Software Testing

Portfolio

Name: Scott Adamson
ExamNo: B160871
Class#: ST

February 3, 2023

To demonstrate the learning outcomes of the course, I will be using a Multiplayer Sudoku project that I am building using the Unity game engine and C# scripts. This project provides an ideal platform to showcase the application of various software testing techniques and to highlight the importance of testing in software development. I will be discussing the requirements of the software, a plan for testing to ensure our app can meet these requirements and a range of potential techniques we can employ to achieve our goals. Finally, I will be evaluating the effectiveness of the testing process and highlighting the lessons learned during the development of this project. Through this case study, we will see how software testing can help to ensure the quality and reliability of software, and how effective testing can lead to a better end product.

Note that the main line of functionality is derived from the SudokuStorage.cs script, and unit tests are contained within the tests folder. It may not be possible to accurately build the entire project as it will require a version of Unity and the multiplayer aspect will require a Photon server, The SudokuStorage class should operate fine without this. I have only included the scripts and tests in the Github. These scripts are intended to be attached to game objects. I have attached screenshots in the github of how the project looks within Unity.

The github can be seen here: `https://github.com/ScottAdamson26/Sudoku-App`

1 LO1

The creation of our Multiplayer Sudoku app arouses many requirements in order to successfully implement a well rounded, functional Sudoku app. To achieve the learning outcomes of LO1, we first begin by exhaustively detailing the requirements of Sudoku Slam. This document of requirements can be found within the GitLab project. As we build our requirements documentation, its clear that the most important of requirements of our app comes with the solving and generation of a valid 9x9 Sudoku grid, which can be quite challenging, particularly doing this in realtime. The test for this section was passed.

1.1 Range of requirements...

The Sudoku app revolves around two key pillars of features: the multiplayer aspect and the logic behind implementing a Sudoku puzzle, both of which provide opportunity to implement various types of testing approaches. A key functional requirement of the system is generating a valid Sudoku. The definition of a valid Sudoku incorporates multiple characteristics, not all of which can be tested at once. Users must be able to interact with one another in real time, which requires various levels of testing in relation to user interactions. Users scenes must be in sync, users may only play a game once two players have joined etc. A measurable attribute the game must possess is a fast Sudoku generation and solving algorithm. The app is intended to provide a seamless multiplayer experience and Sudoku solving algorithms can be time consuming.

1.2 Level of Requirements...

In designing this game, there are various sub components which are integral to proper functioning of the app. This includes, the game scenes, the game objects, the scripts used to control the behaviour of the objects, the server. In order to provide a safe platform to build the game it is important to establish a working communication of all elements in the game via integration testing. This highlighted various errors within the project and allowed us to correct them. The details of this can found in the testing results document, as well as later on in this document. Unit testing may be used when evaluating the effectiveness of our Sudoku validation methods. This can be seen in the test scripts document. Tests should be carried out at run-time. Integration tests may be used to ensure proper setup of the project and allow all components to communicate with each other. This allows components to then be developed separately, reducing the need for scaffolding.

1.3 Test Approach

In order to verify that we are generating valid Sudokus we must use a combination of unit tests as well as measurable quality attributes which we can test at runtime. We evaluate the methods employed in the runtime logic on edge cases so as to ensure expected output before deploying the project. We must so operate systematic testing approach which exacts the most important of cases which represent the most challening of method input, so we can develop a confidence in the program, As there are a sextillion+ valid Sudoku grids its impossible to know whether any one is a valid Sudoku without checking, so it is necessary to validate at runtime. We should perform integration tests upon the app via a mock run through of the app to ensure all components are performing their job with respect to all other components. User input must also be validated for both a valid digit 1-9, as well as being the correct solution. To do this, we must know the correct value and compare the solution grid to the users', ignoring the empty cells. It is not a high requirement of the game to perform security testing, as we are not sharing sensitive information on the server. As the game is for android only, interperability testing is not high on the list of priortities as the devices running the game are all extremely similar and there is no cross platform requirement.

1.4 Appropriateness

As the app is centred around Sudoku puzzle solving it is natural that a large amount of the requirements revolve around the Sudoku related algorithms. Sudoku is a complex combinatorial puzzle and so we should be prepared to handle a large variety of cases, and the testing methods proposed and employed in this document, others and within the tests help us achieve this goal. The multiplayer networking aspect of the game provides us with another unique set of requirements and challenges. One such challenge which is difficult to address within a testing plan is simulating the network conditions of the server once populated with many users in many locations in order to perform appropriate performance testing. This is something which is important to the game but is difficult to be looked at at this stage of development.

2 LO2

The specification for the testing plan may be found in the LO2 test plan document.

2.1 Construction of Test Plan

I describe an appropriate testing strategy for our Multiplayer Sudoku app and where our requirements fit into the lifecycle in the test planning document.

2.2 Evaluation

I provide a critical evaluation of the test plan and any vulnerabilities it has within the test planning document. I believe I have provided a well rounded assessment of the problem and its resulting testing requirements. A major gap in my test plan, which I discuss in the document, is checking if a Sudoku has a unique solution, which is an extremely important to the functioning of a puzzle solving game. However, this is a complex problem, which cannot be tested in the absence of some yet-to-be developed code.

2.3 Instrumentation of Code

In proper evaluation of the project we may need to build some scaffolding in order to test at this stage of development. When carrying out the integration testing, the Sudoku generator was not completed, so in order to test the output of a Sudoku puzzle we must build some code around this which assumes a valid Sudoku grid has been passed. This was an adequate solution as we were now allowed to see the numbers within the game. Users also cannot currently make actual input, so in order to test for valid input we both must test the potential user inputs within our unit tests, passing various arguments to the method as a user would. If we were to analyse more performance metrics, we would perhaps require a larger amount of instrumentation.

2.4 Eval of Instrumentation - Mark: 1

The project makes very little use of instrumentation and so it is difficult to provide a complete evaluation. The test planning document discusses the necessary scaffolding to achieve a certain level of performance from the program for testing purposes.

3 LO3

The testing plan described in the previous object is effective in testing the project and identifying issues within the project.

3.1 Range of techniques

The integration testing of the project, establishing working communication between all major elements, is a valuable step in testing the project. This is important as the communication of all objects is essential to many future elements

of testing and provide the platform for effective future development. Integration testing also allows us to test a massive amount of the multiplayer functionality requirements, without the need for the Sudoku aspect of the game. We make use of unit testing in order to adequately test the most interesting of cases for a valid Sudoku checker. There are a large number of possible Sudoku grids and so in order to test the special cases in a wide variety we can use unit testing which allows us to see how the methods perform with such cases. This is known as systematic testing. This testing is intended to be done as well as runtime validity testing on a generated Sudoku. The unit tests give us confidence we can acccount for every case before deployment but the runtime testing ensures that no puzzle may go out without being tested for validity

3.2 Evaluation

It is helpful that the definition of a valid Sudoku is a rigid measurable attribute which means we can effectively use this to define our evaluation criteria. A suduoku must meet all requirements of a valid sudoku as outlined in the requirements document. This allows us to have a large amount of confidence in the ability of the software to work as intended. Integrity of the puzzle is a high importance and so performing these tests as described are a necessary and valuable step in achieving success for the project. The testing of the program under a stressful load, or even just simultaneous games, is an issue which is difficult to recreate and is under addressed at this stage in the project, We cannot confidently say our program will work as intended, based off of the scope of this testing outline. An integration test can be considered successful when all sub-components of the project are effectively communicating and displaying correct behaviour.

3.3 Results of testing

The testing approaches applied exposed some errors within the project. Integration testing established a connection between all objects, scenes and scripts, and revealed a syntax error in the code which made the code not perform as expected and not wait for two players before loading the game scene. The validity testing helped identify the need to better exact what it means to be symetrical for a Sudoku. This is an issue it is not clear the program does not adequately address and a revision of the testing strategy for this component is necessary. We also can say with confidence that the program can adequately test for valid user input.

3.4 Evaluation of results

The results of the tests lies in the results document. The results yield positive for many of the key areas of a valid Sudoku. It did reveal that we are not meeting two key aspects of Sudoku generation and this is something which we must work on going forward. In integration testing we discovered a key bug preventing proper function of the game lobby.

4 LO4

4.1 Gaps & Omissions

The testing approach as described in this course does not consider the difficulty in testing multiple threads executing simultaneously as the output of certain processes may depend on order of execution, which may differ from user to user and be difficult to assess which orders of execution may cause issues. It is also very difficult to assess the performance and scalability of the project and simulate real traffic. One solution to this is to make use of load testing tools to measure response time and perform resource analysis. Accessability requirements may also be difficult to ensure via testing as this may be very depending on screen size, resolution, OS etc. There are tools which can validate this for us. There is also no mention of security threats.

4.2 Coverage and target levels

The final form of our Sudoku game will contain a full suite of tests which can ensure our algorithms are valid. At this time, the suite does not appropriately assess symmetry or uniqueness, so the tests are still limited in coverage but we do deal with some of the major requirements. The target levels of accuracy for this stage of development is to ensure a valid Sudoku grid in all numerical aspects, i.e. a grid contains one occurence of each digit in each row, column and box.

4.3 Comparison

The test carried out meets the full requirements which are possible to be met at this stage, We have fulfilled all requirements of a valid Sudoku checker, with the exclusion of symmetry and uniqueness, which are yet to be developed. All target levels are met in terms of integration, i.e. full communication is established among all components,

4.4 Improvements

In order to achieve the target levels required for success of the program we must first implement the logic required for a unique solution, as well as implementing an effective method for maintaining symmetry of the grid. This would achieve the final form of our desired target lives relating to the Sudoku grid. The use of stress testing tools would significantly improve our multiplayer functionality requirements by allowing us to determine if the program still performs as expected in a real environment.

5 LO5

5.1 Review

The code in its current state does not meet the entirety of the specification. Some changes must be made to get there. Symmetry and uniqueness have been discussed throughout this work but it cannot be understated the importance

of generating a valid Sudoku every time. We may inspect the code to manually identify defects and non-conformance. We may implement a peer review between software developers to expose the project to more eyes. The software does not currently allow user input, which is vital to the running of the game. We may also construct a checklist to improve our review systems. This involves the writing of a detailed checklist outlining the major aspects to cover in the review.

One section of the code we have reviewed is this section. Upon auditing this code there is a simple yet hard to see error which completely changes the desired function of the code.

```
public override void OnJoinedRoom()
    {

        if (PhotonNetwork.CurrentRoom.PlayerCount == 2);
        {
            PhotonNetwork.LoadLevel("GameScene");
        }
        Debug.Log("OnJoinedRoom() called by PUN. Now this client is in a room. Number in r
    }
```

The error is in the signature definition with the semi-colon.

5.2 CI pipeline

To build a CI pipeline for the project we could follow these steps. To store the code for app, a code repository (such as Git) should be set up. Developers will be able to save their modifications to the code in the repository and then have them accessed by the CI pipeline. The code is then compiled to ensure the changes integrate correctly with the unity project. We then run a set of automated tests to check that any changes made don't break any of the required functionalities. This involves all of unit, integration and system testing. If the code changes build and pass the tests, they can be deployed in a test environment simulating the real-world setting itll operate in.

5.3 Automation

The main tests which allow for automation are the ones concerning the validity of a Sudoku grid. Given the extremely large puzzle set we are working with it is very important this part of the testing is automated during each iteration of a sudoku generating lifecycle. These tests have been automated by adding in the required functionality to the SudokuStorage app, via the Validate() method.

5.4 CI Pipeline functioning

We hope to catch code quality issues, build failures, unit test failures and integration issues with our CI pipeline. Some examples of this would be unintended changes the Sudoku validation methods, this would be caught by the unit testing of our edge cases. If one of these tests were to suddenly fail, the pipeline would catch this.