Thought Process: The "random" play beats some bots but not the stronger ones like Greenberg and Iocane Powder. Altough to pass, I need something that beats Greenberg and Iocane Powder. However, the "something" often "overfit" to specifies bots if the algorithm is designed ad-hocly. So I need something to generalize this.

There's a cool phrase from china saying something like "Born from indigo, yet bluer than indigo". Meaning things derived based on an existing instance may be better. So I took the idea of Greenberg that makes an ensemble of "experts" that implements different algorithms, and have different "experts" playing against different opponents accordingly. The original Greenberg has experts playing strategies like fictitous play (frequency counts), opponent's opponent, markovs chain models, mirror-self. Then Greenberg will maintain the performance of each experts (win - +score, lose - -score), and further do an argmax among all experts solutions.

This is pretty smart as it gets to choose which strategy to play against different opponents, which is the anti-ad-hoc-ness (if that makes sense) we wanted. Now comes the part where I want to beat greenberg and iocane powder.

I researched about greenberg and iocane powder, and found that at the time they're designed, ML isn't as crazy as nowadays. Therefore I believe that adding in some ML stuff (that models opponents move) will increase my win rate. As well, since I learned in class, why not just through in a MDP model. So my agent consists of 7 experts, they're the frequency-based, opponents opponent, markovs model, mirror-self, ML(LSTM), random action, mdp expert. For updating the experts scores, I used the exponential recency-weighted average to accommodate for non-stationarity.

For ml, I have trained lstm model on the data of the original model (without ml and mdp) playing against all 43 bots for 10 episodes each, and each data point consist of a sequence of 200 actions. I used a focal loss as the criteron rather than categorical cross entropy because focal loss can handle class imbalance in case some silly strategy just play a single action every single state.

For MDP, the state represent things happened last round, so s = (my move, opp move), therefore there's in total 3*3=9 states, to hard code the start state, we add one. the actions is as defined in the problem, (0-R, 1-P, 2-S). the reward is the payoff matrix and the trasition model is solved by emperical MDP with value iteration. then used to derive an optmial policy. I'm afraid that if running value iteration every single action will blow up the running time, so I have set a length to run value iteration.

After all of those modeling, the agent can finally take step based on the best expert of random (epsilon-greedy exploration).