SE 3XA3: Test Plan
Sudoku Solver

Team 08, SudoCrew
Rashad A. Bhuiyan (bhuiyr2)
Kai Zhu (zhuk2)
Stanley Chan (chans67)

March 11, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
2022-02-28	0.0	Created Test Plan; started on General Information
2022-03-09	0.1	Updated Plan and Appendix, started System Test Description
2022-03-11	0.2	Updated Sections 3-6

1 General Information

1.1 Purpose

The purpose of this document is to describe the testing, validation, and verification procedures that will be implemented for our Sudoku Solver program. The majority of the unit testing will be accomplished through PyTest as our program is primarily written using Python. Structural testing will be done using individually-created testing classes for proper analysis of branching. Non-functional requirements will primarily be realized through manual testing as there is a heavy emphasis on visualization of our project. System testing is done before proof-of-concept demonstration, revision 0 demonstration, and the final demonstration.

1.2 Scope

This test plan provides a basis for testing the functionality of the extended implementation and every new addition to the original Sudoku Solver algorithm by TechWithTim. Since the extended implementation involves creating a web application and methods of playing the game, the scope of testing covers web-application navigation and management, Sudoku generation and solution algorithms, and image recognition algorithms through machine learning. This document provides testing methodologies that covers the scope of the program as well as outlining all methods and tools used for testing.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Naming Conventions and Terminology**

Terminology	Meaning
PoC	Proof of Concept, one of the deliverables of the 3XA3 project
SRS	Software Requirements Specification, the document outlining all requirements for this program
GUI	graphical user interface, the front-end of the program that users can view and interact
Structural Test	Testing that focusses on the internal structure and branching of the software
Functional Test	Testing derived from the SRS
Dynamic Test	Testing that involves test cases running during execution
Static Test	None-code testing done through code inspection
Manual Test	Testing that is done manually by people
Automated Test	Testing that is done using testing software such as PyTest
Unit Test	Testing that focusses on individual functions and methods
System Test	Testing the entire system as a whole rather than individual components
UI	User interface, the interface that allows for user interaction with the system.
PyTest	A Python library designed to help create tests for Python code.
Mocha	A JavaScript test framework.
MNIST	A Modified National Institute of Standards and Technology data set with a large number of images of handwritten digits.

1.4 Overview of Document

This document contains all information regarding the testing plan for the Sudoku Solver project. An overall plan will be addressed through a schedule as well as creation of a testing team. Furthermore, every functional and non-functional requirement from the SRS will be addressed and tested using various form of blackbox and whitebox testing. PoC testing will cover a specific sample of methods and will highlight some key differences of implementation between the original project and the current implementation.

2 Plan

2.1 Software Description

The purpose of this software application is to provide a comprehensive suite of tools for generating, recognizing, and solving Sudoku puzzles. The application will provide a web-based front-end with an intuitive interface to cater to users of different technical abilities on most modern hardware. Computer vision is also utilized to improve ease of use, by directly interfacing puzzles from print-media to the application.

2.2 Test Team

The team responsible for testing consists of Rashad Bhuiyan, Stanley Chan, and Kai Zhu.

2.3 Automated Testing Approach

Automated unit testing will be performed using PyTest and Mocha for the Flask and JavaScript components of the application, respectively. Additionally, the number recognition system uses a trained neural network model, which can be automatically tested against a test data set for prediction accuracy.

2.4 Testing Tools

PyTest will be used for the unit test of the Flask back-end application. Mocha will be used for the unit test of front-end JavaScript.

2.5 Testing Schedule

Detailed testing schedule and responsibilities are included in the Project Gantt chart, available at https://gitlab.cas.mcmaster.ca/bhuiyr2/sudokusolver_l02_grp08/-/raw/main/ProjectSchedule/Gantt_Sudoku.pdf

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Homepage Navigation

Display Buttons

1. FS-DB-1

   Type: Functional, Dynamic, Manual

   Initial State: Looking at homepage of Flask app

   Input: Click Manual or Image Input button

   Output: Appropriate screen will be shown depending on input type specified

   How test will be performed: The button that specifies either 'Manual Input' or 'Image Input' will be pressed to see if it will redirect the user to the appropriate page that allows them to either upload an image or manually input a Sudoku board.

2. FS-DB-2

   Type: Functional, Dynamic, Manual

   Initial State: Looking at homepage of Flask app

   Input: Click play a game button

   Output: Sudoku game for user to play will be loaded

   How test will be performed: The button that says 'Play a Game' will be pressed to check and see if it will redirect the user to a separate page where they may play a game of Sudoku.

3. FS-DB-3

   Type: Functional, Dynamic, Manual

   Initial State: Old browser

   Input: Enter Sudoku Solver website path

   Output: Error message stating browser mismatch

   How test will be performed: The website will attempt to be launched using an old and/or deprecated browser, such as Internet Explorer, where an error message will be shown stating a mismatch of browser and website.

3.1.2 Image Upload

Initial Upload

1. FS-IU-1

   Type: Functional, Dynamic, Manual

Initial State: Sudoku Image input page with image uploaded

Input: Submit image

Output: Loading symbol will be shown

How test will be performed: The change in website state will be tested by uploading and submitting a Sudoku board to see if a loading icon is displayed.

2. FS-IU-2

   Type: Functional, Dynamic, Manual

   Initial State: Sudoku Image input page with image uploaded

   Input: Submit Sudoku board

   Output: Converted image to array

   How test will be performed: The functions used to interpret the image will be tested by giving an image of Sudoku board and checking output flags to see if the image has been able to be converted to an array.

Image Interpretation

1. FS-II-1

   Type: Functional, Dynamic, Manual

   Initial State: Loading icon after image upload

   Input: Invalid image was originally given

   Output: Error message indicating that image could not be interpreted

   How test will be performed: An invalid image will be uploaded to check whether the function can successfully recognize that it is invalid and output an error message to the user that the image is invalid or could not be recognized. This checks code coverage of the invalid branching path for the image recognition algorithm.

2. FS-II-2

   Type: Functional, Dynamic, Manual

   Initial State: Loading icon after image upload

   Input: Valid image was given

   Output: Array of Sudoku numbers to solver algorithm

   How test will be performed: The function to interpret a Sudoku board will be given a valid Sudoku board to check whether it will send the correct integer array to the Sudoku solver algorithm. This checks code coverage of the valid branching path for the image recognition algorithm.

Solution Output

1. FS-SO-1

   Type: Functional, Dynamic, Manual

   Initial State: Loading icon after image submission

   Input: Legal image with a possible solution

   Output: Display solution to user

   How test will be performed: The overall upload system will be tested using a valid Sudoku board with a known solution where the generated Sudoku solution will be compared to the actual solution to determine whether the interpretation and solution algorithms work effectively. This checks code coverage of the valid branching path for the Sudoku solution algorithm.

2. FS-SO-2

   Type: Functional, Dynamic, Manual

   Initial State: Loading icon after image submission

   Input: Legal image with no solution

   Output: Notify user about no solution

   How test will be performed: The overall upload system will be tested using an invalid Sudoku board with no known solution to check whether the image interpretation algorithm and the solution algorithm will notify the user that there is no solution possible. This checks code coverage of the invalid branching path for the Sudoku solution algorithm.

3.1.3 Manual Input

Board Display

1. FS-BD-1

   Type: Functional, Dynamic, Manual

   Initial State: Homepage of Sudoku Solver Flask app

   Input: Click on Manual Input button

   Output: Empty Sudoku board on Manual Input page

   How test will be performed: The navigation to the Manual Input page will be tested alongside whether an empty grid is shown for the user to input their numbers of the Sudoku board.

2. FS-BD-2

   Type: Functional, Dynamic, Manual

   Initial State: Manual Input page with empty Sudoku board

Input: Number in cell

Output: Number appears in cell

How test will be performed: The manual input function will be tested by attempting to place the numbers 1-9 in any empty cell of the Sudoku board.

Solution Verification

1. FS-SV-1

   Type: Functional, Dynamic, Manual

   Initial State: Semi-populated Sudoku board

   Input: Number that breaks Sudoku rules

   Output: Notification about invalid input

   How test will be performed: The manual input will be tested to see whether it can reject bad input by putting in an invalid input into a row/column/box due to a repeating number and checking if the function notifies the user about an invalid move. This checks code coverage of the invalid input branching path for the manual input function.

2. FS-SV-2

   Type: Functional, Dynamic, Manual

   Initial State: Semi-populated board

   Input: Submit button

   Output: Loading icon

   How test will be performed: The submission function will be tested by filling out the manual input with a legal board and pressing the submit button to see if a loading icon will appear to indicate that the given board is being processed.

3. FS-SV-3

   Type: Functional, Dynamic, Manual

   Initial State: Loading icon after manual submission

   Input: Legal board with no solution

   Output: Notify user about no solution

   How test will be performed: The overall upload system will be tested using an invalid Sudoku board with no known solution to check whether the solution algorithm will notify the user that there is no solution possible. This checks code coverage of the invalid branching path for the Sudoku solution algorithm.

4. FS-SV-4

   Type: Functional, Dynamic, Manual

Initial State: Loading icon after manual submission

Input: Legal board with possible solution

Output: Display solution to user

How test will be performed: The overall upload system will be tested using a valid Sudoku board with a known solution to check whether the solution algorithm will produce the same solution as the previously known solution. This checks code coverage of the valid branching path for the Sudoku solution algorithm.

3.1.4 Play Sudoku

Board Generation

1. FS-BG-1

   Type: Functional, Dynamic, Manual

   Initial State: Sudoku Gameplay page

   Input: Click button to start game

   Output: Loading symbol shown

   How test will be performed: Ensure that when the user presses the button, the website displays a loading symbol.

2. FS-BG-2

   Type: Functional, Dynamic, Automated

   Initial State: Board is valid

   Input: Completed valid Sudoku board

   Output: true boolean value, indicating that the board is valid

   How test will be performed: A completed valid Sudoku board will be used as an input into the function. The test will ensure that the output of the function will be true.

3. FS-BG-3

   Type: Functional, Dynamic, Automated

   Initial State: Board is valid

   Input: Completed invalid Sudoku board

   Output: false boolean value, indicating that the board is invalid

   How test will be performed: A completed Sudoku board that does not follow the Sudoku rules will be used as an input into the function. The test will ensure that the output of the function will be false.

4. FS-BG-4

   Type: Functional, Dynamic, Automated

Initial State: Board is valid

Input: Incomplete valid/invalid Sudoku board

Output: an exception should be raised, indicating that the board is incomplete

How test will be performed: An incomplete Sudoku board will be used as an input into the function. The test will ensure that the function will raise an exception.

5. FS-BG-5

   Type: Functional, Dynamic, Automated

   Initial State: Board has no solutions

   Input: No solution board

   Output: false boolean value, indicating that the board has no solutions

   How test will be performed: A Sudoku board with no solutions will be used as input for the function. The test will ensure that the function outputs false.

6. FS-BG-6

   Type: Functional, Dynamic, Automated

   Initial State: Board has no solutions

   Input: Board with solution(s)

   Output: true boolean value, indicating that the board has solutions

   How test will be performed: A Sudoku board with one or more solutions will be used as input for the function. The test will ensure that the function outputs true.

7. FS-BG-7

   Type: Functional, Dynamic, Automated

   Initial State: Board is unique

   Input: Non-unique board (multiple solutions)

   Output: false boolean value, indicating that the board is not unique

   How test will be performed: A Sudoku board with more than one solution will be used as input for the function. The test will ensure that the function outputs false.

8. FS-BG-8

   Type: Functional, Dynamic, Automated

   Initial State: Board is unique

   Input: Unique board (one solution)

   Output: true boolean value, indicating that the board is unique

   How test will be performed: A Sudoku board with only one solution will be used as input for the function. The test will ensure that the function outputs true.

9. FS-BG-9

   Type: Functional, Dynamic, Automated

   Initial State: Board is unique

   Input: No solution board

   Output: exception should be raised, informing the user that the board has no solutions

   How test will be performed: A Sudoku board with no solutions will be used as input for the function. The test will ensure that the function raises an exception, and informs the user that the Sudoku board has no solutions.

10. FS-BG-10

    Type: Functional, Dynamic, Automated

    Initial State: No board generated

    Input: Number of hints

    Output: A randomly generated unique Sudoku board

    How test will be performed: Generate multiple Sudoku boards with the function, and test that each board generated has a solution and is unique (with the aforementioned functions). Also iterate through each board and ensure that the number of hints generated matches the input.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

Homepage Display

1. SS-HD-1

   Type: Non-Functional, Static, Manual

   Initial State: Homepage of the Sudoku solver Flask app

   Input: Users from testing group will view the homepage

   Output: Users will give feedback on button layout

   How test will be performed: The users from the testing group will each load the Homepage on their own device to check the buttons and see if they look distinct and separate from each other.

2. SS-HD-2

   Type: Non-Functional, Static, Manual

   Initial State: Homepage of the Sudoku solver Flask app

   Input: Users from the test group will change the browser size

   Output: The Homepage scales accordingly, making UI changes based on the dimensions of the window

How test will be performed: The users from the testing group will each load the Homepage on their own device and change the browser's size to see whether the page scales accordingly and changes views depending on screen dimensions.

User Interface Design

1. SS-UID-1

   Type: Non-Functional, Static, Manual

   Initial State: Homepage of the Sudoku solver Flask app

   Input: Users from the test group view the web app

   Output: Users give feedback on the overall UI

   How test will be performed: The users from the testing group will each load the Homepage on their own device and explore the web app on their own, taking note of each UI element and whether it is appropriate.

2. SS-UID-2

   Type: Non-Functional, Static, Manual

   Initial State: Homepage of the Sudoku solver Flask app

   Input: Users with mild forms of colour-blindness view the web app

   Output: Users give feedback on the accessibility of the web app

   How test will be performed: The users officially diagnosed with some forms of colour-blindness will load the Homepage on their own device and explore the web app on their own, taking note of whether every page is accessible and appealing to them.

3.2.2 Usability and Humanity

Ease of Use

1. SS-EU-1

   Type: Non-Functional, Static, Manual

   Initial State: Manual input page or Image Upload page

   Input: Users from test group follow instructions of respective page

   Output: System accepts provided Sudoku board

   How test will be performed: The users from the testing group will check both the Manual Input and the Image Upload pages by following the given instructions about filling out the Sudoku board and submitting them in order to get a valid solution, using a set of Sudoku boards that are both valid and invalid.

2. SS-EU-2

   Type: Non-Functional, Static, Manual

Initial State: Sudoku gameplay page

Input: Users from test group start playing the game

Output: Users play Sudoku and are given solution at the end

How test will be performed: The users from the testing group will attempt to play a game of Sudoku on the Sudoku gameplay page and check whether their given solution is correct and whether the game then provides the actual solution.

Ease of Learning

1. SS-EL-1

   Type: Non-Functional, Static, Manual

   Initial State: How to Play Sudoku page

   Input: Users from the test group read the page

   Output/Result: Users understand rules of Sudoku

   How test will be performed: The users from the testing group will view the instructions of Sudoku shown on the How to Play page and see if they can understand the rules of the game.

3.2.3 Performance

Speed

1. SS-S-1

   Type: Non-Functional, Static, Manual

   Initial State: Upload image page

   Input/Condition: User from test group submits an image

   Output/Result: Loading icon will take MAX_UPLOAD_TIME to upload

   How test will be performed: The users from the testing group will submit an image to be recognized using the image recognition function and will time whether it takes more than MAX_UPLOAD_TIME.

2. SS-S-2

   Type: Non-Functional, Static, Manual

   Initial State: Verify upload page

   Input/Condition: User from test group submits an image

   Output/Result: Loading icon will take MAX_SOLUTION_TIME to upload

   How test will be performed: The users from the testing group will submit an image to be recognized using the Sudoku solution function and will time whether it takes more than MAX_SOLUTION_TIME.

Capacity

1. SS-C-1

   Type: Non-Functional, Static, Manual

   Initial State: Upload Image page

   Input/Condition: User from test group uploads image

   Output/Result: Image is not stored locally

   How test will be performed: The user from the testing group will upload an image and then will check their local device memory to see if the image was saved.

3.2.4 Security

File Integrity

1. SS-FI-1

   Type: Non-Functional, Static, Manual

   Initial State: Upload Image page

   Input/Condition: User from test group uploads image that is too large

   Output/Result: Error is given about file size

   How test will be performed: The user from the testing group will upload an image that is too large to be used and will check whether they are notified about the file size being too large.

Access, Privacy, and Immunity

1. SS-API-1

   Type: Non-Functional, Static, Manual

   Initial State: Solution of Sudoku board

   Input/Condition: User from test group attempts to edit shown solution

   Output/Result: User cannot edit solution

   How test will be performed: The user from the testing group will attempt to edit the given solution to either an image upload or a manual board submission to check on whether they can edit the solution.

4 Tests for Proof of Concept

The primary feature in the PoC is the application's ability to recognize a Sudoku image and display the recognized digits on a web-based front-end. Therefore, tests for the PoC will focus on image extraction, digit recognition, and front-end presentation.

4.1 Board image extraction

This portion of the testing focuses on the extraction of Sudoku board and cell image data from a set of 10 photographs of varying perspective, brightness, and clarity. The output is compared manually with the expected output in each case, since automatic testing of computer vision results is an advanced topic beyond the scope of this project.

1. poc-ie1

   Type: Functional, Dynamic, Manual

   Initial State: SudokuCV object is instantiated

   Input: 10 different photographs of Sudoku boards, and 3 photographs without Sudoku

   Output: flattened and cropped images of Sudoku boards, or raise error if no board is detected.

   How test will be performed: SudokuCV object is supplied with input image through the recognize() method with the show_image parameter set to true. This displays the input and output images for manual validation.

2. poc-ie2

   Type: Functional, Dynamic, Manual

   Initial State: SudokuCV object is instantiated

   Input: 3 photographs without a fully visible Sudoku puzzle

   Output: error string indicating that no Sudoku board can be extracted.

   How test will be performed: Three images with an assortment of problems are supplied to test the error handling of the SudokuCV recognize() method. Specifically, one input image does not have a Sudoku board, another is too small to extract any features from, and a third cropped to include only a part of a board.

3. poc-ie3

   Type: Functional, Dynamic, Manual

   Initial State: SudokuCV object is instantiated

   Input: 10 different photographs of Sudoku boards

   Output: An array of 81 images each cropped from a cell on the 9x9 Sudoku board

   How test will be performed: array of numpy output is displayed using the Opencv imshow() method and manually validated against the input image.

4.2 Digit recognition

1. poc-dr1

   Type: Non-functional, Dynamic, Automatic

   Initial State: a neural network model is trained from the handwritten and printed digits data set

   Input: 10000 MNIST handwritten numbers test set

   Output: percentage accuracy of the model prediction

   How test will be performed: cvtraining module compares the labels of the input data set against the prediction made by the trained model and returns the rate of correct predictions.

2. poc-dr2

   Type: Non-functional, Dynamic, Manual

   Initial State: SudokuCV object is instantiated with trained model

   Input: 10 photographs of Sudoku boards with handwritten and printed numbers

   Output: an integer array of size 81 with recognized digits

   How test will be performed: SudokuCV object supplies the array of recognized digits (0 for empty cells) and a corresponding array of confidence rates for each index. Digits with a confidence greater than 75% are displayed and manually compared against the input images to calculate recognition accuracy.

4.3 Front-end presentation

1. poc-fp1

   Type: functional, Dynamic, manual

   Initial State: Flask app is running using an initialized SudokuCV object

   Input: 10 photographs of Sudoku boards with handwritten and printed numbers

   Output: HTML page displaying extracted board image and table of recognized digits

   How test will be performed: The photographs are uploaded one at a time through the web-based GUI, the output table is manually compared to the original photograph to validate that digits are displayed correctly in the Sudoku grid format.

2. poc-fp2

   Type: functional, Dynamic, manual

   Initial State: Flask app is running using an initialized SudokuCV object

   Input: 3 invalid Sudoku photographs (too small, no Sudoku puzzle, and empty grid)

   Output: HTML page displaying error and suggested solution

How test will be performed: The photographs are uploaded one at a time through the web-based GUI, the output is manually compared against the expected error for each input image.

5 Comparison to Existing Implementation

There are 5 tests that compare the existing implementation of the program to the current implementation. Please refer to:

- FS-SO-1 in Tests for Functional Requirements
- FS-SO-2 in Tests for Functional Requirements
- FS-SV-3 in Tests for Functional Requirements
- FS-SV-4 in Tests for Functional Requirements
- SS-S-2 in Tests for Nonfunctional Requirements

6 Unit Testing Plan

6.1 Unit testing of internal functions

The internal functions of the system mostly include the Sudoku solver algorithm itself, the Sudoku board generator, and the image recognition functions. To test such functions, it is simple enough to give these functions an input and ensure that the output is what is to be expected from that given input. Since most of the internal functions is written in Python and JavaScript, we will be utilizing the PyTest and Mocha automatic testing frameworks for each programming language respectively. Since the board generator creates random boards, it is impractical generate and test every possible board. Therefore, unit testing for Sudoku board generation will create a reasonably sized set of boards, between 100 to 200. Individual functions that check for solvability and uniqueness will be called upon these generated boards to confirm the reliability and validity of the generation system.

6.2 Unit testing of output files

Unit testing the output files of the system will be performed by inputting various Sudoku boards via images (or manual input) into the web application, and ensuring that the outputted solution follows the Sudoku rules and is complete. The outputted Sudoku boards can be scraped from the web application, transformed into an array format, and be used as an input into a function that tests whether a complete Sudoku board follows the Sudoku rules.

7 Appendix

7.1 Symbolic Parameters

The definition of the requirements will likely call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

$MAX\_UPLOAD\_TIME = 3$
$MAX\_SOLUTION\_TIME = 2$