Roulette Game Application

External Documentation

By: Yuhan Peter Bauer, Zephyr Bégin Zhang, and Hakeem Pit Al-Timime

Due Thursday November 27th, 2025

Introduction

This document provides external documentation for the Roulette Game Application, implemented using Python and the Tkinter GUI library. The application simulates a European-style roulette wheel, provides a full betting table, animates a graphical wheel and ball spin, allows the user to place various bets (ranging from 1-1000), and calculates the outcome and payouts according to standard roulette rules. The documentation is divided into two main components: (a) a user guide that explains how to use the program, the basics of roulette and the rules, and (b) a design guide aimed at programmers who may wish to understand the logic behind the code and what was utilized.

1 User Guide

1.1 Overview of the Program

The Roulette Game Application is an interactive desktop program that simulates a casino-style European roulette table. The program renders an animated wheel on the left of the screen, a full betting table on the right, and controls for selecting chips, a spinning the wheel, managing bets, and seeing the user's balance at the bottom. The interface can only be interacted with a mouse, allowing users to place bets, choose amount to bet, spin, and etc directly on the table by clicking the desired betting areas.

The game uses the European roulette layout, which contains 37 numbers: 0 and 1 through 36. The single zero format offers a more realistic model of European roulette odds. When the wheel spins, both the wheel and the ball rotate with differing velocities and decelerations until both come to a stop. The program then determines the winning number based on the ball's final angle relative to the wheel's orientation.

1.2 Basics of European Roulette

European roulette is played on a spinning wheel with pockets numbered 0 to 36. Each number is coloured either red, black, or green (green is reserved for 0). Players wager by placing chips on a designated betting layout on the table. Common bet types include:

- **Single Bet:** A bet on a single number (pays 35:1).
- **Colour Bet:** A bet on red or black numbers (pays 1:1).
- **Even/Odd Bet:** Wagers on the parity of the winning number.

- **Low/High Bet:** Bets on ranges 1–18 or 19–36.
- **Dozens Bet:** Bets on 1–12, 13–24, or 25–36 (pays 2:1).
- **Column Bet:** Bets on one of three vertical columns (pays 2:1).

The user begins with a fixed balance of $2000, which is used to place wagers before each spin. Once chips are placed and the player presses "Spin," the total amount bet is deducted from the balance, and any winnings are returned after the wheel stops.

1.3 How to Use the Program

When the program launches, the main interface appears with three major components: the wheel, the betting table, and the control panel.

1.3.1 Placing Bets

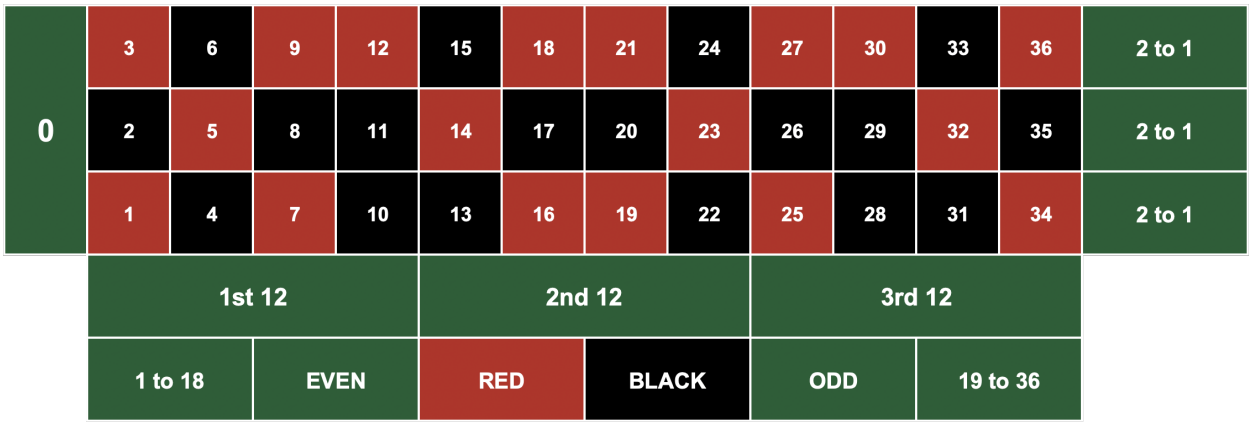

Figure 1: Betting Table

To place a bet, the user must first select a chip denomination from the control panel. Available chip values are 1, 5, 25, 100, 500, 1000. Once selected, the user may click any

valid betting cell on the table (Options in Figure 1 above). Each click adds the selected chip value to the chosen bet. The table clearly displays chips and their total value inside the cell.

Multiple bets may be placed before each spin. The user may also stack additional chips on any previously selected bet area.

1.3.2 Clearing Bets

Before spinning, the user may press the "Clear Bets" button to remove all bets from the table. This also gets rid of all chip indicators and resets the bet summary.

1.3.3 Spinning the Wheel

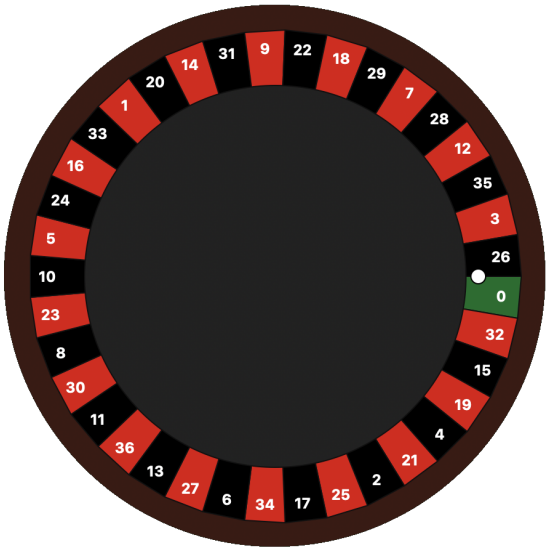

Figure 2: Roulette

To initiate a round, the user selects "Spin." The program first checks that at least one bet has been placed and that the total bet amount does not exceed the available balance. Moreover, upon balance hitting 0 the user will be asked if they would like to quit the program. If these conditions are met, the wheel animation begins.

During the animation, the wheel and ball rotate independently using decreasing angular velocities. When both motions come to rest, the program identifies the winning number by computing the ball's relative position on the wheel. The winning sector is outlined in yellow for clarity.

1.3.4 Viewing Results

The lower portion of the interface displays the result message. It shows:

- the winning number
- the colour of the number
- whether the player won or lost
- and the amount won (if won)

The player's balance is changed accordingly. Bets stay on the table after each round, allowing the player to spin again (if balance is sufficient) or manually clear the bets.

2 Design Guide for Programmers

2.1 Overall Architecture / Software Organization

The program is implemented using object-oriented design, divided into three primary classes:

1. **WheelCanvas** – responsible for drawing and animating the roulette wheel and the ball.

2. **TableCanvas** – responsible for drawing the betting table, capturing user clicks, and displaying chip graphics.

3. **RouletteApp** – the main Tkinter application class, controlling game flow, balance updates, and payouts.

This separation ensures segmenting, making it easier for felllow developers to modify and understand one aspect of the program without affecting the others; additionally, reducing confusion and/or complications.

2.2 Wheel Rendering and Animation

The wheel is drawn using the Tkinter Canvas widget. The numbers on the wheel are shaped as long isosceles triangles then cutoff using a circle in the middle to only show the number. The European number order is stored in a list (From online source referenced at the end), and each sector is generated using geometric computations. The wheel and ball angles are updated incrementally using Tkinter's `after` method, which schedules repeated animation frames.

Randomized initial velocities and deceleration values to make more of a realistic spin. The final winning number is computed by comparing the ball angle to the wheel's rotated position.

2.3 Table Layout and Betting Mechanism

The betting table is also implemented using many rectangles.To expand, each bettable region is represented by a rectangle with an associated identifier, stored in a dictionary. When the

user clicks a rectangle, the program interprets the action as a bet and updates the internal bet dictionary.

Chips are drawn as small circular shapes with numerical labels showing the total bet on that cell. The table supports number bets, colour bets, parity bets, ranges, dozens, and column bets.

2.4 Payout Computation

The payout system iterates over all active bets and applies roulette rules. The implementation uses simple mappings:

- Straight-up win returns 36 times the bet.
- Even-money bets return 2 times the bet.
- Dozens and columns return 3 times the bet.

All winnings are added back to the player balance after the spin concludes.

2.5 Dependencies and Tools Used

- Python 3 (Thonny, Pycharm, Spyder used by us specifically)
- the Tkinter standard GUI library
- the math and random modules
- https://realpython.com/python-gui-tkinter/
- https://www.venetianlasvegas.com/resort/casino/table-games/roulette-basic-rules.html

Conclusion

This external documentation outlines the functionality, usage, and design of the Roulette Game Application. The user guide provides instructions for operating the program, how the program works, and the organization of such.