APPLE OF DISCORD

Interactive Media and Game Development

Major Qualifying Project Report
Submitted to the faculty of
Worcester Polytechnic Institute, Worcester, MA
In partial fulfillment of the requirements for the
Bachelor of Science degree

Submitted To:

Professor David Finkel (Advisor)
Professor Ralph Sutter (Advisor)

Submitted by:

Suzanne DelPrete _____

Robert Doyle _____

Bryce Dumas _____

Benjamin Miller _____

Bolin Zhu _____

Date of Submission: 30 April 2014

_____
Advisor Signature

Abstract

Apple of Discord is an IMGD MQP in which a team of five students conceived, developed, tested, and re-defined a two-point-five dimensional, four person online game in the Unity3D[1] engine. Our conceptual focus was to develop a polished game about the thirteen Greek Olympians that stayed true to religion. We hoped to deliver an end product akin in entertainment value to those of professionally developed products. Every art asset in the game was first hand-sculpted then turned into 2D sprites by our artists. The game has fully functional LAN-based online multiplayer that supports up to four players. *Apple of Discord* blends familiar mechanics from games such as the *Civilization*[2] series, *League of Legends*[3], and *Final Fantasy Tactics*[4] to deliver a unique and easy experience.

[1] Unity3D http://unity3d.com/
[2] *Sid Meier's Civilization* http://www.civilization.com/
[3] *League of Legends* http://na.leagueoflegends.com/
[4] *Final Fantasy Tactics* http://www.final-fantasy-tactics.com/

Acknowledgements

We would like to thank David Finkel, our technology advisor, and Ralph Sutter, our art advisor, who were incredibly helpful in keeping the group on track to complete the project, and offering necessary advice for the actual creation of the game. Professor Sutter's suggestions on the art processes were very effective, and the game looks much better and was far more easily designed than it would have been had we used our original plans for art asset creation. Bryce would also like to thank Britt Snyder, whose critique of the character designs greatly enhanced the quality of work. We would also like to thank Professor Claypool and Mike Voorhis, lab manager for the CS department, who were very helpful in establishing the virtual machine, despite the fact that we opted not to use it.

Table of Contents

Part 1: Introduction

Apple of Discord is a 2.5D multiplayer RTS game in which the Greek gods battle each other for Eris's Apple of Discord. Each player controls one main god, and attempts to destroy their opponents' main base. The players direct civilians and military units to perform certain tasks. Each god has a passive and active ability that aid their citizens or harm their opponents.

This game was developed over nine months, from late August 2013 to early May 2014, with two months of developing concepts and preparation preceding development. Bryce Dumas assumed the role of producer and character artist, sculpting, texturing, rigging, animating and rendering characters. Suzanne DelPrete acted as the environment and UI artist, modeling and rendering environment pieces, and designing UI elements. Robert Doyle, Benjamin Miller, and Bolin Zhu acted at programmers and made important game design decisions. Bolin headed up the networking and UI programming, Rob focused on AI and citizen interactions, as well as buildings, and Ben worked mostly on pathfinding and city management.

To begin our project, we researched the Greek gods from various different sources, not only for artistic reference, but for their overall abilities. Although our gods differed in race, gender, and era, we wanted to stay true to their overall backstories. Our main reference was *D'aulaires' Book of Greek Myths*[5], a remarkably well-researched children's book that contains many major myths from ancient Greece. This book contained often lesser known facts, such as that Poseidon was the lord of horses. While the Greek myths are well known, we verified the rest of our research from various accredited sources across the internet. We also based some of our game play on MOBA style games such as *League of Legends*, and strategy games such as *Age of Empires*[6] and *Sid Meier's Civilization*.

In Part 2, we will talk about gameplay, including the game world, past and present goals, characters, the map and key objects, and overall game rules/actions. Part 3 focuses on the various art styles and asset creation pipeline while Part 4 will focus on the technology used to put *Apple of Discord* together.

[5] D'Aulaire, Indri and Edgar Parin d'Aulaire. *D'Aulaire's Book of Greek Myths.* New York: Delacorte Press, 1962.
[6] *Age of Empires* http://ageofempiresonline.com/en/

Part 2: Gameplay

The following sections detail the game design choices made during the production of *Apple of Discord*.

1. Game World and Setting

The game takes place in Ancient Greece. The player assumes the role of one of the gods of Olympus. Eris, seeking to cause turmoil, offers her golden Apple of Discord to whichever god can best their opponents in combat. The gods war against one another, using armies of followers and their own unique skills. The Apple is awarded to whoever can destroy their opponents' cities and armies.

2. Game Goals

i) Original Idea

At its conception, win conditions of *Apple of Discord* were intended to be a major feature. For each play session, each player would receive a different goal from a large list. Every goal would be attainable by any god, though some god abilities might better suit particular goals. Each player's goal would be kept hidden from other players. This opens a number of interesting options for a player to trick their opponents, by making it seem that they are aiming at a particular different goal, while secretly completing their real one.

We were also interested in offering non-violent options, and making goals that did not pertain entirely towards killing opponents. An effective way to accomplish this task would be to destroy buildings. Destruction is a fairly common theme in comparable MOBA and RTS games; nonviolent actions like diplomacy or deceit are rare and would have been refreshing to see in games.

When we actually tried to formulate goals, however, it proved very difficult to have interesting and balanced options. Many goals we came up with would have been incredibly easy for some gods, and impossible for others. The options that were balanced were often not very fun or engaging. When we did come up with good goals, there were so few that it would be easy to guess which goal everyone was aiming for, and the element of trickery would be lost.

Our original intent for the gods was for them to be similar to "hero" units in MOBAs, with the ability to level up and gain new abilities over time. As we worked on them, however, we ran into more problems of scale. Not only was it difficult to create even a set of four abilities for each god (let alone the entire ability tree for each we had originally intended), but the abilities also proved difficult to implement, due to the fact that they by design work outside the default framework of the game. Gods would also have been able to recruit demigods in order to gain access to some limited selection of

powers from outside their own domain, but needless to say this was not implemented either for similar reasons. In the end, each god got one passive and one active ability, and nothing else.

The game was also originally intended to have side quests randomly assigned by Eris, which could be completed for various benefits. As this was extraneous to the core game, we never implemented side quests to any degree at all, which is likely a good thing overall due to the simplicity of the map.

ii) Current Game Goals

On release, players in *Apple of Discord* will only have one win condition - destroy enemy units, buildings, and cities until there are none left or until they surrender. Whoever remains is the winner, the losers are ranked based on who survived the longest.

3. Game Characters

i) Choosing a Pantheon

When deciding what playable gods would be available to players, we had three options - create a totally original set of gods, bring in gods from a variety of pantheons, or choose a single set of gods. The first option would have provided great artistic freedom, but all of the playable gods would be unfamiliar, and players would need time to get to know our original characters. Merging different pantheons would be difficult since many gods' powers overlap. For instance, it would be difficult to make Zeus and Thor significantly different from each other.

We eventually decided on the Greek pantheon, but knew we had to mix them up in some way, since Ancient Greeks have become a bit cliché in games. Our solution was to give them modern personas.

ii) Official Cast

We originally intended to include the twelve Olympians and Hades, making a cast of 13 playable gods. Given the time constraint, we felt we could not make all the art assets needed for a cast of that size, and so we instead downsized to only four (the number of players in each playthrough of the game).

We settled for Zeus, Hera, Poseidon, and Artemis. This offered an even split of male and female characters. We also chose to represent a variety of races between the four of them. We hope that we have provided a respectful portrayal of characters of color, which is often lacking in mainstream games. We planned to also include characters of non-binary genders, and even greater racial diversity, had we the time and resources to add more characters.

Below are portraits of the four main characters, used as splash art during character select.


POSEIDON


ARTEMIS

4. Game Map and Objects

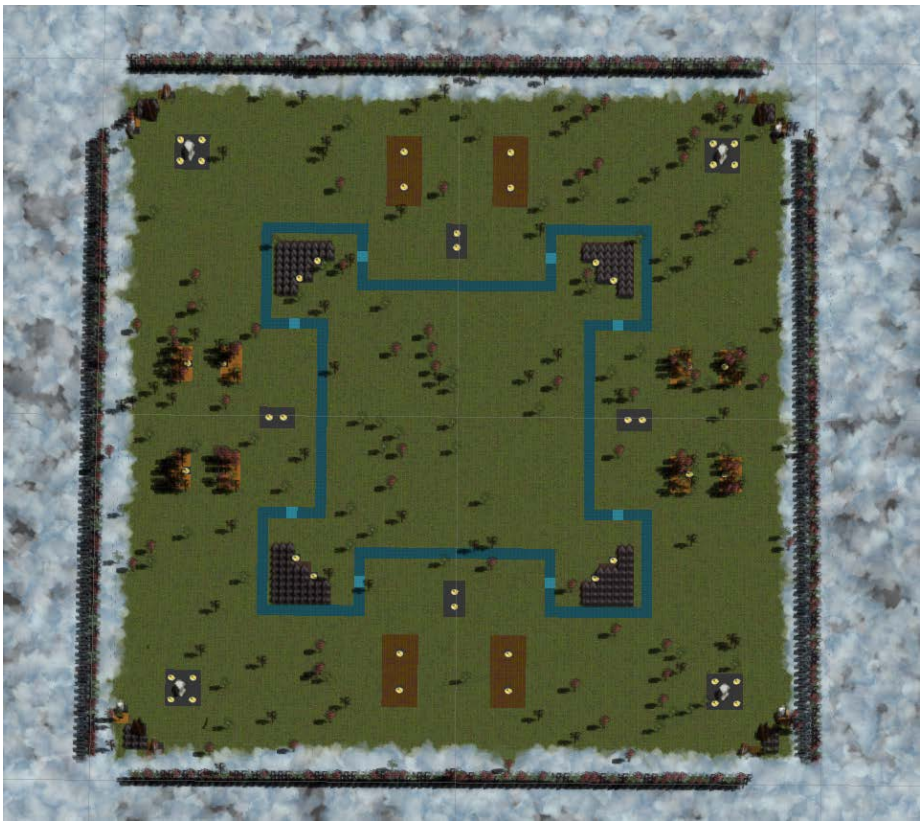
i) Game Map and Locations of Major Features

The game's map is split into four identical quadrants - one quadrant for each player. In the far corner of each quadrant there is a small square where the player can build their city. Right outside the city, further into the corner, are three plots for players to send their workers to harvest the game's three main resources - wood, ore, and food.

Right in the middle of the line between two quadrants there are outposts, which players may claim to expand their building further. They convey a strategic advantage, but must be defended or another player will claim them.

Further out from the city there are larger plots with the three resources. These may help players expand faster than the initial offering, but come with the price of being further away and thus harder to defend. Ore is most important for military expansion, and so we've placed it closest to the center of the map. This makes one player's ore more exposed to the other three, and thus more difficult to defend. The middle of the map is an island surrounded by a moat of water, which separates the ore from the rest of the map, further extending it outside the player's reach.

Below is a top-down view of the entire map:



ii) Cities and Outposts

Building a city is the player's first priority in Apple of Discord. Without cities, players aren't able to generate more units or any military, and will be left mostly defenseless. To start, each player is provided with 2 civilian workers, one each of the resource buildings, and a main hall.

The main hall spawns civilian units, and is also where they go to drop off the resources they've harvested. Players can click on their main hall to view and manage important data about the city, such as which units to prioritize, how much of each resource they have, etc. When a city's main hall is destroyed, the player loses, and whoever destroyed the city can claim the spot as their own to further expand their empire.

A city has four locations designated for buildings. Buildings are necessary to generate military units, increase capacity of resources, increase faith regeneration, and more. Each building requires a certain amount of resources to build. Once the player has enough, they need only select the location they'd like to build on, choose which building they would like to place, and a construction site will appear. When enough time has elapsed for the workers to build it, the building appears and begins functioning.

Outposts are smaller, and come with one extra build space. To claim a neutral outpost, the player needs only move one of their units inside its walls. To usurp an outpost claimed by another player, one needs to completely remove the enemy units and buildings within it.

iii) Resources and Gathering

Managing resources is integral to a player's success in Apple of Discord. There are three natural resources, food, ore, and wood, and each serve an important role in managing the player's population of worshippers. There is also faith, which is consumed when the player's god uses their abilities.

Food is responsible for increasing population count. Every 10 seconds, if enough food is available, a civilian unit will appear at the city hall. The player can choose to suspend growth by clicking on the city hall and deactivating it.

Since civilians are responsible for building and gathering and contribute to faith generation, it is crucial that players keep a good food supply. Extra food is also consumed when making cavalry units; horses need to eat too!

Ore is primarily used for military purposes. Some ore is necessary for buildings, particularly military buildings. It is also used to make new military units. Similarly to food, if enough ore is available, new units will appear near the military building corresponding to their unit type (melee, ranged, or cavalry) every 10 seconds. Players can decide which unit type has priority when ore runs out, or stop spawning units of that type altogether by clicking on any military building and deactivating it.

A large amount of wood is required to build each building, so it is a highly valuable resource, especially in the early game, when the player still has a lot to build. Making ranged units also requires some wood for each unit produced.

Faith is consumed every time a god uses one of their abilities. It replenishes at a constant rate, determined by population. Faith generation can be further increased by building and upgrading religious structures.

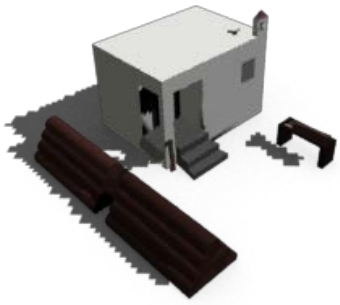
The natural resources are located at set locations on the map, as detailed in the game map section. To begin harvesting resources from resource tiles, the player will need to order two civilian units to build and operate a special improvement on the resource - mines for ore, farms for food, and lumber yards for wood. Once an improvement is in place, idle workers automatically move to collect resources from them and bring them to the nearest city.

iv) Buildings

Since building space is limited, players need to prioritize which buildings will help them the most. In addition, while all buildings function automatically by default, a player may deactivate a building if they do not want to pay for whatever it produces at the moment (i.e. deactivating a barracks to have more food for cavalry). Below are all of the buildings and what purpose each serves, as well as the resource and time cost to build them.

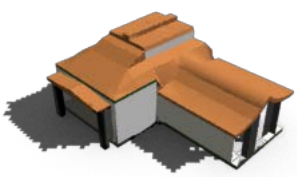
Farm, Lumberyard, and Mine:

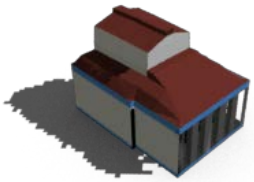
These three buildings are not built in either the city or outposts, but at set resource locations. Once built, workers will move to collect the resources each provides and return them to the main hall. Pictured below are the farm, lumberyard, and mine respectively.


Military Barracks, Archery Range, and Stables:

These buildings will spawn military units - Soldiers, Archers, and Cavalry respectively. Pictured below are these three buildings:





5. Rules and Actions

i) Basic Interaction

The main way in which the player interacts with the game world is by commanding units or groups of units. Players may select a unit by clicking on it, or a group of units by clicking and dragging. Additionally, the player may hold “s” while dragging to select only military units, “v” to select only soldiers, or “b” to select buildings. While buildings are not technically units, they are interacted with through the same interface. However, units and buildings cannot be selected at the same time.

When a group of units is selected, the player may right-click on a location in order to command the units to go to that location. If there are enemies nearby, the units will automatically begin fighting in accordance with their respective stances. The player may also select an individual unit in the group by clicking on the button with that unit’s ID on it in the unit command area at the bottom of the screen.

When only one unit is selected, there are a few more options. Instead of containing the IDs of all the units in the group, the command area for a single military unit will contain four buttons. Three of those buttons have the names of stances, any one of which may be selected in order to determine the unit’s behavior in the presence of enemies. This is a very useful tool, which allows units to be used for various different purposes depending on the stance they are in. Units ordered to take an aggressive stance will charge at the enemy, continuing to chase them even if they run away. Units ordered to be defensive will also charge, but will not chase enemies that run away. Units ordered to “Stand Ground” will stay in place, blocking enemy units from getting past them and attacking units in range (this is most useful with archers, who can act as turrets).The skull button will disband the unit, although there admittedly isn’t much reason to do this in actual gameplay.

Villagers are a bit different from military units. A villager’s stance cannot be set (although it can still be disbanded). Instead, villagers will, by default, gather resources automatically from whichever resource building has the most resources. A villager can also be assigned to collect only from a specific building, in which case it will continually travel between that building and the town hall, stopping only if the assigned building does not contain enough resources for the villager to pick up a full load.

Buildings can also be commanded. If a building is selected, it can be activated or deactivated. In the case of a resource building, deactivating it will prevent it from accumulating resources, while reactivating it will resume resource generation. In the case of a military building, deactivating it will prevent the production of units, even if enough resources exist to do so, and reactivating it will resume unit production. This is useful for prioritizing one unit type over another, say if most of your archers have been killed and you don't want to spend food on soldiers or cavalry as a result.

Finally, players may also activate their god's active power by clicking on the power button. This is a powerful, game-changing ability that can be used to turn the tide of battle at the cost of faith.

ii) Civilian and Military units

Each unit in *Apple of Discord* has five stats: vision, range, aggro, health, and attack. Vision is the radius in which the unit can see enemy units to engage them in combat. A unit will never attack an enemy outside their vision range. Currently, all military units have a vision range of 4. Range is how far away the unit can attack, with 1 being adjacent only and 0 being no attack. Aggro controls how far a unit will chase an enemy before giving up if set to "Aggressive". This is currently 6 for all military units. Health is the unit's health. It is depleted when they get hit, and when it reaches 0 they die. Attack is how much a military unit depletes a targets health when they attack that target.

The first type of military unit is the soldier. Soldiers have more HP than the other units, making them able to take more of a beating and to protect other units behind them. They also cost fewer total resources than other units. They serve well as front-line defenders or as shock troops. Their starting health is 7, their range is 1, and their attack power is 2. Pictured below is a sprite of one of Zeus's soldiers:

The second military unit type is cavalry. Cavalry units move faster than other units, making them function well for guerilla tactics, and are good at taking out archers before accumulating too much damage. Cost is their greatest drawback; these units cost more resources than any other unit. Their starting health is 5, their range is 1, and their attack power is 2. Pictured below is one of Poseidon's Cavalry units:

The third and final military unit type is archers. Archers have less attack power, but can attack at long range, making them good supporting units, especially in large numbers. Their starting health is 5, their range is 2, and their attack power is 1. Note that archers are at a SEVERE disadvantage without support, doing only half as much damage as other units. Even a fight between a soldier and an archer in melee may easily end with the archer's death, despite the fact that they deal double damage to soldiers. Pictured below is one of Artemis' archers:

Military units are arranged in a sort of "rock-paper-scissors" fashion, where archers deal double damage to soldiers, soldiers deal double damage to cavalry, and cavalry deal double damage to archers. This adds an element of having to out-think your opponent in warfare, such as building cavalry to fight against an opponent's archers, or whether they will expect that and send soldiers instead. While this might, on the surface, just seem like an overanalyzed game of rock paper scissors, there is much more information for the players to work with, and it mostly all comes down to how well you can model your opponent's thought patterns.

The only type of civilian unit is the villagers. Villagers are necessary for collecting resources, and constructing buildings. Having enough food to spawn more villagers in the early game will help accelerate each player's growth.

To a certain degree, buildings are also units. While the buildings themselves cannot attack, they do have HP that can be depleted in order to destroy them.

Additionally, there is the God unit. Gods have 1000 HP (200 times as much as the average human). They cannot attack directly, but act as the source for god abilities. Gods are the only units in the game that can respawn, which they can do if defeated by spending faith. A god's passive ability remains in effect even when they are not on the battlefield, but their active ability cannot be used.

iii) God Abilities, Faith points

Gods have two types of abilities: an active god ability, and a passive god ability. Passive abilities are an overall buff which improve some aspect of the god's army or buildings. Active abilities are powerful attacks that can only rarely be used. Each unit a god controls produces some amount of faith points, which go towards filling up the god's faith bar. When the faith bar fills up enough, the god may use their active god ability. The active abilities belonging to the four gods currently in the game are:

Zeus: Summons a storm overhead that hits random enemy units and buildings with lightning for significant damage.

Hera: Summon a swarm of gadflies which remains on the field for a while and deals damage to any enemy units in affected tiles.

Artemis: Fires an arrow across the entire map, hitting and damaging any units or buildings in the way.

Poseidon: Summons a tidal wave in front of him, damaging and pushing back any units struck.

The passive abilities, on the other hand, are not affected by faith. Instead, each god has a static passive that remains in effect through the entire game. They are:

Zeus: Units produce a greater amount of faith per second.

Hera: Villager units have a larger carrying capacity for resources.

Artemis: Archers deal more damage and shoot further

Poseidon: Cavalry move and attack faster

As a result of these abilities, each god has different strategies that they can use. Zeus will have the weakest army, but is somewhat more able to spam his god ability. He might want to attack enemy armies on his own, or with minimal support, while a large number of villagers stay behind to pray to him. Hera will have weak, but plentiful units, due to her improved resource gathering. Her troops are more expendable, allowing her to rush in wave after wave of units, and her god ability allows her to even the playing field a bit. Artemis is good at taking out enemies before they can reach her. She will want plenty of archers, along with soldiers to defend and support them. She might also possibly want to focus on using her god ability to take out stables and cavalry, due to their increased damage against archers. Poseidon will want to focus on guerilla tactics, using his fast moving cavalry to soften up enemies to be finished off by his god ability or archers. Of course, these aren't the only strategies they can use, and players can feel free to experiment.

Part 3) Art Style and Pipeline

1. Art Style

The mood of our art is somewhat more serious and realistic. Our color choices attempt to emulate that of other games in the MOBA genre - darker, with higher saturation. The colors, especially for characters, make them eye-catching and easy to differentiate. We gave each god a unique color or colors, and used the same colors for their outfit, UI, and military and civilian units, to help create a unified look for each god.

A variety of programs were used for creating all art assets in the game. Pixologic's ZBrush[7] was used for sculpting and texturing character models. It also aided in the creation of a few environmental assets, such as the mine, trees, and rocks. Similar to Pixologic, Autodesk's Maya[8] was used for box modeling buildings. Maya can be an efficient way to create low-poly objects. Once modeled, characters and buildings were thrown into Autodesk's 3DSMax[9]. Like Maya, Max can be used for modeling, but we used it as our main resource for rigging and animating characters. We also used Max for rendering purposes as it has a superior lighting and camera rig system. Adobe Photoshop[10] is a tool of all trades that can be used for drawing, painting, 2D animating, etc. For our purposes, we used Photoshop to create textures that could be applied to buildings, terrain tiles, etc. Similarly, Adobe After Effects[11] was used to enhance game visuals. GameMaker[12] is a free engine mainly used with 2D art. We utilized GameMaker's sprite packer to turn our renders into spritesheets.

2. Character Design Pipeline

Our original pipeline consisted of hand-drawing characters and environments in Photoshop. At the encouragement of our art advisor Ralph Sutter, we changed this to a more complicated, but ultimately very time-saving approach. With our original plan, making new animations for different attacks or characters would take a fixed amount of time each. Each animation would take a while, which would then be multiplied by however many were needed, and also by however many gods we included. This adds up to a lot of time.

The character design process that we used instead leveraged both 3D and 2D programs, by first modelling 3D characters, but rendering them out and editing them as 2D sprites. This makes a greater start-up cost to model and rig each character, but once that bottleneck was complete, the time for each

[7] Pixologic http://pixologic.com/
[8] Maya http://www.autodesk.com/products/autodesk-maya/overview
[9] 3DS Max http://www.autodesk.com/products/autodesk-3ds-max/overview
[10] Photoshop http://www.photoshop.com/
[11] After Effects http://www.adobe.com/products/aftereffects.html
[12] GameMaker https://www.yoyogames.com/studio

new animation or character was dramatically lower, and more open to additional animation needs that we might not have foreseen.

First, Bryce sculpted, painted textures on, and remeshed the characters in ZBrush. The remeshing process was easier than it would be for a 3D game, since they only needed to be low-poly enough for 3DSMax, not for actual in-game use. With the finished models imported into Max, Bryce then rigged them using Max's built-in Biped system.

Bipeds were incredibly convenient for a project like this, due to the extreme ease with which animations can be transferred between two bipeds of identical structure. After rigging, Bryce animated one character, and copied the animation to the other three characters. Finally, four cameras were set up, and four different lighting arrangements were made for rendering the animations out as sprites. Four different sets of lights were necessary, otherwise the god's shadow would move around as they changed direction, which would be disorienting for players. The shadows do shift when the player rotates camera views, but this is more excusable since the player's attention will be on the whole board rotating, and likely not on a minor detail like the shadow.

The image sequences rendered from Max were compiled into spritesheets in GameMaker, then imported into Unity. Bolin made a simple script that would play out the animation on a plane. After Effects was used to create particle effects for each of the gods' abilities to add extra polish to the game's visuals. In comparable MOBA and real-time strategy games, abilities are usually very iconic for each character, so missing this step would be a serious missed opportunity to make memorable characters.

Making the civilian and military units was much easier, as a number of shortcuts could be employed. These units were much smaller, so could also be lower poly. They would also be far too small for texture maps to offer much actual detail, so they instead had single colors applied as materials in 3DSMax. This was actually an effective way of visually conveying important information about units - each unit was assigned colors corresponding to their god. Even at a small scale, the color and type of unit still reads perfectly clearly to the player. Thus there should be little confusion even when large groups of units converge to fight each other.

Up close, there are also rather large flaws in the unit characters' rigs. The arms of the melee and cavalry units actually detach from their body. However, at the distance players will be seeing them, these poor deformations are hardly visible; their animation is strong enough that what players see is the motion and action, not the rigging.

Lastly, Bryce made splash art for the characters to be used during character selection. This was primarily achieved with ZBrush renders composited and then painted over in Photoshop. Characterization could be stressed more in these special renders than in the models themselves. The splash art can give each character a unique personality that can appeal to different players. Zeus's expression was made to look more refined and austere, Hera's face is in profile and looks out the corner

of her eye suspiciously, Poseidon smiles comfortably and Artemis's piercing eyes stare forward. This splash art can be viewed in Part 2, Section 3, Item ii.

3. Building Design Pipeline

Unlike the difficult process of character design, building implementation was much easier. Learning from the trials of character design, Suzanne decided to go the 3D to 2D route from the get go. Different from the character models, buildings didn't have to be rigged and animated, so they could be as high or low poly as we pleased. For the basic buildings, Suzanne first modeled the mesh in Maya, then uv-mapped the objects, adding texture details in Photoshop. They tended to have a much lower polygon count because she could add intricate details and designs more easily in Photoshop. For example, some of the buildings had different colored borders, which was implemented post-build. Wall planes did not have to be divided into two to accommodate uv mapping for borders. For more intricate designs, such as the mine, Suzanne modeled and textured the mesh in ZBrush. They tended to have a much higher polygon count because she wanted to create higher quality bump maps. For instance, when creating objects made out of things such as stone, it can be difficult to make a mesh that looks realistic. By having higher poly models with enhanced maps and textures, Suzanne was able to achieve more believability in her models.

After creating objects in Maya and ZBrush, she then imported them into 3DS Max, using a similar camera and lighting system as the character models. We wanted each sprite to look uniform and of a similar environment, so using the same setup was important. Even though each building has four different views, Suzanne was able to achieve the same look using a one-camera rig. She centered the each object's pivot point and rotated it, taking renders of the buildings every ninety degrees to obtain isometric views. Because the lighting setup was the same, everything was rendered as a png, which allowed the projection of the building's shadow onto a transparent background. Usually, the sprites would then be further edited in After Effects to create different color schemes, but we wanted all buildings to be uniform in nature. To differentiate between a building's owner, we used a flag or banner system. Suzanne threw all of the renders into Photoshop, then carefully hand placed a singular flag onto every building. She created four different colored flags, one for each god that was based on their chosen color scheme- yellow for Zeus, green for Hera, red for Artemis, and blue for Poseidon. When these sprites are used versus the normal sprites, they will be used to indicate which building belongs to each player.

Because all of the buildings are static models, the creation process was somewhat less difficult than character creation, though the process was the same. Like the character models, it was most convenient to model them first in Maya versus hand drawing them as sprites in Photoshop. By taking the time to create 3D models, Suzanne could go back and fix things that may have turned out strange in renders or make slight changes to the models when they may have looked too similar to each other.

4. UI

The user interface was one of the last important areas of art to be implemented in the game, and because of this is not as highly detailed as the characters and buildings. In order to keep the attention more on the gameplay, Suzanne decided to keep the UI fairly standard, but drew references from Greek culture. For decorative elements, Suzanne created Doric, Ionic, and Corinthian pillars that were used to separate the different portions of the UI. When it came to border designs, she drew from classical Greek elements, creating simplified versions in black and white. The most complex of the UI would be the four buttons used to control military units. These are more standardized, though, than based on Greek culture. All assets were made in Photoshop.

5. Unused Assets

In the end, a great deal of our original artwork was not implemented in the final product due to time constraints. We wanted to create obstacles to hinder player movement, such as mountains, but these were unable to be implemented. Instead, rocks that were to be originally used as obstacles replaced ore assets. The ore mountains ended up being scrapped. The idea of using rocks for ore instead of the ore mountains turned out to be a good game design choice because it was hard to tell the difference between ore mountains and mines. Two building-related assets that were unused were the city walls and outpost tower. Our original plan was to have players be able to destroy their opponents' city walls and gain access to their main hall, but this was quickly scrapped. In terms of the gods, a character model was created for Hermes, but was unused due to balancing issues and rigging/animating time. Art wise, Bryce and Suzanne tried to take on more than they could chew and were forced to cut down to roughly a fourth of original idea assets. With better planning and more time, these could have been implemented in the final game.

We also spent a good deal of time in A term creating character concepts for the gods who never made it into the game. One example is Aphrodite, who would have been a non-binary gendered character. Since Aphrodite is the goddess of love and beauty, this would have been a powerful stance to take, showing that non-binary people can be icons of beauty and grace as well.

Athena was another subject of our early effort. Our references gave lengthy lists of all the different disciplines Athena represented – diplomacy, war, art, and many others. This reminded Bryce of a quote by Nicki Minaj about how women, particularly those in media, are expected to balance a number of different attitudes and maintain their appearance through all of it. Athena's aesthetic would have been inspired by Nicki, as well as other important black female celebrities who are often given a negative reputation by media in spite of their positive qualities and Jack-of-All-Trades skillsets.

Hades, usually portrayed as male, would also have been female, with a look inspired by actress and model Rinko Kikuchi. The choice of gender in this case was also intentional, as a means of flipping the myth of Hades and Persephone into a version more empowering of women. In the common myth, Hades tricks Persephone into entering the underworld and she becomes his bride. This portrays her with

very little agency, and like many other mythological stories (that of Adam and Eve for example) a woman is blamed for bringing some evil into the world. In some feminist revisions of the myth, Persephone, merciful to the dead sent to the underworld, ventured there of her own will. In this version Persephone becomes the goddess of the underworld. This also makes it more in line with other cultural myths about fertility deities, who far more often are women. The symbolism of death sustaining life would have been very rich, and we regret not pursuing this character concept further.

Part 4: Technology

1. Networking

Bolin's first attempt at networking involved using sockets. As this was a multiplayer game, this method provided a number of advantages, namely that by connecting to the internet, players can join games with others from around the world. This is a standard method for online multiplayer games, such as *League of Legends*. In this case, we would also require a server, which would need to be in operation at all times to keep the game running.

This idea however proved difficult to implement in the time we had. It was more important to have networking functional at all than to have it exactly the way we wanted it. We opted for a LAN-based network instead. One reason LAN is effective is that the only players of our game would be ourselves, our professors, any playtesters and players at Showfest. Since that audience doesn't extend outside of WPI, we didn't need online multiplayer, as everyone would be playing on a WPI network anyway. We also realized this was just a network for a game, and its only purpose would be to allow players to play together, so it didn't need to be more complicated than we could have handled.

The LAN system uses Remote Procedural Calls (RPC) which is a way of sending messages between users by allowing clients to send information to the server which is then relayed to all other users. It works by first creating a server on one of the machines, then having any users act as clients who send and receive information from the server.

While the LAN was easier to set up, Bolin still encountered a number of problems. The first problem arose when trying to track which player sent which action to the server. The server could be given a message, but not know who sent it, and thus would be unable to accurately update game data. For instance if a player tried to select and move a unit, their client would tell the server to select that unit, but the server wouldn't know whether that was an allowable action (i.e. the player attempted to move an opponent's units).

The solution was to generate player ID numbers that would be included in any messages sent so that the server could know which client corresponded to which in-game player. Up to 4 players can send a random number to the server, which assigns them a player number (from 1 to 4) so that it can keep track of which client is in control of which player. There is a very small chance that two clients will generate the same random number to send to the server, and thus would be assigned the same player number. In this event, the server will void the previous assignments and try again.

Once the correct assignments have been made, players always attach their ID number along with any information they send to the server so that the server can authenticate who sent the information and accurately adjust the game information for all players.

Synchronization is arguably the most difficult and important part of game networking, since all players need to be playing with the same observable data. The first issue with synchronization is keeping track of not only which player is sending information, but what units and buildings belong to each player, so that no illegal actions are made. The solution was similar to the previous problem, only a longer ID number is given to each unit. Each time a new unit is spawned, it is given an ID number that contains information about which player it belongs to, and other details about the unit, so that it can be updated whenever affected by any of the players.

A major issue with most networked multiplayer games is latency between different clients. The game server needs to know when to update information so that all players are provided with adequate knowledge about what is happening. If the updates happen irregularly, it will be unfair to some players. For our game, this is hardly an issue, since it uses LAN. There won't be too much lag, and sending updates to all clients ought to not cause any serious problems.

2. Unit Behavior

When not following orders set by a player, units attempt to perform automatic actions when presented with certain criteria. This simple artificial intelligence helps take the burden of micromanagement off the players' shoulders, allowing them to leave units unattended while focusing their attention on another part of the battlefield. The actions of the units are determined by each unit's current stance: aggressive, defensive, or stand-ground. A unit's stance is changed by selecting the unit and selecting the desired stance from the unit's panel in the user interface. However, stances only affect military units and gods; villagers need only path between buildings when not following players' orders.

The default stance of newly created units is aggressive. While in this stance, the unit will scan for enemies within X tiles of its current position. When an enemy enters this aggression range, the unit will move to attack range with the enemy. The unit will only stop attacking the enemy if the enemy dies, the enemy moves out of the aggression range, or another enemy is closer. While in aggressive stance, a unit could theoretically be led across the entire play area while chasing an enemy, so players do need to be mindful of wandering units if they choose to leave their units in this stance. Aggressive stance is best for large engagements, when players want their soldiers to engage enemies at will and chase down enemies fleeing from the battle.

Defensive stance is similar to aggressive stance. Again, units in defensive stance scan for enemies within their aggression range, and they move to engage enemies given the same behavior as aggressive. However, units in defensive stance will not move beyond X tiles from their starting position (defined as the last spot the player ordered them to move to) and will not attempt to engage enemies outside of this range. Additionally, if a defensive unit has moved from its starting position but now has no enemies to engage, it moves back to its starting position. Defensive stance is a good all-around stance that combines the autonomy of aggressive stance with the hands-free management of stand-ground stance.

Stand-ground stance is quite different from the other two stances. As its name implies, units in this stance will not move from their current position unless directly ordered to by the player. They will attack any enemies that come within their attack radius, and ignore all other units. Stand-ground stance is best used for units guarding a choke-point, such as the entrance to a city, and is most effective with archers, which have a larger attack range than the other two unit types.

A fourth stance, flee, was originally planned for the game, but was removed due to time constraints. In the flee stance, units would attempt to escape from nearby enemies. It was intended to be the only available stance for villagers, which have no combat abilities, as well as for wounded units that the player wished to keep safe.

Below is a screenshot of the unit behavior icons and user interface:

The two columns on the left and right act as dividers between the different UI screens, the large square would show the unit's portrait image, the bar shows the units health, which is also shown at right numerically along with unit name and stance information. Along the bottom are the five icons for stances – aggressive, defensive, stand ground and disband

3. Pathfinding

Units in Apple of Discord utilize A* pathfinding to get from one point to another reference. We decided to use A* due to the fact that the game itself was already grid based. As such, A* was much easier to implement than most other methods would have been. Four classes are involved with A*, those being "Actor", "Pathline", "create_grid", and "Tile_Generic".

When a unit (hereafter referred to as an "Actor") wishes to move to a point, it first references the "Move" function in the Actor class. The move function takes two arguments, those being the x and z coordinate of the destination (the y coordinate is never changed). The move function then creates a pathline with that direction. This pathline acts as the head of the path determined by A*, and has a "start point" and "end point" which are both the same, and are both the same as the current position of the Actor. At this point, the Actor instructs the pathline to construct a path by calling the functions "signal" and "extend" until either signal returns false (indicating that no path can be found) or extend returns true (indicating that a path has been found). Then, the Move function stores the head of the

path. As long as the head is not null, calling the move function will cause the Actor to move one square along the path instead of causing it to create a new path. If the Actor is unable to move, it will attempt to do so a second time. If it is still unable to move and the obstruction is another Actor controlled by the same player, the two actors will switch places. If that measure also fails, the move will simply be cancelled.

The Pathline class is a class solely created to implement A*. Unlike many unity scripts, it is a ScriptableObject instead of a MonoBehaviour. This allows pathlines to be created and destroyed without attaching them to an object. A pathline represents a line connecting two different tiles, represented by two coordinate pairs (xstart and zstart, and xend and zend). When "signal" is called, the pathline checks whether there are any squares adjacent to it that are open and passable. If so, it returns true. If not, it checks whether it has any pathlines that it has created. If so, it returns the result of calling signal on that pathline. If not, it returns false. In simpler terms, the pathline checks over the path from its start point onward to see if it's possible to extend it. Furthermore, signal also sets several static variables which, as a whole, tell the path which pathline to extend from. The Pathline class also contains the function "extend", which, along with signal, is one of the two most important functions it contains. When called on any pathline at all, extend will result in the path extending from the point selected by signal. Extend will return true if the destination has been reached, and false otherwise. Extend will also call finalize if the path is completed. Finalize is a cleanup function. It first marks all pathlines on the path to the destination as "final", so the actor knows which path to follow. Then it opens all grid squares closed by the pathfind. If the destination tile is impassible, the pathfind is considered successful if it reaches any adjacent square instead. This allows units to pathfind to buildings or other impassible tiles.

Create_grid is the class that holds the grid as a whole. It contains the function "GetPass" which, given an x and z coordinate, returns whether or not the tile there can be passed through or not. If also contains the grid "closed", which is used to mark spaces on the map as closed during a pathfind (all spaces are open by default). Being able to mark spaces as closed in A* is very important in order to prevent a pathfind from taking infinite time and memory to complete.

Finally, Tile_Generic is the class that comprises each individual tile. A tile can be passable or impassable, and occupied or unoccupied. Occupied tiles return "occupied" when asked about their passability, but no tile has occupied as its default passability. Like create_grid, Tile_Generic is only tangentially relevant to the pathfinding code, but still deserves mention if only because the status of a tile as passable or impassable is extremely relevant during an A* search.

Overall, this pathfinding method is one that works, but that is, to a degree, severely overcomplicated. Other, better methods of implementing A* definitely exist, and the first step in developing pathfinding should have been to research in more detail how others had implemented A* in unity games beforehand.

Part 5: Conclusion

Overall, we succeeded in developing a multiplayer game based around four Greek gods that accurately reflects the style and gameplay as our influences, *Sid Meier's Civilization, League of Legends, and Final Fantasy Tactics*. Each player is able to control a unique character and their followers, whose own strengths and weaknesses can lead to interesting strategies. There is an ultimate end goal that can only be accomplished with wit and cunning on the player's part.

Although we developed a functional game, *Apple of Discord* did not come close scope-wise to our original plans. Over the summer and during our first term, we were most concerned with trying to agree on a game we would all have liked to make. We spent too much time building up ideas for the game, without scheduling specifically how and when we were going to implement those ideas. A lot of the time we just figured we'd have the time to do it in a later term. As we approached our third term, we realized there was no more time; the game had to be completed and the document written. We had to cut a large number of features that weren't well-developed, and stick with a more manageable game. One of the most noticeable features is the fact that nowhere in the game play are references made to Eris or her Apple of Discord. There is no backstory to the game; it comes off as a "defeat and conquer just because we can." Additionally, the game is completely silent. We gave very little thought to game audio, and by the time the game was coming to a close, it was really too late to give it the attention it deserved.

In attempting to make the game we all wanted to make, we failed to make a game that could be developed in the projected time frame. It would likely have been easier to compromise on our game ideas and aim for something smaller with higher quality and polish.

While as a game project *Apple of Discord* didn't turn out as expected, we learned many invaluable lessons. While most IMGD majors work on a game project for their MQP, it is - like other majors - primarily an educational project, and our group has learned a great deal about game development and production.

The process of making four main playable characters, making them low-poly enough for use in game, rigging them using a standard system, animating their main actions and rendering them as sprites, Bryce greatly expanded his knowledge of the entire character design process for 3D game characters.

In creating and debugging several particularly finicky bits of code, Ben learned to plan ahead by creating solid foundational code instead of rushing through it, and to be willing to step back and check if there's a simpler solution instead of trying to force an over-complicated solution to work.

Although she thought she had learned this lesson before, Suzanne was once again reminded of the dangers of procrastination. She is however merely human, prone to make this mistake again. Overall though, Suzanne learned about the difficult process of art asset creation, mainly the 3D to 2D pipeline.

She feels it will benefit her on future game development endeavors and eventually translate to working in the industry.

Bolin was our lead networking programmer, and as such he learned a great deal about the main problems with most networked games, and solutions for many of those problems. His painstaking effort clearly shows in the quality of the networked content. He also made many of the GUI elements and the code implementing them, and knows much more about making GUIs for games than before this project began.

Robert largely worked on the autonomous behaviors of units and buildings, and helped Bolin with initial GUI designs and Ben with pathfinding behavior. He also functioned as the main liaison between the art and tech subteams, and organized and directed his fellow programmers. This led him to a greater understanding of team management, as well as practical programming skills including various pathfinding algorithms and state machines.

Part 6: Works Cited

Books:

D'Aulaire, Indri and Edgar Parin d'Aulaire. *D'Aulaire's Book of Greek Myths.* New York: Delacorte Press, 1962.

Games:

Age of Empires http://ageofempiresonline.com/en/
Final Fantasy Tactics http://www.final-fantasy-tactics.com/
League of Legends http://na.leagueoflegends.com/
Sid Meier's Civilization http://www.civilization.com/

Engines/Software:

Adobe After Effects http://www.adobe.com/products/aftereffects.html
Adobe Photoshop http://www.photoshop.com/
Autodesk 3DS Max http://www.autodesk.com/products/autodesk-3ds-max/overview
Autodesk Maya http://www.autodesk.com/products/autodesk-maya/overview
Pixologic Zbrush http://pixologic.com/
Unity Technologies Unity3D http://unity3d.com/
Yoyo Games GameMaker https://www.yoyogames.com/studio

Part 7: Art Appendix

3DMZ_Art_Assets.zip