T.C.
MARMARA UNIVERSITY
FACULTY of ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

CSE4197 Project Specification Document

Title of the Project

"PROCEDURAL WORLD GENERATION METHODS FOR VIDEO GAMES"

Group Members

150119023 Mehmet Emir Şahin

150119047 Mustafa Can Türker

150119769 Ömer Yiğit Tatlısu

Supervised by

Doç. Dr. Ali Haydar Özer

Table of Contents:

1. Problem Statement

In our project, we will write a series of algorithms that will create ready-to-use procedural generated worlds for games. There are lots of solo developers who need this kind of tool and there is almost no open-source project like this. Developers are not artists, and they usually hate world design when developing their games and don't feel like they need to focus on this part too much. This tool will decrease their workload and allow them to spend more time on more complex aspects of their game while our project takes care of world creation.

2. Problem Description

The gaming industry is getting increasingly competitive for small groups and even more so for solo developers, as these groups spend more effort on satisfying the increasing demands of the fast-paced market of gaming font. Their workload becomes a burden on their lives and in some cases, the games they are developing will be stuck in the early stages for years to come. The games are released after 5-10 years of the development process, and this is a big problem. Most developers are inexperienced when it comes to using artistic tools. They mostly delay the world creation part until they are done with the other parts, and they will use already available assets from other people and fuse them into their project and the result will mostly be mediocre. Our project focuses on curing this problem by creating unique and extendable worlds using simple parameters as input

There are tons of game types and every one of them needs a different type of world. Our project will let the developer create worlds that fit their project.

Another noteworthy feature of our world creation system is that it is free. This means that developers with little to no budget will be able to use the software for free. Also, since our project will be open-source, it can be improved by other developers.

3. Main Goal and Objectives

Our main goal is to create a convincing game world that will be useful to developers with given parameters and create a basic game to test this world.

Objectives:

1. Creating a convincingly detailed world and doing so in a way that gives the developer the freedom they want
2. Populating the world with uniquely named characters and objects.
3. Generating a world history that will make the world more believable and will be unique to that generated world.
4. To be able to statically export the created world to use in other games and other projects.

5. Develop a simple but complete game where we can present all the features of our world-building system in a comprehensible way.

4. Related Work

World creation using procedural generation is a feature that is considered niche even for today's works and is mostly implemented by indie and solo developers for their games. Minecraft [1] is one of the most popular such games, but the real inspiration for the project came from Dwarf Fortress [2], a game that uses ASCII characters as UI images, characters, and tiles.

The dwarf fortress is a hobby project that has been in development for 16 years by two brothers, Zach and Tarn Adams. It can be downloaded and experienced for free by downloading over the internet. The game features a world creation system with an unprecedented level of detail and algorithms for creating history and characters for the world. However, like most projects, the game is not open source.

Another interesting game for a game programmer is "Cube World" [3], another game with a world generation system. This game is an indie game, like most games with unique features. Developed by Wolfram von Funck and his wife Sarah von Funck, this voxel-graphic game's 3D game world, which is very consistent within itself is its biggest feature. The game excels at creating worlds that support the RPG elements in its gameplay.

We will design the physical world creation part of our project using some of the methods used in the voxel-based world creation systems of games like Minecraft and Cube world, which are also common in other games of this kind, and then we will design the history and character creation system like in Dwarf Fortress using various procedural generation techniques.

5. Scope

The project is mostly encapsulated in a way to make it seem like a world creator engine. But beyond the aspect of procedural generation of a game world, we wanted the result of the project to be more of an interactive simulator or even a type of fully-fledged game (depending on the situation and the genre; or the fact that the result is a game can be discarded, as the mainstream idea and the most important aspect for the project is to create a world). Besides the main code and the script to start the program, there are some other small functions and scripts that make the experience a more polished one. For example, these functions and scripts, other than being able to have control over the world; like making the water level so high that it becomes a water world with the mountains becoming a little island, to other quality of life type improvements such as reducing the strain on the computer by trimming the unseen, unimportant and unwanted polygons from the screen, so that the user can have an experience without lag or weird immersion breaking constructs.

If we put the overall idea in a summarizing sentence the nexus of the project is the world. The desired aspects for the result are an immersive procedural generating engine and if possible, its intractability. The miscellaneous aspects are various scripts and functions that will polish the result in any way possible. If the above ideas are realized the project will be finished in its full scope.

5.1 **Constraints**

- The algorithms will only work on the Unity3D engine if no code adaptation is done for any other engine.
- If the created world will be used in the development of a game, the algorithms of this project should be imported into that game project.
- The algorithms must run on a Windows computer. Multithreading methods for the algorithms will be platform-specific.
- If the developer wants to extend the algorithms, this should be done in C#.
- If the created world is to be used directly in another game, that game must have the option to import game maps in the same file format as this project's export.
- Although the necessary optimization will be done within the project, the project algorithms need to be run on a computer with enough processing power. Especially for larger worlds, high RAM capacity and a fast processor are required.

5.2 **Assumptions**

It is assumed that,

- There will be enough development time for the history creation system.
- Markov Chain [4] will lay the groundwork for unique name creation.
- The Greedy Meshing [5] technique is supported by the 3D graphics engine that will be used.
- The Unity Jobs [6] system, which will be used to implement multithreading, is stable enough to be used for a medium-sized project.
- In any stage of procedural generation, no machine-learning model will be needed.

6. Methodology and Technical Approach

We will write this project in C# and use Unity3D as the graphics engine to spawn world objects and to make optimizations in the later stages more comfortable. This engine will also make it easier for us to develop the demo game, which is one of our objectives
We will use these methodologies and technics:

Perlin Noise

We will use Perlin Noise [7], which is also seen in the precedents of the project we will develop, in many stages of our project. Noises are sets of random values over some fixed interval. They can be thought of as 2D matrices with random values on them. The values of each element (or 'pixel' for the 2D image analogy) are defined in grayscale with white color representing zero value and black representing 1. Some examples of noise can be seen below:

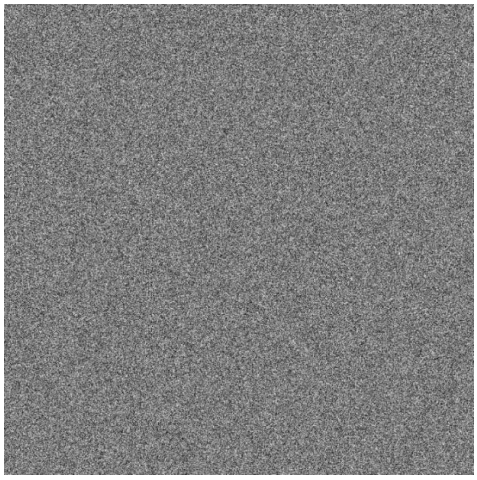

Figure 1. An example of noise that can be used for random generation. The values (pixels) on the 2D matrix are independent and therefore inconsistent.

Noises can be self-consistent. Such noises are used as procedural generation methods. Other types can be used for a random generation as they return pseudo-random numbers. The special noise we will use in our project, Perlin Noise, is self-consistent and will be used as the basis for our world creation algorithms.

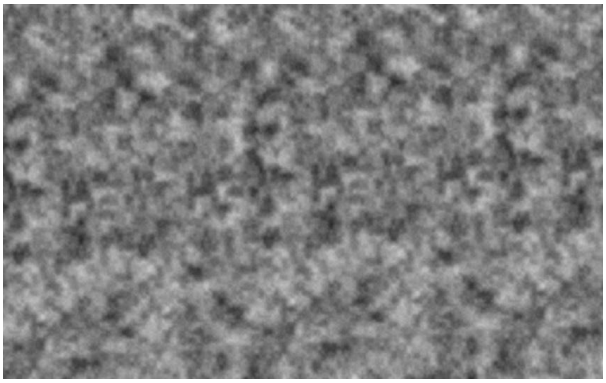

Figure 2. An example of self-consistent noise

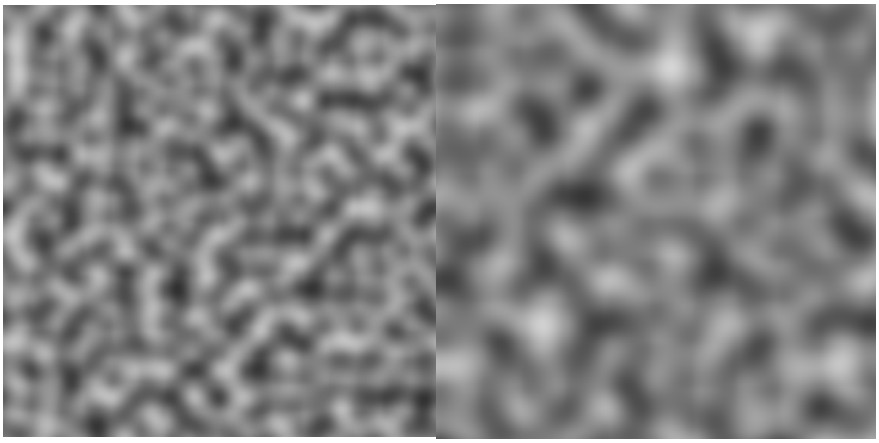

Figure 3. Perlin noise with a scale of 16 and 50 in order

Perlin noise can be used to create plains, mountains, seas, continents, and even caves in procedural generation projects by changing its parameters and using it in multiple layers.

The following example shows a single layer of Perlin Noise with a single layer of Perlin noise. What is done here is that by extracting values according to x and y coordinates from the Perlin Noise created using certain parameters, the vertices defining the plane are raised and lowered according to the values received.

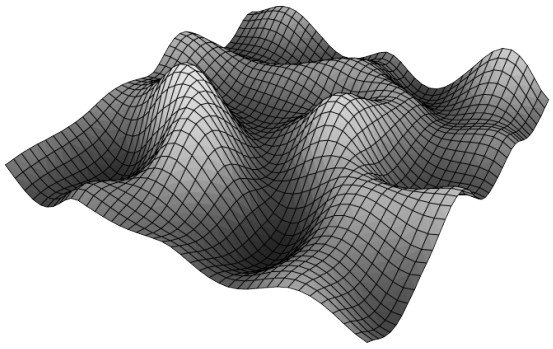

Figure 4. Single-layer Perlin noise practice on 3D terrain

Unlike the example above, we will use multiple layers of Perlin Noise in our project. In addition, we will use voxel-based graphics so that we can get more visually satisfying results and compare them to the worlds in the games we have been inspired by in this genre.

Our world creation system will also allow the user to create 2D worlds instead of 3D. Once the user decides to create a 2D world, they will enter the other necessary parameters into the system and the 2D world will be created. The parameters will be the same as for 3D world creation.

The same algorithm will be used for two-dimensional world creation, but the algorithm will communicate differently with the graphics engine and game objects. Unlike the algorithm's principle of working in three-dimensional space, it will work with pixels instead of voxels, and each pixel will be modified on the same 2D plane for the two-dimensional map, as opposed to creating a new 'game object' for each voxel.

When creating a 2D world, Perlin Noises will be projected directly onto the 2D plane instead of being projected from the 2D array to 3D. The technique of using Perlin Noises in multiple layers will also apply to two-dimensional world creation.

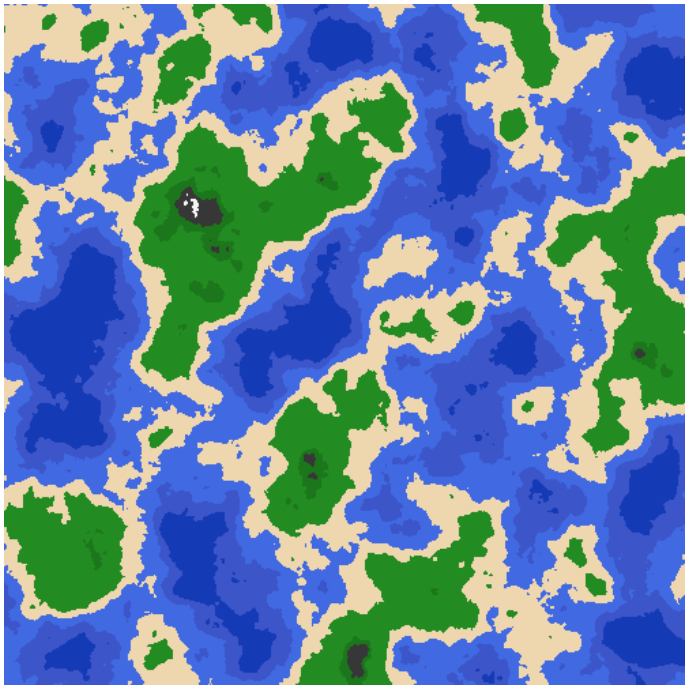

Figure 5. Results of using multiple Perlin Noises for a 2D top-down terrain with color

Greedy Mesh Combining

Since our project will use voxel-based graphics, an independent object will be created for each voxel in the scene during and after world creation. The total number of these independent objects can be in the tens of millions depending on the world to be created. Since each object will have its position, rotation, and 3D material, it may be impossible to perform the algorithms to complete the world creation phase on personal computers. To run such algorithms on a personal computer, many optimization methods should be used during the development. The most effective of these methods is called “Greedy Meshing”.

In Greedy Meshing, the meshes of static objects in the scene are combined into a single mesh and this mesh is drawn with the minimum number of triangles possible. An example of this can be seen below

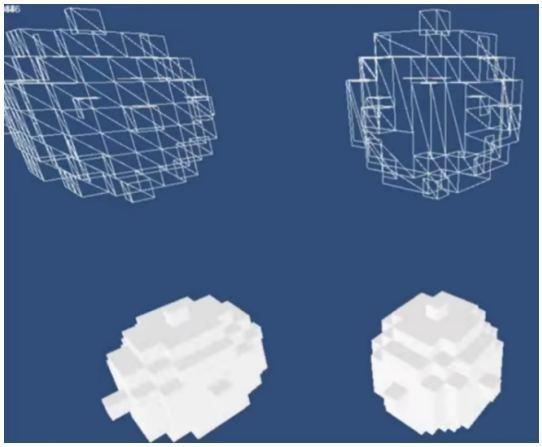

Figure 7. An example of Greedy Meshing

Other Techniques

This project will use the chunking method. This method will disable chunks of voxels that are not close to the camera source. In this way, the load of the created world, especially on RAMs will be reduced. At the same time features that are common among them can be assigned to the voxels in the same chunk. Different biomes (the name given to grouped voxel chunks) can be created. For example, chunks in the rainforest biome will have more trees and the land will be shapeless compared to the other biomes’ chunks.

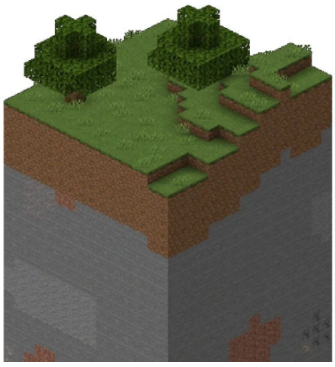

Figure 8. An example of a voxel chunk from the game “Minecraft”

In addition to the chunk system, we can use Unity3D's 3D functions to combine meshes that have the same property at runtime into a single game object. In this way, instead of each of the voxels in the same chunk having position, rotation, and 3D material, we can use these values only for the newly formed singular object. This will reduce the resources used for the 3D data.

In our project, the optimization part will take up most of the programming. Another method we will use for optimization is to draw only the parts of the created world that can be seen. For example, in the game project where the generated world will be used, if the lower layers will not be visible in any way, the voxels in those parts may not be created at all. This method is simple and provides the biggest performance gain compared to the previous methods. However, we will leave this optimization method optional since depending on the project in which the world will be used the formations in the lower layers of the created world may also be important to the design of the game.

Multi-Threading

Creating a detailed world is an action that uses a lot of processing power. To do this on personal computers, we need to maximize the power of modern processors. To achieve this, we need to implement multi-threading in our project.

The first way that comes to mind for multi-threading in this kind of project is to split chunk creation into threads. This way, multiple chunks can be created simultaneously on multi-core processors thanks to the nature of multi-threaded applications. Since we will write our project in C#, the most appropriate method to achieve multi-threading is to use Unit3D's Jobs system. However, Unity Jobs is an undocumented system. It is also unclear how stable it works. For these reasons, it's not clear whether we'll use multi-threading for optimization, or at least whether we'll use Unity's Jobs system for this.

Modifying Vertices Instead of Voxels

As an option that will help solve performance issues and work similarly to the 2d world creation logic of the project, instead of creating a large number of voxel objects in world generation, we will offer to edit a plane using its vertices according to the parameters entered.

The problem with using a single terrain object and editing its vertices instead of creating a 3D object for each element of the terrain is the inability to control the details of the chunks and biomes. Also, the result of this approach will look less satisfying compared to the voxel terrains.

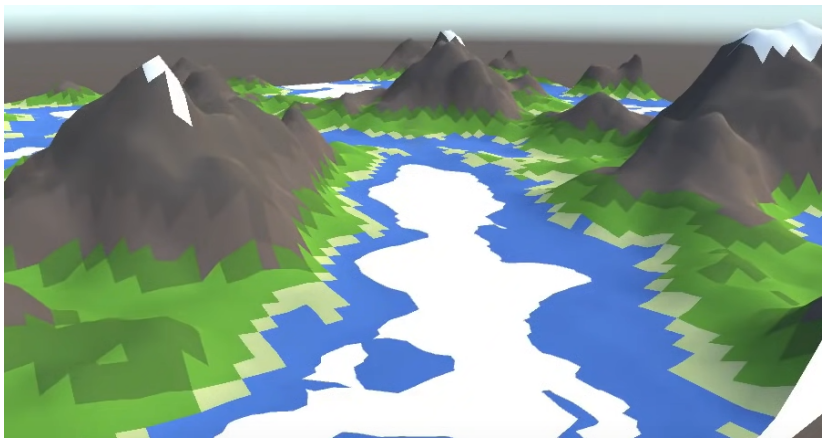

Figure 9. Using a plane instead of chunks of voxel objects as a terrain.

Procedural Text Generation

As we stated, in the later stages of the project we will be extending our world creation system with procedural name, character, and date creation features. In the first phase, we plan to code the procedural name creation. We will use the Markov Chain model for this.

Markov chains are called this way because they follow a rule called the Markov property. The Markov property says that whatever happens next in a process only depends on how it is right now (the state). Using Markov Chain, we can determine the suffix based on the current state of the string and create names accordingly. We plan to set certain rules for names and create strings by determining the suffixes that will come to the strings according to these rules.

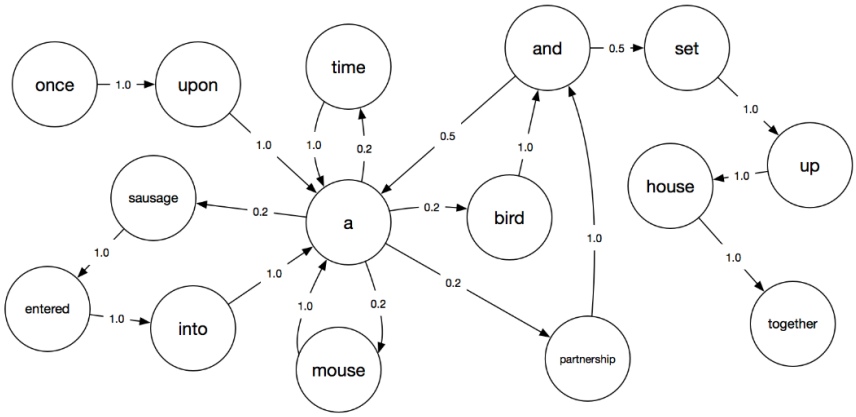

Figure 10. Fairy tale example of Markov Chain

One of the most crucial parts of the project is writing the history of the created world. To achieve this, it is necessary to consider the possible interactions of all factions in the created world. Since we cannot find an open-source example for such a system, we will have to design it completely by ourselves.

Figure 11. A Highly sophisticated history creation example from Dwarf Fortress.

Another approach to writing the history of the world is to simulate the world starting from year zero. This would be easier than creating the world in its final form and then writing history for it but simulating it would add complexity to the project.

While creating history, we also aim to write personality traits for important and unique characters that live in the generated world. To achieve this, depending on the state of the project at that stage, we can code deep algorithms or write the traits randomly but as consistently as possible according to some rules we premade.

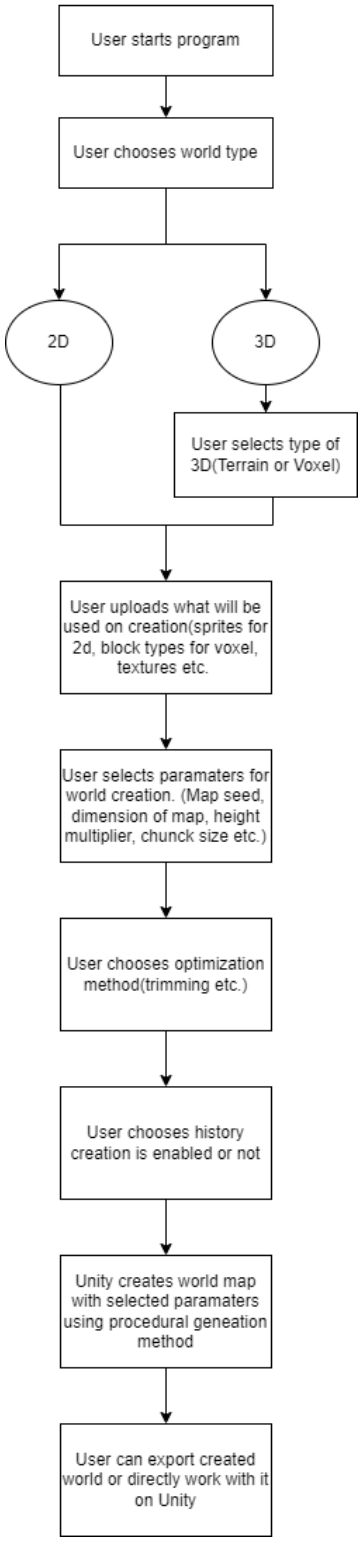

Figure 12. Flowchart of project

In the end, the project works like the above flowchart using the methods listed above. The first user will select the type of the world, if 3d is selected, the user needs to choose the type of 3d (Unity Terrain or Voxel). After selection, the user needs to upload base models for creation (blocks type for voxel, sprites for 2d) and textures. After entering base parameters, the user needs to enter another set of parameters that controls the creation of the world (dimension of the world, chunk sizes, etc.). After every parameter is entered, the user can choose optimization if he/she wishes. Users can add history for their world and Unity will create a world with parameters. After the world is created user can export a map or directly work on it.

7. Professional Considerations

7.1 Methodological considerations/engineering standards

- We will use a version control system called Git; because it helps us to coordinate and work together.
- For visualizing the output of our code we will use a game engine called Unity, it is free and user friendly and it is very popular. We will use built-in methods from Unity.
- C# will have used for the main code because the game engine Unity is built upon C#. As for ide, we will use Visual Studio because they are very user-friendly and can be easily connected to Unity.
- For visualizing our ideas, we will use flow chart diagrams.

7.2 Realistic Constraints

7.2.1 Economical Constraints

There is almost no open-source project that is similar to ours. Open-source ones are free and ours will be free too. We are not planning to profit from this project, we will release it as open source so everybody can use it and improve on it. With our project game, developers will spend fewer resources to implement this part of their projects. Small companies can create more games and with that, they can get more profit

7.2.2 Environmental Constraints

This project has no environmental impact.

7.2.3 Ethical Constraints

We will use only open-source projects as references and our written code. Open source projects which we are looking at are not protected by patents they are using public concepts and designs. Also, this project will not violate security or privacy because it is not using any data from the user's computer. It just creates the world from the input. Because our project is open source, there is actually no real threat to users like hacking, etc.

7.2.4 Health and Safety Constraints

The project doesn't pose health risks to any user because it is software that only aims to fulfill certain virtual tasks.

7.2.5 Sustainability Constraints

In theory, we will need to update codes when the Unity engine updates but in reality, Unity engine updates do not affect the code significantly therefore there won't be any sustainability concerts for this project. This project will live for a long time since it will be open-source and every developer can improve it.

7.2.6 Social Constraints

This project is for every game developer who wants to use it, we don't specify any gender, race, etc.

7.3 Legal Considerations

We don't need any permission to create our project. Everything we use is public and free and we are not using any data from users.

8. Management Plan

8.1 Description of Task Phases

Phase 1: Searching already existing algorithms and projects which were created for procedural generation, including Perlin Noise layering and Markov Chain.

Phase 2: Creating a simple world creation demo with the researched methods. This demo will only have basic features and no performance concerns.

Phase 3: Preparing project specification document (PSD).

Phase 4: Improving world creation demo while looking for bugs etc. The world generation system must be brought to a level of variety and detail that meets the predefined standards.

Phase 5: Optimizing world creation demo for better performance. In this stage, more performant ways of world creation will be deeply researched. The Greedy Meshing and Multithreading techniques will be the main implementations of this phase.

Phase 6: Implementing history creation algorithms and name generation. This phase will be the longest single phase of the project. Dwarf Fortress-inspired history creation algorithms will be written in this phase.

Phase 7: Making the created world exportable and game ready.

Phase 8: Creating a demo game for testing the created world**.**

8.2 **Division of responsibilities and duties among team members**

Team Members	**Task Phases**
Mustafa Can Türker	**1, 2, 3, 4, 5, 6**
Mehmet Emir Şahin	**3, 4, 6, 7, 8**
Ömer Yiğit Tatlısu	**3, 5, 6, 7, 8**

8.3 **Timeline**

Every cell represents a week

Phase/Date	Sep				Oct				Nov				Dec				Jan				Feb				Mar				Apr				May			
Phase 1																																				
Phase 2																																				
Phase 3																																				
Phase 4																																				
Phase 5																																				
Phase 6																																				
Phase 7																																				
Phase 8																																				

9. Success Factors and Risk Management

A. Measurability/Measuring Success

I – Success Factor for Objective 1: The Algorithms need to create a world from scratch which obeys the rules of physics.

II- Success Factor for Objective 2: Added parameters need to work correctly.

III- Success Factor for Objective 3: History creation algorithms need to create history for every part of the world.

IV- Success Factor for Objective 4: Created world can be ported to an already existing game.

V- Success Factor for Objective 5: We should create a new playable game in which we can test the world algorithm.

.

B. Risk Management

POSSIBLE RISK	RESOLUTION
We assumed that there would be enough development time for the history creation system. Depending on the time taken by other phases of the project, there may not be enough time for this part. The ability to create history is one of the most important features of this project. This is why it would be a huge problem if there will not be enough time left.	If we run into this problem, we can develop our history creation system using more inconsistent and primitive algorithms. It would take away from the motivation of our project, but we shouldn't completely remove this feature from the project.
The multithreading system we will implement in the project (Unity Jobs) runs the risk of not being stable enough to run on a project of this scale.	In that case, since it would be impossible to run the algorithms for larger worlds on a personal computer, the user (game developer) would have to use cloud computers to overcome it.
For the algorithms in the project to work on a personal computer, we need to perform aggressive optimizations. Using the Greedy Meshing technique, we will ensure that the created world puts less load on the RAM and GPU. However, Greedy Meshing runs the risk of being too complex for us to become proficient at using it.	We can try different optimization methods instead of Greedy Meshing. With the method we call Trimming, we can save RAM by not creating voxels that will never appear, and we can also search for new optimization methods in case those voxels need to be created.

10. Benefits and Impact of the Project

Solo game developers will use this project for their games. They will run the algorithm with their parameters and it will give a world for them then they will implement this world in their games. Or if they are mod developers for other games they can implement this world in other games.

I. **Scientific Impact:** This project will not be released on any scientific paper. We don't expect any scientific impact.

II. **Economic/Commercial/Social Impact:** With the usage of this project workload of developers will decrease and they can focus on other parts. This will impact their economy and time management in a good way

III. **Potential Impact on New Projects:** Because our project is Open source it can lead to lots of potential new projects. There can be more detailed world creation algorithms.

IV. **Impact on National Security:** This project isn't about security so it doesn't have any impact on this part.

11. References

[1] Minecraft Interactive Experience. (no date). Minecraft Wiki. Retrieved November 29, 2022, from https://minecraft.fandom.com/wiki/Minecraft

[2] *Bay 12 Games: Dwarf Fortress*. (no date). Bay12games. Retrieved November 29, 2022, from http://www.bay12games.com/dwarves/

Bay 12 Games: Dwarf Fortress Talk. (no date). Retrieved November 29, 2022, from http://www.bay12games.com/media/df_talk_combined_transcript.html

[3] Wikipedia contributors. (2022, August 20). *Cube World*. Wikipedia. Retrieved

November 29, 2022, from https://en.wikipedia.org/wiki/Cube_World

[4] Wikipedia contributors. (2022, November 23). *Markov chain*. Wikipedia.

Retrieved November 29, 2022, from https://en.wikipedia.org/wiki/Markov_chain

Markov Chain - Procedural Content Generation Wiki. (2016, February 14). Wikidot. Retrieved November 29, 2022, from http://pcg.wikidot.com/pcg-algorithm:markov-chain

[5] Mikolalysenko, A. (2015, January 23). *Meshing in a Minecraft Game*. 0FPS. Retrieved November 29, 2022, from https://0fps.net/2012/06/30/meshing-in-a-minecraft-game/

[6] Technologies, Unity. (2018, June 15). *Unity - Manual: C# Job System Overview*. Retrieved November 29, 2022, from https://docs.unity3d.com/Manual/JobSystemOverview.html

[7] Wikipedia contributors, *Perlin noise*. (2022, November 2). Wikipedia. Retrieved November 29, 2022, from https://en.wikipedia.org/wiki/Perlin_noise

Mount, Dave, (2018). *CMSC 425: Lecture 14 Procedural Generation: Perlin Noise*. Cs.umd.edu. Retrieved November 29, 2022, from https://www.cs.umd.edu/class/fall2018/cmsc425/Lects/lect14-perlin.pdf