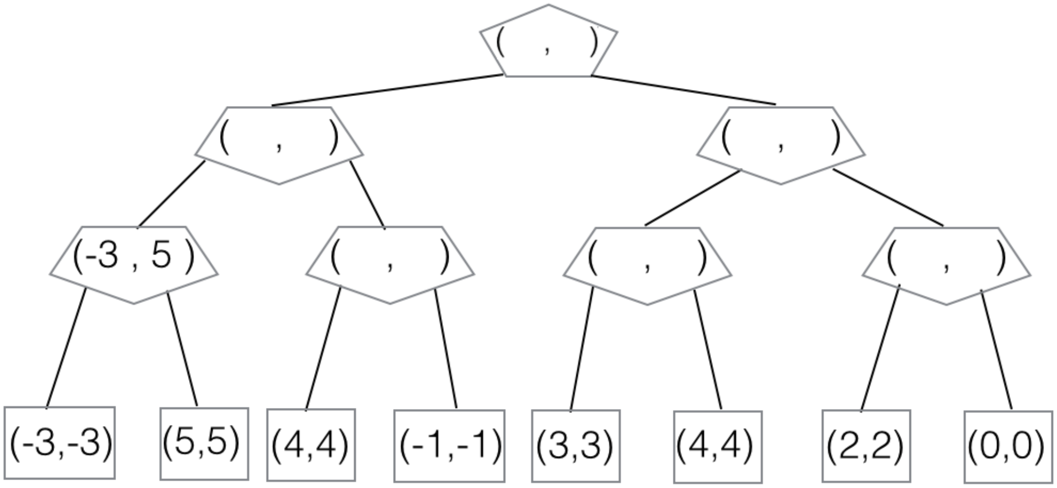
Q5. [12 pts] One Wish Pacman

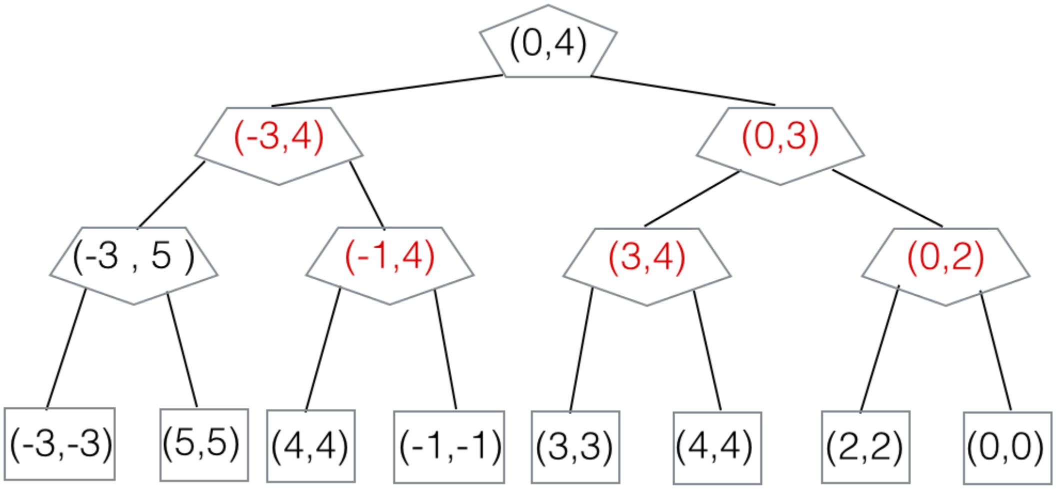
(a) **Power Search.** Pacman has a special power: *once* in the entire game when a ghost is selecting an action, Pacman can make the ghost choose any desired action instead of the min-action which the ghost would normally take. *The ghosts know about this special power and act accordingly.*

(i) [2 pts] 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 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.

(ii) [4 pts] 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
        $u \leftarrow$ 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 $= \max(uList)$ (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).

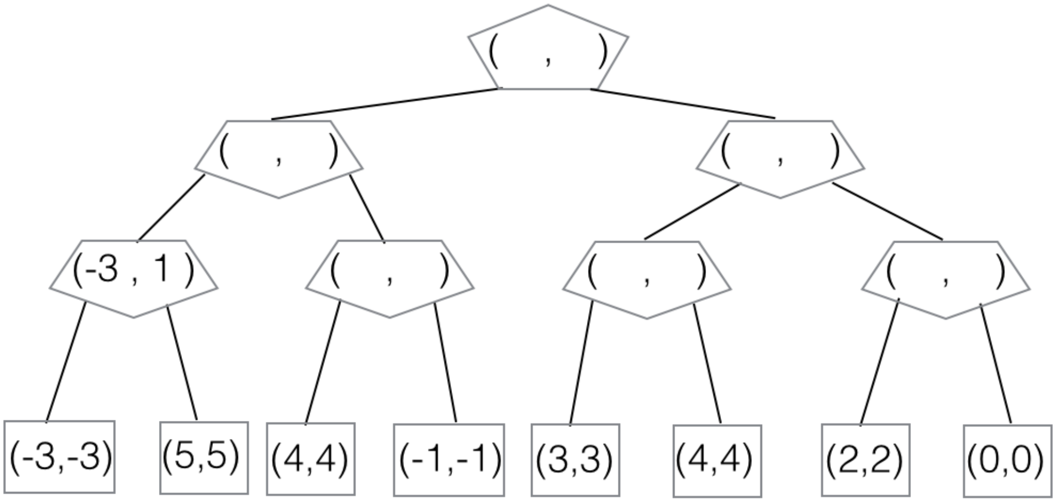
In case b), Pacman uses his power in one of the child subtrees so we consider the v values of the children, Since Pacman is not using his power on this node, the current node acts as a minimizer, making the value in case b) $= \min(vList)$

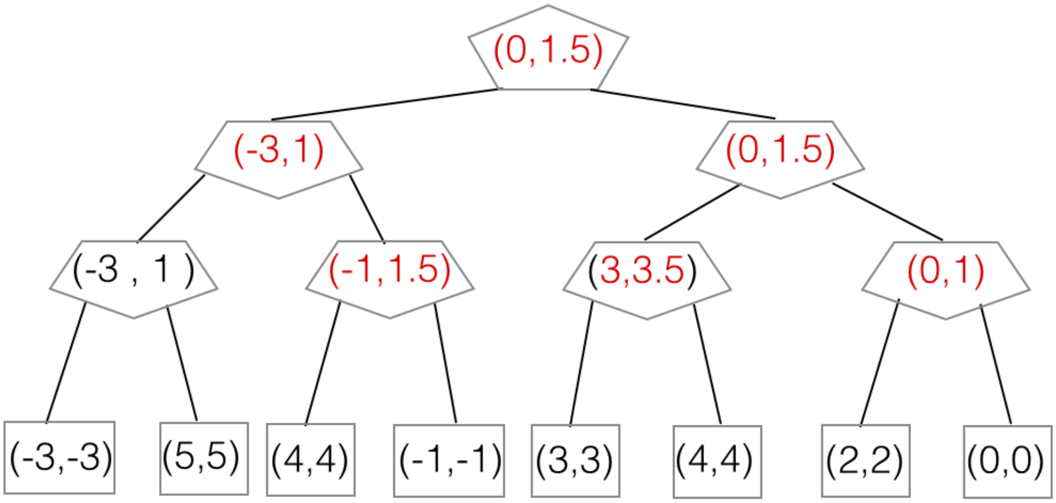
The v value at the current node is the best of the above two cases.

(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) [2 pts] 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) [4 pts] 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 $= avg(uList)$

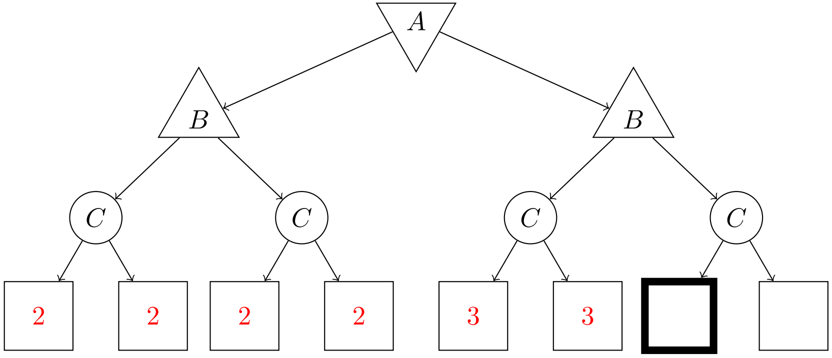
Q6. [9 pts] AlphaBetaExpinimax

In this question, player A is a minimizer, player B is a maximizer, and C represents a chance node. All children of a chance node are equally likely. Consider a game tree with players A, B, and C. In lecture, we considered how to prune a minimax game tree - in this question, you will consider how to prune an expinimax game tree (like a minimax game tree but with chance nodes). Assume that the children of a node are visited in left-to-right order.

For each of the following game trees, give an assignment of terminal values to the leaf nodes such that the bolded node can be pruned, or write "not possible" if no such assignment exists. You may give an assignment where an ancestor of the bolded node is pruned (since then the bolded node will never be visited). Your terminal values *must* be finite and you should not prune on equality. *Make your answer clear - if you write "not possible" the values in your tree will not be looked at.*

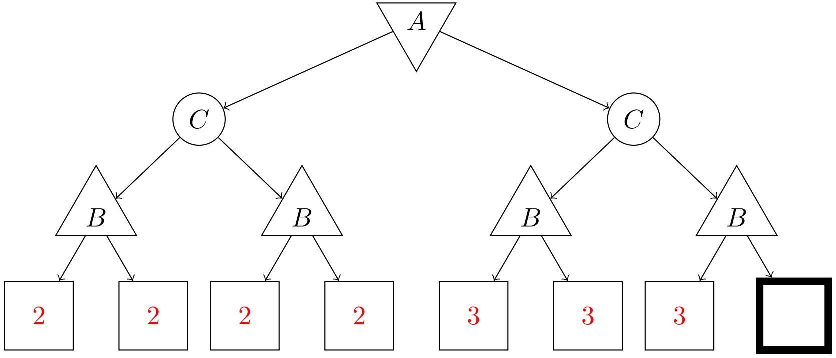
Important: The α-β pruning algorithm does not deal with chance nodes. Instead, for a node n, consider all the values seen so far, and determine whether you can know *without looking at the node* that the value of the node will not affect the value at the top of the tree. If that is the case, then n can be pruned.

(a) [3 pts]



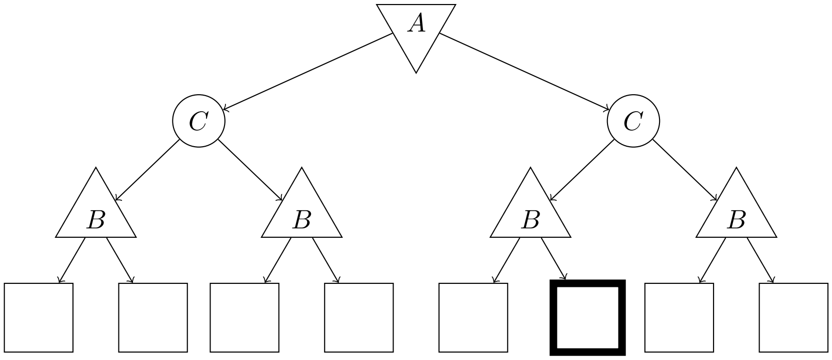
In this assignment, the values of the last two leaves do not matter. When we are about to look at the last chance node, we know that the minimizer A can guarantee a score of 2, but the maximizer B can guarantee a score of 3, and so the last chance node is pruned.

(b) [3 pts]



In this assignment, the value of the last leaf does not matter. When we are about to look at the bolded node, we know that the minimizer A can guarantee a score of 2, but the second chance node C is guaranteed to have a score of at least 3 (because all of the maximizers below it can guarantee a score of 3).

(c) [3 pts]



Not possible. At the bolded node, the minimizer can guarantee a score of at most the value of its left subtree. However, since we have not visited all the children of C, there is no bound on the value that C could attain. So, we need to continue exploring nodes until we can put a bound on C's value, which means that we have to explore the bolded node.

Q7. [11 pts] Lotteries in Ghost Kingdom

(a) Diverse Utilities. Ghost-King (GK) was once great friends with Pacman (P) because he observed that Pacman and he shared the same preference order among all possible event outcomes. Ghost-King, therefore, assumed that he and Pacman shared the same utility function. However, he soon started realizing that he and Pacman had a different preference order when it came to lotteries and, alas, this was the end of their friendship.

Let Ghost-King and Pacman's utility functions be denoted by U_{GK} and U_P respectively.

(i) [2 pts] Which of the following relations between U_{GK} and U_P are consistent with Ghost King's observation that U_{GK} and U_P agree with respect to all event outcomes but not all lotteries?

○ $U_P = aU_{GK} + b \quad (0 < a < 1, b > 0)$
○ $U_P = aU_{GK} + b \quad (a > 1, b > 0)$
● $U_P = U_{GK}^2$
● $U_P = \sqrt{(U_{GK})}$

For all the above options, U_P and U_{GK} result in the same preference order between two non-lottery events ($U_P(e_1) > U_P(e_2) \Leftrightarrow U_{GK}(e_1) > U_{GK}(e_2)$). However, options 1 and 2 also share the same preference order among all lotteries as well.

(ii) [2 pts] In addition to the above, Ghost-King also realized that Pacman was more risk-taking than him . Which of the relations between U_{GK} and U_P are possible?

○ $U_P = aU_{GK} + b \quad (0 < a < 1, b > 0)$
○ $U_P = aU_{GK} + b \quad (a > 1, b > 0)$
● $U_P = U_{GK}^2$
○ $U_P = \sqrt{(U_{GK})}$

As an example, say Ghost-King prefers winning \$2 as much as a lottery : winning \$0 or \$4 with equal probability. For option c), Pacman prefers the lottery much more (more risk-taking) and for option d), Pacman prefers the guaranteed reward.

(b) Guaranteed Return. Pacman often enters lotteries in the Ghost Kingdom. A particular Ghost vendor offers a lottery (for free) with three possible outcomes that are each equally likely: winning \$1, \$4, or \$5.

Let $U_P(m)$ denote Pacman's utility function for $\$m$. Assume that Pacman always acts rationally.

(i) [2 pts] The vendor offers Pacman a special deal - if Pacman pays \$1, the vendor will rig the lottery such that Pacman **always gets the highest reward possible**. For which of these utility functions would Pacman choose to pay the \$1 to the vendor for the rigged lottery over the original lottery? (Note that if Pacman pays \$1 and wins $\$m$ in the lottery, his actual winnings are $\$m$-1.)

● $U_P(m) = m$
● $U_P(m) = m^2$

a) $U_P(m) = m$:
If pacman does not pay, expected utility $= \frac{1}{3}(5) + \frac{1}{3}(4) + \frac{1}{3}(1) = \frac{10}{3}$
If pacman pays up, expected utility $= 1(5-1) + 0(4-1) + 0(1-1) = 4$.

b) $U_P(m) = m^2$:
If pacman does not pay, expected utility $= \frac{1}{3}(5)^2 + \frac{1}{3}(4)^2 + \frac{1}{3}(1)^2 = 14$
If pacman pays up, expected utility $= 1(5-1)^2 + 0(4-1)^2 + 0(1-1)^2 = 16$.

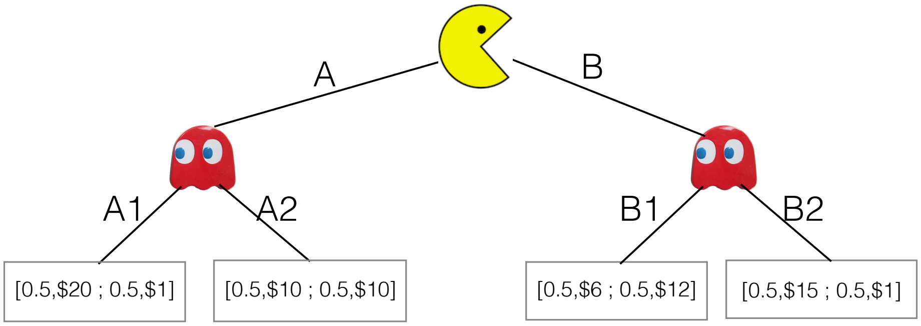
(ii) [2 pts] Now assume that the ghost vendor can only rig the lottery such that Pacman **never gets the lowest reward** and the remaining two outcomes become equally likely. For which of these utility functions would Pacman choose to pay the \$1 to the vendor for the rigged lottery over the original lottery?

● $U_P(m) = m$
○ $U_P(m) = m^2$

a) $U_P(m) = m$:
If pacman does not pay, expected utility $= \frac{1}{3}(5) + \frac{1}{3}(4) + \frac{1}{3}(1) = \frac{10}{3}$
If pacman pays up, expected utility $= \frac{1}{2}(5-1) + \frac{1}{2}(4-1) + 0(1-1) = 3.5$.

b) $U_P(m) = m^2$:
If pacman does not pay, expected utility $= \frac{1}{3}(5)^2 + \frac{1}{3}(4)^2 + \frac{1}{3}(1)^2 = 14$
If pacman pays up, expected utility $= \frac{1}{2}(5-1)^2 + \frac{1}{2}(4-1)^2 + 0(1-1)^2 = 12.5$.

(c) [3 pts] **Minimizing Other Utility.**



The Ghost-King, angered by Pacman's continued winnings, decided to revolutionize the lotteries in his Kingdom. There are now 4 lotteries (A1, A2, B1, B2), each with two equally likely outcomes. Pacman, who wants to maximize his expected utility, can pick one of two lottery types (A, B). The ghost vendor thinks that Pacman's utility function is $U'_P(m) = m$ and minimizes accordingly. However, Pacman's real utility function $U_P(m)$ may be different. For each of the following utility functions for Pacman, select the lottery corresponding to the outcome of the game.

Pacman's expected utility for the 4 lotteries, under various utility functions, are as follows :
$U_P(m) = m$: [A1 : 10.5; A2 : 10; B1 : 9; B2 : 8]
$U_P(m) = m^2$: [A1 : 200.5; A2 : 100; B1 : 90; B2 : 113]
$U_P(m) = \sqrt{m}$: [A1 : 2.74; A2 : 3.16; B1 : 2.96; B2 : 2.44]

(i) [1 pt] $U_P(m) = m$:

○ A1 ● A2 ○ B1 ○ B2

(ii) [1 pt] $U_P(m) = m^2$:

○ A1 ○ A2 ○ B1 ● B2

(iii) [1 pt] $U_P(m) = \sqrt{m}$:

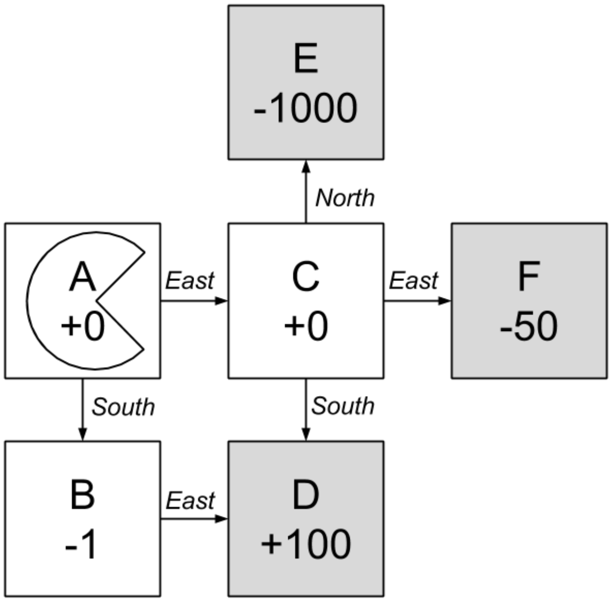
○ A1 ● A2 ○ B1 ○ B2

Since vendor minimizes $U'_P(m) = m$, If Pacman chooses A, vendor would pick A2 and if Pacman chooses B, vendor would pick B2. For $U_P(m) = m$ and $U_P(m) = \sqrt{m}$, Pacman prefers A2 over B2 and for $U_P(m) = m^2$, Pacman prefers B2 over A2 and acts accordingly.

Q8. [13 pts] Indecisive Pacman

(a) Simple MDP

Pacman is an agent in a deterministic MDP with states A, B, C, D, E, F. He can deterministically choose to follow any edge pointing out of the state he is currently in, corresponding to an action $North$, $East$, or $South$. He cannot stay in place. D, E, and F are terminal states. Let the discount factor be $\gamma = 1$. Pacman receives the reward value labeled underneath a state upon entering that state.



(i) [3 pts] Write the optimal values $V^*(s)$ for $s = A$ and $s = C$ and the optimal policy $\pi^*(s)$ for $s = A$.

$V^*(A)$: 100 $V^*(C)$: 100

$\pi^*(A)$: East

The optimal plan without indecisiveness is to go East and then South, collecting no negative rewards.

(ii) [2 pts] Pacman is typically rational, but now becomes indecisive if he enters state C. In state C, he finds the two best actions and randomly, with equal probability, chooses between the two. Let $\bar{V}(s)$ be the values under the policy where Pacman acts according to $\pi^*(s)$ for all $s \neq C$, and follows the indecisive policy when at state C. What are the values $\bar{V}(s)$ for $s = A$ and $s = C$?

$\bar{V}(A)$: 25 $\bar{V}(C)$: 25

Pacman is unaware of his indecisiveness at state C. So he will follow his policy π^* from (i) at state A and go East. When he has reached C, his two best actions are going South and going East. He will then receive the average of the value of taking those two actions, $(100 - 50)/2 = 25$, since he will take either one with equal probability.

(iii) [2 pts] Now Pacman knows that he is going to be indecisive when at state C and decides to recompute the optimal policy at all other states, anticipating his indecisiveness at C. What is Pacman's new policy $\tilde{\pi}(s)$ and new value $\tilde{V}(s)$ for $s = A$?

$\tilde{\pi}(A)$: South $\tilde{V}(A)$: 99

Now Pacman knows about his indecisiveness as C, and so he can anticipate the low value (25) of being in C. He can therefore choose to go South and receive a reward of -1 and from there go East, to get a reward of 100, making for a value from A of 99.

(b) General Case – Indecisive everywhere

Pacman enters a new non-deterministic MDP and has become indecisive in all states of this MDP: at every time-step, instead of being able to pick a single action to execute, he always picks the two distinct best actions and then flips a fair coin to randomly decide which action to execution from the two actions he picked.

Let S be the state space of the MDP. Let $A(s)$ be the set of actions available to Pacman in state s. Assume for simplicity that there are always at least two actions available from each state ($|A(s)| \geq 2$).

This type of agent can be formalized by modifying the Bellman Equation for optimality. Let $\hat{V}(s)$ be the value of the indecisive policy. Precisely:

$$\hat{V}(s_0) = E[R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3)) + \dots]$$

Let $\hat{Q}(s, a)$ be the expected utility of taking action a from state s and then following the indecisive policy after that step. We have that:

$$\hat{Q}(s, a) = \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma \hat{V}(s'))$$

(i) [3 pts] Which of the following options gives $\hat{V}$ in terms of $\hat{Q}$? When combined with the above formula for $\hat{Q}(s, a)$ in terms of $\hat{V}(s')$, the answer to this question forms the Bellman Equation for this policy.

$\hat{V}(s) =$

○ $\max_{a \in A(s)} \hat{Q}(s, a)$ ○ $\max_{a_1 \in A(s)} \max_{a_2 \in A(s),\ a_1 \neq a_2} (\hat{Q}(s, a_1) \cdot \hat{Q}(s, a_2))$

● $\max_{a_1 \in A(s)} \max_{a_2 \in A(s),\ a_1 \neq a_2} \frac{1}{2}(\hat{Q}(s, a_1) + \hat{Q}(s, a_2))$ ○ $\max_{a_1 \in A(s)} \sum_{a_2 \in A(s),\ a_1 \neq a_2} (\hat{Q}(s, a_1) \cdot \hat{Q}(s, a_2))$

○ $\sum_{a_1 \in A(s)} \sum_{a_2 \in A(s),\ a_1 \neq a_2} (\hat{Q}(s, a_1) \cdot \hat{Q}(s, a_2))$ ○ $\sum_{a_1 \in A(s)} \sum_{a_2 \in A(s),\ a_1 \neq a_2} \frac{1}{2}(\hat{Q}(s, a_1) + \hat{Q}(s, a_2))$

○ $\max_{a_1 \in A(s)} \sum_{a_2 \in A(s),\ a_1 \neq a_2} \frac{1}{2}(\hat{Q}(s, a_1) + \hat{Q}(s, a_2))$

○ $\frac{1}{|A(s)|(|A(s)| - 1)} \sum_{a_1 \in A(s)} \sum_{a_2 \in A(s),\ a_1 \neq a_2} (\hat{Q}(s, a_1) \cdot \hat{Q}(s, a_2))$

○ $\frac{1}{|A(s)|(|A(s)| - 1)} \sum_{a_1 \in A(s)} \sum_{a_2 \in A(s),\ a_1 \neq a_2} \frac{1}{2}(\hat{Q}(s, a_1) + \hat{Q}(s, a_2))$

○ $\max_{a_1 \in A(s)} \frac{1}{|A(s)| - 1} \sum_{a_2 \in A(s),\ a_1 \neq a_2} \frac{1}{2}(\hat{Q}(s, a_1) + \hat{Q}(s, a_2))$

○ $\max_{a_1 \in A(s)} \frac{1}{|A(s)| - 1} \sum_{a_2 \in A(s),\ a_1 \neq a_2} (\hat{Q}(s, a_1) \cdot \hat{Q}(s, a_2))$

○ None of the above. Pacman must select the best two actions (two maxes), and then flip a coin to determine which to perform (average their values).

(ii) [3 pts] Which of the following equations specify the relationship between V^* and $\hat{V}$ in general?

○ $2V^*(s) = \hat{V}(s)$ ○ $V^*(s) = 2\hat{V}(s)$ ○ $(V^*(s))^2 = |\hat{V}(s)|$ ○ $|V^*(s)| = (\hat{V}(s))^2$

○ $\frac{1}{|A(s)|} \sum_{a \in A(s)} \sum_{s' \in S} T(s, a, s')\hat{V}(s') = V^*(s)$ ○ $\frac{1}{|A(s)|} \sum_{a \in A(s)} \sum_{s' \in S} T(s, a, s')V^*(s') = \hat{V}(s)$

○ $\frac{1}{|A(s)|} \sum_{a \in A(s)} \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma V^*(s')) = \hat{V}(s)$

○ $\frac{1}{|A(s)|} \sum_{a \in A(s)} \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma \hat{V}(s')) = V^*(s)$

● None of the above. None of the above are valid relationships between V^* and $\hat{V}$. It may be possible to construct MDPs where some of the above are satisfied, but they won't be satisfied for all MDPs.

Q9. [8 pts] Reinforcement Learning

Imagine an unknown game which has only two states $\{A, B\}$ and in each state the agent has two actions to choose from: {Up, Down}. Suppose a game agent chooses actions according to some policy π and generates the following sequence of actions and rewards in the unknown game:

t	s_t	a_t	s_{t+1}	r_t
0	A	Down	B	2
1	B	Down	B	-4
2	B	Up	B	0
3	B	Up	A	3
4	A	Up	A	-1

Unless specified otherwise, assume a discount factor $\gamma = 0.5$ and a learning rate $\alpha = 0.5$

(a) [2 pts] Recall the update function of Q-learning is:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a'))$$

Assume that all Q-values initialized as 0. What are the following Q-values learned by running Q-learning with the above experience sequence?

$Q(\text{A}, \text{Down}) =$ ___1___, $\quad Q(\text{B}, \text{Up}) =$ ___$\frac{7}{4}$___

Perform Q-learning update 4 times, once for each of the first 4 observations.

(b) [2 pts] In model-based reinforcement learning, we first estimate the transition function $T(s, a, s')$ and the reward function $R(s, a, s')$. Fill in the following estimates of T and R, estimated from the experience above. Write "n/a" if not applicable or undefined.

$\hat{T}(\text{A}, \text{Up}, \text{A}) =$ ___1___, $\hat{T}(\text{A}, \text{Up}, \text{B}) =$ ___0___, $\hat{T}(\text{B}, \text{Up}, \text{A}) =$ ___$\frac{1}{2}$___, $\hat{T}(\text{B}, \text{Up}, \text{B}) =$ ___$\frac{1}{2}$___

$\hat{R}(\text{A}, \text{Up}, \text{A}) =$ ___-1___, $\hat{R}(\text{A}, \text{Up}, \text{B}) =$ ___n/a___, $\hat{R}(\text{B}, \text{Up}, \text{A}) =$ ___3___, $\hat{R}(\text{B}, \text{Up}, \text{B}) =$ ___0___

Count transitions above and calculate frequencies. Rewards are observed rewards.

(c) To decouple this question from the previous one, assume we had **a different experience** and ended up with the following estimates of the transition and reward functions:

s	a	s'	$\hat{T}(s, a, s')$	$\hat{R}(s, a, s')$
A	Up	A	1	10
A	Down	A	0.5	2
A	Down	B	0.5	2
B	Up	A	1	-5
B	Down	B	1	8

(i) [2 pts] Give the optimal policy $\hat{\pi}^*(s)$ and $\hat{V}^*(s)$ for the MDP with transition function $\hat{T}$ and reward function $\hat{R}$.
Hint: for any $x \in \mathbb{R}$, $|x| < 1$, we have $1 + x + x^2 + x^3 + x^4 + \cdots = 1/(1 - x)$.

$\hat{\pi}^*(A) =$ ___*Up*___, $\hat{\pi}^*(B) =$ ___*Down*___, $\hat{V}^*(A) =$ ___20___, $\hat{V}^*(B) =$ ___16___.

Find the optimal policy first, and then use optimal policy to calculate the value function using a Bellman equation.

(ii) [2 pts] If we repeatedly feed this new experience sequence through our Q-learning algorithm, what values will it converge to? Assume the learning rate α_t is properly chosen so that convergence is guaranteed.

● the values found above, $\hat{V}^*$

○ the optimal values, V^*

○ neither $\hat{V}^*$ nor V^*

○ not enough information to determine

The Q-learning algorithm will not converge to the optimal values V^* for the MDP because the experience sequence and transition frequencies replayed are not necessarily representative of the underlying MDP. (For example, the true $T(A, Down, A)$ might be equal to 0.75, in which case, repeatedly feeding in the above experience would not provide an accurate sampling of the MDP.) However, for the MDP with transition function $\hat{T}$ and reward function $\hat{R}$, replaying this experience repeatedly will result in Q-learning converging to its optimal values $\hat{V}^*$.

Q10. [14 pts] Potpourri

(a) Each True/False question is worth 2 points. Leaving a question blank is worth 0 points. **Answering incorrectly is worth -2 points.**

(i) [2 pts] [*true* or [✓ *false*]] There exists some value of $c > 0$ such that the heuristic $h(n) = c$ is admissible.
This heuristic is non-zero at the goal state, and so it cannot be admissible.

(ii) [2 pts] [[✓ *true*] or *false*] A^* tree search using the heuristic $h(n) = c$ for some $c > 0$ is guaranteed to find the optimal solution.
Each state in the fringe has priority $f(n) = g(n) + h(n) = g(n) + c$. Only the *ordering* of the nodes in the fringe matters, and so adding a constant c to all priorities makes no difference. So, this is equivalent to using $f(n) = g(n)$, which is UCS, which will find the optimal solution.

(b) [2 pts] Consider a one-person game, where the one player's actions have non-deterministic outcomes. The player gets +1 utility for winning and -1 for losing. Mark *all* of the approaches that can be used to model and solve this game.

○ Minimax with terminal values equal to +1 for wins and -1 for losses

● Expectimax with terminal values equal to +1 for wins and -1 for losses

● Value iteration with all rewards set to 0, except wins and losses, which are set to +1 and -1

○ None of the above

Minimax is not a good fit, because there is no minimizer - there is only a maximizer (the player) and chance nodes (the non-deterministic actions). The existence of a maximizer and chance nodes means that this is particularly suited to expectimax and value iteration.

(c) [4 pts] Pacman is offered a choice between (a) playing against 2 ghosts or (b) a lottery over playing against 0 ghosts or playing against 4 ghosts (which are equally likely). Mark the rational choice according to each utility function below; if it's a tie, mark so. Here, g is the number of ghosts Pacman has to play against.

(i)	$U(g) = g$	○ 2 ghosts	○ lottery between 0 and 4 ghosts	● tie
(ii)	$U(g) = -(2^g)$	● 2 ghosts	○ lottery between 0 and 4 ghosts	○ tie
(iii)	$U(g) = 2^{(-g)} = \frac{1}{2^g}$	○ 2 ghosts	● lottery between 0 and 4 ghosts	○ tie
(iv)	$U(g) = 1$ if $g < 3$ else 0	● 2 ghosts	○ lottery between 0 and 4 ghosts	○ tie

For $U(g) = g$, we get $U(2) = 2$ and $U([4, 0.5; 0, 0.5]) = 0.5U(4) + 0.5U(0) = 0.5(4) + 0.5(0) = 2$. Since $2 = 2$, Pacman is indifferent.

For $U(g) = -2^g$, we get $U(2) = -2^2 = -4$ and $U([4, 0.5; 0, 0.5]) = 0.5U(4) + 0.5U(0) = 0.5(-16) + 0.5(-1) = -8.5$. Since $-4 > -8.5$, Pacman prefers the 2 ghosts.

For $U(g) = 2^{-g}$, we get $U(2) = \frac{1}{4}$ and $U([4, 0.5; 0, 0.5]) = 0.5U(4) + 0.5U(0) = 0.5(\frac{1}{16}) + 0.5(1) = \frac{17}{32}$. Since $\frac{17}{32} > \frac{1}{4}$, Pacman prefers the lottery.

For $U(g) = 1$ if $g < 3$ else 0, we get $U(2) = 1$ and $U([4, 0.5; 0, 0.5]) = 0.5U(4) + 0.5U(0) = 0.5(0) + 0.5(1) = 0.5$. Since $1 > 0.5$, Pacman prefers the 2 ghosts.

(d) Suppose we run value iteration in an MDP with only non-negative rewards (that is, $R(s, a, s') \geq 0$ for any (s, a, s')). Let the values on the kth iteration be $V_k(s)$ and the optimal values be $V^*(s)$. Initially, the values are 0 (that is, $V_0(s) = 0$ for any s).

(i) [1 pt] Mark *all* of the options that are *guaranteed* to be true.

○ For any s, a, s', $V_1(s) = R(s, a, s')$

○ For any s, a, s', $V_1(s) \leq R(s, a, s')$

○ For any s, a, s', $V_1(s) \geq R(s, a, s')$

● None of the above are guaranteed to be true.

$V_1(s) = \max_a \sum_{s'} T(s,a,s')R(s,a,s')$ (using the Bellman equation and setting $V_0(s') = 0$).

Now consider an MDP where the best action in state X is clockwise, which goes to state Y with a reward of 6 with probability 0.5 and goes to state Z a reward of 4 with probability 0.5. Then $V_1(X) = 0.5(6) + 0.5(4) = 5$. Notice that setting $(s,a,s') = (X, clockwise, Z)$ gives a counterexample for the second option and $(s,a,s') = (X, clockwise, Y)$ gives a counterexample for the third option.

(ii) [1 pt] Mark *all* of the options that are *guaranteed* to be true.

○ For any k, s, $V_k(s) = V^*(s)$

● For any k, s, $V_k(s) \leq V^*(s)$

○ For any k, s, $V_k(s) \geq V^*(s)$

○ None of the above are guaranteed to be true.

Intuition: Values can never decrease in an iteration. In the first iteration, since all rewards are positive, the values increase. In any other iteration, the components that contribute to $V_{k+1}(s)$ are $R(s,a,s')$ and $V(s')$. $R(s,a,s')$ is the same across all iterations, and $V(s')$ increased in the previous iteration, so we expect $V_{k+1}(s)$ to increase as well.

More formally, we can prove $V_k(s) \leq V_{k+1}(s)$ by induction.

Base Case: $V_1(s) = \max_a \sum_{s'} T(s,a,s')R(s,a,s')$.

Since $R(s,a,s') \geq 0$, $T(s,a,s') \geq 0$, we have $V_1(s) \geq 0$, and so $V_0(s) \leq V_1(s)$.

Induction: $V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$

$\geq \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_{k-1}(s')]$ (using $V_k(s') \geq V_{k-1}(s')$ from the inductive hypothesis)

$= V_k(s)$.

This immediately leads to $V_k(s) \leq V^*(s)$ (since we can think of $V^*(s)$ as $V_\infty(s)$).

(e) [2 pts] Consider an arbitrary MDP where we perform Q-learning. Mark *all* of the options below in which we are guaranteed to learn the *optimal* Q-values. Assume that the learning rate α is reduced to 0 appropriately.

○ During learning, the agent acts according to a suboptimal policy π. The learning phase continues until convergence.

● During learning, the agent chooses from the available actions at random. The learning phase continues until convergence.

● During learning, in state s, the agent chooses the action a that it has chosen least often in state s, breaking ties randomly. The learning phase continues until convergence.

○ During learning, in state s, the agent chooses the action a that it has chosen most often in state s, breaking ties randomly. The learning phase continues until convergence.

○ During learning, the agent always chooses from the available actions at random. The learning phase continues until each (s,a) pair has been seen at least 10 times.

In order for Q-learning to converge to the *optimal* Q-values, we need every (s,a) pair to be visited infinitely often. Option 5 only does this 10 times, whereas options 1 and 4 choose only one of the many actions possible for a given state s. Only options 2 and 3 visit each (s,a) pair infinitely often.

THIS PAGE IS INTENTIONALLY LEFT BLANK