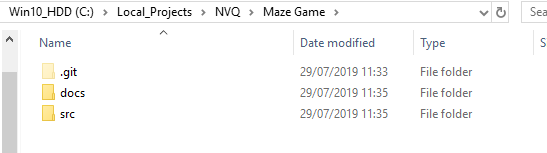
**Final Project – Olde Worlde Phunne Maze Game. Report on Project Progress, Limitations and Future Improvements**

1. **Setting up the project and writing requirements:**

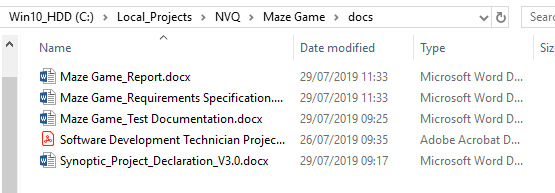
At the outset of the project, the first task I undertook was to read the project outline documentation and gain an insight into the specific requirements of the project. This gave me an understanding of the documentation that the project required me to produce and the solution it required me to write.

With this understanding, I made the decision to write the game using Microsoft’s C# programming language and .Net Core framework, and also to use Visual Studio for the IDE (Integrated Development Environment). I made these decisions as I am familiar with this language and framework combination and these technologies are used within my team. I also made the decision that the game would take the form of a command-line based text-adventure, because a week long development timeframe does not leave much space for creating visuals and this would leave me more time for focussing on writing code to meet the requirements.

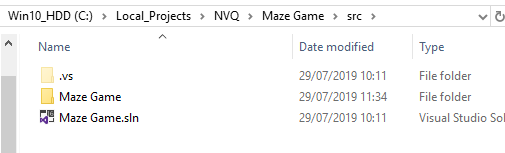
The next step was to put together the word documents needed for the project, including the requirements specification, the test plan and this final summary report. I then created a git repository in my company’s source control server for the project. Git is the source control tool used in my development team for all of our projects. I cloned this repository onto my local machine and moved my documentation files into the project directory under a “docs” folder before committing these files in an initial commit and pushing them to the remote repository. The file structure for this project can be seen in Figures 1-3 below.



**Figure 1: Project Directory**

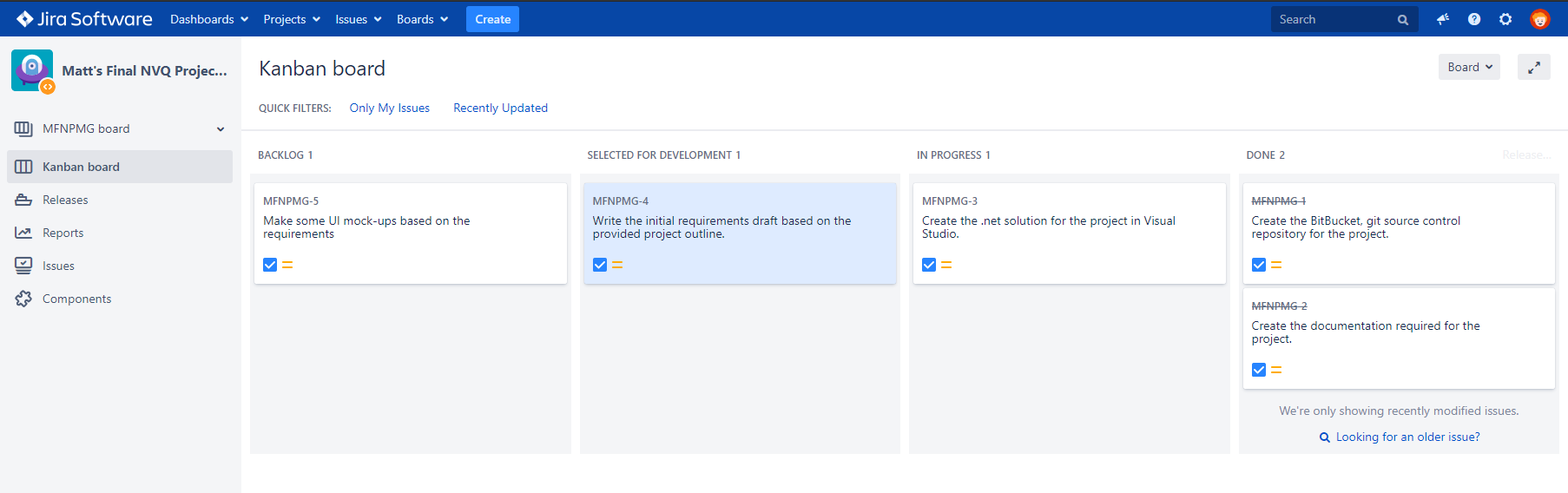


**Figure 2: Project “docs” Directory**



**Figure 3: Project “src” Directory**

Following creation of the project files, I made a Kanban board on my Company’s JIRA server instance in which to register tasks and track their progress. This can be seen in Figure 4 below. This Kanban board would be used throughout the project for me to register the tasks I was undertaking in an effort to effectively complete the project.



**Figure 4: Project JIRA Kanban board**

These initial steps are the same I take with all work-based developments to ensure consistency between projects. The use of a Kanban board to track tasks and issues during development is something I find useful during the course of any development and helped me to organise and manage the tasks required to undertake this development.

With the project file structure and JIRA task board set up, the next task to undertake was to put together a requirements specification for the system, based on the project outline document. The requirements specification structure I put together was based on the requirements specification form I helped develop for my team and is what we currently use for producing specifications for new systems.

1. **Writing the application:**

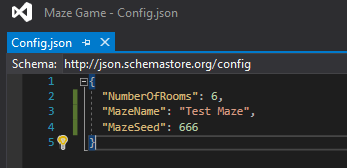
With the project structure and documentation in place, the next step was to write the application. When approaching any new development, I always try to model the project entities first as per the requirements specification, which is what I did next. Following this I wrote the initial program flow structure, including the different stages of the overall program, from:

* Maze initialisation
* Entering and progressing the in-game loop
* Resetting the in-game loop
* Presenting the results.

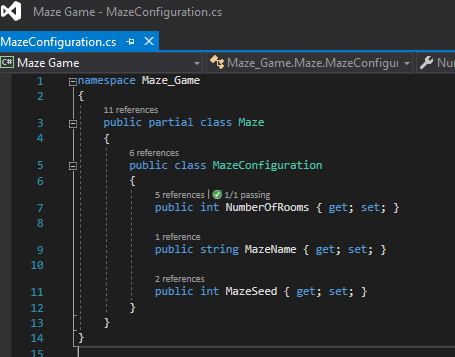
I also later added a user name collection to the start of this sequence, as this is an additional requirement I added to my requirements specification to make the game more interesting.

Following this I abstracted the common commands I would use for the game, including writing the text as a crawl to the command line, instead of all text being written to the screen at once as it commonly is in console applications. As this is not a requirement for the project, it is something I should have perhaps added to the requirements and returned to at a later time. However, I believed it to be important to the feel of the game and user experience. I also believed I could complete the task quickly and it was related to the static “ConsoleHelper” methods I was putting together, so I proceeded to add this functionality. I also added the screen shake effect and static, common methods for looping until the user entered a specific confirmation or command.

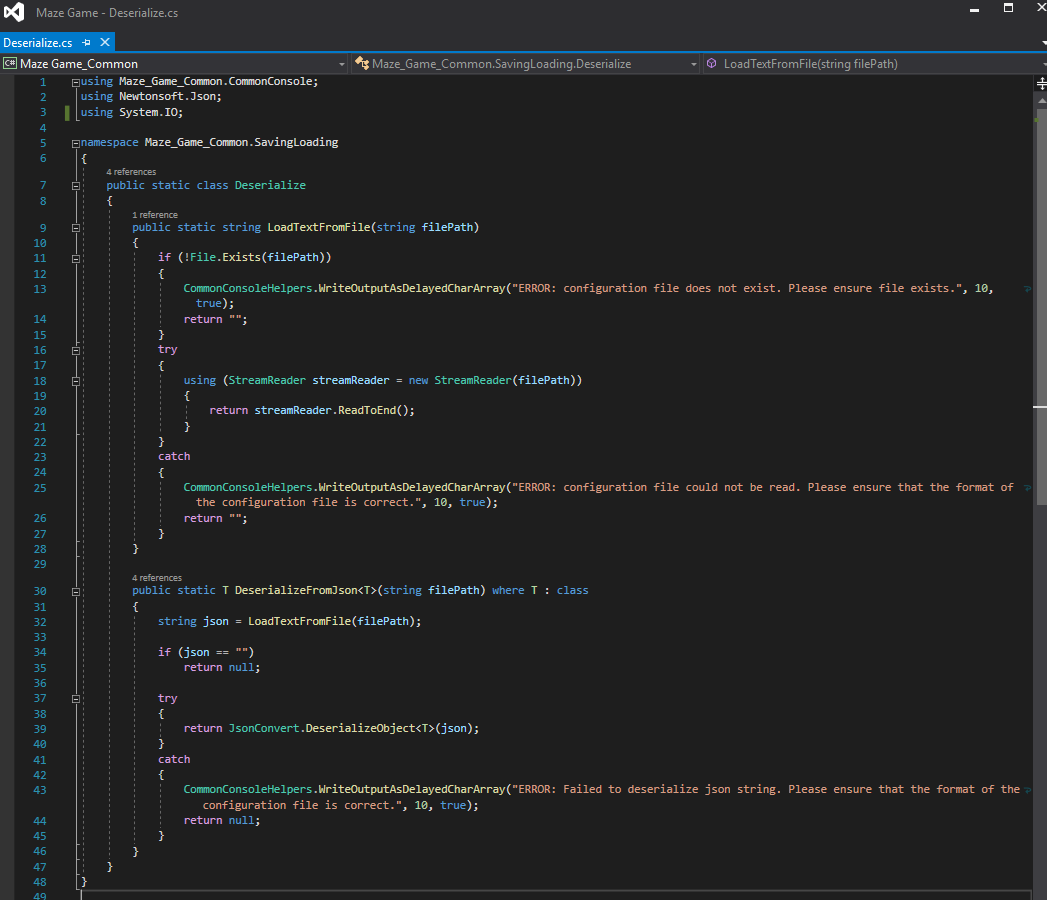
Following some testing and refining of these common methods, I added a common .Net project for abstracted, shared code and moved these helper methods into this project. This is, again, another common practice in my workplace developments that was suitable to use in this development. To this common project I also added a deserialization class, and wrote the code for reading text from a file and parsing json-formatted text into a C# entity. I tested this deserialization code with a json config file, ironed out issues and added error handling that would print error details to the console, as per my requirements specification. The configuration file, C# object and deserialization code can be seen in Figures 5 – 7 below.



**Figure 5: config.json file**

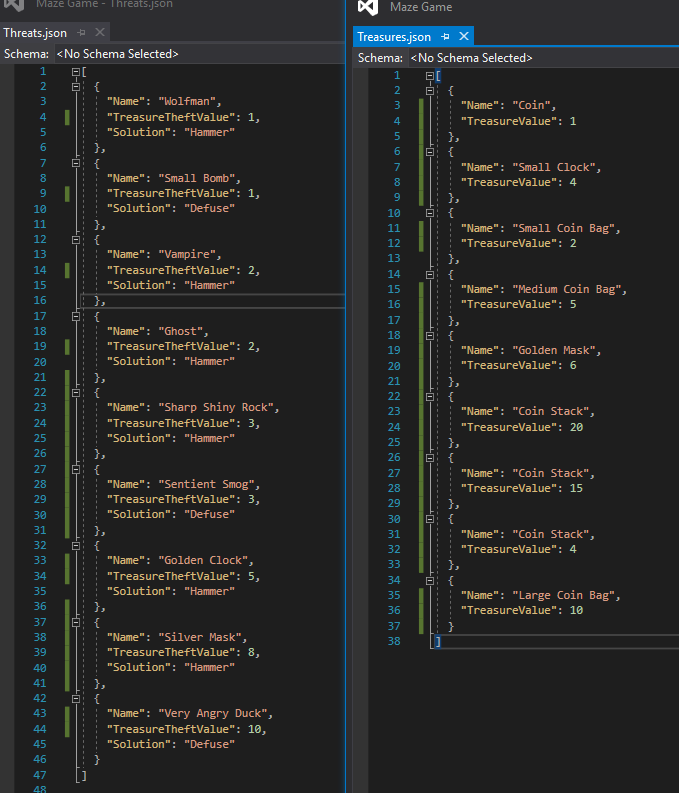


**Figure 6: C# code that relates to the config file**



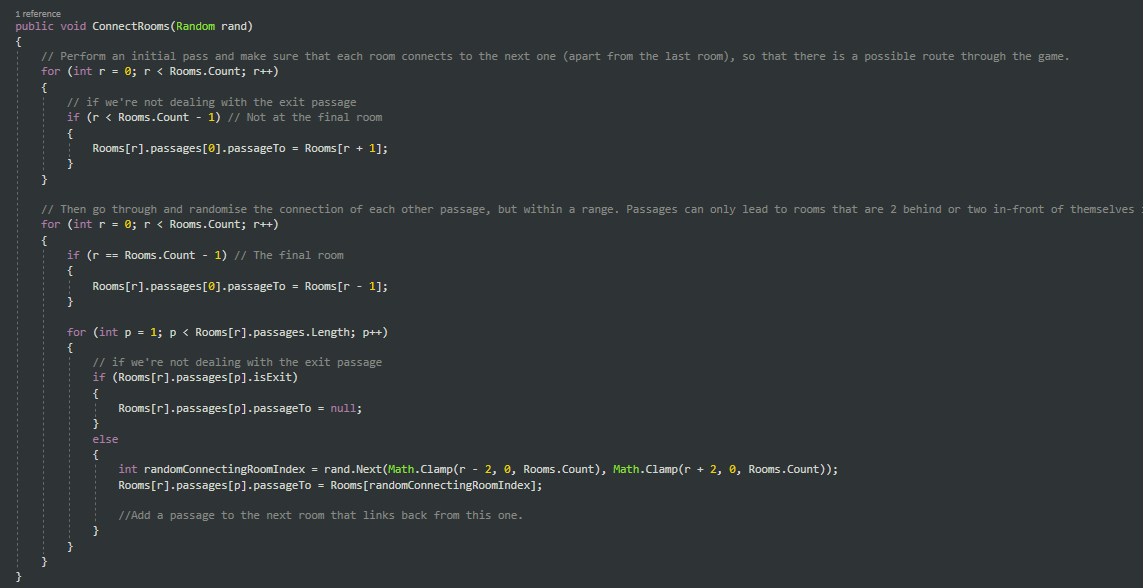
**Figure 7: Deserialization code, with “TryCatch” blocks for error handling and console logging**

The period of developing the core code, testing the application as I wrote new code and then refactoring and refining that code took up most of the week long project time. During this time I came across a few points at which I realised I could quickly improve or add to the design outlined in the Requirements Specification. One example of this is the point at which I decided that, instead of hard-coding a set of Treasures and Threats, I would specify a collection of these items in json files to make it easier to add to and balance the types of these items present in the game. This was straightforward to implement, due to the fact that I had already written code for deserializing json files into C# objects, as discussed above. Figure 8 below shows the Treasures and Threats configuration files.



**Figure 8: Treasures and Threats config files**

One example of an issue I came across during development that halted my progress was trying to implement the algorithm for passage generation. Passages are generated in rooms on room creation, however, passages are connected to other rooms by a separate method, as all rooms need to be created before the passage can be generated. I wrote several implementations of this method that resulted in various errors, from connecting to “null” rooms to all passages connecting back to their originating room (and therefore preventing the player from being able to move between rooms). The final implementation of this method that proved successful can be seen in Figure 9 below. It can be seen that I settled on an algorithm that allows passages to be connected at random to any room that is within 2 indexes behind or in front of the current room in the rooms collection (taking into account the length of the room and the zero index of the collection).



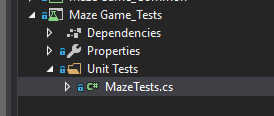
**Figure 9**: **Passage to room connection method**

Whilst writing the code for the application, I often referred back to my organisation’s coding standards document. This document is a work in progress at the moment, but still outlines a number of coding conventions that myself and other team members follow when writing code. These include standards such as capitalising class names and refactoring long methods and abstracting common code. I have included the Coding Standards document alongside this report.

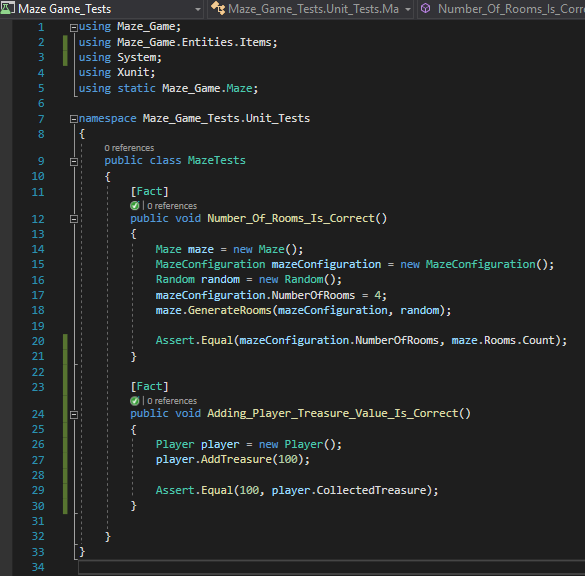
1. **Testing the Application:**

During the course of writing the application, I decided that I had time to add a Unit Testing framework and implement some unit tests. The Unit Testing framework that I added to the project is called XUnit (<https://xunit.net/>) and is the testing framework used in my development team throughout our .Net based applications. Initially, implementing this framework caused some runtime errors, but I managed to fix these and get it working correctly.

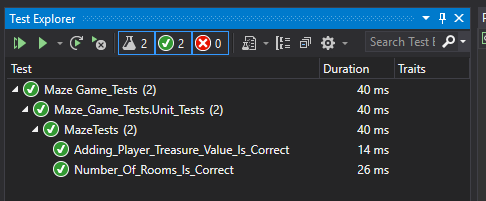
Unfortunately, I didn’t have enough time to add many unit tests to the application Test Project. The testing project along with the unit tests I wrote can be seen in Figures 10 and 11, and the results of the unit tests being run can be seen in Figure 12.



**Figure 10: Unit testing project in Visual Studio**



**Figure 11: Unit Tests written using XUnit**

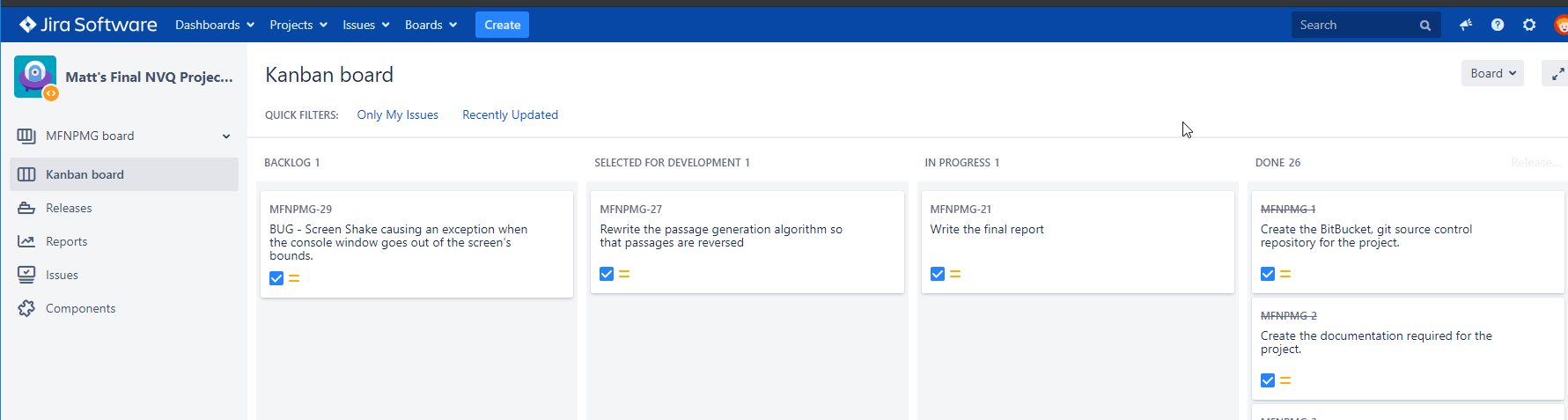


**Figure 12: Unit test results**

As well as implementing unit tests, I was also able to write and carry out a test plan against my application. For test plans, it is common practice in my organisation to write individual tests against the requirements from the requirements specification for a system. The details and results of the tests I carried out against my requirements can be seen in the Test Plan document, provided alongside this report. If given more time, I would have developed a more comprehensive set of tests against the requirements.

1. **Limitations and Future Improvements:**

If I was to continue development on this project, the first step I would take would be to fix the few issues raised by the failed Test Plan tests as well as any remaining bugs. I am currently aware of an application bug not covered in the Test Plan or unit tests that is causing the application to throw an exception when the “ShakeConsole()” code is executed and the console window extends beyond the width or height of the screen view. As can be seen in Figure 13, I have logged this bug, along with a Failed Test plan entry into my project’s JIRA Kanban board for future work.



**Figure 13: Final project JIRA Kanban board**

Given more time I would also have wished to make some elements of the UI more intuitive. This would have involved making some text clearer and the layout of text more consistent throughout the project. I would also have wished to investigate the possibility of adding to the game’s design with more enemy types and a more complex action system in which the player can perform more than just the two “hit” and “defuse” actions on threat objects.

The key limitation of the Maze Game design, currently, is that it is not a visual game with graphics. This means that it is harder to convey some types of information to the player, such as where they are or what is in the room with them. This limitation lead me to add the feature that provides the player with a hint as to whether or not they are getting closer to the end of the maze, or further away from it. This is a fairly simple hint at the moment, however, and could be developed into a more interesting mechanic in time.

Issues like this would have been raised had more playtesting been carried out, as well as more testing in general. A more comprehensive series of automated Unit tests would have ensured that changes to the project would have been less likely to cause a breaking issue and would have simplified project testing whilst adding new features.

There are also many areas of code that I would have been refactored and improved – specifically the code in the “Program.cs” file that controls the flow of the program, which could have been separated out into separate more discrete files to make reading and understanding the code easier.

As I didn’t get around to producing the User Guide until the final day of the project, and I had some issues installing OBS studio to capture video footage of the Maze Game program in-action, the user guide was a rushed production. I would rework the user guide video, including more of an introduction and a better description of the actions the user can perform in the game.