Software Development Technician

Project B - Maze Game

Andrew Hartnell

Design

- **Technology**

The application will be constructed using C# and will be a web application using .NET Framework. This will provide an easy to construct User Interface with basic HTML, CSS, and JavaScript which can be connected to the C# logic. This also relates the original requirements requesting a system to help promote a website, as the client can easily link to another standalone web-based solution.

Using C# .Net as the basis of the application means that future developments and enhancements can be easily made and deployed. Users don't have to manually update the system themselves, as this can be controlled remotely. Furthermore, the requirements state that although it's not essential, eventually user accounts and a login feature would be desired, which can be easily implemented into the application. Furthermore, being web based will allow an easy connection to external APIs, such as logging in with Facebook.

- **Code Structure**

The code will be structured logically, keeping sections separated. This will be achieved by have sections that will deal with the maze (it's generation, keeping track of direction of travel etc.), with the entities (threats, treasure and the player) and the logic that combine the two.

The two sections concerning the maze and the entities will likely be separate class libraries, which will mean the code can be reused in the future and allow for ease of maintenance. The main project will then have logic class which will tie the two together and handle all requests passed on from the controller class. This will allow the code to stay simple and logical, and not spread business logic out too much.

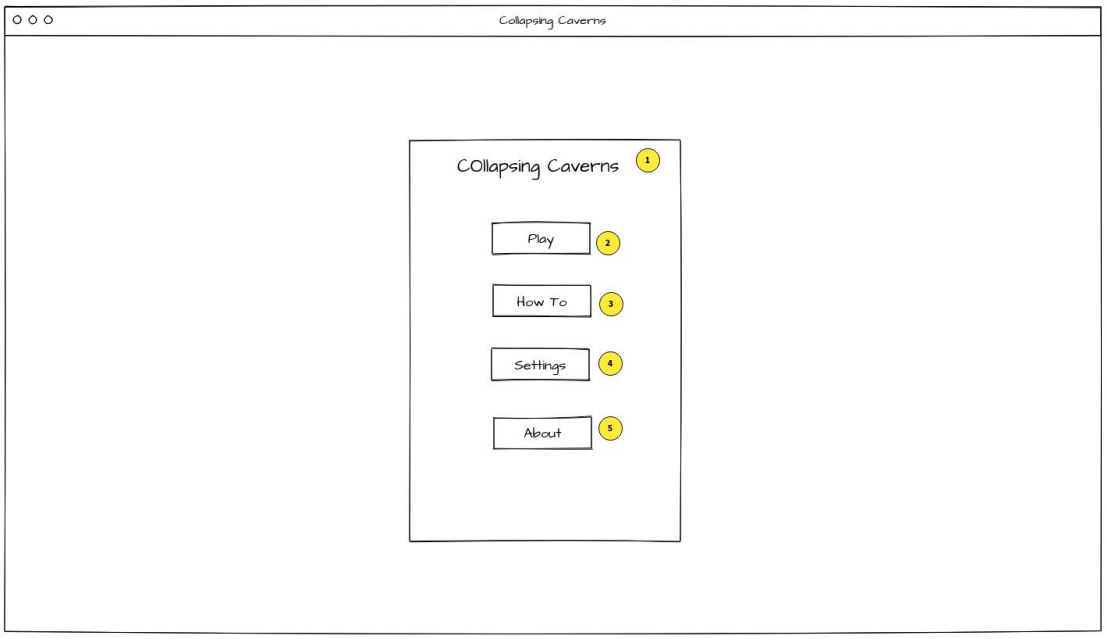
- **Data Format**

The user will be able to save their preferences for the maze, as per the requirements. The simplest way to do this is to create an object that holds the user settings, then convert this into a JSON object that can be save to a text file. When the maze loads, it can then read this file, and convert the JSON back into the C# object, allowing the program to read the properties set by the user.

- **Wireframes**

Below are the wireframes used to design the user interface for the application. It was creating these that the user experience and structure of gameplay was developed, which helped design and structure the code in the back end.

Home Screen



This the first screen the user will see and has the main menu options. The background will be an image or animation that will evoke the feeling of mystery, to echo the name of the application, *Collapsing Caverns*.

1 The name of the applications

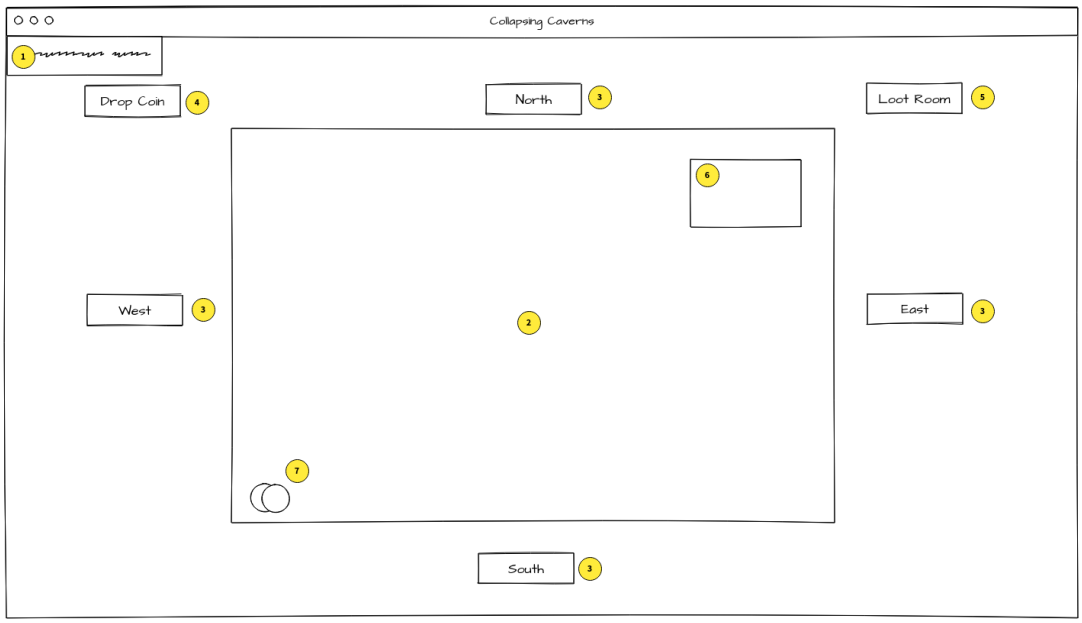
2 The *Play* button. The user clicks this to play the game. It will first display some text to give context to what the player is trying to do, and then will take them to the maze itself.

3 The *How To* button. When the user selects this option, they will be presented with a simple user guide to the game. This will outline the objectives of the game, and how to interact with it.

4 The *Settings* button. This button will give the user some inputs where they will be able to set configurable settings, including a player name.

5 The *About* button. This will show the user some text about the application.

Maze Screen



This screen is the primary screen of the game, as the user will be using it to interact and navigate through the maze.

1 This will display the user's stats, including how much treasure they have.

2 The current room. This will be an image generated dynamically to indicate the doorways of the current room.

3 The directional buttons. These buttons are how the user choses which direction to leave the room and travel to the next room. The buttons will only display for the directions the user can travel for each room.

4 The *Drop Coin* button. This allows the user to drop a coin in a room, as per the requirements.

5 The *Loot Room* button. If there is treasure in the room, then the user may collect it using this button.

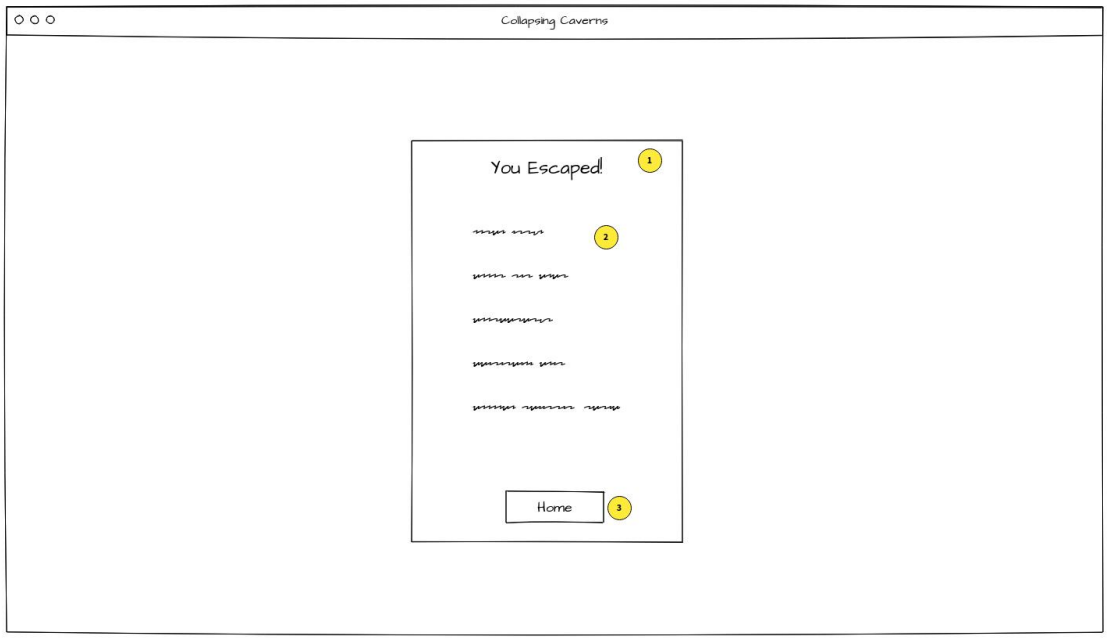
6 Treasure chest image. As the room loads, if there is treasure in the room, this chest will appear to indicate as such to the user.

7 Coin image. If the room has any user dropped coins, then they will be displayed here. Each time a coin is dropped, another coin will appear in real time.

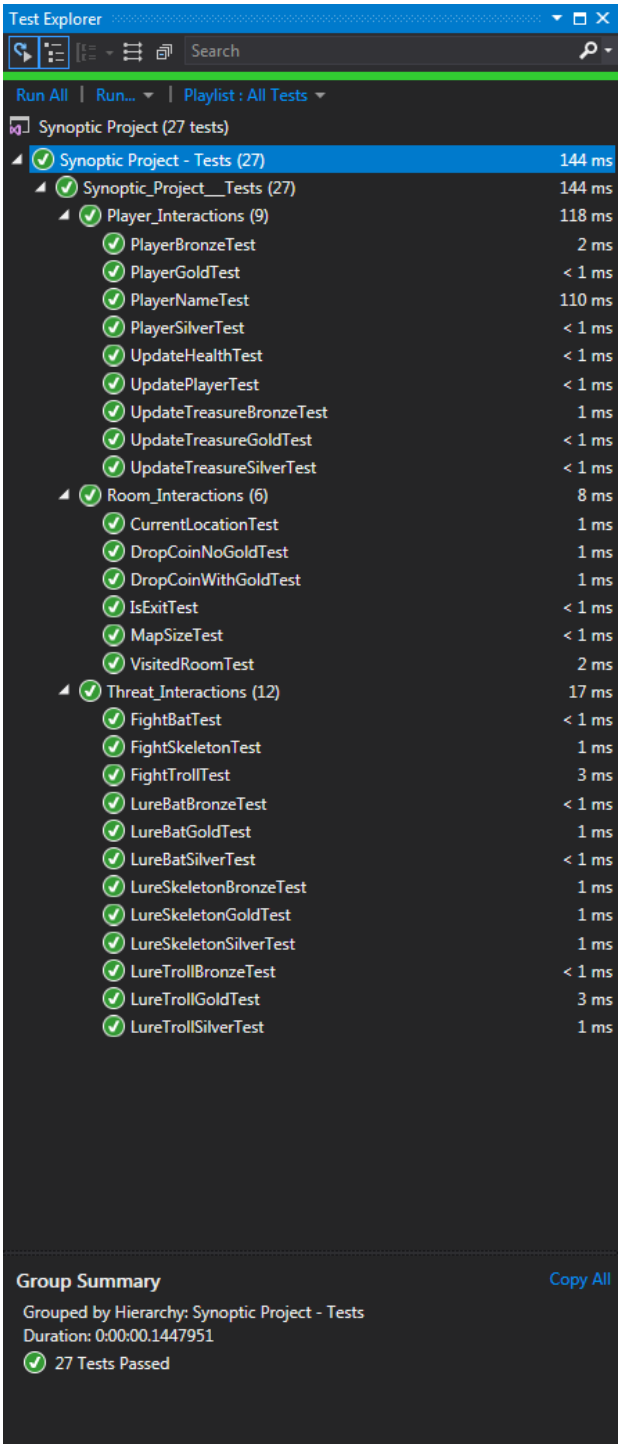
Completion Screen

Once the player has successfully completed the maze, then they are presented with this completion screen.

1 Heading. Informs the user they escaped, could include their player name.
2 Summary. This will display the players finishing stats, such as health or wealth.
3 The *Home* button. Will return the player back to the home screen, where they can choose to play again.



Testing



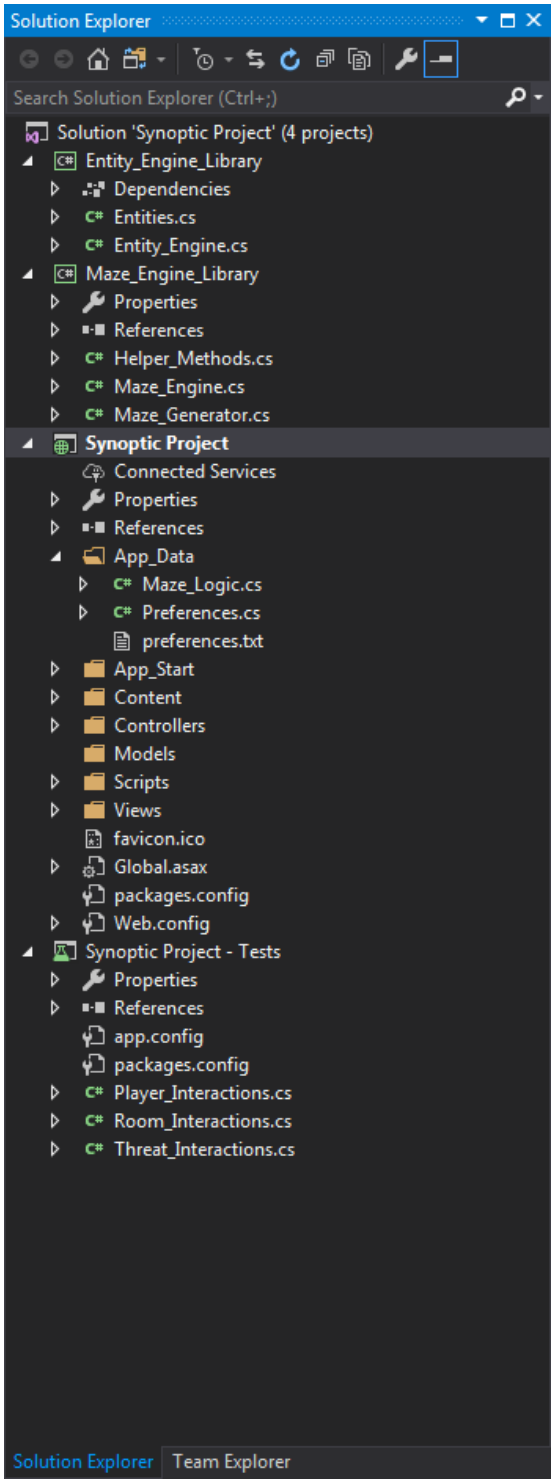
The application has been fully tested with a comprehensive set of automated unit tests. These cover methods across the whole system, including methods in the two class libraries. The diagram to the side shows the tests all passing.

The tests are broken down into logical interactions with the different sections of the application. These are the interactions with a room, the interactions with a threat, and any other interactions the player has with the game.

Each test is structured with the *'Assign, Act, Assert'* format. Simply put, each test method begins by assigning the relevant objects and variables. Then the interaction being tested is completed, and the result captured in a variable. Finally, the result and the expected outcome are compared, with a match producing a successful result.

More can be interpreted from the test by looking at the test project code in the solution.

Structure



As planned in the design section, the program was made up of two class libraries controlling the maze and entity logic, with a logic class in the project combing them together.

The diagram to the side shows the project solution, with the project, class libraries and test project.

The code in each project is structured, named logically, and have comments to aid understanding. All attempts have been made to make the code as understandable as possible.

Dependencies

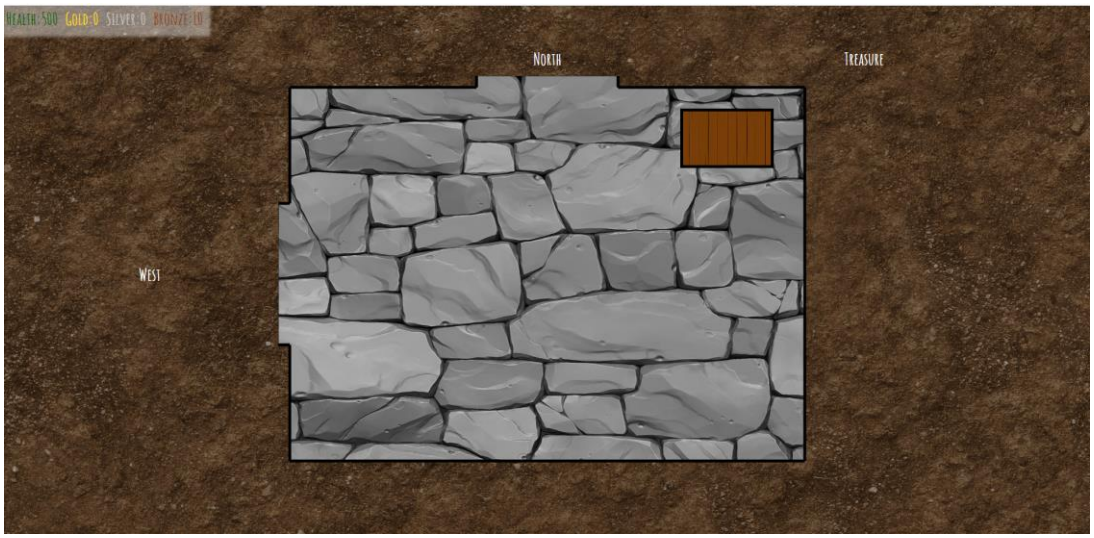
As the application is web based, there are dependencies used in the front end. Listed below are the external libraries used:

- Vanta.js – for the fog background
- Bootstrap – for structuring HTML
- jQuery – for structured JavaScript and use with bootstrap for animations.
- Csshake.css – for animated text
- Google Fonts – for the Amatic SC font used in the application

In the C# code, there are default NuGet packages used, though there is one extra added for this application:

- Newtonsoft.json – for converting C# objects to JSON and back

Completed Screens



User Guide

A user guide for the user is provided in the game from the main menu. This outlines the objectives and how to play. It also gives some indication of tactics to think about when completing the maze.

Evaluation

The program was created quite closely to the specification and didn't deviate too much from the initial designs.

One positive is the success of the maze generating algorithm. Having done research on different maze algorithms, the implementation of the Recursive Backtracking algorithm gives a truly random maze for the user to complete. And it can incorporate a user's configuration into the generation of the maze as well. Something for the future would be to expand on saving a configuration to include saving a particular maze and load it again in the future. This logically wouldn't be too hard to accomplish, similar to the current configuration save, it could implement a JSON structure, which could be read back in.

There are further aspects of the code base and design that could be improved upon in a future iteration of the application.

The first being a full implementation of the difficulty setting. The user can set the game difficulty, and the game loads with a default setting of medium. However, this property is never used as it wasn't a priority to work on. In future, this setting would be read, and can change the weighting on how likely different threats and treasures appear in the maze, as well as their strength and value. This would be an improvement on the game and make it more enjoyable.

Next would be improved error handling. There are several methods that were created to return a boolean. This was with the intention that should a method or task fail, then it would return false and trigger an error message to the user. However, this was unable to be fully implemented and several methods return a static true value regardless of the method's success. This should be worked upon, as well as catching any errors that do occur to make a more robust application and provide a better user experience.

Third would be input validation. The mainly applies to the settings screen, where although there are restrictions on the inputs themselves, there's no validation of the data when it is submitted. This could mean that the user could provide data that could potential affect the performance of the application. Web applications are based on technologies that can easily be manipulated on the front end, and so server-side checks should definitely be implement in the future. This also applies to the inputs on the main maze screen.