Project Report

Even-Odd Sudoku

Dated: 27/April/2025

Instructor

Miss Mehak Mazhar

AI-LAB BCY6A

Usman Islam K224708

Muhammad Bilal K224779

Ismail Ahmed K224792

1. Project Overview

- **Project Topic**:
  The project is an enhanced version of Sudoku, termed "Even-Odd Sudoku," where the added constraint is that no two consecutive numbers in any row, column, or 3x3 grid can have the same parity (odd/even). The game interface is developed using Pygame.
- **Objective**:
  The goal is to create an interactive and challenging Sudoku game that combines traditional Sudoku rules with additional even-odd constraints. The project also focuses on developing the game's UI, including features like puzzle generation, hints, and solving functionality.

2. Game Description

- **Original Game Background**:
  Sudoku is a logic-based puzzle game where the goal is to fill a 9x9 grid with digits such that each number appears once per row, column, and 3x3 sub-grid.
- **Innovations Introduced**:
  - **Even-Odd Constraints**: The game introduces the constraint that no two consecutive numbers in any row, column, or grid can have the same parity. This adds an additional layer of complexity to the traditional Sudoku rules.
  - **Gameplay Impact**: This constraint not only requires logical deduction but also introduces an extra challenge of managing number parity, thus increasing the game's difficulty.

3. AI Approach and Methodology

- **AI Techniques to be Used**:
  - The game doesn't include AI for opponents but uses a backtracking algorithm to solve the puzzle programmatically. The solving function uses recursion and visual updates to solve the puzzle step-by-step.

```
def solve_sudoku(board):
    # Find the first empty cell
    empty = find_empty(board)
    if not empty:
        return True  # Puzzle solved
    row, col = empty
    for num in range(1, 10):
        if is_valid(board, num, row, col):
            board[row][col] = num
            if solve_sudoku(board):
                return True
            board[row][col] = 0  # Backtrack
    return False  # No solution found
```

- **Heuristic Design**: The AI checks for the even-odd constraints while solving the puzzle, ensuring numbers fit both traditional Sudoku rules and the new parity-based conditions.

```
def solve_sudoku(board):
    # Find the first empty cell
    empty = find_empty(board)
    if not empty:
        return True  # Puzzle solved
    row, col = empty
    for num in range(1, 10):
        if is_valid(board, num, row, col):
            board[row][col] = num
            if solve_sudoku(board):
                return True
            board[row][col] = 0  # Backtrack
    return False  # No solution found
```

- **Complexity Analysis**:
  The solution employs a recursive backtracking approach to solve the puzzle, which, although efficient for a 9x9 grid, could become computationally intense with more complex constraints. The added even-odd constraint increases the difficulty in solving and requires more advanced heuristics.

4. Game Rules and Mechanics

- **Modified Rules**:
  - In addition to the standard Sudoku rules, the game includes a restriction where consecutive numbers (within the same row, column, or 3x3 grid) must differ in parity.
- **Winning Conditions**:
  - A player wins once the board is completely filled with valid numbers (respecting both Sudoku and even-odd constraints).
- **Turn Sequence**:
  - Players fill the grid cell-by-cell. Once a number is placed, the game checks if it respects the even-odd constraint as well as the usual Sudoku rules. Players can also request hints or solve the board automatically.

5. Implementation Plan

- **Programming Language**: Python
- **Libraries and Tools**:
  - **Pygame** for GUI and user interaction.
  - **Random** for puzzle generation and hint provision.

```
import random
def generate_puzzle():
```

```
# Generate a Sudoku puzzle with an even-odd constraint
puzzle = [[0 for _ in range(9)] for _ in range(9)]
# Puzzle generation logic...
return puzzle
```

- **Milestones and Timeline**:
  - **Week 1-2**: Finalize game design and rules.
  - **Week 3-4**: Implement puzzle generation, even-odd constraint logic, and backtracking solver.
  - **Week 5-6**: Develop and test the game interface.
  - **Week 7**: Refine game logic, test solving functionality, and add hints.
  - **Week 8**: Conduct final testing and prepare the report.