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.
- For multiple choice questions with *circular bubbles*, you should only mark ONE option; for those with *checkboxes*, you should mark ALL that apply (which can range from zero to all options)

First name	
Last name	
edX username	

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:

(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:

(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:

actions between states:

cost function $c(s, s')$ between neighboring states:

(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:

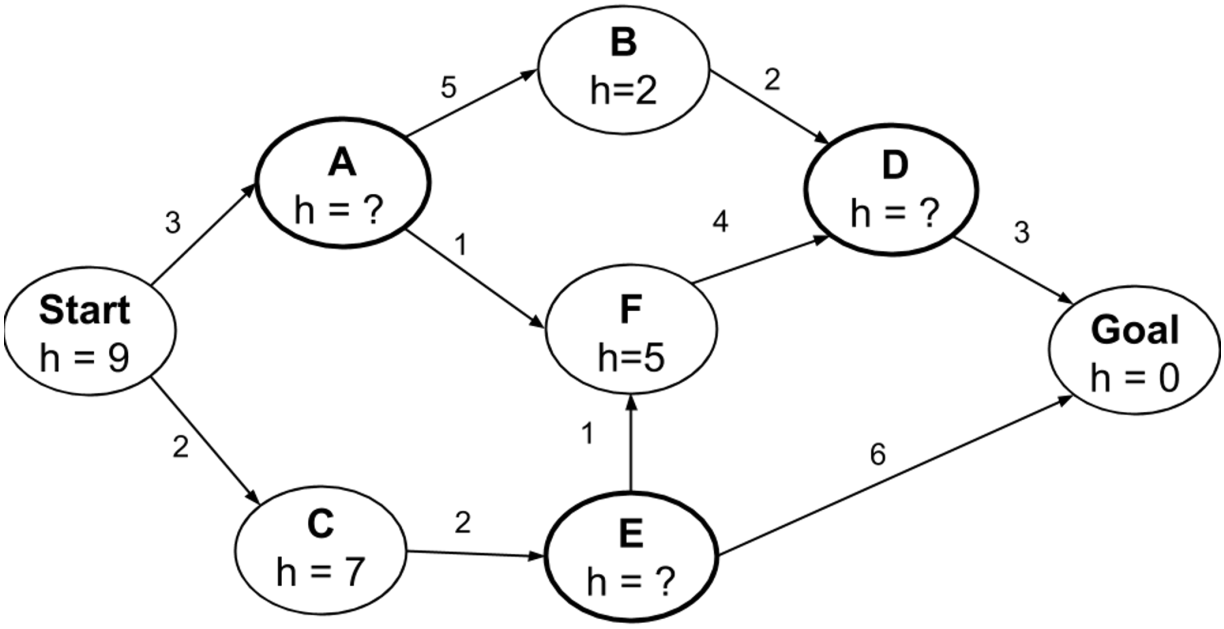
minimal state representation:

actions between states:

cost function $c(s, s')$ between neighboring states:

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	$\leq h(A) \leq$
D	$\leq h(D) \leq$
E	$\leq h(E) \leq$

Q3. [10 pts] PAC-CORP Assignments

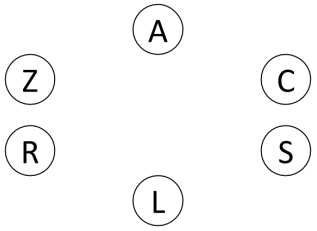
Your CS188 TAs have all secured jobs at PAC-CORP. Now, PAC-CORP must assign each person to exactly one team. The TAs are Alvin (A), Chelsea (C), Lisa (L), Rohin (R), Sandy (S), and Zoe (Z). We would like to formulate this as a CSP using one variable for each TA. The teams to choose from are:

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 (note that doing so only involves the binary constraints).



(b) [2 pts] On the grid below, cross out values that are removed from the domains 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

(c) [2 pts] Consider the filtered domains obtained in part (b) from enforcing the unary constraints. According to Minimum Remaining Values (MRV), which variable should be selected?

◯ A ◯ C ◯ S ◯ L ◯ R ◯ Z

(d) [3 pts] Assume a current set of filtered domains as shown below. Cross off the values that are eliminated by 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

Q4. [5 pts] k-CSPs

Let a k-CSP be a CSP where the solution is allowed to have k violated 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 violated constraints.

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

```
1: function Recursive-Backtracking(assignment, csp, 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, k)
10:            if result ≠ 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?

☐
```
add {var = value} to assignment
n = Get-Total-Number-of-Constraints-Violated(
                    assignment, csp)
if n ≤ k then
    result ← Recursive-Backtracking(
                        assignment, csp, k)
    if result ≠ failure then
        return result
    end if
end if
remove {var = value} from assignment
```

☐
```
add {var = value} to assignment
n = Get-Total-Number-of-Constraints-Violated(
                    assignment, csp)
if n ≤ k then
    if Is-Tree(Unassigned-Variables[csp]) then
        result ← Tree-Structured-CSP-Algorithm(
                            assignment, csp)
    else
        result ← Recursive-Backtracking(
                            assignment, csp, k)
    end if
    if result ≠ failure then
        return result
    end if
end if
remove {var = value} from assignment
```

☐
```
add {var = value} to assignment
n = Get-Total-Number-of-Constraints-Violated(
                    assignment, csp)
if n ≤ k then
    Filter-Domains-with-Forward-Checking()
    result ← Recursive-Backtracking(
                        assignment, csp, k)
    if result ≠ failure then
        return result
    end if
    Undo-Filter-Domains-with-Fwd-Checking()
end if
remove {var = value} from assignment
```

☐
```
add {var = value} to assignment
n = Get-Total-Number-of-Constraints-Violated(
                    assignment, csp)
if n ≤ k then
    Filter-Domains-with-Arc-Consistency()
    result ← Recursive-Backtracking(
                        assignment, csp, k)
    if result ≠ failure then
        return result
    end if
    Undo-Filter-Domains-with-Arc-Consistency()
end if
remove {var = value} from assignment
```

☐ None of the code blocks

Q5. [12 pts] One Wish Pacman

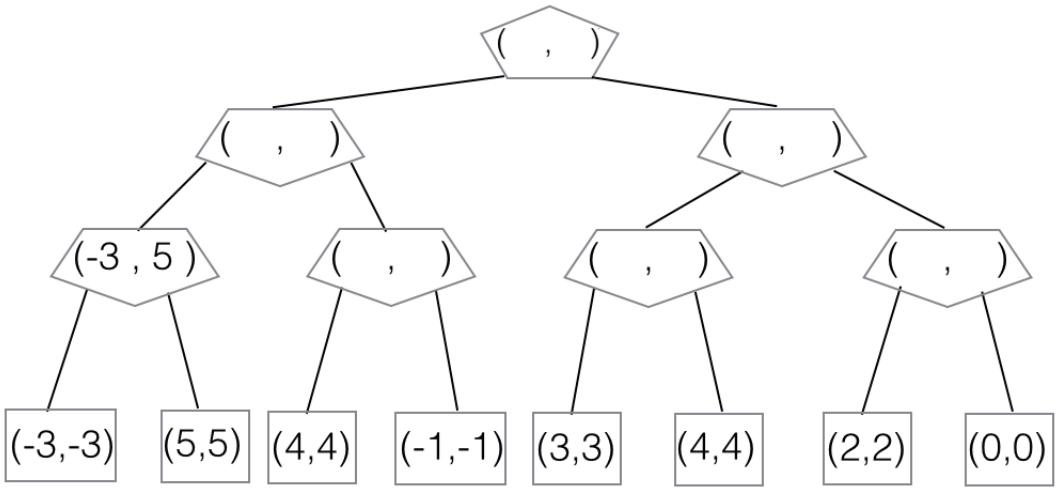
(a) Power Search. Pacman has a special power: *once* in the entire game when a ghost is selecting an action, Pacman can make the ghost choose any desired action instead of the min-action which the ghost would normally take. *The ghosts know about this special power and act accordingly.*

(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 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.



(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 ← 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
```

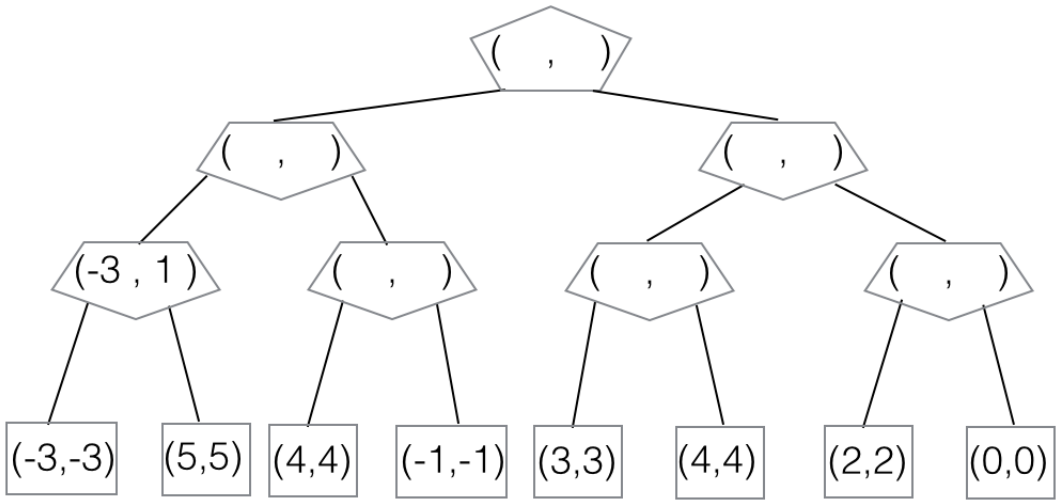
```
function MAX-VALUE(state)
    uList ← [ ], vList ← [ ]
    for successor in SUCCESSORS(state) do
        (u′, v′) ← VALUE(successor)
        uList.append(u′)
        vList.append(v′)
    end for
    u ← max(uList)
    v ← max(vList)
    return (u, v)
end function
```

```
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 ← _____

    v ← _____

    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) [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.



(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 ← 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 ← _____

    v ← _____

    return (u, v)
end function
```

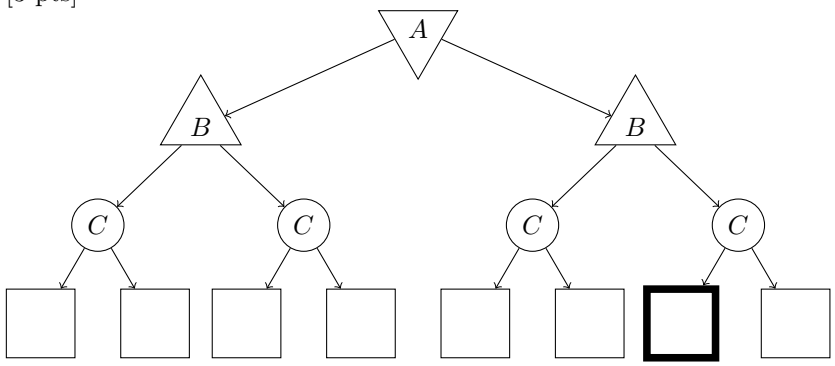
Q6. [9 pts] AlphaBetaExpinimax

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.

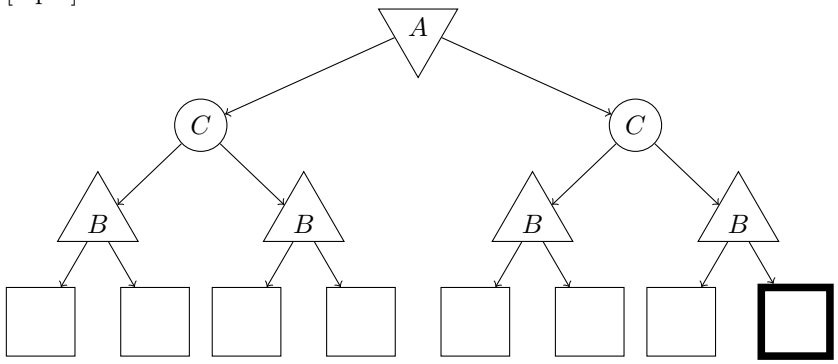
For each of the following game trees, give an assignment of terminal values to the leaf nodes such that the bolded node can be pruned (it doesn't matter if you prune more nodes), or write "not possible" if no such assignment exists. You may give an assignment where an ancestor of the bolded node is pruned (since then the bolded node will never be visited). You should not prune on equality, and your terminal values *must* be finite (including negative values). *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.

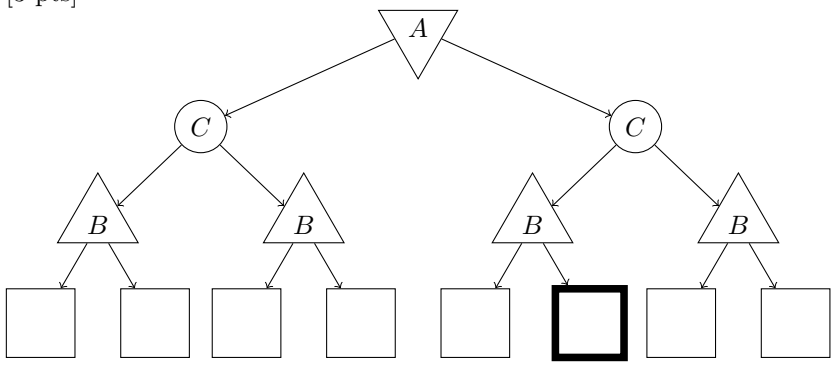
(a) [3 pts]



(b) [3 pts]



(c) [3 pts]



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. Assume both U_{GK} and U_P are guaranteed to output non-negative values.

(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?

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

(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?

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

(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.

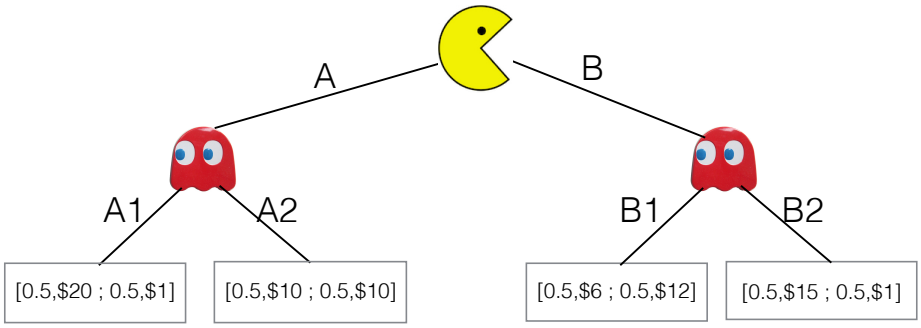
(i) [2 pts] The vendor offers Pacman a special deal - if Pacman pays \$1, the vendor will manipulate the lottery such that Pacman **always gets the highest reward possible**. For which of these utility functions would Pacman choose to pay the \$1 to the vendor for the manipulated lottery over the original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

- ☐ $U_P(m) = m$
- ☐ $U_P(m) = m^2$

(ii) [2 pts] Now assume that the ghost vendor can only manipulate 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 manipulated lottery over the original lottery?

- ☐ $U_P(m) = m$
- ☐ $U_P(m) = m^2$

(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. Note that Pacman knows how the ghost vendor is going to behave.

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

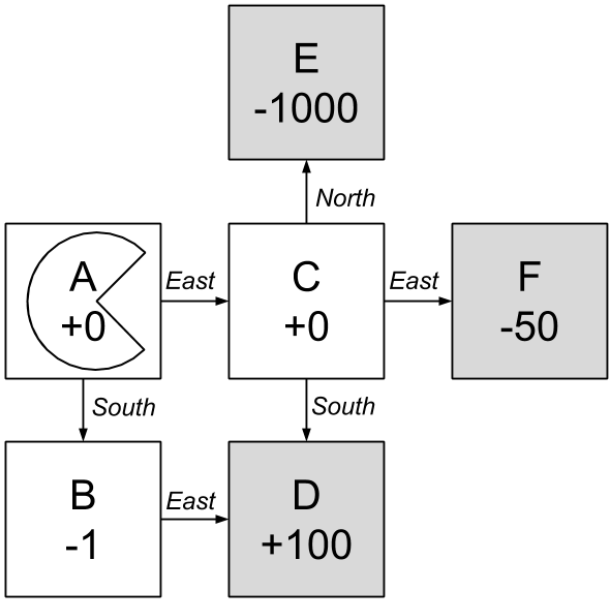
(iii) [1 pt] $U_P(m) = \sqrt{m}$:

◯ A1 ◯ A2 ◯ B1 ◯ B2

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 $s = A$.

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

$\pi^*(A)$: _____

(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): _____ $\bar{V}$ (C): _____

(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): _____ $\tilde{V}$ (A): _____

(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.

(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.

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?

$$Q(\text{A}, \text{Down}) = \_\_\_\_\_, \qquad Q(\text{B}, \text{Up}) = \_\_\_\_\_$$

(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.

$$\hat{T}(\text{A}, \text{Up}, \text{A}) = \_\_\_\_\_, \quad \hat{T}(\text{A}, \text{Up}, \text{B}) = \_\_\_\_\_, \quad \hat{T}(\text{B}, \text{Up}, \text{A}) = \_\_\_\_\_, \quad \hat{T}(\text{B}, \text{Up}, \text{B}) = \_\_\_\_\_$$

$$\hat{R}(\text{A}, \text{Up}, \text{A}) = \_\_\_\_\_, \quad \hat{R}(\text{A}, \text{Up}, \text{B}) = \_\_\_\_\_, \quad \hat{R}(\text{B}, \text{Up}, \text{A}) = \_\_\_\_\_, \quad \hat{R}(\text{B}, \text{Up}, \text{B}) = \_\_\_\_\_$$

(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 $1 + x + x^2 + x^3 + x^4 + \cdots = 1/(1-x)$.

$$\hat{\pi}^*(A) = \_\_\_\_\_, \qquad \hat{\pi}^*(B) = \_\_\_\_\_, \qquad \hat{V}^*(A) = \_\_\_\_\_, \qquad \hat{V}^*(B) = \_\_\_\_\_.$$

(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

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 $k > 0$ such that the heuristic $h(n) = k$ is admissible.

(ii) [2 pts] [*true* or *false*] A^* tree search using the heuristic $h(n) = k$ for some $k > 0$ is guaranteed to 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

(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

(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.

(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.

(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.

THIS PAGE IS INTENTIONALLY LEFT BLANK