University of Layyah
Department of Information Technology

Numbers Puzzle Game
Project Report

A Data Structure Project

A Group Effort By:

Khadija Zafar (UL-BSITM-23-16), Raneem Fatima (UL-BSITM-23-25), Areej Fatima (UL-BSITM-23-39), Raazia Ittahadi (UL-BSITM-23-74)

Course and Lab Instructor: *Mr. Faisal Hafeez*

Class: *BS IT Morning 3rd*

Table of Contents

1. Acknowledgment

We extend our deepest gratitude to our instructor, Mr. Faisal Hafeez, for his invaluable guidance in selecting an appropriate project and programming language. His advice to focus on a project involving data structures and GUI development inspired us to undertake the **Numbers Puzzle Game**. We also thank our group members for their dedication and collaboration, which made this project a success.

2. Summary

This project focuses on developing a Numbers Puzzle Game, commonly referred to as the 15 Puzzle Game. The game challenges players to rearrange numbered tiles in sequential order by sliding them into an empty space. Using fundamental data structures such as **lists** and **stacks**, the project handles game logic, including tile movement, undo/redo functionality, and high-score tracking. The game is implemented using **Python** and the tkinter library for GUI development. Challenges, such as managing high-score states and working with PyCharm, were resolved effectively. This project highlights the integration of programming, problem-solving, and teamwork.

3. Introduction

The Numbers Puzzle Game, also known as the 15 Puzzle Game, is a sliding puzzle where players rearrange tiles in sequential order. This classic game enhances problem-solving and logical thinking skills. Our teacher guided us to select a project that integrates data structures, algorithms, and GUI programming. This project leverages Python's tkinter library for the GUI and uses **lists** and **stacks** to manage game mechanics. By completing this project, we aim to demonstrate the practical application of data structures in a real-world scenario.

4. Objectives

The primary objectives of this project are:

1. To develop a functional **Numbers Puzzle Game** using Python.
2. To demonstrate the use of **lists** and **stacks** for managing game logic.
3. To implement an intuitive graphical user interface (GUI) using Python's tkinter library.
4. To provide advanced features such as:

    - **Undo and Redo Functionality**: Allow players to reverse or reapply their moves.
    - **High-Score Tracking**: Store and display the best scores for each difficulty level.

5. Tools and Technologies

The tools and technologies used in this project include:

1. **Python 3.13**: The primary programming language for implementing game logic.
2. **tkinter**: A built-in Python library for developing graphical user interfaces.
3. **PyCharm IDE**: An integrated development environment for writing and debugging Python code.
4. **Lists and Stacks**: Essential data structures for managing the game grid and undo/redo operations.

6. Methodology

The development process for the **Numbers Puzzle Game** involved the following steps:

1. **Project Selection**:
    - With guidance from our instructor, we chose a project that demonstrates the application of data structures and GUI programming.
    - The Numbers Puzzle Game was selected for its blend of logic, interactivity, and real-world relevance.
2. **Design**:
    - Representing the game grid as a **2D list**, where each element corresponds to a numbered tile.
    - Using **stacks** to manage the states for undo/redo functionality.
3. **Implementation**:
    - Developing the core logic for tile movement and victory conditions.
    - Creating a GUI with interactive buttons for each tile and difficulty level selection.
4. **Debugging and Testing**:
    - Resolving issues related to handling high-score states and the undo/redo stack.
    - Addressing difficulties with PyCharm, such as managing virtual environments and debugging.
5. **Documentation**:
    - Recording the results, challenges, and learning outcomes of the project.

7. Implementation

The **Numbers Puzzle Game** is implemented using Python's tkinter library for the GUI and core data structures like **lists** and **stacks** for game logic.

1. **Game Grid Representation**:
    - The grid is stored as a **2D list**, where each element represents a tile, and the empty tile is represented as an empty string ("").
    - Example:

[
[1, 2, 3, 4],
[5, 6, 7, 8],
[9, 10, 11, 12],
[13, 14, 15, ""]
]

```python
# Function to start the game with a selected level
def start_game(level):
    global size, grid, buttons, move_count_label, undo_stack, redo_stack, difficulty
    difficulty = level
    size = {"Easy": 3, "Medium": 4, "Hard": 5}[level]
    grid = [[(i * size + j + 1) for j in range(size)] for i in range(size)]
    grid[-1][-1] = ""  # Empty tile
    buttons = [[None for _ in range(size)] for _ in range(size)]
    for widget in root.winfo_children():
        widget.destroy()  # Clear the screen
    for i in range(size):
        for j in range(size):
            btn = tk.Button(root, text=grid[i][j], font=("Arial", 20), width=5, height=2,
                            command=lambda row=i, col=j: move_tile(row, col))
            btn.grid(row=i, column=j, padx=5, pady=5)
            buttons[i][j] = btn
    move_count = 0
    undo_stack = []
    redo_stack = []
    move_count_label = tk.Label(root, text=f"Moves: {move_count}", font=("Arial", 16))
    move_count_label.grid(row=size, column=0, columnspan=size)
    shuffle_button = tk.Button(root, text="Shuffle", font=("Arial", 14), command=shuffle_grid)
    shuffle_button.grid(row=size + 1, column=0, columnspan=size // 3, pady=10)
    undo_button = tk.Button(root, text="Undo", font=("Arial", 14), command=undo_move)
    undo_button.grid(row=size + 1, column=size // 3, columnspan=size // 3, pady=10)
    redo_button = tk.Button(root, text="Redo", font=("Arial", 14), command=redo_move)
    redo_button.grid(row=size + 1, column=2 * size // 3, columnspan=size // 3, pady=10)
```

2. **Tile Movement**:
    - Tiles can move into the empty space if they are adjacent to it (above, below, left, or right).

```python
# Function to move a tile
def move_tile(row, col):
    global grid, move_count, undo_stack, redo_stack, high_scores, difficulty
    empty_row, empty_col = [(r, c) for r in range(size) for c in range(size) if grid[r][c] == ""][0]
    if abs(row - empty_row) + abs(col - empty_col) == 1:
        undo_stack.append([row[:] for row in grid])
        redo_stack.clear()  # Clear redo stack after a new move
        grid[empty_row][empty_col], grid[row][col] = grid[row][col], grid[empty_row][empty_col]
        move_count += 1
        update_buttons()
        if is_solved():
            if move_count < high_scores[difficulty]:
                high_scores[difficulty] = move_count
                save_high_scores()
            messagebox.showinfo(
                "Congratulations!",
                f"You solved the puzzle in {move_count} moves!\n"
                f"High Score for {difficulty}: {high_scores[difficulty]} moves."
            )

```

3. **Undo/Redo Functionality**:
    - **Undo**: Saves the current grid state into a stack before making a move.

```
def move_tile(row, col):
                "Congratulations!",
                f"You solved the puzzle in {move_count} moves!\n"
                f"High Score for {difficulty}: {high_scores[difficulty]} moves."
            )


# Function to undo the last move
def undo_move():
    global grid, move_count, undo_stack, redo_stack
    if undo_stack:
        redo_stack.append([row[:] for row in grid])
        grid = undo_stack.pop()
        move_count -= 1
        update_buttons()
    else:
        messagebox.showinfo("Undo", "No more moves to undo!")
```

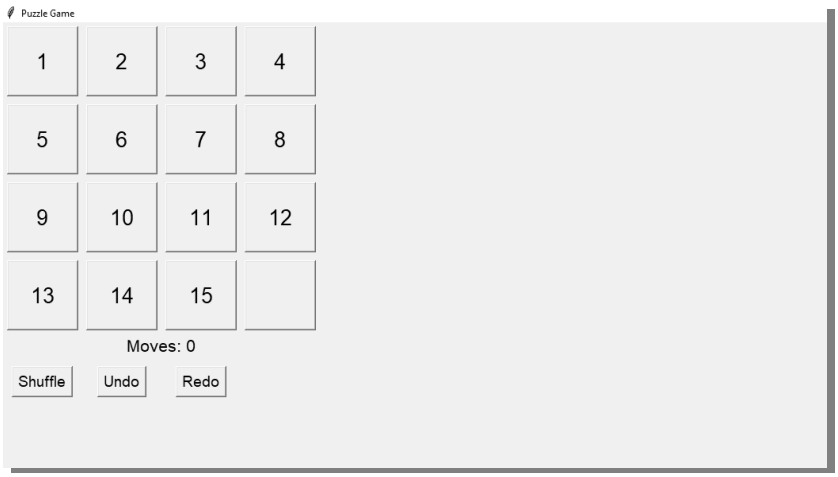
- **Redo**: Stores undone states into another stack, allowing players to reapply undone moves.

```
def undo_move():
        redo_stack.append([row[:] for row in grid])
        grid = undo_stack.pop()
        move_count -= 1
        update_buttons()
    else:
        messagebox.showinfo("Undo", "No more moves to undo!")


# Function to redo the last undone move
def redo_move():
    global grid, move_count, redo_stack
    if redo_stack:
        undo_stack.append([row[:] for row in grid])
        grid = redo_stack.pop()
        move_count += 1
        update_buttons()
    else:
        messagebox.showinfo("Redo", "No more moves to redo!")
```

4. **High-Score Tracking**:
    - Scores are tracked and saved to a file (high_scores.txt) for each difficulty level (Easy, Medium, Hard)

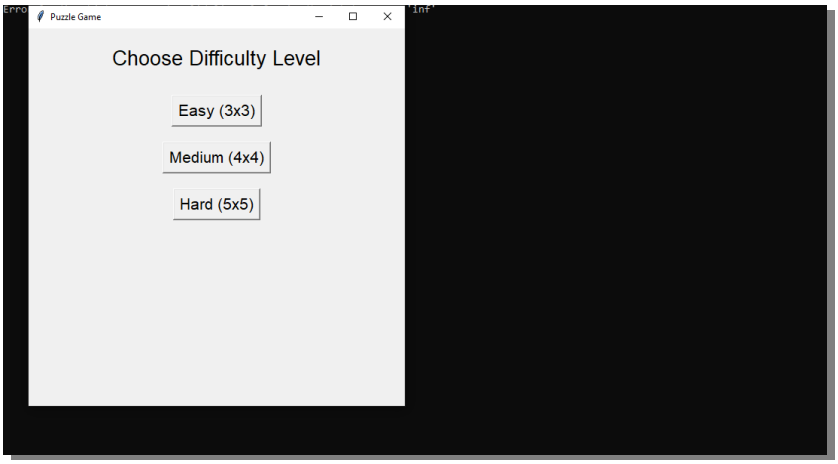
```


# Function to load high scores from file
def load_high_scores():
    global high_scores
    if os.path.exists("high_scores.txt"):
        try:
            with open("high_scores.txt", "r") as file:
                for line in file:
                    level, score = line.strip().split(":")
                    high_scores[level] = int(score)
        except Exception as e:
            print(f"Error loading high scores: {e}")
            high_scores = {"Easy": float('inf'), "Medium": float('inf'), "Hard": float('inf')}

```

5. **Graphical User Interface**:
    - tkinter buttons represent tiles. Clicking a button triggers the move logic.
    - The GUI includes a shuffle button, undo/redo buttons, and a move counter.

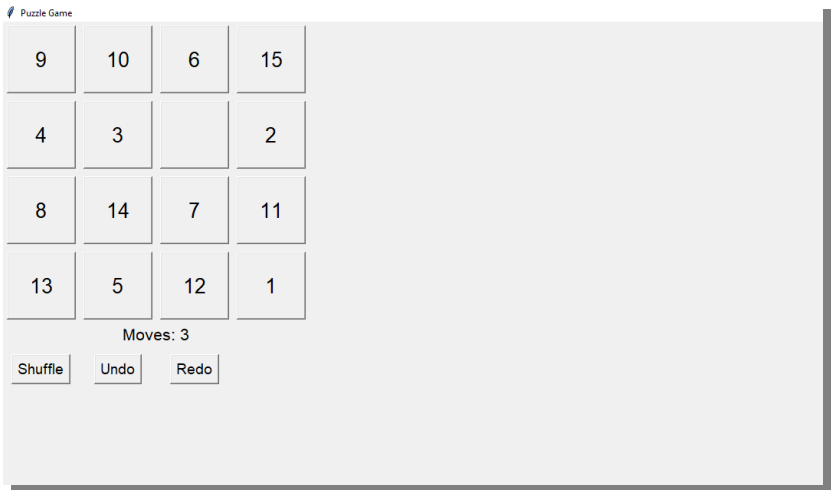


8. Game Working and Features

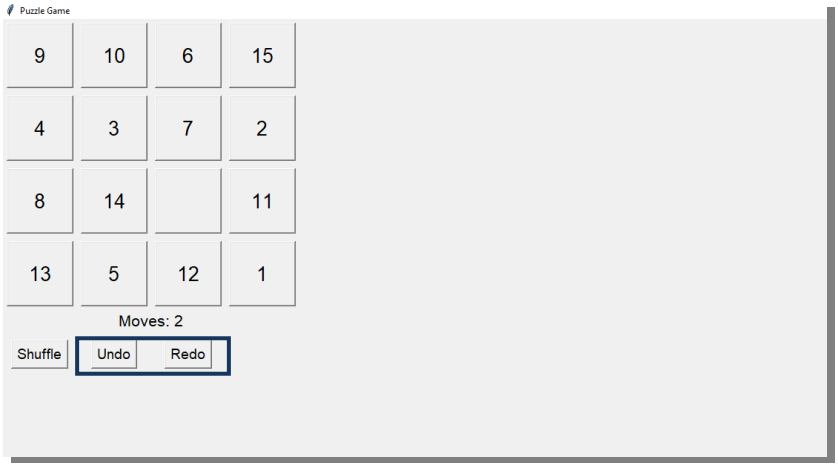
1. **Difficulty Levels**:
    - Players can choose from Easy (3x3 grid), Medium (4x4 grid), and Hard (5x5 grid).
    - The grid is shuffled randomly at the start of each game.



2. **Tile Interaction**:
    - Clicking on a tile adjacent to the empty space moves it into the empty spot.



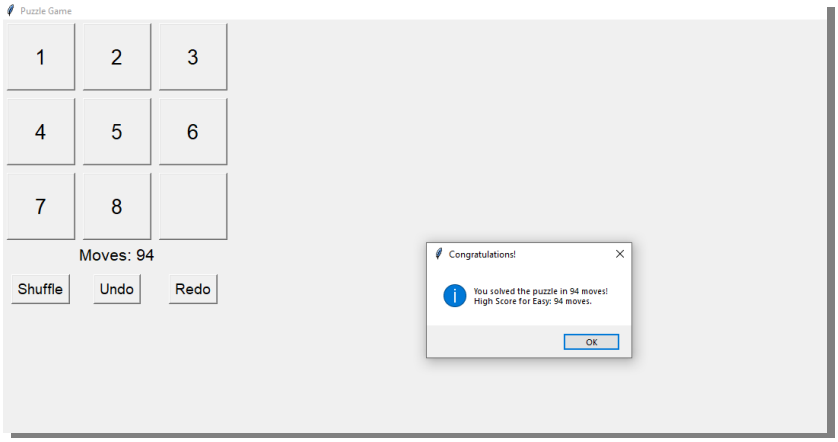
3. **Undo/Redo**:
    - Players can undo their last move or redo an undone move.



4. **High-Score System**:
    - Tracks the lowest number of moves required to solve the puzzle for each difficulty level.

5. **Victory Notification**:

- When the puzzle is solved, the game displays a congratulatory message along with the player's move count and high score.



9. Challenges

1. **Managing High-Score States**:
    - We initially faced issues saving and loading high scores correctly. This was resolved by refining our file-handling logic.
2. **Learning PyCharm**:
    - As beginners in Python, we encountered difficulties managing virtual environments and debugging in PyCharm. Our instructor's guidance helped us overcome these challenges.
3. **Complex State Management**:
    - Implementing undo/redo functionality required careful management of multiple game states.

10. Conclusion

The **Numbers Puzzle Game** is a practical demonstration of how data structures like **lists** and **stacks** can be applied in real-world scenarios. By combining programming logic with GUI development, the project highlights the importance of problem-solving in software development. This game serves as a foundation for more advanced projects in the future.

11. References

1. Goodrich, M. T., & Tamassia, R. (2015). *Data Structures and Algorithms in Python*. Wiley.
2. Python Software Foundation. (2023). "tkinter – Python Interface to Tcl/Tk." Retrieved from https://docs.python.org/3/library/tkinter.html.
3. Hamari, J., Koivisto, J., & Sarsa, H. (2014). "Does Gamification Work?" *IEEE Conference Proceedings*.