cONSTRUCTING AN ACCESSIBLE REAL-TIME STRATEGY GAME.

**Name:** Harry Sutherland

**Student ID:** 13465591

**Title:** Constructing an accessible real-time strategy game

**Qualification:** Bachelor of Information Technology

**Course ID:** PRJ701

**Year:** 2017

**Blog:** https://harrysrtsproject.wordpress.com/

Contents

[Introduction 6](#_Toc497305662)

[Background 7](#_Toc497305663)

[Project Plan 8](#_Toc497305664)

[ImplemEntation 10](#_Toc497305665)

[Example heading 10](#_Toc497305666)

[What 10](#_Toc497305667)

[Why 10](#_Toc497305668)

[How 10](#_Toc497305669)

[Example 10](#_Toc497305670)

[evaluation 10](#_Toc497305671)

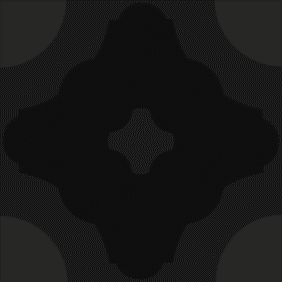
[Height mapping 10](#_Toc497305672)

[What 10](#_Toc497305673)

[Why 10](#_Toc497305674)

[How 10](#_Toc497305675)

[Example 10](#_Toc497305676)

[  10](#_Toc497305677)

[evaluation 11](#_Toc497305678)

[BoX Selection 11](#_Toc497305679)

[What 11](#_Toc497305680)

[Why 11](#_Toc497305681)

[How 11](#_Toc497305682)

[Example 11](#_Toc497305683)

[evaluation 12](#_Toc497305684)

[Command Manager 12](#_Toc497305685)

[What 12](#_Toc497305686)

[Why 12](#_Toc497305687)

[How 12](#_Toc497305688)

[Example 12](#_Toc497305689)

[evaluation 12](#_Toc497305690)

[Commands 13](#_Toc497305691)

[What 13](#_Toc497305692)

[Why 13](#_Toc497305693)

[How 13](#_Toc497305694)

[Example 13](#_Toc497305695)

[evaluation 14](#_Toc497305696)

[Control Groups 14](#_Toc497305697)

[What 14](#_Toc497305698)

[Why 14](#_Toc497305699)

[How 14](#_Toc497305700)

[Example 14](#_Toc497305701)

[evaluation 14](#_Toc497305702)

[Camera Locations and panning. 15](#_Toc497305703)

[What 15](#_Toc497305704)

[Why 15](#_Toc497305705)

[How 15](#_Toc497305706)

[Example 15](#_Toc497305707)

[evaluation 15](#_Toc497305708)

[ACtion keys and rapid fire. 15](#_Toc497305709)

[What 15](#_Toc497305710)

[Why 15](#_Toc497305711)

[How 16](#_Toc497305712)

[Example 16](#_Toc497305713)

[evaluation 16](#_Toc497305714)

[Click Confirmation 16](#_Toc497305715)

[What 16](#_Toc497305716)

[Why 16](#_Toc497305717)

[How 16](#_Toc497305718)

[Example 16](#_Toc497305719)

[evaluation 16](#_Toc497305720)

[Input Controller 17](#_Toc497305721)

[What 17](#_Toc497305722)

[Why 17](#_Toc497305723)

[How 17](#_Toc497305724)

[Example 17](#_Toc497305725)

[evaluation 17](#_Toc497305726)

[Terrain assets 17](#_Toc497305727)

[What 17](#_Toc497305728)

[Why 17](#_Toc497305729)

[How 18](#_Toc497305730)

[Example 18](#_Toc497305731)

[evaluation 18](#_Toc497305732)

[Gui improvements 19](#_Toc497305733)

[What 19](#_Toc497305734)

[Why 19](#_Toc497305735)

[How 19](#_Toc497305736)

[Example 19](#_Toc497305737)

[evaluation 19](#_Toc497305738)

[healthbars 20](#_Toc497305739)

[What 20](#_Toc497305740)

[Why 20](#_Toc497305741)

[How 20](#_Toc497305742)

[Example 20](#_Toc497305743)

[evaluation 20](#_Toc497305744)

[Attack and attackswing Command 21](#_Toc497305745)

[What 21](#_Toc497305746)

[Why 21](#_Toc497305747)

[How 21](#_Toc497305748)

[Example 21](#_Toc497305749)

[evaluation 21](#_Toc497305750)

[Move, patrol and follow commands 22](#_Toc497305751)

[What 22](#_Toc497305752)

[Why 22](#_Toc497305753)

[How 22](#_Toc497305754)

[Example 23](#_Toc497305755)

[evaluation 23](#_Toc497305756)

[Stop and Hold position command 23](#_Toc497305757)

[What 23](#_Toc497305758)

[Why 23](#_Toc497305759)

[How 23](#_Toc497305760)

[Example 23](#_Toc497305761)

[evaluation 23](#_Toc497305762)

[Gather and channel command 24](#_Toc497305763)

[What 24](#_Toc497305764)

[Why 24](#_Toc497305765)

[How 24](#_Toc497305766)

[Example 24](#_Toc497305767)

[evaluation 24](#_Toc497305768)

[Production Queue 25](#_Toc497305769)

[What 25](#_Toc497305770)

[Why 25](#_Toc497305771)

[How 25](#_Toc497305772)

[Example 26](#_Toc497305773)

[evaluation 26](#_Toc497305774)

[Equipping units 26](#_Toc497305775)

[What 26](#_Toc497305776)

[Why 26](#_Toc497305777)

[How 26](#_Toc497305778)

[Example 27](#_Toc497305779)

[evaluation 27](#_Toc497305780)

[Animations and models 27](#_Toc497305781)

[What 27](#_Toc497305782)

[Why 27](#_Toc497305783)

[How 27](#_Toc497305784)

[Example 27](#_Toc497305785)

[evaluation 27](#_Toc497305786)

[Bibliography 32](#_Toc497305787)

AbStract

In recent years the traditional real-time strategy(RTS) game has declined in popularity from one of the most popular genres, into a much smaller player base and popularity. This project was created under the assumption that the major reason for the decline is the bewildering complexity found in many real-time strategy games. The intention of the project is to create a real-time strategy game that is accessible to people with low game literacy or little experience in the genre in unity 3d. Unfortunately, the game did not progress far enough to be considered a more accessible version traditional RTS game. The development was built on a foundational series covering the absolute basics. Developments past this include creating scripts, 2d assets and 3d assets as well as working with unity subsystems such as Nav-mesh, height maps or animation controllers. Later in the development purchased assets were added to help save time on non-software development aspects of the project. The project plan underestimated the workload for the project to reach a point that made a comparison to current RTS games meaningful. However, the project did create a valuable learning experience that covered Unity 3d, C# programming, and game design. Most of the work on the project was focused on systems design which means there is a solid foundation for future progress that is easily expandable. The result is a game that is inconclusive in terms of the original question but has value in other aspects.

# Introduction

this report will detail the motives, intention, execution, results and future developments of a real-time strategy game intended for a less experienced audience. This section intends to give an overview of the project and the report.

The project was intended as an experiment in game design to see if a game built in the popular real-time strategy genre could maintain the depth and appeal whilst becoming easier to engage with for inexperienced players. However meaningful design decisions can only be made on something that resembles a game. In this case a real-time strategy game so it was necessary for the basic components of the genre to be present. This shifted the nature of the project away from building accessible mechanics and towards building systems that can support accessible mechanics. This is not a negative outcome just unintended. The when completed may have been a better starting point for the original question than starting from near scratch.

The background section details the motivations, research and other preparations leading up to the projects start. The journey leading up to the projects start happens nearly 5 months before and practical preparations were started in unity at around four months out. The section will act as a loose timeline from deciding the area of study and narrowing that down to the game in its current state.

The project plan Details the Intended method of the project and serves as an interesting comparison to the actual implementation. The original project plan had a set of milestones and smaller aspects that each milestone consisted of. The project plan discusses each milestone the reasoning for the milestones inclusion in the plan and its position in the schedule. Additionally, the project plan aims to show why aspects of the project fail to match directly to the plan or why aspects differ from the plan significantly. Primarily this section will be focused on the initial planning stages.

The implementation step will detail the actions taken in the development of the project. The implementation step will be by far the largest portion of the report detailing the actual development done and why it was pursued with the intent of covering not all the steps in development but significant ones. As a note, this section will not be focused on how these aspects apply to accessible real-time strategy games but rather the value each step had as a learning tool or practical use of a design pattern. As there are many steps taken in the projects development it is important to structure the information in a way that is digestible and easily understandable. To this end, the implementation section of the report will detail a format for implementation steps following a what, why, how and review for each element if possible there will also be an example of the element working this could be an image a code snippet or a video demonstration. Through this method, the reader should be able to not only understand the method used but also gain an understanding of the games function as a whole. This should give a meaningful but not exhaustive detailing of the implementation process.

Following this, the evaluation section aims to look at the outcome more holistically to connect the smaller more specific evaluations made in the implementation step. This step also intends to go into more detail about how the project lines up against the project plan. How successful the project changes wildly depending on the metrics used so the evaluation section will evaluate in terms of many different metrics.

The future development section will be protracted as well. The project has a massive room for further development. The Section will cover future developments from large project direction decisions to smaller system changes or implementations.

Finally, a conclusion to summarise the report and the final thoughts on the project as a whole.

# Background

This project was approached initially with the intent to do a development of some sort in game design which was then narrowed down to a smaller subset of genres, to an RTS game, to a project on focused on the aspects of RTS games that are prohibitive and finally to the final project plan.

This section intends to cover the motivation and reasoning that led to these steps and resources used. The initial plan was to use the allotted project time to create a game of some description. Early on the decision process was focused far more on what was achievable rather than what would be compelling useful or important. (<https://wordpress.com/view/harrysresearchblog.wordpress.com>) it was later decided that choosing a less interesting or compelling project because it could potentially reach completion was not ideal.  
after that aspect of the decision-making process was put aside it made the selection process one of personal passion and interest.

The RTS genre was chosen for its peculiar history. the real-time strategy genre has experienced a steady decline in since the turn of the century its peak in the 1990’s. (Evans-Thirlwell, 2016) (MOSS, 2017). The question became why what was undoubtedly the most popular genre in the world has become more of a niche. As an example, the major RTS titles Released In the year 2000 number at 30 a number which has decreased from that peak to only single digits within the past years and going as low as only 2 released in 2012. This shows that the popularity has been on the decline for nearly two decades.

As the genre was decided and the intent to make a game was decided all that was left to narrow the scope and focus it upon a single issue. The most obvious direction is to innovate on the genre in some fashion and create a product that adds something of interest or value. However, after the completion of the project, it has become clear a better direction may have been a focus on the process of creation of games in this genre or perhaps a focus on a specific system within the creation process such as unit action queueing, pathing or balancing. At the time, the direction chosen was to focus on creating a project that mitigated the aspects that lead to the popularity decline within the genre.

There are many views as to why the RTS genre declined and there is not one core reason. There are a variety of reasons ranging from production costs to the arrival of popular similar genres such as the MOBA (MOSS, 2017) (Evans-Thirlwell, 2016). One of the more compelling reasons is the barrier to entry posed by the genre. The genre popularised e-sports form of competition in the nineties and is one of the areas of the genre that has declined the least. However, a side effect of this is that the genre became associated with high speed and high skill requirements. The result is games that have an unreachable skill ceiling and an intended hardcore audience. This is also compounded by the fact that there are almost not RTS offerings for the more casual platforms such as mobile or console gaming. The core issue with lowering the barrier to entry, however, is that there will often be a sacrifice of depth which is the core appeal of such games in the first place.

This brought the project to a game that lowers the barrier to entry whilst still giving meaningful choice and strategic depth. The first thing to note is that that is a massive design question that is far too large to be attempted in a project of this size and that the question may better be asked in a research-focused project rather than a development one. This is a symptom of the core question of the project being one of the last things determined in defining the project. This meant that from before the start of the project the goal had shifted from designing to prove or elaborate on a concept to designing with a concept in mind.

Finally, as a preparation for the project, an extremely fundamental RTS game was started as per a tutorial (Sagmiller, 2016), to try and serve as groundwork for the project itself to allow for more time to focus on the core question rather than creating the basic functionalities of the genre

# Project Plan

The planning was not an in-depth process as unity and game design allow for more flexible development. Most games are collections of subsystems that are connected to each other through the game engine. In the case of unity, this means that dependencies are not as crucial as systems are not directly connected. For an example, the Control group system has nothing to do with the combat or camera systems and can be built independently. In the case of this project, the initial tutorial helped put some components in place to help the building process. This allowed for a less strict plan that consisted of features and systems that are needed without a concrete timeline.   
  
This is not to say that the planning was ideal either. There are many systems that could have benefitted from a more concrete design from the beginning to save time on revisions later in the project. Some requirements aren’t clear at the outset and cannot be planned for. however, the project would have benefitted from more granular planning and scheduling.

The initial milestone plan was prefaced with “it is likely that I won't get passed milestone 3.” This is only roughly a third of the way through all the milestones. The reason for additional milestones past that point is for any future work on the game past the initial project time frame and by defining many aspects on later milestones its becomes a lot easier to see exactly what *isn’t* in the scope for the project.

the intention with the milestones was to have odd numbered milestones add functionality and behaviour with even milestones adding polish and refinements. The exception is milestone nine which was added as more of an indefinite catch all step for additional development.

These are the milestones:

MILESTONE ONE: SINGLETON MODE

* One Unit
* One Unit producer
* One movement command
* One combat command
* One other unit action
* One building
* One resource
* Functional UI

This milestone was created to add at least one copy of all core components to the game. Which within an Object-oriented environment means creating the template for one of these components means doing so for all of them. This made miles stone one in to the longest milestone as it meant creating the foundation for most of the content and after it was completed it did a lot of the heavy lifting on many components.

The intent of the milestone was to have a copy of the components in the game with simplistic behaviour and have them functional in a play space.

MILESTONE TWO: GAMIFYING

* add a win condition
* add a mini map
* Add terrain complexity
* add an animation
* add a sound
* make UI attractive
* add visuals for all the stuff i forgot
* create a static challenge to be overcome

this milestone intended to create a more game like experience the core component of that being the win condition and creating a static challenge. For example, many enemy units in an area with no dynamic behaviour that needed to be defeated to win the game. This milestone was to be extremely simplistic but have the bare minimum to qualify as a game

MILESTONE THREE: COMPLEXITY

* add command queue.
* add basic unit AI.
* add additional movement commands (attack move, patrol, stop, Hold Ground).
* add production timers.
* add production queue.
* add a resource.
* add 2 units (3 total)
* add unit groups.
* add control groups

this milestone intended to add complexity to promote player choice with different movement unit behaviours and choices as well as adding more dynamism to the challenge with rudimentary AI. This step being focused on complexity may seem counter intuitive to promoting accessibility but that needs to be despite some degree of complexity that is a hall mark of the genre.

MILESTONE FOUR: POLISHED MINI-GAME

* background music
* Unit sounds
* unit models if possible
* simple animation
* themed visuals
* improve player feedback
* icons
* terrain detail improvement

the intent of this milestone Is to have a more complete experience. The core of this tier is trying to increase the information given to the player. Visuals and sound work to the same aim as “improve player feedback” in that they aim to inform the player in an intuitive sense. A sound when a unit is killed can help the player better understand a battlefield situation. In a similar vein terrain detail can inform player choices. Players are less likely to navigate over obstacles if it looks like a cliff. This milestone was meant to make a solid playable experience.

MILESTONE FIVE: A FULL RACE.

* 2-3 more units.
* more buildings.
* tech restrictions/tree.
* unique actions.
* upgrades.
* statuses.
* race features

this is a milestone focused on expanding the depth of the game. Additional units and buildings should only require models and animations being applied to unit prefabs. The idea of tech restrictions unique actions upgrades statuses and racial features each represent an additional underlying game system this milestone represents ideal outcome, not what wanted to be achieved within the timeframe of the project

MILESTONE SIX: META-GAME AND POLISH.

* a menu prior to the game
* settings that effect the game
* save and load
* visual, audio and feedback polish

these have a milestone not as a target but first and foremost to show a priority for these features as they don’t really assist in creating an accessible game or change the gameplay at all. The additional visual, audio and feedback polish represent that there are always additional touch ups and quality of life changes.

MILESTONE SEVEN: NETWORKING.

* allow synchronized play for two players in one game over a local network

MILESTONE EIGHT: SIMPLE RTS

* the goal is a clear game with a early, mid and late game with a clear win condition that is simplistic but not poor.

MILESTONE NINE: MORE

* additional units / races
* additional polish

These last three are being grouped together as they are all very much unrealistic ideal scenario plans multiplayer is mentioned here to show its priority milestone eight is essentially a completed albeit simple game and nine is an indefinite addition of content and refinement.

as this is a development project, items on this list such as adding sounds or animations would seem unnecessary but those can allow for intuitive delivery of information to the player. Especially is if the player is new to the genre or gaming in general for example if a unit can be seen swinging a sword it can convey what it is happening very easily. It is with accessibility in mind that the planning features so many items like this

this was the initial planning stages and the logic that informed it for the most part. Overall the planning step was well intentioned but overly ambitious and naïve. The plan did serve well as more of a prioritised check list of features and their urgency within the project.

# ImplemEntation

The implementation of the project contains many systems and features that are often self-contained not all added segments will be discussed here just components that were significant or took more than the usual amount of time or effort. For the most part, these will appear in chronological order or implementation within the project

This segment will be formatted in way that discusses the various components of the project’s codebase, animations and assets in a way that allows for understanding of all the component parts the break down for each component will be in this format: -

## Example heading

### What

This section will detail what was created added or changed about the project and how it effects the project

### Why

This section will detail the motivations behind implementing the component of the project.

### How

This will detail how the component was added and detail issues encountered, and resources used

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This will contain an evaluation on the individual component, what went well, what would be changed and what has been learnt in relation to development.

## Height mapping

### What

Height mapping is a short cut method to creating interesting terrain quickly. Height map is a 2d image in greyscale that is interpreted by unity’s terrain engine. the height of the terrain is mapped to the black and white value of the image with white area’s creating terrain at the maximum allowed height for the terrain and complete black areas create 0 height terrain. It is an alternative method to hand generating terrain.

### Why

Unity does have inbuilt terrain editors, but they are limited. It would be time consuming to create and must be built by hand. The height map allowed for an image to be built in Photoshop that could be manipulated in any number of ways before being added to the terrain. Additionally, the terrain can be easily changed with this system.  
One key advantage of the height map was the ability to perfectly mirror the terrain and make sure heights are uniform. Uniform heights in terrain are easier to read from the bird’s eye view perspective while mirrored terrain means the player should never feel at an unfair disadvantage.

### How

An image was created in photoshop that was then applied to the terrain. The image was a made from solid blocs of value that was then horizontally mirrored then vertically mirrored.

The max height of the terrain in unity was then set to 100 unity units high so the percentage value on how white the image was would correlate directly.

### Example

### C:\Users\Administrator\AppData\Local\Microsoft\Windows\INetCache\Content.Word\maperino.png C:\Users\Administrator\Desktop\22835328_1726846270690551_1418970550_n.png

An early prototype map and the final height map used for the game. The final heightmap is extremely dark as the terrain is relatively flat and uniform.

### evaluation

The use of height maps saved an incredible amount of time on the project and allowed for far more precise control over the terrain system.  
there were a lot of issues encountered in the making the height map firstly the exact image type is important and took many different imports to get correct. For example, a height map must have equal height and width be a greyscale raw file that with dimensions that are a power of two plus one to work otherwise some very strange terrain will occur (Heightmaps, 2017). Another delay to the process was that the original prototype shown above could not be used as when converted to the appropriate raw file and dimensions it still visual artefacts left over from jpg format which then became distortions in the terrain itself.

If handled differently the terrain would ideally be less about strict fairness and more about directing the player and easing in mechanics to the player. The current map is a very open area and could benefit from being a more linear focused experience.

Resources Used: (Heightmaps, 2017)

## BoX Selection

### What

Click and drag selection for units. This allows the player to draw a box on screen by holding down mouse button one. Releasing mouse button, one will select every unit within the selection box. This works the same as on a windows desktop environment.

### Why

Before this was implemented the user could only select units by clicking directly on to the target this would replace the current selection unless shift was held in which case the new selection would be added. This was an inefficient way to select multiple units. The selection box solves this issue in a way that computer literate people will be familiar with. Besides being an intuitive selection method outside of the game almost every real-time strategy game has box selection to the point that it is expected behaviour, similar to crouching in a first-person game. To leave out a mechanic like this would have to be a conscious design decision and this project has no reason not to include it.

### How

The first attempt at this failed as the selection was based on the position of the camera. For example, if the box is drawn in the centre of the screen it is accurate but objects near the edge of the screen may be missed as the camera has a degree of tilt and the perspective threw the selection off.

The next step was to create a selection that Selected units in relation to the player this meant drawing the selection box in relation to the screen and then checking if the units position relative to the screen is within the drawn selection box.

This solution worked and is the one in the game currently there were some however some issues. For example, if a unit is clicked and selected a tiny selection box with no dimension is drawn and when the mouse button is released then the box is completed and the current selection is replaced with the whatever is in the infinitesimal box. This turned off single selection unless shift was held to remedy this the selection of a single unit was changed to occur when the mouse button was released instead of pressed. This merely reversed the problem. The current solution is to place all units in to a buffer that will be added to the current selections after all selecting is done. This was a more elegant solution as it meant selection was only changed once per frame.

### Example

Selection video

### evaluation

The selection system took some time but is now working as intended the only issues that remain Is that the check on a whether a unit’s position is within the box occurs from a single point this means that a selection covering more than half of a horse can still not register as the centre of the object is not within the rectangle. Ideally this would be changed at some point in future development.

This system involved learning of how unity handles the creation of rectangles and the position of objects in relation to the screen, viewport and world space.

Resources used: (Zimmer, 2014), (83, 2012), (Korindian, 2014)

## Command Manager

### What

The command manager is a c # class in charge of the ordering and executing “Command Objects” that each have a different behaviour. For example, a command may be to move to a given location. The command manager allows these commands to be queued and take place one after another this is the core of all behaviour of in game units.

### Why

The command manager allows for the player to queue commands to happen one after another. This is useful as it allows the player a greater control of the unit as well as greater tactical depth. The player should be able to order their unit to a location and then tell the unit to defend it.

Like the selection box this is another feature that is a staple of the genre. It would be conspicuous by its absence, like a first-person shooter that didn’t allow you to reload when you wanted. The RTS genre is in terms of core gameplay about multitasking and micromanagement and the ability to give a unit a series of commands is useful. Additionally, the command manager doesn’t make the game any less intimidating to a new user as it is optional provided that the game is not balanced around its use.

### How

The command manager contains a list of command objects that are iterated through in a first in first out fashion. Each command has an execute() function which is called when the command is started which sets up all necessary behaviour for the command. For example, starting the objects motion towards its location.

The Commands themselves tell the command manager when they are complete

To avoid commands updating when they shouldn’t the command manager’s update method calls an update on the current command.

Finally, the command manager also holds a reference to the animator to easier manage synching animations with commands.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

As this class deals with commands there was research made in to the command pattern a software design pattern. While some concepts from the command pattern were used it was not entirely suitable. Command pattern has generic commands that are interpreted by specific objects. Whilst the commands implemented are specific commands interpreted by generic units. Additionally, commands in the command pattern should be reversible in the sense that one command will turn it on one will turn it off. Which this system doesn’t.   
This system does however borrow the use of the execute function and the idea of the command as an object

The current is successful there has been issues with adding a command into any position other than the back of the queue. A wrapper method had to be created to insert the command within the list and properly switch to that command.

Future development would ideally give the responsibility of setting the animation to the command manager and not the command itself. Additionally, the current set up allows for any unit to use any command, the ability to restrict what commands a unit can add to their command list.

This system was also useful for learning, specifically about LINQ and the command design pattern even if it was not used.

Resources Used:

## Commands

### What

Commands are objects given to the command manager and executed in order. When executed they perform a behaviour either indefinitely or until a certain requirement is met.

### Why

Encapsulating the unit behaviour has several advantages such as the ability to destroy, copy or order the different commands as you want.  
 additionally when the commands are objects it is much simpler to edit them in a vacuum without having to change multiple objects.

### How

After the command manager was implemented the first command added was the command base class at first this simply had a set of abstract methods execute(), commandUpdate() and delete().

The command started out inheriting mono-behaviour so that the command could make use of the get component method which returns objects of the given class attached to the given game object.  
 There were however some issues with that approach firstly that mono-behaviour objects aren’t intended to have constructors instead they are supposed to use the Unity Editor to define parameters before runtime and the awake() function to execute code when created.  
the Issue with this is that the commands should be created dynamically at run time which means parameters could not be defined at the time.   
so, the command objects had to be changed from inheriting mono-behaviour to inheriting from object.  
This allowed for constructors but also gave rise to another set of issues. The largest being that without the easy access to all scripts attached to the game objects. the commands then needed that information handed to them through the constructor. This meant that even a basic command like the move command would need around 4 parameters that would be repeated every time the object was created and for more complex object the number of parameters would only increase this wasn’t ideal.

eventually this was changed back to Using mono-behaviour. The core changes this time around was creating a substitute constructor. Each command was given a static method called New () (with a capital N). which will be given parameters specific to that given command including a reference to the game object it should be attached too. the static method will create the command instance which will cause the awake method to be triggered. Then various parameters will be set.

this method works well and is how the commands currently run this makes the timing of the command go: The Awake() method runs, the parameters are set, the start method runs then when it becomes the current command the Execute() method runs and then the command update method will be called every frame whilst it is the current command.

More changes were made after this including Getting the command manager and targeting classes in the base class and adding the Pause method and an isPaused Boolean.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

The command methods probably took the most time as so many revisions were needed. One of the biggest struggles for this component was figuring out how unity handles objects and how unity *wants* to handle objects as well as the timing structure in unity. The command objects being changed so many times also meant that the all the inheriting commands needed to be changed as well.

As such the command class is the aspect of the project that could have benefitted the most from additional planning even if a large amount of the issues with the class were only apparent later in the process.

Over all the command class is in a good place and adding new commands takes only a matter of minutes. The class is now where it was intended to be in terms of progress but was not expected to take so long.

The benefit to taking so long was the learning opportunity provided. Understanding how unity handles instantiation and timing. The use of the static creation method is a concept borrowed from the factory design pattern the factory pattern was researched half way through the implementation of this class.  
 As with the command pattern mentioned earlier this pattern has a lot of useful ideas but didn’t really fit the problem or the environment. Normal factory pattern calls are inherited from a base class but in this case different parameters were needed so that couldn’t be used however what could be used is the concept of a static factory style method that helped get around the instantiation problems encountered. The use of software design patterns in unity is difficult to formally implement but there are however useful aspects throughout the patterns that are applicable.  
in terms of future work on the system the command system is very much functional but perhaps not as efficient as it could be. If any additional changes are made to the class, they are low on the priority list

Resources Used:

## Control Groups

### What

Control Groups is a saved selection of units.  
the control groups let the player save all their units to a number key on the keyboard, generally by holding control and pressing the button. Then later the same selected units can be reselected by pressing the same number

### Why

This is once again a staple of the genre. This allows the player to manage their units more easily which is always a good thing. In a game that sometimes have hundreds of units its valuable to be able to select a certain group of units. For example, the ability to create a group of units to flank and be able to select them later without going to find every unit manually is a timesaver. The player should be fighting the enemy and not the selection system.

### How

This component needs to match a list of units to a corresponding list of numbers on the key board. The first step to this was creating a list that was then filled with number keys. Then creating a corresponding list of lists of units that were of course empty then in the update method that occurs every frame the class will iterate through every button on the button list and if pressed on that frame access the corresponding unit list index on the list of lists.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This feature once implemented worked well and has not been changed since. It works as intended and it is probably one of the most efficient ways to do such a task and the class won’t be changed at any foreseeable point.   
the difficulties creating such a class were in having were mainly matching the key presses with the lists of units. Initially the class had nine individually instantiated lists that were then going to be checked individually in the update method. The list of lists took some time to understand but provides a satisfying and efficient solution.

No additional work is planned on this system.

Resources used:

## Camera Locations and panning.

### What

These are additional controls and features for the game camera. The camera location system functions like the unit group system allowing a user to save a camera’s position using the F-Keys on the keyboard. The camera panning allows for the player to pan the top down camera by moving the mouse cursor next to the borders of the screens.

### Why

The camera panning is a feature so fundamental to the RTS genre that it’s absence was jarring. The lack of the feature is comparable to having a mouse that lacks a scroll wheel. On the other hand, saving camera locations is not nearly so crucial but does occur in a great many games. It allows for players to save areas of strategic importance for later and can speed up a player’s ability to move throughout the play space. The reason a low priority feature was put in was due to the implementation similarities to the control groups.

### How

The camera location implementation was very similar to the implementation of the control groups system. the system stores a list of camera hotkeys and a list of camera co-ordinates and iterates through the list of hotkeys and matches to a location. The camera panning was slightly more complex. as the camera was tilted slightly so was the axis of the camera. This meant that panning north zoomed slightly and panning south zoomed out slightly the solution for this was to create a camera cradle that was not tilted and parent the camera’s movement to the cradle. Then converting the mouse’s position to the viewport position and then adding velocity if the cursor was at the edges of the screen.

### Example

### evaluation

These features change the feel of play substantially and allow for a much more intuitive movement over the battlefield.

There were a few problems with the tilted camera and finding the borders of the screen. The camera location saving had issues with the initial values of the locations as pressing a camera hotkey would move the camera to position 0,0,0. The current solution is to set all camera hotkeys to the default start locations position.

Resources Used

## ACtion keys and rapid fire.

### What

Action keys refers to using a unit’s abilities or actions via a hotkey instead of clicking on them. There are three rows of four abilities. Each button should have a corresponding hot key in: Q, W, E, R, A, S, D, F, Z, X, C and V Order. Additionally, if the hot key is held it should fire many times in rapid fire.

### Why

Hot keys are found in almost all games like this a key difference is in the keys used. The majority of RTS games use the first letter of the action. For example, the move command hotkey would be ‘M’. the version implemented here uses the position of the button in the GUI and maps it to the left-hand side of the keyboard. This means that one only needs to know the location of the button to use the action as opposed to having a unique hotkey for each action. This should make it easier for newer users to become accustomed to using hotkeys without the unnecessary burden of learning.

The rapid-fire functionality is an intuitive behaviour. Holding a key to repeat that keys action is used in almost all desktop environments. It also reduces the strain of repeatedly pressing the same button to when performing an action multiple times.

### How

The basics of the actions manager already had a list of buttons that needed to be matched to the appropriate buttons in a similar fashion to the control groups method of doing so. The rapid-fire method was a little more complex as pressing the button even quite fast sometimes triggered the action many times. To remedy this behaviour a timer was implemented. like holding down a character key in a text editor, the action would trigger on the down press and then frequently at a set interval while held

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This component took a large amount of fine tuning and more issues arrived with commands that were not intended to be repeated.

The result is satisfying and eases play a large amount being able to queue the production of many units reducing the work the player needs to put in. some games have a focus on performing large amounts of menial actions to increase the number of actions and tasks that need to be taken artificially. A newer player should not need to match such a level of mechanical skill. Instead they should be focused on meaningful decisions.

Future development will involve refining the timing on the rapid fire and possibly limiting the actions that use that functionality.

Resources used:

## Click Confirmation

### What

Click confirmation is built upon the actions framework. Certain actions need a target. As an example, the if the move action button is clicked then the user will click a point in the game and the unit will receive a move command to go to that location.  
Click confirmation is a class to handle all the actions of this type without needing to know the all the buttons.

### Why

Certain actions don’t work without first specifying position or unit in the game. For example, if the player presses the attack button they will also need to select a target for the attack. This class enables half of the unit behaviour in the game.

### How

The click confirmation class uses a single instance of a class that is handed a delegate function the class is then “turned on” and the delegate is executed on the next click of the left mouse button.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This component is a good example of the use of the singleton design pattern used throughout the project. Other design patterns were difficult to implement the singleton design pattern is however far easier to implement in unity through the scene edition. The scene editor lets the user apply scripts to objects in the scene before runtime. This means an object in the scene can have a click confirmation script added then when all scripts are started at runtime the click confirmation script will set a static reference to itself. Avoiding some of the more complex methods of creating a singleton class

There were some small issues with this class the largest of these issues being that the action that click confirmation started only worked for the most recently selected unit. If twenty units were ordered to move to a location only one unit would. This meant the class needed a reference to the mouse manager that tracks the selections.

this feature is mostly finished some effects to give the player feedback when the perform an action would be ideal. Making the players current selections be more accessible would likely be another aspect to be explored

resources used:

## Input Controller

### What

The input controller Is a way to give units commands without having to click the buttons on their actions panel. The input controller will give the units different commands based on what unit is being commanded and what it is being clicked on. For example, if a peasant unit is selected and the player right clicks a location then the unit will be issued a move command. If a structure is selected a rally point will be set instead.

### Why

Having to click each action then choose where it goes would be far too time consuming instead a system that assigned commands more dynamically was needed. Having the decision be determined by the unit and selected means that a player can move units, order units to follow allied units and order units to attack with only the right mouse button. This is convenient and easier for the player to understand and reduces the number keys the player will be pressing.

### How

Firstly, a base class was created to have the abstract methods RightClickOnUnit, RightClickOnstructure and right click on resource node. Then two subclasses for structure and unit were created for to inherit these methods each with their own unique behaviours.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This functions well and feels very smooth. This feature in particular makes the game feel like an actual RTS game.

This component offered an opportunity to put OO concepts to good use in a way that did not feel contrived. Allowing for practice and application of previously learnt concepts

There were some issues with things like adding interactions to subsets of game objects. As an example, when a unit is given an order targeting a building then that unit needs to check if the building Is an enemy, an ally or a restocking building. This could potentially cause these classes to bloat over time if more subtypes of units and buildings are added.  
future development will likely include refining the way this class handles decisions ideally in a more efficient and elegant way

Resources used:

## Terrain assets

### What

Various 3D terrain assets such as cliffs, trees, rocks, resources and 2D textures. These assets add visual interest information to the scene.

### Why

The game looked extremely sparse and it was hard to tell aspects apart. The original terrain was a matte grey with the height map applied due to a lack of colouring made different terrain indistinguishable.   
The addition of cliffs and tree’s helps to define areas that area’s that are impassable for units or unsuitable for building construction. This is important for showing the player what they can’t do and not telling them. it is also important to have some degree of visual appeal. No matter how good the backend is if the front end looks unfinished then that is all the user will see. The terrain textures and rocks spread throughout the level work to this end.

### How

the 3D components were all built in Blender 3D whilst the 2D textures for the terrain and models were made in photoshop.

extrude, expand, taper method that was repeated up the trunk. The rocks are simple shapes copied. In a similar fashion, the boulders are large cubes heavily distended and distorted. And the cliff faces are hundreds of boulder models that were rotated and stretched to appear as a very rough wall of stone.  
the textures are simple matte blocks of colour built in photoshop. The colour blocks are then applied to faces on the mesh models or the terrain using the unity texture brush tool.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This addition to the project is probably the most apparent. the result is appealing and useful to the player allowing for a more intuitive read on the terrain to inform decisions. There were difficulties relating to how unity see’s the models once imported with some rocks or boulders appearing especially large or small when in unity.  
the colouring and modelling itself was time consuming and took longer than anticipated.

Another large issue was adding the models to the scene in the numbers required. Unity has a tree placement brush that allows the user to click and drag many trees at a chosen density across the terrain. This tool is undoubtedly useful and is what was used but comes with its own set of draw backs. Firstly, the placement is often a little too random making marking the limits of forested area’s difficult. Secondly, the tool only allows one tree to be placed at a time which makes variety difficult. This was remedied by using low tree density bushes many times over the same area and in general a more manual placement. Thirdly, when trees are placed using this tool they are at a rotation based off blenders edit mode rather than the model’s rotation in blenders scene. What this means is that if a single tree model is dragged in to the scene it will appear upright correctly. But when placed with tree brush tool the trees would appear lying flat against the terrain. The remedy for this is to go in to blenders edit more rotate the object leave edit mode and rotate it back and then hope you had rotated the right way. This took a lot of trial and error and had its own share of problems most notably, all the trees seemingly disappearing when they were in fact up-side down and on the bottom side of the terrain.  
finally, the tree’s placed using this tool don’t have any colliders meaning units can simply walk right through them. The current solution in unity is to use the Nav-mesh system which will automatically remove all path finding around the trees. This is ok except that the diameter of the impassable terrain around the trees was far larger than the tree’s themselves and could not be edited. The solution to this issue was to remove the generated Nav-Mesh and to instead add large invisible cube colliders to the edge of the map. This solution is currently a band-aid fix that only works as currently all the trees are at the edge of the map.

The cliff faces were a different issue in that they all had to be placed manually. Many games auto generate the cliff faces to save on time but developing that system would take more time than simply hand placing for one level.  
the cliff faces as they are now are probably the most visually appealing aspect of the game.  
the cliffs were grouped in a set of 4 boulders that was then grouped in to a set of 3 totalling 12 boulder objects 5 of which were put together to create one “medium Cliff” the corner of the map that the player starts in consists of 6 “Large Cliffs” this sky rockets the number of individual objects within a scene. Which was expected to be an issue surprisingly hundreds of additional objects made almost no difference to the performance of the game at runtime. The final issue encountered with the cliffs was the collision units cannot navigate past a wall in the terrain but the cliff face itself could be clipped into this meant adding colliders retroactively to all the objects which would take more time than simply removing all the cliffs and starting over with colliders factored in. to reduce complexity the colliders were only added to the sets of 12 boulders

Finally, the last aspect added was the colours applied to the terrain itself which took trial and error to get right as the terrain doesn’t look natural with large areas of one colour. It was decided that applying such colour to the whole terrain was not worthwhile and that only the players starting area would be coloured.

In terms of future development on these aspects there is a lot that could be improved. The tree’s lack diversity and could use an alternative method for navigation as the current method has some very strange behaviour on collision currently.  
Auto Generated cliff faces would be a good addition in the future but is low priority.  
the textures on the ground may work better if there is more complexity in the images than just matte colour as currently it is hard to read.   
overall these visual components are low on the priority list for improvement as the current implementation works well enough.

Resources Used:

## Gui improvements

### What

Creating a more visually appealing GUI for the user. Adding visual interest and information to the interface. The UI is 4 components the cash box, info box, mini map and action card. adding small touch ups and icons to these components.

### Why

the user will always have the GUI on screen so more than any other aspect it is important that it looks good. Icons provide the user to tell what action they will perform before they use it. The before icons were added the user had no way of knowing which button was which.

### How

The icon and GUI improvements were all added in Photoshop. The assets were drawn by hand using a drawing tablet save for a few areas using gradients and effects.

### Example

### evaluation

This component took a significant amount of time to complete but had become necessary later in the project as more actions were added. The result is clear and while not perfect the icons show the general intent of the actions they represent.  
  
the stop icon looks slightly different as it was made using shapes rather than drawn by hand. It is probably one of the least clear as to what exactly it does.  
The stone frame that runs around the GUI separating the different areas is took the longest off all the assets added in this section but the result adds a sense of credibility to the game. The core issue when implementing in the game itself was how scalable it was. When the game’s window size was changed the elements of the Hud became unaligned with the frame and occasionally overlapped one another. The current solution to this problem is fixing the aspect ratio to 16:9 this makes the elements appear in game exactly as they do in the scene but prevents flexibility in the scaling of the scene. Additionally, this creates another problem where the mini-map only correctly shows the position of units at certain resolution.  
Future work on these elements will aim to fix the mini-map bug and add additional icons for the place building actions which currently have no icons.   
Improving the portrait images and the mini-map image itself would also be good. The current mini-map is just a top down screen shot of the actual game. the info screen does not show as much as it could and the current portrait is outdated.

Resources Used:

## healthbars

### What

A health bar to give the user at a glance information about the state of their army or structures. The health bar is generally a progress bar representing the current health of your units often with colour to show the health faster

### Why

Before health bars were implemented the only way to tell the health of a unit was to have that unit selected and only one unit at a time. This meant that in larger fights it was nearly impossible to tell which side was winning. If the player can look at their army and see mostly green they know they can continue fighting. If they are in a one on one scenario with another unit they can simply watch instead of switching between the units to check which side is winning.  
Finally as with most other systems added to the game so far health bars are an expected mechanic and to exclude them would have to be a conscious choice. If a player is allowed to know the health of a unit then they should have access to that information as clearly as possible.

### How

The health bar is an image that is filled in from left to right on top of a background image. The two images are then faced towards the camera every frame. These aspects were easy to implement as unity has methods for filling a bar and for quickly facing an object to a point.  
Past that, the health bar script was changed so that health bars are only visible when relevant. If a unit has less than full health the health bar is visible whereas it is otherwise invisible. Additionally, health bars always show on units that you are selected. Finally left Alt will show all health bars while held to allow the player more control on viewing

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

Another key aspect was to make sure that the health bar could be read at a glance and inform player decisions. There seem to be three ways this is done in modern RTS games

* The health bar changes colour to show how close the unit is to death.
* The health bar Drains the recent Damage off the bar slowly but deals it immediately. This is a good way to show the rate of damage
* The health bar changes in some fashion once it reaches a “low health” threshold. This is generally to alert you to do something about it.

It is possible to have multiple of these but for now the current method is colour changing as this can show the current health at a glance of all the units on screen.

There were some issues with interpolating colour. Interpolation works like a slider from A to B and is given a percentage for example 0.0 is A, 1.0 is B and 0.5 is halfway between A and B.

Unity has a method for interpolating colour but there are some issues with it. Firstly, it only goes from A to B using RGB. Secondly, in the case of the green to red in the health bars it is trending each A RGB value to each B value. That does not create a nice green yellow red Transition that game literate people are familiar with. Instead it outputs a muddy brown in to an off red. This is because it is just hue shifting without respect to other aspects of colour. there are more complex and complete solutions but as a focus of this project is also learning, importing a large solution was avoided. Instead in a simpler solution for the problem was creating a static method in the utility class that interpolates between three Colours. This was achieved by interpolating twice. First, from A to B and then B to C. this ensures that 50% health displays a vibrant yellow and makes the colour transition far smoother.

The health bar is currently in good spot currently. If more were to be added it would involve adding the draining effect from above to show how fast a unit was dying and revamping the visuals of the health bar.

Additionally a progress bar added to units and structure to show their progress through continuous actions would help the player with organization.

Resources Used:

## Attack and attackswing Command

### What

A command that tells one unit to attack another unit. The command should move the unit to within the attack range of the target unit then cause them to attack at a set interval. Alternatively if an area in space is selected the unit should move to that location attacking all enemy units on the way to the destination.

### Why

Attacking is the main method of conflict in this style of RTS. This makes this the core mechanic in this game. The game needs methods to reduce the health of enemy units. Previously, this was done by every unit damaging enemies in range on a cooldown. Having the player able choose which unit is attacked gives more control to the player.

### How

The class that chose targets for the old method has been changed to hold an “aggressive” Boolean. While this is true the unit is considered aggressive and will seek out targets within range and issue an attack command towards that unit. The attack command will move the unit towards the current target and will follow the target if they move.  
all units can have one or more weapon scripts attached to them. Weapons have fields like damage, attack speed and range. Weapons have a canAttack Boolean that is true if enough time has passed between attacks. When it is started the targeting script will check every weapon and find the one with the longest range, whenever the target unit is within that range the targeting script will attack if able. every update the targeting script will check if any of the weapon scripts canAttack fields are set as true if they are that weapon will add a new command to the front of the command queue.

The new command is the AttackSwing command this command oversees making sure no other behaviours happen whilst the unit is attacking. For example the Unit moving whilst it should be standing still attacking. The AttackSwing command has a duration after which it will move on to the next command. This duration is there to allow for an attack animation to happen or for the player to change their mind and issue another command.

there is a second timer within the AttackSwing class that determines how long of a delay before damage occurs so that a user cannot rapidly issue attack commands and deal a large amount of damage. Additionally it allows for the designer to better synch the damage to the attack allowing for better player feedback.  
after the attack swing occur the weapons canAttack field is set to false and will stay that way until more time than the weapons attack speed has passed.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This class was probably the most time consuming after the command queue. As it is not as simple as it seems.

There were a great many unintended behaviours in this class that provided opportunities to provide interesting solutions.  
one of the first issues was that an attack command would keep firing even if the unit had already been destroyed. So the Attack command needed checks to see if the unit was dead.

After defeating an enemy a unit would simply stand in place this lead to the addition of the aggressive Boolean on the targeting class. every command when executed sets the targeting classes aggressive field to true or false. This is to stop units from attacking when the player just wants to move from one place to another. A clear example of this issue is that for a time all fights in the game were to the death. If a unit was issued an order away from a fight the unit would immediately look for a target and find the unit it was fighting and turn around to attack.  
Counterintuitively the attack command is not aggressive. As otherwise the unit will constantly change which target that it intends to attack.

If the command manager has no commands it will set targeting to aggressive so that units can defend themselves. One issue encountered was that if a unit found a target it would chase it forever and if the target died it would stand where it died. To remedy this the command class was given a leash distance field. By default the leash distance was 10000. The class was then given a second pseudo-Constructor.

One pseudo-Constructor would be used when an attack order was given targeting a unit and the other was used for attacks that were triggered automatically. The former pseudo-Constructor is given only a target and used the default 10000 distance leash and the unit does not return upon killing the target.

the latter pseudo-Constructor received a leash distance and if the unit went over this it would walk back to it’s the location it started at. This unit walked back to its original location after killing unit as well.

An issue with this is that the unit would kill one enemy and walk back the way it came while being attacked by any other enemy units that happened to be in the fight. Therefore a check was added so that after killing a unit the killer would be issued another attack command targeting any remaining enemy units in the area.

This worked well but caused another issue. If a unit killed a unit then found a new target it would reset its leash location to the position it killed the first target. This could repeat as well allowing a unit to be pulled across the entire map. So a third pseudo-Constructor was added that was only called after killing a unit this would keep the original attack commands leash location

The mechanic now works well and there has not been any aberrant behaviour in the latest version so far.

future changes will include moving the duty of firing the weapons to the weapons themselves as currently the targeting class Is bloated. There are also no ranged weapons as of now which may produce more issues when they are implemented. Finally the weapon hands a long list of parameters to the attack swing object refactoring that to less parameters would be ideal.

## Move, patrol and follow commands

### What

These commands are focused on moving a units position and all function a little differently. The move command will move a unit to a point in the world and is either aggressive or non-aggressive.

The patrol command will move a unit between two points in world space aggressively which is useful for defending an area.

The follow command will make a unit follow another friendly unit wherever it goes and is non-aggressive.

### Why

The move command is a basic aspect of the game and similar functionality is a core component of almost all RTS games.

The patrol command and follow commands are options for the player that allow for more control over the way the units move. the follow command to follow a unit already given direction and the patrol command to have a unit guard a large area.

### How

All three commands make use of the unit nav-mesh system which handles pathfinding.

The move command simply moves to a point then goes to the next command in the list. There is a pseudo-constructor to give the option to make this command aggressive causing the unit to attack all enemies on the way to the destination. This is used when the attack action button is pressed and a position is clicked instead of a unit.

The patrol command is almost identical to the aggressive move command except that it paths back the original location upon reaching its destination.

The follow command is a move command that has the destination updated to the position of the unit it is following every frame.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

These options make for more diversity in options for the player and options are only an issue if there are too many.

Future additions to this class may change the follow command to be aggressive to become more of a guarding command

## Stop and Hold position command

### What

Stop and hold position are very simple commands .

stop makes the unit stop whatever it is doing and then immediately goes to the next command if there is one.

Hold position makes the unit navigation obstacle and lasts indefinitely.

### Why

Stop is useful if the player changes their mind about what they want their units to be doing. The command is a good way to clear the command queue without issuing another unnecessary command. The stop command is also aggressive so if units are moving some where and being attacked hitting stop can be a good way to get them to attack the nearest target

The hold position command is useful as if a unit wants to move through another unit, it will push any units out of its way. The hold position command turns the nav-mesh script off and turns on the nav-meshObstacle. This means that the commanded unit can no longer be moved past. This command allows the player to ensure that a unit stays in one position without leaving to attack. The command can also be used to block off areas by having a wall of units hold position.

### How

The stop command does almost nothing it simply is a command that clears the command queue and makes the unit aggressive it is one of the smallest classes in the project with less than 30 total lines of code.

The hold position command was slightly more complex as it required that all units have a nav-meshObstacle that is off. This needed to be added to all the units. Past that point it simply toggles to features on execution and on destruction

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

These components were added later in the project and by that point the process for creating new commands had been streamlined they turned out well.

The only issue with the classes as they are currently is that the units do not attack enemy units in range whilst using hold position this is because most of the attack framework is based off of commands there is as of yet no way to avoid this but that is an aspect to be looked at later on.

Resources used:

## Gather and channel command

### What

The gather command causes a unit to gather resources from a resource object and take it to a nearby depot.

The channel command is a command for any action that takes time to complete

### Why

There are many ways for games RTS games to generate resources. For this project one that is tactile and transparent is preferred. In the game the player spawns next to resource nodes that a peasant unit can collect resources from and then take back to the base. The resources are then added to the players current balance.

This method makes it clear to the player what the units are doing and exactly how fast the money is being generated and indicates what the player should do to earn money faster. Additionally having the units collect the resources offers a weak point for the player to target.

This mechanics inclusion was not a matter of if it should be included but how it should be included.

The channel command can be used in many places and is very similar to how the attack swing command works in that it helps control timing of actions and behaviour. it allows for generic re-usable actions with a set duration.

### How

Gathering has 4 steps. First, the peasant unit goes to the resource depot which is the closest building with the resource depot script attached there the peasant unit is assigned the resource node with the least peasant units currently on it. Second the peasant unit is issued a move command to the node. Third, the unit is given a channel command at the resource node which plays an animation to interact with the node. Finally, the peasant unit is given another move command back to the base where it loses the resources and they are added to players account balance

The channel command is handed three pieces of information. First a delegate method to be executed after a time. Second, the amount of time before the delegate is called. Third, the amount of time that passes before the command ends. The reason the two different timers is to allow a player that Is micromanaging a unit to save time by giving a unit a new command after the channels effect has happened but before the next command duration is reached. Another delegate method may optionally be added that triggers if the command is cancelled before completion of the channel.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

The gathering component is probably the most visually compelling unit behaviour. The command as it is currently works very well and is very intuitive.

The channel command was added quite simply as it was structurally very similar to the attackswing command. It would be ideal to have both classes inherit from the same superclass or to have the attackswing command inherit from channel. The only real change was the option to add an additional delegate for when the command was cancelled which is an issue that arose during the gather commands development.

Adding the gather command did have its issues. A lot of decisions were made under the assumption that is ideal for the resource gathering to be as consistent and predictable as possible and reasonably balanced between players.  
to this end the number of peasants working at each resource node was limited to three this was to stop one player from snowballing an economy. Especially the AI. As an example if there are no limits on resource gathering the player with the most peasants will get more money and can afford more peasants to get more money this could spiral out of control fast  
by adding in a limit to mining peasants a player that is ahead doesn’t achieve an exponential economy they will just reach maximum income faster

After this addition, it became necessary to divide the peasants equally between each node.   
after this point a new issue was encountered in that the three peasants on each node would run in to each another while mining making the resource gathering clumsy and inconsistent.  
the solution used was simply to minimise the unit collision of the peasants whilst they were mining allowing them to pass through each other as necessary.   
this of course caused another issue where units would stack up on the same position and only look like one unit this made it hard to read how well the players economy was doing the fix to this was to have every resource only allow one player to use it at a time. This works well as assuming that moving to and from the node take one second and the channel command takes one second as well there should be one peasant mining, walking to and walking from the node at any given time.

Future work on this will mostly be quality of life improvements for the system.

One such improvement would be to add numbers above each node displaying how many more peasants can work it.  
  
Another addition would be to replace the move command between the nodes with a new command that interpolates between the node and the depot over a one second time frame. This would allow for a more perfect mining flow. Preparation for this feature is why the peasants walk to the depot prior to mining  
additionally currently non-peasant units can mine which is unintended.

Resources used

## Production Queue

### What

The production queue is a way to control how many units a player can make at a time. The queue has five slots for items to produce the first item in the queue will be built over time and will have an effect when completed. As an example the main building the player makes can build a peasant in 10 seconds and can only build one peasant at a time. The player can queue 5 peasants that will be produced automatically over 50 seconds

### Why

First and foremost, this is once again a staple of the genre and a tried and true mechanic. This would as with many other mechanics typical of RTS games would have to be a deliberate design choice to exclude it.

This is good at limiting the actions of players especially the AI with a set build time the AI cannot build units faster than the player. Limitations like this allow place a ceiling on how efficiently a player can perform a given task which means that a level can be created with the timings as all players would produce units at the same time.

The queue helps players that forget to consistently produce units by allowing them to queue their production. This is beneficial for the same reasons as the command queue.

### How

There were two classes added to create the production queue as well as some changes to the GUI.

The GUI changes requires 5 buttons and a progress bar which took a lot longer than anticipated. This was mostly due to making sure that the most recently selected structure was showing its production queue properly.

The first class created was the Production item class this class contains information on the item being produced such as cost, time to build, icon, a delegate to be called on execution and a delegate to be called on cancellation.

The Second class created was the production manager which is attached to every unit or building that produces something this class contains the list of production items and oversees progressing the first item in the queue and removing it once finished. The class is also in charge of making sure that the buttons on the GUI properly display the image set in the production items field.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

The production queue is working correctly but aspects of it are a little sloppy but nothing serious. There were a few problems encountered throughout this process most of which were fixed.

An early bug would instead of removing items from the building you had selected would remove items from the queue of the last building selected instead.  
another issue was an unintended secondary check of an items cost that would cause completed items to take no effect if the player didn’t have enough money to queue the item. As an example, if a player has 100 gold and queues two peasants at 50 each the peasants will not spawn if the player still has less the 50 gold at the time they complete production.

Items are removed from the queue by clicking on their icon in the production queue. There was an issue at one point where clicking the first item in the queue would remove the second item and clicking the second would remove the third. This would seem to be a minor indexing error except that clicking the third button removed the first item. This was eventually solved but the root cause is still unknown.

Future work will involve delegating tasks to the least busy producer when multiple are selected. For example, if three peasant producing buildings are selected and three peasants are bought, the most recently selected building will queue three peasants.

Resources used

## Equipping units

### What

this is probably the only feature that does not appear in the majority of RTS games. This is alternative way of producing a variety of units. The player will build peasants that can then equip other items at one of three structures. The items are a sword, a shield and a horse. The structure can produce an item in the same way as a unit would be made. Then when a unit is next given a right click input targeting that building the unit will equip that item if possible. For example a peasant can equip a sword and become a swordsman, a swords man can then equip a horse and become a mounted swordsman finally the mounted swordsman can then equip a shield and become a knight with a sword, shield and horse.   
Alternatively a peasant could equip a sword then a second sword and become a berserker.

### Why

The reasoning behind producing units this way is to make the unit creation process intuitive. A newer player can make a sword, a shield and a peasant and then combine them together to make an infantry. This is intended to make unit choices easier to make. If a player wants a unit to do more damage they can give that unit swords. If they want to make a unit durable give it a shield. Similarly a horse will improve speed.

### How

Two classes were created for this component. Firstly a stock manager a class that kept track of how many of each items had been produced for each building additionally it keeps track of what each unit will become when it equips one of the items this class can be customised in the editor.

The method for matching the equipping units to equipped units is like the method used for the control groups in that it’s a set of lists that are matched by indexes.

Secondly an action that creates a production item that adds one stock to the stock manager.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This class actually didn’t take a long time to create and worked really well when finished and is one of the more interesting mechanics in the game as it is one of the only unique mechanics in the project.

The largest issue encountered took far too long for what was essentially a syntax error. When the unit was given a right click command targeting the building with the stock manager the unit would be given a move humanEquip command.This command is supposed to give a move command to the unit to move next to the building before becoming a new unit. Instead of putting the new move command at the front of the queue it was being inserted behind the humanEquip command meaning that more kept being added until the game would crash. This error took an inordinate amount of time to fix considering its simplicity.

Future work will involve copying the command list of any units that are replaced by upgraded ones and making sure that any units that replace a selected unit are also selected.

Resources used.

## Animations and models

### What

These are models and animations for the various units in the game.

### Why

Unit models and animations are visually appealing but also useful to game play. Animations that match a units current actions can inform a player about what a unit is doing at a glance.

### How

A units animations should synch up with what they are currently doing fortunately the command system put in place means that the game knows what any given unit is doing at any time.

The units and models used for the units were purchased off of the unity asset store to save time. Other assets were created to make the project entirely independently built. Which ended up using a lot of time. The results were good but could have been achieved by using prebuilt assets.

One class was created that contains references to animation clips for different activities such as run, walk, attack, idle. These can then have an animation clip dragged in to the editor.  
then a command will call an animation clip from that set of references and play it. As an example, a peasant unit will have an animation clip placed in to the run animation slot on the new class. Then when a move command is started it will play whatever clip is stored in the “run” reference.  
this method allows for the commands to stay generic and means an animation controller does not need to be added for each new unit. Instead the designer only needs to drag and drop appropriate animations in to the slots available.

### Example

If applicable an example will go here showing the component working or in place in the game

### evaluation

This component makes the game far more impressive. But that is because the models and animations are high quality assets.  
Unity does have an animation system in place with the animation controller system. animation controllers are a form of state machine that can dynamically transition between animations based of several factors. This system is powerful but it is also time consuming. Animations were one of the last things added to the game. to devote more time learning and creating a framework to support the animation controller seemed wasteful. Especially when the command system is already an accurate view of the current behaviour of a unit.

Future changes to this system will include death and on hit animations being made use of.

Potentially other models could be added to represent the structures in the game.

As it stands the visuals of units as they are now is appropriate for this style of game.  
  
In the process off adding the animations in was a valuable learning experience in how Unity handles animations and what unity considers best practices for Implementing animations in to its games.

Resources Used:

## IMplementation summary

This concludes the implementation section note that this was not an exhaustive list of implementation steps taken. Just some of the more interesting or noteworthy issues and additions.

# Evaluation

This project is overall quite successful despite not producing a fully functional game. The project had varying levels of success depending on the metrics used.   
The game did not reach far enough in development to make concrete conclusions on player accessibility. It did however produce a many fundamental systems that are crucial in producing a game that does answer such questions.   
The initial intent of the project was focused on creating an accessible RTS game. The ability to explore that topic was limited by underestimating the sheer volume of systems and mechanics needed to achieve the expected minimum for a game to be considered part of the RTS genre.  
It was anticipated that development would take longer than expected because software development always does. What was misjudged was not how long specific systems would take but rather how many systems there are in a game such as this.  
it can be argued that not all the systems are necessary if accessibility and that focusing on basic functionality would provide better results.  
the problem is the genre has been making quality of life and accessibility improvements for decades. Even if there aren’t yet enough considerations for the less experienced player, not including those features and systems would move the project in the opposite direction.   
instead the accessibility aspect of the project became about applying it within the systems that were added rather than adding systems for it.  
How and where these considerations were made is detailed in the implementation section.  
overall the progress made in terms of accessibility is good but unfinished. The project is at a good level of development for continuation.

The second metric for success of this project is the learning opportunities created through its development. The two major areas of learning were in specialist software such as unity and blender 3d and in object oriented development.  
Unity is a high-level game engine, this doesn’t mean that its particularly good just that it has systems and features that handle a large amount of basic game systems. Learning about the  
Nav-mesh, camera, terrain, height-mapping, animation controller, and UI systems. Can be applied to a wide variety of projects and offer fundamental skills that are transferrable to similar systems in other development software.  
blender 3D offered insight in to how models and textures were used in game development rather than how to create such models or textures.  
  
Creating the game offered a great opportunity to apply object oriented theory to a practical development project. This helps to foster a better understanding of why and where some more abstracted coding practices should be used. For example, the singleton pattern is used throughout the project to great effect. Prior to the project the developer was unsure exactly how the pattern should be applied or created.  
Additionally the game’s many different systems provide a wide variety of opportunities for different and interesting coding solutions. For example, the use of lists of lists in the control group manager class is very different to the use of delegates within the production manager class.  
one aspect that didn’t get enough attention throughout development was controlling the access level of classes. Trying to maintain privacy of variables was particularly difficult as values the editor cannot be changed unless the variable in the scripts are public.  
overall the learning gained from the project allows the developer confidently say that they are proficient in both C# and unity and have a solid understanding of object oriented development, which couldn’t be said before.

The project could have progressed further if there was more liberal use of imported assets. Too much time was spent on custom built assets which ended up being unnecessary.

The project was a good start to creating an accessible RTS game and an excellent learning experience. The end result can is playable but is very much a proto-type or a proof of concept rather than a fully-fledged game. This is a more than acceptable result for the given timeframe.

# Future Development

There are a great many areas of the project that could use additional development aside from those mentioned in the implementation step of this report.  
there are three areas that would be the focus of further development additional systems, refactoring and additional assets.

Firstly there are a slew of systems and mechanics that were left out due to time constraints.  
A tech tree system to restrict what units and structures can be built.  
A system to add upgrades that effect all current and future units for the duration of the match. This system when implemented will have a unique spin on the upgrade mechanic. Making the upgrades function a choice between one of a few options that disallows others. This means an upgrade can have a greater effect as the number of bonuses that can be stacked up is limited. Which allows for more dramatic easy to track upgrades like “attacked units take damage over time” rather than a slew of small numerical advantages.  
  
a key system for many intended additions would be the inclusion of an event system for significant unit occurrences. This would track things when a unit attacks, is attacked, kills or is killed. This would allow for mechanics such as gaining money when an enemy is killed or making every third attack deal more damage.

Improvements to the AI decisions or multiplayer would be next. allowing for a challenge to be overcome and turning the project in to a full functioning game  
past that additional content would be added in the form of more units structures and abilities.  
  
 those aspects would also require additional assets. The game is currently lacking images for units and icon for structures implementing these would be first. Past that every new ability, unit and icon will need its own 2d art, models and animations.  
research in to lighting techniques and effects can add an invaluable level of player feedback to the game. There are variety of other non-system assets not mentioned such as sound, music or menu’s.

There are various areas of the game that need to be refactored but were left as they were functional and quality refactoring is time consuming.

By in large the number of features, systems, and assets Are extremely numerous and not all are listed here. Future development is intended and will continue for some time.

# Conclusion

This project was initially intending to develop a RTS game that was accessible to a wider variety of audiences than the genre traditionally appealed to. This was to try and identify ways to overcome issues in this area that are considered a cause for the genre’s decline.

The project could have benefitted from better initial planning past what was done. However, the milestones laid out did provide a useful reference for what should occur next in development.

The projects implementation took the overwhelming majority of project time with work on this area ranging from adding game systems to adding assets. The implementation that was finished was very successful if only a section of a much larger whole.   
Some aspects received much more time relative to what they added compared to others. Assets should have been outsourced where available to save time, however the assets that were added are visually appealing and successful.  
The quality of the systems and mechanics added vary. With some being almost entirely complete and others needing some base level changes in the future.  
There is a large amount of potential additions for future development.

The project was a valuable as a learning experience if not as an experiment in real-time strategy accessibility and is considered overall to be successful.

# Bibliography

83, B. (2012, July 21). *RTS rectangle selection system*. Retrieved from Answers.unity3d: http://answers.unity3d.com/questions/287406/rts-rectangle-selection-system.html

Evans-Thirlwell, E. (2016, March 11). *PCGamer*. Retrieved from PCGamer: http://www.pcgamer.com/the-decline-evolution-and-future-of-the-rts/

*harrysgamedevresearchblog*. (2017, 10 24). Retrieved from wordpress: https://wordpress.com/view/harrysresearchblog.wordpress.com

*Heightmaps*. (2017, 10 06). Retrieved from Unity Manual : https://docs.unity3d.com/Manual/StandardShaderMaterialParameterHeightMap.html

Korindian. (2014, August 31). *RTS Style Drag Selection Box*. Retrieved from Forum.unity.com: https://forum.unity.com/threads/rts-style-drag-selection-box.265739/

*List of RTS Games*. (2017, 10 21). Retrieved from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/List\_of\_real-time\_strategy\_video\_games

MOSS, R. (2017, 9 15). *ArsTechnica*. Retrieved from ArsTechnica: https://arstechnica.com/gaming/2017/09/build-gather-brawl-repeat-the-history-of-real-time-strategy-games/5/

Sagmiller, D. V. (2016, 3 16). Build a realtime strategy game with unity 5. *Build a realtime strategy game with unity 5*. Lynda.com.

Zimmer, J. (2014, April 28). *RTS Style Unit Selection in Unity 5*. Retrieved from Hyunkell.com: http://hyunkell.com/blog/about-me/