Relic

AI and Soundscape in an RTS

Interactive Media and Game Development
Computer Science

A Major Qualifying Project Report
submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

By:

Lucien LeMenager

Alfredo Porras

Evan Saccoccio

Cordell Zebrose

Advised By:

Professor Charles Rich

Professor Keith Zizza

Abstract

Relic: The Aetherian Uprising is a mixed genre game with elements from real-time strategy games (RTS), multiplayer online battle arena games (MOBA), and god games. The player can cast impressive miracles to sculpt the landscape, control a massive avatar, and give objectives to a hoard of Aetherian troops. The game was built using Unity and makes use of the terrain feature to allow smooth changes in the landscape. It includes detailed UI elements, a stunning landscape, and a dynamic multi-stem music system.

Acknowledgements

We would like to thank our advisers, Professor Keith Zizza and Professor Charles Rich, for encouraging us and keeping us focused throughout the project.

We would like to thank Anthony Russo for helping us with the rigs on the Avatar and behemoth. Thanks to all of our playtests who gave use valuable feedback during critical stages of the project.

We would like to acknowledge all of the various tools and assets which we made use of during the project: Unity, NGUI, Eric Marchesin’s A* Code, FreeSounds, SoundDogs, CGTextures.com, Nobiax Free Textures, and 7 Simple Magic Effects by Rilem.

Table of Contents

Table of Figures

1. Introduction

Relic: The Aetherian Uprising is a medieval fantasy god-game with RTS (Real-Time Strategy) and MOBA (Massive Online Battle Arena) inspired elements. Two factions, the Aetherians and the Gremlocks, are set to battle against each other while seeking to gain control of certain artifacts called relics (See Figure 1) that grant supernatural powers to whoever possesses them. The game features the ability for the player to use these relics to alter the terrain and change the course of battle. The game also offers an immersive experience through the use of sound.

Figure 1: Relic Main Menu

1.1 Gameplay

In *Relic: TAU*, the player assumes control over the Aetherian army in order to destroy the evil Gremlock forces. The player must use their Avatar as well as their available allied troops to capture resource points and take out the enemy headquarters.

While playing, the AI controlled enemies will try and counter the player's attacks with their own offensive strategy to destroy the player's headquarters. The player can directly control where their Avatar moves around the level. They can send him to help capture resource points or to take out enemy buildings by left-clicking where they want him to move. The player also has influence over the where the Aetherian troops go by moving "rally flags" on the map. Along with using the Avatar and troops to defeat the Gremlocks, the player can also capture the four different relics placed all over the map in order to use and cast different "miracles" to turn the tide of battle. The four miracles include summoning a volcano, dropping a meteor, causing a flood, and healing. These can be used to alter the terrain of the map and gain the upper hand on the enemy. The movement of the player's troops and the capturing of resource points and relics allow for different strategies to win the game.

1.2 Design

Relic: TAU was created over a nine month span by a team of four students, two programmers and two artists, who also served as sound composer and designer. Preproduction began in early August 2012, a few months after the formation of the team. Production of the game began later in the same month and wrapped up in April of 2013. The project was overseen by two advisors, Professor Keith Zizza to help with sound composition and design and Professor Charles Rich to help with programming. We also received the help of another art student during production for work in character animation.

The original goal of our project was to create an RTS-style game which had unique gameplay mechanics including real-time terrain modification and dynamic sound. The most important technical aspect was to create an interesting Artificial Intelligence (AI) component which would be able to counteract player strategy and provide interesting and challenging gameplay. The most important art aspect was to create a highly detailed and visually pleasing 3D world with a cohesive art style. These aspects would lead to a more balanced game with solid visuals and interesting gameplay.

During the initial design phase we brainstormed about what the core mechanics of our game would be. We knew we wanted to create an RTS-style game and that we also wanted to put a lot of emphasis on terrain modification but we did not have a solid plan for incorporating these elements into a cohesive game. When we began building the game, we saw the potential of terrain modification as a means of drastically changing gameplay. We eventually decided to focus less of our efforts on making a true RTS game and more effort on making the terrain modification an essential aspect of how you played the game.

Our game began to change shape when we really decided to create a game with elements of different game genres such as MOBAs or RTSs. We were inspired by games such as *League of Legends*, *DOTA 2*, and *Black and White 2*. However, instead of just creating a copy of one of those games, we decided to include some different gameplay elements of those types of games which we thought were the most fun. For example, we took the general map shape, a map that has three distinct paths with the enemy and ally HQ on either end, from MOBA style games. We included a light weight version of RTS-style troop control though the inclusion of “rally flags” which allowed the player to dictate allied troop movement.

Early on in our design, we decided to tie the terrain modification to these “miracles” which the player could cast in order to change the shape of the battlefield on the fly. We came up with the idea that each miracle would be tied to these “relics” which are represented as control points all over the map which the player could capture. When a player captured that relic, they would earn that particular miracle power associated with that controlled relic. The player could also lose control of these relics which would provide another interesting aspect of gameplay and strategy.

Another important change from our original design was the position of the camera. At first we had the camera locked to the Avatar character at close proximity. After experimenting with different angles and positions we decided to detach the camera from the Avatar which provided a free roaming experience and better viewpoint to survey how the battle is going. This was a very important design aspect going forward because we were able to get a better sense of scale and determine how different things should look from that point of view.

2. Game Design

The gameplay and design of *Relic: TAU* was influenced by our backstory as well as our core gameplay mechanics. Other similar games also had a strong influence in the style, combat, and overall structure of our game.

2.1 Story

The story is set in a fantasy world where there exist two intelligent races. The Aetherians are a technologically advanced race of humans who only wish the betterment of humanity. The Gremlocks are a race of cold lizard-like gremlins who value strength and brutality. They strive to hold domain over the land, taking it by force.

After being able to produce a behemoth, the Gremlock were able to seize the homelands of the Aetherians. Left with few choices, the Aetherians called upon their god to be manifested in the form of an Avatar to bring an end to the tyranny of the Gremlocks.

While in your physical form you have lost most of your godly powers, but you can regain them through finding relics that will allow you to perform specific miracles. These relics are seeds of power that you had planted in the world for you to find. By manifesting in a physical form, you can have your powers on the physical plane; otherwise you would never have access to your powers.

2.2 Gameplay

The majority of the gameplay in *Relic: TAU* lies with directing the player Avatar to capture resource points and relics in order to be able to help your armies by casting miracles. Figure 2 depicts the Avatar being directed by the player to fight against the enemy Gremlocks.

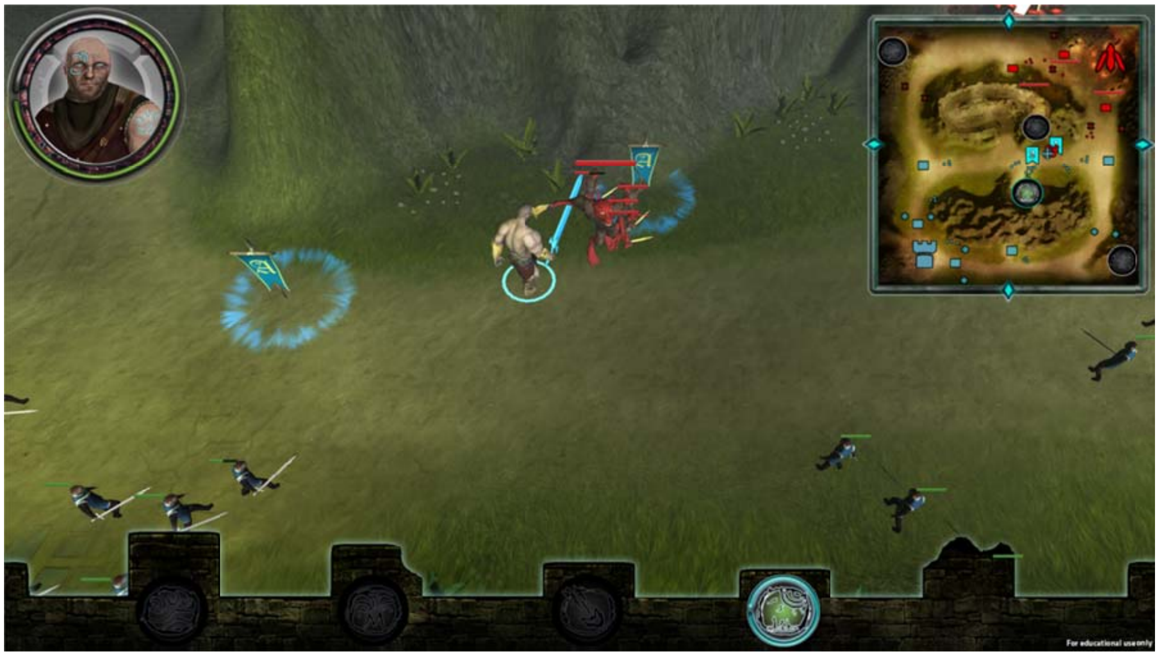


Figure 2: Screenshot of *Relic: TAU*

2.2.1 Relics and Miracles

There are four different relics located in different areas of the map. These relics coincide with the four different miracles which the player can use to help them defeat the enemy Gremlocks. The player can capture these relics by entering and defending the control circle surrounding the relic until the circle changes to the Aetherian color. The color of the control circle indicates who controls the relic. If the control circle is a blue hue, the Aetherians control the relic. If the circle is gray, this indicates that no one controls the relic and it is neutral. If a relic has been captured by the Gremlock forces, then the circle shines red. All of the relics are neutral at the start of each game. The location and status of each of these relics is revealed on the Minimap located at top right hand corner of the screen. Once a player has captured a relic, the corresponding miracle power will show up on the UI miracle selection bar at the bottom of the screen.

Each miracle has its own cooldown time which limits the time between castings. Along with having a cooldown time, each miracle takes a different amount of time to cast. The cooldown and casting times are both indicated by the user interface.

1. Volcano Miracle

Although the volcano miracle is one of the more difficult miracles to attain, it is also one of the most effective. As depicted in Figure 3, the volcano miracle summons a large erupting volcano when cast on the terrain. The volcano immediately destroys any enemies and allies in its path as it pulls the terrain upward. A volcano also has the potential to destroy turret structures. A large volcano can block off different sections and paths of the map for a period of time.

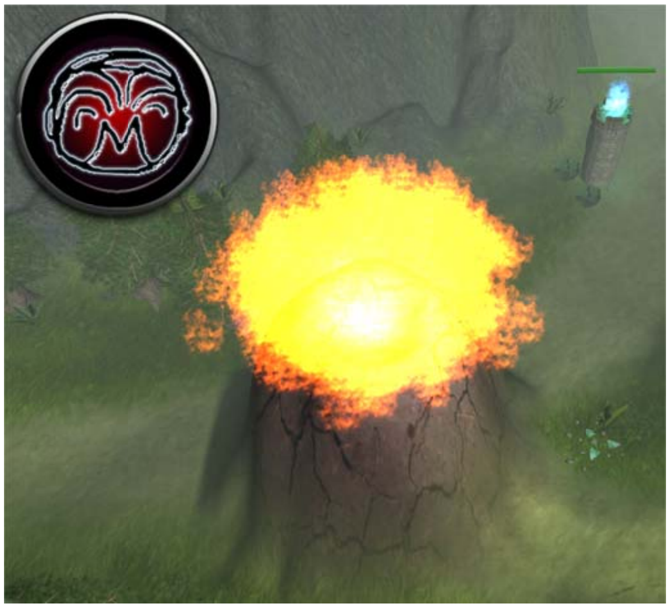


Figure 3: Volcano Miracle

2. Meteor Miracle

The meteor miracle can be used in a similar way to the volcano miracle. As shown in Figure 4, the meteor miracle immediately drops a large meteor onto the battlefield. Like the volcano miracle, the meteor miracle also destroys any enemy or allied troops in its path. However, it does not have the potential to destroy turrets. Once a meteor has fallen, it creates a large crater in the terrain causing the affected area of the map to become impassable for a specific period of time. The meteor relic lies at the top of a mountain which can be reached through a path which can be accessed in the middle of the map.

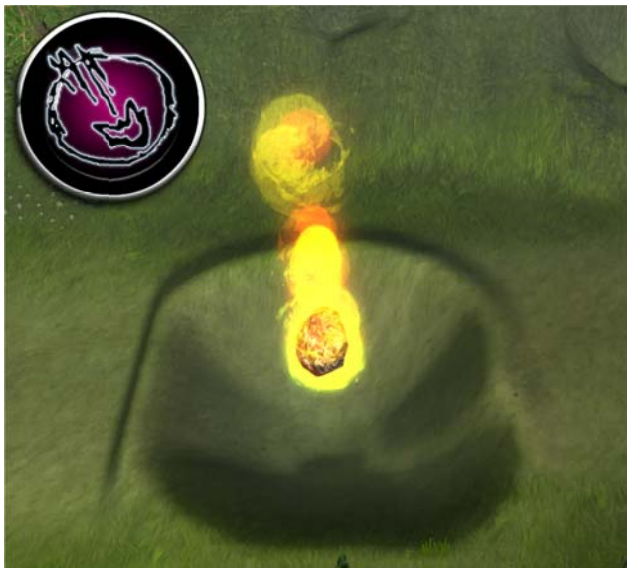


Figure 4: Meteor Miracle

3. Flood Miracle

The flood miracle is the easiest miracle to get but it can also become the most useful. As shown in Figure 5, the flood miracle floods the affected area of the map with water which causes the terrain to smooth back to its original state. The flood also damages the health of any of the troops in the flooded area.

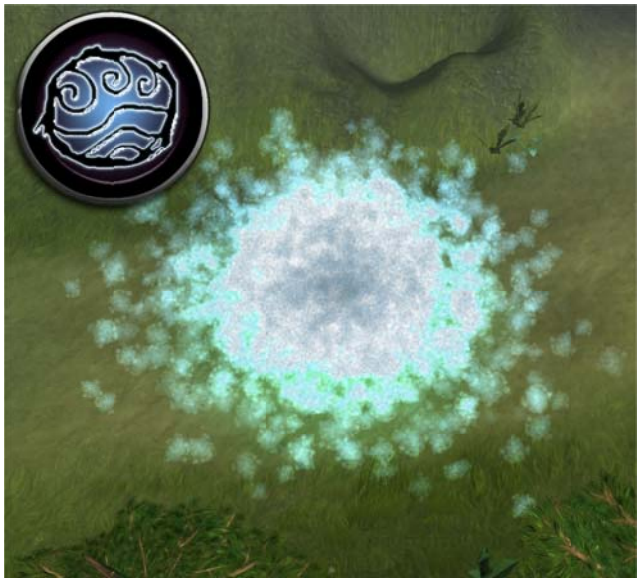


Figure 5: Flood Miracle

4. Heal Miracle

The heal miracle is the only miracle which has no effect on the terrain. As depicted in Figure 6, the heal miracle heals any friendly troops and structures within a certain range of where it is cast. The heal relic lies near the center of the map.

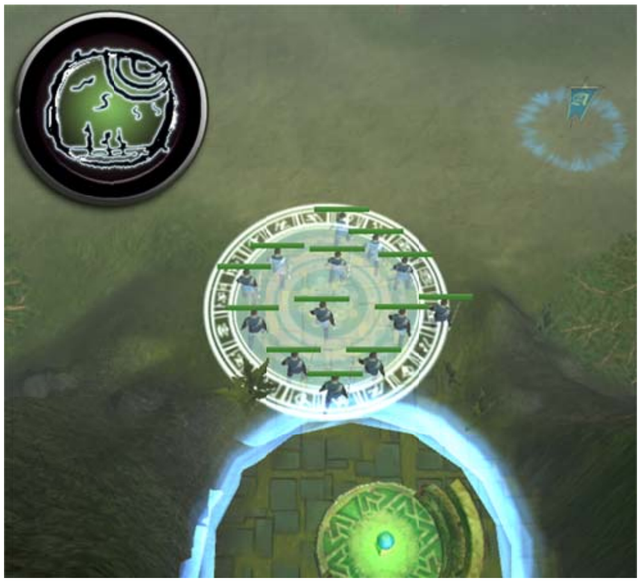


Figure 6: Heal Miracle

2.2.2 Resources

There are a total of four capturable resource points on the map. These points can be captured in the same manner in which the relics are captured. The resource points also have a similar control circle indication as the relics (See Figure 7). When a resource point is captured by either the player or the enemy, that resource point is converted into an extra barracks for that side. A resource point can be captured or lost by either side at any time during the game. Whoever controls the resource point during any given time, gains the advantage of having the extra barracks.

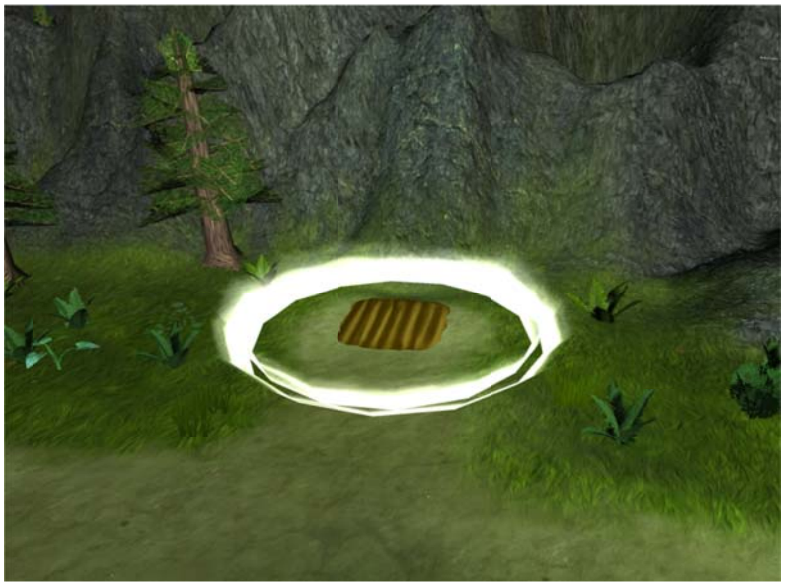


Figure 7: Resource node

2.2.3 Rally Flags

The player is given control over two rally flags during the course of the game (See Figure 8). The player can use these rally flags in order to direct allied troop movement. In placing a rally flag, the player is indicating to the allied troops who are closest to that particular flag that there is an objective there that they should seek. The two flags can be placed wherever the player sees fit. They can also be moved at any given point in time and the troops will react to a flag's movement when it has been placed back down.

Figure 8: Aetherian Rally Flag

2.2.4 Avatar

The Avatar acts as the main leader of the Aetherian forces. He is the strongest of the Aetherians, has the most health, and can deal the most damage to the enemy forces. He is also under the direct control of the player. The player can use the Avatar to fight the Gremlock troops, take out the enemy turrets, or capture resources and relics.

2.2.5 Friendly Units

The Aetherian infantry troops are directly influenced by the rally flags placed by the player. They will head towards whatever area the flag closest to them is placed. Once they are in the area of the flag they will defend that position and attack whatever or whoever is in the surrounding area. If the troops encounter enemies on the way to a rally flag, then they will engage in combat before continuing towards that objective. The Aetherian infantry units can also be used to capture resources and relics as well as destroy Gremlock structures.

2.2.6 Enemy Units

The Gremlock infantry are the enemy equivalent to the Aetherian allied units. The units serve to impede the progress of the Aetherian troops through offensive and defensive combat. The Gremlock units are controlled through a strategic AI hierarchy which directs them to capture resources and relics as well as to destroy the Aetherian structures. The Gremlock infantry also receive a health bonus for every relic that is under their control. The Gremlock infantry units spawn from the Gremlock barracks on the map.

2.2.7 The Gremlock Behemoth

The Gremlock Behemoth is the Gremlock equivalent to the Aetherian Avatar. The behemoth is stronger than the regular Gremlock infantry units. He can be utilized in a similar manor to the Avatar in that he is the most effective means of capturing the control points in the game. The behemoth is also controlled by the enemy strategic AI which delegates where the behemoth should move dependent on what is determined to be most important at a particular moment in time. Like the Avatar, the Behemoth also has a small delay between the time he is killed and the time he respawns on the battlefield. The behemoth spawns in front of the Gremlock headquarters before the AI informs him where to move.

2.2.8 Structures

There are different structures in the game which directly influence the tide of battle. Each side (the Player controlled Aetherians, and the AI controlled Gremlocks) have the same types and number of structures.

Headquarters (HQ)

The Aetherian and the Gremlock headquarters are the most important structures in the game, see Figure 18: Aetherian HQ and Figure 23: Gremlock HQ. Since the entire goal of the game is to destroy the enemy HQ before they can destroy yours, it is an essential aspect which influences gameplay. Although a player may use any number of strategies to accomplish this goal, the only thing which matters is if they manage to drain the health of the enemy HQ first. Each HQ is surrounded by a magical shield which protects it from harm. It is up to the player and the enemy Gremlocks to attack and destroy these shields in order to begin their attack on the main HQ base. The Gremlock HQ is also not susceptible to miracles so it must be destroyed by attacking with the Aetherian Avatar and the Aetherian infantry units.

Towers/Turrets

Each side has a total of six towers which guard the Aetherian and Gremlock sides of the map, see Figure 17: Aetherian Tower and Figure 22: Gremlock Tower. These towers or turrets serve as extra protection for each side's HQ. Each tower shoots fireballs at the opposing troops. Although it is not required to destroy the towers before attacking the enemy HQ, the towers make it very difficult for the Avatar or friendly units to make it to the HQ unharmed.

Barracks

Each side starts the game off with two barracks located near their headquarters and positioned at the start of the two available paths, see Figure 16: Aetherian Barracks and Figure 21: Gremlock Barracks. The two starting barracks will continue spawning units until you win or lose the game. However, the four extra available barracks at the resource nodes will only spawn units for the side which controls it at a given point in time. If a resource node is controlled by the Aetherians (indicated by a blue glowing control circle around the resource node) then it will spawn extra Aetherian infantry units. Inversely, if a resource node is controlled by the enemy Gremlocks (indicated by a red glowing control circle) then it will spawn extra Gremlock infantry units.

2.2.9 Minimap

Since the major gameplay of relic relies upon player strategy, we came up with the idea to include a minimap which would give them important information about the state of the game (see Figure 26: Minimap and Key). The minimap gives the player an overview of what the map looks like and relevant information about the different capture points. Similar to most MOBA and RTS games, the mini map provides information about where the enemy is attacking. Our minimap also includes the locations of the Avatar character and the enemy Gremlock Behemoth. At any moment, the player can glance at the map and attain vital details about the game in real-time. For example, a player may notice that the Gremlocks have captured the meteor relic (indicated by a red icon on the minimap). Then they can decide that they want to send troops to contest that relic. Since there are a lot of events and capture points to keep track of, the minimap provides an organized view of everything the player needs in order to plan ahead.

2.2.10 Technical Gameplay

The game is played as an RTS-style god game with an overhead point of view where the player takes control over the Aetherian troops and their god in human form, the Avatar, in order to destroy the evil enemy Gremlock forces. The player can direct the Aetherian armies with rally flags which they can strategically place to capture resources and relics all over the map. The player can also direct the movement of the Avatar with the mouse in order to assist their troops. Once the player has captured the relics on the battlefield, they gain special powers called miracles which they can cast help them gain the upper hand in the battle against the Gremlocks.

Each miracle has a cooldown time and a casting time to balance them. When a player chooses to start casting a miracle, a small timer icon appears to indicate how long before the miracle casts (See Figure 9). The player must hold-down the right mouse button until the timer finishes and the miracle casts. That miracle cannot be recast until a cooldown period passes as shown by a circle around the miracle icon (See Figure 10).

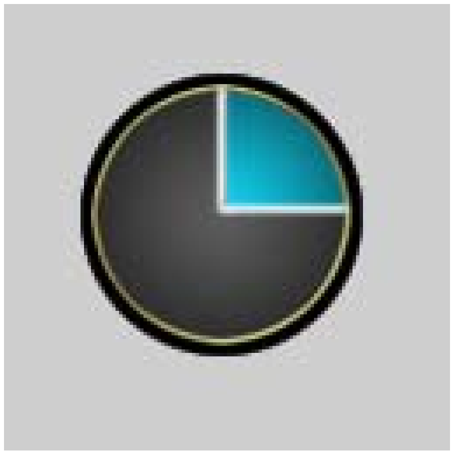


Figure 9: Miracle Casting Time Icon

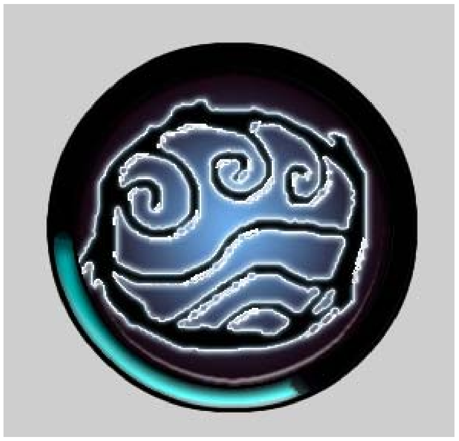


Figure 10: Miracle Cooldown Icon

2.3 Evaluation

Most of the gameplay of *Relic: TAU* focuses on the use of miracles and terrain modification. There is a lot of emphasis placed on using the miracles as a means of winning the game instead of just being about strength in numbers. Although the original idea for the game was to have dynamic combat between the two opposing sides, there was not enough time to work on polishing both miracles and combat. The effort which would have been spent on making the combat more smooth went into making the miracles more interesting and fun.

2.4 Strategies

Although there is only one way of winning the game, a player may take on different strategies to reach this main goal. The only winning condition of the game is that the enemy headquarters is destroyed. The only losing condition of the game is that your headquarters (the Aetherian HQ) is destroyed. It is up to the player to decide how to accomplish this goal. There are a few key strategies which the player may take in order to win the game. However, these strategies are not the only ones a player may take. They can be combined or utilized in different ways at different times to fit each person's unique playing style.

2.4.1 Resource Rush

The first type of strategy which the player may take is trying to collect all of the four resource nodes on the map first. In capturing all of the resource nodes available (circled in green in Figure 11) the player will maximize the number of friendly troops being generated. As the player amasses an army, it becomes easier for the player to storm the Gremlock stronghold and destroy their headquarters. By controlling all of the resources, the Gremlocks will have substantially less units to fight back with. Once all of the resources are captured, the player can more easily capture the four relics on the map to use to destroy the Gremlock towers and units defending the HQ.

Although this strategy is a viable way for the player to attempt to win the game, it is not without some drawbacks. By focusing all of your attention on capturing resources, you neglect capturing the relics. If the Gremlocks are able to capture all of the relics while the player is capturing resources, then the player will still face lethal opposition from the Gremlock forces which draw health and attack bonuses from each relic under their control. In order to avoid this situation, the player must carefully plan out troop movements so that the Gremlocks are unable to capture all of the relics.

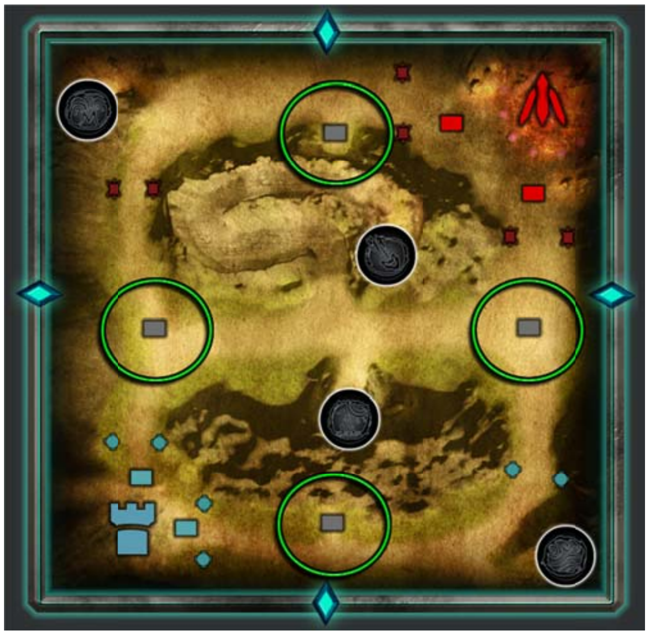


Figure 11: Resource node locations

Advantages:

- Maximizes units available

- Less Gremlock units will be generated than Aetherian units
- Easier to attack and overwhelm Gremlock units and structures (HQ and towers)

Disadvantages:

- Gremlocks may capture all of the relics
- Proper planning needed to keep Gremlocks from capturing relics

2.4.2 Relic Rush

The second strategy which the player may choose to take is to focus all of their attention on capturing the four relics on the map. After capturing each relic (circled in green in Figure 12), the player can utilize the miracle power associated with that relic to destroy the Gremlocks and capture other relics or resources. Unlike the Resource Rush strategy, the Relic Rush strategy is a little less straightforward, in that it is up to the player to decide which relic they want to try and capture first. Having control over one type of relic and miracle power versus another one has its own pros and cons.

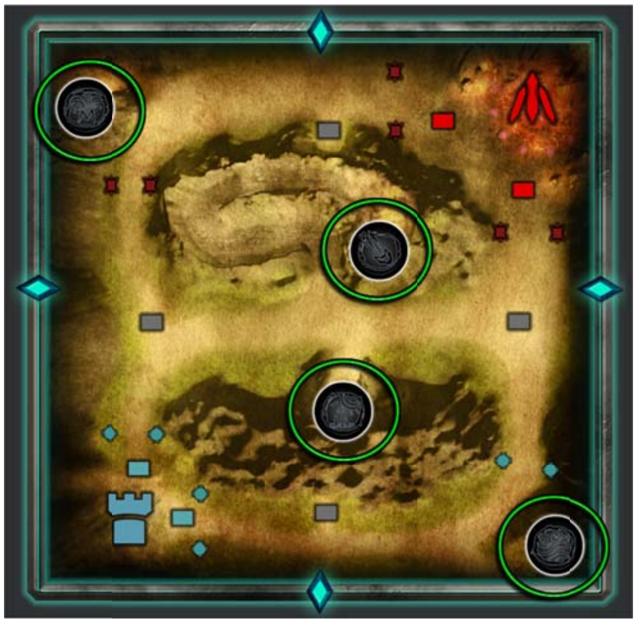


Figure 12: Relic artifact locations

Advantages:

- Miracle powers gained
- Gremlock units are not as strong
- Miracles can be used to capture other resource nodes or enemy towers

Disadvantages:

- Some relics may be more difficult to get than others
- Proper planning needed to decide which relics to capture first
- Gremlocks may capture all the resource nodes

2.4.3 Mixed Approach

The third strategy which the player may take is more of a combination between the first two types of strategies. With this strategy, the player may decide to capture either a particular relic or a particular resource node at any given moment during gameplay.

Figure 13 shows the mixed approach strategy based on priority. Relics and resources circled in green are of first priority for the player to capture, yellow of second priority, and red are of the player’s last priority.

Advantages:

- Control different areas of the map
- Have access to some of the miracles and some resources
- Flexibility to capture relics or resources in key choke points on the map

Disadvantages:

- Difficult to organize
- May become overwhelmed and end up with not enough resources or relics
- May not be prepared to defend against a Gremlock offensive movement

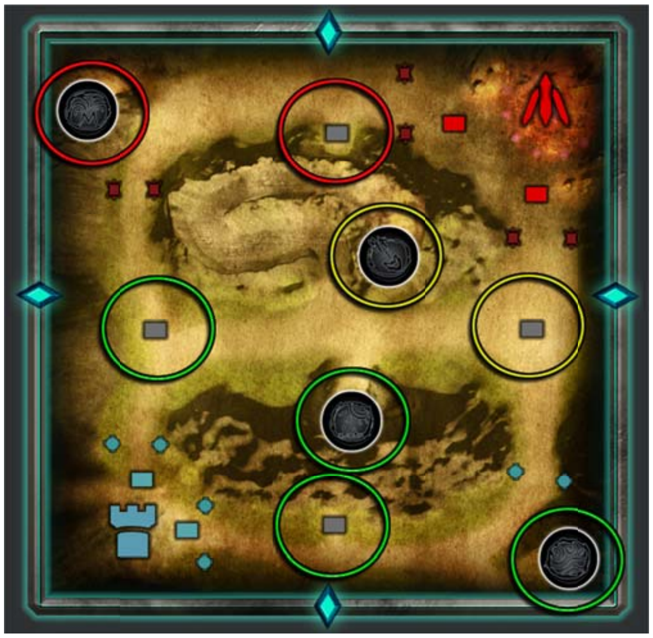


Figure 13: Mixed Approach Strategy

3. Art

Relic: TAU is a 3D medieval fantasy themed game. Since the game also has aspects of an RTS game in gameplay and style, it was important for us to create highly detailed yet low-poly game assets which could be used in-game. Since we knew that we would have a lot of objects on-screen at one time, we chose to appropriate our time accordingly. Those assets which were deemed to be the most important or would appear on screen the most were given the most attention. Smaller and less important objects were given the least amount of attention. In order to tackle the creation of the abundance of art needed for the game we decided to split the creation of assets between the Gremlocks and the Aetherians. Even though different artists would be working on the allies than the enemies, it was important that we adhered to a similar art style and maintained a sense of cohesion.

3.1 Styles and Influences

When creating the art style for *Relic: TAU* we looked at a few different games and established fantasy worlds as major influences for how we thought our game should look and fit in the context of our major gameplay features. A lot of our art style was heavily influenced by the Massive Online Battle Arena game *DOTA 2* and *League of Legends*. Both games include a painted art style with vibrant colors and effects. We wanted our color to be visually striking and believed these brighter colors would immediately attract attention. We also had some got some influence from the world of *Lord of the Rings*, and in particular, the RTS game the *Lord of the Rings: the Battle for Middle Earth 2.* Something we discovered to work well in all of these games was a distinct style for each different army or faction. In the case of our game, we wanted to apply this idea into creating a distinct style for both the Aetherians and the Gremlocks. Similar to many of these games, we also chose to proportion our buildings and structures smaller than how they would normally be in relation to game characters in order to place more emphasis on troop movement and strategic player interaction.

3.2 The Aetherians

In the game, you take control of the Aetherians, who are a fictional race of humans who are being oppressed by the evil Gremlock creatures. The Aetherians have their own distinct colors and style in the game. We wanted it to be easy for the player to identify a friendly area from a foe area very easily. The art style of the Aetherians is very bright and colorful but also somewhat old to reflect that their history and culture.

3.2.1 The Avatar

The Aetherian Avatar is a physical representation of their god in human form. The Avatar is a symbol of great power and magic. The style of the Avatar reflects his purpose in that he can be used as a sort of tank to power through the Gremlock defenses and take over resources and relics easily. The Avatar is physically larger than the other Aetherian units and he is also a completely unique character model (See Figure 14). The avatar also carries a giant sword which he will use in combat against the Gremlock forces.

The style of the avatar was also influenced by games such as *DOTA 2* and *League of Legends*. In those games, the player takes control over unique hero characters that have unique abilities that can used to affect gameplay. In creating the Aetherian Avatar, we drew inspiration from the different themes in our game including miracles when deciding upon how the Avatar should be portrayed. We wanted him to be instantly identifiable to the player and stand out among the other Aetherian troops. We scaled the Avatar so that he is physically taller than everyone else to help make him easier to spot.

There was a lot of time spent on creating the Avatar character model since we knew it would be a major part of the game. Even though the game is played from a birds eye perspective most of the time, we felt it was important to give the Avatar as much detail as possible in order for the player to be drawn to him.

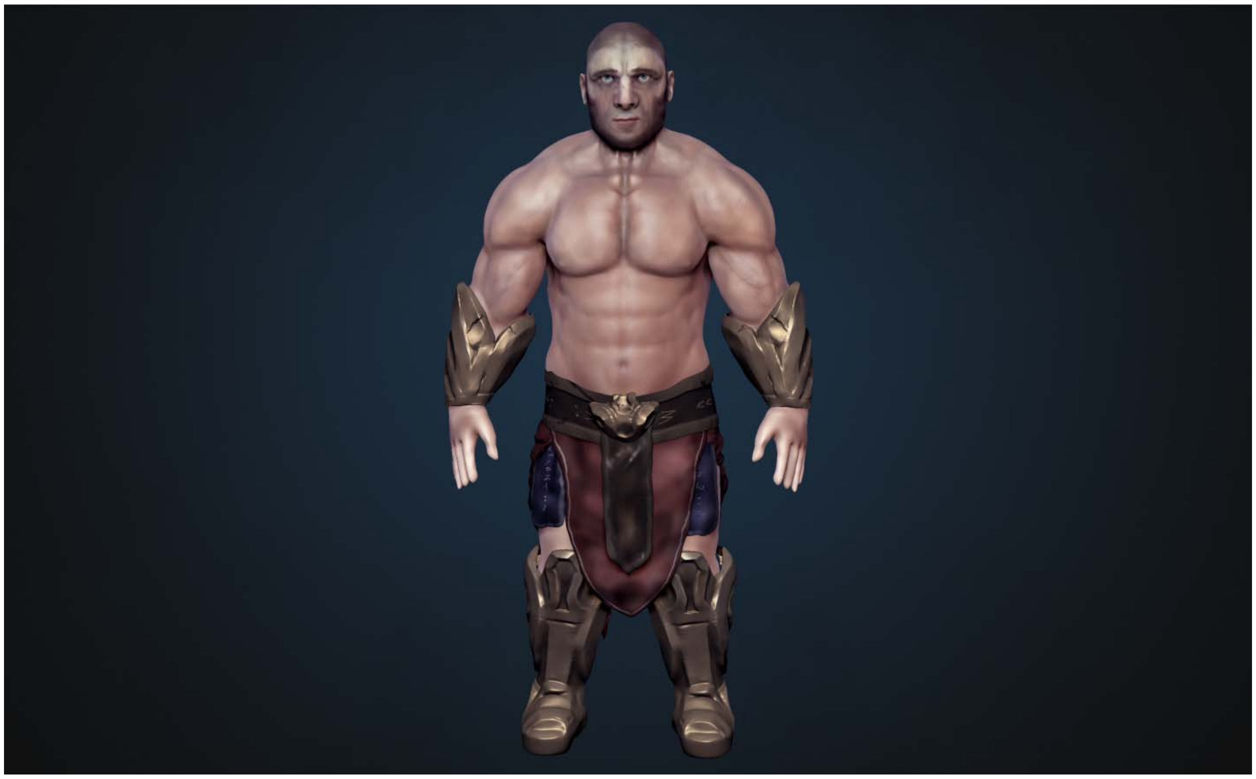


Figure 14: Render of Avatar character model

3.2.2 Aetherian Units

The Aetherian infantry units are much smaller than the Avatar and have a less detailed style. The units are mostly found in large groups in the game. The style of these units is also medieval but with light armor that has a blue color to fit with the

Aetherian color scheme (See Figure 15). Although these characters have a lot less detail, it was important that they looked good from far away.

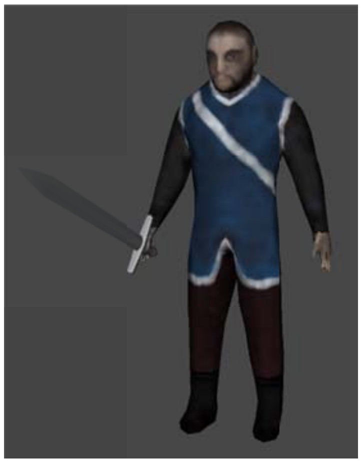


Figure 15: Aetherian unit

3.2.3 Aetherian Structures

The style of all of the Aetherian buildings in the game heavily reflects their purpose in the game world. Many of the structures are made up of stone or wood materials in order to adhere to the medieval theme but they also have some elements which give them a unique feel.

Aetherian Barracks

The Aetherian barracks have an old wooden style to fit with the medieval theme but are also a bit decorated to reflect the rich culture of the Aetherian peoples (See Figure 16). The style of the barracks also fits the overall naturalistic theme of the Aetherian side of the map. The barracks which are put together from wooden beams also reflects craftsmanship on the part of the Aetherians. The barracks are also stylistically small to place more emphasis on the characters and the environment.

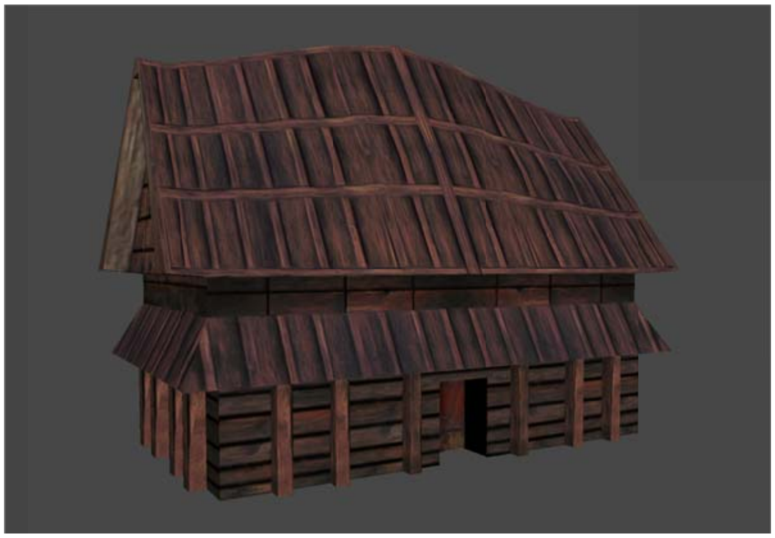


Figure 16: Aetherian Barracks

Aetherian Towers

The Aetherian towers are placed at the major choke points on the Aetherian side of the map. The stone towers are styled to fit with the theme of the Aetherian HQ. As shown in Figure 17, each tower has a burning blue fire located at the top which shoots magical balls of flame at any nearby enemy. The towers are much taller in size to the units in order to stand out and also to give them a vantage point from which they can shoot flames at enemy troops.

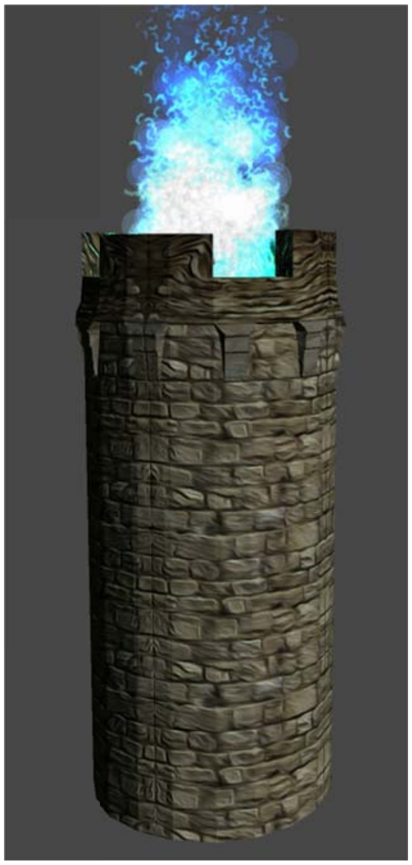


Figure 17: Aetherian Tower

Aetherian HQ

The Aetherian HQ has the form of an elaborate medieval stone castle (See Figure 18). We wanted the castle to stand out among the objects of the map to signify its importance. The overall themes of the Aetherian HQ are brightness naturalism. The castle is made of stone with some areas of blue roof tiling to fit with the Aetherian colors and theme. The HQ also has many different buildings within it which are surrounded by city walls to make it appear more like a large city where the Aetherians would live. Even though the HQ is modeled as a large city, we scaled the entire HQ down to fit into a small corner of the map similar to the way we scaled the other structures in the game. This fit with the style of the structures being much smaller in proportion to the troops.

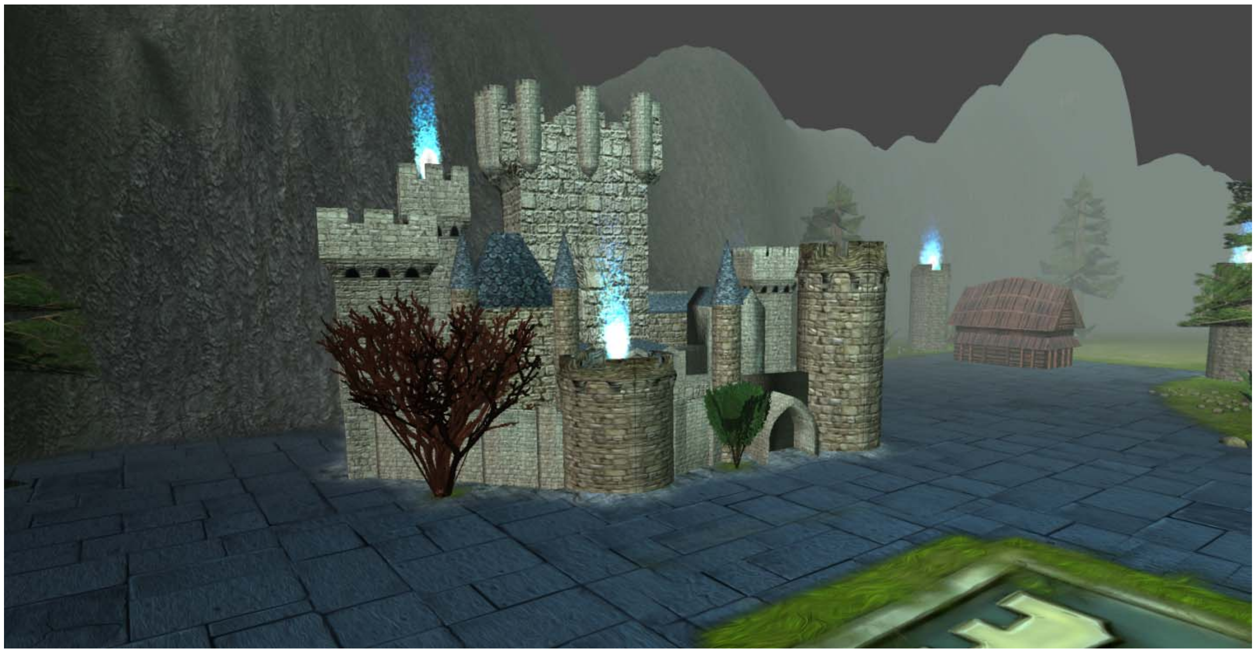


Figure 18: Aetherian HQ

3.3 The Gremlocks

The Gremlock are an evil, bellicose race of gremlin creatures whose goal in life is to expand their empire and destroy anyone standing in their way. Their color scheme contrasts against the Aetherians color scheme with darker colors and in the range of red, orange, and darker gray shades. The general look and feel of the Gremlock makes them seem grungy, evil, and threatening.

3.3.1 Gremlock Units

The Gremlock are clad in grey armor and wield gold swords as shown in Figure 19. Their style is high fantasy medieval. Their appearance was based heavily on kobolds (A race of sentient lizard people originating from Dungeons and Dragons) as well as being a chimera of many animal species including, but not limited to dragons, goats, and wolves. Their color scheme is dark shades of red and gray to accentuate their dark, brooding personalities.

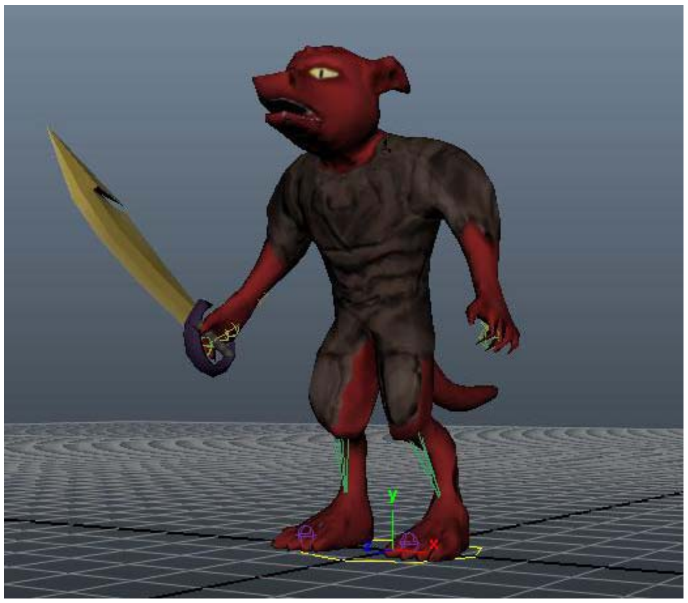


Figure 19: Gremlock Unit

3.3.2 The Behemoth

The Behemoth is a larger, more powerful version of the standard Gremlock unit clad in gold armor, has horns, and a shorter tail as shown in Figure 20. The behemoth is approximately twice as large as a standard unit and instead of attacking with a sword, attacks with claws. He has more health and does more damage to units and structures.

The Behemoth is not nearly as crucial to the gameplay as the Avatar as it acts strictly as a more powerful version of the standard unit, but instead serves as balance against the avatar. His appearance difference only serves to make him more threatening and to give the player an indicator that this guy is not only different, but potentially more powerful.

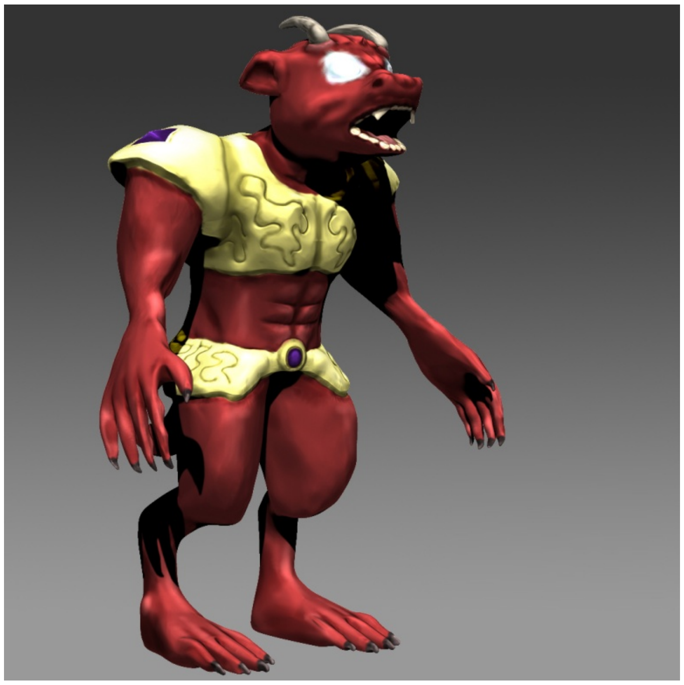


Figure 20: Gremlock Behemoth

3.3.3 Gremlock Structures

Gremlock structures are constructed to look like vastly alternative designs to the Aetherian designs. They are all medieval fantasy with a red, grungy tone, an almost volcanic shade of brick. Some buildings make use of crystals or organic matter in their design.

Gremlock Barracks

The Gremlock barracks is based off of a Middle Eastern Temple design that was modified to look more medieval which can be seen in (See Figure 21). The building has 90º symmetry, with stairs on all sides leading to doors. The roof is a gold colored, bell-shaped dome. These temples serve to appear less as a barracks, and more as a center for training grounds.

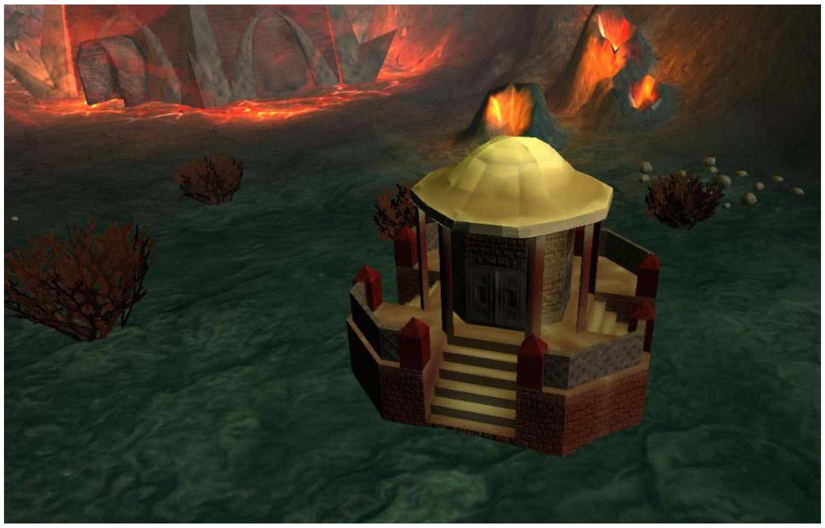


Figure 21: Gremlock Barracks

Gremlock Towers

The Gremlock tower is based on a shorter, wider version of a castle turret with a supportive base shown in Figure 22. The crystal on the top serves as a source for the fire to come from. The tower has 90º symmetry much like the barracks. On a basic level, the tower is made of a 16-sided cylinder and every other face pulls out from the base to add to the support structure of the tower.

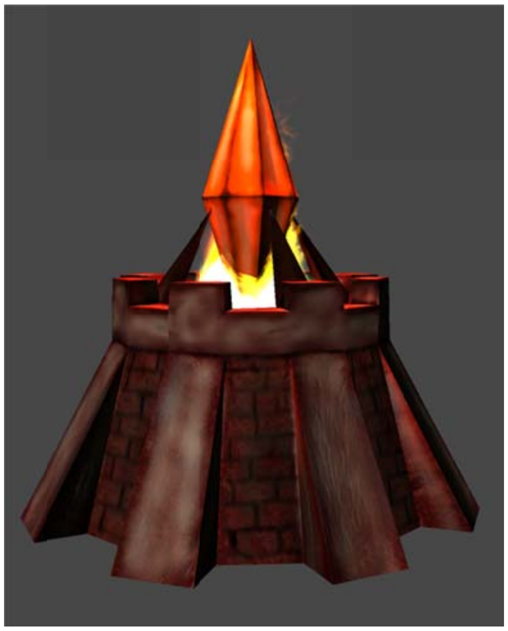


Figure 22: Gremlock Tower

Gremlock HQ

The Gremlock Headquarters, shown in Figure 23, is based on a hexagonal shaped stronghold with a large tower in the center. It has 60º symmetry with the exception of an entrance tunnel at the front of the model. Jagged shapes all around on the building serve to make it appear more threatening and evil. The HQ is constructed using the same volcanic bricks the other buildings use, but also has large lighter colored spikes and arches, almost bone or tooth colored. We wanted to give it an intimidating appearance which is where the single tall spire of a tower originated from.

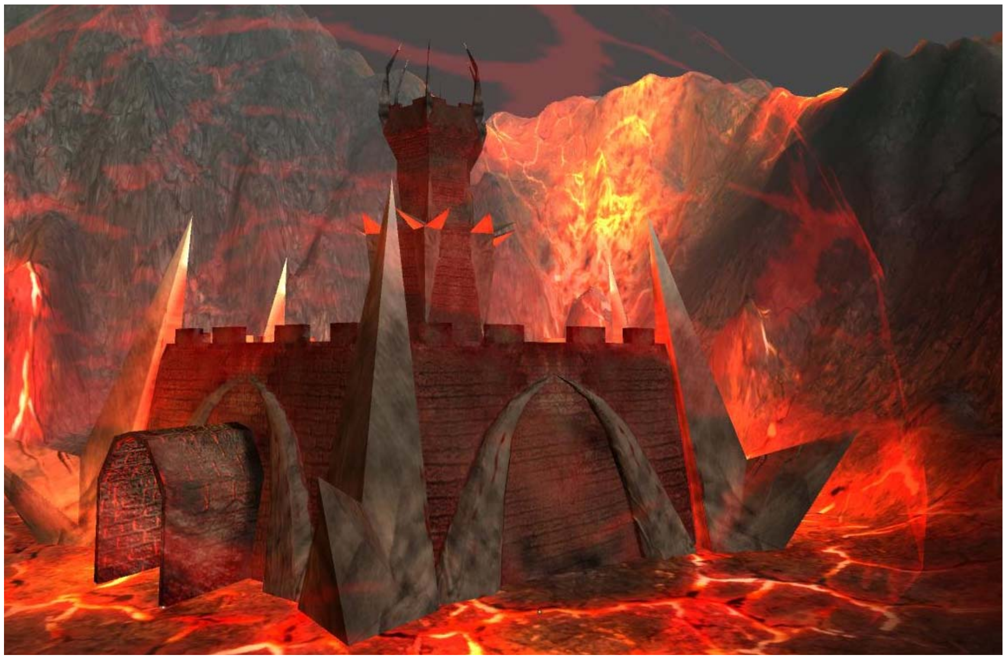


Figure 23: Gremlock HQ

3.4 Textures

World textures were based on real textures that were made painterly using drawing techniques and PhotoShop filters. A lot of the textures were based on real world images of dirt, bricks, and other organic and inorganic surfaces that were adapted to fit the context of the model they were applied. Others, such as the grass texture shown in Figure 24, were drawn from scratch using digital painting techniques to help convey a sense of depth. The textures that were applied to the world terrain also made use of bumpmaps. Bumpmaps are a special texture that tells the computer to change how the light bounces off of a surface, and by doing this, makes the surface appear three dimensional when in fact it is completely flat. Some models also made use of Bumpmaps to make them appear as though they consist of more polygons than they actually had.

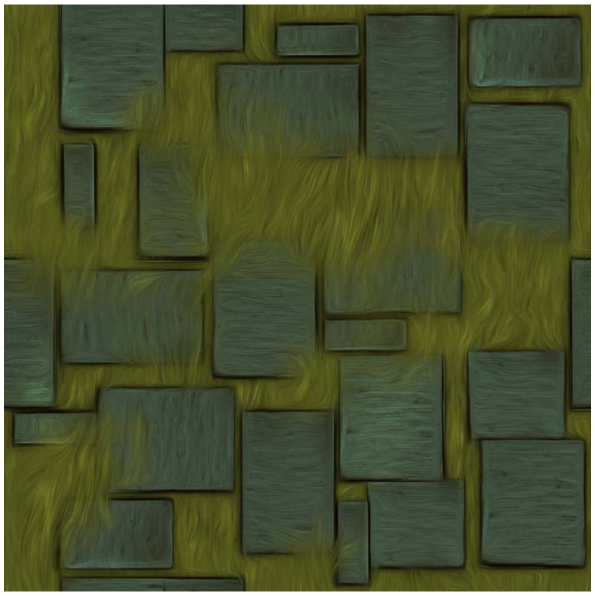


Figure 24: Texture example

3.5 Terrain and Environment Models

Terrain was built using the standard terrain tool in Unity. The terrain uses a modified system allowing for the applied textures to use bump maps. There are a variety of environment models that are distributed depending upon their location. All three ferns reuse the same low-poly model.

3.5.1 Relics

Relics are modeled to look like mystical shrines decorated with runes. The meteor relic shrine can be seen in Figure 25. Each one uses differently colored particle effects to differentiate which one gives which power. Their textures are also color coded depending upon the miracle they are tied. They are designed to look like a Gaelic shrine consisting of columns and other stone structures. The design on the floor has 30^{o} rotational symmetry. The shrine is made to look partially ruined.

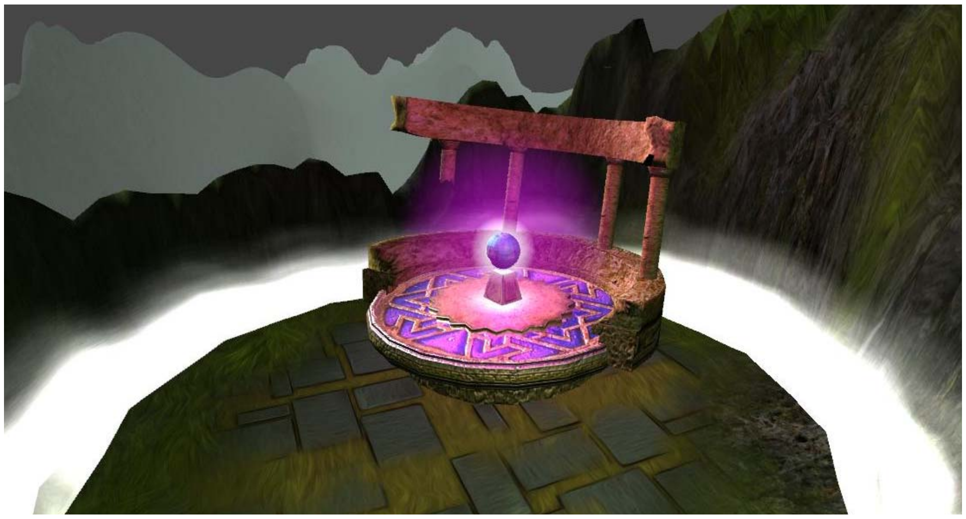


Figure 25: Relic Shrine

3.6 Level/Map Design

The level design is based on one of the eight original level designs that were proposed back in A term. The initial design started with sculpting the terrain to be believable yet still within the realm of fantasy and serve to be a physical barrier against unit movement.

3.7 Lighting and FX

In addition to the placement of the directional lights, point lights were added to make structures stand out more. The initial directional light that was used did not give enough lighting to the map, everything seemed dull with minimal depth. This light projected a light down at a 25^o angle on the map angled projecting northwest. We used point lights on structures and objects, such as the flames on the tops of turrets, in order to add vibrancy to objects.

3.7.1 Particles

Particles were used for miracles, and to give a boost to light emitters. The towers on either side use a color coded particle to give a source for turret fire to originate from. These particles look like differently colored flames that billow out from the top of the tower. The meteor, volcano, and flood miracle make use of particles as well. The meteor as it falls leaves a yellow and white trail as it falls and creates the illusion that the meteor emits light as it falls. The volcano spouts out fiery particles; the flood miracle behaves similarly, but instead uses watery looking particles. The particle systems were developed natively within Unity using the built-in particle engine.

3.8 Animation

Units were given basic animations which would allow them to look like they were walking and attacking. The resource points are also animated and those animations play when a resource point changes hands. All units were animated using Maya and 3DS Max. First when models were created, they required a skeleton in or rig in order to be animated. This rig was built by placing in bones that would correspond to certain parts on the body (such as lower arm, upper arm, upper body, torso, etc.). Once the rig is built, it bound to the model, or skin, and is properly weighted so that specific polygons on the model will move with its corresponding bone properly. Once this is complete, then the model is ready for animating. All the character models initially were given four animation sequences: idle, walk, attack, death. The death animation was not implemented.

The resource point has three states: neutral, Gremlock controlled, and Aetherian controlled. The model has animations to transition between all three of these states (6 animations in total). Because the resource point consists of basic primitive transitions, rigging was not necessary.

3.9 User Interface and Other 2D Art

The graphical user interface is an important part of how our game is played. Like many other RTS style games, the UI was designed with gameplay in mind. We wanted to create a simplistic UI which would help the player play the game and not make it more difficult by confusing them. Our UI places a large emphasis on highlighting the four miracle powers since they are the focal point of the game.

3.9.1 Minimap

The minimap is a scaled representation of the game map located in the upper right hand corner of the screen which gives the player key information about the location of units, the avatar, and the state of the game. Other games such as *DOTA 2* and *League of Legends* influenced the look and function of our game's minimap. The minimap is styled to fit into the game world and reflect the medieval and fantasy parts of the game. Part of the challenge of creating a UI representation of our game map was to have it seem like it belonged in the game space. There was a lot of time spent creating and tweaking different parts of the minimap in order to make it work both functionally and stylistically in the game.

The shape of the minimap is square in order to accommodate the space of the square game level. It is adorned with a dark stone colored border as well as blue magic crystals which give it both artistic flare and continuity with our game's style. The size of the map is a bit larger than standard minimaps found in other games in order to give the player a better view of what is happening on the map. The minimap is a key aspect of how the player receives feedback about the state of play.

As shown in Figure 26, each different structure and type of troop has its own unique minimap icon in order to indicate position on the map. The icons for the Aetherian units and structures are blue in color while the Gremlock icons are red. The Gremlock HQ is represented by a large red spire icon while the Aetherian HQ is represented by a large blue castle. Not only does the minimap contain information relevant to where everything and everyone is on the map, it also shows who is in control over the various capturable points

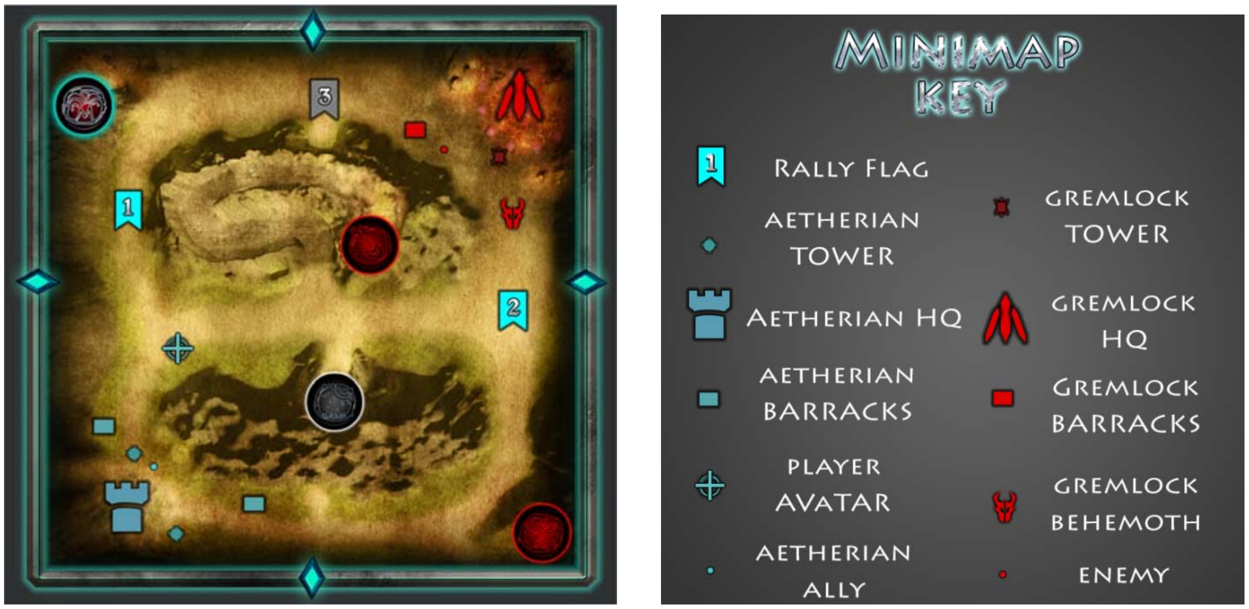


Figure 26: Minimap and Key

Relic icons indicate the location and status of each of the four relics on the map. If a relic icon is colored red, it is controlled by the Gremlocks. If a relic icon is colored blue, it is controlled by the Aetherians. Alternatively, if no one controls a relic, then the icon shows a dull grey color. Similar to the relic icons, icons for the capturable resource points will show a red barracks indicating Gremlock control or blue to indicate Aetherian control. By adding this dynamic color changing system, a player can quickly glance at the map and gain vital information about what is going on. Instead of having multiple menus or submenus for the player to have to look through, most of what the player will want to know is displayed within the easy-to-decipher minimap.

The minimap asset was created in Adobe PhotoShop and then saved as different layers which could then be assembled in Unity. Unlike some of the other UI assets, we did not use NGUI to draw the minimap during the game. The built in Unity interface system was utilized for drawing the dynamic and moving icons onto the map. Each icon was handmade separately in PhotoShop and imported into the game engine. The different icons were then attached to various scripts which tell the game which icons to show and in what location.

3.9.2 Miracle Selection

The miracle selection bar is located at the very bottom of the screen and stretches its entire length. The background for this part of the UI has a stone texture and is shaped like the edges of a battle-damaged castle wall. The castle has four spots for four icons that coincide with the four different miracles and relics in the game. The main purpose for the miracle selection bar is for the player to select an available miracle to cast and to indicate the cooldown time for each particular miracle. In addition to showing which miracle is active, the selection bar also shows the status of each relic and when the player is losing control over it.

Each miracle icon was designed and created in PhotoShop then imported into Unity where it is implemented using NGUI. All of the icons have the same overall tribal theme; however, each icon has their own logo which symbolizes the relic and miracle it represents. The flood miracle is represented by a tribal-style wave symbol and is colored blue. The Volcano miracle is represented by an erupting tribal-style volcano and is colored red. The Meteor miracle is shown as purple-colored falling meteor symbol. The Heal miracle is shown as a healing hand green-colored symbol.

Similar to the minimap, the miracle icons on the selection bar change color to indicate control of each relic. An uncontrolled relic or a relic which is being controlled by the Gremlocks will show up as a grey icon for that coordinating miracle on the miracle selection bar. The miracle selection bar also tracks the progress of a relic being captured in real-time. As a relic is being captured by the player, the corresponding miracle icon will begin to fill up with color until it is captured and then is shown in full color. As a relic is being lost, the corresponding miracle icon will begin to drain its color until it is fully lost and completely grey.

Along with a colored symbol of each miracle, each icon also has its own circular blue magic meter which indicates the cooldown time for each miracle (See Figure 27). Every miracle has its own specific cooldown time which is reflected through the cooldown meters filling up at different time intervals. When the miracle cooldown meter shows full, that miracle is ready to be cast. After selecting a miracle to cast, the player must hold the miracle cast button (right mouse click) for a specified period of time in order to cast that miracle.

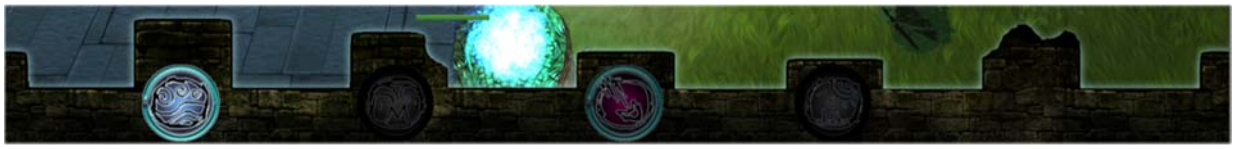


Figure 27: Miracle selection bar

The cooldown and also casting time varies between the four different miracles. The Volcano miracle has the longest cooldown and casting times while the Heal and Flood miracles have the fastest cooldown times and are faster to cast. As the player holds down the miracle cast button, a timer will appear on screen to indicate the time needed to cast that miracle. When the timer fills up with a blue color, the selected miracle will be cast, or placed, in the area of the map which the player has their mouse cursor over.

3.9.3 Avatar Icon

The Avatar icon is located at the top left corner of the map and acts as a stylized representation of the player-avatar as well as gives the player information about the Avatar's current health. The style of the icon is of an elaborate circular shield with an metal studded orange-wood texture (See Figure 28). A picture of the Avatar is displayed at the center of the shield with a circular green-glowing health bar around the outside of the shield. When the Avatar is hurt in the game, the green health bar drains incrementally and turns to a red color when the Avatar's health is in immediate danger of being depleted.

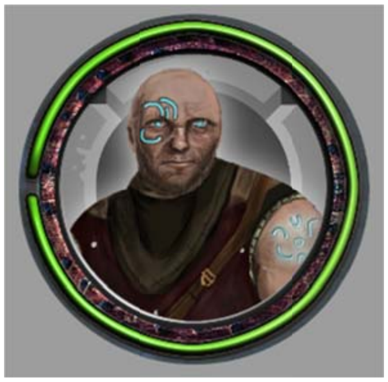


Figure 28: Avatar icon

4. Engineering

When building Relic we needed to resolve several technical obstacles. First, we needed to work out a way to organize the code which would keep it understandable for the months that we worked on the project. Second, we needed to find a way to actually manipulate the terrain in a believable way. Third, we needed to work out systems for how the AI would move around the world when the terrain was changing. It took a lot of work to get through all of these challenges.

4.1 Code Organization

The majority of the code was split into two sections: Controllers and Models. At the beginning of the project we decided that each object would have its own model containing all the fields that defined that object. Then, there would be a set of controllers that acted on the object's model. However each controller would not contain any. This followed the model-view-controller pattern, where the Unity Engine acted as the view.

4.2 Terrain Manipulation

One of the most important features in our game is the ability for the player to dynamically change the terrain during gameplay. Given the way in which Unity handles the terrain, the implementation of this feature is conceptually simple but it also introduces its own sets of problems. The two main characteristics of the terrain that change dynamically during gameplay are the heights of the terrain and the textures on the terrain. Changing the heights of the terrain has a great impact on the pathfinding algorithm (see section 4.5 Pathfinding).

4.2.1 Heightmap Integration

The way in which Unity creates a terrain's topology is fairly simple: it uses a heightmap. The terrain is basically a plane with x and y coordinates and a height value at each position. In order to alter the topology of the terrain we only need to pass in a new value to any given position on the terrain. Unity terrains also allows to pass in a two dimensional array of values to change the heights of a section of the terrain.

Our motivation for changing the terrain in the game was to allow the user to create natural deformations such as raising a volcano or leaving behind a crater after summoning a meteor. Crude approximations of these features could be achieved by using mathematical functions in the script, but these features look too symmetric and unnatural and do not look as pleasant as models created by artists. The challenge then was in figuring out a way of allowing the artists to model changes in the terrain.

Our solution was to use a grayscale image generated by the artists (See Figure 29 b). We then converted that image into RAW format so that Unity was able to automatically generate a new terrain object from that image (See Figure 29 a). We could then save that object into a prefab. Any time that we needed to create a new volcano or crater on the terrain we just needed to access the height information of the terrain on the prefab and pass those values to the desired location on the game's active terrain.

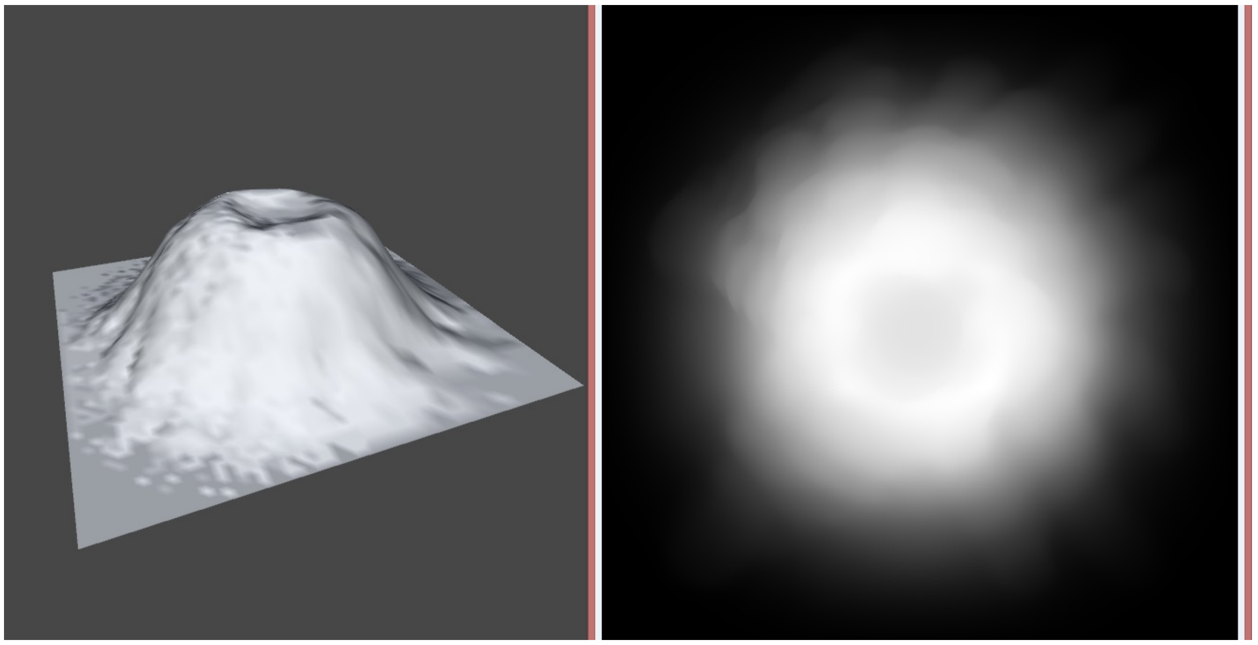


Figure 29: (a) Terrain generated from grayscale image (b) greyscale image of terrain

4.2.2 Changing Textures

The way in which Unity allows scripts to access the texture information on the terrain is somewhat similar to the heightmap information. Each texture on the terrain looks like a plane with x and y coordinates and an alpha value at each position. In order to alter the textures on the terrain a three dimensional array must be created. The first index corresponds to each texture's number while the second and third index corresponds to a position on the terrain. The array must always be the same size, so even when you would just like to change a single value you also have to specify values for the rest of the textures and positions (even if it is just re-assigning the old values). This way of handling texture changes quickly cause performance issues as the size of the terrain increases or the number of textures on the terrain increases.

4.3 AI

The AI was broken into two sections: the strategic AI and the follower AI. The Strategic AI was in charge of computing problems which covered the whole team such as objectives and tracking troop lists. The Follower AI was in charge of moving the individual units around the map and any calculations that need to be made per unit.

4.3.1 Strategist

Each side in the game has a strategic AI which covers any intelligence that needs to happen across the entire team. This boils down to just figuring out priority objectives and keeping a list of all the units in the game.

Objectives

The main purpose of the strategic AI is to pick objectives for the units and assign them. To do this, during each frame tick the AI will check that each objective still exists and is not under that side's control. The Aetherian and Gremlock AI assign objectives slightly differently. The player places the Aetherian's objectives and the Gremlock Strategic AI has to pick targets for their objectives. Once the objectives are decided, both AIs will assign objectives to units which are currently set to idle.

Each strategic AI has two objectives which they can assign their units to. For the Aetherians, the player controls where these two objectives are located by moving two blue objective flags around the map (See Figure 30). The Aetherian strategic AI assigns a new unit to the closer of the two flags when the unit spawns. Similarly, the Gremlock strategic AI must pick locations for the two Gremlock objectives. To do this, it picks the two closest *key locations* to the Gremlock HQ. Key locations are enemy buildings, neutral/enemy relics, or neutral/enemy resource nodes. Then it assigns units to these locations just like the Aetherian AI.

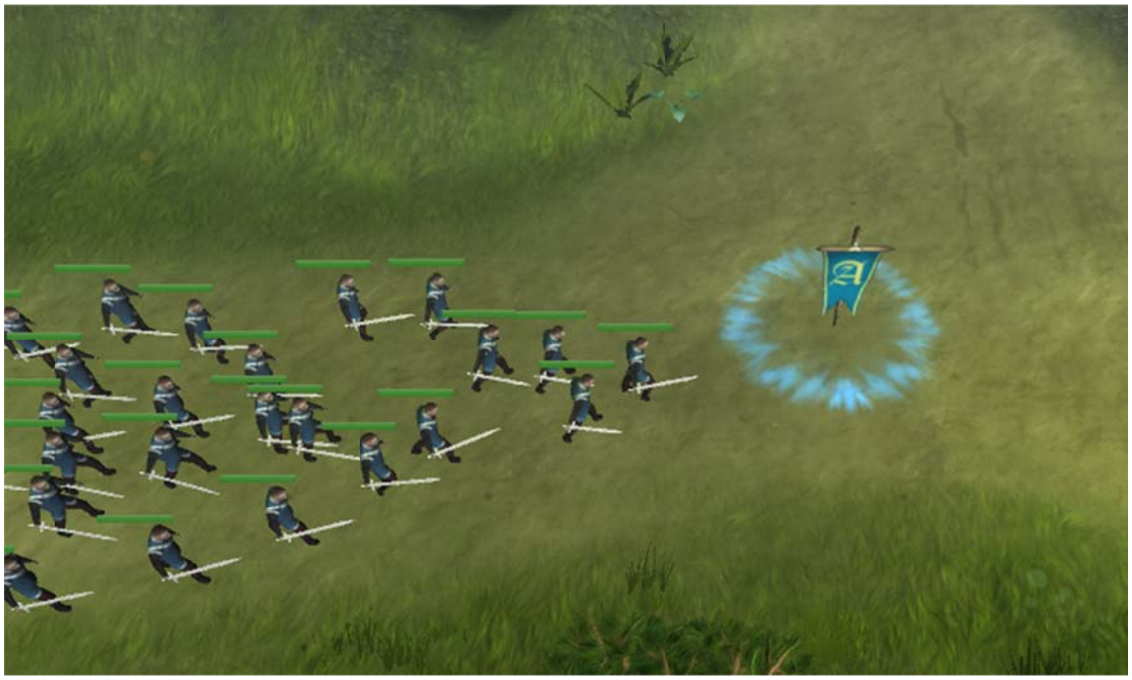


Figure 30: Aetherian Units Moving To An Objective

Troop List

The second purpose of the strategic AI is to maintain a list of all the allied troops and buildings for its side. This gives the units a short list to look through for information such as separation forces and units to attack.

4.4 State Machines

The one of the core decision making systems for the AI are simple state machines which are individually created for each agent. These state machines are used to control the follower agents and the leader agents.

4.4.1 System

The state machines system consists of four core classes: StateMachine.cs, SM_State.cs, SM_Action.cs, and SM_Transition.cs. Every component of the state machine inherits from one of these classes. StateMachine.cs is the top level class which checks for state transitions and provides the given action each time the state machine is updated. SM_State.cs is a class to hold the actions and transitions associated with that state. SM_Action.cs is a class to calculate the steering force required to move the agent. SM_Transition.cs is a class which determines when the state machine should change states and which state to change to.

4.4.2 Leaders

Each group of units has an invisible leader which follows a three-state state machine (See Figure 31). The main state, S_FollowPath, has the leader compute a path on the grid to the current objective and follow that path using a straightforward path following algorithm. The other two states are both S_Wait states where the leader stands still until the state changes. One of these will trigger if the player is dragging the objective and the other will trigger if the leader's units are over a given distance away.

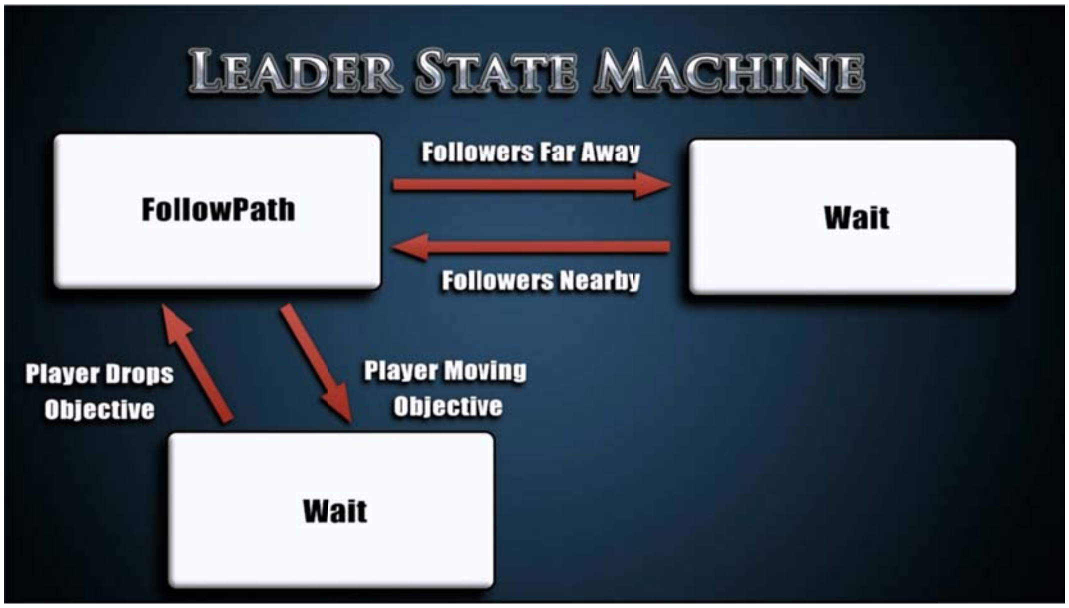


Figure 31: Leader State Machine

4.4.3 Units

The unit's actions are determined by a small state machine containing two states: S_FollowLeader and S_AttackEnemy (See Figure 32). S_FollowLeader has the unit approach the squad leaders directly. If an enemy comes close to the unit, then it switches to S_AttackEnemy where the unit approaches the nearest enemy.

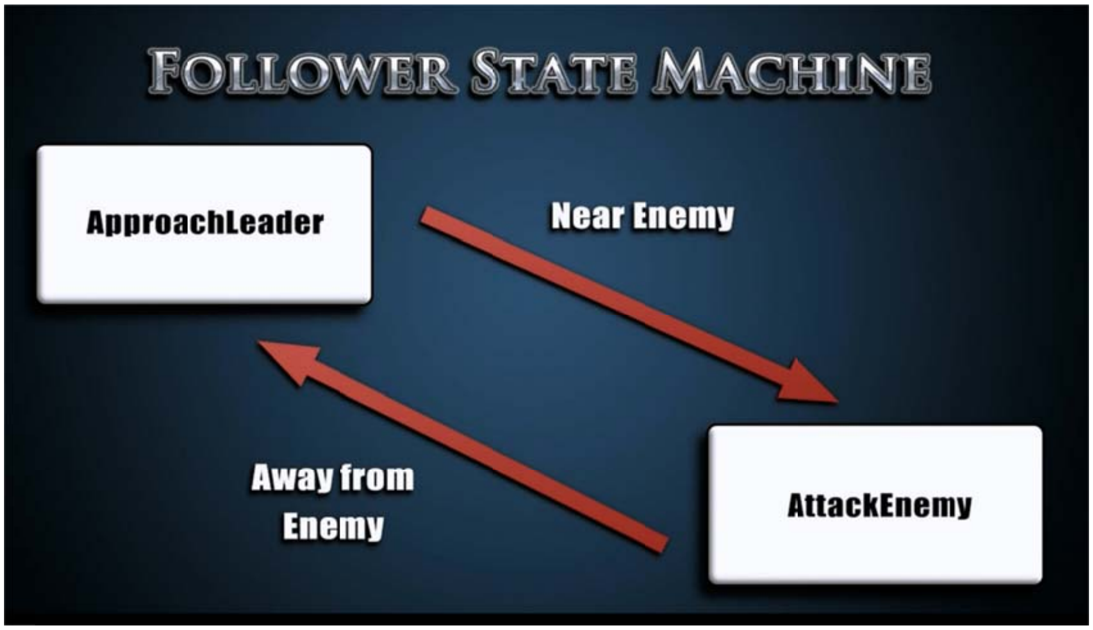


Figure 32: Follower State Machine

4.5 Pathfinding

Pathfinding turned out to be one of the most difficult technical challenges of the project. Since the terrain was changing over the course of the game, we could not set up a waypoint grid ahead of time. Also, using physics-based movement on each unit ended up being too costly. So, we needed to be more creative with our solution.

4.5.1 A* Grid

At the start of the game, a grid of waypoints is dynamically created based on the current state of the map (See Figure 33). Two neighboring nodes are connected together if the height difference between those nodes is small enough. The grid updates periodically to check for changes in the terrain. This grid is used by the leaders to path between two points on the map during the S_FollowPath state. When a leader needs to find a new path, it finds the closest node to itself and the end point then runs a basic A* algorithm to find a path between those two nodes.

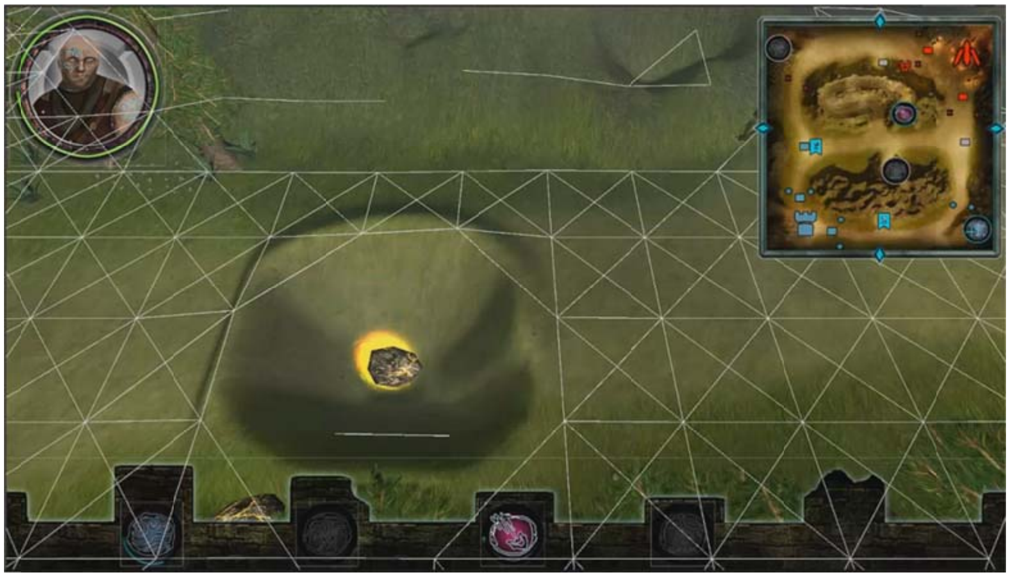


Figure 33: Waypoint Grid Overlaying Terrain

4.5.2 Physics Steering vs. Positional Movement

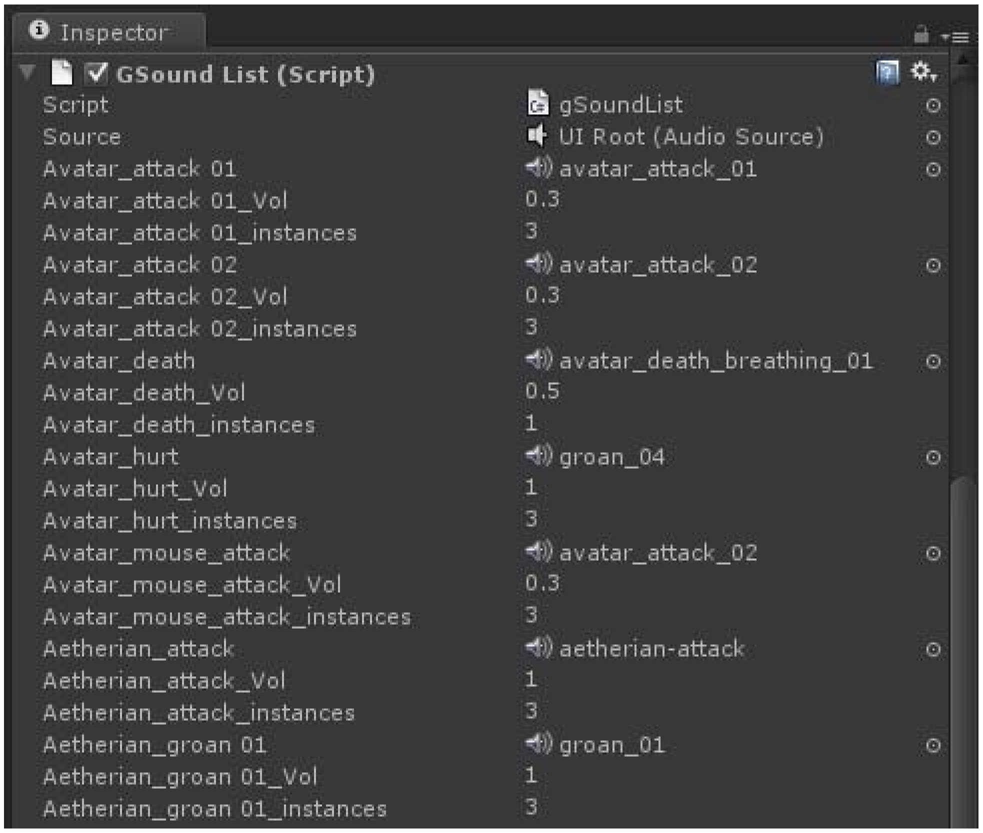
Over the course of the project, we attempted to determine whether to use physics-based movement or positional movement for the units. With positional movement, the unit's position would be changed each time step according to steering forces and all the units would be tied to the ground via a script. With physics-based movement, the units would experience gravity and move around the map via steering forces. Both set ups caused their own sets of problems for us. The positional movement wouldn't prevent the units from walking over the mountains on the map which was quite unrealistic and made all terrain modification pointless. The physics-based movement had difficulty dealing with all the tiny hills and corners on the map and units would constantly get stuck and shake around which looked terrible. We ended up using positional movement for the units and using the A* grid to keep them from walking over the mountains.

4.6 Audio System

In order to produce a more immersive game experience through the use of sounds we made sure to include certain features in the implementation of our audio system. These features were included in order to enhance the pipeline for sound integration, place restrictions on how the sounds are played and provide control over how the sounds are played back.

4.6.1 Sound List

The sound list is used to have a centralized location where all the sounds are collected. It helps with sound integration: any sound that is copied into the project folder can be assigned to any of the available fields, as shown in Figure 34. Each field is accessed by another object’s script in order to acquire the desired sound. This means that anyone who wants to change which sound is played at a particular time only has to assign a new sound to the given field through the Unity editor. In addition, the sound list gives control over certain characteristics of each sound: the volume and the number of instance. The values assigned to these variables are used by the Sound Manager when playing back the sounds.



GSound List (Script)

Script	gSoundList
Source	UI Root (Audio Source)
Avatar_attack 01	avatar_attack_01
Avatar_attack 01_Vol	0.3
Avatar_attack 01_instances	3
Avatar_attack 02	avatar_attack_02
Avatar_attack 02_Vol	0.3
Avatar_attack 02_instances	3
Avatar_death	avatar_death_breathing_01
Avatar_death_Vol	0.5
Avatar_death_instances	1
Avatar_hurt	groan_04
Avatar_hurt_Vol	1
Avatar_hurt_instances	3
Avatar_mouse_attack	avatar_attack_02
Avatar_mouse_attack_Vol	0.3
Avatar_mouse_attack_instances	3
Aetherian_attack	aetherian-attack
Aetherian_attack_Vol	1
Aetherian_attack_instances	3
Aetherian_groan 01	groan_01
Aetherian_groan 01_Vol	1
Aetherian_groan 01_instances	3

Figure 34: Sound list in editor

4.6.2 Sound Manager

The Sound Manager controls which sounds are played when. It was developed out of a need to restrict the number of instances of a particular sound that are played at any one instant. For example, when more than a couple of units are engaged in combat and are all attacking at the same time, the number of sword clashes and grunts quickly becomes overwhelming. This is why any object that wants to play a sound through the Sound Manager has to register the sound first. Once the sound has been registered with the Sound Manager, the object can then ask the manager to play the sound on its behalf. As the sequence diagram of (See Figure 35) shows, the Sound Manager only plays the sound if the maximum number of instance has not already been reached for that particular sound.

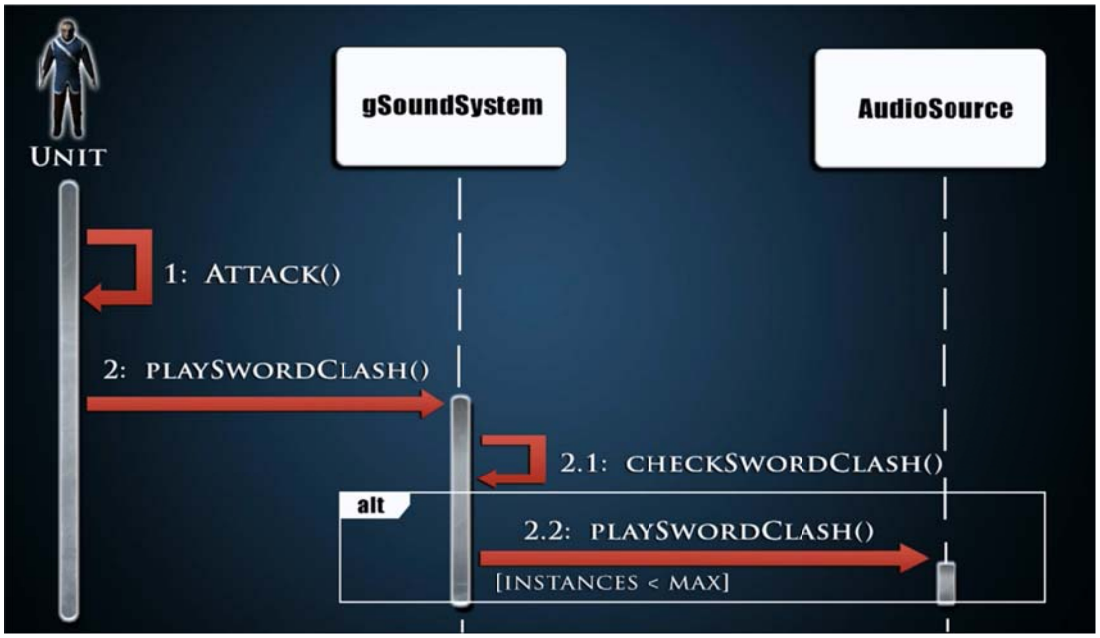


Figure 35: Audio Engine Sequence Diagram

4.6.3 Crowd Sounds

The crowd sounds work as follows: Whenever the camera looks at an area on the map where there is a group of units nearby, a crowd sound is played. The number of units determines how high the volume is when the crowd sound is played. A few units means that the crowd sound volume will be low (See Figure 36 b), and many units means the crowd sound volume will be high (See Figure 36 a). There are a maximum number of units after which the maximum volume setting is used for playing the crowd sound.

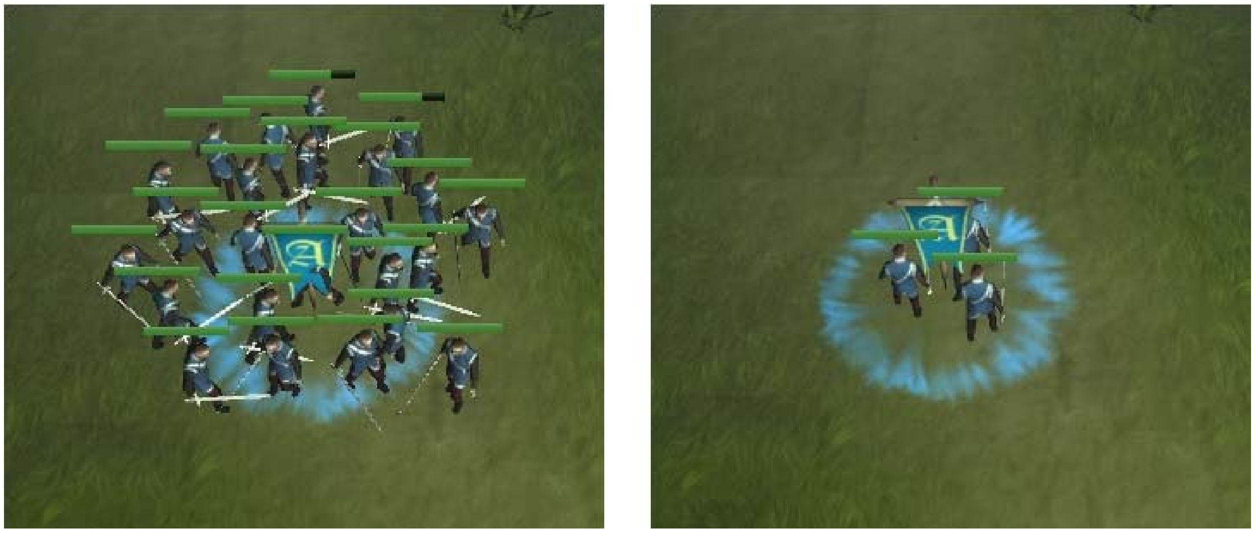
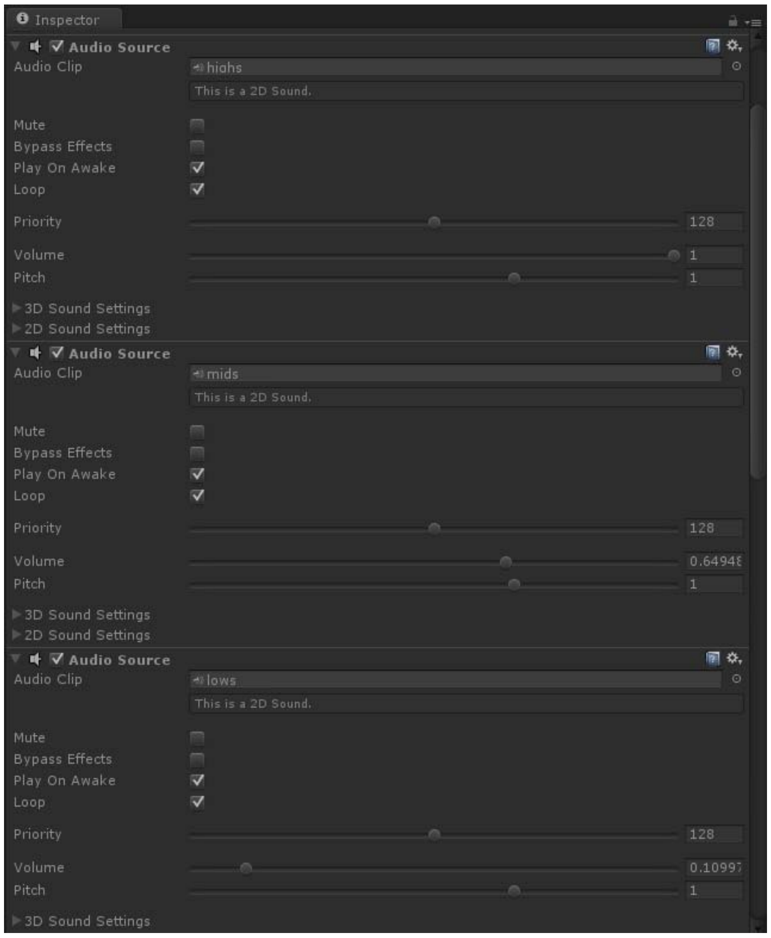


Figure 36: (a) Large crowd (b) small crowd

4.7 Dynamic Music - Three Stem System

One of our goals when we set out to develop the game was to create a more immersive experience with the background music. To accomplish this we implemented a system that dynamically changes music during gameplay. This system takes advantage of three different stems of audio that form part of the same musical piece.

To implement the three stem system we used three different audio sources (See Figure 37). Each source contains a different track or stem of the musical composition, and they are all played simultaneously. As the game changes the volume of each audio source is changed, varying from completely silent to maximum volume. The criteria used for altering the volume values were the proximity of the avatar to the Gremlock headquarters and a linear roll off was used.



```
Inspector
▼ ✓ Audio Source
Audio Clip            highs
                      This is a 2D Sound.
Mute                  ☐
Bypass Effects        ☐
Play On Awake         ✓
Loop                  ✓
Priority              128
Volume                1
Pitch                 1
▶ 3D Sound Settings
▶ 2D Sound Settings
▼ ✓ Audio Source
Audio Clip            mids
                      This is a 2D Sound.
Mute                  ☐
Bypass Effects        ☐
Play On Awake         ✓
Loop                  ✓
Priority              128
Volume                0.64948
Pitch                 1
▶ 3D Sound Settings
▶ 2D Sound Settings
▼ ✓ Audio Source
Audio Clip            lows
                      This is a 2D Sound.
Mute                  ☐
Bypass Effects        ☐
Play On Awake         ✓
Loop                  ✓
Priority              128
Volume                0.10997
Pitch                 1
▶ 3D Sound Settings
```

Figure 37: Audio sources for each stem

4.8 Version Control

We used SVN for version control of the project. It was separated into three sections: trunk, branches, and tags. The trunk held the latest stable build of the project. Branches held alternate version of the project which may or may not have been stable. Tags held references to specific versions of the projects which held value such as alpha, beta, and release. Within the trunk folder, the project had three folders: Builds, Gremlock, and New Assets. The Builds folder held the executable versions of the project to show off. The New Assets folder contained all new assets which were prepared to enter the art pipeline. The Gremlock folder held the Unity project. We only committed the Assets folder and the Project Settings folder to the SVN in-order to avoid conflict among temporary files.

4.9 Miracles

One of the most important design elements in our game was the inclusion of miracles. All the miracles instantiate either an object, a script or both and has an effect over time. After some time the instantiated object destroys itself.

4.9.1 Healing Circle

The healing circle allows the player to heal the avatar and Aetherian units. This is the simplest miracle. It is a spinning cylinder with a collider that sends a message to heal itself to any unit inside of it. In order to prevent the cylinder from clipping with the terrain as much as possible we made sure to raise it enough to avoid enough height differences while at the same time still being low enough so that all units can collide with it.

4.9.2 Meteor

When the player casts this miracle it summons a meteor to fall from the sky, damaging nearby units and leaving behind a crater at the point where it collided with the terrain. Once the miracle has been cast the meteor is instantiated some distance above the ground (out of sight from the camera) and a force is applied to it that causes to head towards the earth at high speeds. Once the meteor hits the ground it instantly creates a crater at the point of impact (see section 4.2 Terrain Manipulation) and sends a message to all nearby units to take damage.

4.9.3 Volcano

When the player casts this miracle it causes a volcano to rise from the ground. Once the volcano has risen from the ground it erupts and damages nearby units. Raising a volcano causes a change on the terrain (see section 4.2 Terrain Manipulation) similar to the meteor miracle, with the difference that the volcano changes the terrain over time. It does so by linearly interpolating between the initial terrain condition and the resulting terrain heights.

4.9.4 Flood

Casting this miracle allows the player to flood a section of the terrain, restoring it to its initial condition before any miracles were cast on it. The flood changes the terrain (see section 4.2 Terrain Manipulation) over time similar to the volcano miracle, but it has to remember the original heights of the terrain from when the game was started. It also uses linear interpolation. Once it finishes restoring the terrain over the established period of time it makes sure to explicitly set the heights to the original values so that irregularities on the terrain are prevented and the heights are fully restored.

4.9.5 Miracle Management

Since miracles are one of the most prominent mechanics in out game their implementation requires attention beyond the miracle itself. Consideration was given to when and how the player could cast miracles and how the effects of the miracles would last over time. The mechanics discussed in this section have a great impact on gameplay and they required fine tuning in order to make them effective.

Capturing Relics

Each faction in the game has to fight over a relic in order to determine who can claim control over its powers. How this works is that any unit within a certain distance of the relic gradually changes the value of a control variable over time towards one extreme or the other, depending to which faction the unit belongs to. Once the control variable reaches a certain value, that relic is captured by the given faction (See Figure 38). Controlling a relic has different effects on each faction. Aetherians can cast miracles when controlling relics and Gremlocks simply become stronger.

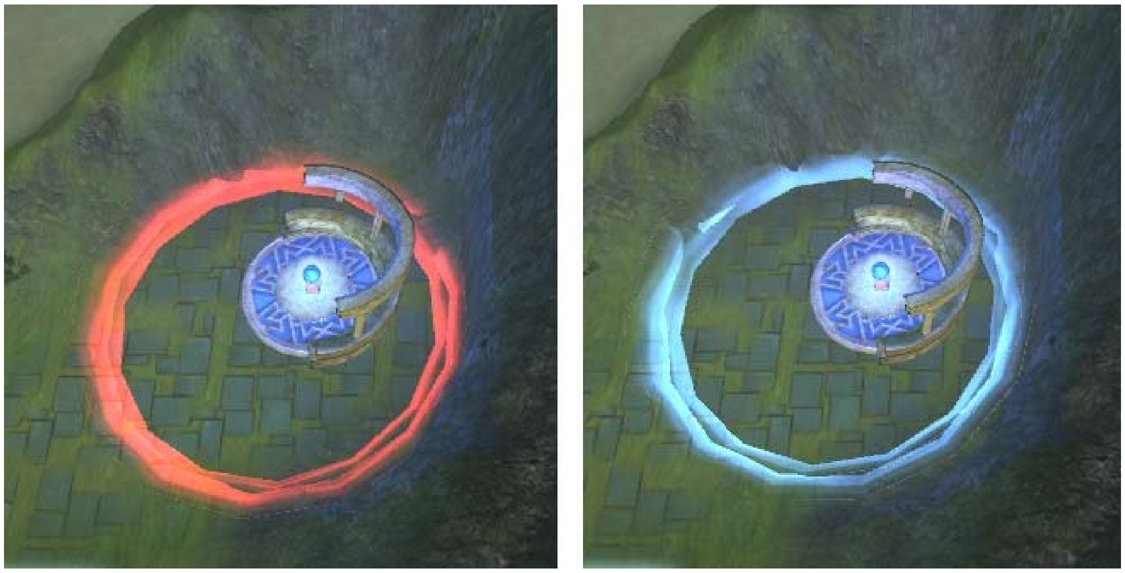


Figure 38: (a) Relic under Gremlock control (b) Relic under Aetherian control

Casting time

In order to balance the game we saw a need to prevent the user from being able to instantly cast a miracle. Instead we added a casting time, forcing the user to think ahead of time where and when a miracle might be useful in the midst of battle. It is basically a timer that depends on the right mouse button being held down. It also needs to be aware of miracle control and cooldowns (see next section).

Cooldowns

Cooldowns were implemented in order to prevent the player from consecutively casting the same miracle multiple times in a short period of time, giving the Aetherians an unfair advantage against the Gremlocks. Each miracle has its own cooldown time. It is basically a timer that runs once a miracle has been cast. It needs to take into account when a relic is controlled by the Aetherians. If a relic is overtaken by the Gremlock and then restored to the Aetherians, the cooldown continues right from where it left off.

Erosion

It is very easy for the player to permanently gain an advantage by blocking certain sections of the terrain with either the meteor or the volcano miracles. In order to make this just a temporary advantage and provide the opportunity for the Gremlocks to regain power, we implemented erosion of the terrain. Any terrain deformation that occurs during gameplay is gradually restored to its original condition over time (See Figure 39). The interpolation between the terrain deformation and the original terrain height values is asymptotic.

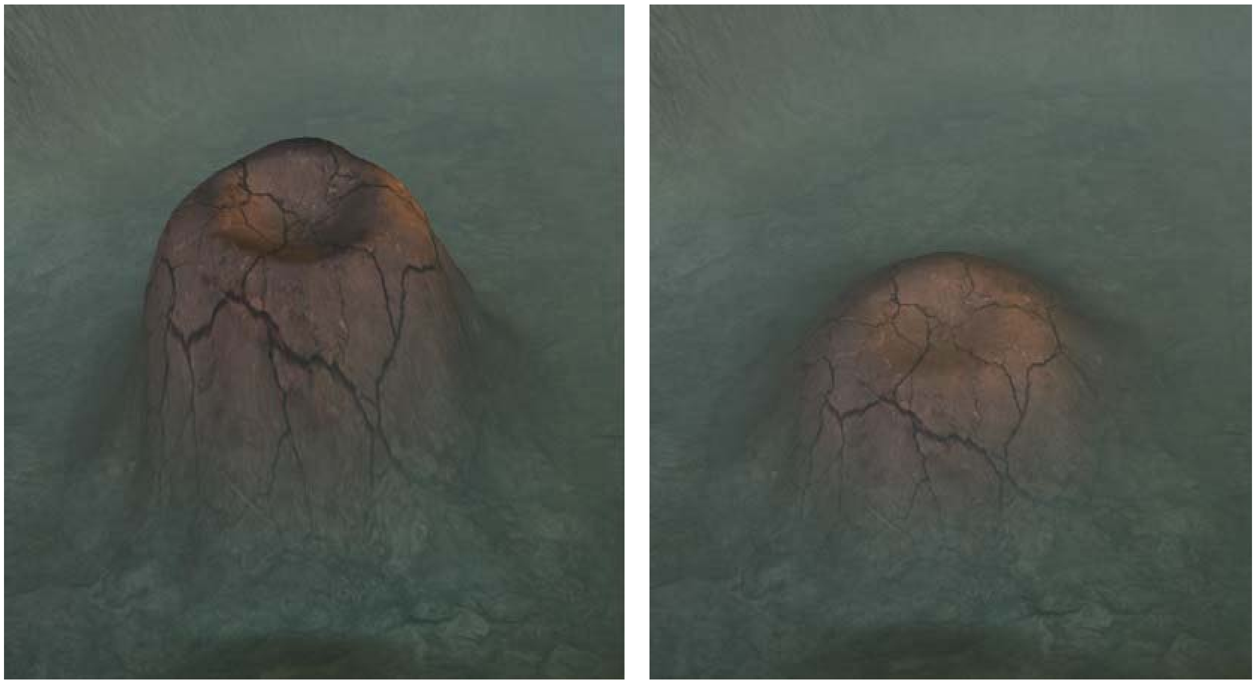


Figure 39: (a) Volcano without erosion (b) Volcano after some erosion

4.10 Combat

We kept the combat system simple, so it could be generalized between units. Each unit which participates in combat has some kind of combat script attached to them which handles the damage messages. When a unit attacks, it finds the closest target within range and sends that target a takeDamage() message. Then, it activates a timer until the next time the unit should attack. The target handles the takeDamage() message and determines whether the unit should be destroyed.

5. Tech Art

As with any game development project one of the most important components is the pipeline for integrating art assets into the game. We had to establish some sort of system for sharing completed assets between team members and for getting those assets into the game. We attempted to take advantage of any method which allows us to automate a step in the pipeline. We will now describe how we approached this problem.

5.1 New Assets Folder

The first step of the process was getting a completed art asset from the artists into the programmers' hands. The primary way in which we accomplished this was by sharing a folder in the SVN where the artists could place a copy of their completed art asset. We called this the 'New Assets' folder. A programmer then could then take that copy and place it into the Unity project, deleting the copy from the 'New Assets' folder. Anything that was still in the folder still needed to be imported into the game. Once the asset had been imported it would often also need extra attention for integrating it into the game.

5.2 Tech Integration

Unity has a nice feature which automatically imports new files placed in the project folder once the editor is opened. Sometimes using this feature is all it takes to make an asset ready to be placed into the game. However, it is more often the case that more effort is required in order to get the assets ready to be integrated into the game.

5.2.1 Models

The integration of models was perhaps one of the most complicated processes in the sense that it required careful attention to properly put together the model in the project. Any time a new model was brought into Unity we had to make sure that the correct textures and normal maps were applied to it. The whole process had to be repeated any time the artist polished the models further.

5.2.2 Animations

Each model also had to be paired with its corresponding set of animations. In order to control the playing of animations each model had an animation controller script. Variables are set by other scripts in order to trigger certain animations; the animation controller then evaluates these variables to determine how to play the animations.

5.2.3 UI textures

Most textures could simply be added to the project folder in order to use them in the game. However, since we were using NGUI for many of our UI elements, adding UI textures required extra steps. NGUI organizes sprites by using an Atlas. What the Atlas basically does is that it maps all the sprites into one big texture. NGUI does offer the ability to add multiple textures into the atlas at once, but every time we tried this, the program would crash. Every sprite had to be added to the atlas one at a time, which was very time consuming.

5.2.4 Audio

Our game also featured many sound effects. One thing we did to alleviate the stress of audio integration was to create a ‘SoundList’ script (see section 4.6.1 Sound

List). This script makes it very easy for the artists to import and change sounds themselves, since the sounds are already implemented on scripts elsewhere.

6. Audio

A major part of the project was to include a fully integrated audio engine which would put sounds in appropriate context and add to the immersion and depth of the game.

6.1 Sound Design Process

In order to get a list of what audio we needed, we established an audio asset list which would declare the amount of polish and the status of a sound. This list was divided up into sections depending upon their usage (environment, buildings & structures, animations, UI, and music). The sounds were then ranked by importance, making the most prominent sounds highest priority, and sounds that may not even be used as the lowest priority.

6.1.1 Recording Sounds

Sounds were recorded from various Foley techniques and other real life audio. The raw audio in turn was modified in Vegas to be made useable by the game engine. This raw audio includes samples such as clashing of knives for sword sounds and vocalized grunts and voices for infantry.

6.2 Sound Effects

Sound effects add a level of feedback and environment to the game. Audio was sequenced and arranged in Sony Vegas as a tracks editor (See Figure 40). This allowed for careful filtering of sounds such as volume adjustment and changing the Equalization. We also made use of various fade effects. Using a track editor such as Vegas also allowed for overlaying of samples and looping of sounds. Vegas was also useful for organization purposes.

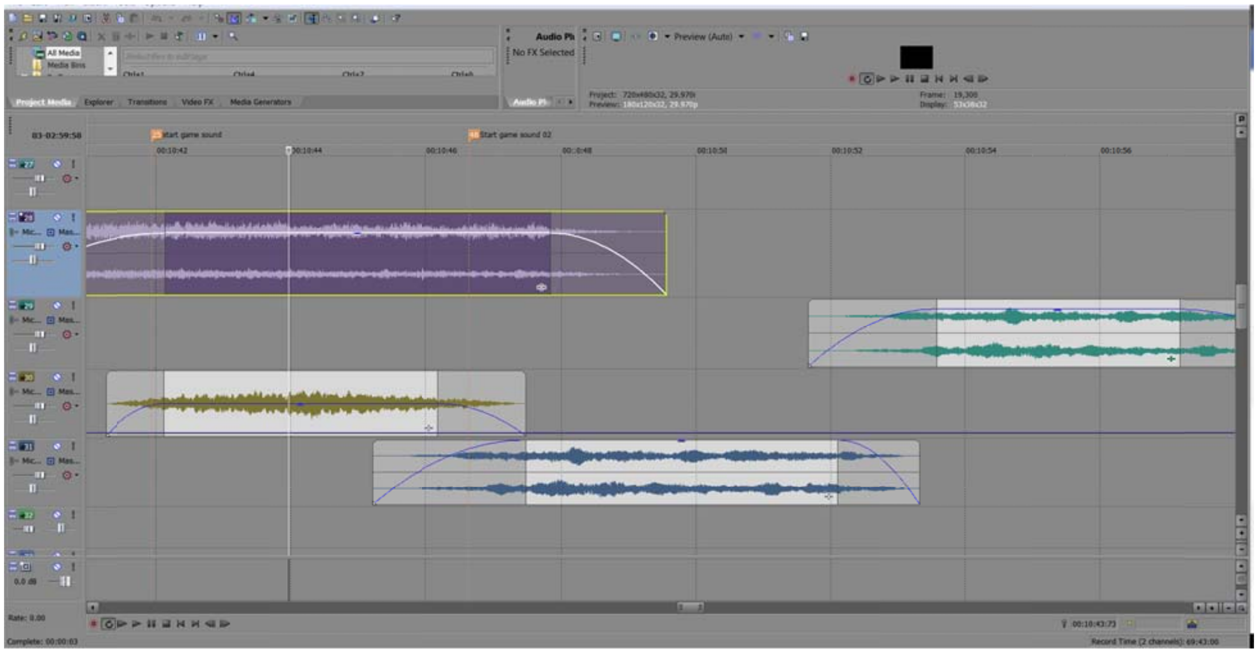


Figure 40: Snapshot of sound FX being edited in Sony Vegas

6.3 Music Composition

The music in-game makes use of a stem system in which the piece is divided up into 3 tracks consisting of different instruments. The stems are named “highs,” “mids,” and “lows.” The high stem consists of guitar, strings, and a high pass drum. The mid stem consists of organ, another guitar, and bass guitar. The low stem consists of drums and a brass section.

Just the highs are played when the Avatar is in the friendly base. Once the Avatar is out of the base and in a more neutral controlled area, the mid stem is added to the mix. When the Avatar enters the area associated with the enemy base, the highs are muted and the lows are added in to increase the sense of danger. A dynamic music system is one of the many methods we use to increase immersion in the game.

6.3.1 Composing in Cubase

The music stems were all part of the same document; the highs and lows consisted of the same melody but played using different instruments. The mids act as a support to the other two stems with chords and whole notes. The goal of the composition was to make something immersive, but not too jarring as to distract from the actual gameplay.

Respective tracks were divided into folders, “lows” “mids” and “highs”. When exporting the stems, two folders were muted while one was playing and it was exported under its respective name.

6.4 Integration

Audio samples were integrated using a database system that was designed from the ground up that allowed for faster audio integration. Within the file system, audio samples were divided into folders dependent upon their use.

6.4.1 Playback

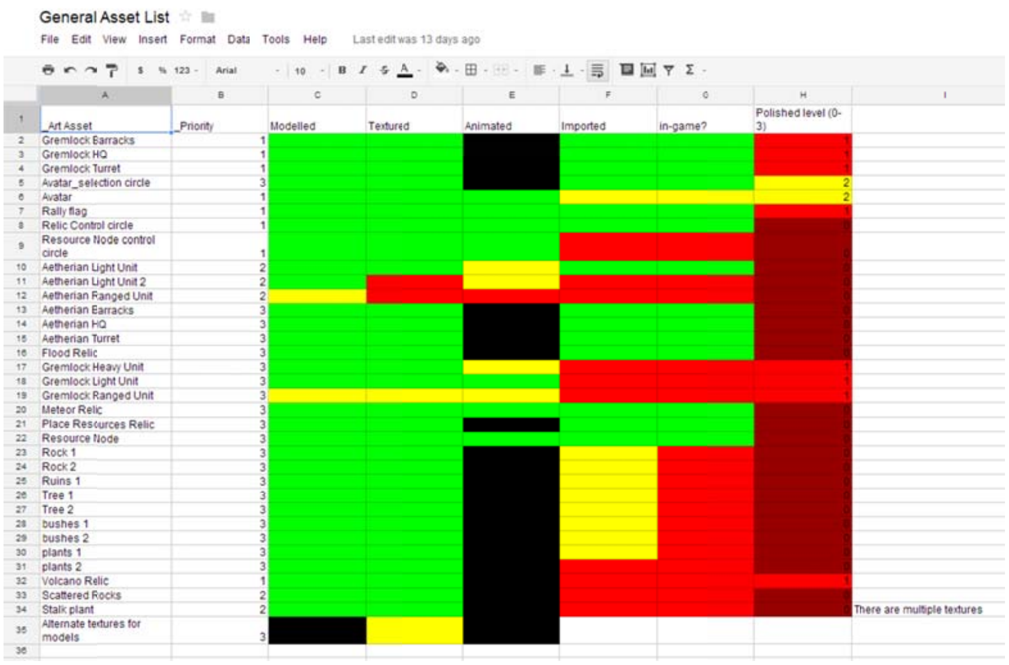
Audio playback was put in place based on triggers. Audio was capable of scaling dynamically and giving certain samples priority as to not become cacophonous. Music and crowd sounds were based on loops that would mute and unmute and change volume dynamically to give the game auditory flavor. Other audio was more explicitly triggered such as the indicator sound to tell the player that either side had captured a relic. The Sound Manager helped manage the sound effects, such as the sword effects, so that the soundscape would not become cluttered.

7. Project Management

Over the course of the project, we made use of a variety of tools to keep the project on schedule. We heavily used Google Drive to track hours, bugs, and share documents. We met regularly with the advisers to go over progress and separately to plan out the week's work. We made good use of email to keep in contact with each other and update to date on what needed to get done.

7.1 Google Drive

We used Google Docs extensively to organize and manage the project. One of the first things we set up was an Asset List which contained all the art, audio, and technical assets which needed to be created (See Figure 41 and Figure 42). Each asset had a priority and a set of colored boxes which showed how complete each asset was. Over the course of the project, we used this spreadsheet to keep track of our hours spent on the project each week. Any time we worked on the project, we would write the amount of time spent into one of the colored boxes to indicate that we had worked for so many hours on that asset at that stage in the art pipeline.

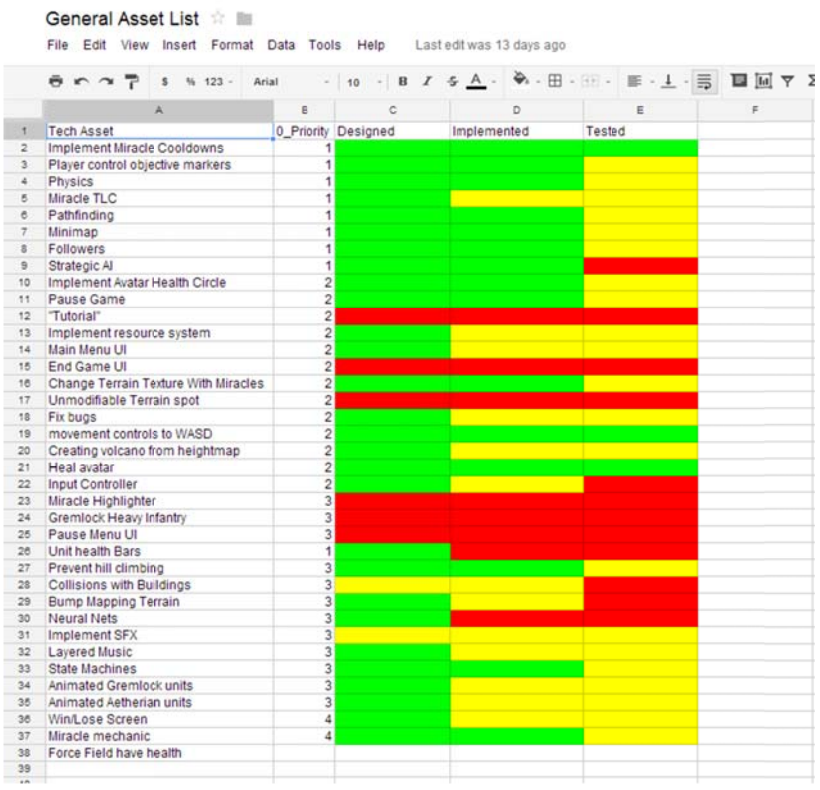


General Asset List

File Edit View Insert Format Data Tools Help Last edit was 13 days ago

	A	B	C	D	E	F	G	H	I
1	_Art Asset	_Priority	Modelled	Textured	Animated	Imported	in-game?	Polished level (0-3)	
2	Gremlock Barracks	1	🟩	🟩	⬛	🟩	🟩	🟥 1	
3	Gremlock HQ	1	🟩	🟩	⬛	🟩	🟩	🟥 1	
4	Gremlock Turret	1	🟩	🟩	⬛	🟩	🟩	🟥 1	
5	Avatar_selection circle	3	🟩	🟩	⬛	🟩	🟩	🟨 2	
6	Avatar	1	🟩	🟩	🟩	🟨	🟨	🟨 2	
7	Rally flag	1	🟩	🟩	🟩	🟩	🟩	🟥 1	
8	Relic Control circle	1	🟩	🟩	🟩	🟩	🟩	🟫 0	
9	Resource Node control circle	1	🟩	🟩	🟩	🟥	🟥	🟫 0	
10	Aetherian Light Unit	2	🟩	🟩	🟨	🟩	🟩	🟫 0	
11	Aetherian Light Unit 2	2	🟩	🟥	🟨	🟥	🟥	🟫 0	
12	Aetherian Ranged Unit	2	🟨	🟥	🟥	🟥	🟥	🟫 0	
13	Aetherian Barracks	3	🟩	🟩	⬛	🟩	🟩	🟫 0	
14	Aetherian HQ	3	🟩	🟩	⬛	🟩	🟩	🟫 0	
15	Aetherian Turret	3	🟩	🟩	⬛	🟩	🟩	🟫 0	
16	Flood Relic	3	🟩	🟩	⬛	🟩	🟩	🟫 0	
17	Gremlock Heavy Unit	3	🟩	🟩	🟨	🟥	🟥	🟥 1	
18	Gremlock Light Unit	3	🟩	🟩	🟩	🟥	🟥	🟥 1	
19	Gremlock Ranged Unit	3	🟨	🟨	🟨	🟥	🟥	🟥 1	
20	Meteor Relic	3	🟩	🟩	🟩	🟩	🟩	🟫 0	
21	Place Resources Relic	3	🟩	🟩	⬛	🟩	🟩	🟫 0	
22	Resource Node	3	🟩	🟩	🟩	🟩	🟩	🟫 0	
23	Rock 1	3	🟩	🟩	⬛	🟨	🟥	🟫 0	
24	Rock 2	3	🟩	🟩	⬛	🟨	🟥	🟫 0	
25	Ruins 1	3	🟩	🟩	⬛	🟨	🟥	🟫 0	
26	Tree 1	3	🟩	🟩	⬛	🟨	🟥	🟫 0	
27	Tree 2	3	🟩	🟩	⬛	🟨	🟥	🟫 0	
28	bushes 1	3	🟩	🟩	⬛	🟨	🟥	🟫 0	
29	bushes 2	3	🟩	🟩	⬛	🟨	🟥	🟫 0	
30	plants 1	3	🟩	🟩	⬛	🟨	🟥	🟫 0	
31	plants 2	3	🟩	🟩	⬛	🟥	🟥	🟫 0	
32	Volcano Relic	1	🟩	🟩	⬛	🟥	🟥	🟥 1	
33	Scattered Rocks	2	🟩	🟩	⬛	🟥	🟥	🟫 0	
34	Stalk plant	2	🟩	🟩	⬛	🟥	🟥	🟫 0	There are multiple textures
35	Alternate textures for models	3	⬛	🟨	⬛				
36									

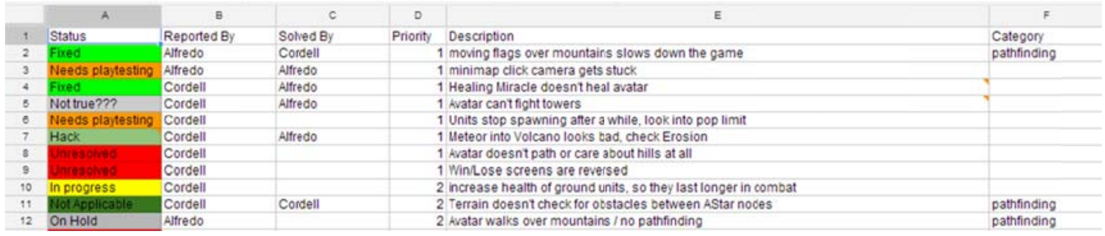
Figure 41: 3D Art Asset List


General Asset List
File Edit View Insert Format Data Tools Help
Last edit was 13 days ago
Arial
10

	A	B	C	D	E	F
1	Tech Asset	0_Priority	Designed	Implemented	Tested	
2	Implement Miracle Cooldowns	1	🟩	🟩	🟩	
3	Player control objective markers	1	🟩	🟩	🟨	
4	Physics	1	🟩	🟩	🟨	
5	Miracle TLC	1	🟩	🟨	🟨	
6	Pathfinding	1	🟩	🟩	🟨	
7	Minimap	1	🟩	🟩	🟨	
8	Followers	1	🟩	🟩	🟨	
9	Strategic AI	1	🟩	🟩	🟥	
10	Implement Avatar Health Circle	2	🟩	🟩	🟨	
11	Pause Game	2	🟩	🟩	🟨	
12	"Tutorial"	2	🟥	🟥	🟥	
13	Implement resource system	2	🟩	🟨	🟨	
14	Main Menu UI	2	🟩	🟨	🟨	
15	End Game UI	2	🟥	🟥	🟥	
16	Change Terrain Texture With Miracles	2	🟩	🟩	🟨	
17	Unmodifiable Terrain spot	2	🟥	🟥	🟥	
18	Fix bugs	2	🟩	🟨	🟨	
19	movement controls to WASD	2	🟩	🟩	🟩	
20	Creating volcano from heightmap	2	🟩	🟨	🟨	
21	Heal avatar	2	🟩	🟩	🟩	
22	Input Controller	2	🟩	🟨	🟥	
23	Miracle Highlighter	3	🟥	🟥	🟥	
24	Gremlock Heavy Infantry	3	🟥	🟥	🟥	
25	Pause Menu UI	3	🟥	🟥	🟥	
26	Unit health Bars	1	🟩	🟥	🟥	
27	Prevent hill climbing	3	🟩	🟩	🟨	
28	Collisions with Buildings	3	🟨	🟨	🟥	
29	Bump Mapping Terrain	3	🟩	🟨	🟥	
30	Neural Nets	3	🟩	🟥	🟥	
31	Implement SFX	3	🟨	🟨	🟨	
32	Layered Music	3	🟩	🟨	🟨	
33	State Machines	3	🟩	🟩	🟨	
34	Animated Gremlock units	3	🟩	🟨	🟨	
35	Animated Aetherian units	3	🟩	🟨	🟨	
36	Win/Lose Screen	4	🟩	🟨	🟨	
37	Miracle mechanic	4	🟩	🟩	🟨	
38	Force Field have health					
39						

Figure 42: Tech Asset List

Once we hit Beta in late C-term, we created a Bug List (See Figure 43) which kept track of all the bugs found during playtesting and what their status and priority was. From then on, the programmers started to reference that spreadsheet to determine what to work on and what needed to be fixed rather than the feature list in the Asset List document.



	A	B	C	D	E	F
1	Status	Reported By	Solved By	Priority	Description	Category
2	Fixed	Alfredo	Cordell	1	moving flags over mountains slows down the game	pathfinding
3	Needs playtesting	Alfredo	Alfredo	1	minimap click camera gets stuck	
4	Fixed	Cordell	Alfredo	1	Healing Miracle doesn't heal avatar	
5	Not true???	Cordell	Alfredo	1	Avatar can't fight towers	
6	Needs playtesting	Cordell		1	Units stop spawning after a while, look into pop limit	
7	Hack	Cordell	Alfredo	1	Meteor into Volcano looks bad, check Erosion	
8	Unresolved	Cordell		1	Avatar doesn't path or care about hills at all	
9	Unresolved	Cordell		1	Win/Lose screens are reversed	
10	In progress	Cordell		2	increase health of ground units, so they last longer in combat	
11	Not Applicable	Cordell	Cordell	2	Terrain doesn't check for obstacles between AStar nodes	pathfinding
12	On Hold	Alfredo		2	Avatar walks over mountains / no pathfinding	pathfinding

Figure 43: Top of Bug List

7.2 Scheduling

At the beginning of the project we decided track the number of hours each person had spent on the project during the week as a means of keeping on schedule. As mentioned above, we tracked our hours for the week in the asset list. At the end of each week, we compiled together the list of hours spent on the project and sent that list to the advisers. We tracked our hours spent on the project each week, rather than the completed tasks for each week.

7.3 Meetings

We met with the advisors for an hour each week to present our progress. At each meeting, we wrote up an agenda to dictate the direction of the meetings. In addition to our meetings with the advisers, we met as a group once a week to divide work and go over what we needed to complete that week.

8. Post Mortem

Relic: TAU was an invaluable experience for us as students to be able to work in a production team developing a 3D game of our own design. Through our journey, there were some things which we felt we succeeded at doing and other things which we felt could have gone better. It is important for us to look at the shortcomings of our project in order to learn about how to avoid or relieve these issues should we encounter them while working on future projects.

8.1 What Went Right

During the development of Relic, there were many things that went well or sometimes exceeded our expectations. The major items are listed below in order of priority.

8.1.1 Unity

Choosing to build our game in the Unity engine was one the best decisions we made. Although there were some drawbacks, overall it made sense for us to choose an engine we were all familiar with. Having had experience working with Unity made it easy for us to get a working prototype up as fast as possible. Since we already knew the basics of developing a game in Unity, we avoided having to deal with having to learn to use a new program. Unity has an excellent community which provides invaluable support to anyone working with the engine. Also, the Unity engine had several features which we took advantage of over the course of development including the various terrain features, the particle systems, and the physics engine. These features greatly speed along the project's development.

When we decided on working with Unity, we also decided that we would use C# as the main language for the project. Unity supports scripts written in C#, JavaScript, and UnityScript, so we had the option of using a few different languages within Unity. Attempting to use multiple languages simultaneously would have ended badly, since there are some difficulties when referencing a C# file in JavaScript or vis-versa. C# is a comprehensive object oriented language which is ideal for game development and we had more experience in it than JavaScript or UnityScript, so we went with C#.

Organizing the code into objects makes it much more readable and easier to debug. This was valuable later in the project as we had to return to code we had written several months earlier. Also, we were able to make use of freely available C# code to simplify our work, specifically the majority of our A* code made use of a C# implementation written by Eric Marchesin. Building the game in Unity was generally good idea and helped us speed the development along.

8.1.2 Google Docs

We found out quickly that Google Documents was a good resource to utilize for real-time group collaboration and also for tracking the extensive list of objectives and goals for our project. Project management became a much easier task by using Google docs to delegate the work that needed to be accomplished each week. Something which also became an important part of our project was the use of tracking sheets on Google docs. Our team developed tracking sheets for all of the art, sound, and technical items which we wanted to get into the game and we used color coding to sort out what items had already been worked on and which ones needed more attention. This method of documentation worked wonderfully to keep our teamed focused and organized for the tasks at hand each and every day.

8.1.3 UI

One of the areas we feel as though we succeeded in was creating a user-friendly UI which was simple yet effective at communicating the mains aspects of gameplay.

The focus of our UI was to have it accessible to everyone who plays and yet have it able to provide them with key information about the state of the game. In creating a simplistic minimap system and the miracle selection toolbar which, depicted the different miracle powers, we feel as though we met our original goals.

8.2 What Went Wrong

Although there were many things which we feel we did right during development, there were some things which we felt we could have done better. These items are listed below in order of priority.

8.2.1 Playtesting

A major area of weakness for our game was playtesting. At times it was difficult for us to organize playtesting as a group and to find playtesters willing to give us constructive feedback about gameplay. Part of the problem of this stemmed from the many different builds and states of the game. Since we were changing major aspects of gameplay in a short span of time, it was difficult for us to determine which build was suitable for playtesting. Most of our playtesting occurred sporadically and between separate small groups of people.   Part of this problem could have been alleviated by being stricter about deadlines for playtest builds and standardizing their distribution. For example, we could have been more proactive about releasing a playable build of the game online and get feedback online as well. Planning better for playtesting could have given us more valuable information on how to better balance the game.

8.2.2 Prototyping

When we started the project, we built a physical prototype of the game and did some minor playtesting. The problem we had was that we never built a functional, digital prototype of the game. We ended up following a waterfall approach to development, in which we attempt to build everything at once and hope that we would have a fun game at the end of the project. The result was that we had no way to properly playtest our game until extremely late in the project. We should have broken the design down more, so that we could build a complete, playable, fun version of the game in a week or two. That system of development would have focused our time and probably resulted in a better game.

8.2.3 Animations

The animations in our game needed more attention than they received. It was challenging for the artists to work on so many things at one time including creating sound effects and music. Not much time was left to spend polishing the character animations and they ended up being neglected. The animations which we were able to put in-game were much more basic than what we had originally planned for. Part of this failure to iterate on animation stemmed from a lack of pre-planning about who was working on what. From the beginning, the artists attempted to divide up animation work evenly between the Gremlock and Avatar characters. The reasoning behind splitting up the work was mostly due to both artists wanting to gain more experience with animation. Since the game was built partially for academic purposes, both of the artists attempted to work on and learn about all aspects of game art. However, looking back, it might have been more efficient to have one person who was dedicated to creating efficient character rigs and having the other person animate each character. Using this method could have helped alleviate problems with animation inconsistency.

8.2.4 UI

Although we feel that we were successful in the design and creation of the user interface, we felt that there were some issues with its implementations should. Part of the user interface was implemented using the default Unity UI system and the other part was implemented using the NGUI system. This difference caused some inconsistency when different assets were being drawn and scaled on the screen. A solution to this issue would have been to have had picked one UI system to stick to for the entire development.

8.2.5 Pathfinding

Pathfinding proved to be a challenge throughout the project. We spent a lot of time attempting to implement either physics-based movement or positional-based movement. The physics-based movement kept having trouble handling tight corners and cost a lot of processing time, while the positional-based movement had difficulty dealing with the changing terrain. In hind sight, we should have just spent more time designing a solid pathfinding system early on before implementing

8.2.6 Unity

Even though working with the Unity engine was a major part that we did right, it did not come without its weaknesses. Though we knew there would be some struggle with having to deal with performance issues, we were not fully prepared to deal with the kinds of problems which came up. Since Unity is designed to be multi-platform, it chooses not to take full advantage of the computer. Specifically, Unity 3.5 does not use the GPU for rendering, so it has trouble rendering complex scenes of objects. We ran into this as we started filling the scene with plants and units, the renderer took a long time to process everything. Many of these rendering issues could have been solved by simply using an engine which made full use of the GPU available on the machine.

8.2.7 Scheduling

Another area which leds to some deficiency in work for our game was scheduling. Each week we kept track of what each group member was working on based off of the number of hours worked. In hindsight, it would have been more productive to delegate specific tasks to be accomplished by each person every week instead of focusing on how much time was spent working.

8.3 What Changed

During the course of developing our game there were some things which were changed between the time of our original concept until completing the final version. These items are listed below in order of priority.

8.3.1 Genre

The overall genre of our game changed many times during production. Although our original design called for the game to be more RTS in style, our final game is more of a crossover between game genres rather than a single genre. Our game has elements of a God game with casting miracles to influence battle movements and altering terrain but it also borrows some elements from other types of games such as MOBAs (Massive Online Battle Area). In a MOBA, two teams battle it out using special powers and troops to destroy the enemy stronghold. Our game map also takes some influence from MOBA games in that it has three distinct paths to get from one side of the map to the other. By the end of our development cycle we had realized that our game was unique in that it combined parts of different game genres and became a hybrid.

8.3.2 Camera

When we first began to create our game, there was a lot of debate surrounding the optimal camera angle and distance from which our game should be played. We looked at a number of different games which were similar to our own and tried out many different combinations of camera angles. In one our earliest game builds, we had the camera entirely attached to the Avatar character from a low centering angle. We later discovered through preliminary playtesting that the camera angle we chose severely limited the player's view of the map and became a hindrance to gameplay. After we changed the camera view to be detached from the Avatar and allowed the player to maneuver the camera freely, the game became much easier to play and understand.

8.3.3 Friendly Troop Controls

In the original concept of our game, the player has no control over the friendly troops units in the game. Like the enemies, the friendly units would be entirely controlled through strategic AI. We had hoped that by having the AI control the Aetherian units, it would make the game simpler and help avoid the potential problems with overwhelming the player with too many extras. However, once we implemented the troop AI, we saw that the gameplay was lacking in interactivity on the part of the player. When we realized the potential gameplay and strategy that went along with controlling the units, we came up with the Rally Flag system as a way for the player to have more control.

8.4 Final Thoughts

Overall, we were successful in building a game that embodied the spirit of our original ideas in look and gameplay. There were a few bumps in the road but we believe that facing these challenges has made us better suited to overcome similar issues in the future.

Appendix A: Concept Art

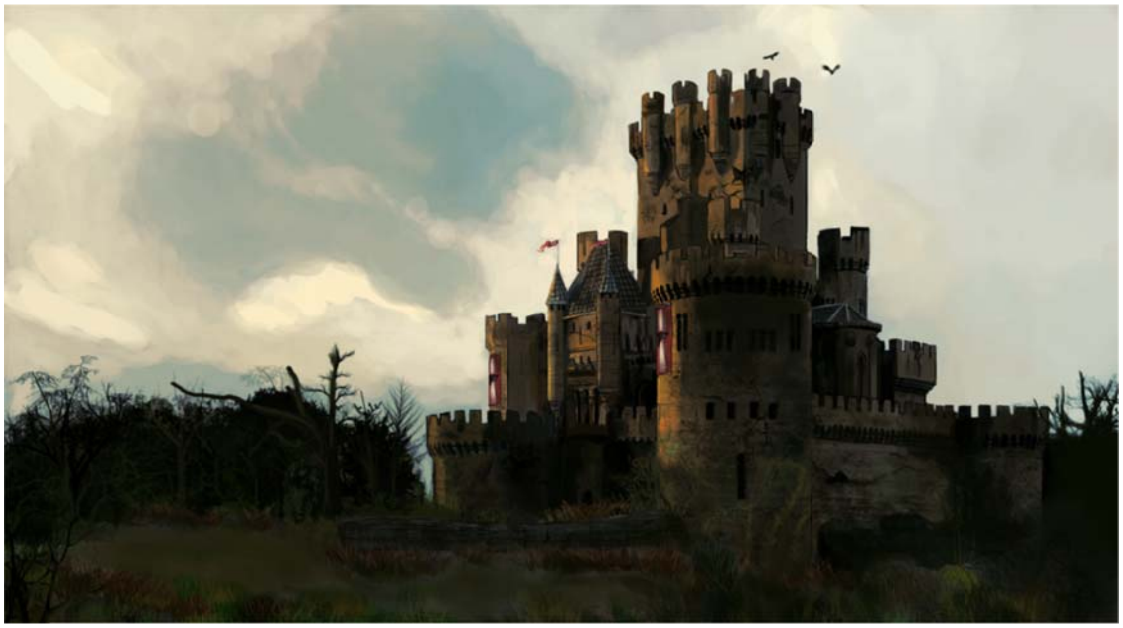


Figure 44: Aetherian HQ Concept

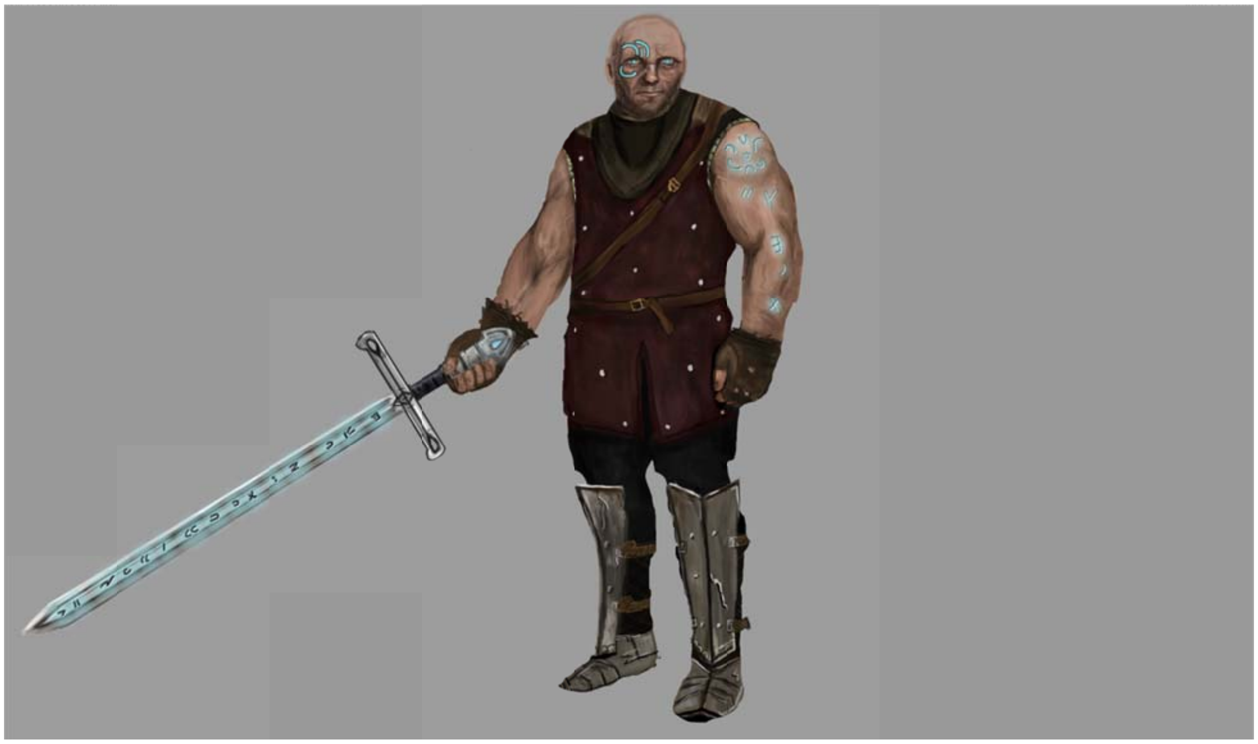


Figure 45: Aetherian Avatar Concept

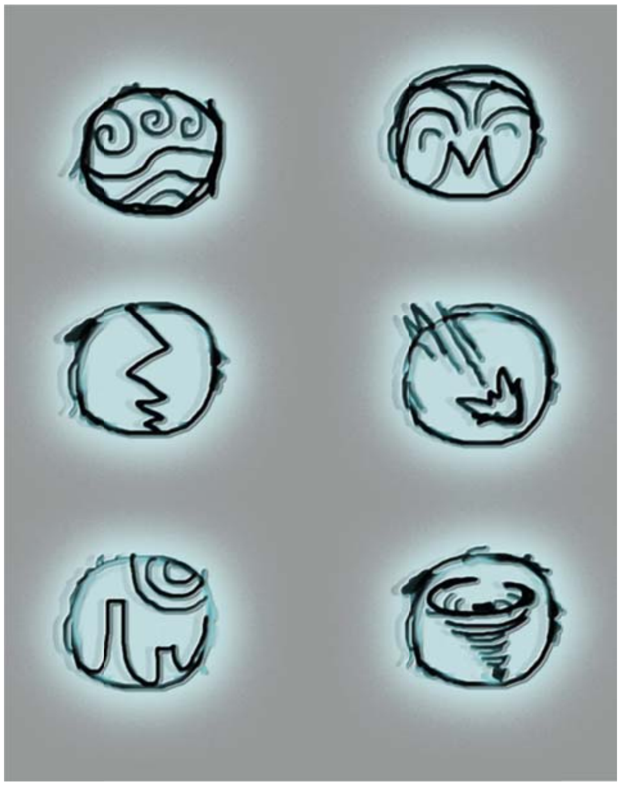
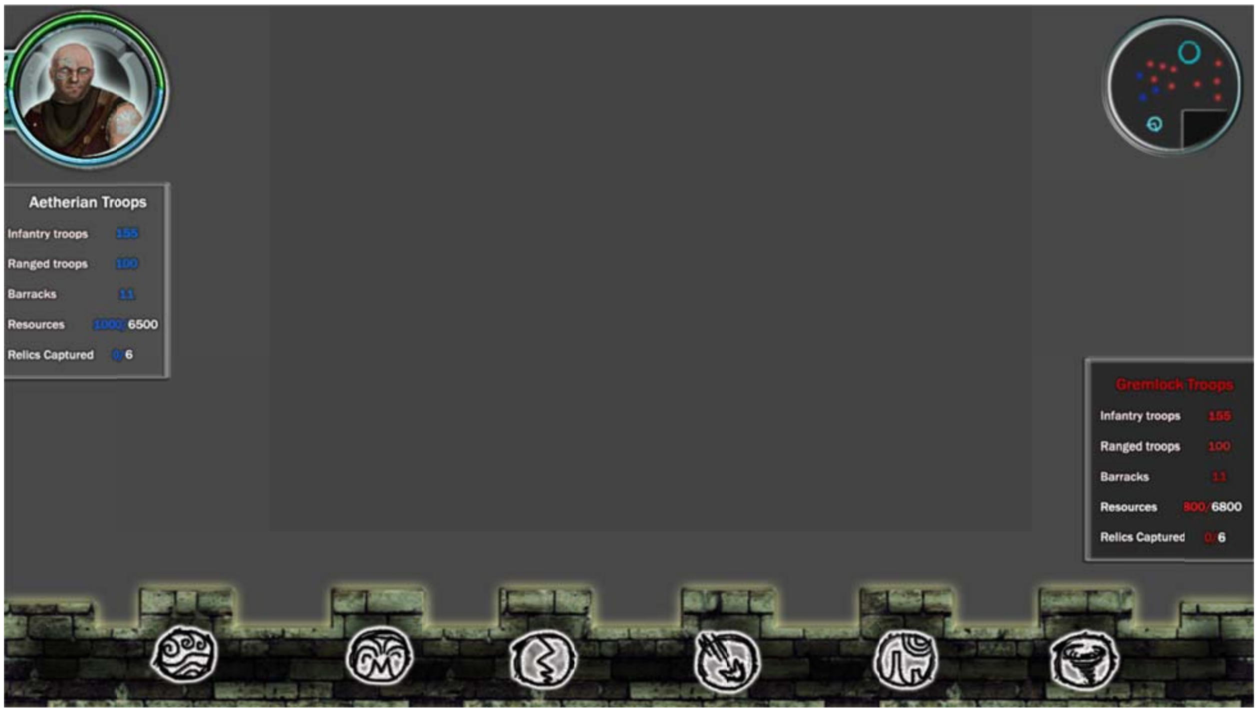


Figure 46: Miracle Icon Concepts



Aetherian Troops	
Infantry troops	155
Ranged troops	100
Barracks	11
Resources	1000/ 6500
Relics Captured	0/ 6

Gremlock Troops	
Infantry troops	155
Ranged troops	100
Barracks	11
Resources	800/ 6800
Relics Captured	0/ 6

Figure 47: User Interface Concept