Homework 5 [100 pts]: RL and Logic

General Information

Deadline

Due: 11:59 PM, **Wednesday**, December 7th, 2022
Late homework will be penalized by 10% per day (where each day starts at 12:00 AM on the due day). There **are only 2 late days** for this homework because we want to get the solutions back as soon as possible for the finals.

Gradescope

Please familiarize yourself with Gradescope if you have never used it before. In particular, when you submit an assignment, Gradescope asks you to select pages of your solution that correspond to each problem. This is to make it easy for graders to find where in your work they should grade.
For the written questions, put your answers in a PDF and submit to the corresponding assignment on Gradescope. You can submit this PDF as many times as you would like before the deadline. You can use the provided Word template and fill in your answer. You may also submit your written response homework in a separate document.
Please make sure to include your responses to the 'Understanding the Code' section of the coding question. For the coding questions, submit your code files (activeRL.py) to the Gradescope autograder. A template starter code is provided for you. Do not modify any of the imports in these files. You will be graded on both visible and hidden test cases. Thus, you should make sure your solution is always correct (possibly by testing it on other cases that you write) and not just whether it succeeds on the visible test cases. This is an important skill for a practicing computer scientist! You can submit to the autograder as many times as you would like before the deadline. While you will not be graded on style, we encourage you to uphold best-practice style guidelines. For more information about the coding refer to the coding portion of the homework.

Setup

You are recommended to use a Python virtual environment for the coding portion. You may follow this tutorial from EECS 485 to create and set up a virtual environment.
We have provided you with a requirements.txt. After creating the environment, as mentioned in the tutorial, activate the environment by executing source env/bin/activate in the terminal. Install the necessary library by using pip install -r requirements.txt.
Feel free to visit us in office hours if you need help on the above or help on setting up tools for debugging.

Written Section [65 pts]

Exercise 1 [10 pts]: Short Questions

Provide a brief explanation of the distinction between the two concepts.

1.1) Model-based vs model-free reinforcement learning

The difference between the two is that model-free uses no transition model while Model-based does use an (either learned or known) transition model to create a policy. Model-based is therefore more risky during the training process but safer once the policy has been created. Model-free is simpler than model-based as it is practicing trial-and-error, but it is therefore much riskier during the testing phase (which is also its training phase as it creates its policy from its real world experiences)

1.2) Passive vs active reinforcement learning

Passive rl has a fixed policy (pi) while active rl is seeking a good policy. Passive's goal is to evaluate the policy pi. Active's method of achieving a good policy is through using current q-value estimates, which creates a feedback loop between the policy a q-val estimations. There is no such loop in passive.

1.3) Exploration vs exploitation (in the context of reinforcement learning)

The RL algorithms have to balance exploration and exploitation as exploration is trying out actions that might not be the most rewarding in order to go down a different path and find a potentially more efficient solution, and exploitation which is taking the most rewarding actions from what the agent has seen so far.

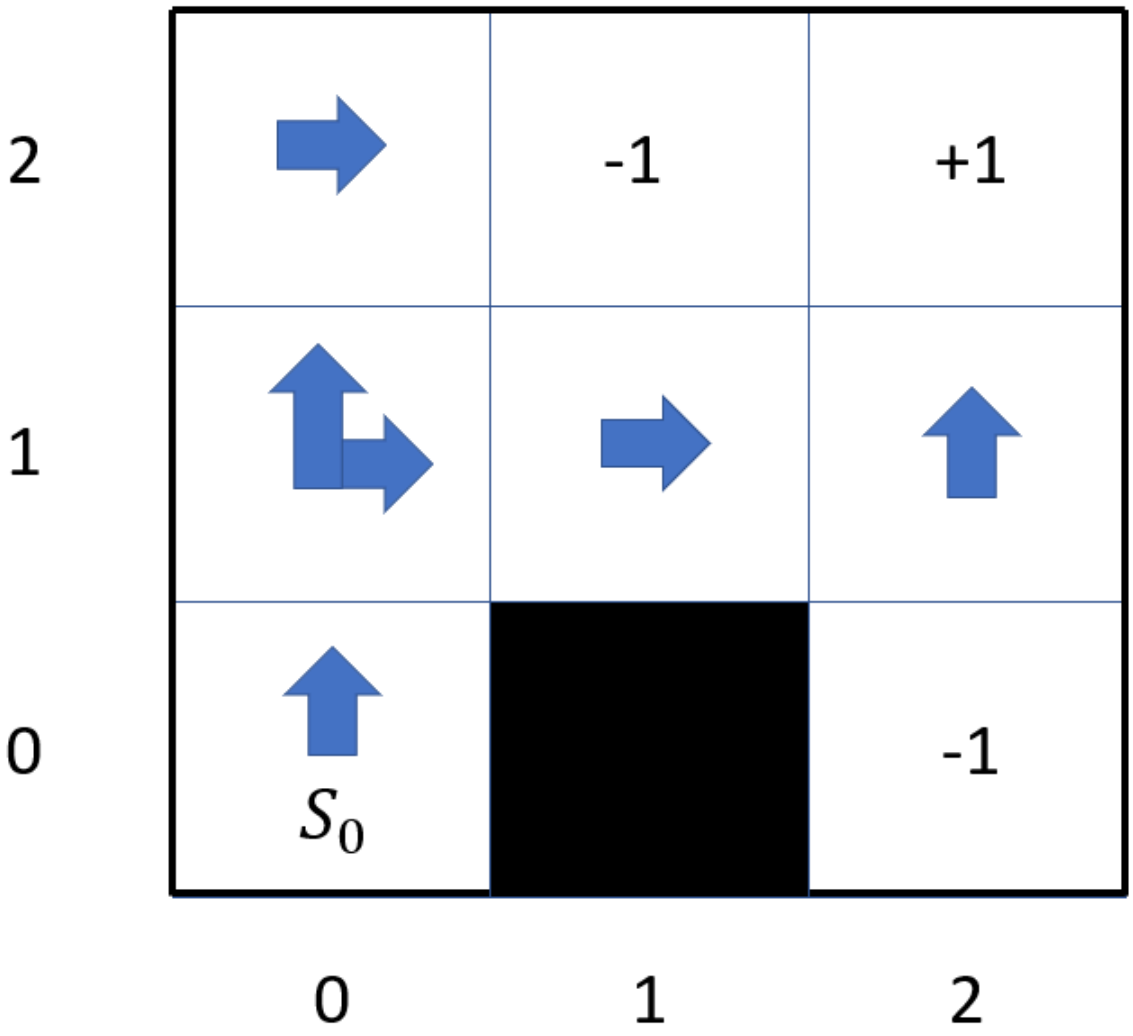
1.4) On-policy vs off-policy reinforcement learning

Off-policy updates assuming that the future policy chooses the best estimated action while on-policy assumes that the future policy and the current policy are the same.

1.5) Monte-Carlo RL vs temporal difference (TD) reinforcement learning

TD rl estimates the next part of the rollout's future reward, while Monte-Carlo waits until the end of the episode to update q-values, as TD employs bootstrapping to update q-values during the episode while Monte-Carlo takes the average.

Exercise 2 [10 pts]: Monte-Carlo RL

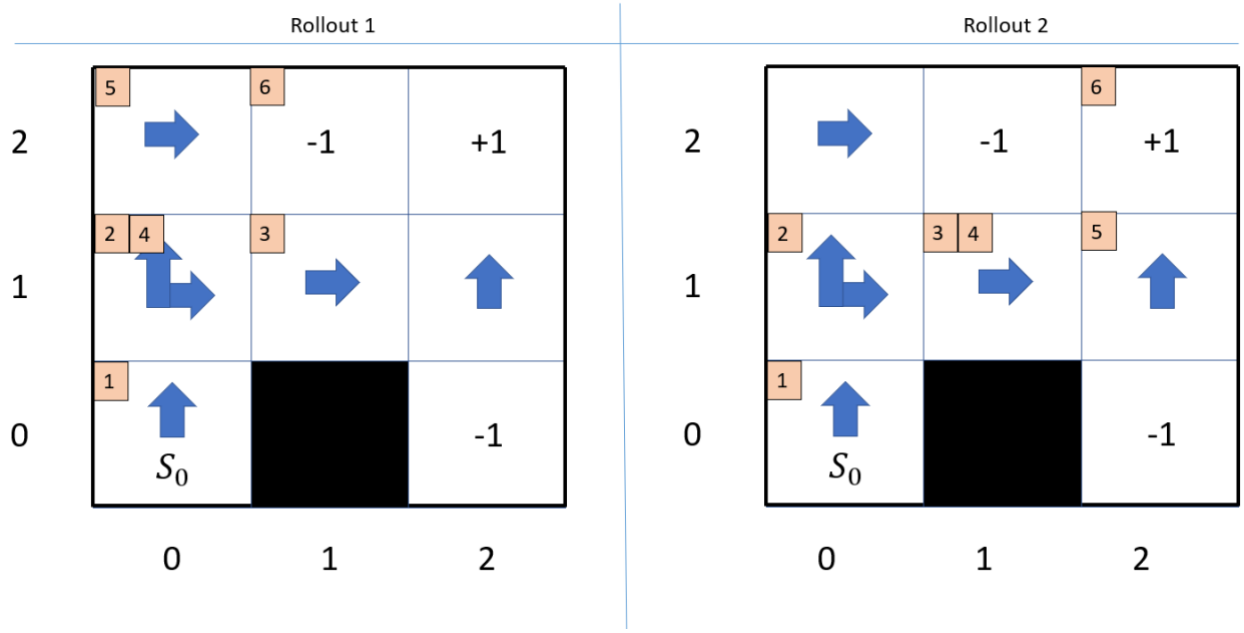


Consider the grid above. A position in the grid is given as (column, row). You start at state (0, 0) as denoted by the label s_0. The bold black lines represent walls that cannot be crossed (i.e. around the exterior of the grid). Similarly, state (1, 0) is a walled state that is unreachable. Assume that if the agent actually moves in the direction of a wall it will hit the wall and stay in its current position. The states with numerical values (states, (1,2), (2,2) and (2,0), respectively) are terminal states with their numerical values as their respective rewards. Assume that every non-terminal state has a reward of -0.1. At any state you can attempt to go north, south, east, or west. For any action, there is a 0.7 probability of moving in the intended direction, a 0.1 probability of not moving at all, and a 0.2 probability of moving in the opposite direction.

We are performing Monte-Carlo RL with a fixed policy on the above grid. Assume that the fixed policy is represented by the blue arrows in the grid. For instance, the policy at

state (0,0) is the action "north". Assume that we are using discount factor, $\gamma = 1$.

The grids below (on the next page) represent the first two rollouts from the agent. For the given rollouts, the boxed numbers at the top left corners of certain states represent the order in which those states were visited. For instance, in the first rollout, we visit the states (0,0), (0,1), (1,1), (0,1), (0,2), and (1,2) in that order.



After the two rollouts, find the final q-values. You only need to include relevant q-values (you don't need to include any q-values that were not updated by the two given rollouts). Show all work.

2) $G_t = r + \gamma G_t$

$N(s, a) = N(s,a) + 1$

$q(s,a) = q(s,a) + \frac{1}{N(s,a)} (G_t - q(s,a))$

Rollout 1

$(0,2), E$

$G = 0 + 1(-1) = -1$

$N((0,2), E) = 1$

$q((0,2), E) = 0 + \frac{1}{1}(-1 - 0) = -1$

$(0,1), N$

$G = -0.1 + 1(-1) = -1.1$

$N((0,1), N) = 1$

$q((0,1), N) = 0 + \frac{1}{1}(-1.1 - 0) = -1.1$

$(1,1), E$

$G = -0.1 + 1(-1.1) = -1.2$

$N((1,1), E) = 1$

$q((1,1), E) = 0 + \frac{1}{1}(-1.2 - 0) = -1.2$

$(0,1), E$

$G = -0.1 + 1(-1.2) = -1.3$

$N((0,1), E) = 1$

$q((0,1), E) = 0 + \frac{1}{1}(-1.5 - 0) = -1.3$

$(0,0), N$

$G_T = -0.1 + 1(-1.4)$

$N((0,0), N)$

$q((0,0), N) = 0 + \frac{1}{1}(-1.4 - 0) = -1.4$

Rollout 2

$(2,1), N$

$G_T = 0 + 1(1) = 1$

$N((2,1), N) = 1$

$q((2,1), N) = 0 + \frac{1}{1}(1 - 0) = 1$

$(1,1), E$

$G_T = -0.1 + 1(1) = 0.9$

$N((1,1), E) = 2$

$q((1,1), E) = -1.2 + \frac{1}{2}(0.9 - (-1.2)) = -0.15$

$(1,1), E$

$G_T = -0.1 + 1(0.9) = 0.8$

$N((1,1),E) = 3$

$q((1,1),E) = -0.15 + \frac{1}{3}(0.8 - (-0.15)) = 0.167$

$(0,1), E$

$G = -0.1 + 1(6.8) = 0.7$

$N((0,1),E) = 2$

$q((0,1),E) = -1.3 + \frac{1}{2}(0.7 - (-1.3)) = -0.3$

$(0,0), N$

$G = -0.1 + 1(0.7) = 0.6$

$N((0,0),N) = 2$

$q((0,0),N) = -1.4 + \frac{1}{2}(0.6 - (-1.4)) = -0.4$

Exercise 3 [10 pts]: Entailment

State whether each of the following statements is True or False and provide a justification. If the first part = T, then the second part must be T, if the first part is false, second part doesn't matter

3.1) $False \models True$

True. There isn't a single model where "false" on the left side of the entails is true, and an empty set is always a subset of an empty set, so False entails True.

3.2) $True \models False$

False. All models of "True" evaluate to True, but none of the models make "False" true, so every model is a counterexample to True entails False.

3.3) $A \wedge B \models A \vee B$

True. The only way "A and B" is true is when both A and B are true, which also makes "A or B" true. As True always implies True for this statement, "A and B" entails "A or B."

3.4) $(A \vee B) \wedge C \models A \wedge B$

False. LHS true when A=F, B=T, C=T, and A=T B=F C=T (and A=T B=T C=T) and in those first two scenarios, "A and B" is false. Therefore, in the above statement, the LHS does not imply the RHS in every model, so the LHS doesn't entail the RHS.

3.5) $A \wedge B \models A \Rightarrow B$

True. LHS is only true when both A and B is true, and true implies true is true, so therefore, A and B entails A implies B.

Exercise 4 [6 pts]: Satisfiability

State whether each of the following sentences is valid, satisfiable but not valid, or unsatisfiable. Provide justification using truth tables, proofs, or equivalence rules.

4.1) $(A \wedge \neg A) \vee (B \vee \neg B)$ valid and satisfiable

A	B	$A \Rightarrow B$	$\neg(A \wedge \neg B)$	$(A \Rightarrow B) \Rightarrow \neg(A \wedge \neg B)$
F	F	T	T	T
F	T	T	T	T
T	F	F	F	T

T	T	T	T	T

4.2) $(A \wedge \neg A) \vee (B \vee \neg B)$ valid and satisfiable, either B or not B will always be true, making the whole proposition true

4.3) $(A \vee \neg B) \Leftrightarrow (A \Rightarrow B)$ satisfiable but invalid

A	B	$(A \vee \neg B)$	A ⇒ B	$(A \vee \neg B) \Leftrightarrow (A \Rightarrow B)$
F	F	T	T	T
F	T	F	T	F
T	F	T	F	F
T	T	T	T	T

Exercise 5 [8 pts]: Conjunctive Normal Form

Convert the following logic statements to conjunctive normal form (CNF). Show your work.

5.1) $(A \vee B) \Rightarrow (C \wedge D)$

5.1) $(A \vee B) \rightarrow (C \wedge D)$

$\neg(A \vee B) \vee (C \wedge D)$

$(\neg A \wedge \neg B) \vee (C \wedge D)$

$((C \vee \neg A \wedge \neg B) \vee C) \wedge ((C \vee \neg A \wedge \neg B) \vee D)$

$(((C \vee \neg A) \wedge (C \vee \neg B)) \wedge ((D \vee \neg A) \wedge (D \vee \neg B))$

CNF: $(C \vee \neg A) \wedge (C \vee \neg B) \wedge (D \vee \neg A) \wedge (D \vee \neg B)$

5.2) $(A \vee B) \Leftrightarrow (A \vee \neg C)$

5.2) $(A \vee B) \iff (A \vee \neg C)$

$((A \vee B) \rightarrow (A \vee \neg C)) \wedge ((A \vee \neg C) \rightarrow (A \vee B))$

$(\neg(A \vee B) \vee (A \vee \neg C)) \wedge (\neg(A \vee \neg C) \vee (A \vee B))$

$((\neg A \wedge \neg B) \vee (A \vee \neg C)) \wedge ((\neg A \wedge \neg\neg C) \vee (A \vee B))$

$((\neg A \wedge \neg B) \vee (A \vee \neg C)) \wedge ((\neg A \wedge C) \vee (A \vee B))$

$[((A \vee \neg C) \vee \neg A) \wedge ((A \vee \neg C) \vee \neg B)] \wedge [((A \vee B) \vee \neg A) \wedge ((A \vee B) \vee C)]$

$(A \vee \neg C \vee \neg A) \wedge (A \vee \neg C \vee \neg B) \wedge (A \vee B \vee \neg A) \wedge (A \vee B \vee C)$

$T \wedge (A \vee \neg C \vee \neg B) \wedge T \wedge (A \vee B \vee C)$

$CNF = (A \vee \neg C \vee \neg B) \wedge (A \vee B \vee C)$

5.3) $(A \Rightarrow B) \wedge \neg(A \wedge \neg C)$

5.3) $(A \rightarrow B) \wedge \neg(A \wedge \neg C)$

$(\neg A \vee B) \wedge (\neg A \vee \neg\neg C)$

$CNF = (\neg A \vee B) \wedge (\neg A \vee C)$

5.4) $\neg\big((A \Rightarrow B) \vee C\big)$

5.4) $\neg((A \rightarrow B) \vee C)$

$\neg((\neg A \vee B) \vee C)$

$\neg(\neg A \vee B \vee C)$

$\neg\neg A \wedge \neg B \wedge \neg C$

$CNF = A \wedge \neg B \wedge \neg C$

Exercise 6 [3 pts]: Propositional Logic

Pick the multiple-choice answer (single answer) that correctly translates each of the following sentences into propositional logic, using the following conventions:

- W_x – true if student x woke up early today.
- G_x – true if student x played video games last night.
- C_x – true if student x has class this morning.

6.1) Either student 1, 2, or 3 woke up early today.

a. $W_1 \wedge W_2 \wedge W_3$
b. $W_1 \wedge W_2 \vee W_3$
c. $W_1 \vee W_2 \vee W_3$
d. $W_1 \vee W_1 \vee W_1$

6.2) If student 1 woke up early, then either he did not play video games last night or he has class this morning.

a. $(\neg G_1 \vee C_1) \Rightarrow W_1$
b. $W_1 \Rightarrow \neg(G_1 \vee C_1)$
c. $W_1 \vee \neg G_1 \vee C_1$
d. $W_1 \Rightarrow \neg G_1 \vee C_1$

6.3) Student 3 didn't play video games last night if and only if students 1 and 2 both didn't play video games last night or if student 3 has class this morning.

a. $\neg G_3 \Leftrightarrow \neg(G_1 \wedge G_2) \vee C_3$
b. $\neg G_3 \Leftrightarrow \neg(G_1 \wedge G_2 \vee C_3)$
c. $\neg G_3 \Leftrightarrow \neg G_1 \wedge \neg G_2 \vee C_3$
d. $\neg G_3 \Leftrightarrow \neg G_1 \wedge \neg G_2 \wedge C_3$

Exercise 7 [10 pts]: Resolution Refutation

Lets assume we have the following propositional logic conventions (same as exercise 6):

- W_x – true if student x woke up early today.
- G_x – true if student x played video games last night.
- C_x– true if student x has class this morning.

Consider the following knowledge base consisting of sentences S_1 through S_5 (already given in CNF form):

S_1: W_1
S_2: G_3
S_3: $\neg G_2 \vee C_2$
S_4: $\neg W_1 \vee \neg G_1$
S_5: $G_1 \vee G_2 \vee \neg G_3$

7.1) Using resolution refutation, prove that student 2 has class this morning (C_2). Start with the negation of the hypothesis. Then list each new sentence in the order you derive them. For each derived sentence, make note of which sentences were resolved to derive it.

assume $\neg G_2$, which is now S_6

S_7: $\neg G_2$ from S_3 and S_6

S_8: $\neg G_3 \vee G_1$ from S_7 and S_5

S_9: G_1 from S_8 and S_2

S_{10}: $\neg W_1$ from S_9 and S_4

S_{11}: $\{\}$ contradiction from S_{10} and S_1

7.2) For each of the sentences in the knowledge base, state whether it is a horn clause, a definite clause, both, or neither. Based on this, could we have used forward chaining instead of resolution refutation to solve this problem? Why or why not?

S_1 = both

S_2 = both

S_3 = both

S_4 = horn

S_5 = neither

can't use forward chaining because not all sentences in the KB are horn clauses

Exercise 8 [8 pts]: Forward Chaining

Lets assume we have the following propositional logic conventions (same as exercise 6 and 7):

- W_x – true if student x woke up early today.
- G_x – true if student x played video games last night.
- C_x– true if student x has class this morning.

Consider the following knowledge base consisting of sentences S_1 through S_6 **(different from exercise 8)**:

S_1: W_1
S_2: W_2
S_3: $W_1 \Rightarrow C_1$
S_4: $C_1 \wedge W_2 \Rightarrow C_2$
S_5: $C_1 \wedge C_2 \wedge W_3 \Rightarrow C_3$
S_6: $W_1 \wedge W_2 \Rightarrow W_3$

Using forward chaining, prove that student 3 has class this morning (C_3). You should list the initial counts for each of the clauses, and the initial agenda. Then as you go through the forward chaining algorithm, write down each interaction (push/pop) with the agenda and how each interaction affects the counts.

Goal: C_3

Start

KB	Sym_count
S_1: W_1	0
S_2: W_2	0
S_3: $W_1 \Rightarrow C_1$	1
S_4: $C_1 \wedge W_2 \Rightarrow C_2$	2
S_5: $C_1 \wedge C_2 \wedge W_3 \Rightarrow C_3$	3
S_6: $W_1 \wedge W_2 \Rightarrow W_3$	2
Queue: {W_1, W_2}	

Pop = W_1

KB	Sym_count
S_1: W_1	0
S_2: W_2	0
S_3: $W_1 \Rightarrow C_1$	0
S_4: $C_1 \wedge W_2 \Rightarrow C_2$	2
S_5: $C_1 \wedge C_2 \wedge W_3 \Rightarrow C_3$	3
S_6: $W_1 \wedge W_2 \Rightarrow W_3$	1

Queue: {W_2, C_1}	

Pop = W_2

KB	Sym_count
S_1: W_1	0
S_2: W_2	0
S_3: $W_1 \Rightarrow C_1$	0
S_4: $C_1 \wedge W_2 \Rightarrow C_2$	1
S_5: $C_1 \wedge C_2 \wedge W_3 \Rightarrow C_3$	3
S_6: $W_1 \wedge W_2 \Rightarrow W_3$	0
Queue: {C_1, W_3}	

Pop = C_1

KB	Sym_count
S_1: W_1	0
S_2: W_2	0
S_3: $W_1 \Rightarrow C_1$	0
S_4: $C_1 \wedge W_2 \Rightarrow C_2$	0
S_5: $C_1 \wedge C_2 \wedge W_3 \Rightarrow C_3$	2
S_6: $W_1 \wedge W_2 \Rightarrow W_3$	0
Queue: {W_3, C_2}	

Pop = W_3

KB	Sym_count
S_1: W_1	0
S_2: W_2	0
S_3: $W_1 \Rightarrow C_1$	0
S_4: $C_1 \wedge W_2 \Rightarrow C_2$	0
S_5: $C_1 \wedge C_2 \wedge W_3 \Rightarrow C_3$	1
S_6: $W_1 \wedge W_2 \Rightarrow W_3$	0
Queue: {C_2}	

Pop = C_2

KB	Sym_count
S_1: W_1	0
S_2: W_2	0
S_3: $W_1 \Rightarrow C_1$	0
S_4: $C_1 \wedge W_2 \Rightarrow C_2$	0
S_5: $C_1 \wedge C_2 \wedge W_3 \Rightarrow C_3$	0
S_6: $W_1 \wedge W_2 \Rightarrow W_3$	0
Queue: {C_3}	

Pop = C_3
returns true, proving C_3

Programming Section [30 pts]: Q-learning and SARSA

Introduction and Objective

In this assignment, we'll be implementing a basic active RL agent to perform Q-learning and SARSA. Our environments will attempt to replicate a simplified version of Pacman. We can define the Pacman environment as a grid of positions with a finite number of rows and columns. We will assume there are walls surrounding the outside of the grid.

In the original game of Pacman, the game consists of some edible dots that Pacman can eat for points, ghosts that can eat Pacman (causing a loss), and some walls that Pacman can't pass through. In the original version of Pacman, Pacman wins by successfully eating all the dots.

To simplify our version of Pacman, in our version, Pacman wins by successfully making it to an exit without hitting any ghosts (while receiving bonus points for eating dots along the way). Therefore, for our version, at each grid position, we can either have an empty space, a wall, an edible dot, a ghost, or an exit with some reward. Note that dots will only provide rewards a single time (after they are consumed, they will be treated as empty spaces). Additionally, we will also make the simplification that the ghosts do not move.

On the other hand, in our version of Pacman, we also assume a non-deterministic environment. We will assume for our transition probability model, that if Pacman attempts to move in a certain direction, we have a 90% chance of successfully moving in that direction, and a 10% chance of moving in the opposite direction.

Input Text File Overview

The input to the **activeRL.py** program will be a text file. The text file will specify the hyperparameters as well as the grid world to run the RL agent on.

The first row of the text file will have 3 values separated by spaces, pertaining to the RL agent hyperparameters. The 3 values will correspond to the chosen discount factor, learning rate, and number of episodes, in that order.

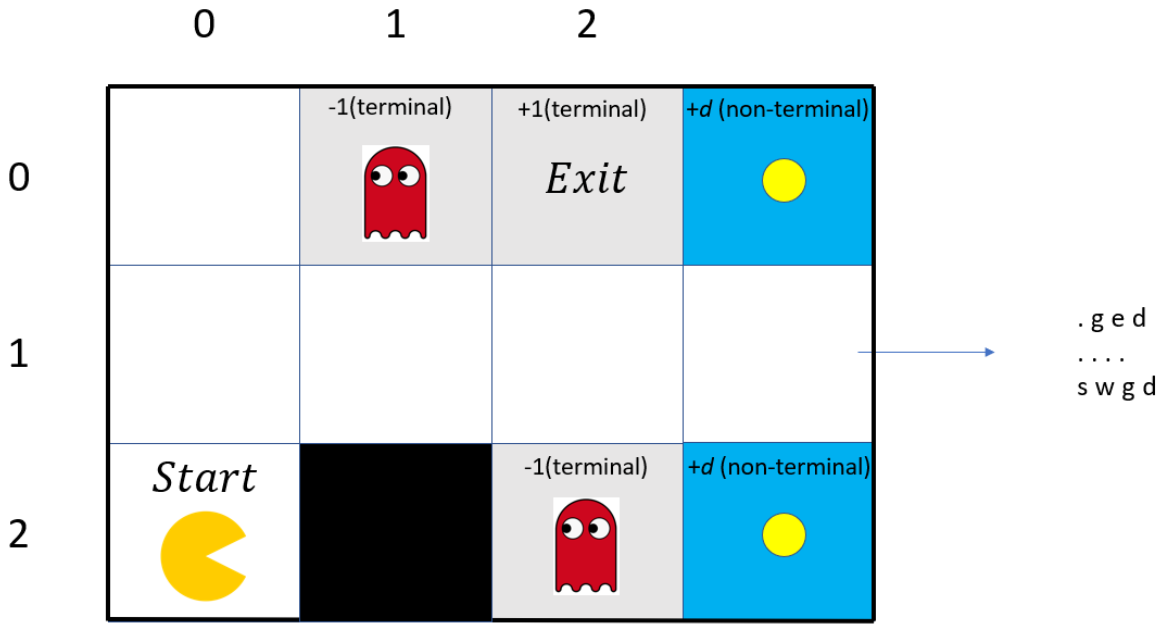
The second row of the text file will specify the size of the Pacman gameboard. This will be two integers corresponding to the number of rows and number of columns of the grid, respectively.

The third row of the text file specifies the empty state reward and the dot rewards, respectively. The first value is the reward corresponding to the 'empty' states in the grid. The second value is the reward corresponding to the 'dot' states in the grid.

We will assume that rewards for the ghosts are -1 and rewards for the exit are 1 (and both will be treated as terminal states).

Finally, the remaining rows will provide the structure of the grid, with "." referring to empty spaces, "s" referring to the start position (also assumed to be an empty space), "w" referring to walls, "d" referring to edible dots, "g" corresponding to ghosts, and "e" corresponding to exits.

For example, if we wanted to convert the following Pacman environment to a text file:



Putting this all together, if we have as input a text file as follows:

1 0.1 50

3 4

-0.04 0.5

. g e d

. . . .

s w g d

Then we would have a discount factor of 1, learning rate of 0.1, and 50 episodes, with the environment illustrated above (the 3 4 row tells the file parser that the environment is of size 3 by 4), and a reward of -0.04 for empty states, and a reward of *d* = 0.5 for the dot states.

Starter Code Overview

You only need to modify the activeRL.py file in the given starter code.
The activeRL.py file contains the following classes:

State

- Has the following member variables:
  - position – tuple corresponding to the state's (row, col) position such that (0,0) corresponds to the top-left as illustrated above. **NOTE: be aware that this is a different from the convention used in exercise 2 which took the form (col, row) and had the origin as the bottom left corner instead.**
  - type – string corresponding to ".", "s", "w", "d", "g", or "e" depending on the given state

based on the input txt file.
    - reward – float corresponding to the numerical reward associated with the state. (i.e. -1 for ghosts, +1 for exit, +*d* for dots – where *d* is the reward for dots based on the text file)
    - terminal – boolean corresponding to whether or not the state is a terminal state (i.e. true for ghost or exit states, false otherwise).
- Has the following function:
    - get_actions – always returns a list containing "north", "south", "east", "west" as these are the actions for any given state. Usage of this function is optional, and you can instead hardcode these values into your code if you prefer.

Grid

- Has the following member variables:
    - num_rows – int corresponding to the number of rows in the grid environment
    - num_cols – int corresponding to the number of columns in the grid environment
    - states – an 2d numpy array of size (num_rows, num_cols). states[i, j] refers to a State class object corresponding to the state information of the grid environment at grid position (i, j).
    - start – a tuple of (start_row, start_col) corresponding to the starting position in the grid environment.
    - empty_reward - corresponding to the reward for empty states (including dot states after they have been consumed) based on the input file
    - visited – an 2d numpy array of size (num_rows, num_cols). Visited[i, j] = 1 if there is a dot at that location that has already been consumed, visited[i, j] = 0 otherwise.
- Has the following functions:
    - visit(state) – marks the given state as visited
    - reset – resets the grid (in this case that only corresponds to resetting visited)
    - get_reward(state) – returns the reward for the given state. If the given state is a dot, it will be marked as consumed.
    - **NOTE: these functions are already called by the RLAgent select_action and execute_action functions, so you will not need to explicitly call these functions.**

RLAgent

- Has the following member variables (**YOU NEED TO FINISH INITIALIZING THEM in __init__):**
    - grid – a grid class object corresponding to the input Pacman grid
    - df – a float corresponding to the discount factor
    - lr – a float corresponding to the learning rate
    - eps – an int corresponding to the number of episodes
    - **current_state** – a state class object corresponding to the current state the agent is in
    - **q** – a dictionary mapping a tuple of the form ((row_pos, col_pos), action) or identically of the form (position, action) to its current q-value. This is also referred to as the q-table.
    - **n** – a dictionary mapping a tuple of the form ((row_pos, col_pos), action) or identically of the form (position, action) to its current count.
- Has the following functions:
    - execute_action(action) – attempt to perform the given action from the current_state. Returns a new state class object corresponding to the new state. **Reminder: our**

environment is not deterministic, so our new state is not necessarily a result of successfully performing the action.

- select_action(state) – selects an action to perform for the given state based on the current q-values.
- **q_learning** – **YOU NEED TO IMPLEMENT THIS FUNCTION.** It should perform the q_learning RL algorithm, as described in lecture, based on the hyperparameters stored in the RLAgent class instance, and return the q-values after it has finished.
- **SARSA** – **YOU NEED TO IMPLEMENT THIS FUNCTION.** It should perform the SARSA RL algorithm, as described in lecture, based on the hyperparameters stored in the RLAgent class instance, and return the q-values after it has finished.
- print_results – prints the current q and n dictionary values for the relevant states. You do need to use this function, but it may be helpful for debugging.

Running/Debugging the Code

The activeRL.py file runs a complete pass of the training process with your given algorithm and prints out the final q-values and n-counts after training. It will take in the input text filename or filepath as an argument. For instance, if you want to run your algorithm on the test1.txt file, you should run **python activeRL test1.txt** from your terminal, or alternatively specify the filename "test1.txt" as an argument in your IDE. Additionally, by default the main function in activeRL.py only runs your q_learning implementation. You should comment that line out and uncomment the line running SARSA if you wish to test your SARSA implementation instead.
If you are struggling to debug your code, also consider writing your own test files following the above specifications outlining how the test files are formatted. After running your code on a test file, it will print out the final q-values for the relevant states after running your algorithm. If you think something is going wrong midway through the algorithm, you may wish to consider using the provided "print_results" function to print out intermediate q-values.

Visualizing a Test Run

In the starter code, we also provide you with a visualize_run.py file. You can run this file to visualize a test-time run after training with the given algorithm. It will take in the input text filename or filepath as an argument. For instance, if you want to visualize your algorithm on the test1.txt file, you should run **python visualize_run.py test1.txt** from your terminal, or alternatively specify the filename "test1.txt" as an argument in your IDE. Additionally, by default the main function in visualize_run.py only runs your q_learning implementation. You should comment that line out and uncomment the line running SARSA if you wish to test your SARSA implementation instead. The file will print out the current position after each of the agent's actions and the corresponding reward. It will also visualize the current position on the grid by printing out the entire grid, and denoting Pacman's current location with a "p".

Understanding the Code [5 Points]

Consider the provided select_action function we gave you in the activeRL.py file.

```python
def select_action(self, state):
    actions = state.get_actions()
    max_q = float('-inf')
    best_action = None
    for action in actions:
        if(self.q[(state.position, action)] > max_q):
            max_q = self.q[(state.position, action)]
            best_action = action

    #in train time:
    #use epsilon-greedy action selection
    #chooses best action with probability 1-epsilon
    #otherwise, returns a random action (with probability epsilon)
    epsilon = 0.3
    rand = random.random()
    if(rand < epsilon):
        rand_a = random.randint(0, len(actions)-1)
        return actions[rand_a]
    else:
        return best_action
```

1) Notice that the select_action function doesn't always return return the action with the best q_value. Why might this be a good idea?
   Because sometimes you need to explore new avenues in the hopes of potentially finding more efficient routes to high reward states. Its important to balance exploitation with exploration.
2) If you now want to use your trained RL model at test-time, how might you want to change the way you select an action for a given state?
   To always follow a policy of enacting the action with the highest q-val at each state.