Sudoku Game

Priscilla Udomprasert
Computer Science,
College of Science
Auburn University at Montgomery
Montgomery, AL
pudompr1@aum.edu

Tyler Bodiford
Computer Science,
College of Science
Auburn University at Montgomery
Montgomery, AL
tbodifo4@aum.edu

Dendrick McGhee
Computer Science,
College of Science
Auburn University at Montgomery
Montgomery, AL
dmcghee2@aum.edu

ABSTRACT*—This project involves the design and implementation of a Sudoku program capable of generating, solving, and validating Sudoku puzzles. The program utilizes a backtracking algorithm to efficiently solve puzzles and includes functionality for creating puzzles of varying difficulty levels by selectively removing numbers from a completed grid. Input validation ensures user entries follow Sudoku rules, and a graphical or text-based interface allows user interaction. This project demonstrates core concepts in algorithm design, recursion, and user interface development, with potential applications in educational tools and puzzle games.

***KEYWORDS*—Sudoku, Algorithms, Quick sort, Scoring System**

I. INTRODUCTION

This group project is centered around the famous logic game - Sudoku. Despite appearing as a simplistic game, the creation of Sudoku can be either relatively simple or considerably complex. Currently, the plan is to implement the ability to generate a Sudoku board along with an updating leaderboard based off of a scoring system.

II. LITERATURE REVIEW

A. *Pencil and Paper Algorithm*

Solving Sudoku puzzles with algorithms is a popular problem that has a variety of solutions that have been implemented over time. A pencil-and-paper style algorithm utilized by [2] integrates preemptive sets and random choice to obtain a solution. The novelty of this algorithm is that it provides a solution that is independent of an electronic device and can be utilized by anyone who has access to a pencil and a sheet of paper. The downside of using this algorithm is that it is notably slower than using a computer-based algorithm.

B. *Harmony Search Algorithm*

Another algorithm for solving Sudoku puzzles is the Harmony Search algorithm. As detailed by [3], Harmony Search is a metaheuristic algorithm that mimics musicians' behavior such as random play and memory-based play. This algorithm was specifically designed to solve optimization problems and has fewer mathematical requirements in comparison to other algorithms for solving Sudoku puzzles. This algorithm also reduces the need for derivative information by using stochastic random searches. Overall, the Harmony Search algorithm allows a significant amount of flexibility for solving Sudoku puzzles, while also producing better solutions. Despite the reported benefits of using Harmony Search, the algorithm does have an issue with being inconsistent, as shown by [4]. This is due to the Harmony Search algorithm not accounting for a single optimal solution for solving a Sudoku puzzle, but instead the algorithm assumes that there are multiple solutions.

III. METHODOLOGY

For this project, we used a simple matrix for the game board. The board is generated as a 2D vector that is represented by a matrix [1]. Unlike traditional Sudoku, we chose to use "0" to represent the empty spaces that the player needs to fill. The number of empty spaces is determined by the player's difficulty choice. For the scoring system, we implemented a high score board with the "Quick Sort" algorithm.

A. *Matrix Creation*

1) *Generate a 3x3 Matrix*

```
// Generate a Sudoku grid with K empty cells
vector<vector<int>> sudokuGenerator(int k)
{

    // Initialize an empty 9x9 grid
    vector<vector<int>> grid(9, vector<int>(9, 0));

    // Fill the diagonal 3x3 matrices
    fillDiagonal(grid);

    // Fill the remaining blocks in the grid
    fillRemaining(grid, 0, 0);

    // Remove K digits randomly to create the puzzle.
    // This is a completed version of the puzzle
    SolvedPuzzle = grid;
    removeKDigits(grid, k);

    return grid;
}
```

2) Select Difficulty

```
switch (playerChoice)
{
case 1:
    k = 5;
    inloop = false;
    break;
case 2:
    k = 10;
    inloop = false;
    break;
case 3:
    k = 15;
    inloop = false;
    break;
case 4:
    while (true)
    {
        cout << "Enter how many missing digits you would like(1 - 40): ";
        cin >> customAmount;

        if (customAmount < 1 || customAmount > 40)
        {
            cout << endl << "That is not in range of the allowed custom amount!" << endl;
            continue;
        }
        else
        {
            k = customAmount;
            inloop = false;
            break;
        }
    }
    break;
default:
```

3) Place "0" Based on "K"

```
switch (playerChoice)
{
case 1:
    k = 5;
    inloop = false;
    break;
case 2:
    k = 10;
    inloop = false;
    break;
case 3:
    k = 15;
    inloop = false;
    break;
case 4:
    while (true)
    {
        cout << "Enter how many missing digits you would like(1 - 40): ";
        cin >> customAmount;

        if (customAmount < 1 || customAmount > 40)
        {
            cout << endl << "That is not in range of the allowed custom amount!" << endl;
            continue;
        }
        else
        {
            k = customAmount;
            inloop = false;
            break;
        }
    }
    break;
default:
```

B. *Puzzle Solving*

1) Obtain User Input

```
while (GameNotFinished)
{
    int userRow, userCol, userGuess;
    cout << "There are " << k << " missing digits from the Sudoku! Your current score is: " << score << endl;
    cout << "Enter your guess as [row][col][digit]" << endl;
    cout << "Row: ";
    cin >> userRow;
    cout << "Col: ";
    cin >> userCol;
    cout << "Number guess: ";
    cin >> userGuess;
```

2) Compare User Input

```
else
{
    if (SolvedPuzzle[userRow - 1][userCol - 1] == userGuess)
    {
        int points = 10;

        if (bonusPoints)
        {
            cout << "Correct guess! " << points+bpamount << " gained! + " << bpamount << " bonus for consecutive correct answers!" << endl;
            score = score + points;
            score = score + bpamount;
            bpamount = bpamount + 5;
        }
        else
        {
            cout << "Correct guess! 10 points gained!" << endl;
            score = score + points;
        }
        bonusPoints = true;
        this_thread::sleep_for(chrono::milliseconds(2000));
        system("cls");
        sudoku[userRow - 1][userCol - 1] = userGuess;
        printPuzzle(sudoku);
    }
```

C. *Score Keeping*

1) Generate Text File of High Scores

```
vector<int> highscores;
if (userExit == 'h' || userExit == 'H')
{
    string highscore;
    ifstream input_file;
    input_file.open("Highscores.txt");

    cout << endl << "  " << "HIGHSCORES" << endl << "==========" << endl;

    while (getline(input_file, highscore))
    {
        highscores.push_back(stoi(highscore));
    }
    input_file.close();

    quickSort(highscores, 0, highscores.size() - 1);

}
```

2) Use "Stack" Data Structure to Reverse Scores

```
stack<int> reverseScoresOrder;

for (auto score : highscores)
{
    reverseScoresOrder.push(score);
}
int list = 10;
if (highscores.size() < list)
{
    for (int i = 0; i < highscores.size(); i++)
    {
        cout << "    #" << i + 1 << ": " << reverseScoresOrder.top() << endl;
        reverseScoresOrder.pop();
    }
}
else
{
    for (int i = 0; i < list; i++)
    {
        cout << "    #" << i+1 << ": " << reverseScoresOrder.top() << endl;
        reverseScoresOrder.pop();
    }
}
```

IV. CONCLUSION

This project successfully achieved to create an entertaining program based on the popular game "Sudoku". By creating a matrix to use as the game board, players are able to efficiently solve puzzles based on their difficulty of choice. Players are also able to keep a continuous record of their highest scores. This was done by creating a text file that is able to be written to and read from. It also able to be displayed from highest to lowest by using a stack to reverse the order of scores.

V. REFERENCES

[1] Comment, et al. "Algorithm to Solve Sudoku: Sudoku Solver." *GeeksforGeeks*, 31 Jan. 2025, www.geeksforgeeks.org/sudoku-backtracking-7/. Accessed 23 Mar. 2025.

[2] Crook, J. F. "A pencil-and-paper algorithm for solving Sudoku puzzles." Notices of the AMS 56.4 (2009): 460-468.

[3] Geem, Zong Woo. "Harmony search algorithm for solving sudoku." Knowledge-Based Intelligent Information and Engineering Systems: 11th International Conference, KES 2007, XVII Italian Workshop on Neural Networks, Vietri sul Mare, Italy, September 12-14, 2007. Proceedings, Part I 11. Springer Berlin Heidelberg, 2007.

[4] Weyland, Dennis. "A critical analysis of the harmony search algorithm—How not to solve sudoku." Operations Research Perspectives 2 (2015): 97-105.