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	

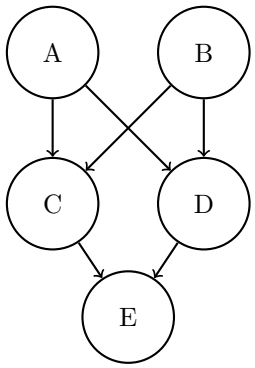
For staff use only:

Total	/??

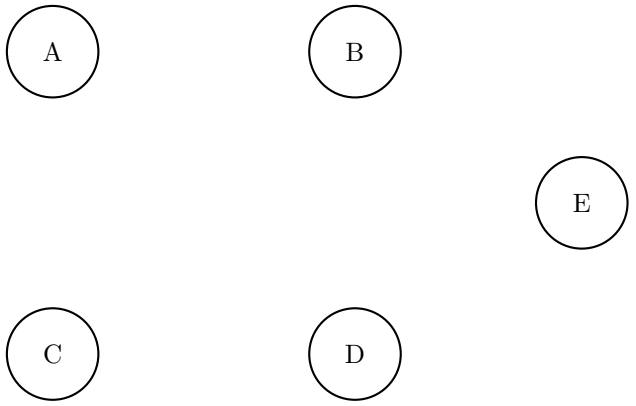
THIS PAGE IS INTENTIONALLY LEFT BLANK

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:



(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 $P(A) \cdot P(B|A) \cdot P(C) \cdot P(C|A) \cdot P(D)$

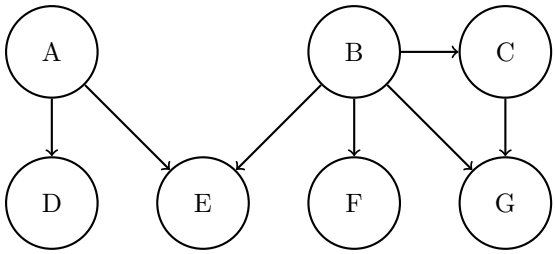
(iv) TRUE FALSE $P(A|B) \cdot P(B|C) \cdot P(C|D) \cdot P(D|A)$

(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] $P(A) \cdot P(B|A) \cdot P(C|A) \cdot P(E|B, C, D)$

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

(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?

(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?

- Which choice would create the **smallest** factor f_1?

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.

(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.

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*
$V^{\pi}(s)$						

(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*					*Stop*

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

(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.

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

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

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

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

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

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

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

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

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

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

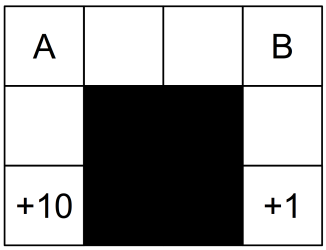
◯ $V^*(s_i) = \sum_j \max\left\{\frac{-i}{6} \ , \ -1 + \gamma V^*(s_j)\right\}$

◯ 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 $V_k(A)$?

$k =$_____ $V_k(A) =$_____

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

$k =$_____ $V_k(B) =$_____

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

$k =$_____ $V^*(A) =$_____

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

$k =$_____ $V^*(B) =$_____

(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 $V^*(A)$?

$k =$_____

$V^*(A) =$_____

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($\rightarrow$) and down($\downarrow$). From state B, the possible actions are left($\leftarrow$) and down($\downarrow$). 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	$\downarrow$	$G1$	0
$G1$	exit	X	10

Episode 2 ($E2$)

s	a	s'	r
B	$\downarrow$	$G2$	0
$G2$	exit	X	1

Episode 3 ($E3$)

s	a	s'	r
A	$\rightarrow$	B	0
B	$\downarrow$	$G2$	0
$G2$	exit	X	1

Episode 4 ($E4$)

s	a	s'	r
B	$\leftarrow$	A	0
A	$\downarrow$	$G1$	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 _____

(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.

☐ $E1, E2, E3, E4$ ☐ $E1, E2, E1, E2$ ☐ $E1, E2, E3, E1$ ☐ $E4, E4, E4, E4$
☐ $E4, E3, E2, E1$ ☐ $E3, E4, E3, E4$ ☐ $E1, E2, E4, E1$

☐ Other _____

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, $L_{n\pm k} \sim E_n$.

(ii) TRUE FALSE Under PacLad's preferences, for any k, if $m \geq n$, then $L_{m\pm k} \succeq L_{n\pm k}$.

(iii) TRUE FALSE Under PacLad's preferences, for any k, l, if $m \geq n$, then $L_{m\pm k} \succeq L_{n\pm l}$.

(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

☐ $L_{4\pm1} \sim L_{4\pm2}$ ☐ $E_4 \succeq E_3$ ☐ $L_{4\pm1} \succ E_4$ ☐ None

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

☐ $L_{4\pm1} \sim L_{4\pm2}$ ☐ $E_4 \succeq E_3$ ☐ $L_{4\pm1} \succ E_4$ ☐ None

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.

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

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

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

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

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

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

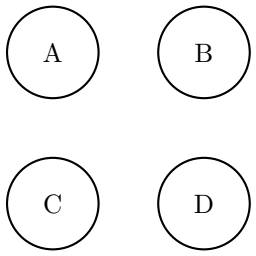
◯ Other _____

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

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-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
- ☐ $f_6 = 0$

(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.

(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)

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

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.

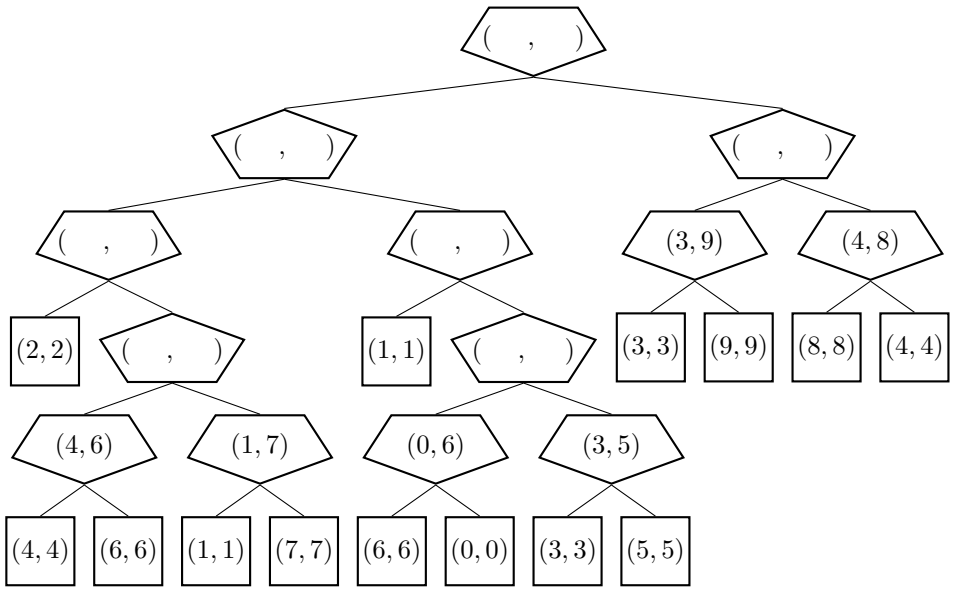
(a) [7 pts] In the minimax algorithm, the value of each node is determined by the game subtree hanging from that node. For this version, we instead define a value pair (u, v) for each node:

- 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 (⬠).



(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.

(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 _____

(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) ← (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) ← (+∞, −∞)
    for successor in SUCCESSORS(state) do
        (u′, v′) ← VALUE(successor)
        [ (i) ]
        [ (ii) ]
        (u, v) ← (ū, v̄)
    end for
    return (u, v)
end function
```

```
function PACMAN-VALUE(state)
    (u, v) ← (−∞, +∞)
    for successor in SUCCESSORS(state) do
        (u′, v′) ← VALUE(successor)
        [ (iii) ]
        [ (iv) ]
        (u, v) ← (ū, v̄)
    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
    ū ← u′
end if
```

$\mathbf{A_2}$
```
if u′ < v then
    ū ← u′
end if
```

$\mathbf{A_3}$
```
if v′ < u then
    ū ← u′
end if
```

$\mathbf{A_4}$
```
if v′ < v then
    ū ← u′
end if
```

$\mathbf{A_5}$
```
if u′ > u then
    ū ← u′
end if
```

$\mathbf{A_6}$
```
if u′ > v then
    ū ← u′
end if
```

$\mathbf{A_7}$
```
if v′ > u then
    ū ← u′
end if
```

$\mathbf{A_8}$
```
if v′ > v then
    ū ← u′
end if
```

$\mathbf{B_1}$
```
if u′ < u then
    v̄ ← v′
end if
```

$\mathbf{B_2}$
```
if u′ < v then
    v̄ ← v′
end if
```

$\mathbf{B_3}$
```
if v′ < u then
    v̄ ← v′
end if
```

$\mathbf{B_4}$
```
if v′ < v then
    v̄ ← v′
end if
```

$\mathbf{B_5}$
```
if u′ > u then
    v̄ ← v′
end if
```

$\mathbf{B_6}$
```
if u′ > v then
    v̄ ← v′
end if
```

$\mathbf{B_7}$
```
if v′ > u then
    v̄ ← v′
end if
```

$\mathbf{B_8}$
```
if v′ > v then
    v̄ ← 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

(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, α, β)
    if state is leaf then
        (u, v) ← (UTILITY(state), UTILITY(state))
        return (u, v)
    end if
    if state is Ghost-Node then
        return GHOST-VALUE(state, α, β)
    else
        return PACMAN-VALUE(state, α, β)
    end if
end function
```

```
function GHOST-VALUE(state, α, β)
    (u, v) ← (+∞, −∞)
    for successor in SUCCESSORS(state) do
        (u′, v′) ← VALUE(successor, α, β)
                ...                  # same as before
        (u, v) ← (ū, v̄)
        [ (i) ]
        [ (ii) ]
    end for
    return (u, v)
end function
```

```
function PACMAN-VALUE(state, α, β)
    (u, v) ← (−∞, +∞)
    for successor in SUCCESSORS(state) do
        (u′, v′) ← VALUE(successor, α, β)
                ...                  # same as before
        (u, v) ← (ū, v̄)
        [ (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 $\mathbf{C_1}$–$\mathbf{C_8}$ prune the search and the code blocks $\mathbf{D_1}$–$\mathbf{D_8}$ 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'.

$\mathbf{C_1}$
```
if u < α then
    return (u, v)
end if
```

$\mathbf{C_2}$
```
if v < α then
    return (u, v)
end if
```

$\mathbf{C_3}$
```
if u < β then
    return (u, v)
end if
```

$\mathbf{C_4}$
```
if v < β then
    return (u, v)
end if
```

$\mathbf{C_5}$
```
if u > α then
    return (u, v)
end if
```

$\mathbf{C_6}$
```
if v > α then
    return (u, v)
end if
```

$\mathbf{C_7}$
```
if u > β then
    return (u, v)
end if
```

$\mathbf{C_8}$
```
if v > β then
    return (u, v)
end if
```

$\mathbf{D_1}$ $\alpha \leftarrow \min(\alpha, u)$ **$\mathbf{D_2}$** $\alpha \leftarrow \min(\alpha, v)$ **$\mathbf{D_3}$** $\beta \leftarrow \min(\beta, u)$ **$\mathbf{D_4}$** $\beta \leftarrow \min(\beta, v)$

$\mathbf{D_5}$ $\alpha \leftarrow \max(\alpha, u)$ **$\mathbf{D_6}$** $\alpha \leftarrow \max(\alpha, v)$ **$\mathbf{D_7}$** $\beta \leftarrow \max(\beta, u)$ **$\mathbf{D_8}$** $\beta \leftarrow \max(\beta, v)$

(i) [1 pt] ◯ $\mathbf{C_1}$ ◯ $\mathbf{C_2}$ ◯ $\mathbf{C_3}$ ◯ $\mathbf{C_4}$ ◯ $\mathbf{C_5}$ ◯ $\mathbf{C_6}$ ◯ $\mathbf{C_7}$ ◯ $\mathbf{C_8}$ ◯ None of these code blocks are needed

(ii) [1 pt] ◯ $\mathbf{D_1}$ ◯ $\mathbf{D_2}$ ◯ $\mathbf{D_3}$ ◯ $\mathbf{D_4}$ ◯ $\mathbf{D_5}$ ◯ $\mathbf{D_6}$ ◯ $\mathbf{D_7}$ ◯ $\mathbf{D_8}$ ◯ None of these code blocks are needed

(iii) [1 pt] ◯ $\mathbf{C_1}$ ◯ $\mathbf{C_2}$ ◯ $\mathbf{C_3}$ ◯ $\mathbf{C_4}$ ◯ $\mathbf{C_5}$ ◯ $\mathbf{C_6}$ ◯ $\mathbf{C_7}$ ◯ $\mathbf{C_8}$ ◯ None of these code blocks are needed

(iv) [1 pt] ◯ $\mathbf{D_1}$ ◯ $\mathbf{D_2}$ ◯ $\mathbf{D_3}$ ◯ $\mathbf{D_4}$ ◯ $\mathbf{D_5}$ ◯ $\mathbf{D_6}$ ◯ $\mathbf{D_7}$ ◯ $\mathbf{D_8}$ ◯ None of these code blocks are needed