Test Document

2. Test Plan ("How to verify that the system does what it should do?")
    - at least teamSize * 1 page (single-spaced, 12-pt font, 1-in margins) [for example, at least 3 pages for a team with 3 members]
    - for each required feature/behavior in the Requirement Document, discuss test cases to verify the feature/behavior.
    - Generally, test cases are designed to find bugs. Hence, besides the "usual" input values, "ununsal" input values should be consdiered as well.
    - [e.g. page 53 of IEEE Standard for Software Test Documentation]

Level Test Case Outline (full example)
1. Introduction (once per document)
1.1. Document identifier
1.2. Scope
1.3. References
1.4. Context
1.5. Notation for description
2. Details (once per test case)
2.1. Test case identifier
2.2. Objective
2.3. Inputs
2.4. Outcome(s)
2.5. Environmental needs
2.6. Special procedural requirements
2.7. Intercase dependencies
3. Global (once per document)
3.1. Glossary
3.2. Document change procedures and history

1. Introduction

Sudoku Solvers will be making an application that allows the user to play algorithmically generated sudoku puzzles of different difficulty levels as well as a solver that will be able to solve the current puzzle for the user or give the user hints for some cells that they are wanting help with. It will do this with a variety of algorithms, each algorithm for the solver and the generator will need test cases to make sure that they can function correctly and in a timely manner. Along with testing the algorithms, the GUI elements will need to be tested, such as the menu buttons and the interactive sudoku board.
1.2

1.1 Document identifier

This level test case aims to provide useful information and procedures to take during our later testing phase of the project. This is a draft and subject to change with basic test procedures that will be refined further down the line.

2. Details

Sudoku Solvers aims to design puzzle generation, auto-solver, hint system, difficulty, save system, and variations.

Test Case ID: SS-TC-Solver-001
Title: Test that the system returns a correct puzzle
Test Steps:
Solver:

1. Run the solver on a board missing one number
2. Run the solver on a board with most of its information
3. Run the solver on a standard board
4. Run the solver on an empty board
5. Run the solver on an incorrect board

The tests will provide insights on how fast it takes to find solutions at different stages and make sure the solver accurately understands and utilizes the rules of the puzzle to create its solutions.
Prerequisites: The sudoku puzzle has not been completed yet
Expected result: The puzzle is displayed with all cells filled in and correctly

Test Case ID: SS-TC-Generate-002
Title: Test that system generates a puzzle with a unique solution
Test Steps:
Puzzle Generation:

1. Randomly generate numbers onto a board
2. Run a solver algorithm to prove it can be solved
3. Run a solver algorithm to prove that there exists only one unique solution

These three tests, in sequence will find bugs in board instantiation, filling tiles, solving, and making sure a board is proper and has only one solution
Prerequisites: The system is not performing other features simultaneously
Expected Result: The generated puzzle has at least minimum cells filled in to have the puzzle only have one unique solution

Test Case ID: SS-TC-Hints-003
Title: Test hint provides a number for an unfilled cell
Hint Systems

1. Test clicking the Hint interface
2. Test generated hint is valid in the rules of Sudoku
3. Test generated hint is not an already solved tile
4. Test generated hint is part of the board's unique solution by using it on a board with a known solution
5. Test generating a hint for every empty space perfectly solves the given board

These tests will make sure the system is responsive, accurate, and helpful
Preconditions: The puzzle has not been completed and there is one number missing from the board
Expected: The system inserts a number not already on the board into an unfilled cell that is correct
Variants:

1. Display an empty board
2. Display a solved board
3. Generate a board
4. Generate hints on a board
5. Run a solver algorithm on a generated board

Testing the displays makes sure that all of the visual differences and information is accurately displayed and handled. Generating a board makes sure there are no bugs in the data representation or generator algorithms. Generating hints and solving a board make sure the rules are perfectly followed and properly applied at each step.
Difficulty Test Cases:
Since difficulty can be subjective, we will have play testers with a variety of Sudoku skills give their input on where we should draw the line on how easy or hard a certain board with x amount of starting numbers is, and then categorize each difficulty from there. We know that the minimum amount of numbers that a sudoku board needs to have 1 solution is 17, so 17 will be the upper bound of our hard difficulty.