# Frontline General

*Final Report*


3RTS Studios®

McLain Johnson

Christopher Olson

Diego Ruvalcaba

Jingdi Zhang

Department of Computer Science and Engineering

Texas A&M University


29 April 2020

# Table of Contents

## 1. Executive Summary

In this project we set out to address an issue in real-time strategy (RTS) games today. We identified that RTS titles contain many moments of gameplay that consist of the player waiting for their units to carry out the commands they issued. To keep the player engaged throughout the course of the game, we introduced a gameplay mechanic that allows the player to switch between a Commander Mode (RTS gameplay) and Unit Mode (3rd person action gameplay). In addition to implementing this switching mechanic in our game, we created a commander AI for the player to play against. This AI has been trained using a deep reinforcement learning algorithm that incorporates the switching mechanic for its own units. The end result is an RTS/3rd Person Action game that is engaging and provides a replayability factor due to the decision making of the AI and novelty of gameplay.

## 2. Introduction

### 2.1. Problem Background

Every genre has its own shortcomings in gameplay and how engaging the game can be for a player. A common shortcoming for many genres is not utilizing the core mechanic effectively and this leads to moments of downtime for the player. We ascertained that the core mechanic of both real-time strategy (RTS) and the 3rd person action game is enemy engagement. We have identified key problems for RTS and the 3rd person action genres where the games are not utilizing the core mechanic.

The issue with real-time strategy games today is after commanding troops the player then has to wait to see how the combat plays out. These moments don't require any strategy and could be considered as less engaging for the player. There are ways RTS games have tried to engage the player during this time, with features such as resource management, however, this is not considered the core mechanic of the game.

The issues we had identified with 3rd person action games were that there is no large-scale planning or strategy. There are some existing titles that attempt to involve strategic elements in the gameplay, such as Rainbow Six Siege. However, we consider this to only consist of small scale tactical decisions between individual players. Engagements in these types of games typically last for a few moments after which the player moves on to the next engagement.

### 2.2. Needs Statement

There is a need to provide players continual engagement throughout their gameplay session.

Currently, RTS and 3rd person action games do not address the issue of continual engagement with the player using the core mechanic throughout the gameplay. There is a lack of game titles that try to address these issues by either thoroughly providing more engaging RTS gameplay or provide more strategic elements in 3rd person action games. This has created an unfulfilled niche in the game industry, as many players do not enjoy either genre for a lack of these elements.

### 2.3. Goal and Objectives

Our goal was to create a game that is able to immerse the player by continually engaging them with the core mechanic throughout the gameplay. We sought out to address the problems with

the lack of consistent player engagement in RTS titles and the absence of strategy in 3rd person action games.

We accomplished this by creating a hybrid-genre game that provides consistent player combat engagement throughout the gameplay as well as the large-scale strategic elements found in the RTS genre. Our game allows the player to alternate between large-scale strategic gameplay and small scale skirmishes. By doing this we give the player the opportunity to switch to a more engaging 3rd person action experience during moments of less critical RTS gameplay. Conversely, we enable the player to switch to a large-scale strategic style of gameplay in between third-person engagements.

### 2.4. Design constraints and Feasibility

Our primary constraints are development time and the particular skills of each team member. We used 11 weeks to develop our solution, and games, in general, take a long time to build with experienced game developers. There are many skills needed to create a video game such as programming, design, art, and production. Our team, however, is not as experienced so time becomes even more of a constraint. That being said, we are hoping to create a game that is still engaging to the player without having to be as polished as most games.

Our technical constraints are primarily based on the CPU performance required by a large number of agents in an RTS system. Additionally, the graphical fidelity is limited by the ability of the graphics card of the system. Our target minimum system build is an Intel i5 processor with an Nvidia 1050 Ti. We believe it is feasible to implement our game under these technical constraints based on RTS games that utilize a similar gameplay mechanic.

### 2.5. Literature and technical survey

Natural Selection 2 (2012)

Natural Selection 2 is a fast-paced multiplayer shooter that combines elements of RTS with a first-person shooter. This game is a multiplayer game where one player is the commander and is able to see the world from a top-down perspective and the other players are infantry from a first-person perspective. This is a game that does not have the same game mechanic to switch between roles but it provides a lot of the basis for the gameplay style for the hybrid genre we have tried to create. In Natural Selection, the players in first-person must go around the map collecting resource nodes that give the commander resources. The commander can then use these resources to give his players an advantage by increasing the stats for each player. This is a very useful dependency that actually gives both the commander and the unit players reason to want to cooperate. Although we had to implement a switching game mechanic, we did learn from this dependency between roles to incentivize our players to switch between roles. Something that this game lacks, however, is the balance between the human and the alien teams. Since both teams have different play styles, it took a long time for the developers to create a game that felt balanced every game. We have circumvented this issue by creating the same types of units for each team even if they might have a different sprite to represent them.

Dragon Age: Inquisition (2014)

Dragon Age: Inquisition is a strategy role-playing game where you command a group of characters into battle in a very fantastical domain. It provides the player with a unique gameplay

style where you can switch between commanding your squad tactically to try and defeat the enemy and playing as one of the squad to give your team the advantage. This relates to our goal of a 3rd person action game that also incorporates themes of strategy. That being said, it doesn't give the player any ability to wage large-scale attacks, since the max size for a squad is four, and therefore lacks the large-scale strategy that is a staple for many RTS games. It makes up for this lack of engagement by providing a very throughout storyline and world to engage the player in the immersive world. This compares to our game only in the sense of the mechanic of the game that allows the player to switch roles from skirmish type battles into small scale strategic battles. Although the story is compelling for the player, we want to keep the arena style of gameplay that is common to most RTS games where there are two or more players facing head to head. While developing, we believe will provide more action in each gameplay whereas playing Dragon Age, most of the game is spent exploring and leveling up your characters in the large world. We believe that to build off their idea of switching perspectives, in our game we had to put this into an RTS game and ensure that the player is engaged throughout.

Warshift (2016)

Warshift is another hybrid genre that tries to combine RTS with RPG gameplay. In this science fiction-themed game you can choose to be in commander mode where you can build your base, command and upgrade troops or you can come down as your main character and be a part of the battle. This game utilizes our same game mechanic but limits the switch to your 'main' character in the game which you can upgrade and customize to your preference. This differs from our game where you can switch perspective by taking control of any player on your battlefield from your team. Although this concept is interesting, it might lead the player into greedily upgrading their main character to dominate the map and only playing in the first-person mode. Conversely, another player could choose to just dedicate as many resources to their troops and solely play from an RTS commander perspective. With our game mechanic, players that take control of a unit get a boost to damage. This incentives players to switch between modes and use them effectively to gain the advantage in any small battle, but then switch back to move units around so they can be better positioned and ready for the next battle as extra damage does not help the full battle at large.

Battle Zone: Combat Commander (2018)

This sequel to Battle Zone is a blend between FPS and RTS set in a futuristic science fiction environment. This game actually implements the hybrid genre by allowing the player to control all of the RTS aspects of the game from the first-person perspective. Players command ships that they choose and then fight in battles on the map. They can then return to base to send out new vehicles, build, gather resources, and command their army of ships. Although this is an interesting take on the hybrid genre, it really does not allow the player to have any scope of the large-scale strategy that they are trying to achieve. Only being able to see what is around you in first-person puts you at a disadvantage as a commander of your troops since you do not have a sense of the whole map. By switching the perspective as we have done in our game, it allows the player to switch between roles and either prioritize their large-scale strategy or smaller critical battles where they want the advantage. In this game, there are only vehicles and this limits really what can be done in the first-person perspective. It effectively creates only ranged weapon types that may all have the same playstyle. In our game, we have close quarters melee units that

increase the strategy of the game. By not choosing vehicles, and having melee units, the player knows that he must have the units in close proximity to the enemy in order to defeat them instead of getting picked off from a distance by enemy vehicles with range.

Eximius: Seize the Frontline (2018)

Eximius: Seize the Frontline is a newer game that is similar to the style of game we have implemented. Eximius is a first-person shooter and RTS hybrid game that focuses on squad-based combat. The games feature a 5v5 multiplayer experience where each team is composed of 4 squad officers and a commander. This game features our core mechanic the closest and allows the player to switch between a commanding mode where they can build defenses, resources, and create and command troops and a first-person shooter mode where they become one of their squad members to give an advantage. This game takes place in a more modern military setting and gives the player the opportunity to play in small scale skirmishes or strategize their offense from a large scale. We have built on their ideas of a hybrid RTS/FPS game in certain aspects. First, the switching between perspectives can be very disorienting as they use a lower resolution rendering of the map in commander mode and a very high resolution rendering in the first-person mode. This can make players want to spend more time in the FPS mode as it feels better but actually most of the time is spent in commander mode. We hope to improve on this by using the same map for both modes and only changing the camera view to switch between them. This rendering issue, however, leads to the next issue which is that although it has created a hybrid genre, it really does not fulfill the needs we addressed which is player engagement. Most of the game is actually spent just building, assigning jobs, and doing resource allocation as opposed to battles where the switching mechanic aspect would be most useful. The final issue that we have improved on from Eximius is that it talks a lot about its multiplayer abilities but fails to have a strong fanbase to supply new gamers with players to play against. We have improved this by creating a very intuitive AI so that they can play anytime regardless of how many people are online.

### 2.6. Evaluation of alternative solutions

Co-op play

In the game Natural Selection 2, there is a commander role and then the units role. Each player gets assigned a role for the whole game without switching and there is only one commander on each team. The commander on each team gets a top-down view of the world whereas the unit players on his team interact in first-person. The commander is in charge of the RTS aspect of the game and relies on the other players on his team to roam around the map building resource bases. If a unit on the same team builds a resource base then the commander can use these resources to train and upgrade his troops. This co-op play is a very interesting game mechanic but it lacks the ability to switch between roles. Since we are trying to keep player engagement, the commander has no opportunity to be engaged once he has trained his teammates and is waiting on more resources to make the next step. Without the ability to switch the first-person units will not be able to incorporate any large-scale strategy into their game play and can only focus on small-scale defence of their resource bases and attacking enemy resource bases. Likewise, the commander will not be able to engage in any high speed action in the game and will only be handling resource management. Where the games core mechanic is unable to satisfy our needs by engaging the player throughout it actually does implement a useful feature. The dependency

that ties both the unit player and the commander player is resource management. This dependency is what forces the units and commander to have to work together to achieve a common goal. Since we've identified that a problem of our game mechanic is going to be how useful each role is and how we can incorporate the switching between roles into the player's strategy, I think it is important to note this dependency as it could help incentivize our player to switch between roles more intentionally.

Small-Scale RTS + FPS hybrid

This hybrid genre is most similar to Dragon Age: Inquisition which is a strategy game where you can command the 4 members of your squad. This game also contains our core mechanic which is after being able to command your 4 members you can choose to play as one of them to give your team the advantage. This being said, it really lacks the sense of strategy that is iconic to most RTS titles. In its core, the strategy used in Dragon Age is more about exploring the map and story of the game to upgrade your squad as opposed to the standard strategy that is required for RTS games such as resource management and squad placement. Where Dragon Age places its importance in its story and immersive world we want to focus on our strategy and also our arena playstyle where players face head-to-head against an opponent in one game. We learned from this game however that we do not want to have a small-scale RTS as it limits the possibilities for switching to be meaningful by not giving the player enough opportunities to utilize the core game mechanic. We also were able to take away how we want to implement the camera change between roles as Dragon Age does this quite effectively without disorienting the player.

Large scale RTS + main character control

In the game Warshift a player controls their vehicle army as an RTS commander and also has the ability to switch between this role and a main character that they play as. While in commander mode the player can command and upgrade their forces as well as upgrade their main character. Warshift has the same game mechanic as ours as it also seeks to engage the player during slower moments of RTS gameplay. That being said it is different from our game mechanic as you can only switch to your main character as opposed to any unit on your team. There is however a shortcoming for having the ability to switch to your main character. You could theoretically just choose to upgrade your main character and only play in that mode. This would remove the strategy from the game that we are trying to create. You could also never unitize your main character and just choose to upgrade your troops and work on your strategy. Therefore we do not think this game mechanic will incentivise our player enough to switch between modes.

RTS + FPS (FPS perspective)

This game mode is the ability to play an RTS style game but from the perspective of a first-person shooter. This game style is most similar to Battle Zone and allows the player to control a unit and control and command troops when they are at their base. This is an interesting perspective for an RTS game that tries to make RTS games seem more engaging. That being said however, being in first-person perspective puts the commander at a major disadvantage as they cannot even see the whole map from top-down. Without being able to see your forces that you are commanding, it is easy to argue that you focus less on your large-scale strategy and focus more on how your character will shift the tide of the battle. We learned from this game that

changing the perspective is crucial to allow the player to fully experience the strategy aspects of the RTS side of the game.

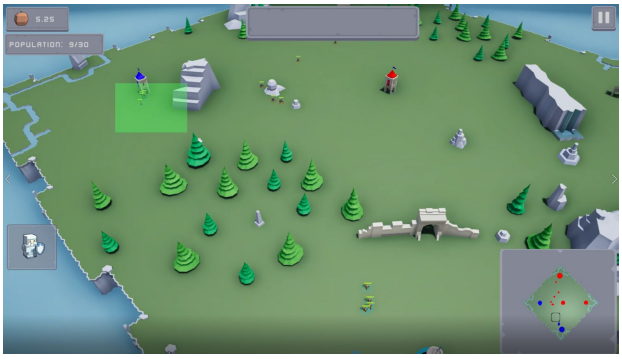
RTS + single vehicle control

This game type was demonstrated in Warshift as well as Battle Zone and features an RTS style game with a main focus on vehicle units. At first, we thought that vehicle units would be easier for us to implement as our team has less experience in animations and graphics. Once we began to see these features implemented in other RTS games however we soon realized that having a variety of different units with different attack types and techniques added strategy and made the game more interactive. Since vehicles are mostly ranged type attacking forces, it limits strategies to those that only have ranged attacks. We have also found it to be more engaging from a 3rd person action perspective to be able to play as melee or ranged characters. This is why we decided to move away from vehicles and implement other units with a variety of attacks.

## 3. Final Game Design

### 3.1. Overview

In order to address the needs previously identified, we decided to implement an RTS game that allows the player to switch to a third person unit. We have found that combining the strategic elements of a traditional RTS game (purchasing and commanding a large number of units) with the short-lived exhilarating action of a third person action game is the best way to accomplish our goal. The game will be centered on the gameplay mechanic of switching between Commander mode (RTS) and Unit mode (3rd person action), and every aspect of the game should encourage use of that mechanic. The player will be able to switch between the two roles, and compete against an AI opponent that can do the same.


*Top Left: Start Menu, Top Right: Commander mode combat, Bottom Left: Unit mode combat, Bottom Right: Victory Scenario*

Next, we will define some of the key details of each major component:

### 3.1.1. Gameplay Logic

Game loop:

- This component is responsible for containing the logic that dictates the game. The game loop controls what is happening in the game at any given time, and runs upon starting the match. It is here that the other components, such as the User and AI, will be authorized to execute their logic. Following User and AI operations, this loop updates the score information, and checks if any win conditions have been met.

Resource management:

- Resources refer to the currency used to purchase units in this game. While many RTS games have the player sending units around the map to collect resources, we have decided to take a different route. We have a small passive income stream of resources for both the player and AI opponent, however, resources can also be obtained by defeating enemy units or capturing resource points. Resource points are placed in the center and two opposite corners from the player's and AI opponent's base. These three aspects of our resource system will encourage the player to engage the enemy units, as well as utilize Unit mode to get a better chance at gaining more resources. The player or AI can then spend the resources they gained on additional units at their base.

Win Condition:

- Enemy base destruction is the traditional method of winning an RTS game. The goal of this is to destroy the opponent's base before they destroy yours.

### 3.1.2. User

The user's interaction with the game consists of the two parts, each with their own input system and other methods.

- Commander Mode - RTS Gameplay

This mode is the traditional RTS style mode. The player is looking at the map from a top-down perspective, and can freely move their camera over the map using the mouse or keyboard. It is in this mode that the player can command troops on the field. This is done by highlighting the troops the player intends to command, and then selecting a point on the map or target. If the player left clicks the target, the units being commanded will navigate to that position. If the player right clicks the target, the units will rush to engage in combat with the target.

If the player wants to switch into Unit Mode, they must select a single unit on the field and press the switch button on the UI. If they have multiple units selected, the game automatically chooses one for the player to enter Unit Mode with.

- Unit Mode - 3rd Person Gameplay

  When in Unit Mode, the user is playing as a particular unit in 3rd person. This mode is intended to feel like a traditional 3rd person action game. The camera shifts from the overhead RTS camera to a closer, over the shoulder orbital camera. The camera orbits around the player as they move their mouse. This allows the player to see all directions without having to change the orientation of the unit character. Additionally, this enables the player to view the front of the character, which is not possible with a static over the shoulder 3rd person camera. To move, the player can use either the WASD/arrow keys, or a controller joystick. Forward input (ex: W or UP-ARROW key) will always move in the direction the camera is currently facing. The player can attack using the left mouse button (LMB), sprint by holding SHIFT while moving forward, and jump by pressing SPACE.

  Due to the interactive nature of a 3rd person character controller, a particular character animation may not finish playing or looping at the correct time to shift to a new animation. An example is that Unity does not know when the player will press the SHIFT key to begin sprinting. To address this, the animations are combined using a blend tree. This smoothly blends between animation state changes that are a result of the actions a player takes.

### 3.1.3. AI Opponent

The player competes against an opponent AI. This AI is limited to the same game actions as the player as well as has the same amount of information that is given to the player. It uses this information as features in its decision making algorithm. The commander AI will be able to decide when to switch between modes, build and command forces, as well as perform tactics to capture resource control points. The AI commander also has the same limitations as the player and therefore loses control of their allied troops once they choose to switch perspectives to unit mode and power up a specific unit. Since the AI does not actually change perspectives, it simply chooses a unit to mark as powered up and the AI unit makes new decisions based on their new buffed stats. The AI makes use of this to try and outsmart the opponent and win the game.

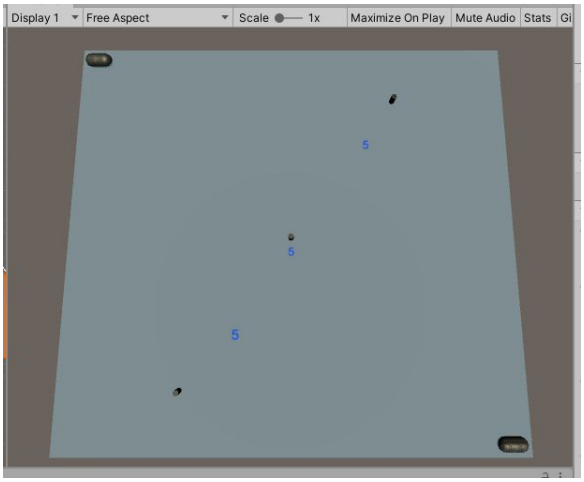
We are going to implement a deep reinforcement learning approach for each unit that will act as the tactical manager for the commander AI. These individual learning agents will have a shared neural network as their state function approximator. This will allow units with similar states to be able to work cooperatively. Each unit will extract hand-picked features that were chosen based

on the domain knowledge we have about the game to win. The AI will take snapshots of the gamestate to extract key features in its decision making such as resource income, available resources, how defensive/offensive an opponent is acting, etc. as well as following key rules of thumb to decide its strategic decision making. Once a unit extracts its current features for a given state it can then decide what the best action is for it to take.

There are however some issues with having a reinforcement learning agent in a real-time strategy domain. These issues are sparse rewards between actions, a continuous actions space, actions that are durative, as well as being able to make decisions and actions in real time for multiple agents. To overcome these issues we decided to discretize the action space depending on if an agent is near a reward or not. This would help training time significantly as it won't spend time testing out all the different actions that may be extraneous. Since we are also giving the agent hand-picked features it should reduce the complexity of the problem for the AI to solve. As for rewards, they will be based on key objectives in the game such as defeating an enemy, attacking the enemy base, capturing a resource control point, or winning the game. To handle the problem with actions being durative we decided to have a specified time between decisions. This specified time is reduced when it is closer to rewards or when it may need to make actions faster and is increased when they are far from rewards.

The AI used a Soft Actor Critic (SAC) algorithm to train on this environment. The SAC algorithm is a reinforcement algorithm that focuses on entropy-regularized learning. This means that it tries to maximize the amount of rewards that it receives while trading off with how random its actions are in the environment.

### 3.1.4. Unit/Agent AI



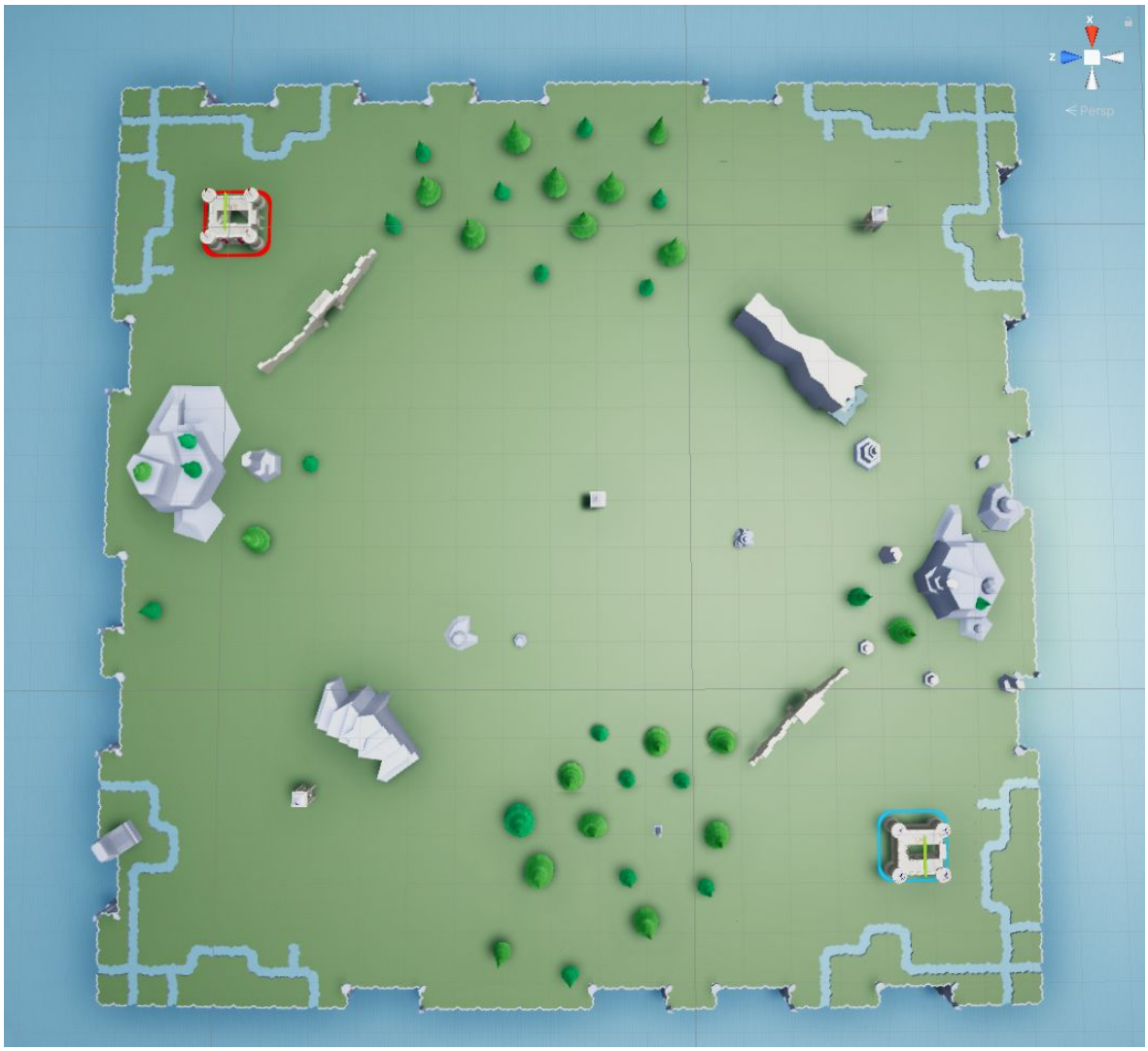

This component refers to the decision-driving logic for each individual unit.

Since there are relatively few actions the agent AI can make, we determined a finite state machine is the best way to implement them. These agents will derive from an abstract agent class that contains fields and methods that would be common across different agent types, such as health and taking damage. The advantage of using a finite state machine (FSM) for each of the enemy types is that we can reuse some states. While an attack state will vary largely between an archer agent and swordsman agent, their travel state would be identical. We have listed below the states we foresee the agent AI requiring.

1. Idle
2. Moving
3. Attacking
4. Dead
5. Possessed

### 3.1.5. Level Design

When designing our level, we referenced popular RTS levels from other games, such as *Halo Wars*. The level has a symmetric layout, offering no significant advantage to either team. The player's base is in the bottom right of the map, and the enemy's base is in the top left. Each base is protected by a front gate. This gate acts as a bottleneck for defending units to hold to protect their base, although there are alternate routes to the base via either the forest or the stone path. To the left of either base there is a small forest. To the right, an area populated by large stones, creating a path against the edge of the map. The purpose of these features is to allow for alternate unit movement options for the player. In the middle of the map are three control point towers. They are separated by large rock structures to prevent units from having line of sight of other control points. The middle of the map remains largely open to allow for an unobstructed view of the battle around the control points.



### 3.1.6. Theme

We decided on a fantasy based theme with ground-based units. We determined that ground combat between humanoid units would be more engaging for the player than other options. This gives the player a way of understanding why they are controlling sword fighting units. We chose to use a sword fighting unit over others like archers, because the upclose fighting of the sword units would give the player an easier view (only focusing on one type of unit) on how to make their strategic decisions. This allows the player to choose how they should use their units and how many they should commit to a given fight.

## 3.2. Gameplay

To illustrate what the gameplay is like, we have provided an example game scenario below:

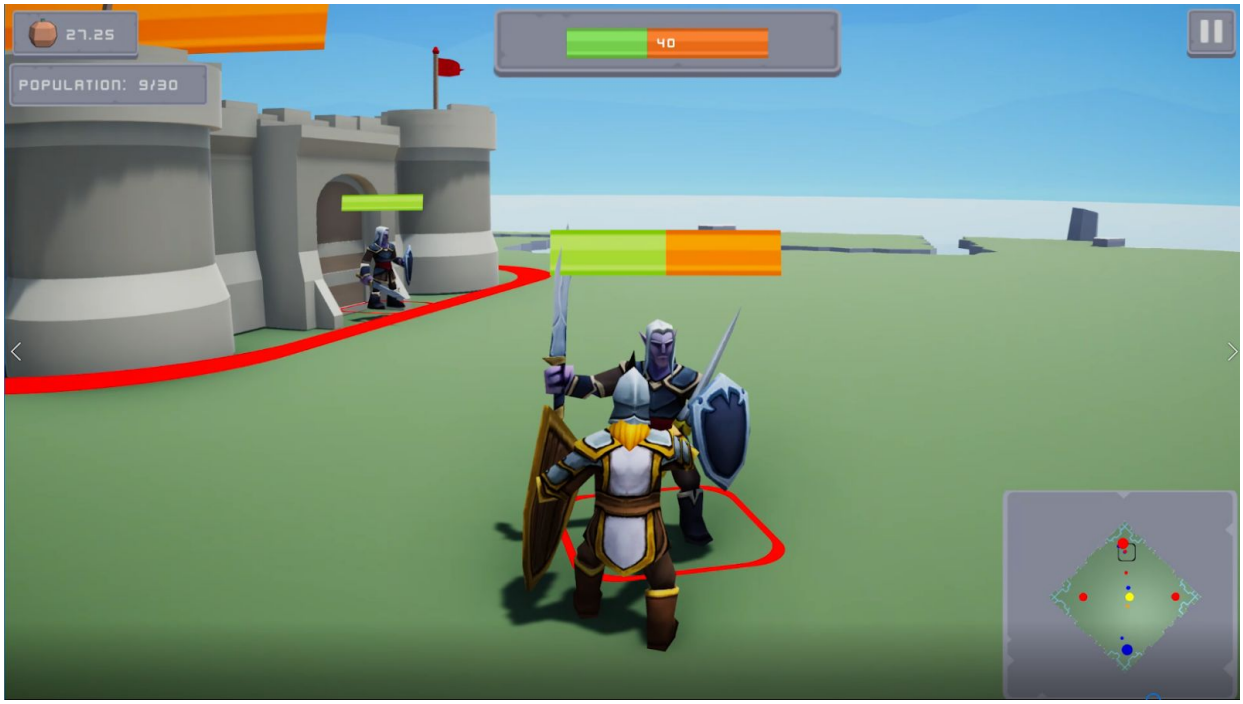
When the player runs the game program they are brought to a Start Menu screen, where they can either start the game or quit the application. When the player clicks Start Game, the game begins. The player begins in Commander Mode and is met with a view above their own base.


The player then buys their first unit by clicking on the unit button on the left side of the screen. This is done using the resources the player possesses, viewable in the top left corner of the screen. After the unit spawns in front of the base, the player selects it and orders it to one of the three capture points on the map. The player continues purchasing units and ordering them to go to control points until they make contact with the enemy. When this happens, the control point goes from Blue to Yellow, indicating that it is contested for ownership (*next page*).


To ensure the player will keep this point, they select a local unit and enter Unit Mode. They then defeat the few enemy units contesting the point with their increased damage from Unit Mode.


After securing the point, the player continues purchasing units and sending them to further reinforce control points, entering Unit Mode when necessary to preserve their unit's lives. Eventually, the player mounts an offense against the enemy's base when they have gathered enough units.


During this time, the player enters in and out of Unit Mode to defend their offensive team against the enemy units spawning at the enemy base.




This continues until the player's units have destroyed the enemy base, winning the game for the player. The player then has the option to play again, or exit the application.

This gameplay scenario displays how our Commander-Unit mode switching mechanic provides the player opportunities to practice strategic decision-making, as well as participate in an active

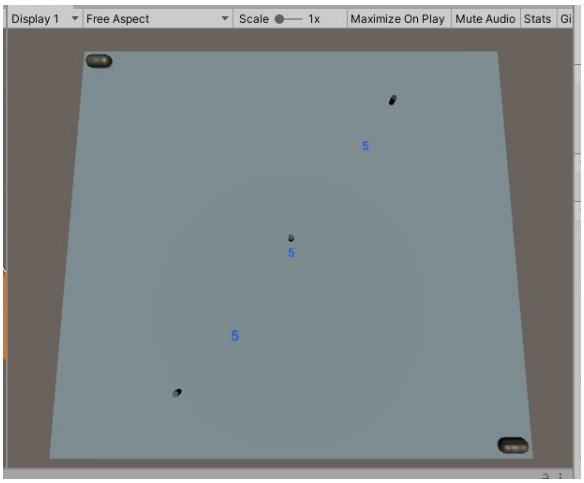
combat role. At no point in this scenario was the player waiting idly for the next event to happen, but instead felt in control of the outcome of their actions. This is the intended result of our design, which we will strive to facilitate as we develop the game.

### 3.3. Development Process

#### 3.3.1. Commander AI

In the following section we will describe some highlights during development that led to the creation of the commander AI. These highlights are in a pseudo-chronological order and were the stepping stones into the final implementation of the AI.

- Decision to use Deep Q-Learning Agent over Decision Search Trees
    - We originally were trying to try a simple approach by using a decision tree for the AI to make actions. This however led to issues mostly because at the beginning of the creation of the game we were not sure how we should represent game states as well as how to determine how valuable each state was. This approach was dropped for a more versatile approach by transitioning to state representations using an approximation function as well as training on more direct rewards from the training environment.
    - We decided to just use a single Q-learning agent for each unit so that we could share the neural network between them. This would increase the rate of them learning since it would take less time for a specific unit to locate a reward as well as allow them to exploit previous agents' knowledge. This simplifies the problem as well since the AI doesn't have to consider the location of every agent in its game space.
- AI Playground for testing
    - Since the game still had to be developed there was not an effective way of being able to train. We decided that we would need to develop a simple playable representation of our game in order to be able to train in an environment that would be fast enough to have thousands of episodes worth of learning. We created an "AI Playground" to be able to find a feature set that would be most useful for the AI to utilize in their decisions. A screenshot of this playground can be seen to the right and visualizes the bases, resources capture points, and who the resource capture points belong to as well as how much time is left to capture. By default, there is a timer set to 5 seconds that counts down when a player is near enough to it, once they capture it, the timer goes away and then displays the team that has captured it. This way we could actually train our model in the playground without having to train in our RTS engine. After we trained it to our liking in the playground, we can then move the neural network to the

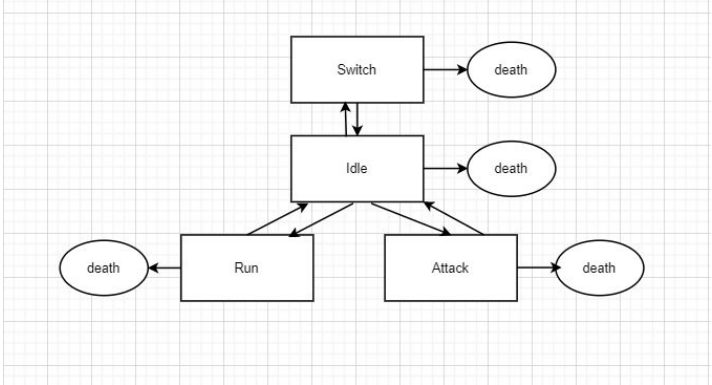


agents in the RTS engine and have then make decisions based on their observations from the actual domain of the game.
    - This allowed us to be able to narrow down a good feature set for the domain as well as test various things for the AI. Testing was used for training, validations of observations on the state, validations for actions being taken in the game, and tuning of hyper-parameters.
- Discretizing action space based on distance from local rewards
    - One issue with training in RTS environments is that the action space that the agent can work in is continuous. To overcome this, we decided to discretize the action space based on how far away the agent was from the nearest reward. The agent is able to mask actions that would be unnecessary when it is far away from a reward. This way, we can reach rewards sooner, by learning on a smaller action set when they are far away from rewards and learning a more complex action set when they are closer to rewards and need to have a more advanced behavior.
- Added a decision time step
    - A decision time step tells a unit to make a decision every *n* seconds. Adding a decision time step was something that was needed since in a RTS domain the actions are durative. We also change the decision time step for each unit depending on their state so that when they are near rewards they have time to make more decisions as opposed to when they are far away from rewards.
- Feature set for AI
    - Some key features include, distance to resources, distance to enemy base, if a specific unit is possessed, distance to closest enemy, etc.
- Using Proximal Policy Optimization and Soft-Actor Critic for Deep Q-Learning
    - We experimented with on policy learning implementations (PPO) and an off policy learning (SAC) in our playground to test to see which one would be able to train faster and more effectively.
    - The key difference between the two algorithms is how they learn. PPO is on-policy which means that the algorithm learns by following a specific policy. This is in contrast to off-policy where it learns the value of the optimal policy independent of the agent's actions.
    - After running multiple tests on the speed and effectiveness of training using both algorithms, we learned that SAC was able to converge to a better optimal sooner and better than the on-policy counterpart.
- Emerging gameplay
    - After training, the AI would learn to always capture points that would be the safest to capture (closest points, points with least enemy units, points that can be captured before the player can).
    - AI would also learn how to defend control points that they own.
    - AI learned that they can switch perspectives to gain an upper hand every time they begin an important engagement between units.

### 3.3.2. Unit AI

To implement the unit AI, we use the finite state machine to define different behaviors for the Unit AI. Currently, we define the following states: Idle, Move, Attack, Death and Possessed. When entering a

specific state, the game animation associated with the state would play. Unit agents are able to move to a certain position by clicking the mouse and to attack automatically if an enemy is within the attack range. When entering the Possessed state, which means to switch to Unit mode, it will enter an possessed state and disable the transition between the state machine. There are slight differences between the player side and the AI side. For the Player side, the input of moving is the mouse click position. For the AI side, the input of moving is a specific vector3, which is a vector consisting of x,y,z components. The picture below indicates the relationship between each state.




### 3.3.3. Third Person Controller and Camera

- Basic locomotive actions implemented
    - Using the run animation and current input states for horizontal and vertical input sources (WASD, Arrow keys, joysticks), the player can move the character forward, backward, left, and right.
- Sprinting and jumping actions/animations added
    - When the player is moving in any direction, pressing the SHIFT key will increase their movement speed, and transition to a sprinting animation.
    - When pressing SPACE BAR at any time, the player character will be translated vertically as a jumping animation plays. No other locomotive/combat actions can be taken in this airborne state.
- Orbital 3rd person camera integrated
    - Added a camera controlled by the movement of the user/s mouse. It revolves around the character at a fixed distance. It also checks for collision with objects in the environment to prevent clipping.
- Animator blend tree implemented for all possible animation transitions
    - The animations for each state of the player controller (idle, running, sprinting, jumping, falling, attacking) are fed into a blend tree when transitioning. This smoothly transitions the movement of the character model by interpolating the model's rigging/bones while changing animations.

### 3.3.4. RTS Engine

- Description of the RTS Engine
    - The engine runs on several managers that control each part of the game. The managers are the Game, Terrain, Input, Selection, Movement, Attack, Task, Unit, UI, Upgrade, Resources, Building, Mission, Main Camera, and minimap

managers. All these managers are what creates a basic game for anyone to create an RTS Engine with. The Terrain, Movement, Attack, Task, Unit, Upgrade, Resources, Building, Mission managers were removed from the main game. This was done to remove extra not used scripts from our game.
    - We used the Game, Main Camera, Minimap, UI, Selection, and Input managers. These managers are essential to the functionality of the in game HUD (Heads up display or the UI interface). These managers were combined with our developed code to manage different parts of our game and update the HUD as things happened.
- How we used the RTS Engine
    - We used the engine for many of the UI elements, camera movement, and Unit selection functionality.
    - The UI has elements for the Main Menu as well as the in game HUD. The in game HUD gives the player all the information such as population count, map of the play area, and the button to create units.
    - The camera functionality that we used was pulled from the engine and easily implemented into our game. We did this so we did not have to re-code all of the functionality of an already created asset.
    - The Unit selection functionality was pulled from the engine as well. The manager that provided the selection box as well as the individual selecting of units. The manager already provided a location when the right mouse was clicked. This allowed for easier integration to tell units where to move.
- UI Manipulation
    - The UI has many connections to the game so that the player has updated information at all times. We are manipulating the in game HUD to display only the important information with extra resources and buttons that are not being used from the engine being removed.
    - The Resources part had 4 different resources, but we are only using one type of resource so the others were removed.
    - Buttons such as the God Mode or builder selector were unnecessary with our game so they were removed to keep the HUD clean and easy for a user to understand.
- Documentation
    - The engine came with a lot of documentation that was read so we could understand how to pick pieces from the engine and use them for our game. We also had access to all of the code that made the engine run, so while we were adding RTS functionality, we could pull already written code from the engine.

### 3.3.5. Integration

Integration of each component of the game needed some form of API in order to communicate. Each of these communication methods will be described below. We originally had planned to integrate all of our parts into the RTS engine, however, due to restrictions on time we've decided to integrate the RTS engine into the AI playground (AI test scene) that was created to test and train the commander AI during our development. This allows us to still keep the useful UI from the RTS engine without having to manipulate a lot of their code so it works for what we needed.

By doing this, we had saved time integrating the commander AI as it is already implemented and it also allowed us to save time by working in an already familiar and stripped down code base.

**Commander AI**

Combining the commander AI was easy since it had already been implemented into our AI Playground. We only needed to integrate the AI's commands into the finite state machine (FSM) so that way both the player and the AI only make changes to the game state through the FSM of each unit.

Input:

- The input will be the state of the game for each individual unit. For *n* units, there will be *n* states that will be processed.

Output:

- A command is generated for every unit to execute in the form of a position (Vector3). Each individual unit is given its own command. For *n* units, there will be *n* positions generated. These commands will then be processed by each unit's animator to change the state of that unit AI's FSM.

**RTS Engine**

The RTS Engine has all of the components for an already built RTS Game. We had found that pulling parts from the RTS Engine to put into another system was easier than manipulating and adding our own part to the engine as the engine has some very webbed systems. Pulling parts allowed us to specifically use what we need and not have extra stuff that is not needed. Below is how the player and the AI will interact with elements from the RTS Engine.

* *Vector3:* A one dimensional array consisting of three individual decimal values. In our context, this defines a position point in our 3D scene.

Input:

- Selection (only used by Player):
    - Players select units by creating a bounding box using their mouse or selecting a specific unit using the left click action on a mouse.
- Destination for specified unit(s):
    - After selection, the player uses a right mouse click on a location of the map (terrain the game is played on, not the minimap) to issue destinations for selected unit(s). This is processed as a position vector (Vector3) within Unity from a ray cast from the camera.
    - AI will input a specific destination for each unit by passing a Vector3 as well the identifier for that unit.
- Purchase:
    - The AI issues the command to purchase a unit directly to the main base of the AI.
    - The player issues this command by clicking on the UI button (located to the bottom left of the screen) to purchase a unit. This button issues a command to the main base of the player to purchase and spawn a new unit. The amount of resources that the player has to spend is displayed in the upper left hand corner.

Output:

- Selection (only used by Player):
    - Units that are selected will become highlighted for their own team with a green square under that unit.
- Destination:
    - Units that are already selected will move towards the place the player selected.
- Purchase:
    - Spawns in a prefab (pre made unit that has all of the components needed) of the unit for the player or AI.

**Unit AI**

When beginning integration, we needed to be able to combine this section with the commander AI and the RTS Engine. We first combine the Unit AI with the 3rd person mode so that the AI can be disabled and does not try fighting against what the player wants to do while in control of the Unit. Then we combined it with the Commander AI and player's UI so that both the AI and player could spawn and control its units. They would issue commands directly to the Unit AI which can be interpreted and executed on. Below is the API that we have thought about and created inside of the Unit AI for easier integration.

Input:

- The inputs of the API are coordinates of a destination in the form of a Unity Vector3.
- Input of a tab press

Output:

- State change to move to that destination that was specified in the inputed Vector3. When the unit reaches the position that was given, the unit will then decide whether it is in range of an enemy to attack or just wait for the next command.
- Will disable the Unit AI if the tab button is pressed so the unit is free for the switching mechanic.

**Switching Mode Mechanic**

To enable the switching mode mechanic, we needed to combine the Commander Mode and the Unit mode (different from the Commander AI and Unit AI). We need to make sure the transition between two modes is smooth, since it involves the camera movement.

Input:

- Player:

  Commander Mode -> Unit Mode: Utilizing the RTS Engine's selection functionality, the player selects a single unit or multiple units with their mouse. The player can then press the TAB key to switch into Unit Mode with the single selected unit or one of the multiple selected units.

  Unit Mode -> Commander Mode: The player can switch back to Commander Mode by pressing the TAB key, or freeing their mouse (which is used for camera rotation) by pressing CTRL, then clicking the UI switch button with their mouse.

- AI:

  From Commander Mode: Because the enemy AI simply increases the combat statistics of the unit it wishes to enable "Unit Mode" with, it does not need to interact with other components. The Commander AI determines when it should enter Unit Mode with its own logic, then simply increases the values of the combat-related member variables belonging to that unit.

Output:

- Player:

  Changing of control scheme (RTS -> third person controller), shifting of camera from RTS overhead to third person camera, and change of UI to represent current abilities.

- AI:

  Alteration of a specific unit's combat-related variables. The unit's combat-related member variables are increased when switching from Commander Mode to Unit Mode, and decreased when switching from Unit Mode to Commander Mode. When in Unit Mode, the Commander AI cannot issue commands to the other individual units.

## 4. Project Retrospect

After creating our game and having people playtest we found some key insights for our game. We continue in this section highlighting some things we found went right and somethings that did not. We also include a section that highlights some ideas of changes that could be made to improve the game.

### 4.1. What went right

- Players that made good use of the core game mechanic were able to beat the AI.
- Utilizing the core game mechanic also led to more engaging games.
- Level design made the world more immersive from the 3rd person perspective and gave the game depth.

### 4.2. What went wrong

- After play testing we found mixed results with the AI. Some people found it very difficult, while others found it to be too easy.
- Slight learning curve due to the lack of controls in the menu of the game.
- Found a bug that doesn't allow the user to create a bounding box sometimes on the right side of the map.
- Players are able to spawn more than the unit limit, if they spam unit spawner button rapidly.
- AI is too offensive and doesn't defend their base when they are winning.

### 4.3. What we would change

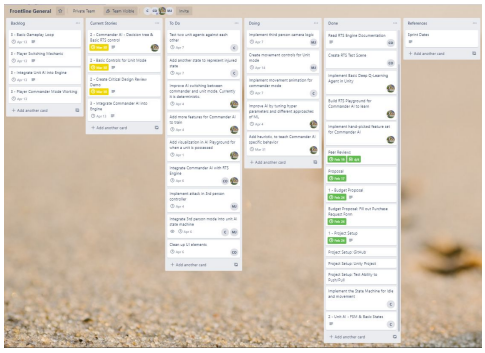
- Create a more defensive AI by introducing new features to train on. These features would be able to specify when the opponent is pushing their base.
- Fix bugs that we found in playtesting.

- Add a controls menu in the game as well as a quick tutorial on how to play.

## 5. Project Management

### 5.1. Team Management

Game development is a little bit different from normal projects, so we will not use the Gantt chart to track the current progress of the implementation. Instead, we will use the Agile and Scrum approach to manage our project. The tool we will use is Trello which is suitable to implement the Agile approach. The essential lists are Sprint backlog, current Sprint, To do, Doing, Done. The Sprint backlog includes all the larger tasks we need to do before the critical design review. Currently we have three Sprint and each Sprint duration takes around two weeks. The Current Stories section includes the current Sprint. The TO DO list is detailed tasks needed to accomplish during the Current Stories section. Here, we break larger tasks to smaller tasks. At the current stage, the detailed tasks in TO DO list do not assign tasks to particular people. After the initial Sprint period, each person in our group will be responsible for several tasks. Once a task finishes, we will move the tasks from the TO DO section to DONE section.



In the 5.2 section, there is a screenshot of all of our main tasks and when they were started and completed. This shows the project progress over time as well as shows critical design changes such as when we decided to use the AI playground logic instead of the RTS engine.

## 5.2. Project Timeline

**Frontline General**

| PROJECT TITLE | | COMPANY NAME | 3RTS Studios |
|---|---|---|---|
| PROJECT MANAGER | | DATE | 4/29/20 |

| WBS NUMBER | TASK TITLE | WEEK 1 | WEEK 2 | WEEK 3 | WEEK 4 | WEEK 5 | WEEK 6 | WEEK 7 | WEEK 8 | WEEK 9 | WEEK 10 | WEEK 11 | WEEK 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 10 Feb | 17 Feb | 24 Feb | 2 Mar | 9 Mar | 16 Mar | 23 Mar | 30 Mar | 6 Apr | 13 Apr | 20 Apr | 27 Apr |
| 1 | **Design and Set up** | | | | | Spring Break | Covid-19 | | | | | | |
| 1.1 | **Design** | | | | | | | | | | | | |
| 1.1.1 | Gameplay Logic | | | | | | | | | | | | |
| 1.1.2 | Unit AI Design | | | | | | | | | | | | |
| 1.1.3 | Commander AI Design | | | | | | | | | | | | |
| 1.2 | **Project Set Up** | | | | | | | | | | | | |
| 2 | **Programming** | | | | | | | | | | | | |
| 2.1 | **Commander AI** | | | | | | | | | | | | |
| 2.1.1 | Create Basic AI | | | | | | | | | | | | |
| 2.1.2 | Conduct AI Training | | | | | | | | | | | | |
| 2.1.3 | Implement Base logic | | | | | | | | | | | | |
| 2.1.4 | Implement Resource Points | | | | | | | | | | | | |
| 2.2 | **Unit AI** | | | | | | | | | | | | |
| 2.2.1 | Implement State Machine | | | | | | | | | | | | |
| 2.2.2 | Implement Idle, moving, attacking, and dying | | | | | | | | | | | | |
| 2.2.3 | Implement click movement/set destination | | | | | | | | | | | | |
| 2.3 | **Third Person Mode** | | | | | | | | | | | | |
| 2.3.1 | 3rd Person camera/movement controls | | | | | | | | | | | | |
| 2.3.2 | **Implement 3rd person aiming/attacking** | | | | | | | | | | | | |
| 2.3.3 | Implement Commander <-> Unit Switching | | | | | | | | | | | | |
| 2.4 | **RTS Engine** | | | | | | | | | | | | |
| 2.3.1 | Understanding Documentation | | | | | | | | | | | | |
| 2.3.2 | Creating map for integration | | | | | | | | | | | | |
| 2.4 | **Integration** | | | | | | | | | | | | |
| 2.4.1 | integrate Commander AI into RTS Engine | | | | | | | | | | | | |
| 2.4.2 | Basic Unit AI control for player | | | | | | | | | | | | |
| 2.4.3 | Commander AI spawning Unit AIs | | | | | | | | | | | | |
| 2.4.4 | RTS UI into Command AI scene | | | | | | | | | | | | |
| 2.4.5 | RTS map integration into Commander AI scene | | | | | | | | | | | | |
| 2.4.6 | 3rd person integration in Commander AI scene | | | | | | | | | | | | |
| 2.4.7 | RTS UI integration with 3rd person mode | | | | | | | | | | | | |
| 2.4.8 | RTS UI controlling Unit movement | | | | | | | | | | | | |
| 3 | **Testing and Validation** | | | | | | | | | | | | |
| 3.1 | Testing | | | | | | | | | | | | |
| 3.2 | Validation | | | | | | | | | | | | |
| 4 | **Documentation** | | | | | | | | | | | | |
| 4.1 | Proposal Document | | | | | | | | | | | | |
| 4.2 | Proposal Presentation | | | | | | | | | | | | |
| 4.3 | Critical Design Review Document | | | | | | | | | | | | |
| 4.4 | Critical Design Review Presentation | | | | | | | | | | | | |
| 4.5 | Project Demo | | | | | | | | | | | | |
| 4.6 | Final Document | | | | | | | | | | | | |
| 4.7 | Final Presentation | | | | | | | | | | | | |

## 5.3. Validation and Testing

In order to validate our games success, we did both a design and technical validation process. This process gave us a sense on how well our game fulfilled our goal of combining the strategy from a RTS game and combat action from a third person action game. We got our design and technical validations from getting user feedback on how they feel the games went after allowing them to play it a few times. We also included a survey that the player would fill out after they played to gather important data regarding the playtest. Gathering data after the user plays the game will be the premises behind how we will validate our design and technical constraints.

The design validation was the process of testing our games core mechanic. We asked people to play our game and kept track of how many times they switched between Commander and Unit views. This will give us a metric on whether or not the core mechanic was used effectively by the player to keep them engaged in the action for most to all of the game. We also asked other questions in the survey to learn how difficult they thought the AI was, how useful they found the game mechanic to winning the game, as well as other data on their sentiment towards RTS and 3rd person action games. We will also try to keep track of the amount of time the player stays in the Unit view and Commander view throughout the game. This way we can look at all this data and try to find a pattern of win percentage based on how well they utilized the core mechanic. It will also give us a sense of if the core mechanic actually led to more engaging gameplay.

Another way we performed design validation was by allowing two AIs to play against each other where only one of them can utilize the core mechanic of switching perspectives. This way, we can see that the AI that is unable to use the core mechanic should be at a disadvantage and lose the game. We were able to validate this in the AI playground, where AIs that were unable to use the core mechanic lost more games than won against the same AI who was able to utilize the core mechanic.

For technical validation, we proceeded with a similar process to the design validation. We tried to play games on Windows system computers of varying computational ability by asking our friends to playtest on their own personal computers. We gathered information while the game was being playtested and found that our game was able to play on computers with no dedicated graphics card and found no variations in framerate or ability to run across multiple computers. This validated our technical requirements of the specs of the computer that would be required to play our game.

### 5.4. Division of Labor and Responsibilities

When we started we had a clear definition of what each person was going to start doing and working on. We had broken the project into four parts that each of our four team members could take. Part one was the Commander AI. This was given to Diego as he felt the most confidence in creating the AI for the player to play against. Part two was Unit AI. This AI's implementation was done by Jingdi who had the responsibility for implementing an AI that would move to locations and attack automatically when in range of an enemy. Part three is the third person mode or unit mode. This was given to McLain with the responsibility of giving the user the same abilities that the Unit AI has, however giving the player the control of the unit. The final part was creating the UI and map and visuals used for any RTS game. We bought an asset pack that gave us useful UI and RTS elements that we could use to add to our game and make it run. This part was given to Chris with the responsibilities of reading the documentation for the RTS Engine and creating a map for the game with the assets provided in the engine.

After completing each part to an extent that we could combine the parts. We began with the intention with the following breakdown of tasks for each person:

- Commander AI <- RTS Engine
    - This section will be integrated by Chris and Diego as they worked on these two sections.
- Unit AI <- 3rd Person Mode
    - This section will be integrated by McLain and Jingdi as they created the two sections.

With this plan of integration done, we then convien to integrate these now two parts into the final version of the game. This means that the combined RTS Engine and Commander AI part will be combined with the Unit AI and 3rd Person Mode. This is how we moved forward in our integration and the testing, for making sure the sections run together. Testing of the parts was done by the group members that worked on each part. This was done to make sure that the parts worked together and would be ready to be combined into one game.

More testing followed when the game was fully combined. We each did our own testing to make sure that the game can be played without game breaking bugs. We then sent out the game to some friends to have them play and give us feedback on the game.

### 5.5. Budget

To build a game with great visual effect, we required the purchase of the following Unity assets. These assets help us to expedite our development progress. Since our project is a software based game, no additional cost needed.

- RTS Engine - $60
- Elf character set - $20