Term Project: “*Roulette*”

DCY Student

January 13, 2006

1. Introduction
The roulette is a popular gambling game found in all major casinos. In contrast to many other gambling games such as black jack, poker, backgammon, or bingo, a roulette player’s odds are purely based on luck, but not dependent on other players’ moves, decisions, or participation. It earns some of its fans for that reason. From a mathematical point of view, the odds of the game can be calculated easily, and accurately. Both the word and the game have a French origin dated back to the 18th century. “Roulette” means a small wheel in French. This small wheel appeared in Paris around 1765. The game was introduced by French emigrants into the United States, and into the first casinos in New Orleans around 1800. Legend was that Franco Blanc, founder of the first casinos in Monte Carlo, sold his soul to the devil for the game’s secret. Try adding up the numbers from 1 to 36—you may get a surprise! [1]

The purpose of this project is to write a simulator for the game in Java. Given the limited resources we have, it is not possible to create a simulator of professional standard, or that fully simulates all practical aspects of the game. However, our simulator does simulate main aspects of the game, including placing bets, different betting options (e.g., high, low, number, dozen, or odd bets), drawing a number, and losing or winning a bet according to the corresponding payoff multiples.

Equally important, one purpose of this exercise is to demonstrate and practice the knowledge and skills learned in the course. Indeed, the author has found this a valuable exercise to put everything together into practice. For simplicity, from now on, we shall call the simulator by the name, *Roulette*

2. Description
Roulette covers all features shown in the “Term Project Ideas” document, plus some additional ones. A special flow chart is constructed to show running of the simulator. (See Figure 1.) Below, we describe six general steps in running Roulette, together with examples from screen sessions. The first column of Figure 1 is a traditional flow chart. For each element in the chart, the second column of the figure labels the particular step the element corresponds to. Thus, using the chart, the reader can easily correspond each element in the flow chart to (i) the corresponding step described here and (ii) the corresponding screen session display (when it is printed here).
For example, the first element in the flow chart after start is “Print greetings and instructions,” whose second column (Section Number) reads, “2 (step 1)”. So it is described by the first step below.

The reader is encouraged to first study the six-step description below to get a feel of running Roulette, which is very straightforward, and then use the flow chart to get an overall understanding of the simulator’s structure and organization.
.

Step 1. Roulette first prints a greeting message and asks the user to type in the name of each player and the amount of chips ($) to be changed.
A blank line should be returned when no more players will participate. For example,

```
Welcome to Roulette!
First you will type the names of the players.
When you are done entering names, just enter
        a blank line to continue.
What's the name of player #1?   John F. Kennedy
How much money does John F. Kennedy start with?  $ 1000
What's the name of player #2?   Barbara Bush
How much money does Barbara Bush start with?  $ 2000
What's the name of player #3?   Alberto Gonzalez, Jr.
How much money does Alberto Gonzalez, Jr. start with?  $ 1500
What's the name of player #4?
```

Step 2. A summary of all players' information is then printed:

```
SUMMARY OF PLAYERS :
-------------------------------------------------------------------------------
( 1)       John F. Kennedy has $ 1000       ( Status:   Active )

( 2)          Barbara Bush has $ 2000       ( Status:   Active )

( 3) Alberto Gonzalez, Jr. has $ 1500       ( Status:   Active )

No. of Active Players = 3

-------------------------------------------------------------------------------
```

At this point, let us define

<u>Definitions/Notations:</u>
N = the number of players entered, participating in the game;
n = the number of players *actively* participating, *i.e.*, who still have money left and have not chosen to leave. Initially n = N. As time goes on, n will decrease, and gradually drop to zero. However, N stays the same after it has been recorded.
In the example shown above, initially, N = n = 3 and all three players are active.

Step 3. If there are no active players left, *i.e.*, when n becomes zero, the game would terminate.

Step 4. The next active player is identified and asked to place his/her bet.

- The value of his chips is printed.
- A brief summary of the betting options and payoff ratios are given.
- He/She is asked to state the amount and type of bet to be placed.
- If the bet is on a specific number, on a dozen bet, or a split bet, an additional input is required to complete the bet specification.

Example:

```
Ok ** John F. Kennedy ** you have $ 1000
```

```
BETTING RULES :
-------------------------------------------------------------------------------
If you bet correctly on a NUMBER from 1 to 36 you win ** 35 ** times
        your bet, plus your original bet.

If you bet correctly on the number coming up
        LOW (1-18), HIGH (19-36), EVEN (2, 4, 6, etc.), ODD (1, 3, 5, etc.)
        you win ** the amount of your bet ** , plus the original bet.

If you bet correctly on a SPLIT BET, i.e., on two consecutive numbers
        you win ** 17 ** times the amount of your bet plus the original bet.

If you bet correctly on a DOZEN bet, i.e.,
        whether the no. is in (1-12), (13-24), or (25-36)
        you win ** 2 ** times the amount of your bet plus the original bet.

Note: You always LOSE if the winning no. is 0.
-------------------------------------------------------------------------------
How much do you want to bet?  $ 100

BETTING MENU :
-------------------------------------------------------------------------------
Select what you want to bet from the menu:
N for betting on an actual NUMBER ;
H for betting on HIGH ;
L for betting on LOW  ;
E for betting on EVEN ;
O for betting on ODD  ;
S for betting on SPLIT BET, and
D for betting on DOZEN.

-------------------------------------------------------------------------------
Enter your selection:
H
```

Step 5. Roulette would draw a number and tell the player whether he/she has won or lost, and the net amount of chips left.

Step 6. If the player has no chips left, he must leave; otherwise, he can choose to leave or continue. (A player leaving is considered "becoming inactive" and will decrease n by 1.)
Example:

```
The wheel goes round and round and round ....   and it comes up with *** 18 ***

Sorry. You have LOST.
You now have $ 900.  Want to Keep Going? (y/n)  n
```

At this point, Roulette will loop back to Step 2 above and print an updated summary, e.g.,

```
SUMMARY OF PLAYERS :
-------------------------------------------------------------------------------
( 1)       John F. Kennedy has $ 900        ( Status: Inactive )

( 2)          Barbara Bush has $ 2000       ( Status:   Active )

( 3) Alberto Gonzalez, Jr. has $ 1500       ( Status:   Active )

No. of Active Players = 2

-------------------------------------------------------------------------------

Ok ** Barbara Bush ** you have $ 2000
```

3. Betting Options

The reader should now have a good overall picture of running Roulette, and is encouraged to try running the program. We now discuss in more details the seven betting options available, as shown in "Betting Menu" in Step 4 above. Let *r* be the number drawn by Roulette.

(i) A NUMBER bet: The player wins if r is the same as the number the player is has bet on. To bet on this option, the player will be asked to input the number he is betting on in Step 4, e.g.,

```
Enter your selection:
N
Which no. in (1 - 36) do you want to bet on?    0
Which no. in (1 - 36) do you want to bet on?    3
```

(ii) An EVEN or (iii) ODD bet: The player wins if r is even or odd, respectively, as the bet is on.

(iv) A LOW or (v) or HIGH bet: The player wins if r is between (1-18) or (19-36), respectively, as the player has bet on.

The next two betting options are features not included in the original project ideas documents.

(vi) A SPLIT bet: In this betting option, the player selects two consecutive numbers in (1-36) to bet on, and he wins if any of the two numbers are drawn.

(vii) A DOZEN bet: In this betting option, the player bets on r falling on one of the three dozens of numbers, *i.e.*, in (1-12), (13-24), or (25-36). Again, in step 4 above, the user will be asked to specify the particular dozen he is betting on, e.g.,[1]

```
Enter your selection:
D
Give 1, 2, or 3 for betting on (1-12), (13-24), or (25-36), respectively.
        4
Give 1, 2, or 3 for betting on (1-12), (13-24), or (25-36), respectively.
        0
Give 1, 2, or 3 for betting on (1-12), (13-24), or (25-36), respectively.
        3
```

[1] Notice from the example that when only a specific range of numbers are acceptable, Roulette will not accept any input outside of the range. It will simply loop back and request the same input again.

4. Payoff Ratios

By payoff ratio, we are referring to the amount the gambler will get as a reward for each $1 of his bet if he wins. After researching on several websites on the Internet, the following payoff ratios are used, which is slightly different that in the project suggestion document for the single-number betting option. In any case, these ratios are coded as public static final int constants in the Java programs and can be modified easily as desired.

Type of Bet	Payoff Ratio
Number	35
Dozen	2
Split	17

Type of Bet	Payoff Ratio
High or Low	1
Even or Odd	1

Note that the payoff does not include the original $1 bet. So suppose one has $10 and places $2 on a split bet and wins, he will end up with $10 + $2 × 17 = $44. If he loses the bet, he will have only $8 left.

5. Software Implementation (Overview)

The software is written in Java (JDK version 1.5), compiled and run on ice.fas.harvard.edu. The program is stored in two files:

- *Gambler.java* performs functions and contains variables one would associate with a player at a roulette table. For example, this includes keeping track of how much money he has, the amount and type of each bet, and whether he wants to play or leave.
  This is a relatively small file, containing mostly accessor and mutator methods.

- *RouletteTable.java* performs functions and contains variables one would associate with a roulette table at a casino, such as interfacing with the gamblers, taking bets, "turning the wheel" to get a number for each play, doing the accounting, and taking money from / giving money to the player after a play. Most of the working of Roulette is done and controlled by the program in this file.

A program listing is attached with the project submission.

Let us discuss a few details related to the implementation of Roulette which a programmer, developer, or user of Roulette may want to pay attention to.

- **<u>Section 5.1</u>**
  Recall that Roulette keeps count of the number of players (N) who join the game initially, and the number of active players (n). Initially, n = N, but any player may choose to leave the game after a bet, or he may have lost all his money be forced to leave the game. In either case, he is becoming "inactive" and n will decrease, until n becomes 0 when the game finishes.

  Although Roulette supports multiple players, the current implementation allows one player to bet at a time, as shown in the "project ideas" document. *While* loops are used in the main method to allow the players to take turn to bet:

```
while ( N  > 0 )   // there is at least one active player left
{
        // search for the next active player, i.e., his/her index in gambler[ ]

        while ( ! dealer.thePlayerIsActive( i = ++ i % n ) )  ;

          dealer.placeYourBets( i ) ;
          dealer.turnRollette();
          dealer.doAccounting( i ) ;
          dealer.checkIfContinue( i ) ;
          dealer.printSummaryOfPlayers();
}
```

Here, the first *while* loop is executed as long as there is an active player left. The variable i is an index of the next *active* player to play. Since the method dealer.thePlayerIsActive(j) returns true or false, depending on whether player j is active or not, the second *while* loop keeps increasing the index i until the next active player's index is reached. To maintain the index within range, index arithmetic is performed with a modulus of N, as all players have indices in { 0, … , N-1 }.

- **Section 5.2**
  Another detail worth mentioning is that in each instance of the object Gambler, there is an instance variable, *int numberBet*. This is used for storing information for several kinds of bets as follows. For betting on

  (1) specific number, *numberBet* is the specific number the player is betting on;
  (2) split bet, *numberBet* is the smaller of the 2 consecutive numbers he is betting on, and
  (3) dozen bet, *numberBet* is 1, 2, or 3, corresponding to the ranges (1-12), (13-24) and (25-36), respectively, the player is betting on.

- **Section 5.3**
  The methods in RouletteTable.java are grouped so that each group perform functions that correspond to one of the methods called in main( ), plus an additional group of "getter" and "setter" methods. Anyone needing to understand the program coding should refer to the flow chart again. The third column of the chart lists the methods in the file RouletteTable.java that correspond to element of the chart and different steps of running as described in Section 2.

6. How To Implement A Biased Wheel

In our current implementation (of an unbiased wheel), the following line is used to generated a number between 0 and 36 randomly in method turnRoulette( ), in Roulette.java:

```
int n =  ( int ) ( Math.random() * ( MAX_R_NUMBER + 1 ) ) ;
```

where MAX_R_NUMBER = 36 represent the maximum number present on the wheel. Suppose instead, we want to implement Roulette with a bias so that a HIGH number will be generated with a probability p = 70%, the following changes should be implemented.

First p should be declared as a constant at the beginning of the class definition:

```
public static final double P = 0.7 ;
```

Then in method turnRoulette( ), the local variable, n, for the number drawn, should be generated by:

Method 1:

First generate a random number. If it is less than p, generate a HIGH number (i.e., one in19-36) randomly; otherwise generate a LOW number (i.e., one in 0-18) randomly. In Java, this looks like:

```
int n ;

if   ( Math.random( ) < P )
{
      n =  1 + HALF_NUM  + ( int )  ( Math.random( ) *  HALF_NUM  ) ;     // generate a random number in [19,36]
}
else
{
      n =   ( int )  (  Math.random( ) * ( 1 + HALF_NUM  )  ) ;                       // generate a random number in [0,18]
}
```

where HALF_NUM is a (public, static, and final) integer constant equal to 18.

The code fragment above achieves the required distribution because with probability p, the first assignment to n is used—where n becomes a random integer in [19,36], and with probability (1-p) the second assignment is used—where n becomes a random integer in [0,18].

In fact, a better method that does not require generating a second random number is possible, using knowledge in conditional probability.

Method 2:

```
int n ;
double randNum = Math.random( ) ;

if   ( randNum  < P )       // p must be > 0
{
      randNum  / = P ;

      n =  1 + HALF_NUM  + ( int )  ( randNum *  HALF_NUM  ) ;     // generate a random number in [19,36]
}
else
{
     randNum = ( randNum – P ) / ( 1 – P ) ;

      n =   ( int )  (  randNum * ( 1 + HALF_NUM  )  ) ;                    // generate a random number in [0,18]
}
```

In this method, if the first random number generated, randNum, is less than p, conditional on this fact, randNum is uniformly distributed in (0,p). So instead of generating a second random random number, one can simply use (randNum/p) as the second required random number, since it is uniformly distributed between 0 and 1.
Similarly, if randNum >= p, conditional on this fact, we know that randNum is uniformly distributed in (p, 1). Hence [randNum-p / (1-p)] can be used as the second random number, as it is also uniformly distributed between 0 and 1.

7. Conclusion and Summary

In this report, we have described an implementation of the roulette game by Java programming. In plain English and examples from monitor sessions, we have shown the reader how Roulette simulates the game step by step, from asking multiple players to enter their names, changing the required amount of chips, to placing bets on different options, drawing a winning number, and doing the reward/loss calculations. As additional features, the simulator has included two extra betting options (for split bets and dozen bets) not covered by the original project proposal.

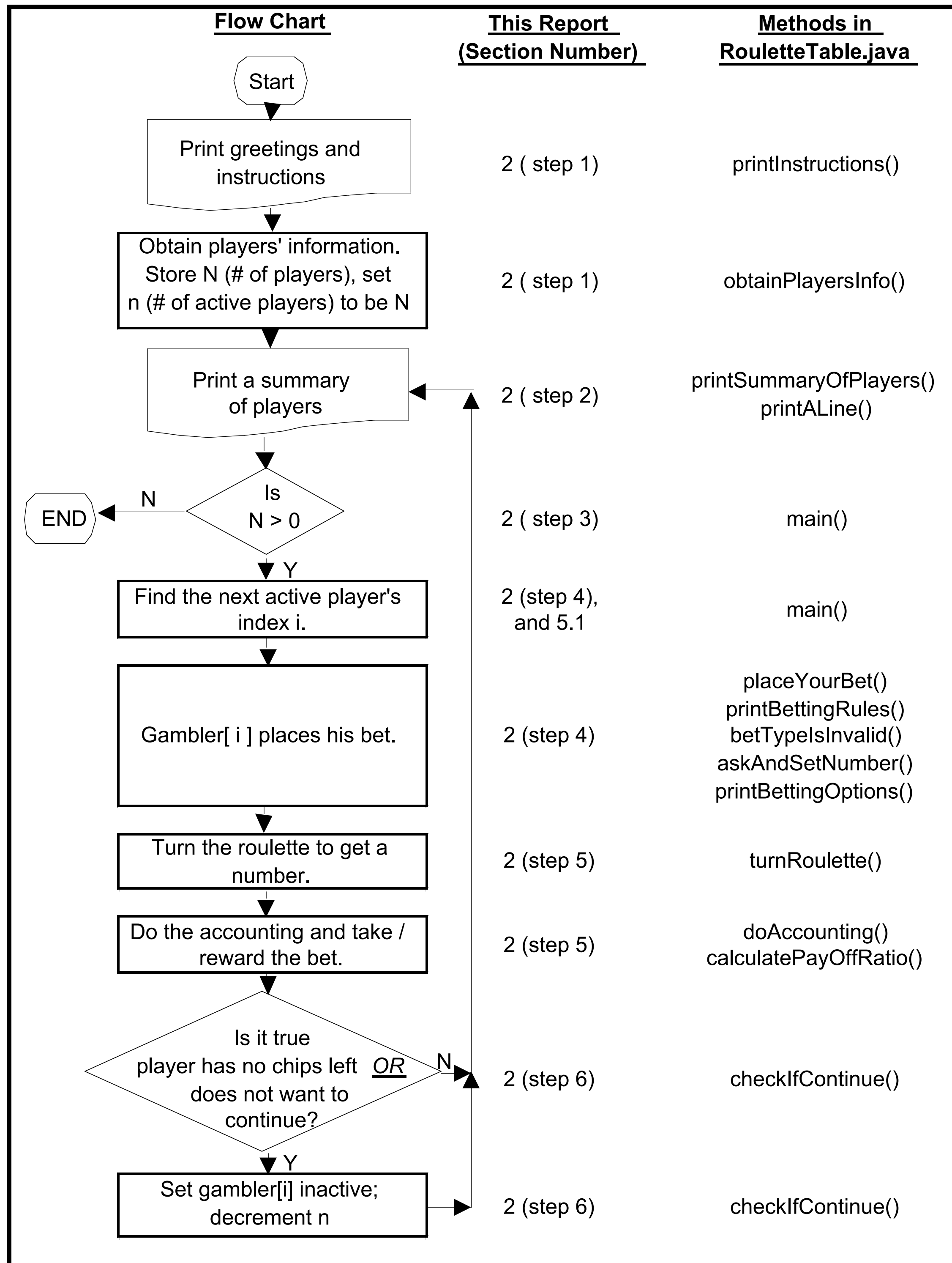
We have discussed in a little bit more details the different betting options, payoff ratios, and several aspects in our software implementation. To answer one of the questions in the proposal, in Section 6, we have proposed two methods for implementing a biased wheel. The first one is more straightforward. The second one is not complicated but uses knowledge of conditional probability to half the required number of random numbers.

Last but not the least, a special flow chart is created (see Appendix 1). This chart is unusual because it links together (i) the process flow and organization, (ii) the description with monitor session examples in Section 2, and (iii) the Java methods used in the implementation, allowing the user to have an overall view of the program structure, organization, and functions.

8. Reference

[1] “History of Roulette”, http://www.gamblecraft.com/review/roulette/history.htm

Appendix 1: Figure 1. Flow Chart of Running Roulette



Appendix 2: Program Listings of RouletteTable.java and Gambler.java. (submitted separately).