Test Results

Software Testing

Name: Scott Adamson
Class#: Software Testing

February 3, 2023

1 Integration Testing

Our Unity game consists of scenes, which consist of game objects whose behaviour is defined in C# scripts. This relies a lot on the proper communication between components and so performing some integration testing is key to success for our app. Early in our development stage it is important for us to build the foundations of the project and establish communication before too much code is written. So we build our menu scene, where we establish a connection from the menu screen to the game scene. The join button connects us to the scene. We now add the required Photon networking behaviours so as to allow two players to join the game. Once 1 player joins, they should wait on a 'connecting' screen while waiting for another user. Once implemented, this failed, and a user went straight to the game scene. The scripts were not attached to the game objects. After testing again, the 'connecting' screen showed, but still loaded the user into a game without a 2nd player. Upon inspecting the relevant code (below) it can be seen that there was an additional semi-colon in the if statement, causing the contents of the if statement to execute. This was a key relation fixed due to the integration testing. We then built a script for the grid squares, which loaded, but did not display their numbers. The text script had not been given the attachment to the game object representing the grid squares.

```
// This is code containing an error in the if statement definition
public override void OnJoinedRoom()
    {
        // if two players are in the lobby, load the game scene
        if (PhotonNetwork.CurrentRoom.PlayerCount == 2);
        {
            PhotonNetwork.LoadLevel("GameScene");
        }
        Debug.Log("OnJoinedRoom() called by PUN. Now this client is
        in a room. Number in room: " + PhotonNetwork.CurrentRoom.PlayerCount);
    }
```

2 Grid Validity

Grid validity is a runtime test to be carried out, and so this has been implemented via the Validate() method in SudokuStorage. In order to assess the code for this I have also included it within a unit test within the testing script. The test is named isValidGrid(). I have outlined some edge case Sudoku grids to ensure the algorithm is capable of properly assessing the validity of the grid. It is important to select a range of grids representing the extremes which the method may be faced with. The algorithm was a success on all provided grids.

3 User Input

We must implement a method to check the validity of user input. I have encoded this logic within the unit testing script in order to test its effectiveness outside of the app environment. The method is called testCorrectInput(). Given two 2d arrays, where on represents the solution grid and one represents a grid where

1 of the cells contains an unvalidated user input value. Upon testing there was an obvious logic error within the program and we could easily fix this. We may now confidently say that this algorithm for testing valid user input is effective.

4 Symmetry

We implemented a testing method for testing any given grid is symmetric. This is a requirement of a valid Sudoku. Upon testing, it became obvious the code is not performing as expected and is not capable of properly testing if a Sudoku is symmetric. This is now a known issue and is to be fixed.

5 Uniqueness

As discussed in various other sections, I have not yet developed the logic necessary in order to determine if a Sudoku is unique. I have created the necessary skeleton which returns true by default so this can be swapped with the uniqueness validation when it is ready.