Let's ~~*Gamble*~~ *Maximize our Win Percentage!*

Maxwell McKee

mmckee30@gatech.edu

Group members: Maxwell McKee

ISYE 6644 Project – Blackjack Simulation

Abstract

This project showcases the process and results of a Blackjack simulation implemented in python. The first strategy being evaluated is "standing" at various values operating under either a "hard" or "soft" approach. The end result is deciding what stand value and what hand/soft approach will make this strategy most effective. The best candidate parameters of this strategy are then compared against an implementation of the "basic" strategy. The basic strategy refers to a set of rules that help a player make the best possible decision based on their hand and the dealer's "upcard". The consensus basic strategy shifts depending on conditions such as the dealer's rules, betting rules, the number of decks used, and which special rules are allowed. Of the strategies tested, the most effective was a hard hit on 18 with a winning percentage of ~55% against the dealer. The results are easy to visualize, and the code encourages the user to change conditions and compare outcomes.

Background and Problem Statement

We look to determine which of our tested strategies are most effective on a per-game basis. First, we want to know whether a soft strategy (treat Aces as 11 points when deciding to hit or stand) or a hard strategy (treat Aces as 1 points when deciding to hit or stand) is more/less effective, or if it really matters. Next, we simulate which threshold value to stand at is most effective. Finally, we select the optimal strategy from the process above and play it against a typical basic strategy consisting of known rules for optimal Blackjack gameplay.

The desired outcome of this simulation was to decisively conclude which strategy (of the ones implemented) has the highest win percentage. This means that results should be consistent when certain starting conditions are held constant, but with different deck shuffles each trial. This also requires implementing the code such that it is practical to run enough trials to produce consistent results.

A couple things that are important to note: (1) The simulation uses a consistent strategy for the dealer – the dealer will hit as long as their hand value is less than 17, with Aces counting at 11 points until the value of the hand is evaluated at the end of the round (2) This simulation does not incorporate betting amounts – it focuses on winning independent games, so the basic strategy uses rules unrelated to a player's bets.

The rules of Blackjack, specifically in the context of this simulation, will be described next. A deeper dive into the simulation will follow: What is coded into the simulation? What parameters can be changed? How do you use the program? What are the program results?

Rules and Strategies

The primary goal of Blackjack is to have a hand value closer to 21 than the dealer's hand without exceeding 21. Players initially receive two cards, and they can choose to "hit" to receive additional cards or "stand" to maintain their current hand value. Cards are worth their face value, with the exception of the Ace counting as either 1 or 11 points, whichever proves to be more beneficial. The dealer's moves are predetermined, hitting until they reach a hand value of at least 17. The player has more options such as "doubling down" and "splitting". These involve actions taken on a player's bets, but this simulation focuses on maximizing wins on a per-game basis.

A simple Blackjack strategy is to stand when your hand value reaches a certain threshold. The simulation provides results on the best value to stand at. Additionally, the simulation results help determine whether to count Aces as 11 points or as 1 point when evaluating whether to hit or stand. If a player has at least one Ace in their hand and count it as 11 points - this is a "soft" strategy, otherwise it is a "hard" strategy.

The details of this simulation's implementation of the basic strategy are detailed more later.

Simulation and Main Findings

Parameters and Code

When running the simulation the following inputs that can be changed are: seed, trials, number of decks, strategies, when to stand, omit first *x* trials, and some chart visualization options.

The purpose of *seed* is two-fold: Be able to be able to replicate results and randomize the deck shuffles. The results will change for the same seed if the other conditions are altered. With the delightful packages we have access to, using a seed for these purposes is so easy that even a University of Georgia student could implement it. Although, in the spirit of Simulation 6644, the seed feeds into a linear congruential generator class. The class makes it so that one input seed can iteratively generate many (depends on period of LCG) new pseudorandom seeds. These seeds determine the shuffle order of each deck for a hand of Blackjack. The user can change the LCG used, but by default it is the below equation, which is the same used by the POSIX rand() function.

$$(110351545 * seed + 12345) mod(2^{32})$$

The user species how many games of Blackjack should be played for each strategy using the *trials* parameter. The *number of decks* parameter is trivial.

Both the *strategies* and *when to stand* parameters are list types. The function and interaction of these parameters is best explained with an example. Suppose we use a list of two strategies: *[strategy_soft, strategy_hard]*; and a *when to stand* array: *[15, 17]*. There will 4 different resulting simulations (*soft_hit_on_15, soft_hit_on_17, hard_hit_on_15, hard_hit_on_17*) as the number of simulations is the cross product of the two lists.

The *omit first x trials* parameter just allows the visualization to not include the results of the first *x* trials. The visualization has trials on the x-axis and win/loss ratio on the y-axis. For each hand of Blackjack, a win is a 1, a tie is a 0, and a loss is a -1. The chart shows how the cumulative average of these results changes over the course of the number of trials. So, not including the first *x* trials helps scale the axes better by removing noise caused by low iterations.

How to Use It

The code is intended to be user friendly. There is a main code *blackjack_final.ipynb* python file and a *classes.py*.

The *classes.py* file has a class appropriately and uneventfully named *simulate_round* that makes an instance of a simulation given a shuffled deck, a player hand, and a dealer hand. The strategy functions exist within this class. To make a new strategy, the user can write a new strategy function within this class. The other class is *LCG* which allows for a specified linear congruential generator function and can spit out the next pseudorandom seed.

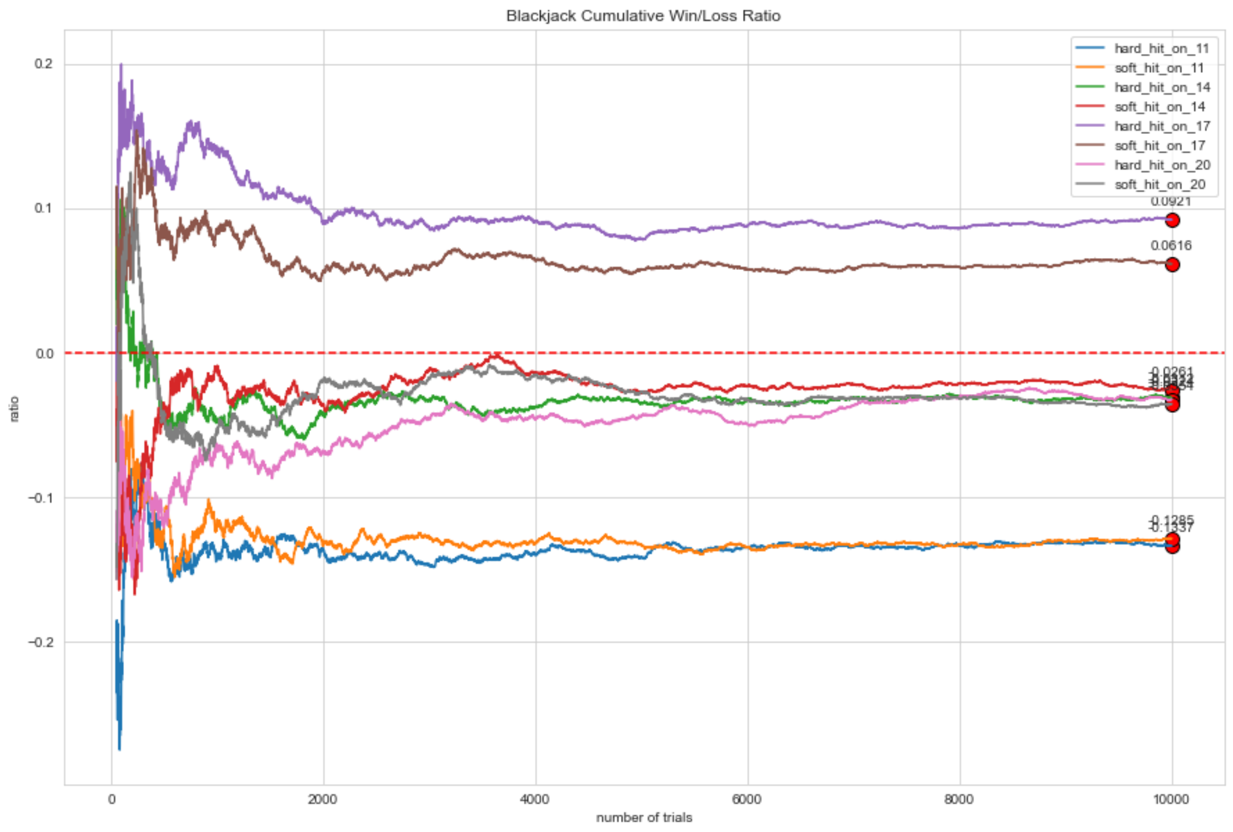
The *blackjack_final.ipynb* file has a bunch of functions necessary for executing the objective and keeping the user input portion approachable. To run a simulation of various strategies and visualize the results just change the values of the input dictionary, pass it into a function that creates a simulation instance under each combination of conditions, then pass both into the visualization function.

Results

First, we compared whether using a soft hit or a hard hit strategy makes a difference, holding the *when to stand* parameter constant. Is one consistently better than the other? Two results under different seeds are shown below. The input conditions are also shown.

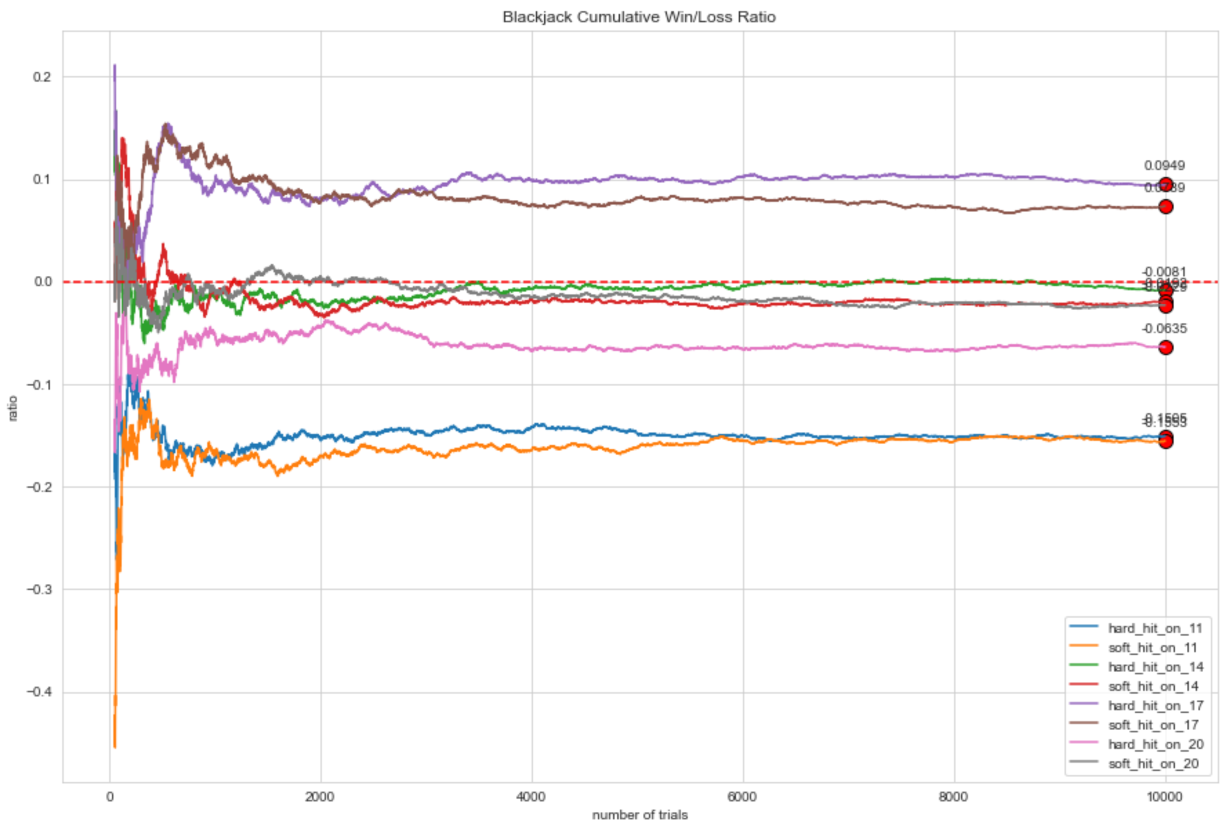
```
conditions_dict = {
    'seed':686,
    'trials':10000,
    'number of decks':4,
    'strategies':['strategy_hard', 'strategy_soft'],
    'when to stand':[11,14,17,20],
    'chart title':'Blackjack Cumulative Win/Loss Ratio',
    'y-axis title':'ratio',
    'x-axis title':'number of trials',
    'chart style':'whitegrid',
    'figsize':(15, 10),
    'omit first x trials':50
}
sim_dict = niceWatch_runIt(conditions_dict)
line_chart(conditions_dict, sim_dict)
```

```
hard_hit_on_17: Based on the results of 10000 games, you can expect to win 54.6% of the time.
soft_hit_on_17: Based on the results of 10000 games, you can expect to win 53.0% of the time.
soft_hit_on_14: Based on the results of 10000 games, you can expect to win 48.6% of the time.
hard_hit_on_14: Based on the results of 10000 games, you can expect to win 48.4% of the time.
hard_hit_on_20: Based on the results of 10000 games, you can expect to win 48.3% of the time.
soft_hit_on_20: Based on the results of 10000 games, you can expect to win 48.2% of the time.
soft_hit_on_11: Based on the results of 10000 games, you can expect to win 43.5% of the time.
hard_hit_on_11: Based on the results of 10000 games, you can expect to win 43.3% of the time.
```



```
conditions_dict = {
    'seed':747,
    'trials':10000,
    'number of decks':4,
    'strategies':['strategy_hard', 'strategy_soft'],
    'when to stand':[11,14,17,20],
    'chart title':'Blackjack Cumulative Win/Loss Ratio',
    'y-axis title':'ratio',
    'x-axis title':'number of trials',
    'chart style':'whitegrid',
    'figsize':(15, 10),
    'omit first x trials':50
}
sim_dict = niceWatch_runIt(conditions_dict)
line_chart(conditions_dict, sim_dict)
```

```
hard_hit_on_17: Based on the results of 10000 games, you can expect to win 54.7% of the time.
soft_hit_on_17: Based on the results of 10000 games, you can expect to win 53.6% of the time.
hard_hit_on_14: Based on the results of 10000 games, you can expect to win 49.5% of the time.
soft_hit_on_14: Based on the results of 10000 games, you can expect to win 49.0% of the time.
soft_hit_on_20: Based on the results of 10000 games, you can expect to win 48.8% of the time.
hard_hit_on_20: Based on the results of 10000 games, you can expect to win 46.8% of the time.
hard_hit_on_11: Based on the results of 10000 games, you can expect to win 42.4% of the time.
soft_hit_on_11: Based on the results of 10000 games, you can expect to win 42.2% of the time.
```



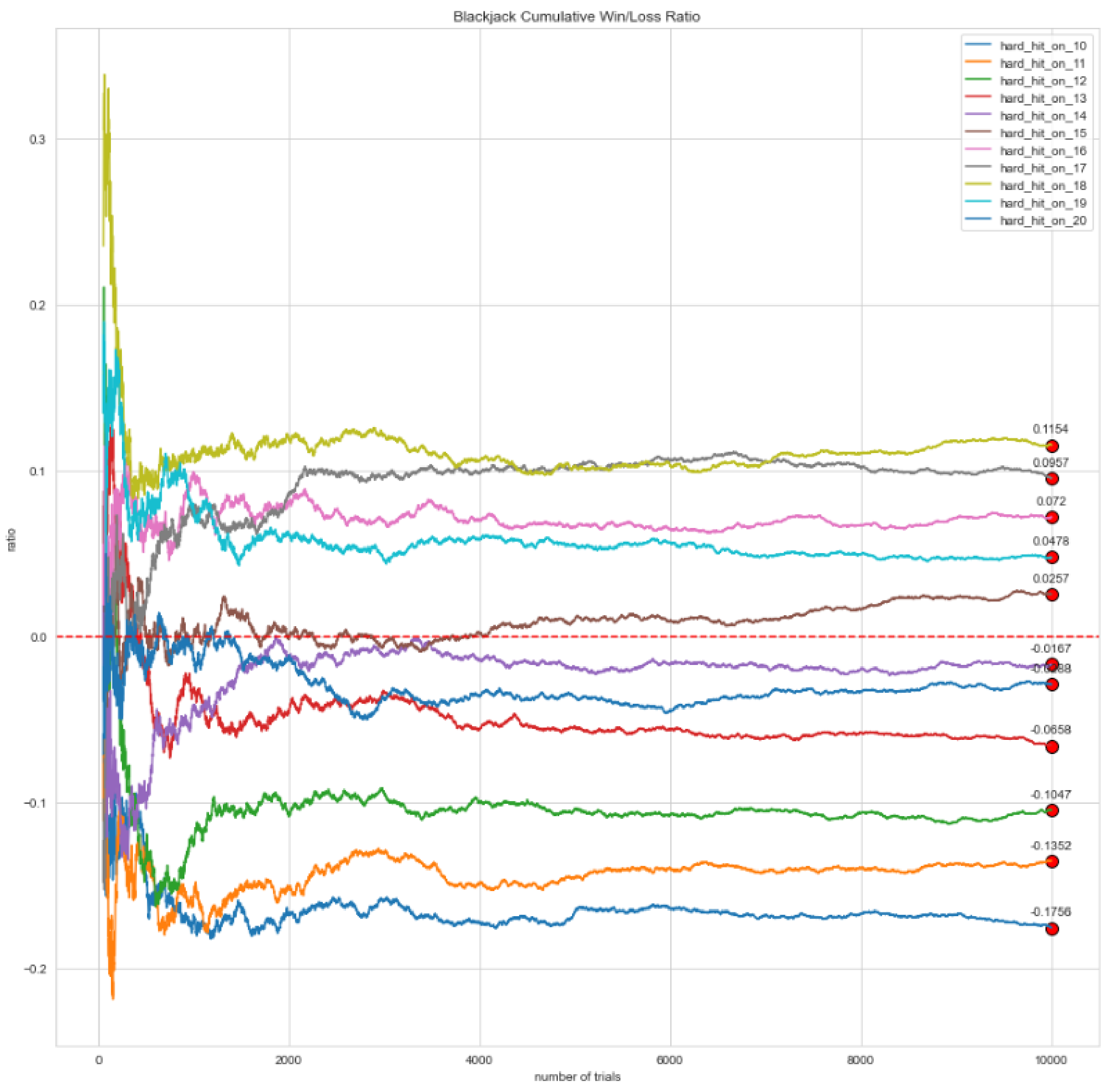
These charts indicate that the *when to stand* threshold is definitely important as these thresholds are consistently in the same order when ordered by win percentage. Although, whether the strategy is soft or hard does not seem to make much of a difference - the win/loss ratios are always comparable and which one wins out changes across different simulation seeds, even with 10,000 trials.

With an optimal *when to stand* threshold, it looks like a hard hit may marginally be more effective. So, we proceed to finding a strategy that will compete against the basic strategy knowing we are slightly more likely to be successful under a hard hitting strategy.

Next, we want to figure out which *when to stand* hard hitting threshold is most effective.

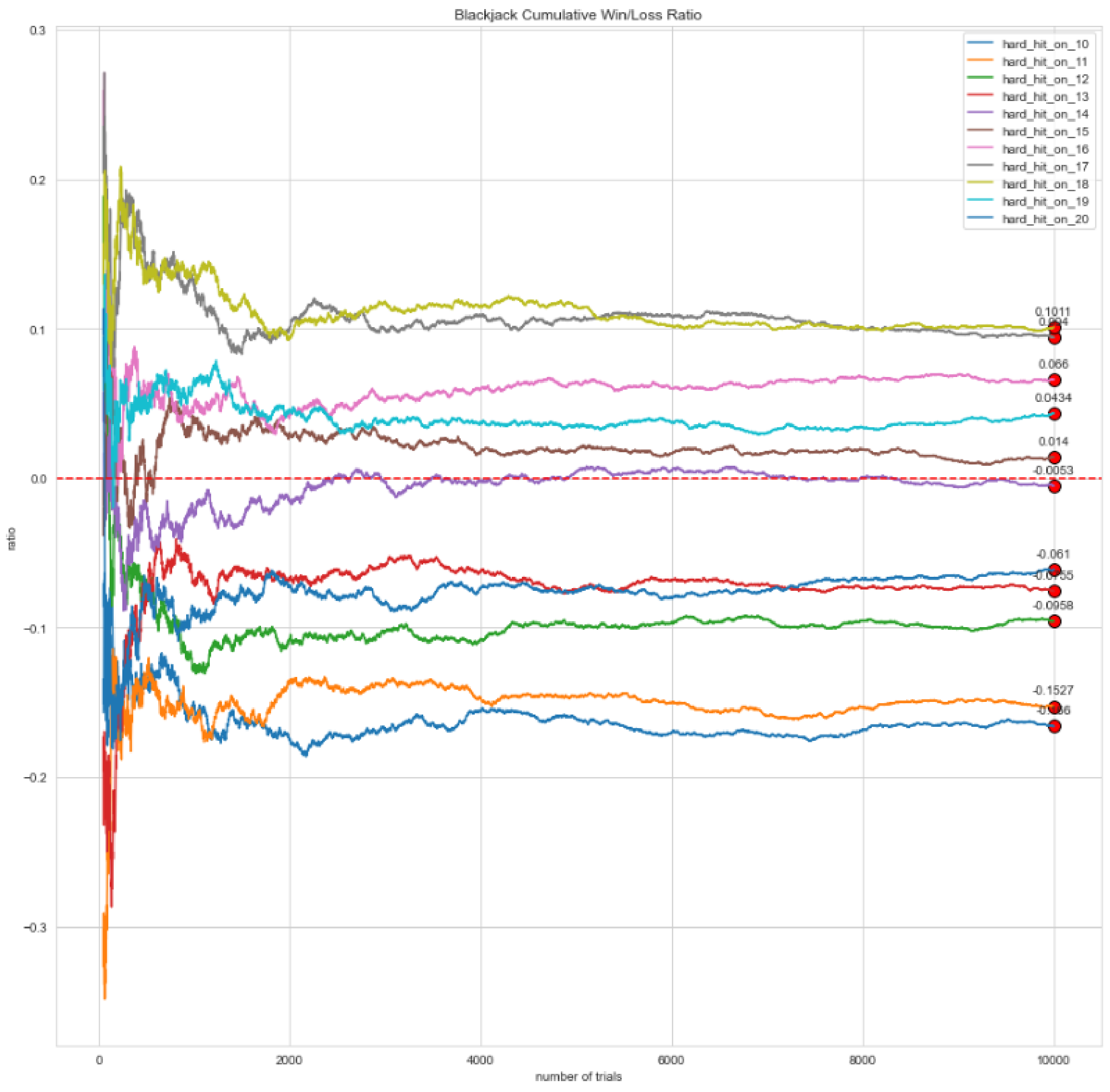
```
conditions_dict = {
    'seed':90210,
    'trials':10000,
    'number of decks':4,
    'strategies':['strategy_hard'],
    'when to stand':list(range(10, 21, 1)),
    'chart title':'Blackjack Cumulative Win/Loss Ratio',
    'y-axis title':'ratio',
    'x-axis title':'number of trials',
    'chart style':'whitegrid',
    'figsize':(15, 15),
    'omit first x trials':50
}
sim_dict = niceWatch_runIt(conditions_dict)
line_chart(conditions_dict, sim_dict)
```

```
hard_hit_on_18: Based on the results of 10000 games, you can expect to win 55.7% of the time.
hard_hit_on_17: Based on the results of 10000 games, you can expect to win 54.7% of the time.
hard_hit_on_16: Based on the results of 10000 games, you can expect to win 53.6% of the time.
hard_hit_on_19: Based on the results of 10000 games, you can expect to win 52.3% of the time.
hard_hit_on_15: Based on the results of 10000 games, you can expect to win 51.2% of the time.
hard_hit_on_14: Based on the results of 10000 games, you can expect to win 49.1% of the time.
hard_hit_on_20: Based on the results of 10000 games, you can expect to win 48.5% of the time.
hard_hit_on_13: Based on the results of 10000 games, you can expect to win 46.7% of the time.
hard_hit_on_12: Based on the results of 10000 games, you can expect to win 44.7% of the time.
hard_hit_on_11: Based on the results of 10000 games, you can expect to win 43.2% of the time.
hard_hit_on_10: Based on the results of 10000 games, you can expect to win 41.2% of the time.
```



```
conditions_dict = {
    'seed':9000,
    'trials':10000,
    'number of decks':4,
    'strategies':['strategy_hard'],
    'when to stand':list(range(10, 21, 1)),
    'chart title':'Blackjack Cumulative Win/Loss Ratio',
    'y-axis title':'ratio',
    'x-axis title':'number of trials',
    'chart style':'whitegrid',
    'figsize':(15, 15),
    'omit first x trials':50
}
sim_dict = niceWatch_runIt(conditions_dict)
line_chart(conditions_dict, sim_dict)
```

```
hard_hit_on_18: Based on the results of 10000 games, you can expect to win 55.0% of the time.
hard_hit_on_17: Based on the results of 10000 games, you can expect to win 54.7% of the time.
hard_hit_on_16: Based on the results of 10000 games, you can expect to win 53.3% of the time.
hard_hit_on_19: Based on the results of 10000 games, you can expect to win 52.1% of the time.
hard_hit_on_15: Based on the results of 10000 games, you can expect to win 50.7% of the time.
hard_hit_on_14: Based on the results of 10000 games, you can expect to win 49.7% of the time.
hard_hit_on_20: Based on the results of 10000 games, you can expect to win 46.9% of the time.
hard_hit_on_13: Based on the results of 10000 games, you can expect to win 46.2% of the time.
hard_hit_on_12: Based on the results of 10000 games, you can expect to win 45.2% of the time.
hard_hit_on_11: Based on the results of 10000 games, you can expect to win 42.3% of the time.
hard_hit_on_10: Based on the results of 10000 games, you can expect to win 41.6% of the time.
```



A *hard_hit_on_18* seems to be the consistent winner with a win percentage over the dealer ~55% of the time.

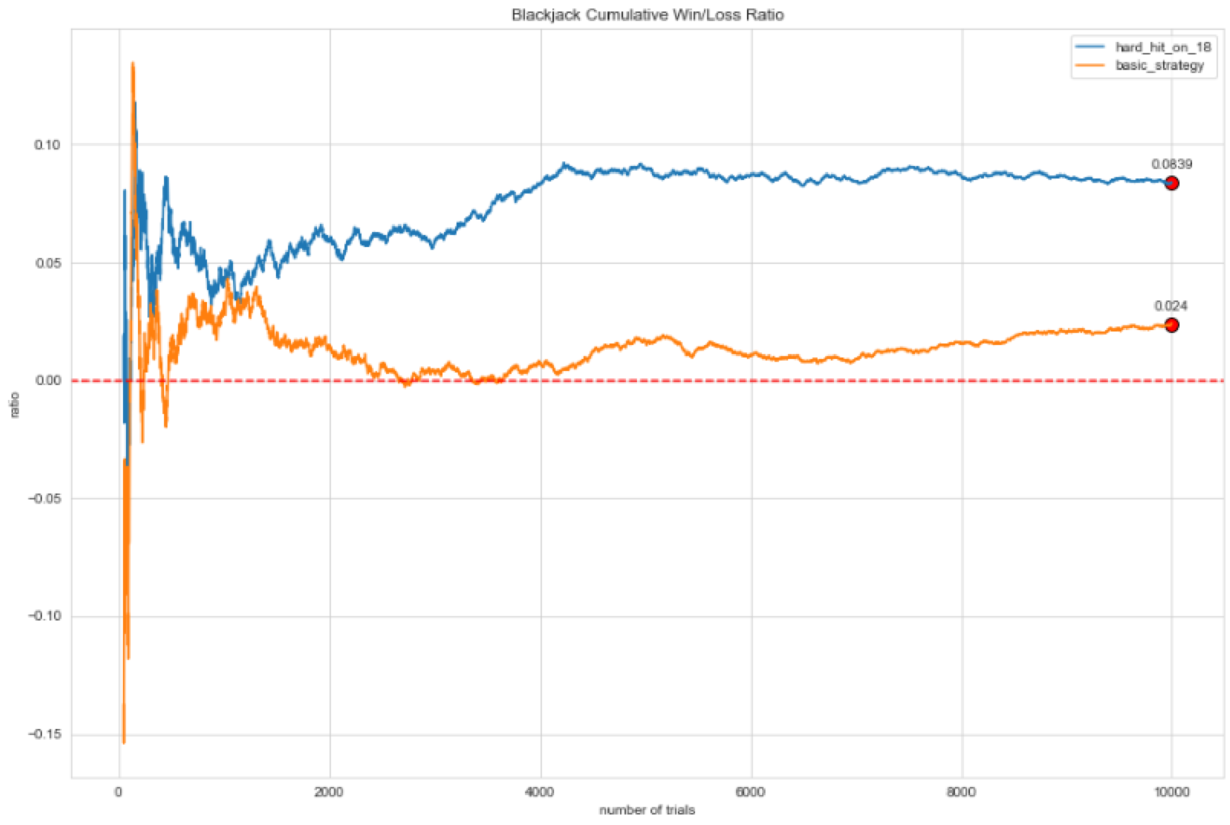
Finally, we will run our candidate simple strategy against our basic strategy. The following highlights the rules around this implementation of the basic strategy. Note that the basic strategy leverages the rule that the opposing player can see the dealer's "upcard" to make their decisions, which is the second card the dealer was dealt. For the basic strategy, the player should stand if…

1. The player has an Ace and the player's and value is between 19 and 21
2. The player has an Ace, the player hand value is 18, and the dealer's upcard value is less than 11
3. The player does not have an Ace and the player's hand value is between 17 and 21
4. The player does not have an Ace, the player's hand value is between 13 and 16, and the dealer's upcard value is between 2 and 6
5. The player does not have an Ace, the player's hand value is 12, and the dealer's upcard value is between 4 and 6

The grand reveal of this simulation is that while both strategies are marginally favorable to the dealer's set strategy, the *hard_hit_on_18 strategy* beats the *basic_strategy* about 3% more often.

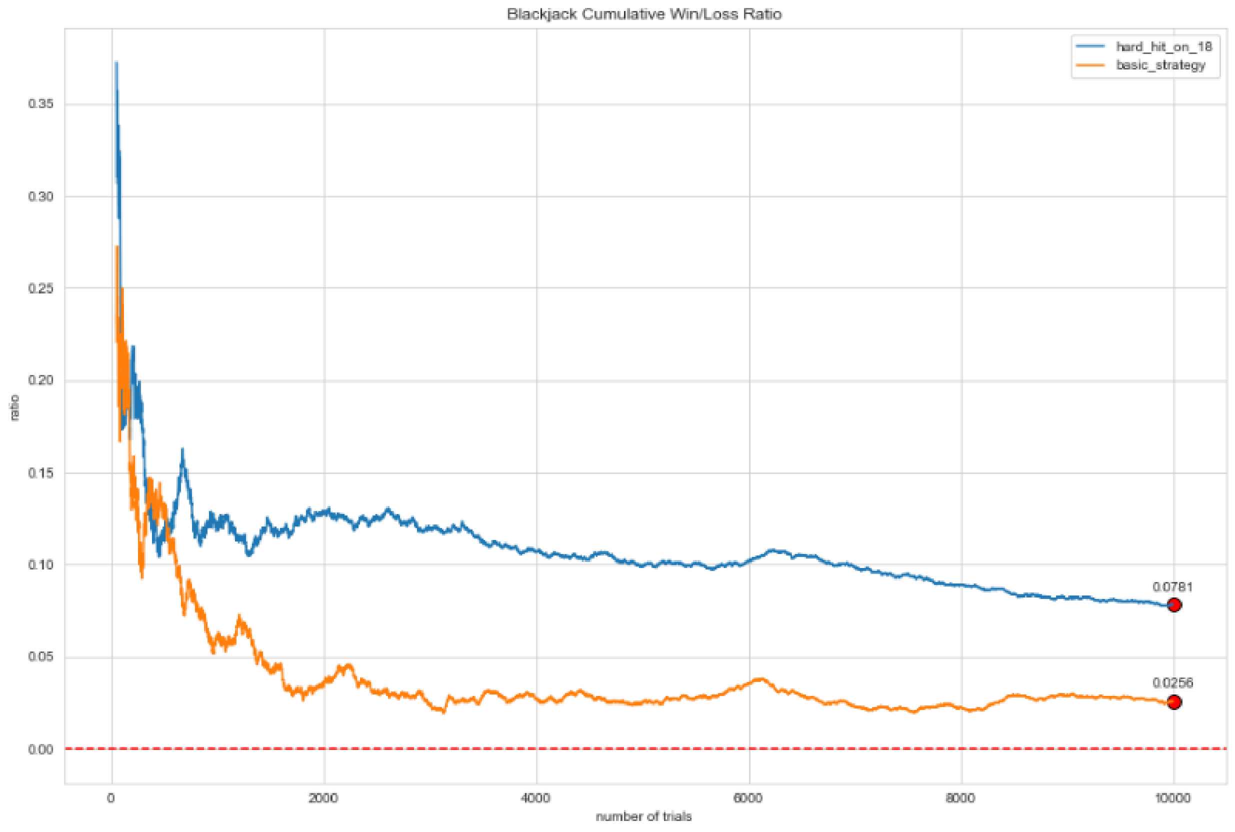
```python
conditions_dict = {
    'seed':5432,
    'trials':10000,
    'number of decks':4,
    'strategies':['strategy_hard', 'strategy_base'],
    'when to stand':[18],
    'chart title':'Blackjack Cumulative Win/Loss Ratio',
    'y-axis title':'ratio',
    'x-axis title':'number of trials',
    'chart style':'whitegrid',
    'figsize':(15, 10),
    'omit first x trials':50
}
sim_dict = niceWatch_runIt(conditions_dict)
line_chart(conditions_dict, sim_dict)
```

```
hard_hit_on_18: Based on the results of 10000 games, you can expect to win 54.1% of the time.
basic_strategy: Based on the results of 10000 games, you can expect to win 51.2% of the time.
```



```
conditions_dict = {
    'seed':9999,
    'trials':10000,
    'number of decks':4,
    'strategies':['strategy_hard', 'strategy_base'],
    'when to stand':[18],
    'chart title':'Blackjack Cumulative Win/Loss Ratio',
    'y-axis title':'ratio',
    'x-axis title':'number of trials',
    'chart style':'whitegrid',
    'figsize':(15, 10),
    'omit first x trials':50
}
sim_dict = niceWatch_runIt(conditions_dict)
line_chart(conditions_dict, sim_dict)
```

```
hard_hit_on_18: Based on the results of 10000 games, you can expect to win 53.9% of the time.
basic_strategy: Based on the results of 10000 games, you can expect to win 51.2% of the time.
```



Conclusion and Afterthoughts

Even though the strategies programmed into this simulation are relatively simple, we found out that a *hard_hit_on_18* strategy can be quite effective. I think it is worth the time for students of Simulation to take the time to make their own LCG. LCG's may make sense conceptually when learning reading a lesson about them, but implementing one provides some more clarity on how it can generate seeds, create probabilities, and randomize lists. Making the simulation conditions easy to edit made it more fun to play with since visualizing results could be made quickly – this is a good thing to note whenever making simulations for the future. My program could definitely be improved by adding a parameter for betting amounts. This would allow to much more diverse strategies and would be more realistic for learning an applicable strategy.