Reinforcement Learning Blackjack Agent

CS6330 – Project 1

By Jack Schlederer

March 27, 2022

Abstract

The goal of this first project was to build a simplified Blackjack game with a player whose decisions are governed by the policy output from a reinforcement learning algorithm. Each of three experiments ran 1,000 Blackjack hands against a dealer who hits up to 17, then stands. The "house edge", or the percentage of the games the dealer wins minus 50%, was captured for each experiment and serves as the primary success metric for the project.

Experiment 1 – Hand-crafted policy

For the first experiment, a "policy" for the automated player was built manually. For the purposes of these experiments, a policy is simply a two-column lookup table that associates an action (hit or stand) with a game state, which for Blackjack is the sum of the cards in the player's hand. Convention Blackjack strategy recommends, in most situations, to hit up until 17 and then stand, so this became the policy for the first experiment, as shown in Figure 1.

Figure 1: Policy for first experiment.

State	Action
<17	Hit
≥17	Stand

This resulted in the player winning 424 games and the dealer winning the remaining 576, the shown in Figure 2, for a house edge of 7.6%.

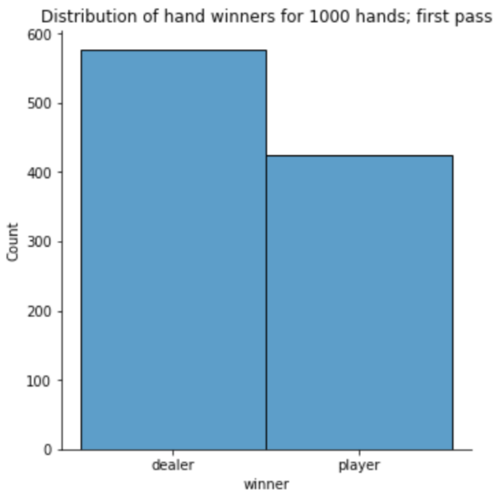

Figure 2

This is a modest baseline based on conventional strategy. The following two experiments implement different q-learning algorithms for training the player's policy to attempt to improve the house edge for the player.

Experiment 2 – Q-learning algorithm I

For the second experiment, a reinforcement learning algorithm was written to incrementally update a Q-table over the course of 10,000 hands. For these experiments, a Q-table is represented by a 3-column lookup table associating every possible state/action pair with a q, or quality, value. The quality values are updated each hand, depending on the result of the hand (whether the player won or lost) and an associated rewards table, which is a 4-column lookup table representing the "reward" for each combination of the current state, an action, and the state the action resulted in. For each algorithm the learning algorithm took, the reward was looked up in the rewards table, and a new q value for the state/action pair was calculated using the following formula:

$$q[s, a] = (1 - \text{alpha}) * q[s, a] + \text{alpha} * (r + \text{lambda} * q[s', a'])$$

… where:

- q[s,a] is the current q value of the state/action pair
- alpha and lambda are constants (both set to 0.1)
- q[s', a'] is the q value of the action paired with the result state which has the highest q value

For the second experiment, the dealer went first and the player went second, so it wasn't possible to declare the player the winner unless they hit 21 or if the dealer busted, so the rewards table rewarded the hand value for player hand states of 1-20, 1000 for 21, and -1000 for player busts. After training the Q-table with 10,000 hands, the policy shown in Figure 3 was compiled using the action for each state that had the highest q value.

Figure 3: Policy for the second experiment.

State	Action
2, 4-11	Hit
≥12	Stand

1,000 hands were then run against the dealer (whose results are shown in Figure 4), resulting in a house edge of 10.2%.

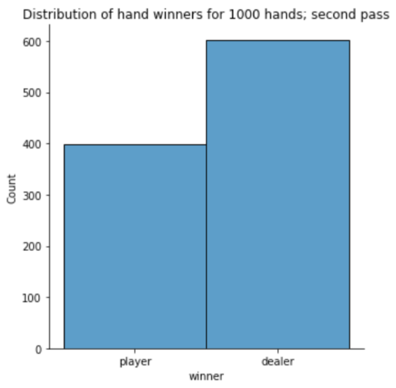

Figure 4

This is a slightly worse house edge than the first experiment, but it also fails to account for hands in which the player wins (i.e. their hand total exceeds the dealers without busting) but does not hit 21. Experiment 3 covers that case.

Experiment 3 – Q-learning algorithm II

For the last experiment, much of the Q-learning algorithm was kept the same with one key difference. This time around, the dealer plays their hand first, which allows their hand total to be added to the rewards table. This way, a reward can be calculated for each player hand-dealer hand-state combination, meaning that the player's hands can be rewarded for beating the dealer while not necessarily achieving 21 exactly. For example, a player hand of 20, a dealer hand of 18, and the "stand" action would have a reward of 1000, since that action would result in the player winning. The policy produced by this new algorithm is shown in Figure 5.

Figure 5: Policy for the third experiment.

State	Action
1, 3-18	Hit
19-21	Stand

This policy, when run against the same 1,000 hand experiment, whose results are captured in Figure 6, yielded a house edge of 13.8%, slightly worse than that of the second experiment.

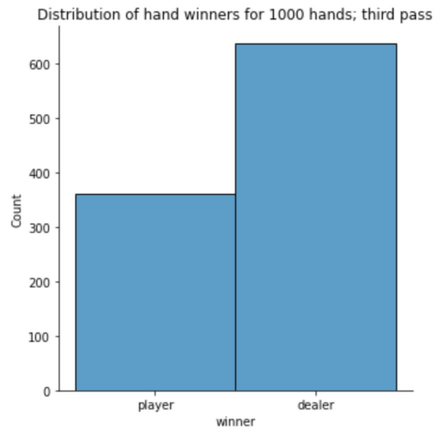

Figure 6

Potential improvements

While it seems that the Q-learning policies did slightly worse at the game than the hand-crafted policy, they still performed quite well in going from performing blind, random actions to a degenerate gambler who can *almost* play Blackjack! In the future, it would be interesting to see if a better policy could be generated by using a grid-search of the lambda and alpha hyperparameters to optimize their values. Furthermore, improvements to the game itself would be interesting enhancements to the algorithm, such as giving the player the option to split, double-down, and take into account the dealer’s up-card, all of which would be present in a standard Vegas Blackjack game.