This document comprises of the ‘design’ phase for the first prototype of my project. This will consist of outlining the structure of the code and physical designs of elements such as the user interface of the prototype. This prototype is being built as a testing build for the procedural generation; the prototype will only contain the code for it and objects that will provide debug information on the procedural generation. Having this prototype will allow me to outline and repair any issues with the procedural generation – this will allow me to easily build the other prototypes on top of it.

**User Interface Design – Visual Elements**  
The user interface is the other main component of this prototype. Its purpose for being included is to be used to present debug information as well as providing an environment to access and initiate the map generation from and stop it if need be. This is possible via Unity’s editor however this won’t be available to the focus group that I am working with – they will only have access to a build of it. For them to test multiple seeds and essentially ‘break-test’ my prototype, I will need to include these features in the UI.

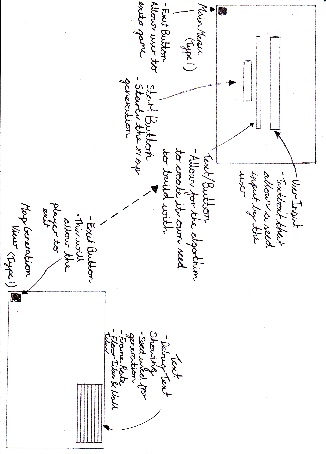


Figure 1 UI Initial Sketch

There are two main parts to the UI, these are the main menu and debug heads-up-display. The main menu will allow for the user to insert a word to use as a seed (or select the computer to generate a random seed) and start the map generation, while the debug HUD will provide system information while the map is being generated.

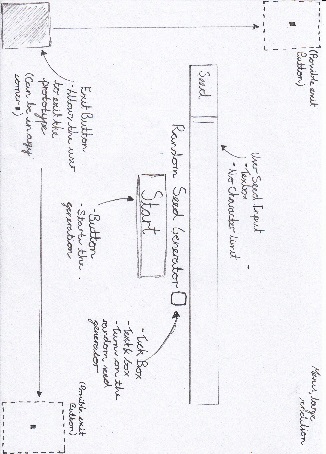


Figure 2 UI Main Menu

I started off with rough sketches of what I wanted the UI to look like as well as what the UI to contain. The first sketch which I decided to develop on further contained small drawings of my desired menu and HUD (figure 1). While not exactly detailed it does outline what objects I want to have as well as their functions.

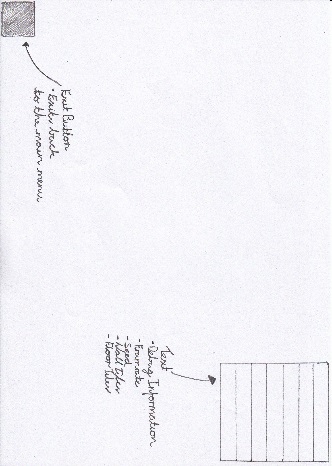


Figure 3 UI Debug Information

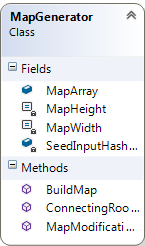
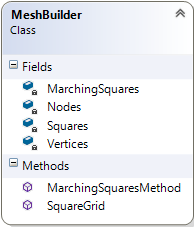
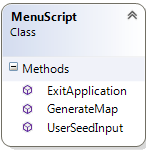
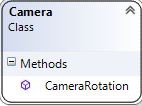
Afterwards I created full A4 sketches of the two parts to the UI. The main menu (figure 2) is not like a common main menu (with a title and some sort of aesthetic), it is rather barebones – only consisting of the seed input box, a button to enable/disable the random seed generator and a start button. I also included a button to exit the build, however this is not exactly important as the user can use a command to exit the build (e.g. Alt+F4 or Esc).  
The HUD put on top the map generation is even less compared to the main menu, only containing an exit button and debug information (figure 3). This debug component will consist of information such as the framerate of the build currently, the seed used and information about the map generation such as the amount of floor space compared to wall space. This information will give me an indication of how well the system used is coping with the map generation and if any optimisation or scale-back is required for the prototype.

**Identifying Classes – How will it be built?**When designing the prototype, I will need to consider what classes are going to be used for the algorithm and eventual code. These classes will hold chunks of code that will perform certain actions for the entire prototype. It is a lot better, for the sake of troubleshooting and debugging, to arrange the code into their specific tasks.

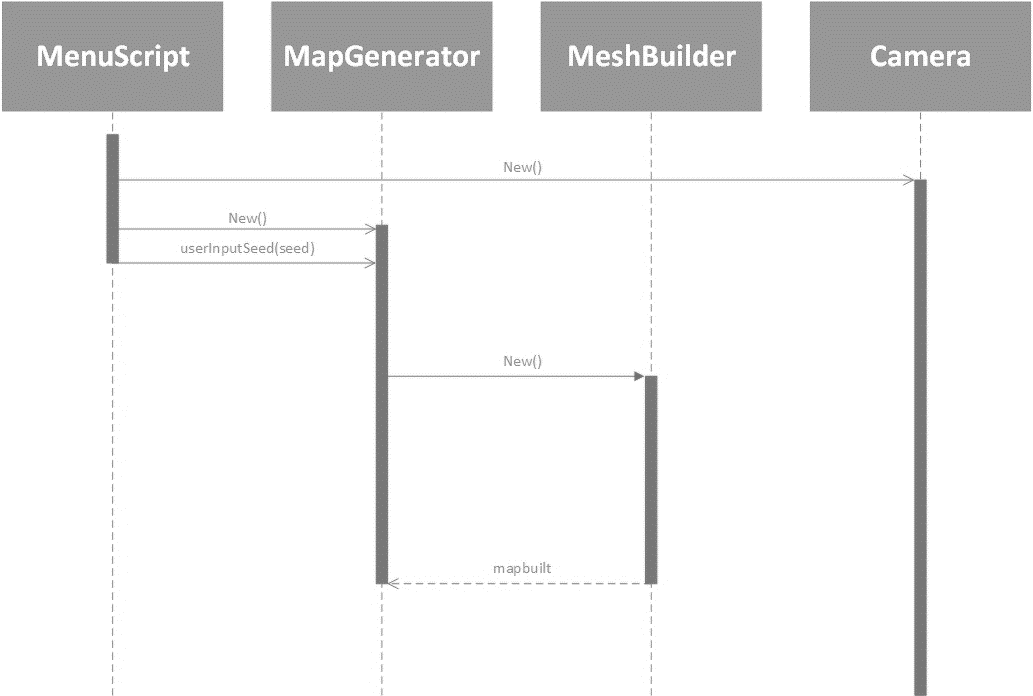
Naming Conventions  
It is important to work with naming conventions to allow ease in troubleshooting errors and organising variables, classes and methods. The naming convention that I will be working with is ***CamelCase***; this is where the first letters of each word are capitalised – resembling the shape of a common camel.

Class/Script Diagrams  
This section of the design document lays out the structure of the classes in the prototype and what’s within them. The section will be discussing the purpose of each class, the purpose and usage of the methods within the classes and their accessibility levels. Concerning accessibility levels of methods and fields, I plan for most to have private access modifiers – the reason for it to prevent any methods that shouldn’t be communicating with each other doing so; if for a special reason that the method or field will have to be public, it shall be stated with reasoning below.

* **Map Generator:** This is the class manages the main part of the procedurally generated map. The class creates the array that is holding the data for the procedural map as well as modify the array with the cellular automata algorithm and other modification methods.
  + Field – Map Array: The data for the procedurally generated map will be held in a 2D array, with the dimensions of the array being that of the size of the map. After the modifications, this data can be passed over to the Mesh Builder class to be made into a physical map in Unity. There are multiple fields that need access to the field in order to modify it and so I will keep the field public.
  + Field – Seed Input Hash Table: The seed that is inputted by the user will be stored in a hash table. This hash table will be accessed only by the build map method to build the initial map generation.
  + Method – Build Map: This method will be the first to modify the map array, using the data from the hash table made from the seed to fill the array. This method will be built up upon the cellular automata algorithm.
  + Method – Map Modifications: This method performs the first stage of modifications onto the map generation array. This will contain conditionals in relation to the rules set for the map generation.
  + Method – Connecting Rooms: The second stage of modifications to the map generation array is connecting rooms that were identified within the map. This method performs both the identification of rooms and the connection of rooms.
* **Mesh Builder:** This class contains the methods to build the physical rendition of the map made by the Map Generator class within the Unity environment. This class contains a lot of unique fields and methods that are only used by it but internally constantly communicate with each other – because of that, the majority of the functions and fields will be public in access.
  + Field – List of Squares: The fundamental algorithm within this class is the Marching Squares algorithm. This algorithm works with squares and nodes; this list stores all the squares made from the map array and works in tandem with the list of nodes.
  + Field – List of Nodes: This list works alongside the list of squares to set up the data for the Marching Squares algorithm with the data contained for each node consists of their respective square(s).
  + Field – Dictionary of Marching Squares: A dictionary is required for the marching squares algorithm. In the dictionary, will contain each of the situations where triangles will be generated based upon the status of the nodes made from the map array – having the information in a dictionary allows for easy access within algorithms.
  + Field – List of Vertices: This list will be containing the data on the edges made when building the triangles for the physical map and will be working in tandem with another list holding the data on triangles generated.
  + Method – Square Grid: This method builds the grid of squares and nodes from the data of the map array – this will be also using the list of squares and list of nodes.
  + Method – Marching Squares: This method will be performing the marching squares algorithm using data from the Square Grid method and dictionary of Marching Squares. Depending on the status of the squares and nodes, certain triangles will be generated in the physical map.
* **Camera:** The camera class is there to simply control the camera object within Unity. The camera will be used to orbit the map to provide a visual view on the map generated.
  + Method – Camera Rotation: There will only be one method for this class and that will be for the camera to rotate around the map.
* **Menu Script:** The Menu Script is here to manage the UI for the main menu of the prototype and the Unity components that make up that user interface. The script also handles with the user being able to input a seed into the Map Generator script. Due to how the Unity engine works, each Unity component will require its own method for its role/task.
  + Method – Exit Application: This method, allowing people to close the prototype and return to desktop, will work with two 2 separate components in the menu: A button and a pop-up box – these will act as the “quit” and “are you sure?” respectively.
  + Method – Generate Map: This method will simply act as a trigger to open the next scene in Unity and therefore start the procedural map generation.
  + Method – User Seed Input: This method will allow the player to input a seed for the procedural map generation via a textbox object in the menu; whatever is written into the textbox will be save and then sent to the Map Generator script.

Diagrams

Class Communication  
To show communication between classes, I have created a sequence diagram. The actual communications between classes is would be quite minimal, due to how the Unity engine operates and so the majority of the communication that is done between scripts is to simply trigger another script to start via object within the Unity environment.



**Pseudo Code Algorithms**

