1. This policy runs for 300 episodes for a total of 200 iterations in each episode. For each iteration, we check the conditions in the observations and set the action based on that. If the previous velocity is negative and the next velocity is positive, the action is set to 2. If the previous velocity is positive and the next reward is velocity, we set the current action to 1. We store the previous velocity based on that. If we are done (reached a goal position of 0.5), we break out of the loop and stop doing any actions. If the action is 1, we do not push the car anywhere. If the action is 2, we push the cart to the right.
2. This policy runs for 500 episodes for a total of 200 iterations in each episode. For each iteration, we check the pole angle and pole velocity at the tip and set the action based on that. If we have a positive pole angle with positive pole tip velocity, we set the action to 1 which pushes the cart to the right. Else, the cart moves to the right. If the pole angle is negative and the pole velocity is negative, the cart moves to the left. Else the cart moves to the right. The total reward for each episode is 200 at the maximum. We do not iterate above that.
3. For the cross entropy method, the best episodes are selected the the network learns from these episodes how to select an action based on the reducing reward. Logits are used for storing the probability distributions and these are yielded by the softmax layer. The mountain car is trained with a batch size of 25 and then with a learning rate of 0.01. Then, it is run for 100 episodes for a total of 200 time steps in each episode. From this cross entropy, the mountain car learns how to climb the hill accordingly.

Search algorithms: Learned about two different types of search algorithms namely informed and uninformed search algorithms. Search problems are used to take a start space and return a sequence of actions that transforms the start state into a goal state. They take a state space, a successor function that has costs associated with actions, and the start and goal states. Learned about the general algorithm for a search. Learned about uniformed search algorithms such as depth-first search (expand a deepest node first), breadth-first search (expands a shallowest node first), uniform cost search (expand a cheapest node first). Learned about informed search algorithms and search heuristics. Learned about the greedy search (expand the closest node first), A star search (combination of greedy search and uniform cost search). Learned about the admissibility of search heuristics.

Adversarial search: Learned about deterministic and stochastic environment. Learned about calculating policies that determine the actions moving forward to the next state of a problem. Deterministic problems in the states of an environment and calculate the set of actions for an agent to solve the problem. This is done with the transition function. Learned about the different states for deterministic problems (terminal and non-terminal states). Learned about minimax search and how each node’s minimax value is computed to attain the best achievable utility against a rational opponent. For minimax searches, the agent determines if the minimum or the maximum outcome of the state’s successor and returns the value based on the optimal successor. Learned about alpha-beta pruning where alpha is the max agent’s best option on path to root and beta is the min agent’s best option on path to root. Learned about expectimax search and how it is used to compute the average values by computing only the max nodes in a search tree. This is done by calculating the probability of the success of the successor and adds the score of the successor multiplied with the probability to the total score.

Markov Decision Processes (MDP): Learned about non-deterministic search problems and about the general structure of a MDP (set of states, set of actions, transition function, reward function, start state, terminal state). In MDPs, action outcomes depend on the current state of the agent. Here, policies provide actions for each state and maximize the utilities that follow. Learned about discounting and how each reward is discounted for each state making the MDPs goal state to maximize the amount of reward achieved. Policies should be able to compute maximum utilities. Learned about value iteration and how we perform an expectimax from each state until the algorithm converges. Learned about Bellman equations and how they do a one-step lookahead to compute the optimal values. These equations are computed through value iteration and they converge until the discount is less than 1 or the search tree hits a maximum depth. Learned about computing actions for an agent based on values. This is done with a mini-expectimax and is called as policy extraction. Learned about policy extraction where the utilities are calculated for a fixed policy until convergence. Another way to update the policy is to look at the one step ahead with the resulting values. This is done until the policy converges.

Reinforcement Learning: Learned about the reinforcement learning and the general structure of a reinforcement learning problem. It follows the same structure as that of a MDP except the fact that we do not know the model or the reward function. The agent has to learn about the environment, states, and actions based on what they learn and the rewards as well before acting out on the environment. There are two different types of reinforcement learning: model-based learning and model-free learning. In model-based learning, the agent learns an approximate model of the environment based on the training and it attempts to solve for values if the learned model is correct. The agent first trains in the empirical MDP model and then solves the MDP. In model-free learning, the task of policy evaluation is to learn the state values given that the transitions and the rewards are not given. In this case, the agent is first trained without any knowledge of the environment. Learned about direct evaluation and how agents keep a track of the discounted rewards for the actions it performs. This is passive reinforcement learning. In active reinforcement learning, the transitions and rewards are not known and the actions are chosen to learn the optimal policy. Learned about Q-value iteration and Q-learning. In Q-learning, we receive a sample transition, the old estimate, and the new estimate. This is incorporated into a running average. Q-learning always converges to an optimal policy. Q-learning is done through random actions where small probabilities lead to random actions and large probabilities enable to agents to act on current policies. Agents explore previously unexplored states. Learned about exploration functions for reinforcement learning. Learned about the linear approximation of Q-learning function with linear regression. Learned about minimizing the errors for Q-learning. Learned about policy search with features and linear approximation. Learned about neural networks and how we can implement neural networks with Q-learning.

Note: Questions 1-3 describe the three problems solved for the homework. Question 4 answers the overview of all the algorithms used in the course.