Gage Swenson
Dr. Edwards
Numerical Analysis
18 June 2021

Term Assignment Report
Baccarat Simulation

Baccarat is a casino card game where players bet on one of three outcomes, player, banker or tie, between two hands: the player and the banker. The game originated in Europe, and migrated into American casinos from the Carribean during colonization. The name Baccarat comes from the Italian word for “zero,” so dubbed because the 10 card and face cards all have a face value of 0. Between four to six cards are dealt each hand, beginning with two to the player and two to the banker. The goal is to get as close to 9 as possible. If the hand goes over 9, then the sum of the cards is modulo 10. The rules for whether the player draws is fairly simple. If neither hand is dealt a natural win, which is an 8 or a 9, then the player draws under 6. If the player doesn’t draw, then the banker draws below 6. If the player did draw, then there is a tedious list of rules for whether or not the banker draws the 6th card, based on the value of the last card that was dealt to the player, and the banker hand’s value. The players don’t have a say as to whether to draw another card or not. The outcome is based purely on the luck of the draw, with no skill involved. Players simply bet on either the player, banker, or tie, while the card dealer does most of the work.

A winning bet for the player is paid out 1:1. Same goes for the banker hand, except the bank gets 5% commission for a winning bet on the banker. There are also no-commission tables, where the bank pays ½:1 for a winning bet on the banker when the banker draws a 6, and no commission otherwise (Nobuhito); however, this is not the case in my simulation. A tie pays out 8:1, but it also has a House advantage of 14.36%, so it’s not a very good strategy. The player and banker hand each have a House edge of just over 1%, so if a player goes into a casino with $100, and plays 100 hands of Baccarat betting $1 on either the player or the banker every hand, they should statistically walk away with around $98.

This is all evidenced through my simulation. For the first phase of the term assignment, I laid the framework for playing a game of Baccarat in C++. There is a class for the card, which has a suit and a face value. There is a class for the deck, which is a list of cards for the top of the deck, as well as a discarded pile. There’s also a shuffle function, and a populate function which builds a shoe with *n* decks of cards. For my simulation, there are 6 decks in a shoe and 8 discrete shuffles per shuffle. The deck shuffles whenever there are fewer than 6 cards in the shoe, because a hand of Baccarat may deal up to 6 cards. There is a player class, which has a balance, a list of bets for each hand, as well as some stats for each player as the game progresses. Lastly, there is a table class, which contains a banker, a list of players, a deck, as well as the logic of playing a hand, rewarding & debiting bets, etc.

For phase one, I made it so that a user could play a game of Baccarat by setting up a table. To do this, the terminal prompts the user for the number of players, and the balances of each of those players, as well as a minimum bet. This creates an instance of the table object which keeps track of the whole game. In a game, the user may select to either view balances of the banker and the players, or play a hand. Playing a hand involves collecting a list of bets from each of the players, then dealing the cards to determine the winner. After the winner, or lack thereof, is decided, the bets are debited, rewarded, or pushed, depending on the outcome. The bets are then cleared, as well are the hands.

Phase two involved programming a simulation, to see what the effect of various betting strategies would have over the course of potentially thousands of hands. I wasn’t sure how to implement this as a class, so instead I did it in a more imperative fashion. If the user decides to run a simulation, then the user must set up a table by entering the player and banker balance, as well as an optional maximum number of hands to be played. In the simulation, there is only one player, and the minimum bet is $1, but this could become parameterized. I thought the simulation would be more clear when looking at only one player. In addition to setting up a table, the user must also choose which betting strategy to implement over the course of the game. The user must choose between one of three betting strategies: constant, variable, or random. A constant bet bets the same amount each hand. A variable bet bets *n*% of the player’s bankroll every hand. A random bet bets a random percentage of the player’s bankroll. The user must then decide which outcome to bet on. The player may bet on the player deterministically, or the banker or the tie. There are also a few other options which allow a non-deterministic bet on the possible outcomes, with varying probabilities. Once everything is decided, then the simulation will play hand after hand of Baccarat until either the player or banker goes broke, or until the maximum number of hands per simulation has been reached.

My findings indicate that betting a constant amount on every hand is the best strategy to maximize your enjoyment at a casino. Betting a constant amount on every hand may sometimes produce big wins for the player, as much as 400%, over the course of thousands of hands; of course, with no limit to the number of hands, the player will go bankrupt with virtual certainty, so long as the banker starts out with a balance sufficient enough to cover any gains made. Betting a variable amount actually proved to be a decent strategy if you're trying to maximize your number of hands, but didn't do nearly as good of a job in maximizing the player's bankroll. Upon pondering, I believe this is because as the player inevitably loses, they bet less, which earns them less, but allows them to play more hands. If the minimum bet were higher than $1 as it is at most casinos, the player would probably not fare any better betting a variable amount as opposed to a constant amount. The random bet, as expected, results in much fewer hands played, but occasionally produces huge wins for the player. The gambler's ruin principle can be demonstrated by use of the simulation. If the player places a constant bet with 10% of his initial bankroll, they will have a much higher likelihood of going bankrupt prematurely than if they had only bet 1% of their balance.

In hindsight, I wish I knew how to graph the data in an interesting way. Using the list of player's stats, one could pretty easily graph the player's balance for each hand throughout a game. A single graph wouldn't be very interesting; what would be interesting is to show thousands of graphs, but this is analogous to playing thousands of games in my simulation, which is much easier to do. What would also be interesting is to abstract the simulation portion of the code, so that thousands of simulations could run and generate statistics. This way, the theoretical outcomes of the game could be concretely verified by my program, but I simply ran out of time. Instead, I think it's sufficiently interesting to manually run dozens of games, and see that the outcomes do in fact line up quite nicely with the theoretical outcomes. All-in-all, the House always wins, and I think my simulation does an adequate job of demonstrating that.

Nobuhito, Abe. "Effects of Sequential Winning vs. Losing on Subsequent Gambling Behavior: Analysis of Empirical Data from Casino Baccarat Players." *Taylor & Francis*, 13 Sept. 2020, www.tandfonline.com/doi/full/10.1080/14459795.2020.1817969.