CS 188 Fall 2019 — Introduction to Artificial Intelligence — Practice Midterm

- You have approximately 110 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 *check-boxes*, you should mark ALL that apply (which can range from zero to all options). FILL in your answer COMPLETELY

First name	
Last name	
SID	
Name of person on your left	
Name of person on your right	

Your Discussion/Exam Prep* TA (fill all that apply):

- ☐ Shizhan (Tu)
- ☐ Carl (Tu)
- ☐ Emma (Tu)
- ☐ Mesut* (Tu)
- ☐ Jesse (Tu)
- ☐ Cathy (Tu)
- ☐ Peyrin* (Tu)
- ☐ Andy (Tu)
- ☐ Wilson (W)
- ☐ Ryan (W)
- ☐ Lindsay (W)
- ☐ Gokul* (W)
- ☐ Rachel (W)
- ☐ Henry* (W)
- ☐ Alan (W)
- ☐ Andreea (W)
- ☐ Chandan (W)
- ☐ Sherman* (W)
- ☐ Mike (W)
- ☐ Danny* (W)
- ☐ Jinkyu (W)
- ☐ Lawrence (W)
- ☐ Albert (W)

For staff use only:

Q1.	Potpourri	/10
Q2.	Pushing Boxes	/12
Q3.	Search Algorithms	/12
Q4.	CSPs: Potluck Pandemonium	/13
Q5.	Variants of Trees	/8
Q6.	Reward Shaping	/21
Q7.	Q-uagmire	/13
	Total	/89

To earn the extra credit, one of the following has to hold true. Please circle and sign.

A I spent 110 or more minutes on the practice midterm.

B I spent fewer than 110 minutes on the practice midterm, but I believe I have solved all the questions.

Signature: _____

To submit the practice midterm, scan and upload the PDF to Gradescope.

SID: _____

Q1. [10 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) [2.0 pts] [*true* or *false*] There exists some value of $k > 0$ such that the heuristic $h(n) = k$ is admissible.
This heuristic is non-zero at the goal state, and so it cannot be admissible.

(ii) [2.0 pts] [*true* or *false*] A^* tree search using the heuristic $h(n) = k$ for some $k > 0$ is guaranteed to find the optimal solution.
Each state in the fringe has priority $f(n) = g(n) + h(n) = g(n) + k$. Only the *ordering* of the nodes in the fringe matters, and so adding a constant k 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) 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 $V_\infty(s)$).

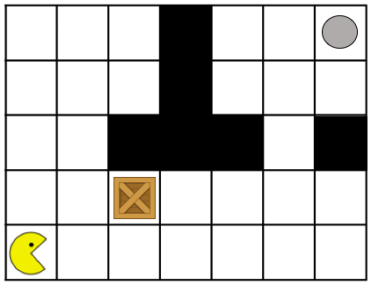
(d) [2 pts] Consider an arbitrary MDP where we perform Q-learning. Mark *all* of the options below in which we are guaranteed to learn the *optimal* Q-values. Assume that the learning rate α is reduced to 0 appropriately.

- ☐ 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.

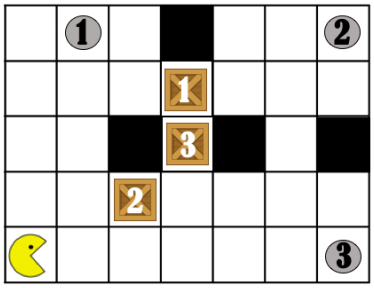
In order for Q-learning to converge to the *optimal* Q-values, we need every (s,a) pair to be visited infinitely often. Option 5 only does this 10 times, whereas options 1 and 4 choose only one of the many actions possible for a given state s. Only options 2 and 3 visit each (s,a) pair infinitely often.

SID: _____

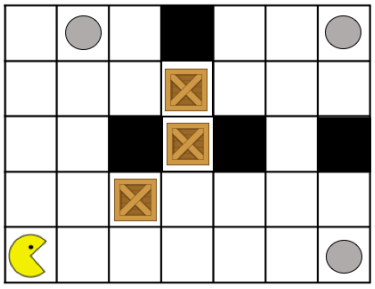
Q2. [12 pts] Pushing Boxes



(a) One box



(b) Numbered boxes and buttons



(c) Any box to any button

Pacman has to solve several levels of mazes by pushing boxes to circular buttons in the maze. Obviously, Pacman can only push a box (he does not have hands to pull it!). Pacman pushes a box by standing behind it and moving into its position. Pacman is not strong enough to push more than one box at a time. You can assume that the maze is $M \times N$ and that initially no box is upon any button. At each timestep, Pacman can just move either up, down, left, or right if he does not collide with any wall or the box that Pacman is pushing does not collide. Each action has a cost of 1. Actions that do not result in Pacman or a box being moved still have cost of 1. The figures display a possible configuration for each maze.

Note that for all parts of this question, d_{Man} is the Manhattan distance.

(a) In the first level, Pacman has to push a single box to a specific button (Figure 1a).

(i) [2 pts] What is the size of the minimal state space? Express your answer using the symbols M and N.

$(MN)^2$ The mininmal state space corresponds to the position of the Pacman and the position of the box. Since each of them can be in MN positions the size of the state space is $(MN)^2$.

(ii) [2 pts] What is the branching factor? The answer should be a whole, positive number.

4 Pacman has 4 actions and the dynamics are deterministic. Therefore, the branching factor is 4.

(b) In the next level things get trickier for Pacman. Now, he has to push 3 boxes to 3 different buttons. Each box and button are numbered, and Pacman has to push the box to the button with the same number (Figure 1b).

(i) [2 pts] What is the size of the minimal state space? Express your answer using the symbols M and N.

$(MN)^4$ The mininmal state space corresponds to the position of the Pacman and the position of the 3 boxes. Since each of them can be in MN positions the size of the state space is $(MN)^4$.

(ii) [2 pts] Which of the following heuristics are admissible?

- ☐ d_{Man}(Pacman, button 1) + d_{Man}(Pacman, button 2) + d_{Man}(Pacman, button 3) - 3
- ■ d_{Man}(box 1, button 1) + d_{Man}(box 2, button 2) + d_{Man}(box 3, button 3)
- ☐ d_{Man}(box 1, box 2) + d_{Man}(box 1, box 3)
- ■ min(d_{Man}(box 1, button 1), d_{Man}(box 2, button 2), d_{Man}(box 3, button 3))
- ☐ None of the above

The first one is not admissible since when, for instance, two of the boxes are placed and you are about to place the last one, the cost in that state is 1. However, the distance between Pacman and the buttons can be any number. The third one is not admissible neither because, for instance, the goal state does not have the value of 0.

(c) In the third maze, the 3 boxes can go to any of the 3 buttons (Figure 1c).

(i) [2 pts] What is the size of the minimal state space? Express your answer using the symbols M and N.

$MN\binom{MN}{3}$ The mininmal state space corresponds to the position of the Pacman and the position of the 3 boxes. Each of them can be in MN positions, and we do not care about knowing which box is where. Then size of the state space is $MN\frac{(MN)^3}{3!}$.

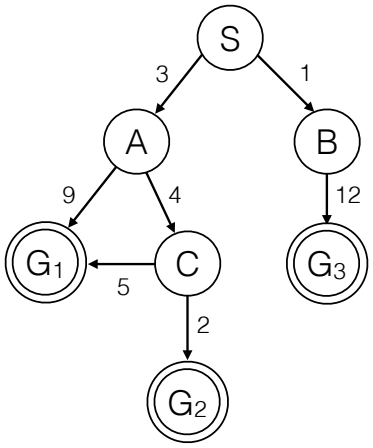
(ii) [2 pts] Which of the following heuristics are consistent?

- ☐ $\max_{ij} d_{\text{Man}}$(box i, button j)
- ■ $\min_{ij} d_{\text{Man}}$(box i, button j)
- ☐ $\max_{j} d_{\text{Man}}$(Pacman, button j)
- ■ $\min_{i} d_{\text{Man}}$(Pacman, box i) - 1
- ☐ None of the above

The first one is not consistent because is clearly not admissible. In a goal state the heuristic is not 0 but the maximum distance between any box and button. The third one is not consistent neither because is also not admissible; for instance, the heuristic in the goal state does not have a cost of 0.

SID: _____

Q3. [12 pts] Search Algorithms



	A	B	C	S
H-1	0	0	0	0
H-2	6	7	1	7
H-3	7	7	1	7
H-4	4	7	1	7

(a) Consider the search graph and heuristics shown above. Select **all** of the goals that **could** be returned by each of the search algorithms below. For this question, if there is a tie on the fringe, assume the tie is broken **randomly**.

(i) [1 pt] DFS — G_1 ● G_2 ● G_3 ●

(ii) [1 pt] BFS — G_1 ● G_2 ○ G_3 ●

(iii) [1 pt] UCS — G_1 ○ G_2 ● G_3 ○

(iv) [1 pt] Greedy with H-1 — G_1 ● G_2 ● G_3 ●

(v) [1 pt] Greedy with H-2 — G_1 ● G_2 ○ G_3 ○

(vi) [1 pt] Greedy with H-3 — G_1 ● G_2 ○ G_3 ●

(vii) [1 pt] A* with H-2 — G_1 ○ G_2 ● G_3 ○

(viii) [1 pt] A* with H-3 — G_1 ○ G_2 ● G_3 ○

(b) For each heuristic, indicate whether it is consistent, admissible, or neither (select more than one option if appropriate):

(i) [1 pt] H-1 — Consistent ● Admissible ● Neither ○

(ii) [1 pt] H-2 — Consistent ○ Admissible ● Neither ○

(iii) [1 pt] H-3 — Consistent ○ Admissible ○ Neither ●

(iv) [1 pt] H-4 — Consistent ● Admissible ● Neither ○

Q4. [13 pts] CSPs: Potluck Pandemonium

The potluck is coming up and the staff haven't figured out what to bring yet! They've pooled their resources and determined that they can bring some subset of the following items.

1. Pho
2. Apricots
3. Frozen Yogurt
4. Fried Rice
5. Apple Pie
6. Animal Crackers

There are five people on the course staff: Taylor, Jonathan, Faraz, Brian, and Alvin. Each of them will only bring one item to the potluck.

1. If (F)araz brings the same item as someone else, it cannot be (B)rian.
2. (A)lvin has pho-phobia so he won't bring Pho, but he'll be okay if someone else brings it.
3. (B)rian is no longer allowed near a stove, so he can only bring items 2, 3, or 6.
4. (F)araz literally can't even; he won't bring items 2, 4, or 6.
5. (J)onathan was busy, so he didn't see the last third of the list. Therefore, he will only bring item 1, 2, 3, or 4.
6. (T)aylor will only bring an item that is before an item that (J)onathan brings.
7. (T)aylor is allergic to animal crackers, so he won't bring item 6. (If someone else brings it, he'll just stay away from that table.)
8. (F)araz and (J)onathan will only bring items that have the same first letter (e.g. Frozen Yogurt and Fried Rice).
9. (B)rian will only bring an item that is after an item that (A)lvin brings on the list.
10. (J)onathan and (T)aylor want to be unique; they won't bring the same item as anyone else.

This page is repeated as the second-to-last page of this midterm for you to rip out and use for reference as you work through the problem.

SID: _____

(a) [1 pt] Which of the listed constraints are unary constraints?

i ○ ii ● iii ● iv ● v ●

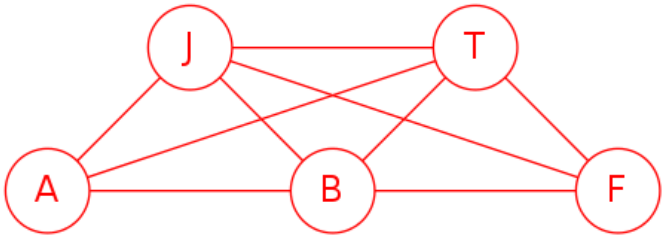
vi ○ vii ● viii ○ ix ○ x ○

(b) [2 pts] Rewrite implicit constraint viii. as an explicit constraint.

(F, J) ∈ { (3, 4), (4, 3),
(2, 5), (5, 2),
(2, 6), (6, 2),
(5, 6), (6, 5),
(1, 1), (2, 2), (3, 3),
(4, 4), (5, 5), (6, 6)
}

(c) [1 pt] How many edges are there in the constraint graph for this CSP?

There are 9 edges in this constraint graph.



(d) [2 pts] The table below shows the variable domains after all unary constraints have been enforced.

A		2	3	4	5	6
B		2	3			6
F	1		3		5	
J	1	2	3	4		
T	1	2	3	4	5	

Following the Minimum Remaining Values heuristic, which variable should we assign first? Break all ties alphabetically.

A ○ B ● F ○ J ○ T ○

(e) To decouple this from the previous question, assume that we choose to assign (F)araz first. In this question, we will choose which value to assign to using the Least Constraining Value method.

To determine the number of remaining values, enforce arc consistency to prune the domains. Then, count the total number of possible assignments (**not** the total number of remaining values). It may help you to enforce arc consistency twice, once before assigning values to (F)araz, and then again after assigning a value.

The domains after enforcing unary constraints are reproduced in each subquestion. The grids are provided as scratch space and **will not** be graded. Only numbers written in the blanks will be graded. The second grid is provided as a back-up in case you mess up on the first one. More grids are also provided on the second-to-last page of the exam.

(i) [2 pts] Assigning F = 1 results in 0 possible assignments.

A		2	3	4	5	6
B		2	3			6
F	1		3		5	
J	1	2	3	4		
T	1	2	3	4	5	

A		2	3	4	5	6
B		2	3			6
F	1		3		5	
J	1	2	3	4		
T	1	2	3	4	5	

Assigning F = 1 leaves no possible values in J's domain (due to constraint viii).

(ii) [2 pts] Assigning F = 3 results in 5 possible assignments.

A		2	3	4	5	6
B		2	3			6
F	1		3		5	
J	1	2	3	4		
T	1	2	3	4	5	

A		2	3	4	5	6
B		2	3			6
F	1		3		5	
J	1	2	3	4		
T	1	2	3	4	5	

Assigning F = 3 leaves J's domain as $\{4\}$. Enforcing arc consistency gives A = $\{2, 3, 5\}$, B = $\{6\}$, and T = $\{1, 2\}$. Therefore, the 5 possible assignments are (A, B, F, J, T) = (2, 6, 3, 4, 1), (3, 6, 3, 4, 1), (5, 6, 3, 4, 1), (3, 6, 3, 4, 2), (5, 6, 3, 4, 2).

(iii) [2 pts] Assigning F = 5 results in 3 possible assignments.

A		2	3	4	5	6
B		2	3			6
F	1		3		5	
J	1	2	3	4		
T	1	2	3	4	5	

A		2	3	4	5	6
B		2	3			6
F	1		3		5	
J	1	2	3	4		
T	1	2	3	4	5	

Assigning F = 5 leaves J's domain as $\{2\}$. Enforcing arc consistency gives A = $\{3, 4, 5\}$, B = $\{6\}$, and T = $\{1\}$. Therefore, the 3 possible assignments are (A, B, F, J, T) = (3, 6, 5, 2, 1), (4, 6, 5, 2, 1), (5, 6, 5, 2, 1).

(iv) [1 pt] Using the LCV method, which value should we assign to F? If there is a tie, choose the lower number. (e.g. If both 1 and 2 have the same value, then fill 1.)

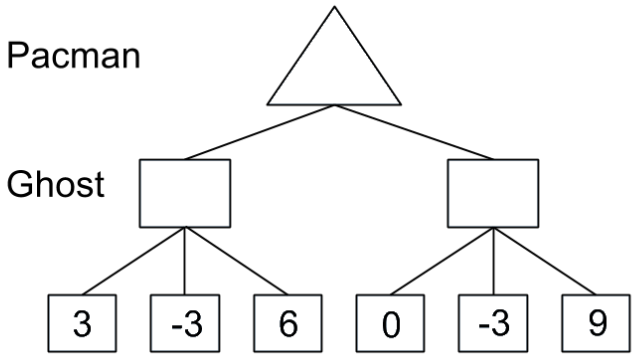
1 ○ 2 ○ 3 ● 4 ○ 5 ○ 6 ○

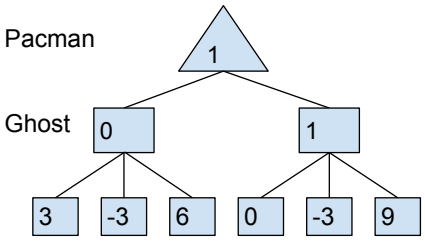
SID: _____

Q5. [8 pts] Variants of Trees

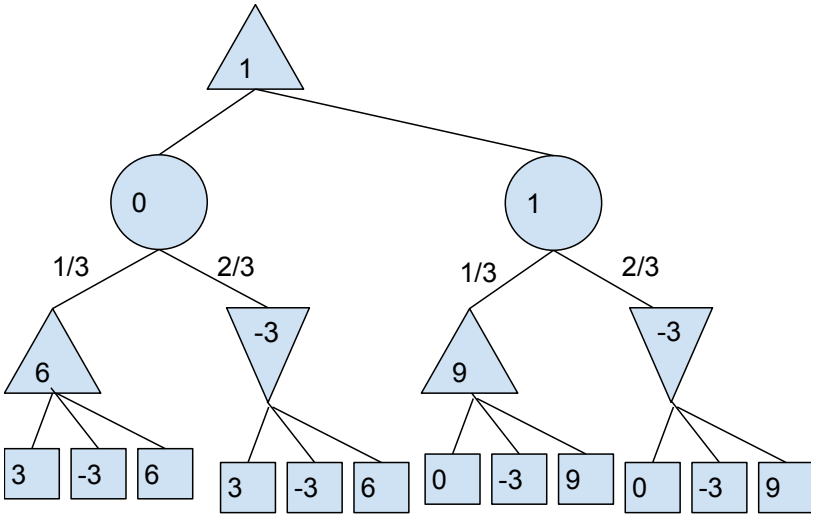
(a) Pacman is going to play against a careless ghost, which makes a move that is optimal for Pacman $\frac{1}{3}$ of the time, and makes a move that that minimizes Pacman's utility the other $\frac{2}{3}$ of the time.

(i) [2 pts] Fill in the correct utility values in the game tree below where Pacman is the maximizer:

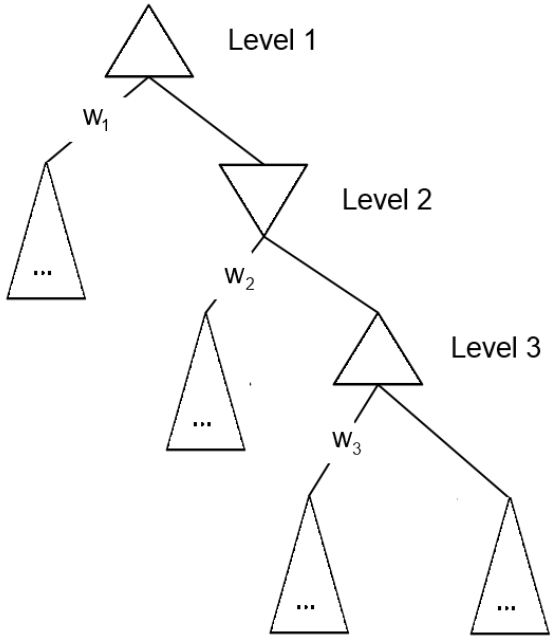




(ii) [2 pts] Draw a complete game tree for the game above that contains only max nodes, min nodes, and chance nodes.



(b) Consider a modification of alpha-beta pruning where, rather than keeping track of a single value for α and β, you instead keep a list containing the best value, w_i, for the minimizer/maximizer (depending on the level) at each level up to and including the current level. Assume that the root node is always a max node. For example, consider the following game tree in which the first 3 levels are shown. When the considering the right child of the node at level 3, you have access to w_1, w_2, and w_3.



(i) [1 pt] Under this new scenario, what is the pruning condition for a max node at the n^{th} level of the tree (in terms of v and $w_1 ... w_n$)? $v > min_i(w_i)$, where i is even;

(ii) [1 pt] What is the pruning condition for a min node at the n^{th} level of the tree?
$v < max_i(w_i)$, where i is odd;

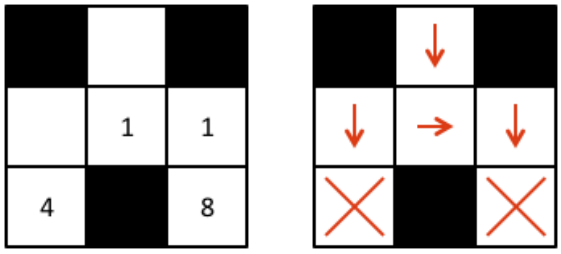
(iii) [2 pts] What is the relationship between α, β and the list of $w_1 ... w_n$ at a max node at the n^{th} level of the tree?

- ◯ $\sum_i w_i = \alpha + \beta$
- ◯ $\max_i w_i = \alpha, \quad \min_i w_i = \beta$
- ◯ $\min_i w_i = \alpha, \quad \max_i w_i = \beta$
- ◯ $w_n = \alpha, \quad w_{n-1} = \beta$
- ◯ $w_{n-1} = \alpha, \quad w_n = \beta$
- ◯ None of the above. The relationship is $\beta = min(w_2, w_4, w_6 ...), \quad \alpha = max(w_1, w_3, w_5 ...)$

Q6. [21 pts] Reward Shaping

Consider the following Gridworld-like environment. The robot can move deterministically Up, Down, Right, or Left, or at any time it can exit to a terminal state (where it remains). The reward for any non-exit action is always 0. If the robot is on a square with a number written on it, it receives a reward of that size **on Exiting**. If the robot exits from any square without a number written on it, it receives a reward of 0 (and still exits to a terminal state). Note that when it is on any of the squares (including numbered squares), it can either move Up, Down, Right, Left or Exit. However, it only receives a non-zero reward when it Exits on a numbered square. **The robot is not required to exit on a numbered square; it can also move off of that square. However, if it does not exit, it does not get the reward.**

(a) [3 pts] Draw an arrow in **each** square (including numbered squares) in the following board on the right to indicate the optimal policy PacBot will calculate with the discount factor $\gamma = 0.5$ in the board on the left. (For example, if PacBot would move Down from the square in the middle on the left board, draw a down arrow in that square on the right board.) If PacBot's policy would be to exit from a particular square, draw an X instead of an arrow in that square.



From the middle-right square, we discount once before getting to the 8, making the value of moving that way 4, higher than anything else reachable. Similarly, from the middle square, the value of moving right is 2 (as opposed to 1 for exiting or 1 for moving left), and the value of moving down from the top state is 1. From middle-left, the value of moving down is 2 as opposed to 1 for moving right. From bottom-left, the value for exiting is 4 as opposed to 0.5 for moving up.

The key thing to notice for this problem is that the value of a policy that moves toward an exit decays by a factor of 1/2 for every step. With this in mind, you can compare actions from each state by considering how far you are from each goal. Alternatively, any algorithm we've seen for exactly solving MDPs (value iteration, policy iteration, Q-value iteration) would have worked.

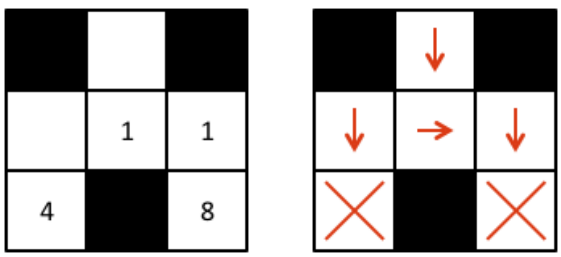
The agent now operates in a new environment with an additional reward function $F(s, a, s')$, which is added to the original reward function $R(s, a, s')$ for every (s, a, s') triplet, so that the new reward function $R'(s, a, s') = R(s, a, s') + F(s, a, s')$.

(b) [3 pts] Consider an additional reward F_1 that favors moving toward numbered squares. Let $d(s)$ be defined as the Manhattan distance from s to the nearest numbered square. If s is numbered, $d(s) = 0$.

$$F_1(s, a, s') = 6\left(d(s) - \frac{1}{2}d(s')\right).$$

F_1 is always 0 when s' is a terminal state (equivalently, when a is the Exit action).

Fill in the diagram as in (a) in the following board to indicate the optimal policy PacBot will calculate with the discount factor $\gamma = 0.5$ and the modified reward function $R'_1(s, a, s') = R(s, a, s') + F_1(s, a, s')$.



This added reward makes no difference to the optimal policy. For any state s in this MDP, $d(s)$ can only be 0 (on a numbered square) or 1 (on a blank square). Any sequence of steps which takes us from a numbered square to another numbered square either stays on numbered squares the whole time, in which case $d(s) = 0$ everywhere and thus $F_1 = 0$ on every transition, or we take a step from a numbered to a non-numbered square and back, in which case we accrue F_1 rewards as follows: if our sequence of states and actions is s_1, a_1, s_2, a_2, s_3 with $d(s_1) = d(s_3) = 0$ and $d(s_2) = 1$, then we get

$$\begin{aligned} F(s_1, a_2, s_2) + \gamma F(s_2, a_2, s_3) &= 6\left(d(s_1) - \frac{1}{2}d(s_2)\right) + \gamma 6\left(d(s_2) - \frac{1}{2}d(s_3)\right) \\ &= 6\left(0 - \frac{1}{2}\cdot 1\right) + \frac{1}{2}\cdot 6\left(1 - \frac{1}{2}\cdot 0\right) \\ &= -3 + 3 \qquad\qquad = 0. \end{aligned}$$

What this means is that total effect of switching which square we exit from by F_1 is 0, so F_1 actually makes no difference. Intuitively, because of our discount factor $\gamma = \frac{1}{2}$, every negative reward will cancel out with the positive rewards, and all but the first and last terms in all the F_1 rewards added together will cancel out. When we are determining the optimal policy, we must compare the actions we can take at state s. $d(s)$ will be the first term in the sum of F_1 rewards, and the last term will always be 0 since we exit from a numbered square, so F_1 makes no difference as to the optimal action.

There is a more rigorous proof that F_1 will not change optimal policies that is beyond the scope of the course. You can also confirm that this policy is optimal in this particular maze by evaluating this policy in the modified environment and confirming that those values satisfy the Bellman equations.

(c) [1 pt] If the robot now executes this policy π in the **original** environment, without the extra added rewards F, what is $V^\pi(s)$ where s is the top-most state?

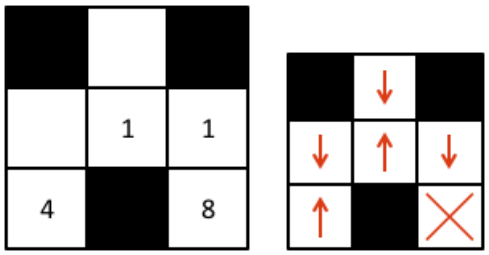
The robot takes three steps before it can exit and gather the reward of 8, so the value of the top-most state is $\gamma^3 \cdot 8 = \boxed{1}$.

(d) [3 pts] Consider a different artificial reward that also favors moving toward numbered squares in a slightly different way:

$$F_2(s, a, s') = \begin{cases} 6 & d(s') < d(s) \text{ i.e. } s' \text{ is closer to a numbered square than } s \text{ is,} \\ 0 & d(s') \geq d(s). \end{cases}$$

F_2 is always 0 when s' is a terminal state (equivalently, when a is the Exit action).

Fill in the diagram on the right as in (a) to indicate the optimal policy PacBot will calculate with the discount factor $\gamma = 0.5$ and the modified reward function $R_2'(s, a, s') = R(s, a, s') + F_2(s, a, s')$ in the board on the left. Break ties in the following order: Up, Down, Right, Left, Exit.



Unlike F_1, F_2 actually encourages cycles. Since there are no penalties for moving away from numbered squares here as there were previously, the agent can accrue large positive rewards by stepping on and off a numbered square repeatedly. Specifically, for stepping on a numbered square, it gets reward 6, then if it steps off and on again, it gets reward $\gamma^2 \cdot 6 = 3/2$. This cycle gives a geometric sum worth $\frac{6}{1-\gamma^2} = \frac{6}{3/4} = 8$. We consider then the value of being on a non-numbered square to be 8, and we can apply the same reasoning as in (a) to determine the optimal policy. If the agent is closer to where it can exit for 8 points, it moves toward that. If it is closer to the cycle where it can accrue F_2 rewards worth 8, it moves toward that cycle. (Note that $F_2 = 0$ when moving between two different numbered squares.)

(e) [1 pt] If the robot now executes this policy π in the **original** environment, without the extra added rewards F, what is $V^{\pi}(s)$ where s is the top-most state?

In any policy that is optimal in the altered environment, the agent will get stuck in a cycle if it starts from the top state, meaning it will never exit. In the original environment, exiting is the only way to get any non-zero reward, so it will get no reward and the value is $\boxed{0}$.

(f) [4 pts] For each of the following conditions on $F(s, a, s')$, state whether the condition is necessary and/or sufficient for the set of optimal policies to be unchanged in a general Gridworld-like MDP (i.e. an MDP following the rules laid out at the beginning of this question, but with any arbitrary board configuration) by adding F to the reward function. Assume $\gamma = 1$, all states are reachable from all non-terminal states, and there is at least one positive number on the board. Note that the set of optimal policies is unchanged between a pair of MDPs when a policy is optimal in one MDP if and only if it is also optimal in the other.

(i) [1 pt] Condition 1: If M is the maximum number on the board, then in the modified MDP, the set of all optimal policies is all policies such that no matter where you start, you will exit from a square showing M.

■ necessary ■ sufficient ☐ neither

This is equivalent to optimal policies being unchanged because this is a full description of optimal policies in the *unmodified* MDP. Any square is reachable from any other, and since we are dealing with the undiscounted case, if we are at a square showing less than M, it is always better to move to a square showing M and exit than to exit from the current square.

(ii) [1 pt] Condition 2: If M is the maximum number on the board, $|F(s, a, s')| \leq M$ for all s, a, s' with s' not a terminal state.

☐ necessary ☐ sufficient ■ neither

For a counterexample for necessity, consider $F(s, a, s') = k(d(s) - d(s'))$, a function similar to F_1. For any k, the arguments in the solution to (b) still hold, and so for arbitrarily large additive rewards, we can maintain optimal policies. For a counterexample for sufficiency, use the board from part (a), use $\gamma = 1$, and let $F(s, a, s')$ be 1 everywhere. Condition (iv) is fulfilled, since $|1| < 8$, but the policy in the modified environment will never exit since it can move back and forth between squares to get infinite reward.

(iii) [1 pt] Condition 3: The value function is unchanged; that is, $V'(s) = V(s)$ for all s, where V' is the value function in the modified environment.

☐ necessary ☐ sufficient ■ neither

This is not necessary, because we can design some additive reward that increases all of the values by 1 (for example, by making $F = 1$ for exiting from the highest numbered square), and we will still extract the same policy. It is harder to see that it is not sufficient. Suppose this condition is fulfilled and all the values are unchanged. To extract a policy for V values, we still have to do a one-step lookahead, which allows the additive rewards to add some influence. Suppose for a concrete counterexample that we use the board from (a) and have a reward of 1 added for exiting from the top-most state. The value of that state is still 1 as it was in the unmodified environment (see part (c)), but we have added a new optimal policy: exiting from that square now also achieves optimal value.

(iv) [1 pt] Condition 4: The Q-value function is unchanged; that is, $Q'(s, a) = Q(s, a)$ for all s and a, where Q' is the Q-value function in the modified environment.

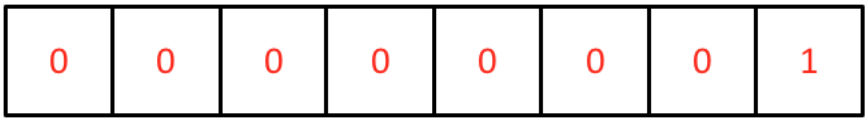
☐ necessary ■ sufficient ☐ neither

This is not necessary for the same reason that the previous condition is not necessary. However, it is sufficient, because extracting a policy from Q-values only requires taking an argmax, rather than doing the one-step lookahead. The Q-values explicitly designate a policy, making this a stronger condition than unchanged V-values that is actually sufficient.

Consider the following new Gridworld-like environment consisting of 8 states all in a row with all squares blank except the rightmost, which shows a 1. We restrict actions to Right, Left, and Exit. (The rules are exactly the same as in (a), except that the Up and Down actions are no longer available).

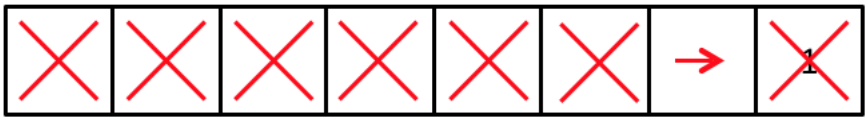
(g) In this environment, initialize a policy π_0 in which we exit at every state.

(i) [1 pt] Fill in the following diagram with values for V^{π_0}, the result of running policy evaluation with $\gamma = 1$. (For example, if the leftmost square has value 4 when following this policy, write a 4 in that square on the diagram. Note that this is the same MDP as above, but the 1 has been removed for your convenience. The robot still receives a reward of 1 for exiting from the rightmost square.)



The value of exiting from each state is 0 for a non-numbered square and 1 on the square showing a 1.

(ii) [1 pt] Let π_1 be the new policy after one step of policy improvement. Break ties in the following order: Exit, Left, Right. As in (a), draw an arrow or an X in each square to represent this policy.



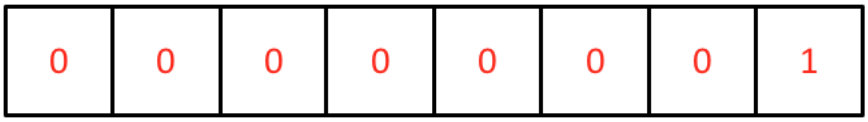
For the rightmost square, the optimal action is to exit, giving reward 1. From the square immediately to the left of that, the optimal action is to move to the right, since the rightmost square has value 1. As seen in the previous answer, with the current values, from all other squares, all actions give reward zero and sum of future rewards equal to 0, so all actions are equal and we go with the action that takes precedence in the tiebreaking order: Exit.

(iii) [1 pt] How many iterations of policy iteration are necessary to converge on the optimal policy? 7 iterations; after each one, one more state will switch to moving right instead of exiting. Some students considered the iteration in which we know we've converged, after we've seen the same policy twice, so credit was also awarded for the answer 8.

(h) We now reintroduce the extra reward $F_1(s, a, s') = 6\left(d(s) - \frac{1}{2}d(s')\right)$ from part (b). For the rest of this question, we are working in a modified version of the long environment above where the reward function is $R'(s, a, s') = R(s, a, s') + F_1(s, a, s')$.

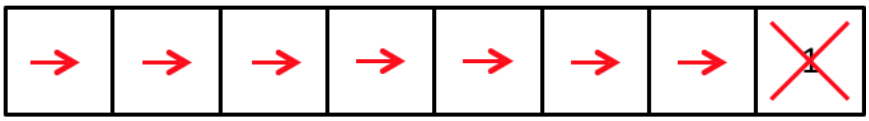
Once again, initialize a policy π_0 in which we exit at every state.

(i) [1 pt] As in (g), fill in the following diagram with values for V^{π_0}, the result of running policy evaluation with $\gamma = \frac{1}{2}$ in the **modified** environment.



Since the added rewards are zero for all exit transitions, for π_0, the added rewards make no difference whatsoever.

(ii) [1 pt] Let π_1 be the new policy after one step of policy improvement in the **modified** environment. Break ties in the following order: Stop, Left, Right. As in (a), draw an arrow or an X in each square to represent this policy.



Because policy improvement takes the action that maximizes $R'(s, a, s') + \gamma V(s') = R(s, a, s') + F_1(s, a, s') + \gamma V(s')$ (since transitions are deterministic) and moving right from all squares but the rightmost gives positive F_1 value, the action chosen at all of those states is to move right.

(iii) [1 pt] How many iterations of policy iteration are necessary to converge on the optimal policy? Only one iteration is necessary. The policy described in part (ii) is optimal. Some students considered the iteration in which we know we've converged, after we've seen the same policy twice, so credit was also awarded for the answer 2.

There was a typo in the original version of the exam in which $\gamma = 1$ for part (h). Most students gave the intended answer, but credit was also awarded to students who answered the question correctly with $\gamma = 1$.

SID: _____

Q7. [13 pts] Q-uagmire

Consider an unknown MDP with three states (A, B and C) and two actions ($\leftarrow$ and $\rightarrow$). Suppose the agent chooses actions according to some policy π in the unknown MDP, collecting a dataset consisting of samples (s, a, s', r) representing taking action a in state s resulting in a transition to state s' and a reward of r.

s	a	s'	r
A	$\rightarrow$	B	2
C	$\leftarrow$	B	2
B	$\rightarrow$	C	-2
A	$\rightarrow$	B	4

You may assume a discount factor of $\gamma = 1$.

(a) Recall the update function of Q-learning is:

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

Assume that all Q-values are initialized to 0, and use a learning rate of $\alpha = \frac{1}{2}$.

(i) [4 pts] Run Q-learning on the above experience table and fill in the following Q-values:

$Q(A, \rightarrow) =$ 5/2 $\quad$ $Q(B, \rightarrow) =$ $-1/2$

$$Q_1(A, \rightarrow) = \frac{1}{2} \cdot Q_0(A, \rightarrow) + \frac{1}{2}\left(2 + \gamma \max_{a'} Q(B, a')\right) = 1$$
$$Q_1(C, \leftarrow) = 1$$
$$Q_1(B, \rightarrow) = \frac{1}{2}(-2 + 1) = -\frac{1}{2}$$
$$Q_2(A, \rightarrow) = \frac{1}{2} \cdot 1 + \frac{1}{2}\left(4 + \max_{a'} Q_1(B, a')\right)$$
$$= \frac{1}{2} + \frac{1}{2}(4 + 0) = \frac{5}{2}.$$

(ii) [2 pts] After running Q-learning and producing the above Q-values, you construct a policy π_Q that maximizes the Q-value in a given state:

$$\pi_Q(s) = \arg\max_a Q(s, a).$$

What are the actions chosen by the policy in states A and B?

$\pi_Q(A)$ is equal to:

- ○ $\pi_Q(A) = \leftarrow$.
- ● $\pi_Q(A) = \rightarrow$.
- ○ $\pi_Q(A) =$ Undefined.

$\pi_Q(B)$ is equal to:

- ● $\pi_Q(B) = \leftarrow$.
- ○ $\pi_Q(B) = \rightarrow$.
- ○ $\pi_Q(B) =$ Undefined.

Note that $Q(B, \leftarrow) = 0 > -\frac{1}{2} = Q(B, \rightarrow)$.

(b) [3 pts] Use the empirical frequency count model-based reinforcement learning method described in lectures to estimate the transition function $\hat{T}(s, a, s')$ and reward function $\hat{R}(s, a, s')$. (Do not use pseudocounts; if a transition is not observed, it has a count of 0.)

Write down the following quantities. You may write N/A for undefined quantities.

$\hat{T}(A, \rightarrow, B) =$ 1 $\quad \hat{R}(A, \rightarrow, B) =$ 3

$\hat{T}(B, \rightarrow, A) =$ 0 $\quad \hat{R}(B, \rightarrow, A) =$ N/A

$\hat{T}(B, \leftarrow, A) =$ N/A $\quad \hat{R}(B, \leftarrow, A) =$ N/A

(c) This question considers properties of reinforcement learning algorithms for *arbitrary* discrete MDPs; you do not need to refer to the MDP considered in the previous parts.

(i) [2 pts] Which of the following methods, at convergence, provide enough information to obtain an optimal policy? (Assume adequate exploration.)

- [x] Model-based learning of $T(s, a, s')$ and $R(s, a, s')$.
- [ ] Direct Evaluation to estimate $V(s)$.
- [ ] Temporal Difference learning to estimate $V(s)$.
- [x] Q-Learning to estimate $Q(s, a)$.

Given enough data, model-based learning will get arbitrarily close to the true model of the environment, at which point planning (e.g. value iteration) can be used to find an optimal policy. Q-learning is similarly guaranteed to converge to the optimal Q-values of the optimal policy, at which point the optimal policy can be recovered by $\pi^*(s) = \arg\max_a Q(s, a)$. Direct evaluation and temporal difference learning both only recover a value function $V(s)$, which is insufficient to choose between actions without knowledge of the transition probabilities.

(ii) [2 pts] In the limit of infinite timesteps, under which of the following exploration policies is Q-learning guaranteed to converge to the optimal Q-values for all state? (You may assume the learning rate α is chosen appropriately, and that the MDP is ergodic: i.e., every state is reachable from every other state with non-zero probability.)

- [x] A fixed policy taking actions uniformly at random.
- [ ] A greedy policy.
- [x] An ϵ-greedy policy
- [ ] A fixed optimal policy.

For Q-learning to converge, every state-action pair (s, a) must occur infinitely often. A uniform random policy will achieve this in an ergodic MDP. A fixed optimal policy will not take any suboptimal actions and so will not explore enough. Similarly a greedy policy will stop taking actions the current Q-values suggest are suboptimal, and so will never update the Q-values for supposedly suboptimal actions. (This is problematic if, for example, an action most of the time yields no reward but occasionally yields very high reward. After observing no reward a few times, Q-learning with a greedy policy would stop taking that action, never obtaining the high reward needed to update it to its true value.)