CS 188 Spring 2016 Introduction to Artificial Intelligence Midterm V2

- 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	
Name of Person to Left	
Name of Person to Right	

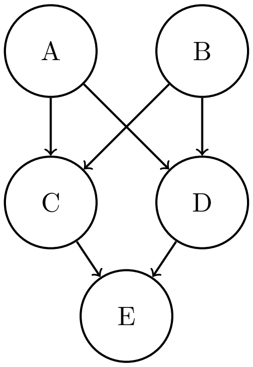
For staff use only:

Total	/??

THIS PAGE IS INTENTIONALLY LEFT BLANK

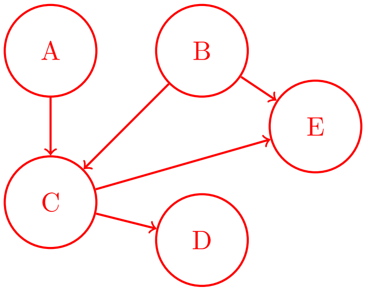
Q1. [14 pts] Bayes Nets and Joint Distributions

(a) [2 pts] Write down the joint probability distribution associated with the following Bayes Net. Express the answer as a product of terms representing individual conditional probabilities tables associated with this Bayes Net:



$P(A)P(B)P(C|A,B)P(D|A,B)P(E|C,D)$

(b) [2 pts] Draw the Bayes net associated with the following joint distribution:
$P(A) \cdot P(B) \cdot P(C|A,B) \cdot P(D|C) \cdot P(E|B,C)$



(c) [3 pts] Do the following products of factors correspond to a valid joint distribution over the variables A, B, C, D? (Circle TRUE or FALSE.)

(i) TRUE [✓ FALSE] $P(A) \cdot P(B) \cdot P(C|A) \cdot P(C|B) \cdot P(D|C)$

(ii) [✓ TRUE] FALSE $P(A) \cdot P(B|A) \cdot P(C) \cdot P(D|B,C)$

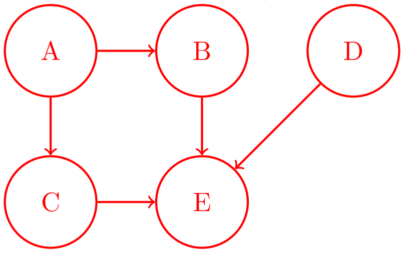
(iii) TRUE [✓ FALSE] $P(A) \cdot P(B|A) \cdot P(C) \cdot P(C|A) \cdot P(D)$

(iv) TRUE [✓ FALSE] $P(A|B) \cdot P(B|C) \cdot P(C|D) \cdot P(D|A)$

(d) What factor can be multiplied with the following factors to form a valid joint distribution? (Write “none” if the given set of factors can’t be turned into a joint by the inclusion of exactly one more factor.)

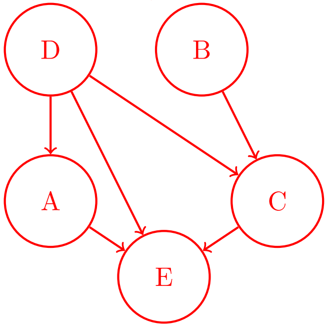
(i) [2 pts] $P(A) \cdot P(B|A) \cdot P(C|A) \cdot P(E|B,C,D)$

P(D) is missing. D could also be conditioned on A,B, and/or C without creating a cycle (e.g. $P(D|A,B,C)$). Here is an example bayes net that would represent the distribution after adding in $P(D)$:

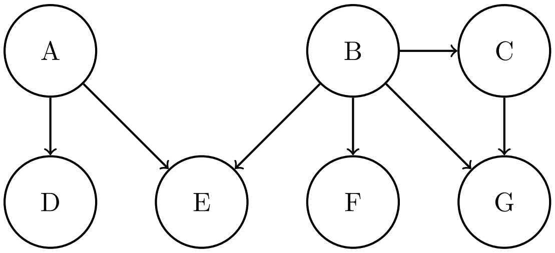


(ii) [2 pts] $P(D) \cdot P(B) \cdot P(C|D,B) \cdot P(E|C,D,A)$

P(A) is missing to form a valid joint distributions. A could also be conditioned on B, C, and/or D (e.g. $P(A|B,C,D)$. Here is a bayes net that would represent the distribution is $P(A|D)$ was added in.



(e) Answer the next questions based off of the Bayes Net below:
All variables have domains of {-1, 0, 1}



(i) [1 pt] Before eliminating any variables or including any evidence, how many entries does the factor at G have?

The factor is $P(G|B,C)$, so that gives $3^3 = 27$ entries.

(ii) [2 pts] Now we observe $e = 1$ and want to query $P(D|e = 1)$, and you get to pick the first variable to be eliminated.

- Which choice would create the **largest** factor f_1?
  Eliminating B first would give the largest f_1:, $f_1(A,F,G,C,e) = \sum_{B=b} P(b)P(e|A,b)P(F|b)P(G|b,C)P(C|b)$. This factor has 3^4 entries.
- Which choice would create the **smallest** factor f_1?
  Eliminating A or eliminating F first would give smallest factors of 3 entries: either $f1(D,e) = \sum_a P(D|a)P(e|a)P(a)$ of $f1(B) = \sum_f P(f|B)$. Eliminating D is not correct because D is the query variable.

Q2. [8 pts] Pacman's Life

Suppose a maze has height M and width N and there are F food pellets at the beginning. Pacman can move North, South, East or West in the maze.

(a) [4 pts] In this subquestion, the position of Pacman is known, and he wants to pick up all F food pellets in the maze. However, Pacman can move North at most two times overall.

What is the size of a minimal state space for this problem? Give your answer as a product of terms that reference problem quantities such as (but not limited to) M, N, F, etc. Below each term, state the information it encodes. For example, you might write $4 \times MN$ and write number of directions underneath the first term and Pacman's position under the second.
$MN \times 2^F \times 3$. Pacman's position, a boolean vector representing whether a certain food pellet has been eaten, and the number of times Pacman has moved North (which could be 0, 1 or 2).

(b) [4 pts] In this subquestion, Pacman is lost in the maze, and does not know his location. However, Pacman still wants to visit every single square (he does not care about collecting the food pellets any more). Pacman's task is to find a sequence of actions which guarantees that he will visit every single square.

What is the size of a minimal state space for this problem? As in part(a), give your answer as a product of terms along with the information encoded by each term. You will receive partial credit for a complete but non-minimal state space.
$2^{((MN)^2)}$. For every starting location, we need a boolean for every position (MN) to keep track of all the visited locations. In other words, we need MN sets of MN booleans for a total of $(MN)^2$ booleans. Hence, the state space is $2^{((MN)^2)}$.

Q3. [13 pts] MDPs: Dice Bonanza

A casino is considering adding a new game to their collection, but need to analyze it before releasing it on their floor. They have hired you to execute the analysis. On each round of the game, the player has the option of rolling a fair 6-sided die. That is, the die lands on values 1 through 6 with equal probability. Each roll costs 1 dollar, and the player **must** roll the very first round. Each time the player rolls the die, the player has two possible actions:

1. *Stop*: Stop playing by collecting the dollar value that the die lands on, or
2. *Roll*: Roll again, paying another 1 dollar.

Having taken CS 188, you decide to model this problem using an infinite horizon Markov Decision Process (MDP). The player initially starts in state *Start*, where the player only has one possible action: *Roll*. State s_i denotes the state where the die lands on i. Once a player decides to *Stop*, the game is over, transitioning the player to the *End* state.

(a) [4 pts] In solving this problem, you consider using policy iteration. Your initial policy π is in the table below. Evaluate the policy at each state, with $\gamma = 1$.

State	s_1	s_2	s_3	s_4	s_5	s_6
$\pi(s)$	*Roll*	*Roll*	*Stop*	*Stop*	*Stop*	*Stop*
$V^\pi(s)$	3	3	3	4	5	6

We have that $s_i = i$ for $i \in \{3, 4, 5, 6\}$, since the player will be awarded no further rewards according to the policy. From the Bellman equations, we have that $V(s_1) = -1 + \frac{1}{6}(V(s_1) + V(s_2) + 3 + 4 + 5 + 6)$ and that $V(s_2) = -1 + \frac{1}{6}(V(s_1) + V(s_2) + 3 + 4 + 5 + 6)$. Solving this linear system yields $V(s_1) = V(s_2) = 3$.

(b) [4 pts] Having determined the values, perform a policy update to find the new policy π'. The table below shows the old policy π and has filled in parts of the updated policy π' for you. If both *Roll* and *Stop* are viable new actions for a state, write down both *Roll/Stop*. In this part as well, we have $\gamma = 1$.

State	s_1	s_2	s_3	s_4	s_5	s_6
$\pi(s)$	*Roll*	*Roll*	*Stop*	*Stop*	*Stop*	*Stop*
$\pi'(s)$	*Roll*	*Roll*	*Roll/Stop*	*Stop*	*Stop*	*Stop*

For each s_i in part (a), we compare the values obtained via Rolling and Stopping. The value of Rolling for each state s_i is $-1 + \frac{1}{6}(3 + 3 + 3 + 4 + 5 + 6) = 3$. The value of Stopping for each state s_i is i. At each state s_i, we take the action that yields the largest value; so, for s_1 and s_2, we Roll, and for s_4 and s_5, we stop. For s_3, we Roll/Stop, since the values from Rolling and Stopping are equal.

(c) [2 pts] Is $\pi(s)$ from part (a) optimal? Explain why or why not.

Yes, the old policy is optimal. Looking at part (b), there is a tie between 2 equally good policies that policy iteration considers employing. One of these policies is the same as the old policy. This means that both new policies are as equally good as the old policy, and policy iteration has converged. Since policy iteration converges to the optimal policy, we can be sure that $\pi(s)$ from part (a) is optimal.

(d) [3 pts] Suppose that we were now working with some $\gamma \in [0, 1)$ and wanted to run **value iteration**. Select the **one** statement that would hold true at convergence, or write the correct answer next to Other if none of the options are correct.

○ $V^*(s_i) = \max\left\{-1+\frac{i}{6}\ ,\ \sum_j \gamma V^*(s_j)\right\}$

○ $V^*(s_i) = \max\left\{i\ ,\ \frac{1}{6}\cdot\left[-1+\sum_j \gamma V^*(s_j)\right]\right\}$

○ $V^*(s_i) = \max\left\{-\frac{1}{6}+i\ ,\ \sum_j \gamma V^*(s_j)\right\}$

○ $V^*(s_i) = \max\left\{i\ ,\ -\frac{1}{6}+\sum_j \gamma V^*(s_j)\right\}$

○ $V^*(s_i) = \frac{1}{6}\cdot\sum_j \max\left\{i\ ,\ -1+\gamma V^*(s_j)\right\}$

○ $V^*(s_i) = \frac{1}{6}\cdot\sum_j \max\left\{-1+i\ ,\ \sum_k V^*(s_j)\right\}$

○ $V^*(s_i) = \sum_j \max\left\{-1+i\ ,\ \frac{1}{6}\cdot\gamma V^*(s_j))\right\}$

○ $V^*(s_i) = \sum_j \max\left\{\frac{i}{6}\ ,\ -1+\gamma V^*(s_j)\right\}$

● $V^*(s_i) = \max\left\{i\ ,\ -1+\frac{\gamma}{6}\sum_j V^*(s_j)\right\}$

○ $V^*(s_i) = \sum_j \max\left\{i\ ,\ -\frac{1}{6}+\gamma V^*(s_j)\right\}$

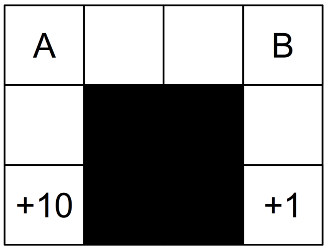
○ $V^*(s_i) = \sum_j \max\left\{\frac{-i}{6}\ ,\ -1+\gamma V^*(s_j)\right\}$

○ Other ________

Q4. [12 pts] MDPs: Value Iteration

An agent lives in gridworld G consisting of grid cells $s \in S$, and is not allowed to move into the cells colored black. In this gridworld, the agent can take actions to move to neighboring squares, when it is not on a numbered square. When the agent is on a numbered square, it is forced to exit to a terminal state (where it remains), collecting a reward equal to the number written on the square in the process.

Gridworld G



You decide to run value iteration for gridworld G. The value function at iteration k is $V_k(s)$. The initial value for all grid cells is 0 (that is, $V_0(s) = 0$ for all $s \in S$). When answering questions about iteration k for $V_k(s)$, either answer with a finite integer or ∞. For all questions, the discount factor is $\gamma = 1$.

(a) Consider running value iteration in gridworld G. Assume all legal movement actions **will always succeed** (and so the state transition function is deterministic).

(i) [2 pts] What is the smallest iteration k for which $V_k(A) > 0$? For this smallest iteration k, what is the value $V_k(A)$?

$k =$ 3 $\qquad V_k(A) =$ 10

The nearest reward is 10, which is 3 steps away. Because $\gamma = 1$, there is no decay in the reward, so the value propagated is 10.

(ii) [2 pts] What is the smallest iteration k for which $V_k(B) > 0$? For this smallest iteration k, what is the value $V_k(B)$?

$k =$ 3 $\qquad V_k(B) =$ 1

The nearest reward is 1, which is 3 steps away. Because $\gamma = 1$, there is no decay in the reward, so the value propagated is 1.

(iii) [2 pts] What is the smallest iteration k for which $V_k(A) = V^*(A)$? What is the value of $V^*(A)$?

$k =$ 3 $\qquad V^*(A) =$ 10

Because $\gamma = 1$, the problem reduces to finding the distance to the highest reward (because there is no living reward). The highest reward is 10, which is 3 steps away.

(iv) [2 pts] What is the smallest iteration k for which $V_k(B) = V^*(B)$? What is the value of $V^*(B)$?

$k =$ 6 $\qquad V^*(B) =$ 10

Because $\gamma = 1$, the problem reduces to finding the distance to the highest reward (because there is no living reward). The highest reward is 10, which is 6 steps away.

(b) [4 pts] Now assume all legal movement actions **succeed with probability** 0.8; with probability 0.2, the action fails and the agent remains in the same state.
Consider running value iteration in gridworld G. What is the smallest iteration k for which $V_k(A) = V^*(A)$? What is the value of $V^*(A)$?

$k =$ ____ ∞ ____

$V^*(A) =$ ____ 10 ____

Because $\gamma = 1$ and the only rewards are in the exit states, the optimal policy will move to the exit state with highest reward. This is guaranteed to ultimately succeed, so the optimal value of state A is 10. However, because the transition is non-deterministic, it's not guaranteed this reward can be collected in 3 steps. It could any number of steps from 3 through infinity, and the values will only have converged after infinitely many iterations.

Q5. [8 pts] Q-learning

Consider the following gridworld (rewards shown on left, state names shown on right).

Rewards

+10	+1

State names

A	B
G1	G2

From state A, the possible actions are right(→) and down(↓). From state B, the possible actions are left(←) and down(↓). For a numbered state (G1, G2), the only action is to exit. Upon exiting from a numbered square we collect the reward specified by the number on the square and enter the end-of-game absorbing state X. We also know that the discount factor $\gamma = 1$, and in this MDP all actions are **deterministic** and always succeed.

Consider the following episodes:

Episode 1 ($E1$)

s	a	s'	r
A	↓	$G1$	0
$G1$	exit	X	10

Episode 2 ($E2$)

s	a	s'	r
B	↓	$G2$	0
$G2$	exit	X	1

Episode 3 ($E3$)

s	a	s'	r
A	→	B	0
B	↓	$G2$	0
$G2$	exit	X	1

Episode 4 ($E4$)

s	a	s'	r
B	←	A	0
A	↓	$G1$	0
$G1$	exit	X	10

(a) [4 pts] Consider using temporal-difference learning to learn $V(s)$. When running TD-learning, all values are initialized to zero.
For which sequences of episodes, if repeated infinitely often, does $V(s)$ converge to $V^*(s)$ for all states s?

(Assume appropriate learning rates such that all values converge.)
Write the correct sequence under "Other" if no correct sequences of episodes are listed.

□ $E1, E2, E3, E4$ □ $E1, E2, E1, E2$ □ $E1, E2, E3, E1$ ■ $E4, E4, E4, E4$
□ $E4, E3, E2, E1$ □ $E3, E4, E3, E4$ □ $E1, E2, E4, E1$

■ Other See explanation below

TD learning learns the value of the executed policy, which is $V^{\pi}(s)$. Therefore for $V^{\pi}(s)$ to converge to $V^*(s)$, it is necessary that the executing policy $\pi(s) = \pi^*(s)$.

Because there is no discounting since $\gamma = 1$, the optimal deterministic policy is $\pi^*(A) = \downarrow$ and $\pi^*(B) = \leftarrow$ ($\pi^*(G1)$ and $\pi^*(G2)$ are trivially exit because that is the only available action). Therefore episodes $E1$ and $E4$ act according to $\pi^*(s)$ while episodes $E2$ and $E3$ are sampled from a suboptimal policy.

From the above, TD learning using episode $E4$ (and optionally $E1$) will converge to $V^{\pi}(s) = V^*(s)$ for states A, B, $G1$. However, then we never visit $G2$, so $V(G2)$ will never converge. If we add either episode $E2$ or $E3$ to ensure that $V(G2)$ converges, then we are executing a suboptimal policy, which will then cause $V(B)$ to not converge. Therefore none of the listed sequences will learn a value function $V^{\pi}(s)$ that converges to $V^*(s)$ for all states s. An example of a correct sequence would be $E2, E4, E4, E4, ...$; sampling $E2$ first with the learning rate $\alpha = 1$ ensures $V^{\pi}(G2) = V^*(G2)$, and then executing $E4$ infinitely after ensures the values for states A, B, and $G1$ converge to the optimal values.

We also accepted the answer such that the value function $V(s)$ converges to $V^*(s)$ for states A and B (ignoring $G1$ and $G2$). TD learning using only episode $E4$ (and optionally $E1$) will converge to $V^\pi(s) = V^*(s)$ for states A and B, therefore the only correct listed option is $E4, E4, E4, E4$.

(b) [4 pts] Consider using Q-learning to learn $Q(s, a)$. When running Q-learning, all values are initialized to zero. For which sequences of episodes, if repeated infinitely often, does $Q(s, a)$ converge to $Q^*(s, a)$ for all state-action pairs (s, a)

(Assume appropriate learning rates such that all Q-values converge.)
Write the correct sequence under "Other" if no correct sequences of episodes are listed.

■ $E1, E2, E3, E4$
■ $E4, E3, E2, E1$
□ $E1, E2, E1, E2$
■ $E3, E4, E3, E4$
□ $E1, E2, E3, E1$
□ $E1, E2, E4, E1$
□ $E4, E4, E4, E4$

□ Other ________

For $Q(s, a)$ to converge, we must visit all state action pairs for non-zero $Q^*(s, a)$ infinitely often. Therefore we must take the exit action in states $G1$ and $G2$, must take the down and right action in state A, and must take the left and down action in state B. Therefore the answers must include $E3$ and $E4$.

Q6. [9 pts] Utilities

PacLad and PacLass are arguing about the value of eating certain numbers of pellets. Neither knows their exact utility functions, but it is known that they are both rational and that PacLad prefers eating more pellets to eating fewer pellets. For any n, let E_n be the event of eating n pellets. So for PacLad, if $m \geq n$, then $E_m \succeq E_n$. For any n and any $k < n$, let $L_{n\pm k}$ refer to a lottery between E_{n-k} and E_{n+k}, each with probability $\frac{1}{2}$.
Reminder: For events A and B, $A \sim B$ denotes that the agent is indifferent between A and B, while $A \succ B$ denotes that A is preferred to B.

(a) [2 pts] Which of the following are guaranteed to be true? Circle TRUE or FALSE accordingly.

(i) TRUE [✓ FALSE] Under PacLad's preferences, for any n, k, $L_{n\pm k} \sim E_n$.
All we know is that PacLad's utility is an increasing function of the number of pellets. One utility function consistent with this is $U(E_n) = 2^n$. Then the expected utility of $L_{2\pm 1}$ is $\frac{1}{2}U(E_1) + \frac{1}{2}U(E_3) = \frac{1}{2}(2+8) = 5$. Since $U(E_2) = 2^2 = 4$, $L_{2\pm 1} \succ E_2$. The only class of utility functions that give the guarantee that this claim is true is linear utility functions. This is a mathematical way of writing the PacLad is risk-neutral; but this is not given as an assumption in the problem. 2^n is a good counterexample because it is a risk-*seeking* utility function. A risk-avoiding utility function would have worked just as well.

(ii) [✓ TRUE] FALSE Under PacLad's preferences, for any k, if $m \geq n$, then $L_{m\pm k} \succeq L_{n\pm k}$.
The expected utility of $L_{m\pm k}$ is $\frac{1}{2}U(E_{m-k}) + \frac{1}{2}U(E_{m+k})$, and that of $L_{n\pm k}$ is $\frac{1}{2}U(E_{n-k}) + \frac{1}{2}U(E_{n+k})$. Since $m - k \geq n - k$, $E_{m-k} \succeq E_{n-k}$, so $U(E_{m-k}) \geq U(E_{n-k})$. Similarly, since $m + k \geq n + k$, $E_{m+k} \succeq E_{n+k}$, so $U(E_{m+k}) \geq U(E_{n+k})$. Thus $\frac{1}{2}U(E_{m-k}) + \frac{1}{2}U(E_{m+k}) \geq \frac{1}{2}U(E_{n-k}) + \frac{1}{2}U(E_{n+k})$ and therefore $L_{m\pm k} \succeq L_{n\pm k}$.

(iii) TRUE [✓ FALSE] Under PacLad's preferences, for any k, l, if $m \geq n$, then $L_{m\pm k} \succeq L_{n\pm l}$.
Consider again the utility function $U(E_n) = 2^n$. It is a risk-seeking utility function as mentioned in part (i), so we should expect that if this were PacLad's utility function, he would prefer a lottery with higher variance (i.e. a higher k value). So for a counterexample, we look to $L_{3\pm 1}$ and $L_{3\pm 2}$ (i.e. $m = n = 3$, $k = 1$, $l = 2$). The expected utility of $L_{3\pm 1}$ is $\frac{1}{2}U(E_2) + \frac{1}{2}U(E_4) = \frac{1}{2}(4 + 16) = 10$. The expected utility of $L_{3\pm 2}$ is $\frac{1}{2}U(E_1) + \frac{1}{2}U(E_5) = \frac{1}{2}(2 + 32) = 17 > 10$. Thus $L_{n\pm l} \succ L_{m\pm k}$. Once again, this is a statement that would only be true for a risk-neutral utility function. A risk-avoiding utility function could also have been used for a counterexample.

(b) To decouple from the previous part, suppose we are given now that under PacLad's preferences, for any n, k, $L_{n\pm k} \sim E_n$. Suppose PacLad's utility function in terms of the number of pellets eaten is U_1. For each of the following, suppose PacLass's utility function, U_2, is defined as given in terms of U_1. Choose **all** statements which are guaranteed to be true of PacLass's preferences under each definition. If none are guaranteed to be true, choose "None." You should assume that all utilities are positive (greater than 0).

(i) [2 pts] $U_2(n) = aU_1(n) + b$ for some positive integers a, b

■ $L_{4\pm 1} \sim L_{4\pm 2}$ ■ $E_4 \succeq E_3$ □ $L_{4\pm 1} \succ E_4$ □ None

The guarantee that under PacLad's preferences for any n, k, $L_{n\pm k} \sim E_n$ means that PacLad is risk-neutral and therefore his utility function is linear. An affine transformation, as this $aU_1(n) + b$ is called, of a linear function is still a linear function, so we have that PacLass's utility function is also linear and thus she is also risk-neutral. Therefore she is indifferent to the variance of lotteries with the same expectation (first option) and she does *not* prefer a lottery to deterministically being given the expectation of that lottery (**not** third option). Since a is positive, U_2 is also an increasing function (second option).

(ii) [2 pts] $U_2(n) = \frac{1}{U_1(n)}$

□ $L_{4\pm 1} \sim L_{4\pm 2}$ □ $E_4 \succeq E_3$ ■ $L_{4\pm 1} \succ E_4$ □ None

Since U_1 is an increasing function, U_2 is decreasing, and thus the preferences over deterministic outcomes are flipped (**not** second option).

The expected utility of $L_{4\pm 1}$ is $\frac{1}{2}(U_2(3) + U_2(5)) = \frac{1}{2}\left(\frac{1}{U_1(3)} + \frac{1}{U_1(5)}\right)$. We know that U_1 is linear, so write $U_1(n) = an + b$ for some a, b. Then substituting this into this expression for $\mathbb{E}[U_2(L_{4\pm 1})]$ and simplifying algebraically yields $\frac{1}{2}\left(\frac{8a+2b}{15a^2+8ab+b^2}\right) = \frac{4a+b}{15a^2+8ab+b^2}$. By the same computation for $L_{4\pm 2}$, we get $\mathbb{E}[U_2(L_{4\pm 2})] = \frac{4a+b}{12a^2+8ab+b^2}$. Since we only know that U_1 is increasing and linear, the only constraint on a and b is that a is positive. So let $a = 1$, $b = 0$. Then $\mathbb{E}[U_2(L_{4\pm 2})] = \frac{1}{3} > \frac{4}{15} = \mathbb{E}[U_2(L_{4\pm 1})]$ and thus $L_{4\pm 2} \succ L_{4\pm 1}$ (**not** first option). Similarly, for this U_1, $U_2(4) = \frac{1}{U_1(4)} = \frac{1}{4} < \frac{1}{3} = \mathbb{E}[U_2(L_{4\pm 2})]$ and thus $L_{4\pm 1} \succ E_4$ (third option).

What follows is a more general argument that could have been used to answer this question if particular numbers were not specified.

In order to determine PacLass's attitude toward risk, we take the second derivative of U_2 with respect to n. By the chain rule, $\frac{\mathrm{d}U_2(n)}{\mathrm{d}n} = \frac{\mathrm{d}U_2(n)}{\mathrm{d}U_1(n)} \cdot \frac{\mathrm{d}U_1(n)}{\mathrm{d}n}$. Since U_1 is an increasing linear function of n, $\frac{\mathrm{d}U_1(n)}{\mathrm{d}n}$ is some positive constant a, so $\frac{\mathrm{d}U_2(n)}{\mathrm{d}n} = a\frac{\mathrm{d}U_2(n)}{\mathrm{d}U_1(n)} = -a\frac{1}{(U_1(n))^2}$. Taking the derivative with respect to n again and using the chain rule yields $\frac{\mathrm{d}^2U_2(n)}{\mathrm{d}n^2} = \frac{\mathrm{d}}{\mathrm{d}U_1(n)}\left(-a\frac{1}{(U_1(n))^2}\right) \cdot \frac{\mathrm{d}U_1(n)}{\mathrm{d}n} = \frac{1}{2}a^2\frac{1}{(U_1(n))^3}$. U_1 is always positive, so this is a positive number and thus the second derivative of PacLass's utility function is everywhere positive. This means the utility function is strictly convex (equivalently "concave up"), and thus all secant lines on the plot of the curve lie above the curve itself.

In general, strictly convex utility functions are risk-seeking. To see this, consider $L_{n\pm k}$ and E_n. The expected utility of $L_{n\pm k}$ is $\frac{1}{2}U_2(n-k) + \frac{1}{2}U_2(n+k)$, which corresponds to the midpoint of the secant line drawn between the points $(n-k, U_2(n-k))$ and $(n+k, U_2(n+k))$, which both lie on the curve. That point is $(n, \mathbb{E}[U(L_{n\pm k})]) = (n, \frac{1}{2}U_2(n-k) + \frac{1}{2}U_2(n+k))$. The utility of E_n is $U(n)$, which lies on the curve at the point $(n, U_2(n))$. Since U_2 is strictly convex, the secant line lies above the curve, so we must have $\mathbb{E}[U_2(L_{n\pm k})] > U(n)$.

With that proof that PacLass is risk-seeking, we can address the remaining two options: she is not indifferent to the variance of a lottery (**not** the first option), and she prefers the lottery over the deterministic outcome (the third option).

PacLass is in a strange environment trying to follow a policy that will maximize her expected utility. Assume that U is her utility function in terms of the number of pellets she receives.

In PacLass's environment, the probability of ending up in state s' after taking action a from state s is $T(s,a,s')$. At every step, PacLass finds a locked chest containing $C(s,a,s')$ pellets, and she can either keep the old chest she is carrying or swap it for the new one she just found. At a terminal state(but never before) she receives the key to open the chest she is carrying and gets all the pellets inside. Each chest has the number of pellets it contains written on it, so PacLass knows how many pellets are inside without opening each chest.

(c) [3 pts] Which is the appropriate Bellman equation for PacLass's value function? Write the correct answer next to 'Other' if none of the listed options are correct.

○ $V^*(s) = \max_a \sum_{s'} T(s,a,s')[U(C(s,a,s')) + V^*(s')]$

○ $V^*(s) = \max_a \sum_{s'} T(s,a,s')U(C(s,a,s') + V^*(s'))$

● $V^*(s) = \max_a \sum_{s'} T(s,a,s') \max\{U(C(s,a,s')), V^*(s')\}$

○ $V^*(s) = \max_a \sum_{s'} T(s,a,s') \max\{U(C(s,a,s')), U(V^*(s'))\}$

○ $V^*(s) = \max_a \sum_{s'} T(s,a,s')U(\max\{C(s,a,s'), V^*(s')\})$

○ $V^*(s) = \max_a \sum_{s'} T(s,a,s')U(\max\{U(C(s,a,s')), V^*(s')\})$

○ Other ________

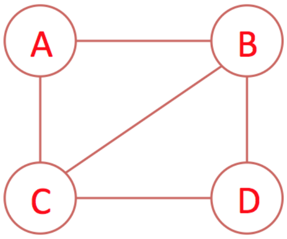
First see that unlike in a normal MDP where we maximize the sum of rewards, PacLass only gets utility from one chest, so her utility is a function of the *maximum* reward she receives. At state s, we choose the action a which maximizes PacLass's expected utility, as normal. To take that expectation, we sum over each outcome s' of taking action a from state s. The terms of that sum are the probability of each outcome multiplied with the utility of each action. In a normal (undiscounted) MDP, the utility of the triple (s,a,s') is $[R(s,a,s') + V^*(s')]$. Here, instead of taking the sum, we have to take the max. But in this MDP, unlike in a normal MDP, we have a unit mismatch (equivalently a type mismatch) between the rewards, which are in units of food pellets, and PacLass's utility (which is in general units of utility). This is crucially important because PacLass's utility is not given to be increasing; maximizing $C(s,a,s')$ directly is not guaranteed to maximize utility. Since value is defined to be the expected utility of acting optimally starting from state s, V^* represents a utility, so it does not make sense to take $U(V^*(s'))$. We must compare the utility of taking the new chest containing $C(s, a, s')$ pellets, $U(C(s, a, s'))$ to the utility of taking some other chest, $V^*(s')$. Thus the only correct answer is the third option.

Q7. [17 pts] CSPs with Preferences

Let us formulate a CSP with variables A, B, C, D, and domains of $\{1, 2, 3\}$ for each of these variables. A **valid assignment** in this CSP is defined as a complete assignment of values to variables which satisfies the following constraints:

1. B will not ride in car 2.
2. A and B refuse to ride in the same car.
3. The sum of the car numbers for B and C is less than 4.
4. A's car number must be greater than C's car number.
5. B and D refuse to ride in the same car.
6. C's car number must be lesser than D's car number.

(a) [2 pts] Draw the corresponding constraint graph for this CSP.



Although there are several valid assignments which exist for this problem, A, B, C and D have additional "soft" preferences on which value they prefer to be assigned. To encode these preferences, we define utility functions $U_{Var}(Val)$ which represent how preferable an assignment of the value(Val) to the variable(Var) is.

For a complete assignment $P = \{A : V_A, B : V_B, ...D : V_D\}$, the utility of P is defined as the sum of the utility values: $U_A(V_A) + U_B(V_B) + U_C(V_C) + U_D(V_D)$. A higher utility for P indicates a higher preference for that complete assignment. This scheme can be extended to an arbitrary CSP, with several variables and values.

We can now define a modified CSP problem, whose goal is to find the valid assignment which has the maximum utility amongst all valid assignments.

(b) [2 pts] Suppose the utilities for the assignment of values to variables is given by the table below

U	U_A	U_B	U_C	U_D
1	7	10	200	2000
2	6	20	300	1000
3	5	30	100	3000

Under these preferences, given a choice between the following complete assignments which are valid solutions to the CSP, which would be the preferred solution.

○ A:3 B:1 C:1 D:2

● A:3 B:1 C:2 D:3

○ A:2 B:1 C:1 D:2

○ A:3 B:1 C:1 D:3

Solution 2 has value $U_A(3) + U_B(1) + U_C(2) + U_D(3) = 5 + 10 + 300 + 3000 = 3315$, which is the highest amongst the choices

To decouple from the previous questions, for the rest of the question, the preference utilities are not necessarily the table shown above but can be arbitrary positive values.

This problem can be formulated as a modified search problem, where we use the modified tree search shown below to find the valid assignment with the highest utility, instead of just finding an arbitrary valid assignment.

The search formulation is:

- State space: The space of partial assignments of values to variables
- Start state: The empty assignment
- Goal Test: State X is a valid assignment
- Successor function: The successors of a node X are states which have partial assignments which are the assignment in X extended by one more assignment of a value to an unassigned variable, as long as this assignment does not violate any constraints
- Edge weights: Utilities of the assignment made through that edge

In the algorithm below $f(node)$ is an **estimator of distance** from *node* to *goal*, ACCUMULATED-UTILITY-FROM-START(*node*) is the sum of utilities of assignments made from the *start-node* to the current *node*.

```
function ModifiedTreeSearch(problem, start-node)
    fringe ← Insert(key : start-node, value : $f(start\text{-}node)$)
    do
        if IsEmpty(fringe) then
            return failure
        end if
        node, cost ← remove entry with maximum value from fringe
        if Goal-Test(node) then
            return node
        end if
        for child in Successors(node) do
            fringe ← Insert(key : child, value : $f(child)$ + Accumulated-Utility-From-Start(child))
        end for
    while True
end function
```

(c) Under this search formulation, for a node X with assigned variables $\{v_1, v_2....v_n\}$ and unassigned variables $\{u_1, u_2, u_3...u_m\}$

(i) [4 pts] Which of these expressions for $f(X)$ in the algorithm above, is guaranteed to give an optimal assignment according to the preference utilities. (Select **all** that apply)

□ $f_1 = \min_{Val_1, Val_2, ...Val_m} U_{u_1}(Val_1) + U_{u_2}(Val_2) + ... + U_{u_m}(Val_m)$

■ $f_2 = \max_{Val_1, Val_2, ...Val_m} U_{u_1}(Val_1) + U_{u_2}(Val_2) + ... + U_{u_m}(Val_m)$

□ $f_3 = \min_{Val_1, Val_2, ...Val_m} U_{u_1}(Val_1) + U_{u_2}(Val_2) + ... + U_{u_m}(Val_m)$ such that the complete assignment satisfies constraints.

■ $f_4 = \max_{Val_1, Val_2, ...Val_m} U_{u_1}(Val_1) + U_{u_2}(Val_2) + ... + U_{u_m}(Val_m)$ such that the complete assignment satisfies constraints.

■ $f_5 = Q$, a fixed extremely high value ($\gg$ sum of all utilities) which is the same across all states

□ $f_6 = 0$

Because we have a maximum search we need an overestimator of cost instead of an underestimator for the function f, like standard A^* search. ModifiedTreeSearch is A^* search picking the maximum node from the fringe instead of the minimum. This requires an overestimator instead of an understimator to ensure optimality of the tree search.

(ii) [3 pts] For the expressions for $f(X)$ which guaranteed to give an optimal solution in part(i) among $f_1, f_2, f_3, f_4, f_5, f_6$, order them in ascending order of number of nodes expanded by ModifiedTreeSearch. Based on the dominance of heuristics, but modified to be an overestimate instead of an underestimate in standard A* search. Hence the closer the estimate is to the actual cost, the better it does in terms of number of nodes expanded. So the ordering is option 4 < option 2 < option 5.

(d) In order to make this search more efficient, we want to perform forward checking such that, for every assignment of a value to a variable, we eliminate values from the domains of other variables, which violate a constraint under this assignment. Answer the following questions formulating the state space and successor function for a search problem such that the same algorithm [1] performs forward checking under this formulation.

(i) [3 pts] Briefly describe the minimal state space representation for this problem? (No state space size is needed, just a description will suffice)

Each element of the state space is a partial assignment along with the domains of all variables

(ii) [3 pts] What is the Successor function for this problem?

The successors for a node X, are generated by picking an unassigned variable and a corresponding value to assign to it. The successor state has a partial assignment which is the partial assignment of X, extended by the new value assignment which we picked. It is important then to also prune the domains of the remaining unassigned variables using forward checking to remove values which would violate constraints under the new assignment. Successor states which have empty domains or violated constraints are removed from the list of successors.

Q8. [19 pts] Game Trees: Friendly Ghost

Consider a two-player game between Pacman and a ghost in which both agents alternate moves. As usual, Pacman is a maximizer agent whose goal is to win by maximizing his own utility. Unlike the usual adversarial ghost, she is friendly and helps Pacman by maximizing his utility. Pacman is unaware of this and acts as usual (i.e. as if she is playing against him). She knows that Pacman is misinformed and acts accordingly.

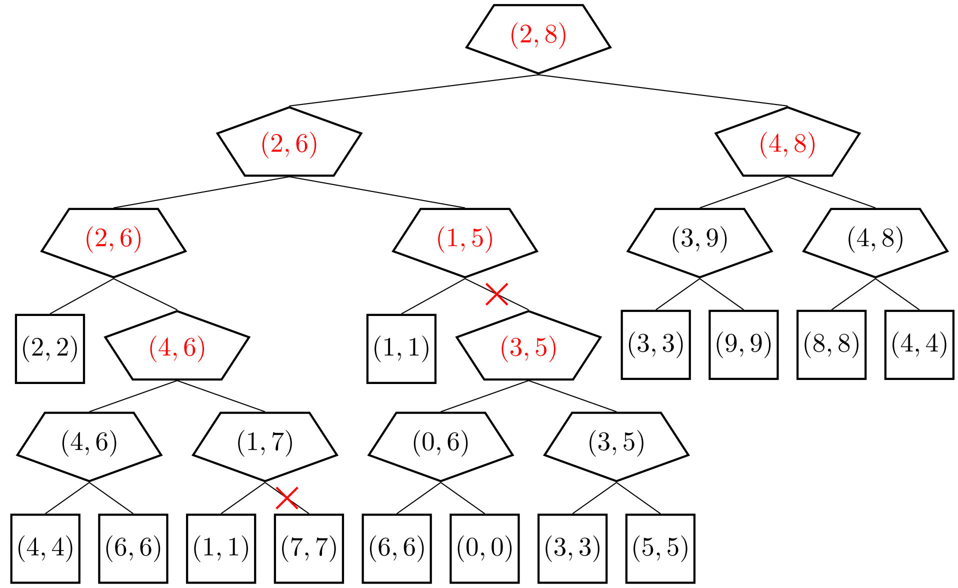
(a) [7 pts] In the minimax algorithm, the value of each node is determined by the game subtree hanging from that node. For this version, we instead define a value pair (u, v) for each node:

- u is the value of the subtree as determined by Pacman, who acts to win while assuming that the ghost is a minimizer agent, and
- v is the value of the subtree as determined by the ghost, who acts to help Pacman win while **knowing** Pacman's strategy.

For example, in the subtree below with values (4, 6), Pacman believes the ghost would choose the left action which has a value of 4, but in fact the ghost chooses the right action which has a value of 6, since that is better for Pacman.

For the terminal states we set $u = v = \text{Utility}(State)$.

Fill in the remaining (u, v) values in the modified minimax tree below, in which **the ghost is the root.** The ghost nodes are upside down pentagons (⯂) and Pacman's nodes are rightside up pentagons (⬠).



The u value of Pacman's nodes is the maximum of the u values of the immediate children nodes since Pacman believes that the values of the nodes are given by u. The v value of Pacman's nodes is the v value from the child node that attains the maximum u value since, during Pacman's turn, he determines the action that is taken.

The u value of the ghost nodes is the minimum of the u values of the immediate children nodes since Pacman believes the ghost would choose the action that minimizes his utility. The v value of the ghost nodes is the maximum of the v values of the immediate children nodes since, during her turn, she chooses the action that maximizes Pacman's utility.

The value of this game, where the goal is to act optimally given the limited information, is 8. Notice that the u values are minimax values since Pacman believes he is playing a minimax game.

For grading purposes, we marked down points if the value of a node is incorrect *given* the values of the immediate children nodes. That is, we penalized only once for each mistake and propagated the error for the values above. This also means that a value that is the same as in the solutions could be marked as incorrect if its value should be different when using the values of the children nodes provided by the student.

(b) [3 pts] In the game tree above, put an 'X' on the branches that can be pruned and do not need to be explored when the ghost computes the value of the tree. Assume that the children of a node are visited in left-to-right order and that you should **not** prune on equality. Explicitly write down "Not possible" below if no branches can be pruned, in which case any 'X' marks above will be ignored.

Two branches can be pruned and they are marked on the tree above.

Branches coming down from Pacman's nodes can never be pruned since the v value from one of the children nodes might be needed by the ghost node above Pacman's, even if the u value is no longer needed. For instance, if the game was simply minimax, the branch between the nodes with values $(4, 8)$ would have been pruned. However, notice that in the modified game, the value 8 needed to be passed up the tree.

On the other hand, branches coming down from the ghost nodes can be pruned if we can rule out that in the previous turn Pacman would pick the action leading to this node. For instance, the branch above the leave with values $(7, 7)$ can be pruned since Pacman's best u value on path to root is 4 by the time this branch is reached, but the ghost node already explored a subtree with a u value of 1.

(c) [1 pt] What would the value of the game tree be if instead Pacman **knew** that the ghost is friendly?

Value (i.e. a single number) at the root of the game tree is 9

In this game where Pacman knows that the ghost is friendly, both players are maximizer players, so the value of the game tree is the maximum of all the leaves.

(d) [4 pts] Complete the algorithm below, which is a modification of the minimax algorithm, to work in the **original setting** where the ghost is friendly unbeknownst to Pacman. (No pruning in this subquestion)

```
function VALUE($state$)
    if $state$ is leaf then
        $(u, v) \leftarrow$ (UTILITY($state$), UTILITY($state$))
        return $(u, v)$
    end if
    if $state$ is Ghost-Node then
        return GHOST-VALUE($state$)
    else
        return PACMAN-VALUE($state$)
    end if
end function
```

```
function GHOST-VALUE($state$)
    $(u, v) \leftarrow (+\infty, -\infty)$
    for $successor$ in SUCCESSORS($state$) do
        $(u', v') \leftarrow$ VALUE($successor$)
        (i)
        (ii)
        $(u, v) \leftarrow (\bar{u}, \bar{v})$
    end for
    return $(u, v)$
end function
```

```
function PACMAN-VALUE($state$)
    $(u, v) \leftarrow (-\infty, +\infty)$
    for $successor$ in SUCCESSORS($state$) do
        $(u', v') \leftarrow$ VALUE($successor$)
        (iii)
        (iv)
        $(u, v) \leftarrow (\bar{u}, \bar{v})$
    end for
    return $(u, v)$
end function
```

Complete the pseudocode by choosing the option that fills in each blank above. The code blocks $\mathbf{A_1}$–$\mathbf{A_8}$ update $\bar{u}$ and the code blocks $\mathbf{B_1}$–$\mathbf{B_8}$ update $\bar{v}$. If any of the code blocks are not needed, the correct answer for that question **must** mark the option 'None of these code blocks are needed'.

$\mathbf{A_1}$
```
if $u' < u$ then
    $\bar{u} \leftarrow u'$
end if
```

$\mathbf{A_2}$
```
if $u' < v$ then
    $\bar{u} \leftarrow u'$
end if
```

$\mathbf{A_3}$
```
if $v' < u$ then
    $\bar{u} \leftarrow u'$
end if
```

$\mathbf{A_4}$
```
if $v' < v$ then
    $\bar{u} \leftarrow u'$
end if
```

$\mathbf{A_5}$
```
if $u' > u$ then
    $\bar{u} \leftarrow u'$
end if
```

$\mathbf{A_6}$
```
if $u' > v$ then
    $\bar{u} \leftarrow u'$
end if
```

$\mathbf{A_7}$
```
if $v' > u$ then
    $\bar{u} \leftarrow u'$
end if
```

$\mathbf{A_8}$
```
if $v' > v$ then
    $\bar{u} \leftarrow u'$
end if
```

$\mathbf{B_1}$
```
if $u' < u$ then
    $\bar{v} \leftarrow v'$
end if
```

$\mathbf{B_2}$
```
if $u' < v$ then
    $\bar{v} \leftarrow v'$
end if
```

$\mathbf{B_3}$
```
if $v' < u$ then
    $\bar{v} \leftarrow v'$
end if
```

$\mathbf{B_4}$
```
if $v' < v$ then
    $\bar{v} \leftarrow v'$
end if
```

$\mathbf{B_5}$
```
if $u' > u$ then
    $\bar{v} \leftarrow v'$
end if
```

$\mathbf{B_6}$
```
if $u' > v$ then
    $\bar{v} \leftarrow v'$
end if
```

$\mathbf{B_7}$
```
if $v' > u$ then
    $\bar{v} \leftarrow v'$
end if
```

$\mathbf{B_8}$
```
if $v' > v$ then
    $\bar{v} \leftarrow v'$
end if
```

(i) [1 pt]
● $\mathbf{A_1}$ ○ $\mathbf{A_2}$ ○ $\mathbf{A_3}$ ○ $\mathbf{A_4}$
○ $\mathbf{A_5}$ ○ $\mathbf{A_6}$ ○ $\mathbf{A_7}$ ○ $\mathbf{A_8}$ ○ None of these code blocks are needed

(ii) [1 pt]
○ $\mathbf{B_1}$ ○ $\mathbf{B_2}$ ○ $\mathbf{B_3}$ ○ $\mathbf{B_4}$
○ $\mathbf{B_5}$ ○ $\mathbf{B_6}$ ○ $\mathbf{B_7}$ ● $\mathbf{B_8}$ ○ None of these code blocks are needed

(iii) [1 pt]
○ $\mathbf{A_1}$ ○ $\mathbf{A_2}$ ○ $\mathbf{A_3}$ ○ $\mathbf{A_4}$
● $\mathbf{A_5}$ ○ $\mathbf{A_6}$ ○ $\mathbf{A_7}$ ○ $\mathbf{A_8}$ ○ None of these code blocks are needed

(iv) [1 pt]
○ $\mathbf{B_1}$ ○ $\mathbf{B_2}$ ○ $\mathbf{B_3}$ ○ $\mathbf{B_4}$
● $\mathbf{B_5}$ ○ $\mathbf{B_6}$ ○ $\mathbf{B_7}$ ○ $\mathbf{B_8}$ ○ None of these code blocks are needed

As stated in part **(a)**, the u and v values for the ghost node is **(i)** the minimum of the u values and **(ii)** the maximum of the v values of the children nodes, while the u and v values for Pacman's node is **(iii)** the maximum of the u values and **(iv)** the v value that attains the maximum u value among the u values of the children nodes.

(e) [4 pts] Complete the algorithm below, which is a modification of the alpha-beta pruning algorithm, to work in the original setting where the ghost is friendly unbeknownst to Pacman.
We want to compute Value(Root Node, $\alpha = -\infty$, $\beta = +\infty$). You should **not** prune on equality.
Hint: you might not need to use α or β, or none of them (e.g. no pruning is possible).

```
function VALUE(state, $\alpha$, $\beta$)
    if state is leaf then
        $(u, v) \leftarrow$ (UTILITY(state), UTILITY(state))
        return $(u, v)$
    end if
    if state is Ghost-Node then
        return GHOST-VALUE(state, $\alpha$, $\beta$)
    else
        return PACMAN-VALUE(state, $\alpha$, $\beta$)
    end if
end function
```

```
function GHOST-VALUE(state, $\alpha$, $\beta$)
    $(u, v) \leftarrow (+\infty, -\infty)$
    for successor in SUCCESSORS(state) do
        $(u', v') \leftarrow$ VALUE(successor, $\alpha$, $\beta$)
        ...                          # same as before
        $(u, v) \leftarrow (\bar{u}, \bar{v})$
        (i)
        (ii)
    end for
    return $(u, v)$
end function
```

```
function PACMAN-VALUE(state, $\alpha$, $\beta$)
    $(u, v) \leftarrow (-\infty, +\infty)$
    for successor in SUCCESSORS(state) do
        $(u', v') \leftarrow$ VALUE(successor, $\alpha$, $\beta$)
        ...                          # same as before
        $(u, v) \leftarrow (\bar{u}, \bar{v})$
        (iii)
        (iv)
    end for
    return $(u, v)$
end function
```

Complete the pseudocode by choosing the option that fills in each blank above. The code blocks **C_1–C_8** prune the search and the code blocks **D_1–D_8** update α and β. If any of the code blocks are not needed, the correct answer for that question **must** mark the option 'None of these code blocks are needed'.

C_1	**if** $u < \alpha$ **then**
	return (u, v)
	end if
		C_2	**if** $v < \alpha$ **then**
			return (u, v)
			end if
				C_3	**if** $u < \beta$ **then**
					return (u, v)
					end if
						C_4	**if** $v < \beta$ **then**
							return (u, v)
							end if

C_5	**if** $u > \alpha$ **then**
	return (u, v)
	end if
		C_6	**if** $v > \alpha$ **then**
			return (u, v)
			end if
				C_7	**if** $u > \beta$ **then**
					return (u, v)
					end if
						C_8	**if** $v > \beta$ **then**
							return (u, v)
							end if

D_1	$\alpha \leftarrow \min(\alpha, u)$	**D_2**	$\alpha \leftarrow \min(\alpha, v)$	**D_3**	$\beta \leftarrow \min(\beta, u)$	**D_4**	$\beta \leftarrow \min(\beta, v)$
D_5	$\alpha \leftarrow \max(\alpha, u)$	**D_6**	$\alpha \leftarrow \max(\alpha, v)$	**D_7**	$\beta \leftarrow \max(\beta, u)$	**D_8**	$\beta \leftarrow \max(\beta, v)$

(i) [1 pt] ● **C_1** ○ **C_2** ○ **C_3** ○ **C_4**
○ **C_5** ○ **C_6** ○ **C_7** ○ **C_8** ○ None of these code blocks are needed

(ii) [1 pt] ○ **D_1** ○ **D_2** ○ **D_3** ○ **D_4**
○ **D_5** ○ **D_6** ○ **D_7** ○ **D_8** ● None of these code blocks are needed

(iii) [1 pt] ○ **C_1** ○ **C_2** ○ **C_3** ○ **C_4**
○ **C_5** ○ **C_6** ○ **C_7** ○ **C_8** ● None of these code blocks are needed

(iv) [1 pt] ○ **D_1** ○ **D_2** ○ **D_3** ○ **D_4**
● **D_5** ○ **D_6** ○ **D_7** ○ **D_8** ○ None of these code blocks are needed

As stated in part **(b)**, it is possible to prune based on Pacman's best option on path to root just as in minimax (**(i)** and **(iv)**), but it is not possible to prune based on the ghost's best option on path to root (**(ii)** and **(iii)**).