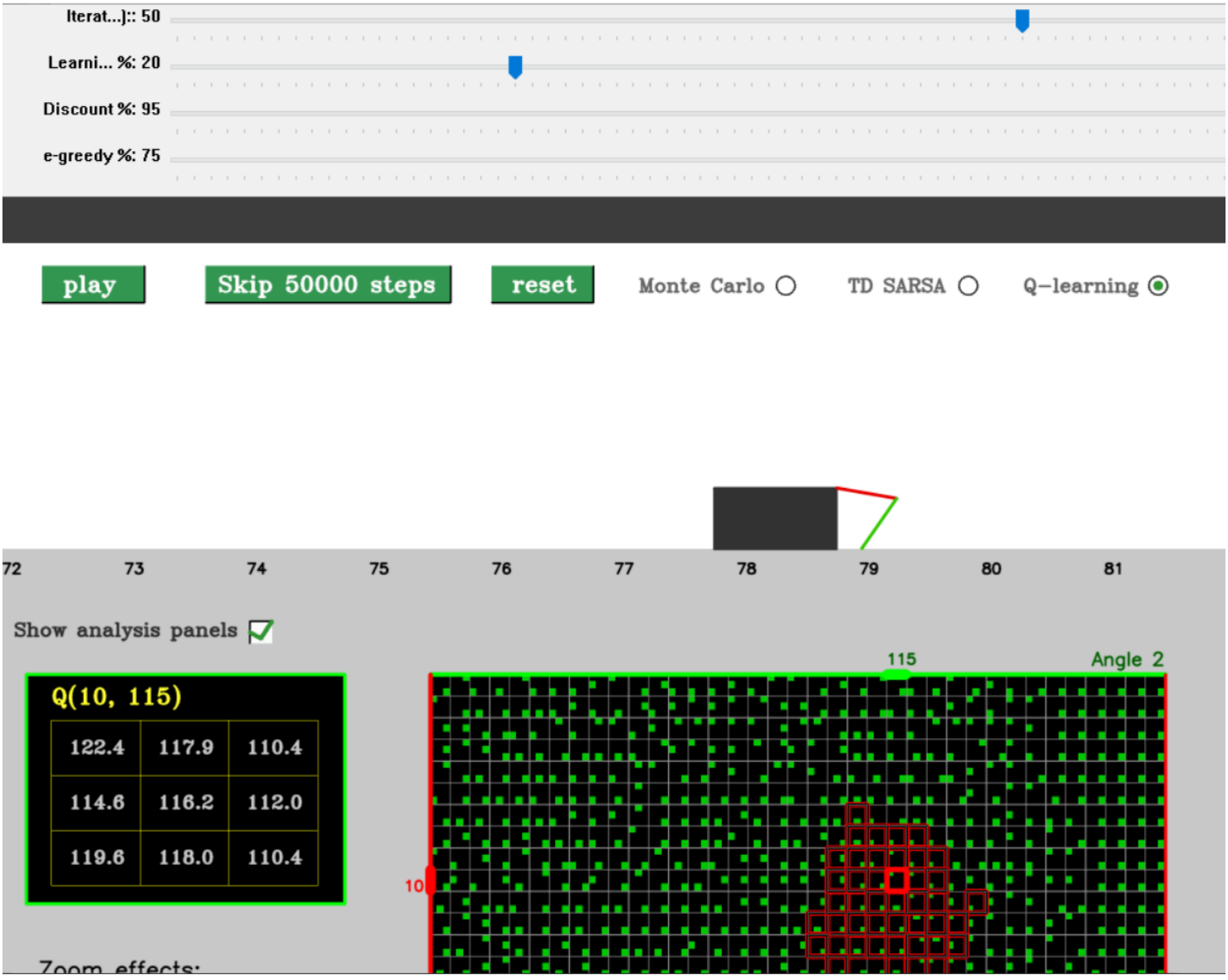
Q-Learning based Reinforcement

Vincent Durham

I was able to implement both the chooseAction() and onQLearning() functions to get the bot to learn and move forward. With more learning, the bot was able to move forward quickly. It was interesting seeing the results of different variables with the sliders provided. The best way to get the bot moving fast is to learn(skip steps) a few times with the original sliders, and then increase the e-greedy to above 75 and learn(skip steps) a couple more times to know the best path to take to move forward. The chooseAction() function uses a list of possible action indexes and probabilities set to (1-epsilon)/9. Then, it finds all of the q-values of the possible moves the bot could make, and the highest q-value gets epsilon added to its probability index. Using numpy, an action index is selected randomly using the probabilities provided. That action index is used to provide the angle updates for the next move. The onQLearning() loop through steps to find the current state and location, and the next move's state and location for the bot. The bot is set to the next move, and the q-value for the move made is updated in the former current state.

-The analysis panel after two learning steps before beginning to walk.

- A picture of the q-value version of the analysis and where the arms have visited after multiple learning steps (The bot is already 78 units moved)



- A picture of the bot after multiple learning steps and running for a couple of minutes. It is 500 units moved from the original 0 position.