MASM Sudoku Project Report

Project Overview

The program creates a fully functional Sudoku game with a text-based user interface, allowing players to interact with a 9x9 grid through simple command-line inputs.

Technical Architecture

Core Components

1. **Data Structures**
   - board: A 9x9 grid stored as a one-dimensional array of 81 DWORDs
   - editable: A parallel array tracking which cells are user-editable
   - Various string constants for UI display
   - Input handling variables
2. **Key Procedures**
   - Game initialization procedures
   - Board manipulation functions
   - Input handling and validation
   - Game state verification
   - User interface rendering

Game Flow

The program follows this execution sequence:

1. Initialize the Sudoku board with a valid, complete solution

2. Remove a set number of values (40) to create the puzzle
3. Mark initial cells as non-editable
4. Enter the main game loop:
    - Display the current board state
    - Accept user input (row, column, value)
    - Validate move according to Sudoku rules
    - Update board if valid
    - Check for win condition
    - Repeat until the puzzle is solved or user exits

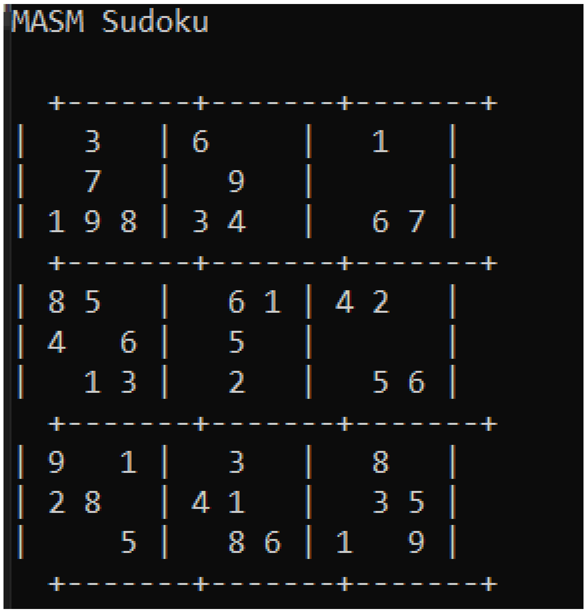
Implementation Details

User Interface

The game uses the Irvine32 library to provide a text-based interface. The DisplayBoard procedure renders the Sudoku grid with appropriate formatting:

- Horizontal and vertical lines separate the 3x3 boxes
- Numbers 1-9 are displayed in filled cells
- Empty cells are represented by spaces

Example board display:



```
MASM Sudoku


  +-------+-------+-------+
|   3   | 6     |   1   |
|   7   |   9   |       |
| 1 9 8 | 3 4   |   6 7 |
  +-------+-------+-------+
| 8 5   |   6 1 | 4 2   |
| 4   6 |   5   |       |
|   1 3 |   2   |   5 6 |
  +-------+-------+-------+
| 9   1 |   3   |   8   |
| 2 8   | 4 1   |   3 5 |
|     5 |   8 6 | 1   9 |
  +-------+-------+-------+
```

Move Validation

The ValidateMove procedure implements comprehensive validation of Sudoku rules:

1. Checks if the selected cell is editable
2. For new values (1-9), verifies that:
    - The value doesn't already exist in the same row
    - The value doesn't already exist in the same column
    - The value doesn't already exist in the same 3x3 box
3. Always allows clearing a cell (value 0)

This ensures all moves maintain a valid Sudoku state.

Win Condition

Two procedures verify the win condition:

1. CheckSolved: Confirms all cells are filled (no zeros)
2. VerifySolution: Validates that the solution follows Sudoku rules (no repeats in rows, columns, or boxes)

Only when both conditions are met is the win message displayed.

Algorithm Analysis

Random Number Generation

The program uses the Irvine32 RandomRange function to:

- Remove a random selection of numbers from the initial solution
- Generate random values (though this is unused in the final implementation)

Sudoku Validation

The solution validation algorithm uses bit manipulation for efficient checking:

- Uses a bitmask to track seen digits in each row, column, and box
- The bt (bit test) and bts (bit test and set) instructions provide efficient duplicate detection
- For each row, column, and 3x3 box, it ensures all numbers 1-9 appear exactly once

Limitations and Potential Improvements

1. **User Experience**
    - The current interface is functional but minimal
    - Adding color-coding for initial vs. player-entered values would improve readability
    - Supporting arrow key navigation would enhance usability

2. **Puzzle Generation**
    - Currently uses a static pre-defined solution with random removals
    - Could implement true procedural generation of unique puzzles
    - Difficulty levels could be added by adjusting the number of removed cells
3. **Performance Optimizations**
    - The code includes some unused procedures from an alternative implementation approach
    - Optimizing the validation checks could improve performance for large operations
4. **Error Handling**
    - Input validation could be enhanced with more specific error messages
    - A hint system could be implemented to assist players

Conclusion

This MASM Sudoku implementation demonstrates effective use of x86 assembly language to create an interactive game. The program successfully:

- Maintains and displays a valid Sudoku board
- Handles user input with appropriate validation
- Enforces game rules during play
- Verifies the solution when the board is completed

Despite the low-level nature of assembly language, the program achieves a clean separation of concerns through its modular procedure design. While there are opportunities for enhancement, the current implementation provides a complete and functional Sudoku gaming experience.