UNIVERSITY OF TORONTO
Faculty of Applied Science and Engineering

ROB311H1 S – Artificial Intelligence

Final Project

Instructor: Matthew Giamou

April 20-22, 2021

Time allowed: **24 Hours**

This exam contains 12 pages (including this cover page) and 3 questions. You do not need to submit this document: answer each question in your PDF submission to Quercus, or with code submitted on Autolab where indicated. Include your name and student number on the first page of your PDF submission. **Make sure you have read, signed, and submitted the final project instructions on Quercus before proceeding.** You have 24 hours, good luck!

Distribution of Marks

Question	Points	Score
Constraint Satisfaction	35	
Sales Chatbot POMDP	35	
Learning to Drive	30	
Total:	100	

5	0	1	8	9	0	2	7	6
0	4	0	0	2	7	0	0	0
8	0	7	1	0	3	0	0	0
0	9	6	0	0	0	0	5	1
0	0	8	0	0	5	0	0	0
0	5	0	9	1	6	4	8	0
0	0	2	7	3	8	0	1	4
1	0	0	2	0	0	0	6	7
0	0	0	0	5	0	0	2	0

Figure 1: An easy instance of Sudoku that appears as an Autolab test case and in your starter code. The cells that are filled with 0's need to be assigned numbers from 1-9 such that each row, column, and 3x3 sub-square (indicated with colour) contains each digit from 1-9 exactly once.

Question 1: Constraint Satisfaction (35 points)

Read the entire question before attempting it, as the subsections are connected. Only the code in `q1_constraint_satisfaction.py` for part (c) is graded by Autolab, but you must submit all code you use for this entire question via:

```
tar cvf handin.tar q1_constraint_satisfaction.py OTHER_CODE.py
```

where `OTHER_CODE.py` is replaced with zero or more Python files named whatever you like which contain any additional code that you used to answer this question. The link to the Autolab submission portal for the tar file is here.

You are being interviewed by Spigot, a leading gaming company, for a position involving AI programming in an upcoming game. The interviewer has given you a series of questions about Sudoku, a best-selling game on their HotVapour gaming platform.

(a) (5 points) **Boolean Satisfiability**
Formulate Sudoku as a boolean satisfiability (SAT) problem. For the example in Figure 1

(which is given as an array in the starter code), use your formulation to write the clauses that describe the constraints on the entries in the top left 3x3 square that arise from their membership in that square. In other words, you do not have to provide the constraints on the variables that arise due to their row and column membership, only write the clauses that describe 3x3 square membership constraints. Clearly define what each variable and constraint represents, and explain how you would extend these constraints to the full 9x9 puzzle.

(b) (5 points) **Binary Constraints**
Formulate Sudoku as a constraint satisfaction problem (CSP) involving only binary *constraints*. Note that this does not mean that your variables are boolean, only that constraints are restricted to pairs of variables. As in 1a, give the variables, their domains, and the binary constraints that describe the restrictions on the unassigned entries in the top left 3x3 square of the puzzle in Figure 1. Once again, you do not have to provide the constraints on the variables that arise due to their row and column membership: only provide the binary constraints that describe 3x3 square membership constraints. Clearly define what each variable and constraint represents, and explain how you would extend these constraints to the full 9x9 puzzle. How does this binary CSP compare with the formulation in 1a: which would you use when programming a solution to Sudoku, and why?

(c) (12 points) **Autolab: Inference-Based Sudoku Solver**

Implement a Sudoku solver that combines search with an inference method for CSPs. Implement your method in the `sudoku_solver` function template in `q1_constraint_satisfaction.py`. You are **not** permitted to use any libraries other than `numpy` which is already imported in `q1_constraint_satisfaction.py`: any additional imports will be stripped by Autolab's preprocessor. Autolab will grade you on the two (easy and hard) Sudoku puzzles given in the handout code, as well as a third hidden (moderate difficulty) Sudoku puzzle. Each puzzle is worth 4 marks, and you will be given a part mark proportionate to the number of cells that are not in conflict if you fill in each empty (zero) square with a number from 1-9. Do not modify any of the non-zero digits given as part of the problem's input. You are limited to **10 submissions** for this question. As usual, your **latest** submission will be graded (not your submission with the highest score). Be sure to test your solver on your own machine first to avoid running out of submissions and abusing the Autolab server.

(d) (5 points) **Algorithm Description**
Describe the algorithm you implemented in 1c. Be sure to motivate your design decisions. If you were unable to complete your algorithm, describe any remaining work or room for improvement. What inference method did you choose, and how does it fit into your search strategy? Comment on how Sudoku differs from the N-Queens problem - would the method you used to solve N-Queens in Project 2 work well for Sudoku? Why or why not?

(e) (8 points) **Difficulty of Sudoku**
How does the *difficulty* of a Sudoku puzzle relate to the number of unassigned cells? You can explain your answer using theory or intuition for up to 3 marks, but you need to conduct a mini-experimental analysis using your solver and some measure of computational effort for full marks. Feel free to modify the given Sudoku instances or create your own as part of this experiment.

Question 2: Sales Chatbot POMDP (35 points)

Read the entire question and then all the handout code before attempting this question, as the subsections are connected. Only the code in `q2_pomdp.py` (and any saved arrays that you need in files like `lookup_table.npy`) for part (c) is graded by Autolab, but you must submit all code via:

```
tar cvf handin.tar q2_pomdp.py lookup_table.npy OTHER_CODE.py
```

where `OTHER_CODE.py` can be one or more files named whatever you like which contain any additional code that you used to answer this question. It may be convenient to modify `q2_support.py`, which is included in your starter code, and use that as your submission of `OTHER_CODE.py`. Note that your `.npy` file can have any name you'd like, and there are instructions for saving and loading these files in the handout code. The link to the Autolab submission portal for the tar file is here.

You leave your job at Spigot for a new role at tech giant Froogle. Your first task is to implement a chatbot agent that tries to sell products to individuals online. A natural language processing (NLP) model has been trained to recognize three non-terminal states that a potential customer it is communicating with might be in:

1. *Annoyed*: the potential customer is frustrated and on the verge of leaving the chat.
2. *Neutral*: the potential customer is not annoyed with your chatbot, but they are not particularly convinced that making a purchase is a good idea.
3. *Engaged*: the potential customer is interested in the product your chatbot is selling.

Unfortunately, the model is imperfect and only detects the customer's current non-terminal state with probability $p_{\text{cor}} = 1 - p_{\text{err}} = 0.6$, returning each of the other two states with probability $p_{\text{err}}/2 = 0.2$. Each non-terminal state has an opportunity cost of 0.01 (i.e., a reward of -0.01) for Froogle (and therefore your chatbot). There are also two *terminal* states that your model can recognize with perfect accuracy:

1. *Hangup*: the customer has left the chat without making a purchase. This terminal state yields a reward of -1.
2. *Sale*: the customer has ended the chat after agreeing to make a purchase. This terminal state yields a reward of 2.

Your chatbot will make use of another NLP model that responds to a customer with phrases that fall into three categories:

1. *Aggressive*: your chatbot pushes the potential customer to make a purchase.
2. *Informative*: your chatbot educates the potential customer about the product's benefits.
3. *Apologetic*: your chatbot humbly requests the potential customer's forgiveness for its irritating pushiness.

Throughout the starter code in `q2_pomdp.py`, the following integer-based encodings, summarized as Python dictionaries, are used for indexing and referring to states and actions:

```
states =  {0: 'Hangup', 1: 'Annoyed', 2: 'Neutral', 3: 'Engaged',
           4: 'Sale'}
actions = {0: 'Aggressive', 1: 'Informative', 2: 'Apologetic'}.
```

The transition dynamics for this decision problem are described in the 3D-array attribute `T` of the `POMDP` object created by the `get_chatbot_pomdp` function in `q2_pomdp.py` such that:

$$\texttt{T}[s, a, s'] = P(s'|s, a). \tag{1}$$

For example, the probability of the customer transitioning from the *Neutral* state to the *Annoyed* state after the chatbot makes an *Aggressive* action is:

$$P(\textit{Annoyed}|\textit{Neutral}, \textit{Aggressive}) = \texttt{T}[2, 0, 1] = 0.2. \tag{2}$$

Similarly, the noisy perception model is summarized in the 2D-array attribute `P` such that

$$\texttt{P}[e, s] = P(e|s). \tag{3}$$

The POMDP's transition dynamics and rewards are summarized graphically in Figure 2. For all parts of this question, use a discount factor of $\gamma = 1$.

(a) (5 points) **Belief-Space MDP Model** Describe an interpretation of this POMDP as a belief-space MDP model (2 marks). Give the following probabilities (1 mark each):

1. The probability of observing *Hangup* after taking action *Aggressive* when your agent is certain that it is in state *Annoyed*.
2. The probability of observing *Annoyed* after taking action *Informative* when your agent believes the customer is *Annoyed* with probability 0.5, and *Neutral* with probability 0.5.
3. The probability of observing *Neutral* after taking action *Apologetic* when your agent believes that its probability of being in each non-terminal state is equally likely.

(b) (15 points) **Visualizing Utility and Policies**
Implement a value iteration scheme for the belief-space MDP equivalent of the chatbot POMDP that estimates the expected value of states in the continuous belief space. Your estimate should use a uniform discretization of $P_{annoyed} \in [0, 1]$ and $P_{neutral} \in [0, 1]$. Use the plotting functions in `q2_support.py` to plot the expected value (i.e., utility) and corresponding best action to select (i.e., policy) after taking 2, 5, and 10 actions (i.e., lookahead depths for conditional plans). Assign a value (and optimal action) of 0 to all points in each 2D grid plot that do not correspond to admissible beliefs (i.e., when $P_{annoyed} + P_{neutral} > 1$): the sample code in `q2_support.py` demonstrates this. Comment on the trend in the value function estimate as the number of steps increases. What are the best belief states to be in at each lookahead depth? Comment on the trend in the policy as the number of steps increases. What mistakes do policies that do not look far ahead make? Your code may take 10 or more minutes to run, even when implemented efficiently. If you are unable to produce value estimates for plans with a lookahead depth of 5 or 10, present plots for the greatest depth you can compute and discuss the trend you see. **Hint**: some method of removing dominated plans is necessary for depths greater than 2. The textbook (AIMA pg. 663) recommends a linear programming approach, but much simpler approximate methods can work well for our relatively small POMDP.

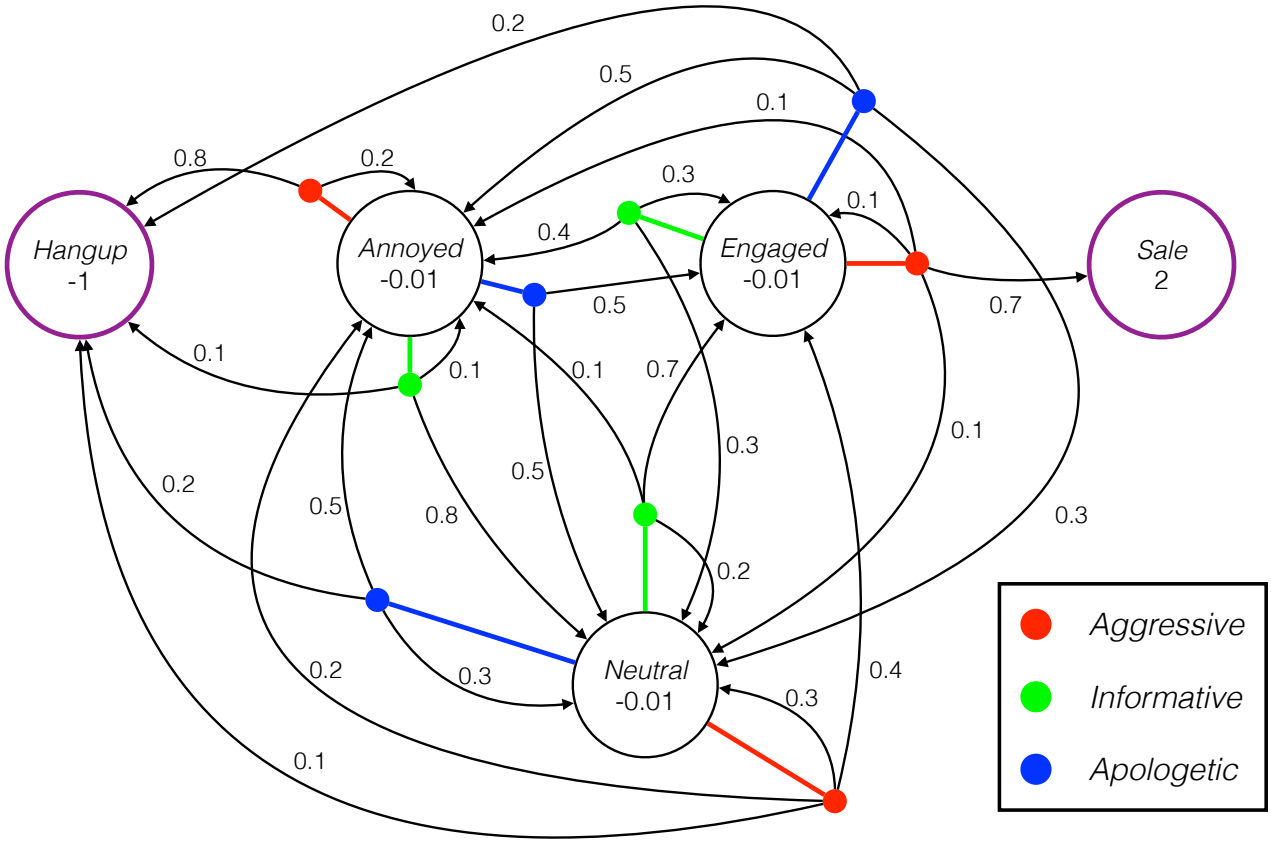


Figure 2: Graphical representation of the transition dynamics for the POMDP used in Question 2. The 3D tensor `T` in the `POMDP` object in `q2_pomdp.py` implements these dynamics as a lookup table.

(c) (10 points) **Autolab: Test Your Policy**
Implement the class `ChatbotSolver` in `q2_pomdp.py`. This function takes in a belief state at the start of each chat session with a customer, and outputs an action as a function of a percept (observation) at every time step afterwards. The belief state will be encoded as a `numpy` array $b = [P_{hangup},\ P_{annoyed},\ P_{neutral}, P_{engaged}, P_{sale}]$ with probabilities $P_{annoyed} + P_{neutral} + P_{engaged} = 1$ and $P_{hangup} = P_{sale} = 0$ (i.e., we never begin in a terminal state). Your function will be evaluated over N runs. Your implementation should incorporate your solution to 2b, but you should be able to achieve part marks with a simpler approach if you are unable to complete 2b. You do **not** have to run your value iteration algorithm from 2b within your implementation o `ChatbotSolver`, as this would take far too long on the Autolab server. Instead, run it on your own machine when answering 2b, and save the resulting policy or a related lookup table as a `numpy` array in a `.npy` format. Instructions for saving and loading `.npy` files can be found in the initialization method for `ChatbotSolver` in `q2_pomdp.py`. You are limited to **10 submissions** for this question. As usual, your **latest** submission will be graded (not your submission with the highest score). Be sure to test your solver on your own machine first to avoid running out of submissions and abusing the Autolab server.

(d) (5 points) **Algorithm Description**
Describe the methods you used in 2b and 2c. If you did not complete these questions, describe an approach you think would work, or propose solutions that fill in any gaps in your implementation.

QUESTION 3: LEARNING TO DRIVE (30 points)

Read the entire question and then all the handout code before attempting this question, as the subsections are connected. Only the code in `q3_rl.py` for part (b) is graded by Autolab, but you must submit all code you use (e.g., for the plots in 3a) via:
`tar cvf handin.tar q3_rl.py OTHER_CODE.py`

where `OTHER_CODE.py` can be one or more files named whatever you like which contain any additional code that you used to answer this question. The link to the upload portal for the tar file is here.

Bored of your stint at Froogle, you take a new position with Goober, the world's leading autonomous pizza delivery service, where they are attempting to develop a cheap learning-based approach to driving a car. Goober's vehicles have wheels that are covered in pizza grease, leading to the stochastic transition dynamics depicted in Figure 3 and described in the class `CarMDP` in `q3_rl.py`. You are tasked with developing a reinforcement learning-based prototype that drives a loop through a finite grid Markov decision process (MDP) environment like the one in Figure 4. Your agent always begins in the top left corner ($x = 0, y = 0$) facing *South* and receives a reward of 10 and terminates a training episode if it loops counterclockwise around the grid map and transitions from cell position $x = 1, y = 0$ to the starting cell. The agent's absolute orientations and action set are encoded in `q3_rl.py` as:

```
orientations = {0: 'North', 1: 'East', 2: 'South', 3: 'West'}
A = {0: 'Forward', 1: 'Left', 2: 'Right', 3: 'Brake'}
```

The agent's complete state at each time step is encoded as the 3-element tuple (x, y, *Orientation*). For example, the starting state S_0 is encoded (0, 0, 2). Cells with obstacles are drawn in black in Figure 4 and, along with cells outside of the fixed 9-by-6 grid map, represent terminal "crash" states that reward a penalty of -5. Any other non-terminating action wastes time and money and therefore rewards a penalty of -0.01. You must develop a **model-free** reinforcement learning algorithm that takes actions and observes rewards and subsequent states. For all parts of this question, use a discount factor of $\gamma = 1$.

(a) (10 points) **Reinforcement Learning Algorithm**

Implement a tabular model-free reinforcement learning algorithm in the class `ReinforcementLearningAgent` in `q3_rl.py`. Since this is model-free RL, your solution should not make any assumptions about the transition dynamics, the size of the state space, or the rewards for actions. Choose any parameter used by your algorithm (e.g., exploration rate ϵ or step size α). For three or more values of your parameter, test your algorithm on the sample MDP given in `q3_rl.py`, saving the value of the returns for each episode. Note that each episode must begin in the designated initial state $S_0 = (0, 0, 2)$. In the same figure, plot the average reward vs. episode over 10 or more evaluations of each parameterization of your algorithm (use a legend and a different colour to distinguish each curve from the othersf). Each evaluation should involve at least 10,000 episodes. The code at the end of `q3_rl.py` shows an example plot for the default random policy (which you

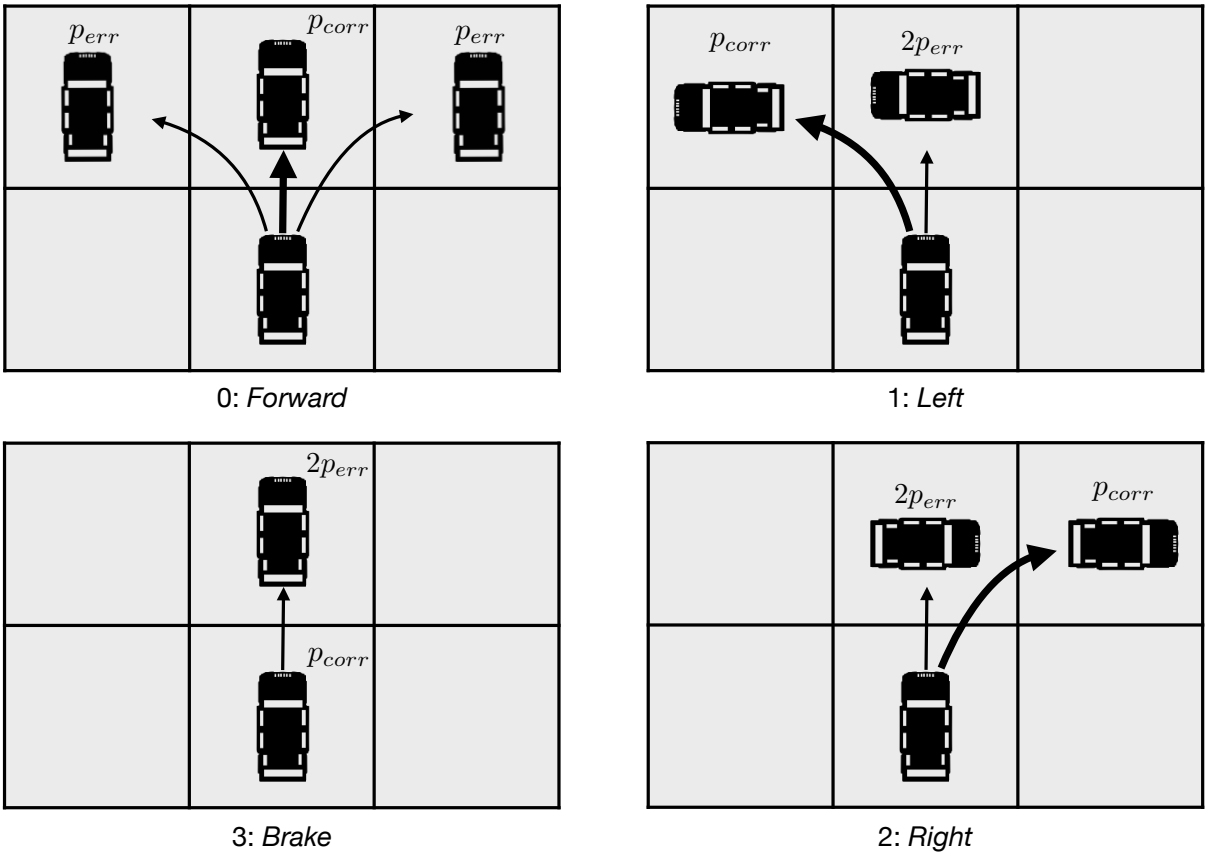
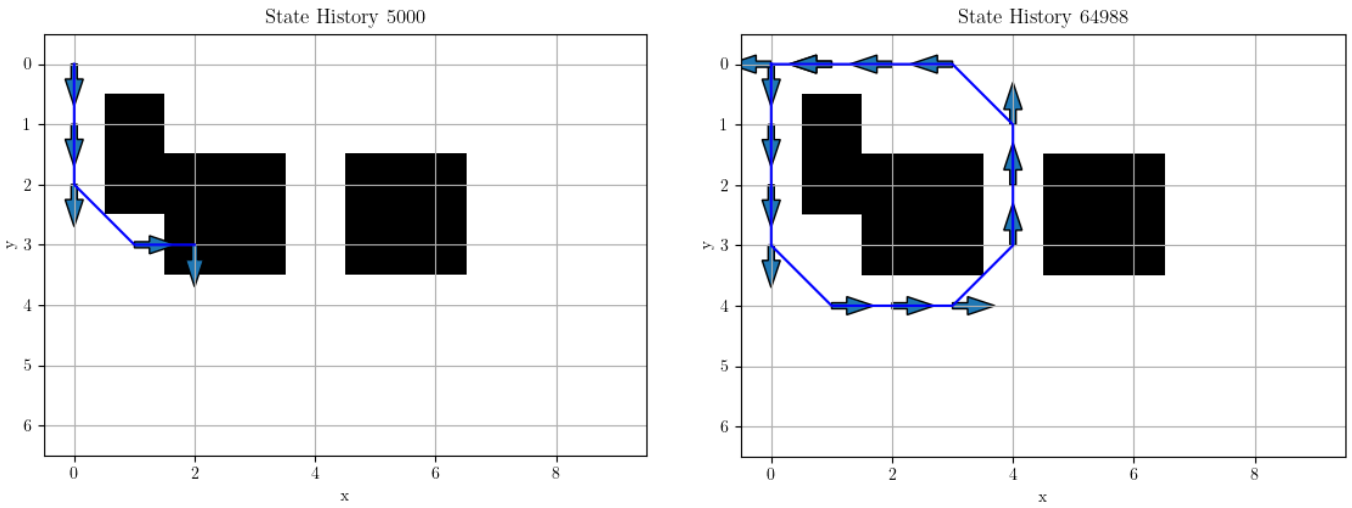


Figure 3: Probabilistic transition dynamics of the car MDP in Question 3 for each of its four actions (*Forward*, *Left*, *Right*, and *Brake*). The diagrams show all possible outcomes (i.e., $p_{corr} + 2p_{err} = 1$.). Each action's effect is relative to the car's orientation: in the four figures shown here, the car is in the *North* orientation. For example, the figures would simply be rotated 90 degrees clockwise if the car were in the *East* orientation. The brake action is not very useful for the MDP in the handout code, but it is useful for the hidden MDP used by Autolab to test your model-free RL algorithm.



(a) Unsuccessful episode (b) Successful episode

Figure 4: A comparison of a successful and failed episode at different points in a simple model-free RL algorithm's interactions with the car MDP environment. After 5000 episodes, the agent still frequently crashes into an obstacle. After 64,988 episodes, the agent is often able to achieve its goal of looping back to the start.

will be replacing) in `ReinforcementLearningAgent`. Discuss any trends you see. **Hint**: Python dictionaries may be helpful for storing tabular data on problems of this size.

(b) (15 points) **Autolab: Test Your Algorithm on an Unseen Environment**
Submit your implementation of `ReinforcementLearningAgent` in `reinforcement_learning.py` via Autolab. Your algorithm will be tested on a random grid MDP similar to the one provided in `reinforcement_learning.py`, but with different, **nonstationary** transition dynamics and a different grid map. More specifically, the nonstationarity across training episodes includes both a slow "drift" in the MDP's dynamics, and sudden major changes to the state space's structure and reward function (e.g., new obstacles appearing on the track). You have a limited number of submissions, so you may want to test your algorithm offline on variants of the provided MDP class. Your algorithm will be run for 50,000 episodes by Autolab, so you should test it to make sure it can run in less than a minute on your machine first. You are limited to **10 submissions** for this question. As usual, your **latest** submission will be graded (not your submission with the highest score). Be sure to test your solver on your own machine first to avoid running out of submissions and abusing the Autolab server.

(c) (5 points) **Algorithm Description**
Provide as many technical details as possible, and explain what motivated your design decisions. If applicable, discuss how your plot in 3a motivated your choice of parameters or other algorithmic details. How did your algorithm balance exploration and exploitation? How did the nonstationarity of the test in 3b affect your choice of algorithm and parameters? If you did not complete a working algorithm, do your best to describe an algorithm you think would

work, or explain what remains to be finished or improved upon.