This document comprises of the initial analysis of my project, including deciding what the project will actually be and who it is aimed for. It will also comprise of features that are required for my project to be counted as a success and any problems/solutions that could come up during the process of the project.

**Problem Identification – What is my Project?**  
For my project, I want to create a game. The game will be a first-person cave-crawler in which you will navigate a cave collecting valuables placed throughout the cave before going to an exit to end the game. To develop the game I will be using the Unity engine; the reason for this is that Unity provides an easy-to-use environment to build 3D games while allowing me to still go loose on coding and building algorithms. To build the cave that the player will navigate through, I will be developing a procedural generation algorithm based upon cellular automata that will build the caves for me set by some small rules.

**Stakeholders – What is my project aimed at?**  
My computer science project is being worked alongside a peer group and so my target audience for the project will be those peers in said peer group. These people will be giving me feedback on design ideas and prototypes of my project, providing me with constructive criticism and advice on improvements that I can make. This would make the game being built tailored to them as a group and therefore will appropriate to their needs in a game of this kind. I went to them to form an initial discussion; this would allow me to gather their initial thoughts and comments on the idea and what they would like to see within a game such as this.

As I was discussing my initial project ideas, I thought of including an enemy into the maze; this would be a singular enemy slowing chasing the player across the maze. An idea that I had coming from this was instead of having a specific entity, have an ‘enveloping fog’ as the person described it as. This ‘fog’ will slowly fill the entire maze from the starting point to the end point adding a sort of dynamic time limit. This could potentially be better than a single entity as it can give the overall game concept a good difficulty addition as it is ultimately built on navigation. This however would probably come as secondary, with me starting to work on this feature only after other parts of the project have been completed.

I was later questioned on my method of in-game movement and controls. I didn’t want to add any ‘different’ controls and instead stick to standard first-person controls – using the W-S-A-D keys to move and the mouse to control the camera. An idea presented to me was to have a mash-sprint system; in order to sprint in most games one button must be held or toggled, this idea instead requires the player to mash a designated button to sprint. I didn’t like this idea as the idea of constantly mashing a button just for movement was rather pointless and was like a barrier to the player for movement.

One of the main chunks of the discussion was how I would implement my idea of random maze generation. A problem that came out of the discussion of dead ends; while it is okay to have dead ends in a maze, random generation can potentially cause a situation where it is virtually impossible to complete the maze. An idea to combat this that was discussed was building a labyrinth instead of a maze. This would mean that all ‘rooms’ if I decided to build the cave this way will be connected, allowing all directions to be ways to the exit.

**Research – Inspiration from past creations**There are many games that have been made that take advantage of procedural generation as well as there being many other methods of procedural generation. Looking at these will give me an idea on what is important in my project and may also give me insight into anything else I can include in my game:

Game – Minecraft  
*Minecraft is a sandbox video game originally created and designed by Swedish game designer Markus "Notch" Persson, and later fully developed and published by Mojang. The creative and building aspects of Minecraft enable players to build constructions out of textured cubes in a 3D procedurally generated world­* – Description of Minecraft from Wikipedia, last updated 6 April 2017.

The factor that brings players constantly back into the game Minecraft seems to be the fact that it is a new adventure every time a new game is started – this is primarily due to the procedurally generated world made in every save. The environment is a fundamental resource in Minecraft and having that change every time a new world is built, means that your choices during gameplay will be vary every single playthrough – sometimes you will have a resource rich world or a barren wasteland. With a lot of factors edited via certain characters within seeds, the range of differences are vast.

Algorithms – Cellular Automata Algorithm  
The concept of cellular automata is based upon a gird of cells. The cells within the gird have a finite amount of states, usually the states are 0 and 1 (on and off) but cells can have multiple depending on what the algorithm is being used for. The state of each of the cells are based upon, among other factors, its neighbours. The neighbours of a cell are the surrounding cells, each with their own value as well. Depending on the values on these neighbours will determine the value of the cell – these dependent values are the rules of the cellular automata algorithm. The general process of a cellular automata algorithm is as followed:

* The states of all the cells are initialised, being the same throughout the entire grid – in a binary cellular automata grid, this would be the value 0. A single cell is selected (usually the first at 0,0) and has its state changed to start the algorithm
* From this point and moving across the grid, the states of cells will be changed based upon the cell’s neighbouring tiles.
* This is done until the final cell in the grid is changed in state

The main aspiration from modern usage of cellular automata comes from gaining complexity from simplicity. Something with so few rules can bring about heavily complex data structures as long as it has a high number of iterations. You can include complexity within the rules of the algorithm but in most circumstances, it is not necessary.

In terms of using this algorithm for procedural map generation, it is often favoured for building better maps; since each cell of the grid is modified individually, the finished generations come out as more ‘natural’ in comparison to connecting set and defined rooms.

Algorithms – Marching Squares Algorithm  
Marching Squares is a computer graphics algorithm used commonly within games and visual graphics to generate triangle meshes. The algorithm per triangle is based on 4 separate nodes creating a square – these nodes can be one of two states, on or off (0 or 1). Depending on what nodes are active/inactive, a certain triangle will be made. There is a total of 16 different triangles that can be made (case 0 to case 15).

This can be used in combination with the cellular automata algorithm to create the visual representation of the data structure being made via the cellular automata algorithm – making a physical map.

**Computational Methods – Building Techniques**

Abstraction – Removing Unwanted Detail  
In a game, there are a lot of factors in comparison to the associated reality that are just unnecessary and will simply cripple the gameplay of certain aspects. This leads to a lot of abstraction when it comes to them and this is no exception in my project.

* Movement System – No Respiratory System: Walking and sprinting will eventually cause tiredness and fatigue in a person, forcing them to slow down or even stop in order to gain their breath. Navigating through an unknown cave will have players going back and forth throughout it – having a respiratory system will just slow down the gameplay and will cause it to become simply tedious.
* Camera Control – No Blinking Mechanics: Having the player’s viewing blacked out every couple of seconds is annoying. There isn’t a reason for it within games and it is annoying to use in games that have it. This is not needed and so I won’t include it.
* Cave Generation – Realistic Structure: Caves in themselves are random or ‘procedural’, but they are however set in a stable structure – with unstable structures eventually collapsing. Focusing on whether a specific generated cave is ‘stable’ is simply tedious and will only increase the time and effort for generating a cave/map.

Decomposition – Breaking Down the Project  
Each component of the project is very complex consisting of multiple parts within them. Detailing out and break down each part of the components, it will be easier to develop the entire project – connecting the smaller parts together like a jigsaw puzzle.

* Procedurally Generated Map
  + Part One – Taking User Seed Input: Before I can start building the data for the procedural generation, I will need it to take in another set of data in the form of a seed as reference to build it. This seed will be able to be made by the user via inputting a set of characters.
  + Part Two – Building the Data: The first thing that will have to be done for the map generation is generating the actual data for the procedural generation. The data will consist of randomly placed values that will act as a ‘first draft’ for the currently procedurally generated map.
  + Part Three – Modifying the Data: Once the initial data has been set, the data in place will be changed. This will be due to a set of defined rules for the procedural generation that I would have created – current values which don’t match the rules set will be changed accordingly.
  + Part Four – Creating the Physical Map: With the data set, it can be used in the Unity engine to build a physical rendition of the map in the environment.
* Player Object & Controls
  + Part One – Physical Player Object: A body for the player will need to be required before anything else can be implemented in, it acts as an empty for which more things can be put inside. Here will also have to be the point what the object will look like – some sort of humanoid model or even having the object fully transparent.
  + Part Two – Movement Controls: After create the player object, I can focus on adding the movement controls and mechanics; adding movement controls and mechanics to body already formed would make it easier to see in active use and dynamically tweak while testing.
  + Part Three – Camera Control: With the movement now in place by this point, I will be able to add in a camera and implement the viewing controls for it. This would be easily implemented within the movement controls.
  + Part Four – Collision Physics: By this point the player will be fully controllable and so I would be able to work on interacting with the procedurally generated cave. This will require the procedurally generated map to be completed.
  + Part Five – Spawning: The final part to tackle for the player object and the controls is spawning the player in the procedurally generated map. This should be done after the collision physics have been implemented or else there would be issues with clipping through the environment.

Pipelining – Connecting Processes  
Some of the processes that I have lined out in the decomposition section generate data, this data in most circumstances will be used in other processes to perform their tasks.

* User Inputted Seed to Procedural Generation Data: The seed inputted by the user of the game will be taken by the procedural generation algorithm to generate the map data.
* Procedural Generation Data to Physical Map Build: The data that is going to be generated from the procedural algorithm will be fed in to the algorithm for building the physical rendition of the map within the Unity environment.
* Procedural Generation Data to Player Spawning: The data that is going to be generated from the procedural algorithm will be fed into the algorithm for the spawning of the player object. The algorithm for the player spawning would need to search through the data provided to find a spot where it is safe to spawn the player object.

**Specific Computations – Searches, Calculations & Data Structures**

Searching Methods

* Player Spawning Method: Spawning the player in the procedurally generated map is the only search method that I have lined out. This will require the method to search through the data set made via the procedural map generation algorithm and place the player object in a suitable area. As there would be multiple values in the data set that would be suitable for the player, I have the idea of randomly selecting a space out of the ones that are suitable – this will require sorting these values.

Comparisons

* Procedural Map Generation: A lot of comparisons will be made during the modification process of the procedural map generation – these comparisons are ultimately based upon the rules set into the procedural generation algorithm.
  + Wall Comparisons: During the modification process of the procedural generation, walls will be changed to floors and vice versa. They way this will be done is via comparisons depending on the neighbouring walls.
  + Room Comparisons: The procedural algorithm will generate ‘rooms’ within it, these ‘rooms’ will range from enormous open areas to small rooms that aren’t worth keeping. These ‘rooms’ will have to be compared to the set rules as well as each other to see which ‘rooms’ stay and which ones get converted into walls.

Real-Time Calculations

* Player Object – Collision Detection: Collision detection will allow the player object to interact properly with the walls, being able to hit and be stopped by the walls of the map without going through the walls. This detection has been be constant - as long as the player object is within the map, the collision detection calculations must be active.

Pre-Determined Calculations

* Procedural Map Generation: The calculations for the procedural generation algorithm are static and singular calculations. Once the map generation is complete, it doesn’t have to be re-done again unless the player restarts the game to have a new map. This means that the calculations for it can be allowed to take a while – it doesn’t have to be pushed to be as fast as possible.

Data Structures

* Array Structure – Generated Map: The data for each procedurally generated map will be stored in a 2-dimensional array. Each position in the array will be associated with a co-ordinate within the physical map that will be made from this data.
* File – User Inputted Seed: The seeds that the players input will be saved as a text file. This text file will be able to be read by the procedural generation algorithm to be used to generate the map.
* Hash Table – Seed Hash for Procedural Generation: To read and effectively use the seeds provided by users, the procedural generation algorithm will use a hash table – certain characters will determine certain factors for the initial map generation, combing these characters (words or phrases) will then combine these factors for the generation.
* Dictionaries – Cellular Automata Neighbours: Within the cellular automata algorithm for procedural map generation, the modifications are done via neighbouring points within the generated map. These rules for neighbouring points would be easy to access if it was stored in a dictionary – it would provide an easy reference that can be accessed for all the rules.

**Essential Features – The Three Prototypes**  
I decided to create this project on a three-prototype format, having an iterative development process for each prototype with the subsequent building upon the last prototype. My reasoning for this is to have a more critical development process for each essential part of my game.  
Each prototype will focus on a required feature for my game so by the time the third prototype is complete the game ‘proof of concept’ will be finished.

Prototype One: Map Generation & User Interface  
The first prototype that I would like to create will include the procedural map generator and the user interface of the game and have them at working and functional states. The features that are required in this prototype include:

* **Map Generator:** The most important part of this project is the procedural map generator itself and as such writing the algorithm and building the generator is top priority. To build it, I have decided I will be writing a cellular automata algorithm and create a physical map out of the tile-map created via seeds written by the user.
* **Inspection Camera:** I want to include an inspection camera to provide a visual look at the map generated by the cellular automata algorithm. This inspection camera will allow me to identify any anomalies in maps generated, provided me with a basis to troubleshoot problems.
* **UI Menu:** Another required feature is a menu for the game. The reason why I am building it now is so that I can use the menu to test inputting seeds into the map generation algorithm while in active play of the prototype, including a text box to input words to use as seeds. When later features are added through other prototypes the menu will change to a start point to initiate the game.

Prototype Two: Player Movement System  
In my second prototype, I will incorporate the movement system for the player to navigate through the generated maps. The features required in this system include:

* **Control of Character:** The fundamental of the movement system is the movement itself. This would have the character’s x-axis and z-axis movement being controlled by the user using W-S-A-D keys. This can be done via an object component available in Unity or it can be implemented via code placed into the object. I will build the control via code as this will allow me to better modify the code and make specific tweaks compared to a set component.
* **Camera Control:** Part of the movement system for a first-person perspective is the camera control. This will also allow the player to visually navigate the cave and will also provide the axis direction for x and z axis movement.
* **Collision Detection:** Collision detection prevents the player object from clipping through environments and falling into the infinite void of emptiness. This is vital to implement so the player can actually navigate the map and ultimately play the game.
* **Spawning:** The play entity will have to be able to spawn inside the majority of maps generated by the map generator above the floor of the maps generated.

Prototype Three: Gameplay Mechanics  
The third and final prototype will add the gameplay mechanics to the game. This prototype will also include a revision of the UI. The features that are required for this prototype are:

* **Collectables:** The collectables that are in the game will be functional within the game. This includes the visual design of the collectables and the algorithms to be able to be collected by the player and being able to be spawned in each procedurally generated map. This will add the main gameplay aspect to the project.
* **Play Collection:** An algorithm will need to be added to the player object in order to be able to collect the collectables within the map – the collectables will have to be deleted when the player object touches the collectable.
* **End Gate & End Game:** A way to complete the game and end the game will be added in this prototype. This will take the form of an exit game that will need to be able to be place inside any procedurally generated map without it being inside the walls for example. When the player enters this, they will have to be taken back to the menu.
* **UI Revision:** All aspects of the UI will have a revision overhaul to fully incorporate the added aspects of the game. This UI revision will also include an in-game heads-up-display to present to the player the map generation his is playing on, the number of collectables he has collected and the amount of time the player has spent within that map generation.

**Limitations of the Proposed Prototype**With the plans and layouts that I have made during this analysis document, there are some factors that are going to be complex to the point of becoming a heavy limitation factor – whether it is in the development process or in the finished prototype itself.

Algorithms – Space Complexity & Time Complexity  
The largest limitation factor the algorithms within the project is complexity. Each of these algorithms are going to take time and are going to take a certain amount of space – this is something that is unavoidable. The problem with this is the increasing complexity as more data is added into the algorithm.

* Procedural Map Generation: Space complexity wouldn’t be an issue for the procedural generation algorithm – there will be a set size of the map and so the data structure for the map will be the same size no matter how many generations. Time complexity could be a changing factor however; depending on the initial generations, the amount of modifications made could be very little or very large – while the rules are going to always be the same, what the rules effect will forever change.
* Seed Hash Table: The method of converting user input text into a hash table for use in a seed is a linear process in both time complexity and space complexity – the reason being that it is all entirely dependent on the size of the seed. The longer the seed, the more characters that are needed to be processed, requiring the method to take more time; the increasing length would also require a larger hash table meaning more space required.

Prototype Complexity & Abstraction  
The development of this project is locked to a deadline, requiring a working prototype by the deadline – some features because of this are going to have to be cut:

* Lighting System – Dark Cave & Flares: An initial idea which was being discussed when this project was started was to have a flare system that is required to navigate through the cave, as it would be near pitch black. This is an idea that is not only heavily complex, but is also not needed for the game.
* Player Object – Player Model: A common form of abstraction for games in a first-person perspective is to make the player model invisible; this is usually done for two reasons: convenience and reducing graphical processes. I am following on with this idea, allowing me to save time and focus on other aspects.

**Solution Requirements – What’s in the Final Outcome?**The final solution, after the three prototypes, will be required to perform functionally with the following list of things from the main aspects of the project working in order. These include the following features stated below:

The Map

* **Procedural Generation:** The map itself will be built via the algorithm method cellular automata, with all the maps being fully navigational through with cave spots and large areas being fully connected to each other.
* **Textures:** The textures of the cave will be added to the walls and the floor of the procedurally generated map.
* **Seed Input:** The procedural generation will have a seed input for users to type in their own word(s) to build their own procedurally generated map. If no word is inputted for the seed, the algorithm will get the computer to create its own seed and procedurally generate a map out of that.
* **End Gate:** The map, when generated, will have an end gate positioned somewhere in the map so the player can finish the level when they have collected all the collectables or if they want to exit the current map.

The Player

* **Player Movement:** The Player movement will have to be completely functional with control on the keyboard and mouse using W-S-A-D keys or the arrow keys. The physics will also have to be correct, having a set walking speed and set stop instead of having momentum.
* **Camera Control:** The first-person perspective camera must also be working, with the player being able to move the direction of the camera using the mouse; the direction of the camera will also correspond to the forward direction of the player movement.
* **Collision Detection:** The player entity will also have to have collision detection working – being able to have the ability to impact the walls of the map as well as ‘collect’ the collectables that would be in the map.
* **Player Entity:** The asset that would be use to represent the player entity will be included. Even though the player will have a low chance of actually seeing the visual asset, it is required for collision detection.

The User Interface

* **UI Main Menu:** The menu of the game will be fully functional, with a method of starting the game, closing the application, inputting a word that would be used as a seed for the procedurally generated map and the ability to start the actual game.
* **Heads-Up-Display:** The HUD will be functionally for use in-game, indicating to the player how many collectables they have collected and how long that have been playing in the current map for.
* **Visual Design:** The main menu, buttons and HUD will be designed to fit the aesthetic that would be in the game.

The Collectables

* **Design:** The visual design of the collectables will be included into its prefab.
* **Player Detection:** The detection of the player will have to cause the collectable to be ‘collected’ and disappear from the map when the player comes into contact with the collectables.
* **Collectable Spawning:** The collectables will have to be able to spawn in any procedurally generated map in areas that will be able to be access and therefore the collectables can be obtained.

The game itself will most likely not be a demanding program hardware wise, the only hardware that would most likely be required is dedicated graphics of some kind, whether it would be a graphics card or integrated graphic memory. As Unity is able to build the project into an application for the majority of computer operating systems, no specific OS or software is required – the game can be built for Windows, Mac and Linux for all architecture types.  
I came to these conclusions from the focus group peers that I will be creating this project beside. The initial focus group discussion I had laid out ideas and refined others for what my game could be and what would be in the game to be considered a finish product, making the game tailored to their wanted experience – this information acquired from the focus group allowed me to set up what would the final requirements for the project.

**Success Criteria – What counts as a success?**As the project will be built upon three prototypes, each prototype really needs its own checklist to see what needs to be counted as a success. In order to test whether these prototypes are meeting the success criteria, I will get my focus group peers to test out the prototypes. As I am working on the project with them giving me feedback and criticism, the project will be tailored to them and therefore will have to meet their needs.

Prototype One  
For the first prototype to be counted as a success, the map generation will have to be successful. The generator must be able to build maps that would allow the player to navigate across the entire cave; there can be no closed off areas that would be in-accessible to the player. While the UI and the navigation camera are important, they do not necessarily need to be functioning as those elements are for the sake of easily checking the map generation.  
I will be testing my prototype with the following test(s):

* I will allow the peer group to access the prototype, letting them input their own seeds to see if the algorithm in the project can:
  + Generate a map with seemingly any word
  + Being able to generate usable maps, with cave spaces connected with tunnels that would be able to be accessed from
* When the prototype is active, the peer group will monitor the camera that will be rotating around the map. They will determine if they are able to see the map and will use this for the other test above if successful.

Prototype Two  
The second prototype focus on the player and the mechanics in the player entity; for this to be a success, the player entity must be fully controllable. The forward and backwards movement should be connected to the first-person perspective camera, giving the entity the ability to turn. The collision detection must also be working, preventing the player entity from clipping through the map. In order to actually test whether the player entity and mechanics are available:

* Spawn the player in a testing box, the peer group will be able to control the entity to test the controls of the entity. They can determine whether the movement speed and momentum is of a good standard, the box will also be there test the collision physics.
* The player entity will also be spawned in a generated map (which is a test in it of himself) and the same test as stated above – this will test it in the active game environment.

Prototype Three  
The third and final prototype is the closest thing to the final solution, this will include new UI revisions, the collectables in the game and start-game/end-game. All of these will have to function together without fault. Any problems can possibly cripple the rest of the prototype and as such all aspects should be tested.

* The peer group will be allowed to play the entire prototype in whatever way they choose, doing anything within the grounds of the prototype. They will be taking down notes on the game, taking priority on the following topics:
  + Functioning UI – whether the buttons and the seed inputs are functional
  + Collectables – If there were any problems with collectable spawning and/or collecting said collectables
  + End Gate – Being able to exit the game via the end gate
  + Bugs & Glitches – Any other issues that don’t apply to specific features or topics (e.g. clipping)