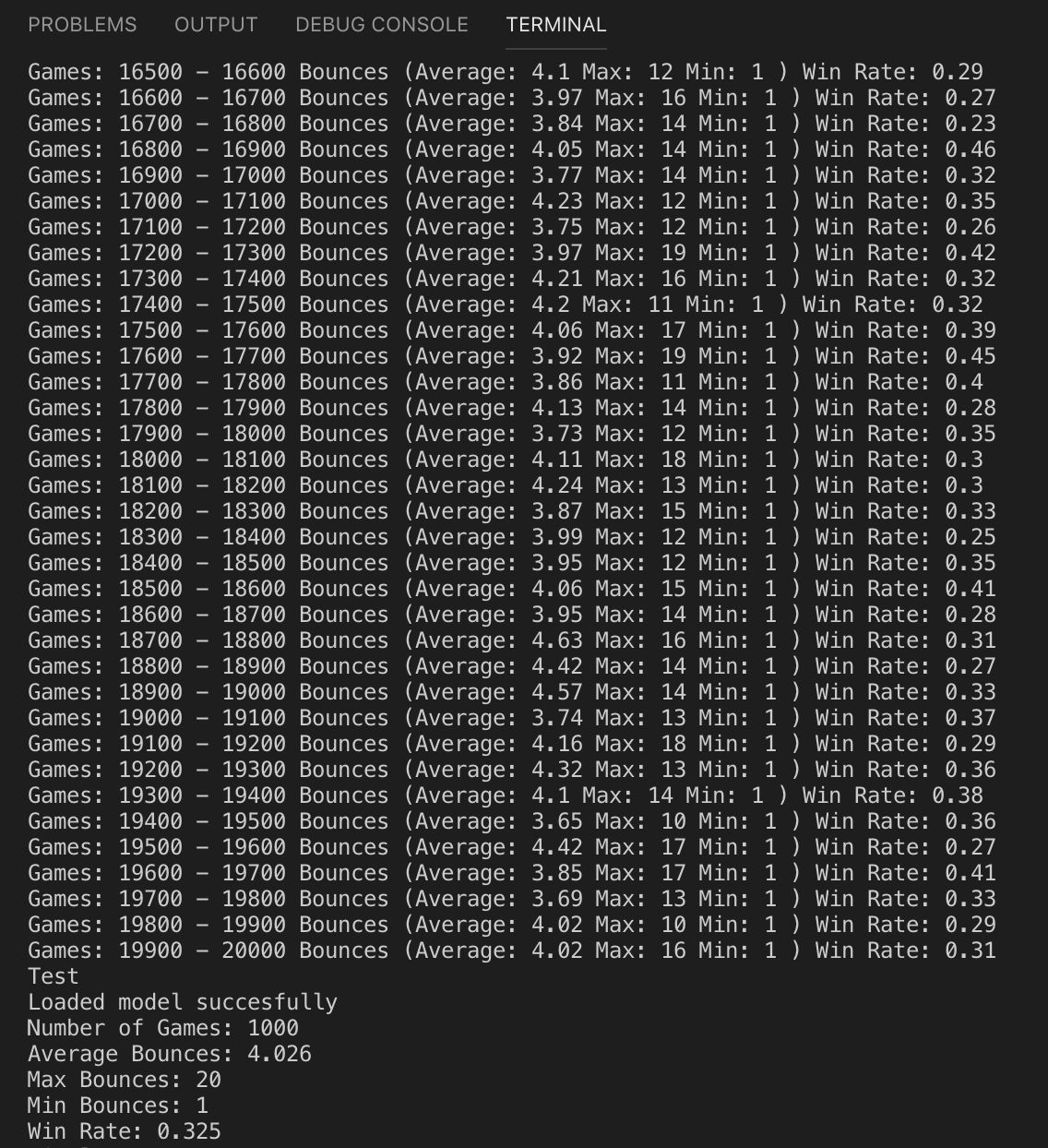
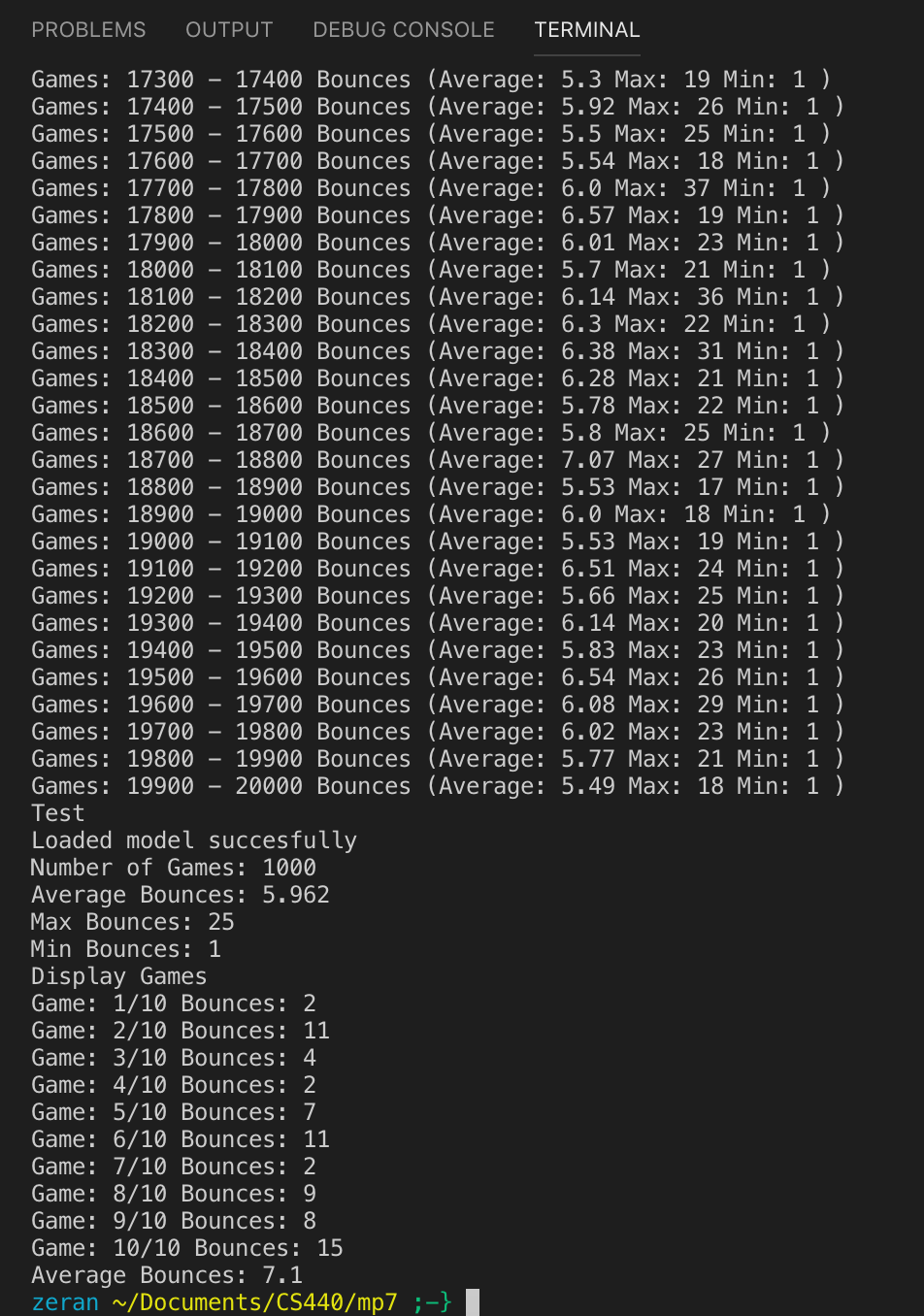
For extra credits, we experimented the sequence action model, a variation of Q-Learning algorithm.

The idea of our own model is, after discretization, the state space of this game and the actions can lead to a directed graph, with a state space of vertices and each vertex has all possible actions as edges. In this setup, each bounce, or each termination, can be interpreted as a path from the starting state to the terminating state. Along this path, if there is a bounce in the end, each action deserves a reward, while if there is a failure, each action must all be punished because they together lead to the end of game.

During training, the agent has an internal memory of past actions, and all actions are rewarded/punished whenever a bounce/death occurs.



Our model can be trained in a short period of time. Typically, 20k iterations are enough to reach 6~7 average bounce performance, with a winning rate of 0.35~0.45 in opponent mode. We believe that with sufficient tuning and adjustment of our parameters and algorithm, for example, integrating exploration rate and learning rate, will lead us to a much better performance in this model. However, this baseline model already has impressive performance and learning speed without using those hyper parameters.

Interestingly, this model is actually our own re-discovery of a technique called “Reverse update” in section “speedup of goal-based problems in reinforcement learning” found in the slides (63-64) of a Washington University CS 573. (<https://courses.cs.washington.edu/courses/cse573/12au/slides/08-rl.pdf>)

To run our training code, change line 13 of mp7.py to “from agent\_ec import Agent” and proceed as normal.