4Leaf Co.

FUNCTIONAL

Jordan Quick (PM)

Frances Coronel

Calvin Chambers

Anesha Passalacqua

Table of Contents

1.0 Introduction

4 Leaf Co. was tasked to create a program that teaches a user the correct way to play Blackjack. We decided to create a console game that will act as the dealer and plays Blackjack with the user. Depending on the user's action, the console game will also provide a text prompt indicating which action was in the fact the most statistically ideal in order to teach the player what the best plays are for each hand scenario. These helpful text prompts will be created by inputting a text file that has a statistically created strategy table for Blackjack into the console game. This basic strategy table is derived from a computer simulation where a million hands are analyzed. Through the computer's trial and error, we can find out which decisions are best for the player, given every possible combination of player and dealer hands. In this case, the strategy table will have the most statistically ideal moves that would lead the player to reach Blackjack depending on the current dealer/player hands. Ultimately, this program will be used as both a learning experience and interactive activity to provide users with a better understanding of this fun game.

1.1 Goals

The goal of this project is to teach the player how to correctly play the game of Blackjack. This application will provide an intermediate knowledge of the game while providing the user with an editable strategy table. The

application will assist and guide the user when playing the game. The game will devise various combinations to achieve the score of “21”. The user will be able to view the possibility of winning with a statistical analysis for assistance.

2.0 Product Specification

2.1 Database

2.1.1 MySQL

SQL (Structured Query Language) is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL). SQL is the most popular language for adding, accessing and managing content in a database. It is most noted for its quick processing, proven reliability, ease and flexibility of use. MySQL is an essential part of almost every open source PHP application. Good examples of PHP/MySQL-based scripts are phpBB, osCommerce and Joomla.

It is necessary to download MySQL Workbench from the MySQL website in order to manage and manipulate the blackjack table.

2.2 Desktop App

The desktop game will function as the Dealer and the Player will make decisions based on the cards shown throughout the different plays whether they will split, double-down, hit, or stand every time. The desktop game will be able to teach the Player how to correctly play Blackjack based on the

strategy card developed by the database program which allows for the highest probability of winning. The user will be able to interact with the desktop game via the console application and will be able to edit their strategy card.

2.3 Frequency Statistics

The Frequency Program will formulate a frequency table to show the odds of a number of occurrence for each possible hand derived from the database for a million hands.

2.4 Probability Statistics

The Probability Statistics will formulate a strategy table giving the probability of winning after each hand combination. After each card is drawn and each move is made, then a new percentage will be conducted and the strategy table will change. The Percentages will be formed to show the percentage of a win conducted by the selected moves (Split, Double Down, Hit or Stand). The strategy table will be presented in the game program with the formulated statistics.

2.5 Strategy Table

This basic strategy table is derived from a computer simulation where a million hands are analyzed (Program 2). Through the computer's trial and error, we can find out which decisions are best for the player, given every possible combination of player and dealer hands.

From those million hands, we will then be able to create four different tables, each representing the four different moves we implemented (Split, Double Down, Hit, and Stand) in the console game (Program 3). Our program will select the highest probability from these four tables which converts to an ideal player action which in turn converts to a single character that will represent the move.

In this case, the strategy table will have the most statistically ideal moves that would lead the player to reach Blackjack depending on the current dealer/player hands.

This table of single characters representing an ideal player moves can then be inputted as a text file into our console game. The console game interprets this text file and creates helpful text prompts that let the player know what action they should have taken after each play, regardless of which move the player actually did. Figure 1 below showcases how this text file of the strategy table will look like when used for the console game.

The idea of using a text file for the strategy table allows the player to use their own strategy table if needed (i.e. more flexibility allowed).

We came up with an overall algorithm for the strategy table that can referenced under "`StrategyMethod.java`". This file describes how we would choose the best move using the million hands simulation from our existing database. It returns the best move in the form of a character which

will then be used to generate the actual strategy table (e.g. ‘D’ for double down, ‘P’ for split, ‘H’ for hit, and ‘S’ for stand).

```
X   2  3  4  5  6  7  8  9  10 A
5   H  H  H  H  H  H  H  H  H  H
6   H  H  H  H  H  H  H  H  H  H
7   H  H  H  H  H  H  H  H  H  H
8   H  H  H  H  H  H  H  H  H  H
9   H  D  D  D  D  H  H  H  H  H
10  D  D  D  D  D  D  D  D  H  H
11  D  D  D  D  D  D  D  D  D  H
12  H  H  S  S  S  H  H  H  H  H
13  S  S  S  S  S  H  H  H  H  H
14  S  S  S  S  S  H  H  H  H  H
15  S  S  S  S  S  H  H  H  H  H
16  S  S  S  S  S  H  H  H  H  H
17  S  S  S  S  S  S  S  S  S  S
18  S  S  S  S  S  S  S  S  S  S
19  S  S  S  S  S  S  S  S  S  S
20  S  S  S  S  S  S  S  S  S  S
21  S  S  S  S  S  S  S  S  S  S
A,2 H  H  H  D  D  H  H  H  H  H
A,3 H  H  H  D  D  H  H  H  H  H
A,4 H  H  D  D  D  H  H  H  H  H
A,5 H  H  D  D  D  H  H  H  H  H
A,6 H  D  D  D  D  H  H  H  H  H
A,7 S  D  D  D  D  S  S  H  H  H
A,8 S  S  S  S  S  S  S  S  S  S
A,9 S  S  S  S  S  S  S  S  S  S
A,10   S  S  S  S  S  S  S  S  S  S
A,A P  P  P  P  P  P  P  P  P  P
2,2 P  P  P  P  P  P  H  H  H  H
3,3 P  P  P  P  P  P  H  H  H  H
4,4 H  H  H  P  P  H  H  H  H  H
5,5 D  D  D  D  D  D  D  D  H  H
6,6 H  P  P  P  P  H  H  H  H  H
7,7 P  P  P  P  P  P  H  H  H  H
8,8 P  P  P  P  P  P  P  P  P  P
9,9 P  P  P  P  P  P  P  P  S  S
10,10  S  S  S  S  S  S  S  S  S  S
```

Figure 1. An example of how the strategy table will look like as a plain text file in order to be inputted into the console game for analysis.

3.0 Program Behavior Specifications

3.1 Database and Record Layout

The database will collect a random handset of 20 cards for each deck size. When implementing a simulator for hand analysis in blackjack, we are playing with a deck of “integers” using SQL in the MySQL Workbench. To get a better idea of this layout, Figure 2 can be referenced below.

```
mysql> use bj;
Database changed
mysql> SELECT * FROM blackjackdatabase;
+----------------------+-----------+
| idBlackJackDatabase  | CardValue |
+----------------------+-----------+
|                    1 | Ace       |
|                    2 | 2         |
|                    3 | 3         |
|                    4 | 4         |
|                    5 | 5         |
|                    6 | 6         |
|                    7 | 7         |
|                    8 | 8         |
|                    9 | 9         |
|                   10 | 10        |
|                   11 | Jack      |
|                   12 | Queen     |
|                   13 | King      |
|                   14 | Ace       |
|                   15 | 2         |
|                   16 | 3         |
|                   17 | 4         |
|                   18 | 5         |
|                   19 | 6         |
|                   20 | 7         |
|                   21 | 8         |
|                   22 | 9         |
|                   23 | 10        |
|                   24 | Jack      |
|                   25 | Queen     |
|                   26 | King      |
|                   27 | Ace       |
|                   28 | 2         |
|                   29 | 3         |
|                   30 | 4         |
|                   31 | 5         |
|                   32 | 6         |
|                   33 | 7         |
|                   34 | 8         |
|                   35 | 9         |
|                   36 | 10        |
|                   37 | Jack      |
|                   38 | Queen     |
|                   39 | King      |
|                   40 | Ace       |
|                   41 | 2         |
|                   42 | 3         |
|                   43 | 4         |
|                   44 | 5         |
|                   45 | 6         |
|                   46 | 7         |
|                   47 | 8         |
|                   48 | 9         |
|                   49 | 10        |
|                   50 | Jack      |
|                   51 | Queen     |
|                   52 | King      |
+----------------------+-----------+
52 rows in set (0.00 sec)
```

Figure 2. Database and Record Layout

3.2 Program 2 Behavior

Program 2 will utilize the database and calculate the frequency that a particular hand appears out of *n* hands for the dealer's one card and the user's first two cards.

1. Calculates random hands and tallies the frequency of their occurrence
2. Suit doesn't matter until playing the game
3. There is a frequency table for each deck: 1,2, 4, 6, and 8
4. The cards shall be drawn in the following order:
    - 1 - Player
    - 2 - Dealer (Up Card)
    - 3 - Player
    - 4 - Dealer (Down Card)
5. Randomizes one million hand combinations for statistic purposes
6. Below is pseudocode for this component:

```
import statements

/** DECLARE & INITIALIZE ARRAYS */

// array containing player cards
Array[i]
// array containing dealer cards
Array[j]
// array containing card counts for frequency table
Array[i][j] //frequency table

/**
 * Location Details
 * Sum  Array[i][j] = Array [0-9][0-8]
 * Pair Array[i][j] = Array[10-18][0-8]
 * Aces Array[i][j] = Array[19-27][0-9]
```

```
*/

/** DECLARATIONS */
Boolean pair, ace;
int sum = 0, tally = 0;

/** ASK FOR NUMBER OF DECKS TO USE */
// 1, 2, 4, 6, 8

/** GRAB VALUE OF CARDS FROM DATABASE */
/** USE DATABASE TO GRAB 1st 4 CARDS WITH A HAND AT ANY SINGLE
TIME */
/**
 * CARDS BEING READ
 * 1 - Player
 * 2 - Dealer Down
 * 3 - Player
 * 4 - Dealer up
 *
 * User's 2 cards 1, 3
 * Dealer's up card 4
 * Card #2 is down
 */
/**
 * LOOP STARTING NOW
 */
/**
 * get sum of cards
 * get value of dealer card
 * determine if there is a pair
 * determine if there is an ace
 * if there is no pair or ace tally at location array[i][j]
 */

/** LOCATIONS OF i and j */
[i] = 0-9
[j] = 0-9

/**
 * IF USER HAS SUM
 * AND DEALER HAS ACE
 * TALLY AT LOCATIONS
*/
[i]= 0-9
[j]= 9
```

```
/** DETERMINE IF CARDS ARE PAIR */
if (value of usrCard1 == value of usrCard2)
  pair is true;

/** IF CARDS ARE PAIR , CHANGE TALLY LOCATIONS*/
if (there is a pair go to the pair range in array)
  Array[i] locations: 10-18

/**
 * DETERMINE PAIR USER HAS
 * GET CARD VALUES
 * GET SUM OF DEALER CARD
 * TALLY AT LOCATION ARRAY[i][j]++
 */

//if user has a pair and dealer has an ace
tally at location array[i][j]++;

/** DETERMINE WHETHER THERE IS AN ACE OR NOT */

// if user has an ace and a value of 2nd card and dealer has a
sum
tally at location array[i][j]++;

// if user has an ace and value and dealer has an ace
tally at location array[i][j]++;

// if user has a pair of aces and dealer has a sum
tally at location array [i][j]++;

// if User has a pair of aces the dealer has an ace
tally at location array[i][j]++;

// Ace is true
// tally at locations
[i] = 19-27
[j] = 9

/**
 * ONCE HAND IS RETRIEVED FROM DATABASE
 * INCREMENT PLACE IN ARRAY
 * REPEAT PROCESS
 */
Array[i][j]++;
```

```
/** LOOP ENDS HERE */
/** PRINT OUT ARRAY IN TABLE FORMAT */
print array[i][j]
```

3.3 Program 3 Behavior

Program 3 will calculate the probability of hitting, splitting, doubling down or standing. The highest probability for each scenario Split, Double Down, Hit or Stand) will be used in creating the strategy table.

3.4 Blackjack Game

From start to finish the game itself will assume the role as the dealer to play against the user. The game will have an editable strategy table, asked how many decks he/she want to play with and the hand is built. The process of the game is as follows:

1. ThePlayer is dealt two cards.
2. The dealer deals two cards, one being visible on the table.
3. The player is 'processed' - i.e. one player will keep playing their hand until they are forced to 'Stand','Bust', etc..
4. The dealer 'processes' their own hand until they either 'Stand' or go 'Bust'.
5. The table is evaluated, returns are calculated, and the game is concluded.

Regardless of what action the player takes, the console game will generate a text message after each play which states what action should have been

taken in according to statistics and the results generated from Programs 2 and 3 (Million Hands Analysis & Probabilities).

For example, if a player decides to stand when in fact they would have been better off hitting with that particular hand scenario, then the console game will let them know (e.g. “You should have hit.”). These text prompts allow the player to recognize their mistakes after each play but does not allow them to go back and change their action.

3.4.1 Scenarios

3.4.1.1 Split

All possible scenarios for each hand must be played and if any scenario is a winner then that hand wins.

3.4.1.2 Stand

If there’s a high chance there will be a bust should a card be taken, it is recommended to stand.

3.4.1.3 Hit

A card is taken until the result of the cards at hand is 17+ or a bust.

3.4.1.4 Double Down

A card is taken until the result of the cards at hand is 17+ or a bust.

3.4.2 Rules

Below are the rules of Blackjack:

- Value of hand is calculated by adding the cards in the hand
    - Numeric cards retain its numeric value
        - e.g. a number 2 card would have a value of 2
    - Jack, Queen, and King all have a value of 10
    - An Ace can either have a value of a 1 or 11
        - An Ace will have a value of 11 unless it places the hand over 21
        - After the first Ace in the hand, the second and so on Aces would have a value of 1
- Player may double down on 9, 10, 11 and Ace anything
- Player may double down or split again after a split
- When Dealer hits soft 17, it is defined as an Ace counted as 11
- Blackjack pays 3:2

4.0 User Specifications

4.1 Overview

User specific functionalities of this project include:

- Blackjack strategy card
    - The user will be able to edit their strategy card generated by the database and input it for use in the console game.
- Blackjack game that teaches a user the correct way to play Blackjack

- If a player tries to make the wrong move i.e. Hitting when they should stood, a text message will show up and show the user the correct move that should have been made.

5.0 Security

5.1 Overview

The Software Development Life Cycle as it relates to the current iteration is that 4Leaf Co. is in the Architecture and Design phase of the product leading into Development. The Architecture and Design phase sets up the background of the programs and initial design of the product. Each function is tested for functionality within each program of the product.

6.0 Testing

6.1 Overview

Throughout the development of the Blackjack program, tests will be conducted on a regular basis to ensure the product is in time for the final delivery. As we get closer to final delivery, there will be an increase in the amount of tests that are conducted. The components that will be tested throughout the development include the database, statistic programs: the probability of winning based on *n* hands and probability of winning during the game, along with the actual blackjack game itself.

7.0 Tracking and Control Mechanisms

7.1 Document Revision History

Version	Implemented By	Approved By	Date	Reason
1.0	Frances Coronel	Jordan Quick	1/21/16	Iteration 1
2.0	Calvin Chambers Frances Coronel	Jordan Quick	2/11/16	Iteration 2
3.0	Calvin Chambers Anesha Passalacqua Frances Coronel	Jordan Quick	3/3/2016	Iteration 3
4.0	Calvin Chambers Anesha Passalacqua Frances Coronel	Jordan Quick	3/31/16	Iteration 4

5.0	**Calvin Chambers** **Frances Coronel**	**Jordan Quick**	**4/26/16**	**Iteration 5**