Test Plan

Software Testing

Name: Scott Adamson
Class#: Software Testing

February 3, 2023

1 Test Planning

There are a few key areas which require careful testing if we are to ensure proper functioning of the Sudoku game. This section aims to highlight a few of them and discuss some of the implications of beginning to test these.

1.1 User Input

As this is a multiplayer Sudoku game, where users collaboratively complete generated Sudoku puzzles, it is extremely important to the functioning of the game that users cannot enter an incorrect input. Input must conform to a valid Sudoku described below. This means we should be willing to divert some computational and time resources toward this goal, as the game will not be in a correct state otherwise. In the games current stage of development it is not possible for a user to actually input a value, but this is not necessary in order for us to implement the tests required to achieve this and do a small amount of scaffolding to properly test this. The test would require a reference completed version of the Sudoku in order to know if user input is correct. This test must be carried out each time an input is placed. The time complexity of comparing grid cells of a 2D array would be $O(n^2)$, this means the runtime of the test is proportional to the grid size. The Sudoku grid will always be 9x9 and so the time implications of running such a test are not significant. In general it is recommended to include methods for checking these requirements rather than using unit testing during actual gameplay. Valid input is any digit 1-9. If invalid, don't allow the input. It is also possible to prevent this with buttons of the numbers so as only correct inputs can be made. This way cause UI issues.

1.2 Integration

The are a few components which may require communication within our game and as such some integration testing is required. This ensures our project lays within a certain structural layout. For example, the scripts generating Sudokus have to integrate well to the game object containing the Sudoku grid that users can interact with. It is also important the users can see the changes made by the opposing user. As discussed in the Integration and Component-based Software Testing lesson, its best to trial this early and also provides us with the ability to test model other scenarios and aspects of the game. For that reason, I rank this highly among things to be tested. At the time of carrying out this test, the Sudoku generating was yet to be completed and so it may be necessary to provide some scaffolding code to simulate the proper output of the game to the UI.

1.3 Valid Sudoku

In order for a Sudoku to be considered valid there are several requirements. These include:

- Number of cells: A valid Sudoku grid must have 9 rows and 9 columns, 81 cells total

- Unique numbers : Each row, column and 3x3 box must contain only one instance of each number 1-9
- Symmetry: A valid Sudoku must be symmetrical.
- No empty cells: a completed Sudoku must have no empty cells.

As there are a few requirements for a valid Sudoku, all adding complexity to the problem, it is important that we analyse many generated grids from the algorithm in order to be confident in the programs ability to perform. We should perform a systematic test, in which we carefully select the cases most likely to cause issues with a grid validator. This is so as we can better ensure we are handing all possible types of cases we may see. There are an extremely high number of possible valid Sudoku grids and so this would be impossible to say that the algorithm can generate a valid grid 100% of the time, so a grid should be tested for validity after generation if we are to be certain we can deliver a valid Sudoku to the end-user. We should also be informed if an invalid grid is generated, so we can analyse its edge-case.

1.4 Removal of cells

In order to take a valid Sudoku grid and make it such that a user has to complete it, we need to remove some cells from the grid. It is important that this is done correctly in order to properly verify that a user is completing the Sudoku in the correct way. This should be tested before deployment of the code under a large volume of grids to ensure this is tested appropriately. Testing must ensure that all numbers that aren't 0 (0 meaning the cell is hidden) are the same after each removal. This test can actually be considered the same as the one for user input, as the main idea is the same.

1.5 Two Players

This game is a two-player app and so the game should ensure that a maximum of two players can join the game, and after the first player has joined, the screen should switch to a waiting screen until such time as another player has joined. As the game is still in development, it is necessary for us to scaffold the game lobby so as we can test users joining the game and we have a suitable environment in which to carry on developing other areas.

1.6 Unique Solution

Is is also extremely important that a Sudoku with some cells removed contains only 1 valid solution, so as to avoid confusion of the puzzle solver. However, this is an extremely complex task, requiring the development of Dancing Links/Algorithm X tasked with solving exact cover problems. Once this has been developed, it is very straightforward to test, however this is not a task which may be done at this current time. This involves some scaffolding, which is described in more detail in the scaffolding section.

1.7 Performance

As this is a Multiplayer game, user-interaction and performance of the game is a very important aspect. This is difficult to test (more explained in the next section) and will require some degree of learning with real user behavior, but basic manual testing can be done.

2 Quality of Test Plan

I believe the above specification well summarises the major elements within the Multiplayer Sudoku app that require extensive testing. However, there are some limitations to the testing plan. Performance and network communications are extremely important to the functioning of the game. However, It can be challenging to set up a test environment that accurately simulates real-world network conditions like latency, bandwidth limitations, and dropped packets when testing communication between multiple clients and the server. Testing user interaction can be difficult because it requires simulating the actions of multiple players in a simulated setting and recreating user-like behaviour is difficult. Stress testing is important but may be done at a later stage. The testing approach for integration could also include integration with an actual mobile device, instead of the simulated Unity environment. This would test a more realistic scenario.

3 Appropriateness of Test Approach

I believe I have described above a valid and necessary testing approach for various concerns of the game. Naturally, with this being a real-time multiplayer app a large amount of the testing will be required to happen at runtime, though there are various levels of testing we can carry-out in order to be confident in the abilities of the software.

4 Necessary scaffolding

There is some scaffolding necessary in order to simulate a fully valid Sudoku grid. Most of the requirements involved in checking a Sudoku puzzle with empty cells is valid are simple, but checking for uniqueness is a complex task not yet surmounted. So, we may integrate the test for this and simulate a true value so the code may assimilate easily when ready, and the other aspects of validity can be adequately tested.