Magic Square Game

By Muhammad Omer Azhar

a) Overview

This project involved designing a game where the user generates a square by providing an odd integer. The generated square is then shuffled, and the user must rearrange its elements until it becomes a "magic square." A magic square is a grid where the sum of all rows, columns, and diagonals are identical. The assignment required creating an interactive experience where the user inputs commands in a specific format, allowing them to manipulate the square and reach the goal. The focus was on developing a program that provides a smooth, user-friendly interface, validates inputs, and handles errors efficiently.

b) Solution Design

The program was organized using three core classes to ensure modularity and maintainability:

1. **Base Class (**Q1b_24092763**)**: Handles square generation and shuffling, as well as general game mechanics.
2. inputHandling(Subclass): Validates and processes user inputs, ensuring smooth communication between the user and the program.
3. gameMechanics(Subclass): Manages gameplay interactions, such as moving elements within the square and checking the victory condition. Also handles user inputs.

Key design decisions included:

Expected User Inputs:

The game begins by prompting the user to input an odd integer, which determines the size of the square to be generated. Following this, players provide inputs in the format "row column direction" (e.g., `1 2 u` to move an element up) to rearrange elements within the square. This format ensures precise control over the game while maintaining simplicity and ease of use for the player.

Magic Square Generation:

The program utilized a mathematical algorithm specifically designed for generating odd-order magic squares, ensuring that the grid met the required properties from the outset. To add variety, randomization was applied through the use of nested loops, which shuffled the square while carefully preserving its structural integrity. This combination ensured both consistency in results and a unique experience for each game session.

Victory Verification:

Logic was implemented to confirm that all rows, columns, and diagonals sum to the same target value.

Error Handling:

The program includes checks for invalid inputs (e.g., non-integer values, out-of-range indices, unsupported commands) and provides detailed feedback to guide the user. The modular structure made debugging easier, improved scalability, and allowed for future enhancements.

c) Discussion of the Completed Solution:

Scope and Performance:

The game successfully fulfils the requirements by providing a clear objective, interactive gameplay, and efficient handling of user commands. It guides players on input formats and allows them to end the game at any time.

Quality:

Careful attention was given to error handling and user feedback, ensuring the program operates smoothly and reliably. Edge cases were accounted for, and exceptions were handled gracefully to prevent crashes.

Highlights:

The game's random shuffling mechanism guarantees that each session starts with a unique configuration, making the experience fresh and engaging for players. Additionally, the clear and concise input guidance ensures the game is approachable, even for users who are unfamiliar with the concept of magic squares. This combination of features enhances both the replayability and accessibility of the game.

Challenges and Solutions:

The process of developing the game involved tackling several challenges. First, implementing robust input validation was crucial. This included ensuring that the square size provided by the user was a positive odd integer and rejecting any invalid or negative values. Handling edge cases was another key aspect, such as managing situations where the row and column indices entered by the user were out of range or the commands were incorrectly formatted. Additionally, designing a shuffling algorithm required careful thought to balance randomness with maintaining the square's integrity, ensuring gameplay remained consistent. Finally, creating an intuitive and user-friendly interface within the constraints of a text-based application was important to provide clear instructions and a smooth experience for the user. These steps ensured the game functioned reliably while being engaging and accessible.

Additional Features: The program goes beyond basic requirements by providing detailed error messages and interactive prompts, making the gameplay more engaging. The game even prints out warning messages to warn the user, if decides to end the game, he wont be able to edit his square anymore and it will be checked whether it is a magic square or not.

d) Software Test Methodology:

To ensure the program was both correct and robust, a comprehensive testing strategy was adopted. Individual methods, such as generateSquare, validatedUserInput, and isMagicSquare, were rigorously tested through unit testing to verify their specific functionalities. Integration testing was performed to ensure that the main classes—Q1b_24092763, inputHandling, and gameMechanics—interacted smoothly without errors. Edge cases, such as minimum-sized magic squares, invalid inputs, and out-of-range values, were examined to confirm that the program could handle unexpected scenarios gracefully. Gameplay was tested manually to evaluate the user experience and validate the victory condition logic, ensuring it worked as intended. Lastly, consistency testing was conducted by running the program multiple times to verify that the shuffling algorithm consistently generated varied results while maintaining the square's integrity. This thorough approach ensured the game performed reliably under all tested conditions.