a) Environment description:

The code implements a simplified version of a two-player poker game that follows the rules that are specified in the project's logistics. A few classes and methods from the https://rlcard.org/ library were used such as Card, Dealer, printCard() etc. The key points of the implementation are:

- Dealer and Deck: A deck of 20 cards is used consisting of ranks '10', 'J', 'Q', 'K', and 'A' in all possible suits ('S', 'H', 'D', 'C'). The **PokerDealer** class represents the dealer, responsible for initializing and shuffling the deck as well as dealing cards to the players.
- Players and opponents: The player and opponent classes are designed so that any player type can play against any opponent type(Random or Threshold), but it is also possible for players or opponents to play against each other. Each one has attributes such as the number of chips they have (**in_chips**), the number of chips they have won (**win_chips**), their hand (**hand**), and their status (**status**).
  - **HumanAgent**: This agent represents a human player who can interactively play the game by displaying the player's hand and the public cards and prompting the user to choose a valid action ('call', 'fold', 'check', or 'raise').
  - **PolicyAgent**: This agent follows a specific policy for decision-making, which depends on whether the agent is playing as player 1 or 2 against a random or threshold opponent. The policy considers the game state, including the agent's hand and the public cards, to determine the best action to take in order to maximize the agent's chances of winning.
  - **QLearningAgent**: This agent is similar to PolicyAgent, but the opponent type is irrelevant to the decision making and Q Learning algorithm is used.
  - **RandomAgent**: This agent chooses a legal action to play randomly.
  - **ThresholdAgent**: This agent plays based on certain thresholds, such as calling or raising with a high-rank hand.
- Judging the winner: The **PokerJudger** class determines the winner/winners of the game based on the players' hands and the public cards and returns the payoffs for each player.
- Gameplay: The **PokerGame** class initializes the game environment and handles the flow of the game. The game involves an ante of 0.5 chips, a bet of 1 chip, and 2 rounds of play. The **playGame()** method of PokerGame plays a single game of poker, at which the players take turns and make actions until the game is over. Players' actions are determined by their respective **step** methods, which return the chosen action based on the current state.

b) Policy Iteration:

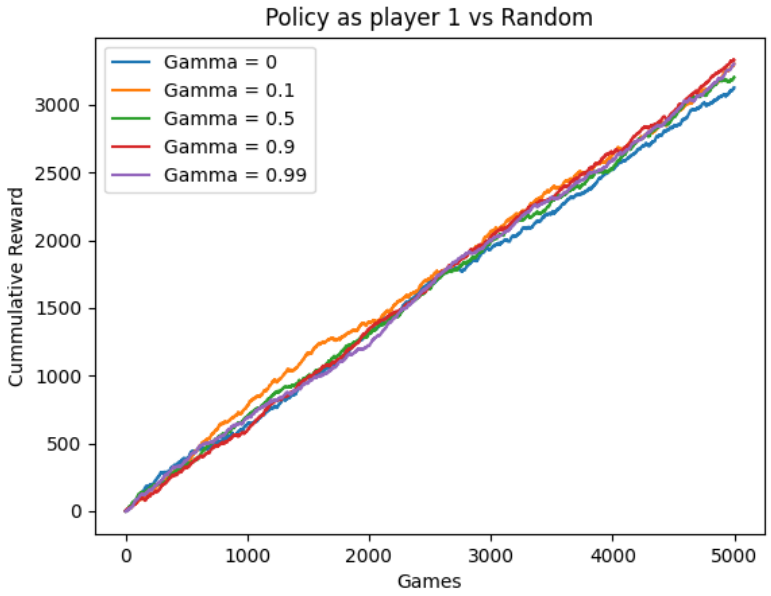
First, in order to check whether the Policy Iteration algorithm indeed converges to the optimal, we have used 2 controls:

1. During policy evaluation, before returning the new policy, we check whether the new value function that is calculated is close enough to the previous one. We use the parameter epsilon to define how close the new value function we want to be to the previous one.

2. During policy iteration, we stop the iteration only when the new policy that occurs is exactly the same as the one in the previous step.

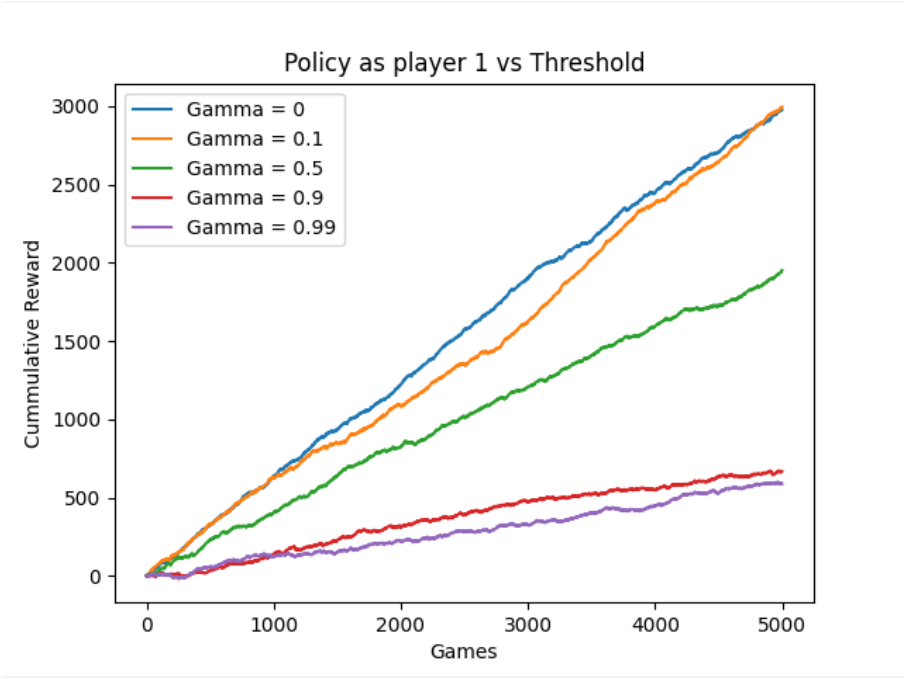
After we prove that our policy iteration converges, we need to find out which gammas and epsilons are better for playing against a Random or Threshold Agent.

Here is how gamma affects the performance of Policy as Player 1 vs Random:



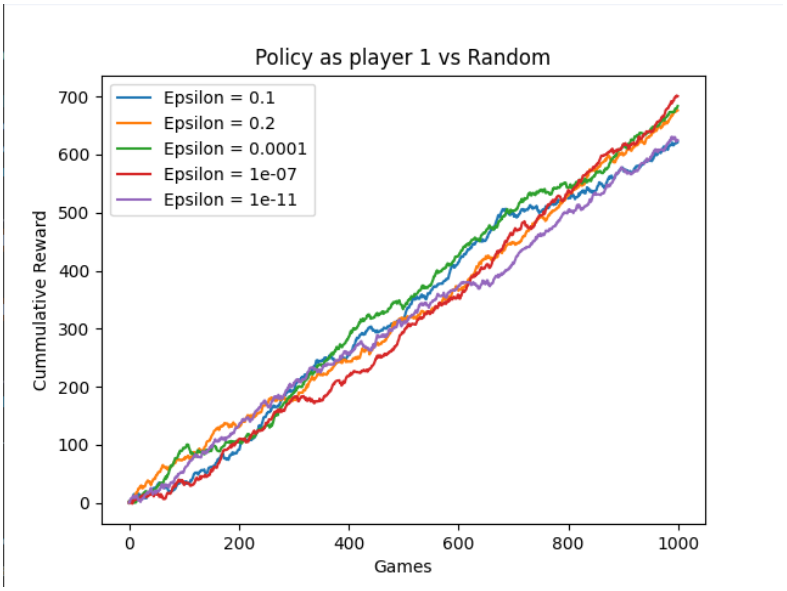
We can see that gamma does not really affect our performance, whether we play risky or strategically, we get similar rewards, because Random opponent plays randomly. We can see a similar performance while Policy is player 2 and Random is player 1. Therefore we consider that our optimal gamma is 0.9.

We do the same experiment with Threshold as opponent:



We can see that gamma really affects our performance. When we play risky (gamma = 0.1), we gain way more rewards than when we try to maximize our long term rewards. This is logical since Threshold will call only when he has a high card or pair; therefore, if we play risky and raise a lot, he will probably fold. Therefore, we consider gamma = 0.1 as the optimal gamma when playing vs. Threshold Agent.

We used the same process to find the optimal epsilons. Playing against Random opponent gave us the following graphs:

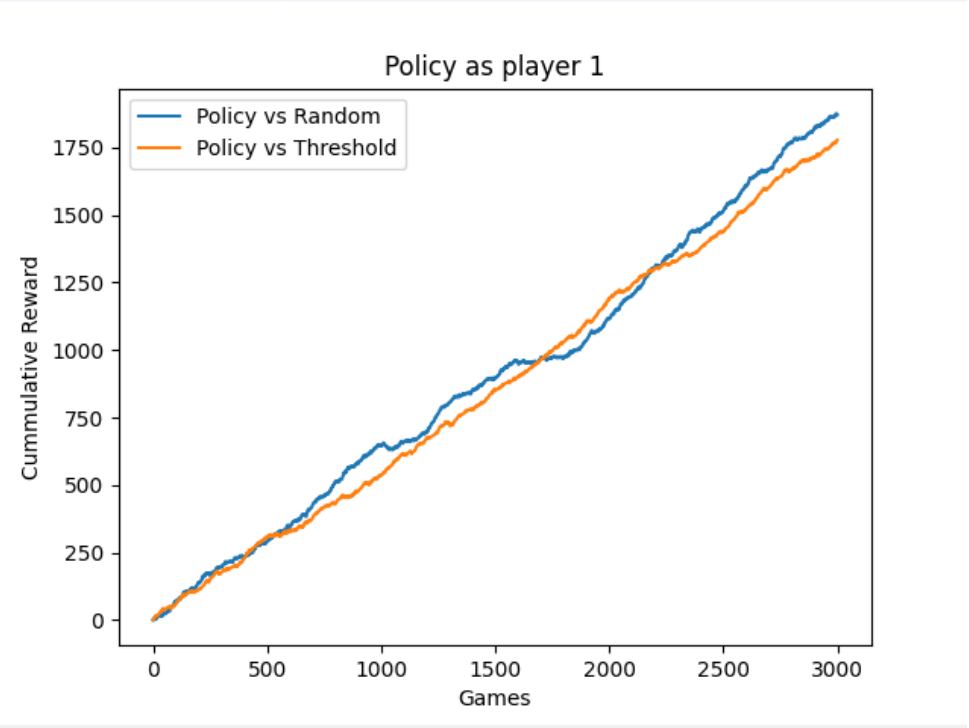


Since epsilon seems to play a less significant role here, we consider 1e-07 as the optimal epsilon. We also checked whether it increased the iterations using PI, but they did not change significantly.

The same epsilon seemed optimal for playing against the Threshold agent too.

Next, we tested our Policy agent vs the 2 opponents using the parameters we found above to calculate the average reward.

When Policy is Player 1, we get the following:

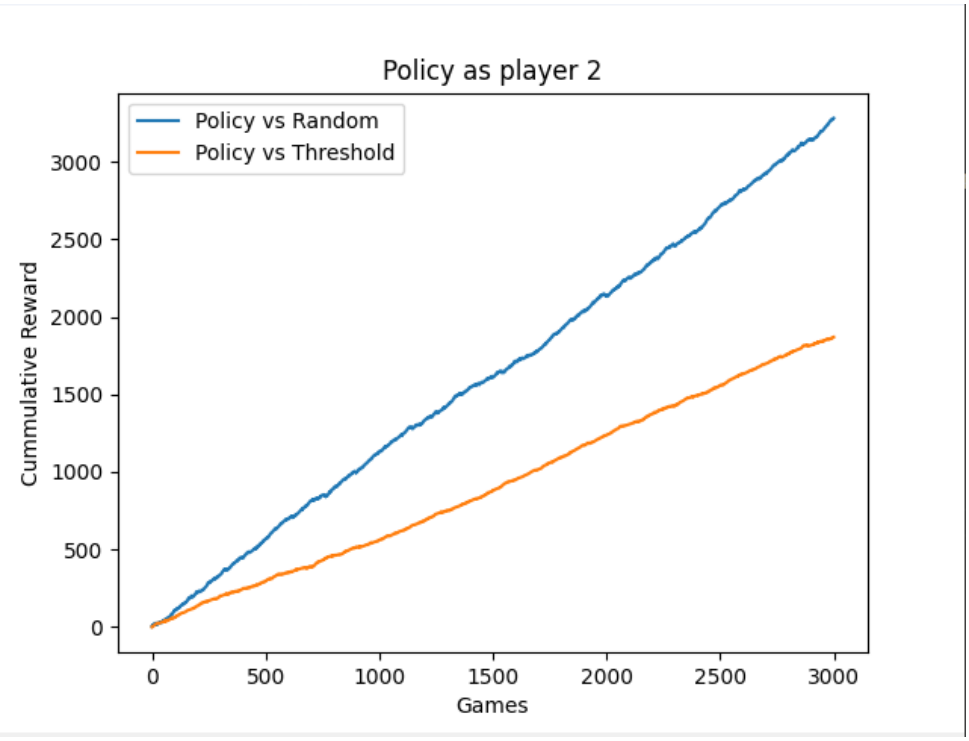


Playing against Random, we get a mean reward = 0.6235.

Playing against Threshold, we get mean reward = 0.5923

We can see that our algorithm is winning approximately the same rewards for both opponents. The cumulative reward is positive, meaning that we mostly win games, so our Policy Iteration definitely works. We would expect to win less vs. the Threshold opponent , but it seems that because he usually folds if we raise, it results in more wins than expected when playing against a tight opponent.

Things change a bit when Policy Agent is Player 2:



Playing against Random, we get mean reward = 1.094

Playing against Threshold, we get mean reward = 0.6231

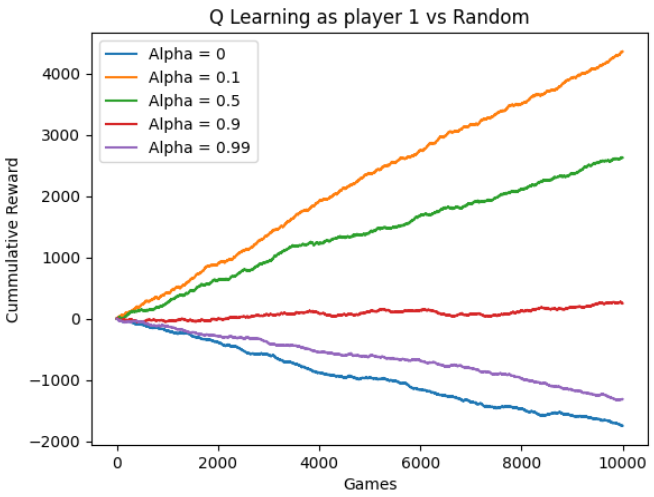
Playing against Random now accumulates better rewards than before.This may be a result of Random Agent folding randomly, that in the case of Random being Player 1, it is a more frequent phenomenon. That's because Player 1 can sometimes play 2 times per round, while Player 2 only one.

In the case of playing against the Threshold opponent, we accumulate similar rewards as before with the new mean reward pretty close to the previous one (when Threshold is Player 2). This is expected since the behavior of Threshold in both cases is pretty much the same.

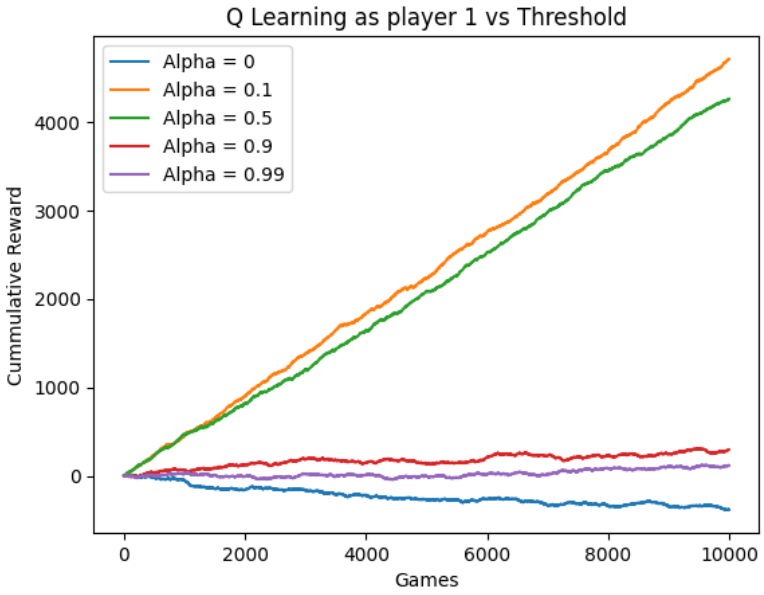
c) Q Learning:

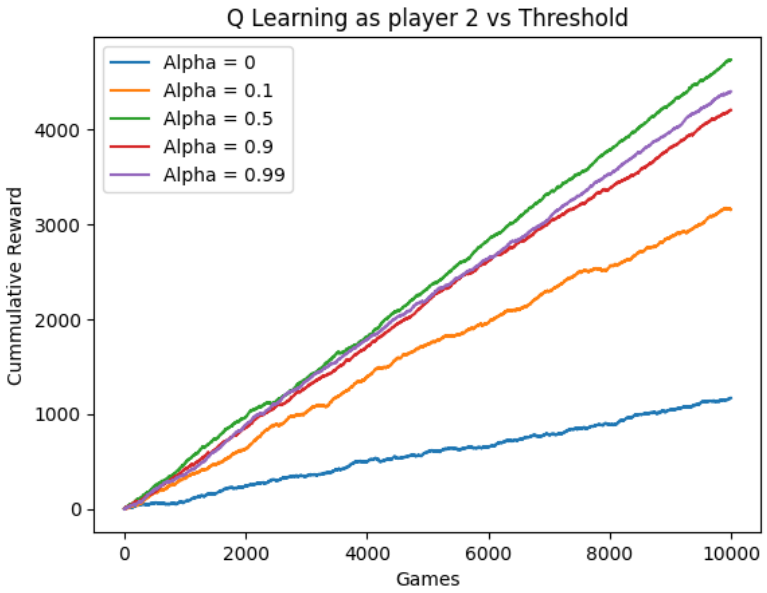
We run experiments to find out the optimal alphas, gammas and epsilons that enhance the agent's performance against a Random or Threshold opponent.

First, we set gamma = 0.9 and epsilon = 0.1 and plot the cumulative reward for different alphas.



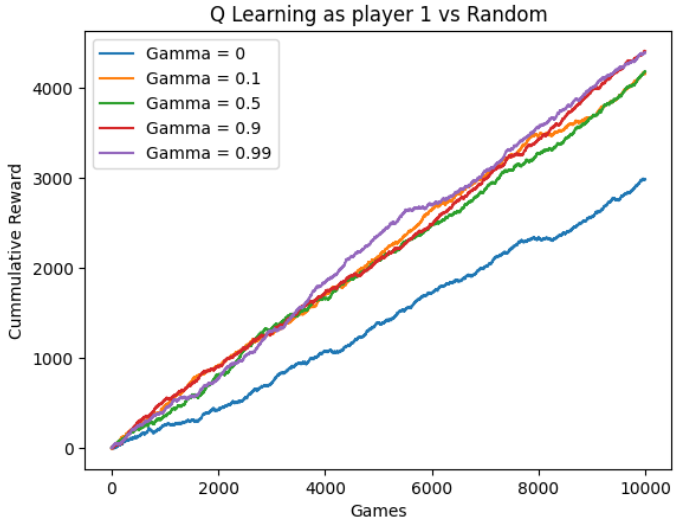
When playing against a random opponent, either our agent is player 1 or 2, the optimal alpha seems to be 0.1. This is because with a small learning rate we explore more the uncertain environment of the random opponent and thus adapt better to the opponent's actions.



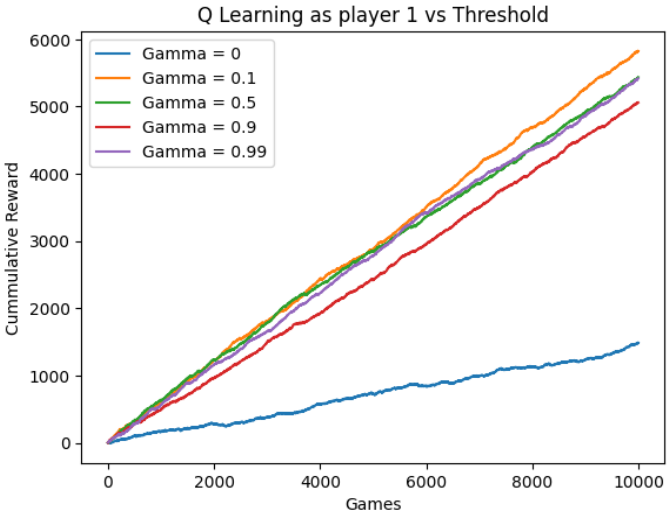


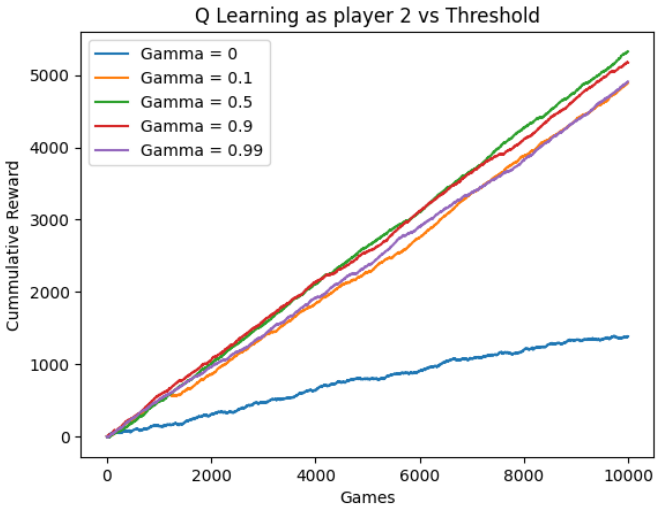
When playing against a threshold opponent, the optimal alpha seems to be 0.1 when our agent is player 1 and 0.5 when our agent is player 2. This difference is because as player 1, a lower alpha value allows cautious learning, while as player 2, a higher alpha value enables assertive adaptation.

Afterwards, we keep the best alphas for each case and move on to test performance for different gammas:



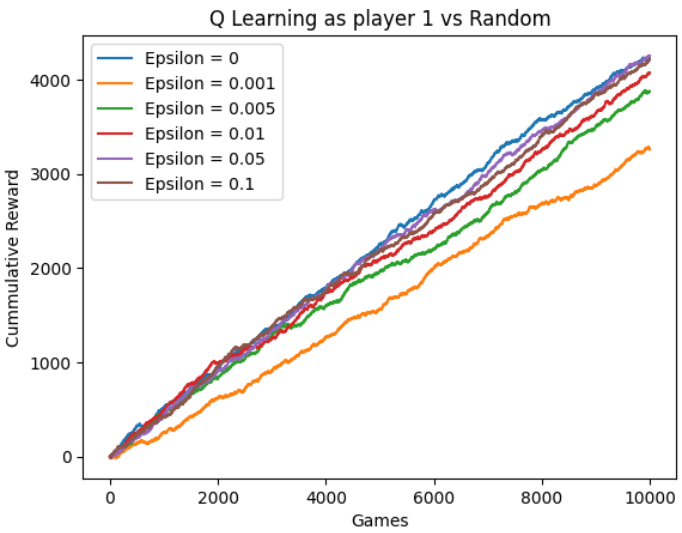
Either playing as player 1 or 2 against a Random opponent, Q Learning agent performs better with gamma = 0.99. A high gamma value improves performance because it considers long-term rewards and enables the agent to learn and optimize its strategies effectively.



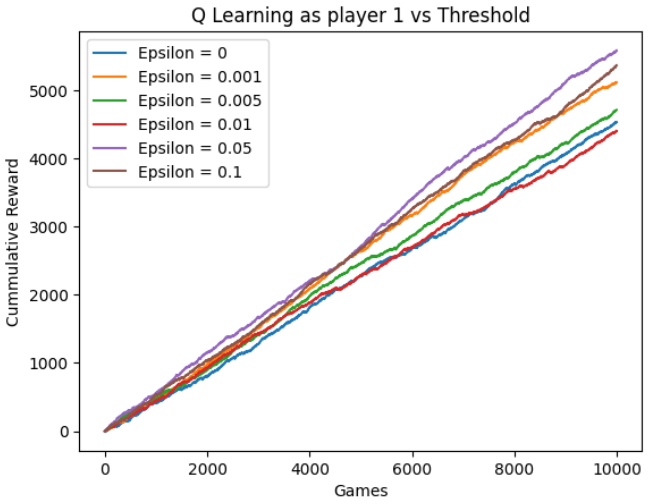


When playing against a threshold opponent, the optimal gamma seems to be 0.1 when our agent is player 1 and 0.5 when our agent is player 2. This is because when our agent is player 1, he emphasizes on immediate rewards and exploits the opponent's weaknesses with more confidence. But when he is player 2, he strikes a balance between short-term gains.

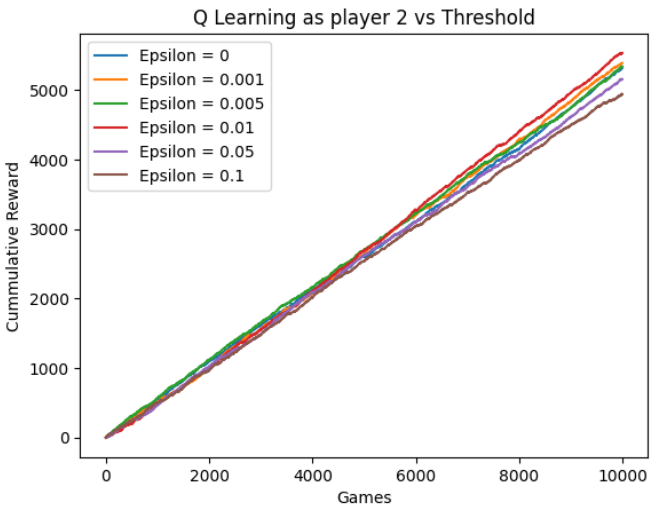
Finally, we keep the best alphas and gammas and test performance for different epsilons:



When playing as player 1 or player 2 against a Random opponent, the optimal epsilon seems to be 0. This is because random opponents don't have a fixed strategy, so there is no need for exploration to adapt to changing opponent behavior.

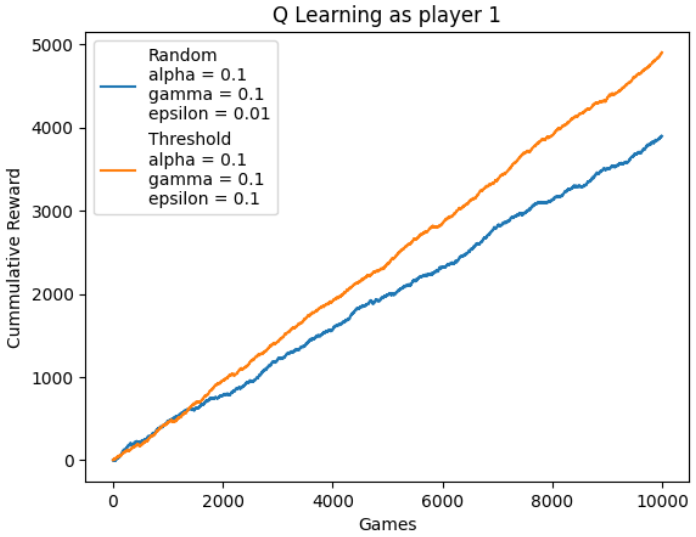


When playing as player 1 against a Threshold opponent, the optimal epsilon seems to be 0.05. This allows the agent to mainly exploit its learned policy while still exploring occasionally for potential improvements.

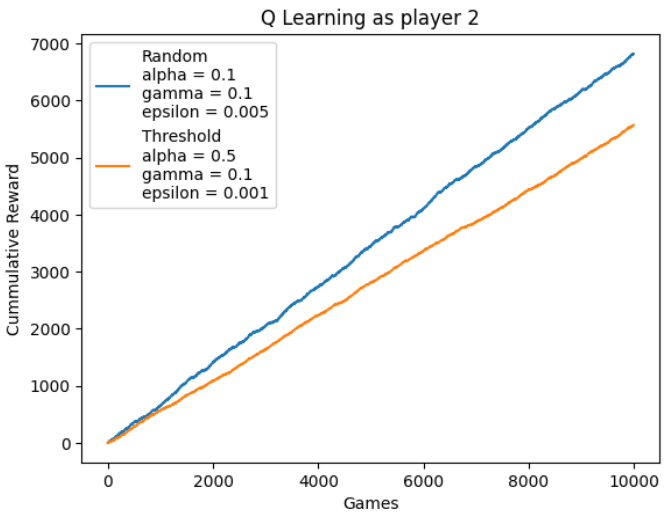


When playing as player 2 against a Threshold opponent, the optimal epsilon seems to be 0.01. In this case, the agent prioritizes exploitation over exploration, and he relies on the learned policy and maximizes expected rewards based on the Q-values.

After determining the best parameters for each case, we compare the performance of our agent against the 2 opponents by plotting the cumulative reward and calculating the average reward.



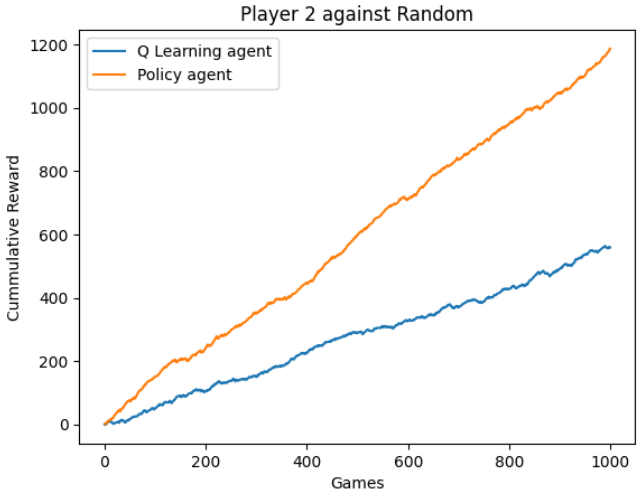
The mean rewards for player 1 are 0.3894 for playing against a random opponent and 0.49005 for playing against a threshold opponent. The cumulative reward is positive, meaning that we mostly win games, so our Q Learning algorithm definitely works. It seems that the deterministic nature of a threshold opponent enables the Q Learning agent to learn and adapt its strategy more effectively, leading to better performance in comparison to playing against a random opponent.

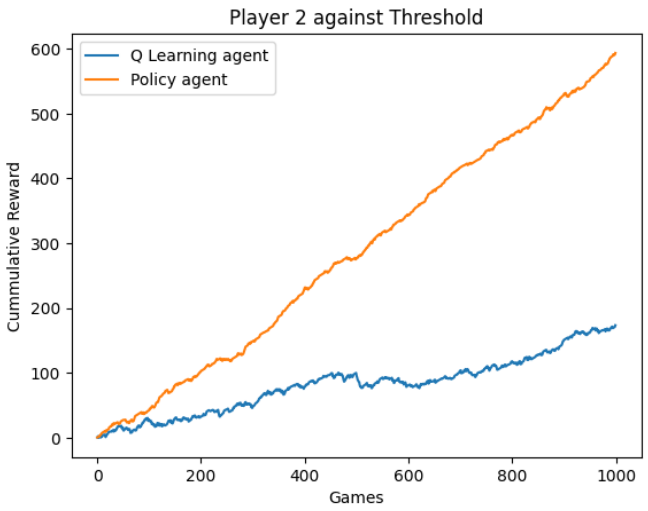


The mean rewards for player 2 are 0.6814 for playing against a random opponent and 0.5566 for playing against a threshold opponent. The cumulative reward is positive again, but now the Q Learning agent performs better against a Random opponent. This could be because of the less control he has over the game dynamics which forces him to rely more on reacting to the opponent's actions. The threshold opponent's deterministic strategy may make it harder for the agent to adapt and find optimal actions, leading to lower performance.

d) Comparison of the two algorithms:

Finally, after calculating the optimal parameters for policy iteration and q learning, we compare the performance of our 2 agents against both opponent types. Being player 1 or 2 doesn’t seem to make a difference.





It seems that Policy agent performs better than Q Learning agent. This is due to the fact that Policy Agent is tailored for the two types of opponents we made so it does not need any exploration to find the optimal solution. On the other hand, Q Learning explores and learns the every opponent through episodes of games and it's solution is sub-optimal compared to the one derived from Policy Agent.