CS7641 Assignment 4: Markov Decision Process

Jack Chai Jie Feng
jchai41@gatech.edu

FROZEN LAKE

Frozen lake is a grid world problem where an agent starts at position [0, 0] (top right hand corner) and needs to make its way to the goal at [n, n] where the agent will win and the game ends. Other tiles are either floor or hole tiles and if the agent steps into a hole, it loses and the game ends. The grid world generated guarantees a path from [0, 0] to [n, n] so it is always possible for the agent to win.

The agent can move in 4 directions: up, down, left and right. However each step has a ⅓ chance of moving in each perpendicular step (i.e. if the agent decides to move up, it has ⅓ chance of moving up, ⅓ chance of moving left and ⅓ chance of moving right). If the agent is at the edge of the grid world, a step towards the wall will result in it returning to its position. A 10x10 grid (figure 1.1) is generated for this experiment, as such, the agent can explore up to 100 states.

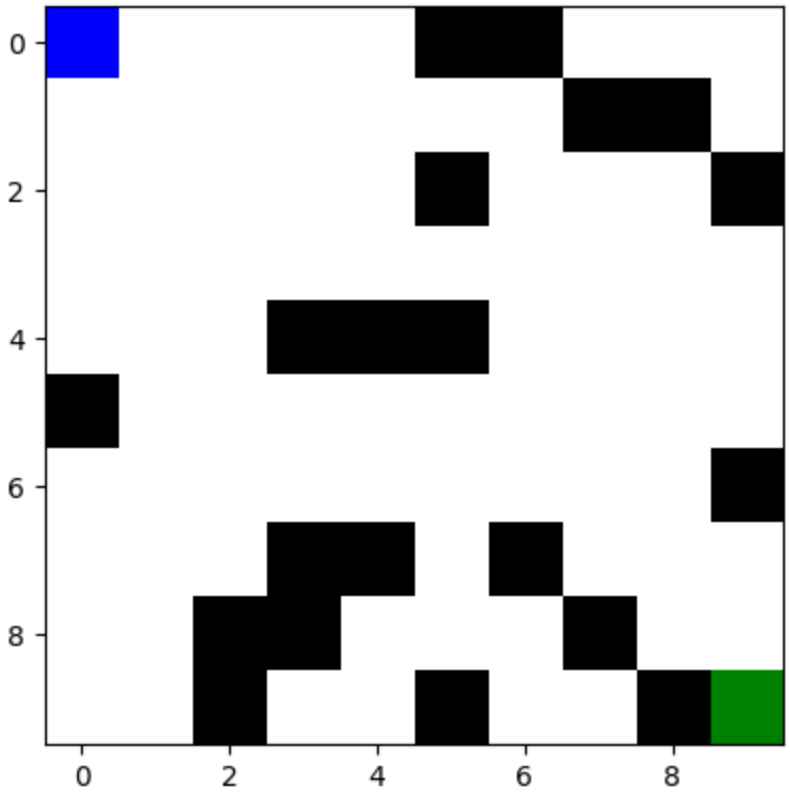

Figure 1.1: 10x10 frozen lake map

This MDP is interesting as it is a small 2d grid world that is great for visualization purposes. Each of the algorithms used below uses a discount factor of 0.99 to encourage the agent to find the shortest path to the goal without falling into any hole.

VALUE ITERATION

Value iteration calculates the value of each state iteratively by accounting for the best action in the current state and the future expected value of the next state. The initial values used in this experiment are zeros and a large enough iteration will allow the algorithm to converge to the optimal values. The converge criteria is when the absolute value difference between iterations is less than theta (1e-10). Our experiment shows that the algorithm took 975 iterations and 0.76 seconds to convergence.

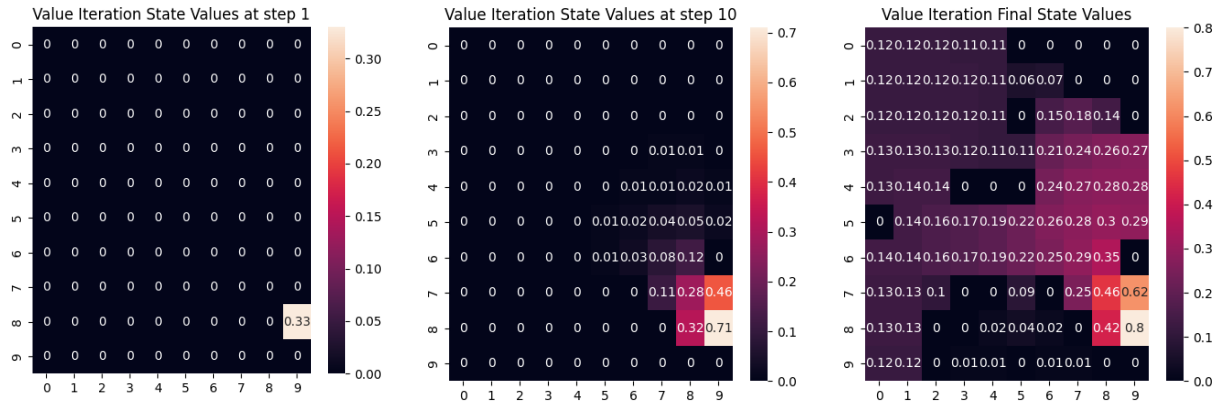

Figure 1.2: Value iteration value states at different iteration

POLICY ITERATION

Oftentimes, the policy converges long before the value converges. Policy iteration takes advantage of this by splitting each iteration into 2 steps: policy evaluation and policy improvement. In the policy evaluation step, the value function of the current policy is iteratively updated until convergence, similar to the value iteration process. In the policy improvement step, a new policy is greedily constructed based on the current value function by selecting actions that maximize the expected return. Policy iteration iterates between improving the policy and evaluating its value until the policy converges to the optimal policy. Our experiment shows that the algorithm took 14 iterations and 0.99 seconds to convergence.

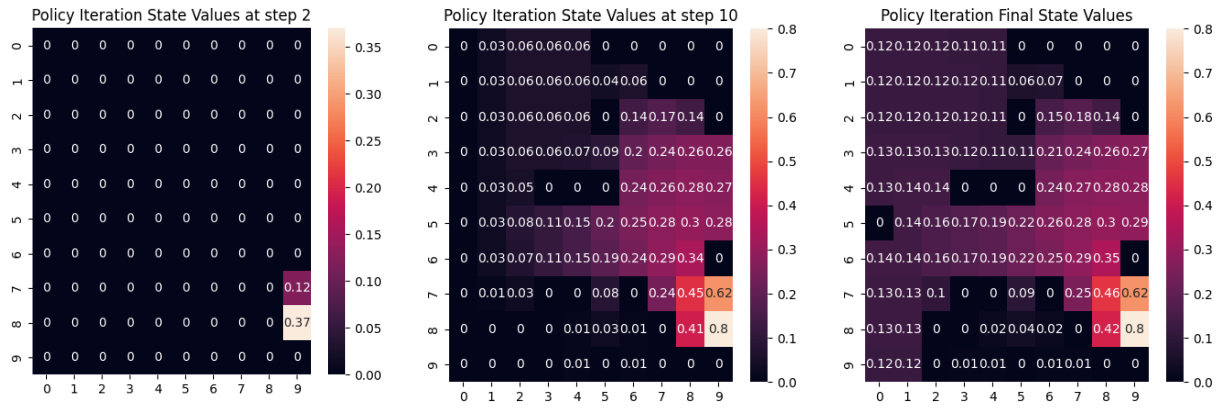

Figure 1.3: Policy iteration value states at different iteration

Q LEARNING

Q learning assumes that reward and transition probabilities are not known and the agent will learn these values by interacting with the environment. The agent will learn Q values where it represents the expected cumulative reward obtained by taking an action in the current state and then following the optimal policy thereafter. Since the Q values are unknown in the beginning, the agent is more inclined to take random actions (exploration phase) and will slowly transition to taking most optimal actions (exploitation phase) as it interacts with the environment more. More formally, we set the probability of taking a random step (epsilon) as an exponential decay schedule of 0.9 over 2,000,000 episodes and it took 2129.19 seconds.

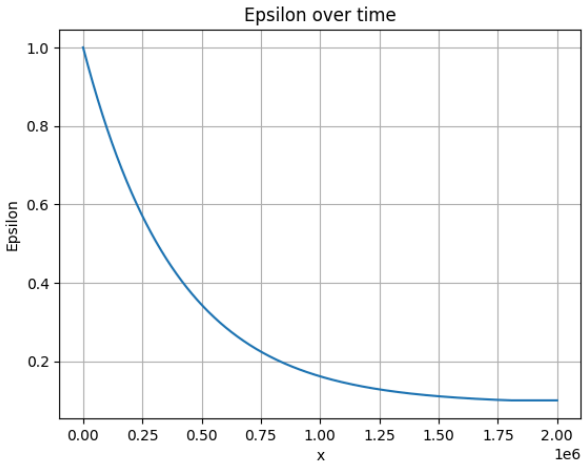

Figure 1.4.1: Epsilon over episodes

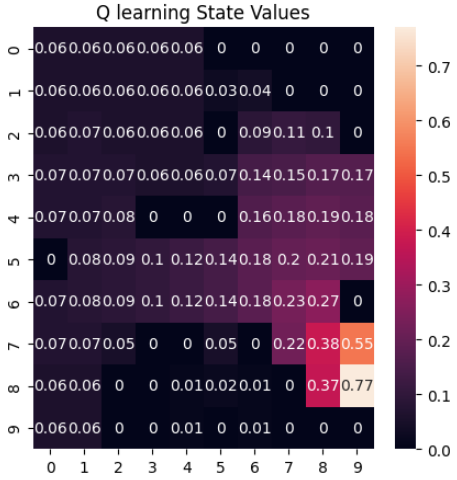

Figure 1.4.2: Q learning value states at final iteration

ANALYSIS

While value iteration took more iterations than policy iteration to converge (975 vs 14), the time taken for value iteration is less than policy iteration (0.76s vs 0.99s). This is due to more computation complexity in each policy iteration. Both figures 1.2 and 1.3 show that the values are spreading from the goal tile over iterations and eventually converge to the same state value.

Comparing the 2 iteration algorithms against the q learning algorithm is not so straightforward. We can see that the final state value is quite similar to the 2 iteration algorithms but not exactly the same. Another method to measure the outcome similarity is to check the similarity between the policies by using the number of same actions as numerator and total number of states as denominator. We found that q learning and policy iteration has 0.85 similarity, where there are 2 regions where subtle difference in action (left and down) will result in the same outcome illustrated in figure 1.5.

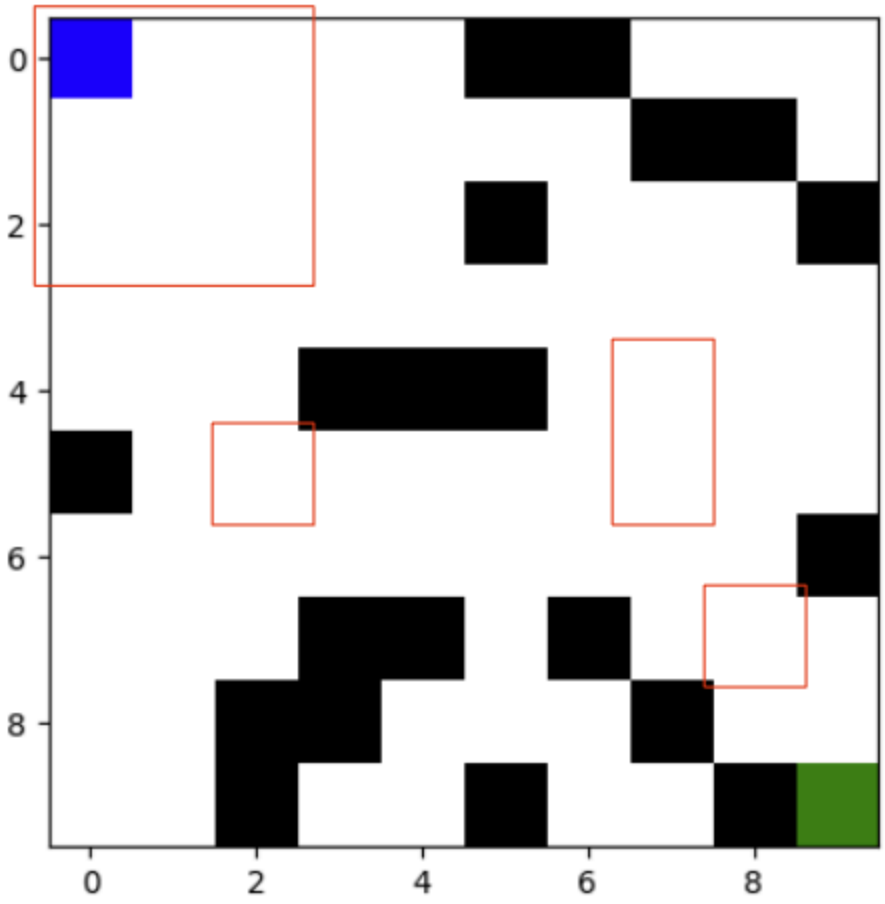

Figure 1.5: Tiles where both left and down are the most optimal actions

BLACKJACK

Blackjack is a card game where the goal is to beat the dealer by obtaining cards that sum to closer to 21 (without going over 21) than the dealer's cards. Both the agent and the dealer are dealt 2 cards and the agent knows both of its card and 1 of the dealer's card. The agent can decide to hit or stick. Hit will result in the agent drawing a card and stick is when the agent decides to stop drawing. After sticking, the dealer will reveal both its cards and draw cards until it has at least 17 points and the game ends with either win, lose or draw.

There are 13 types of cards in a deck, namely 2-10, Jack, Queen, King, Ace. Face cards (Jack, Queen, King) have a point value of 10. Aces can either count as 11 (called a 'usable ace') or 1. Numerical cards (2-9) have a value equal to their number. At any point of the game, the agent is aware of observation: its current value, the dealer's single faced up card value, and whether it has a usable ace or not. The number of states is calculated as such: usable ace, also known as 'soft' (12-21); not usable ace, also known as 'hard' (4-21); and blackjack gives us 29 states. The single dealer face up card has 10 states so the total number of observable states is 290.

This is a simplified version of blackjack. In the real world, gamblers use a technique called counting cards to outsmart the dealer. Counting cards involves estimating the number of big and small cards left in the deck so gamblers can take advantage of this estimated knowledge to increase their bet sizes. If we were to map counting cards as a state, we can easily imagine the state space blowing up. However, it is not applicable in experiment as this simple environment does not account for removing dealt cards. Nonetheless, this is still an interesting problem because unlike the frozen lake problem described above, there are many more unknowns throughout each game, from the dealer's face down card to the different cards that both the agent and dealer will hit. As each game is relatively short and there is no urgency to end the game faster, no discounting factor is used in the below algorithms for this MDP problem.

SInce the blackjack environment is designed to have different game play each time (unlike frozen lake where the grid world is generated and used repeatedly), we will also measure the number of times of wins, losses and draws over 1000 games after training.

VALUE ITERATION

Our experiment shows that the algorithm took 12 iterations and 0.03 seconds to convergence. Over 1000 games, the player lost 462 times, drew 107 times and won 431 times with 33 natural blackjacks.

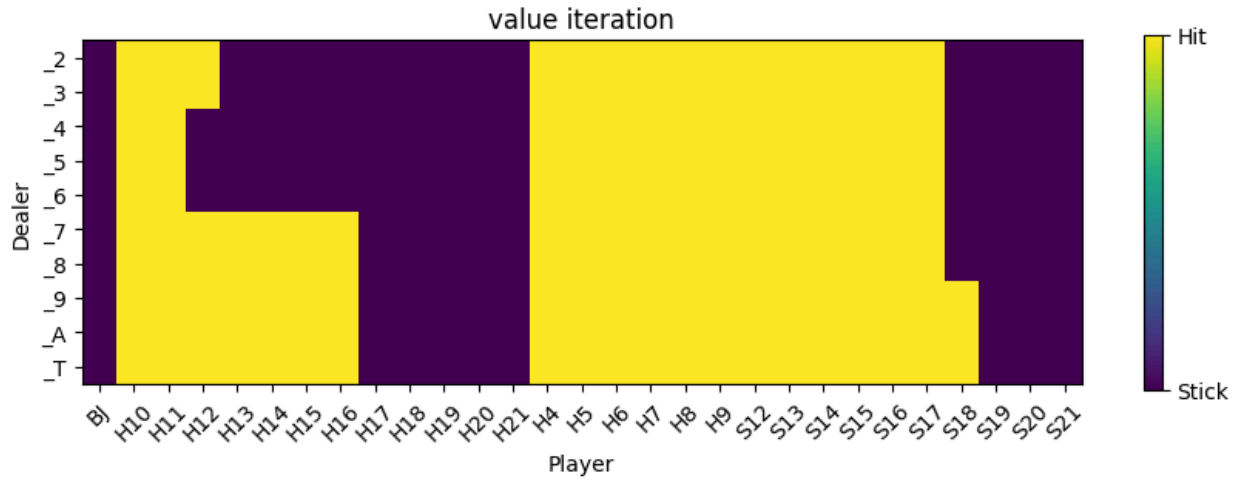

Figure 2.1: Value iteration final policy grid

POLICY ITERATION

Our experiment shows that the algorithm took 4 iterations and 0.05 seconds to convergence. Over 1000 games, the player lost 475 times, drew 89 times and won 436 times with 44 natural blackjacks.

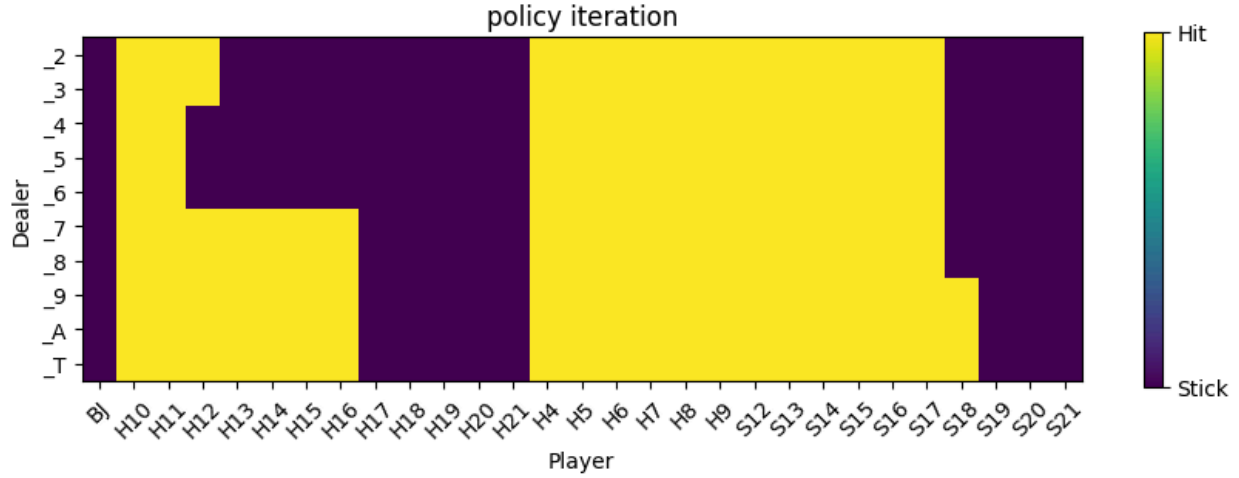

Figure 2.2: Policy iteration final policy grid

Q LEARNING

Our experiment consists of 4,000,000 training episodes and took 463.43 seconds to complete. Over 1000 games, the player lost 458 times, drew 90 times and won 452 times with 56 natural blackjacks.

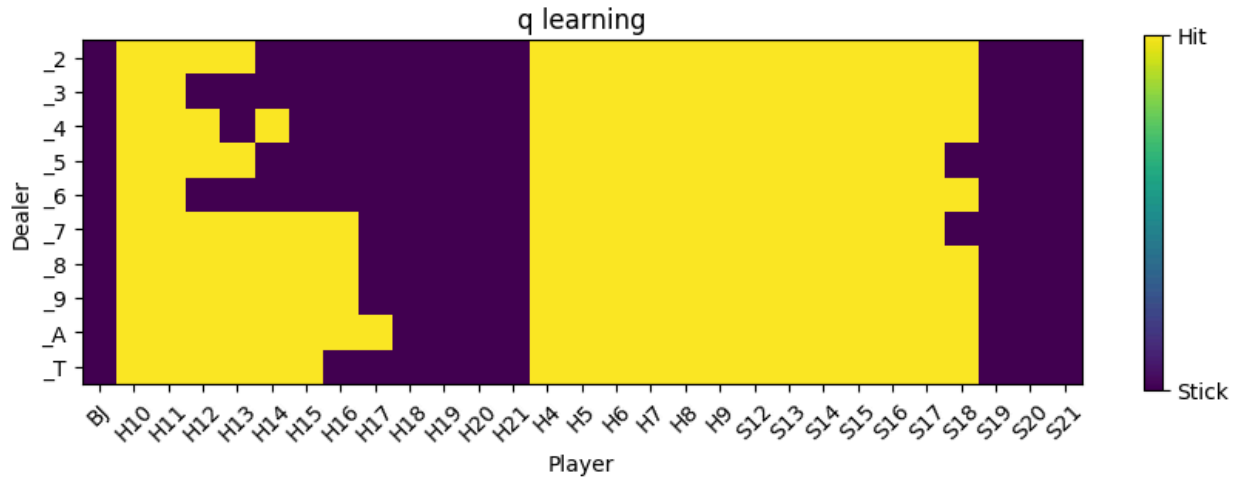

Figure 2.3: Q learning final policy grid

ANALYSIS

Similar to frozen lake MDP, value iteration took more iterations and yet less training time to converge (12 vs 4 iterations and 0.03s vs 0.05s). To measure the similarity of both algorithms, both converge to the same final state value and therefore policy. Figures 2.1 and 2.2 illustrate that the same policy is learnt from both iteration algorithms.

We use the same method of comparison against the q learning algorithm. We can see that the final policy is quite similar to the 2 iteration algorithms but not exactly the same. We found that q learning and policy iteration have 0.96 similarity. Figure 2.3 shows the subtle difference in actions around the low dealer (2-6) and hard mid player points (12-14) region.

All algorithms have very similar performance (with accepted deviation) of ~45% losses, ~45% wins and ~10% draws. The game of blackjack consists of too many random variables and the state given by the environment does not have sufficient information for our learners. A possible way to improve our learners' abilities is to consider adding more observable states to the learner such as counting cards. However, this is not explored in this experiment due to the simplification of the environment where the dealt cards are being tracked in the deck.