CS 188 Fall 2024 — Introduction to Artificial Intelligence — Midterm

- You have 110 minutes.
- The exam is closed book, no calculator, and closed notes, other than one double-sided cheat sheet that you may reference.
- For multiple choice questions,
  - ☐ means mark **all options** that apply
  - ◯ means mark a **single choice**

First name	
Last name	
SID	
Name of person to the right	
Name of person to the left	
Discussion TAs (or None)	

Honor code: "As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others."

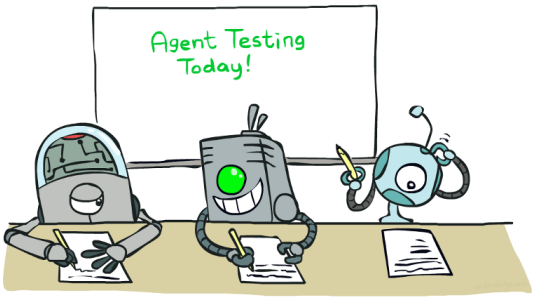
By signing below, I affirm that all work on this exam is my own work, and honestly reflects my own understanding of the course material. I have not referenced any outside materials (other than my cheat sheets), nor collaborated with any other human being on this exam. I understand that if the exam proctor catches me cheating on the exam, that I may face the penalty of an automatic "F" grade in this class and a referral to the Center for Student Conduct.

Signature: _____

Point Distribution

Q1.	Word Game	19
Q2.	Easter Island	15
Q3.	Pacman.io	19
Q4.	Pranav's Phunky Puzzle	14
Q5.	Spelunking	16
Q6.	Potpourri: Arbitrary Abstraction	17
	Total	100

It's testing time for our CS188 robots!
Circle your favorite robot below.
(ungraded, just for fun)



Q1. [19 pts] Word Game

Consider the following word game. You start with 4 letters in a sequence, and you want to transform the sequence into a goal sequence by swapping adjacent letters. On each turn, you can choose one of the following actions, each with cost 1:

- Swap the 1st and 2nd letters.
- Swap the 2nd and 3rd letters.
- Swap the 3rd and 4th letters.

The objective of the game is to get to the goal sequence with the minimum number of swaps. Here is an example:

Goal: *HATS*			
H	*S*	*A*	*T*
Swap *A* and *S*			
H	*A*	*S*	*T*
Swap *T* and *S*			
H	*A*	*T*	*S*

When we call the successor function on a given state, we add the successor states onto the fringe in the following order:

1. The state after swapping the 1st and 2nd letters.
2. The state after swapping the 2nd and 3rd letters.
3. The state after swapping the 3rd and 4th letters.

For example, when we call the successor function on state "ABCD", we add [BACD, ACBD, ABDC] on the fringe, in that order.

For parts (a) and (b), consider the following start sequence and goal sequence:

Goal: *TOOK*			
K	*O*	*T*	*O*

(a) [4 pts] What is the resulting solution from running BFS graph search?

- ◯ KOTO → OKTO → OTKO → **TOKO** → TOOK
- ◯ KOTO → OKTO → OTKO → **OTOK** → TOOK
- ◯ KOTO → **KTOO** → TKOO → TOKO → TOOK
- ◯ BFS fails to find a solution

(b) [4 pts] Running DFS results in this solution: KOTO → KOOT → OKOT → OKTO → OTKO → OTOK → TOOK

How many states were *expanded* to find this solution during DFS graph search?

Hint: You expand a state when you call the successor function on that state.

◯ 4 ◯ 5 ◯ 6 ◯ 7 ◯ 8 ◯ 9 ◯ 10

(c) [4 pts] Select all admissible heuristics.

- ☐ The number of letters not in the right place.
- ☐ The sum of distances for each letter between its current position and the closest identical letter in the goal sequence.
- ☐ For each adjacent pair (i.e., 1st and 2nd letters, 2nd and 3rd letters, 3rd and 4th letters), count the number of pairs that do not match the corresponding pair in the goal sequence.
- ☐ The minimum distance from any letter to the closest identical letter in the goal sequence.
- ◯ None of the above.

For the rest of this question, we change the cost of each action as follows:

- Swap 1st and 2nd letters: Cost 1.
- Swap 2nd and 3rd letters: Cost 2.
- Swap 3rd and 4th letters: Cost 3.

If two states on the fringe have the same priority, break ties alphabetically. For example, if ABCD and BACD have the same priority, pop ABCD off the fringe first.

(d) [3 pts] Which statements are true about running UCS graph search on this game, assuming a solution exists?

- ☐ UCS will always find a solution.
- ☐ UCS will always return the lowest-cost solution.
- ☐ UCS will always expand fewer states than BFS.
- ◯ None of the above

Consider the following heuristic: The maximum distance from any letter to its position in the goal sequence.

For example, for the goal "ABCD," the state "CDBA" has a heuristic of 3, because A has distance 3, B has distance 1, C has distance 2, and D has distance 2.

For part (e), consider the following start sequence and goal sequence:

Goal: *WORK*			
O	*K*	*R*	*W*

(e) [4 pts] Which solution is returned by greedy graph search?

- ◯ OKRW → **ORKW** → ORWK → OWRK → WORK
- ◯ OKRW → OKWR → OWKR → **OWRK** → WORK
- ◯ OKRW → OKWR → OWKR → **WOKR** → WORK

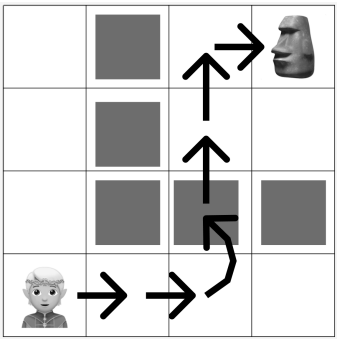
Q2. [15 pts] Easter Island

Edgar is an elf on Easter Island, seeking Pietru the Paradise Dweller—an ancient Moai statue—for midterm wisdom.

Easter Island is represented as an $N \times N$ grid, with k randomly placed obstacles. At each step, Edgar can either move up, down, left, or right (if unobstructed). He also has 3 jumps to use during the journey.

A jump is an action that puts him on top of an obstacle adjacent to him in the direction he chooses. For example, in the figure to the right, Edgar jumps north from $(3, 0)$ to the top of the obstacle at $(3, 1)$. **He may continue to walk on top of adjacent obstacles until he descends**. Descending does not require him to use a jump.

Edgar knows the location of all obstacles, as well as his starting position and the position of the Moai statue on the grid. Edgar also knows what the legal moves are, at every position. If there are no legal moves—for example, if he's surrounded by obstacles with no jumps remaining—his mission ends. All actions have a cost of 1, and he cannot remain still.



In this example, straight arrows are regular moves and the curved arrow is a jump.

(a) [3 pts] Edgar is comparing DFS, BFS, and A* Search using the trivial heuristic $h(n) = 0$. Assume the branching factor is 4, and the nearest solution is 4 moves away. Select all true statements.

Hint: You expand a state when you call its successor function.

- ☐ A* Search will find the solution by expanding fewer states than BFS.
- ☐ A* Search will always expand $4^0 + 4^1 + 4^2 = 21$ states or fewer.
- ☐ BFS will always find a solution at an equal or shallower depth than DFS.
- ◯ None of the above.

(b) [2 pts] Edgar uses A* Search with an admissible heuristic. What is the worst-case time complexity in terms of the branching factor b and the depth of the shallowest solution d?

- ◯ $O(n)$, where n is the number of states
- ◯ $O(b^d)$
- ◯ $O(d^b)$
- ◯ $O(b)$

(c) [3 pts] Edgar now uses greedy graph search with heuristic $h(n) = E(n)/100$, where $E(n)$ is his Euclidean distance to the statue at state n. Select all true statements.

- ☐ Greedy graph search is complete, because it only evaluates legal moves.
- ☐ Greedy graph search will always find a solution with $h(n)$, because $h(n)$ is an admissible heuristic.
- ☐ Greedy graph search will always find a shorter path than A* search with $h(n)$ as its heuristic.
- ◯ None of the above

Edgar wants to design an admissible heuristic function $h(n)$. He has the following functions:

1. $M(n)$ = the Manhattan distance from Edgar to the goal for state n.
2. $E(n)$ = the Euclidean distance from Edgar to the goal for state n.
3. $O(n)$ = the smallest number of obstacles that lie along any path of distance $M(n)$ between Edgar and the goal for state n.
4. $J(n)$ = the number of jumps remaining.

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

- ☐ $M(n)$
- ☐ $E(n)$
- ☐ $O(n)$
- ☐ $J(n)$
- ◯ None of the above

(e) [4 pts] Which of the following are **admissible** heuristics?

- ☐ $\min(O(n), J(n))$
- ☐ $E(n)/3$
- ☐ $O(n) + 1$
- ☐ $M(n) + O(n)$
- ◯ None of the above

(f) [1 pt] Edgar decides to use the simulated annealing algorithm with an initial temperature of 100 and a cooling schedule of $T_i = \frac{T_0}{1+i}$, where i is the number of iterations. Approximately what is the algorithm's temperature after 50 iterations?

- ◯ 0
- ◯ 2
- ◯ 10
- ◯ 50

Q3. [19 pts] Pacman.io

Xavier and Matei are playing a new online game called Pacman.io.

They each move their own Pacman around a 20×20 grid, taking turns, with actions {UP, DOWN, LEFT, RIGHT, STOP}.

To start, 100 food dots are placed around the grid in **random locations**. After a food is eaten by either Pacman, another food is **immediately** placed in a random empty grid location.

Both Pacmen start at power level 1 and gain 1 power level for each food eaten.

A player wins if their Pacman reaches power level 100 or "eats" the other player's Pacman.

"Eating" occurs as follows: When both Pacmen occupy the same grid location, the Pacman with higher power level eats the other one. If they have the same power level, nothing occurs, and both Pacmen occupy the same position.

Xavier represents this game with a game tree.

(a) [4 pts] Which of the following are valid state representations for this problem?

- ☐ 100 booleans for food, and each Pacman's (x, y) coordinates and power level.
- ☐ A list of visited food locations, and each Pacman's (x, y) coordinates and power level.
- ☐ A list of current food locations, and each Pacman's (x, y) coordinates and power level.
- ☐ Each Pacman's (x, y) coordinates and power level only.
- ◯ None of the above.

(b) [4 pts] Xavier chooses to represent the state with 400 booleans for the food locations (one for each grid location), each Pacman's (x, y) coordinates and their power levels. What is the size of the state space using this state representation?

Represent the answer in the form:

$$A^B \times B^C$$

Fill in the boxes for A, B, and C using only positive integers.

A: ☐ B: ☐ C: ☐

(c) [4 pts] Xavier considers running depth-limited minimax. In general, which of the following are valid reasons for running depth-limited minimax **instead of** a full minimax search?

- ☐ Searching the full tree is often computationally infeasible due to a large branching factor.
- ☐ Using an evaluation function on the leaf nodes in depth-limited minimax is more accurate than computing the utilities for the entire minimax tree.
- ☐ A suboptimal agent is better modeled by a depth-limited minimax tree.
- ☐ Depth-limited minimax does alpha-beta pruning for us.
- ◯ None of the above.

(d) [5 pts] Xavier needs a function to compute the utility at each leaf node in the minimax tree.

- Case 1: Xavier's Pacman has a higher power level than Matei's. For two states with the same power levels, the function should prefer states where the Pacmen are closer.
- Case 2: Matei's Pacman has a higher power level than Xavier's. For two states with the same power levels, the function should prefer states where the Pacmen are farther apart.

Write an evaluation function that behaves as above, using these terms:

- Manhattan Distance between Xavier and Matei's Pacmen, $MH(p_x, p_m)$.
- Power level of Xavier's Pacman, ℓ_x.
- Power level of Matei's Pacman, ℓ_m.

Your function cannot be piecewise—write one expression for all states.

You may assume that $\ell_x \neq \ell_m$ and $MH(p_x, p_m) \neq 0$.

Note: An example of "two states with the same power levels" is a pair of states where $\ell_x = 10$ and $\ell_m = 5$ in both states.

(e) [2 pts] Suppose there are now a known, arbitrary number of Pacmen on a grid of known, arbitrary size. After a Pacman is eaten, the eater will gain power level equal to the power level of the eaten Pacman. The win conditions are to reach power level P_{max} first or eat all other Pacmen.

Which of the following setups represents this new scenario the best for Xavier's Pacman?

◯ Run minimax with all other Pacmen as minimizer nodes. The evaluation function is the power level of Xavier's Pacman.

◯ Run expectimax with all other Pacmen as a expectation nodes. The evaluation function is power level of Xavier's Pacman.

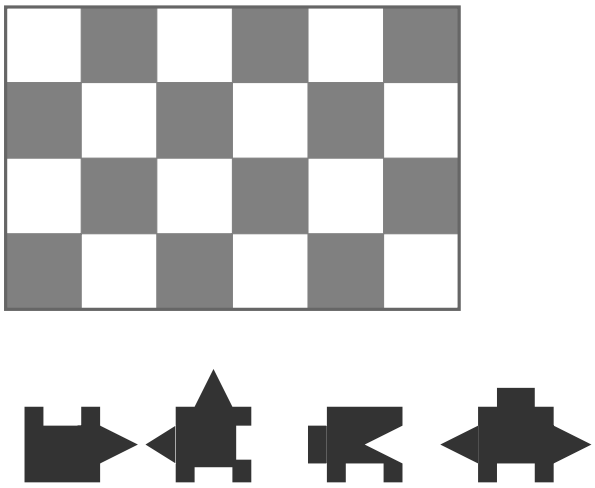
◯ Construct a game tree with each Pacman maximizing their own utility. The evaluation function for each Pacman is their own power level.

Q4. [14 pts] Pranav's Phunky Puzzle

Pacman is a big fan of jigsaw puzzles and designs a hard puzzle:

- There is an $M \times N$ grid, where each element in the grid fits exactly one puzzle piece.
- There are 5 different pieces. Each piece may be used more than once. Each piece may be rotated by 0°, 90°, 180°, or 270°.
- If a piece has a flat edge, that edge must be touching the edge of the grid.
- The goal is completely fill the grid by placing pieces, forming a $M \times N$ rectangle.

An example grid and pieces are shown below:



(a) [2 pts] Suppose we define the CSP variables to be each grid position. What is the size of the domain for each variable before filtering?

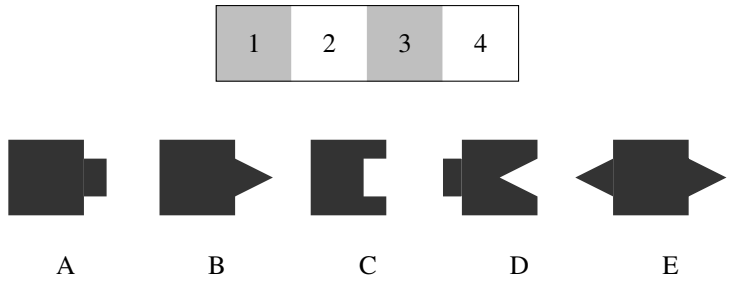
- ◯ 5
- ◯ 10
- ◯ 15
- ◯ 20

(b) [2 pts] If we have d pieces, rather than 5, what is the time complexity of running backtracking search on this puzzle without using any optimizations?

- ◯ $O(dMN)$
- ◯ $O\left((4d)^{MN}\right)$
- ◯ $O\left(d^{MN}\right)$
- ◯ $O\left((MN)^{4d}\right)$

For the rest of the question, we modify the puzzle so that pieces can only be rotated 0° or 180°.

Consider the following puzzle, with a 4×1 grid, and the following 5 pieces:



Reminders: Pieces may be reused. If a piece has a flat edge, that edge must be touching the edge of the grid. For example, piece A cannot go in position 2 or 3.

Pacman solves the puzzle using backtracking search. First, he assigns puzzle piece C to position 1.

Note: If there is **any orientation** of a piece that would fit in a position, include that piece in the domain of that position.

(c) [4 pts] After applying unary constraints, apply arc consistency to $(2 \rightarrow 1)$ and then $(4 \rightarrow 2)$.
What pieces are in the domains of positions 2 and 4?

Domain of position 2:

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ E
- ◯ None of the above

Domain of position 4:

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ E
- ◯ None of the above

(d) [4 pts] Continuing from the previous subpart, Pacman applies arc consistency on $(2 \rightarrow 3)$ and then $(3 \rightarrow 2)$.
What pieces are in the domains of positions 2 and 3?

Domain of position 2:

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ E
- ◯ None of the above

Domain of position 3:

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ E
- ◯ None of the above

(e) [2 pts] Consider the domains you computed in the previous subparts.
According to the MRV heuristic, which variable should we assign next? Break ties by picking the lower number.

- ◯ 2
- ◯ 3
- ◯ 4

Q5. [16 pts] Spelunking

Pacman is mining gemstones in deep, dark caves with an autonomous robot!

Each cave is a separate MDP. The caves have the same state spaces, but not necessarily the same transition and reward functions.

Every time the robot takes an action a, it wirelessly sends a sample to Pacman: (s, a, s', r), where s is the state it was in, a is the action it took, s' is the next state, and r is the reward it received.

The reward is $+10$ if the robot discovers a gemstone, and 0 otherwise.

The state transition function is deterministic, and the state and action space is finite. You know nothing else about the state and action spaces, the state transition function, or the reward function.

Pacman wants to use Q-learning on the samples to learn a policy π.

Each subpart is separate and independent.

(a) [4 pts] Pacman sends his robot into Cave A, using a random policy. Select all settings that can individually cause π, the learned policy, to be **suboptimal**.

- ☐ The robot runs until it visits all states once.
- ☐ The robot runs until it discovers all the gemstones in the cave.
- ☐ Pacman sets α to 1, and reduces it slowly to a very small number.
- ☐ Pacman adds an exploration bonus to the reward, which approaches zero when a state is visited a large number of times.
- ◯ None of the above.

(b) [2 pts] Pacman collects samples from Cave B and runs Q-learning to learn π, the optimal policy in Cave B.

Then, Pacman follows π in Cave C. What do we expect to happen?

- ◯ π is the optimal policy.
- ◯ In expectation, π performs better than a random policy, but it is not optimal.
- ◯ In expectation, π performs about as well as a random policy.

(c) [4 pts] Pacman rates caves by their *danger*, D, from 1 (safest) to 100 (most dangerous), and sets the discount factor as $\gamma = \frac{D}{100}$ before running Q-learning in each cave. What effect does this have on a robot following the learned policy, π?

Select all that apply.

- ☐ If $D = 100$, the robot will try to collect all gemstones.
- ☐ If $D = 1$, the robot will move randomly.
- ☐ In more dangerous caves, future rewards will be more important for the robot.
- ☐ The horizon scales inversely with D; that is, the more dangerous the cave, the shorter the horizon.
- ◯ None of the above.

For the next three subparts, Josh proposes the new algorithm described below:

1. Josh collects an extremely large amount of samples using a random policy, π_{rand}.
2. Josh computes Q_{rand} by running Q-learning with $\alpha = 0.5$, $\gamma = 0.99$, and the samples from step 1.
3. Josh extracts a policy from Q_{rand} to get π_{smart}.
4. Josh collects an extremely large amount of samples using π_{smart}.
5. Josh initializes Q_{smart} to Q_{rand}.
   He then runs Q-learning to update Q_{smart} using $\alpha = 0.1$, $\gamma = 0.99$, and the samples from step 4.
6. Josh extracts a policy from Q_{smart} to get π_{final}.

(d) [2 pts] Q_{rand} is not optimal. Why?

- ◯ Q-learning with randomly generated samples does not give an optimal Q-function.
- ◯ The discount, γ, is too high.
- ◯ The learning rate, α, is too high.

(e) [2 pts] One way to view this algorithm is that step 1 focuses on **(A)**, and step 4 focuses on **(B)**.

- ◯ **(A)**: exploitation, **(B)**: exploration
- ◯ **(A)**: exploration, **(B)**: exploitation
- ◯ **(A)**: finding good states, **(B)**: finding bad states
- ◯ **(A)**: finding good actions, **(B)**: finding bad actions

(f) [2 pts] Is Q_{smart} more accurate (i.e., closer to Q^*), compared to Q_{rand}?

- ◯ For all states, Q_{smart} is strictly more accurate than Q_{rand}.
- ◯ For all states, Q_{smart} is at least as accurate as Q_{rand}.
- ◯ Q_{smart} is more accurate in some states, and less accurate in other states.
- ◯ Q_{smart} is identical to Q_{rand}.

Q6. [17 pts] Potpourri: Arbitrary Abstraction

Arbitrary means you know the definitions we discussed in class and the details mentioned in the problem, and nothing else.

(a) [2 pts] Consider an arbitrary two-agent, zero-sum game. We use a complete minimax game tree to decide the action we take. Is it possible to achieve more utility by taking another action?

- ◯ Always possible
- ◯ Sometimes possible
- ◯ Never possible

For the next four subparts, consider an arbitrary, deterministic, *sparse* MDP, with a single terminal state.

In a sparse MDP, the reward function $R(s, a, s')$ returns zero for all transitions, (s, a, s'), except for a single transition that ends at the terminal state.

For every two states, there exists a sequence of actions that connect them, i.e. the state space graph is connected. You know nothing else about the transition function, $T(s, a, s')$, other than that it is deterministic.

Suppose there are N non-terminal states, and for every state, you can take M actions.

(b) [1 pt] If we run value iteration, when is the **earliest** possible iteration k that value iteration converges—the smallest k where $V_k = V^*$?

◯ 0	◯ 2	◯ M	◯ $N \times M$
◯ 1	◯ N	◯ $N + M$	◯ N^M

(c) [1 pt] When is the **latest** possible iteration k that value iteration converges—the largest k where $V_k = V^*$, but $V_{k-1} \neq V^*$?

◯ 0	◯ 2	◯ M	◯ $N \times M$
◯ 1	◯ N	◯ $N + M$	◯ N^M

(d) [1 pt] If we run policy iteration, when is the **earliest** possible iteration i that policy iteration converges—the smallest i where $\pi_i = \pi^*$?

Assume that we initialize with a random policy.

◯ 0	◯ 2	◯ M	◯ $N \times M$
◯ 1	◯ N	◯ $N + M$	◯ N^M

(e) [1 pt] When is the **latest** possible iteration i that policy iteration converges—the latest i where $\pi_i = \pi^*$, but $\pi_{i-1} \neq \pi^*$?

Assume that we initialize with a random policy.

◯ 0	◯ 2	◯ M	◯ $N \times M$
◯ 1	◯ N	◯ $N + M$	◯ N^M

(f) [2 pts] Consider three random variables, A, B, C. We know that A and B are conditionally independent given C.

Select two terms that are equivalent.

- ☐ $P(A)$
- ☐ $P(A|C)$
- ☐ $P(A|B, C)$
- ☐ $P(A|B)$
- ◯ All the terms are different.

For the next three subparts, consider two arbitrary MDPs, M_X and M_Y. They have the same state space and action space, but each MDP has its own unknown transition function, T_X and T_Y respectively. We run model-based learning on both MDPs to get $\hat{T}_X, \hat{T}_Y$.

M_Z is a third arbitrary MDP that has the same state and action spaces as M_X and M_Y, but has a weighted average transition function, $T_Z = k_X T_X + k_Y T_Y$, where $k_X + k_Y = 1$. You do not know k_X and k_Y.

Let's explore how we might "reuse" our transition functions $\hat{T}_X, \hat{T}_Y$ to estimate $\hat{T}_Z$.

(g) [2 pts] Blinky gives us a sample, (s, a, s'), taken from M_Z.

Suppose that $\hat{T}_X(s, a, s') >> \hat{T}_Y(s, a, s')$, and $\hat{k}_X = \hat{k}_Y = 0.5$.

$A >> B$ means that A is much greater than B.

How should we change the weights, $\hat{k}_X$ and $\hat{k}_Y$, to correspond with Blinky's information?

- ☐ $\hat{k}_X$ should increase.
- ☐ $\hat{k}_X$ should decrease.
- ☐ $\hat{k}_Y$ should increase.
- ☐ $\hat{k}_Y$ should decrease.
- ◯ None of the above.

(h) [4 pts] Pacman suggests that we should use the following learning update for $\hat{k}_X$:

$$\hat{k}_X \leftarrow \hat{k}_X + \alpha\Big(\hat{T}_X(s, a, s') - \hat{T}_Z(s, a, s')\Big)$$

Select all true statements about this update.

- ☐ $\hat{k}_X$ increases when a sample is more likely from M_X than M_Z.
- ☐ $\hat{k}_X$ increases when a sample is more likely from M_Z than M_X.
- ☐ $\hat{k}_X$ does not change when $\hat{k}_X$ is correct, i.e. matches Blinky's true value.
- ☐ $\hat{k}_X$ can only increase.
- ◯ None of the above.

(i) [3 pts] What else needs to be done in addition to Pacman's update in the previous subpart to correctly update $\hat{k}_X$?

Hint: There is a one-word answer, but you can answer with up to 8 words. What kind of function is $\hat{T}_Z$?