CS3IP – TURN-BASED STRATEGY GAME DEVELOPMENT

Contents

1 - Business Context

Education. The cultivation of one's wealth of knowledge in a controlled environment – and for many, education is a large part of our lives. Education is an essential part of human culture when it comes to success, however the terms boring, monotonous, and similar negative insinuated words, are often used in when association to their experiences in their youth.

The project that is being proposed serves as a solution to mitigate some is the grievances, youths experience with education. This is done through combining education with entertainment to alleviate some of the misgiving's students have with their learning environments. Games will be used as the medium to foster an enjoyable educational environment for the users, to develop essential soft skills that may otherwise not be taught and developed in their study space.

1.1 – The Problem in Depth

To put the problem into perspective, 75% and 60% were values for two conducted case studies pertaining self-reported negative feelings towards school, with 'stressed' (79.83%) and 'bored' (69.51%) being the most prevalent [1]. Whilst the studies are by no means representative of the entire population, it does give an indication that a majority of students undergoing education have clear dissatisfactions with the methods employed with teaching. This has several implications: with the most critical point being a decreased drive and motivation for studying – this in turn reduces learning retention, and performance in school, which is counterproductive, being the exact opposite of what education is aspiring for.

Entertainment is perceived as a means of recreation, serving as a method to cure boredom, as well as relieving some accumulated stress; making it the logical solution to address the issue.

1.1.1 - But why Games in particular?

The foremost reason is very evident – the large consumer base of the gaming industry. 3.24 billion [2]. That is the number of recorded people who have been identified to carry the title 'gamer' in 2022. 7.98 billion is the current approximation of the global population, suggesting that approximate 4 out of 10 people have experience with video games. It's also worth noting that this statistic doesn't factor in that 1/3 of the world doesn't have access to the internet, which implies that the proportion of people who play games would actually be much higher if taking into consideration the availability of the option.

'Since 2012, the video game industry has increased in value in eight of the last 10 years. In fact, the gaming space is now over 3x what it was in 2012 ($31.23 billion).' [2]

Currently sitting at a net worth of $106.8 billion, the exponential growth of this market sector is a testament that the video game industry a hotspot for general interest.

Now then what does this all suggest? Based on public interest alone, using games as a medium is a feasible concept. This fact is further cemented when taking into consideration that 70% of gamers aged between 8-15 treat it as a daily routine [3], suggesting a high degree of acceptance from an age group considered as part of the target audience.

1.1.2 - Aren't there other alternatives that could be employed?

From an objective standpoint, whilst there are many available means of entertainment that could serve a similar purpose [4] – that number is greatly diminished when considering the technical viability of implementing. The most crucial aspect to consider is interest, therefore niche hobbies such as stargazing or theatre are not applicable options. The number can be further whittled down based on practicality – travelling is an example that is unrealistic to implement in a consistent manner when factoring time and expenditure.

Based on this, the most viable options are games, music, and reading. Each of these items are very subjective based on the individual in terms of interest in genre. However, overall games grants more creative liberties towards the scope of development, being also able to integrate both reading and musical aspects into the mix.

1.2 – Direction of the Project

The basis of this game intended to be developed is from a critically acclaimed game many know as 'Chess'. Why chess? It's an existing game that is already used as an educational tool to a certain degree, meaning it's clearly doing something right – in addition, it would be more socially accepted based on relevancy.

Chess is a game that develops several fundamental skills that we as potential members of society will one day need. *Visual memory*, *cognitive abilities*, *competitive skills*, *creativity*, *attention span*, *risk taking*. These are all essential skills that can be developed from a simple game of patience. Medicine also has relevancy in that regard, with chess being used as a therapeutic treatment to ADHD, which has plagued education.

Then will the game be a recreation of chess? Yes, but no – the project is intended to use the underlying premise of the game and take it one step further. At its core, what chess is, is a 2-player turn-based strategy game (TBS for short), which is the intended development route. The aim is to enhance the previously mentioned skills to a higher degree by developing a more complex and less restrictive system of traditional chess – this will take inspiration from successful case studies of games from the same genre (Fire Emblem, Advanced Wars, and Civilization are prime examples).

1.3 – What are the implications and risks?

Subjectivity of interest is the key point of consideration; a lack of interest will cause a relapse of the issue that the project was intended to address. Chess is a strategy game at its core, and this is a point of concern, with the sales of the genre being 3.7% based on studies in the US in 2018 [5] – to a lesser extent it can be considered a niche genre which Chess also suffers from.

Fortunately, video games do not have to be restricted to a single genre, so this issue can be mitigated to a degree by including elements that would enhance interest value towards the game without detracting educational benefits offered from the project. Fire Emblem a popular Japanese role-playing game (JRPG) can be used as a case study here, being a relatively successful franchise in the strategy genre that integrates role-playing elements, in the form of compelling narrative, and action through its strategy integrated combat systems.

The concept of themes can be treated as another area that requires attention, which happens to tie in towards the notion of genres. If a genre can be appraised as a broader perspective of collective ideas, a theme is one of those ideas, that is taken as a subject to develop around. For a game, a theme can be expressed as various topics, the ambience or environment of a game, the gameplay that can be expected, or even the art style. Now then, how does this tie into risks? We live in a society where even the most mundane topics can be treated as offensive depending on the context, naturally as a game that can allegedly designed to be employed as an educational tool, the choice of themes can be a breaking point, after all censorship and age ratings exist for a reason. Obvious themes that need to be avoided are gore, adult themes, and even political agendas. Even to a lesser extent, themes of violence can be treated as a theme that should tread with caution despite being commonplace.

Another point of concern is time. Strategy games can be difficult to develop depending on the complexity of the proposed systems, a lot of logic will need to be implemented, with an equally high amount of testing required to maintain what is perceived as *'balance'*. Balance ensures that systems are less likely to be abused to cause unintended effects that could detract from gameplay experience.

Whilst not the most ideal scenario, this issue can be addressed by breaking down the intended systems to its simplest forms. Ensuring that the core gameplay is kept intact allow for the basic functionality to be implemented within the allotted timeframe. This places priority on basic functionality, with more advanced features being possible development additions based on the permitted time available.

The public perception of games being associated with education is another topic that may negatively impact the project. There exists controversy between games and the negative influence it has towards education, with educators and parents stereotypically being on the opposite side of the fence to students who enjoy their past time of gaming. Games are typically seen as a distraction or have a negative influence on behaviour such as encouraging violence. Whilst these views have lessened with recent times, they still exist which may raise some opposition, especially for environments that will affect future generations, such as education.

The game Chess was chosen as the basis for the project specifically to address this concern. Chess is an existing game that is adopted as extra curriculum for many schools and is also accepted as a professional hobby due to the extensive and valuable skillset it helps develop, with many being crucial skills for certain professions. The game was also intended to nurture skills that aren't traditionally taught in a practical environment providing a high degree of educational value. It's intended to simulate an environment that forces critical thinking, potential risk taking, informed decision making, all in a controlled and competitive environment – all of which are important skills for society. Taking Chess as a basis alongside the development of core skills will help alleviate some worries.

1.4 – Choice of Technology

Unity is the engine will be employed for this project for three key reasons.
Firstly, I'm a complete novice when it comes to game development. Unity has one of, if not the extensive game development communities available – this means more support and learning material.
Secondly, Unity is more versatile, not only being more developed and easier to debug than

most other engines, but it also supports both 2D & **3D**, cross-platform, and a plethora of frameworks, let alone the large community supported asset store. This provides more creative freedom and flexibility on towards the approaches to the game.
Finally, one of my modules is game development which also ties into Unity. It's a no brainer to choose Unity, where I can potentially get support from a certified lecturer, as well as tie the skills learned from both modules.

One of the main concerns is the programming language Unity supports – C#. This is a language I've never had experience in handling, making it point of concern especially with time being a valuable resource. Fortunately, C# is considered to be a very similar language to Java, which I fortunately have used many times in the past. Both being object oriented languages should help expedite the learning and development process.

1.5 – Target Audience

The expected end-users will be members of full-time education, meaning people within the age range of 3-18 at the bare minimum, with anywhere up to ages of 30 if factoring in further education. This means the target audience will range between children to young adults overall. The expected environment the product will be used are study spaces, with teachers and parents being additional potential end-users intended for educational purposes of students/children respectively rather than self-use.

Secondary stakeholders to take into consideration are teachers, parents, as well as the boards of education – as they opinions take precede in terms of implementing the product in educational environments, as intended. Therefore, it's also necessary to cater the product towards their interests.

1.6 – Aims & Objectives

The overall intention of the project is to:

> 'Develop a game that could be employed in an educational environment to develop core skills such as critical thinking, or decision making.'

The ambiguity of this claim is the method involved with determining success of the project, as without a measurable goal, success or failure becomes obscure.

As a game, there are fundamental elements that need to be considered to determine success. Impressions and engagement of the game are some of the most critical factors, if a player does not enjoy the game, the purpose of the game loses its meaning, as no one would play it.

Impression can be straightforwardly measured through a questionnaire for qualitative data to cross-examine, this can be done by creating a set of scaled questions, such as "On a scale of 1-5, did you enjoy the game?". Similarly, quantitative data can be acquired to obtain opinions as feedback such as "What did you dislike the most about the game?" The questionnaire would be handed out at the ending of a testing session of willing participants, where they would fill in their thoughts. Alternatively, interviews can be conducted for similar effects, however this is both time consuming, and has a higher likelihood of inducing bias.

Engagement on the other hand would likely require observations to accurately measure, as engagement is something subconsciously done by a user, making their own response more abstract. This can be measured by observing a player's behaviour whilst they're testing a game. The observer will identify the level of engagement of each observee by keeping track of their visible emotions and actions. Numerical values can be assigned to responses such as 'the number of times the user looked away from the game', or 'the number of times the player spoke to the other player'.

Asides from game values, educational value is another point of consideration for measurement based on the intended aim of the game, in this case it would be the development/increase of core skills. To keep things simple, a quiz will be provided prior and after playing the game, these would be multiple choice questions with pre-determined answers that test decision making skills, and critical thinking – each answer has varying points assigned. This is done before a game to identify their 'current' decision making score at the time, which would be compared to the answers of a different quiz conducted after testing the game.

As multiple processes would be performed, this can be summarised as a process that will be conducted during the main testing phase of game after a suitable prototype has been developed. During this, willing participants will be provided a survey to identify their original decision-making score. After which, users will compete against one another to induce competition, this would facilitate a more suitable environment for observation as they'll be less pressured from being observed. An incentive can also be provided to reduce the dissatisfaction involved with being observed. After the testing process of the game, users would once again be provided with a quiz (with a different set of questions and answers), as well as a questionnaire to determine their impressions of the game.

To ensure that the aim is met, measurable objectives can be summed up as the following:

- Average user impression of the is **above 25** when tallying the scaled responses of the questionnaire.
- Average score of user engagement is **above 15** when tallying numerical observations of each participant.
- An average **increase of 5** in participant decision-making scores, when comparing pre- and post-game quiz scores.

These objectives will serve as milestones to identify whether success has been met, after the completion of the project.

Note: Questionnaires/quizzes/observations has not yet been conducted, these processes and values are subject to change.

1.7 – Software Development Methodology

Based on the context of the situation, an empiric approach was chosen as the most suitable methodology. A predictive approach such as waterfall is highly dependent on the skill and experience of the developer, which as a student I currently lack.

Predictive approaches are rigid, and hence lack flexibility, developers cannot 'backtrack' as there are clearly defined stages. This poses two major issues, and the first is the lack of feedback and user-involvement - this decreases the quality of the software, as all feedback is processed near the end of the development. This leaves no room to identify what is good nor bad about the software from the end-user perspective, nor the means to implement changes that would act on this, as testing similarly falls under the same issue.

The second issue involves uncertainty, which causes predictive approaches difficult to implement. Both Unity and C# are tools unfamiliar to me, which may raise issues in development, as I'll be required to learn how to utilise both whilst developing the software. A longer development time, higher utilisation of resources, and more bugs are potential consequences of this. An additional point of consideration is time commitment, as I'll be working on five coursework projects in tandem over the course of the year. Any issues that arise from other projects will likely influence the amount of time I have available, hence making it another uncertainty that may cause problems. External factors should also be considered such as newly released games that can serve as case studies and inspiration for the on-going project, which would otherwise be difficult to factor in with a predictive approach.

Agile can be considered a constantly adapting process of management, making it more suitable for handling uncertainty. As there is a clear aggressive deadline, a feature driven development of method of Agile will be employed for planning and management. This allows me to cut down or increase the scope of the project depending on the time I have available, which is the biggest uncertainty at large.

Scrum and Kanban were the most likely candidates for the chosen methodology, however after careful deliberation, Kanban was chosen. Scrum necessitates clearly defined roles, however I'll be working as an individual rendering a large portion of Scrum obsolete, as the Sprint cycle will reduce in effectiveness. It can be argued that I could potentially take on the responsibility of all three roles, however this is both time consuming, and difficult to implement.

Kanban will be employed using Trello to serve as the board. A virtual board whilst may have accessibility issues of potential maintenance or down-time, it's much quicker and simpler to implement than a physical board. Trello will be used to define the pipelining of tasks, which will be assigned tags based on what the task involves (description) and the priority of the task in accordance with the overall software. For example, a leader board which serves to enhance user-experience will have a lower priority than a functional UI which is a fundamental element of the software. The board will essentially be used for task tracking, where tasks are assigned clear deadlines that are flexible as long as they do not fall under the critical path. The board will be updated per daily stand-up through self-evaluation and review of the progress made. Tasks can then be reassigned into categories, based on the progress – for example, tasks with lower priority that have not completed would be moved into a 'Backlog', that would be completed at a later date when there is sufficient time, whilst completed tasks will be moved into 'Complete'.

2 – Background Research

This section covers secondary research regarding the intended deliverable of the project. This will be used to justify certain decisions made regarding both the scope and the requirements, by sourcing known information.

2.1 – What is a Strategy Game?

Strategy games can be defined as.

‘A game in which players’ decision-making skills have a high significance in determining the outcome.’ [6]

Whilst the general definition is clear, what decision-making refers to can be rather ambiguous. Many games contain dialogue options, however can the choice of deciding between dialogue options be considered ‘strategic’ in of itself? ‘High significance’ to the outcome suggests otherwise, however this is not necessarily the case if a dialogue affects the entire routing of narrative as it does with many visual novels.

That being said, visual novels are a completely separate genre to strategy games. What sets them apart is due to how the decision-making influences the game. Whilst dialogue options can be treated as decision making, more times or not, this is entirely based around a narrative, similarly to reading different endings of a book - at the end of the day the outcome is fixed, and only presents an illusion of choice, on what ending they ‘choose’.

Decision-making on a fundamental gameplay level is what determines whether a game is strategic or not. Players should feel responsible for their actions, and this is done by dynamically reflecting success and failure. To put it into perspective, in a game of chess, actions are committed based on moving a piece, which affects the state of the game. Is the piece or other pieces left vulnerable as a result? Can I use this move to threaten enemy pieces? Should I sacrifice this piece to save another? These are thoughts that the player may rationalise before making a decision, and the player must accept the consequences that follow with their decision.

This has led to strategy games; despite being a relative niche, to become a very broad genre of games, stemming numerous sub-genres of games based on the notion of ‘strategy’. Tactical role-players, auto chess, JRPGs, board games, and card games, are some of many examples’ strategy games encompasses, and to a lesser extent, this expands to more popular genres such as tactical shooters, and evens MOBAs (multiplayer online battle arenas).

2.2 – Game Design

If a card game such as Hearthstone requires the player to make a decision on what card/s to use in a given turn-dynamic environment, then a tactical shooter such as Rainbow Six Siege (or Siege) would require the player to make time-critical decisions for their given role and context. But what sets these decisions apart?

2.2.1 – Resources

Most if not all strategy games have some form of resource they must consider when it comes to decision-making, whilst the type and quantity of resources varies, the overall concept is present. This could be a virtual economy that the user needs to micromanage, or the time that they need to be aware of. Effective resource management correlates to effective decision-making and vice versa. Using the previous examples, Hearthstone would require the user to keep track of their 'mana', their 'health', as well as the cards on hand. 'Siege' on the other hand would require the player to keep track of their quantity limited equipment (e.g., ammo), and the time left within a round as the bare minimum.

Resource is a game design element that will be heavily emphasised in the final project. Not only does it help promote resource management as a skill (essential in the real-world), but it also mimics real-world decision-making scenarios – both of which fall in line for facilitating an educational game.
This will be implemented in two ways.
The first will be a virtual economy that players are expected to manage. How 'effectively' a player acquires and utilises resources will play a pivotal role in the win conditions set. This encourages careful planning of resource consumption and will help factoring resource management as a decision-making consideration.
Being a turn-based system, the time is relative to the number of turns that have passed or will come to pass. Economy is something that grows over time hence making time a significant factor to consider when implementing realism to resource management.
The simplest method of mapping resource to time is to utilise the existing 'turn' system. This can be done by allowing the economy to grow by generating resource on a per-turn basis. Whilst this lowers the realism involved by reducing uncertainties, this also allows for a more controlled environment for players to plan. Several games use a similar economy generation system and has proven to be effective – this will be covered more in the Case Studies section.
Time is another factor that will be implemented that not only addresses the issue of inactivity, but also promotes time-critical decision making in a less restrictive and punishing environment as opposed to real-time strategy games.

2.2.2 - Real-Time Vs Turn-Based

Strategy games can commonly be defined as one of two categories – real-time or turn-based. These are defined based on interaction time in correlation to the gameplay. As the name suggests, real-time is a continuous and reactive mode of play where both the state and environment of the game is dynamically changed with every second that passes. Turn-based however operates on 'turns' which are essentially phases of game state, the state of the game does not change until an action has been submitted by the player who's 'turn' is currently ongoing.

Both modes of strategy have their own advantages and disadvantages, but overall, turn-based was chosen as opposed to real-time. What a turn-based system offers is a greater degree of abstract planning due to time being less of a constraint. Whilst this does detract from the realism you would expect in decision-making when compared to real-time systems, this establishes a 'balance' in rewarding players for strategic thinking. Real-time systems are heavily dependent on reaction time, whilst reaction time in formulating decisions is suitable for education, it becomes an issue when it becomes ‘gameplay that rewards rapid mouse clicking’, which turn-based systems do not suffer from.

2.2.3 – Strategy Vs Tactics

Strategy and tactics can be defined as two separate entities, despite having strong relevancy to one another. Tactics refers to a single or multiple actions employed to reach a goal – from a militaristic gameplay point of view, this would refer to the usage of troops in a given battle. Strategy on the other hand can be defined as “a plan of action designed to achieve a long-term or overall aim” [7]. If strategy is the plan towards an aim, tactics are the steps required, this would mean factoring in a much larger scale of variables such as the environment, if applied with the same example used for tactics.

Strategy will be the focus of the game, as it falls in line with the policy of developing core skills. When making informed decisions, a broader perspective always needs to be considered in the context, what’s the point of making short-term profits, when the long-term benefits are neglected? Decision making naturally improves as more factors are added into the equation, which a strategy game can provide more of in comparison to a tactics focused game. A strategy game focuses on a much larger scope outside of a single battle or encounter – this would provide players with a richer experience of decision making.

As mentioned previously, the game expands on the fundamental idea of chess, but at a larger scale. Factors such as environment/terrain, the situation of multiple battlefields, and the pool of resources, are things the player must constantly be aware of, which adds to increasing the ability of the player’s information gathering and processing skills. A tactics game which limits the scope, would greatly dampen the user’s experience in developing the stated skills as it narrows their field of vision, hence making strategy the optimal choice despite the scale of the project increasing consequently.

2.2.4 – Map and Exploration

A map serves as a virtual space, players are expected to interact with, acting as a game board if using Chess as an example. Modern strategy games utilise maps in a way that enriches gameplay experience by allowing map design to influence strategy in a dynamic manner. The map may serve as its own ecosystem that may influence decisions that can either be beneficial or detrimental, and it’s such choices that enhances gameplay and learning – to put it into perspective, an open field battle on grassy plains, would greatly contrast to a siege held by a defending and attacking front due to the terrain and physical obstructions.

‘Fog of War’ is one of the more well-known design techniques employed in both real-time and turn-based strategy that influences the map. It refers to non-visible or non-explored environments, hindering player visibility as a ‘black fog’ inducing an element of uncertainty for all players involved. Using Chess as an example, players would have a more restricted understanding of the game state, as their vision is impeded due to not knowing where some if not all the enemy pieces are located. Such mechanic can be utilised to not only place ambushes, but also allow ambushes, and it’s up to the players themselves to make deductions or choices based on their narrowed vision, on how to tackle the situation.

The intended deliverable will utilise a map simulating various terrain one may expect in a real-world – said terrain will dynamically influence actions available to players, whether it be untraversable, or influencing combat mechanics. This will serve to enhance decision making skills, as the environment will be factored into making optimal situations. Whilst fog of war may be employed to facilitate uncertainties that may be found in the real-world, this feature will unlikely be implemented as it in turn increases the complexity of the game. Complexity should be maintained to a degree where it doesn't serve as a barrier for players to enjoy a game due to skill levels, hence such feature will be removed or left as optional features that can be toggled based on player discretion.

2.2.5 – Number of Players

Games fall under two categories – single player and/or multiplayer. Single player refers to a game with only a single participant. For a strategy game, this generally refers to a player being pitted against an AI. 'Campaigns' are often featured in such games, where players are led through a narrative containing a series of matches against AI, that become progressively more difficult as the story progresses.

Multiplayer on the other hand, is the exact opposite, being a game featuring multiple players as either allies, and/or opponents. Naturally, the gameplay becomes more dynamic as a result, as opposed to AIs which follow a set algorithm or pattern which can be exploited if discovered. Having other players as opponents allows for competition to grow, as skill is determined on decision making and strategic thinking, naturally making a multiplayer game an obvious choice for the software. The number of players present is another factor to consider for multiplayer, the more players present, the more chaotic the state of the game becomes, as well as the duration of games. To keep thing simple, two players will be pitted against each other as reminiscent of chess, allowing for easier understanding of the game due to familiarity, however additional players is an idea that can be later explored.

Something to consider would be the utilization of a campaign being employed as an alternative game mode to introduce functionalities available in the game, this would act as a tutorial that will guide players via narrative on how they can play the game. Time constraints make this becomes less feasible; hence multiplayer will be placed with higher emphasis for priority.

2.2.6 – Setting and Themes

Whilst not mutually inclusive, 'warfare' and strategy games are a commonly used combination of genres. Engagement and entertainment are necessary to create a successful game, and war is a theme that has been identified to pair well with strategy related games, based on the trend of successful strategy titles.

The environment of the game can define many major components of the game. The narrative, graphics, even the gameplay itself. A scenario may sometimes be needed to depict the direction the game will lead towards and would eventually be used to build a worldview. Having a clear understanding of what kind of worldview, you want to build will aid in constructing an expansive and immersive world.

Questions that need to be asked are whether the setting is fictional or non-fictional, is there a backstory, where is the game set place in, and when? Fiction entails an imaginative world, hence a higher degree of freedom in creating an immersive world, for younger target audiences, this is generally more appealing. Non-fictional settings however contain a higher degree of realism, using real world scenarios or stories to create a virtual world – this can potentially be used to enhance historical significance in order to increase the educational value of the game by facilitating historical events such as the Napoleonic Wars, or the Three Kingdoms period.

2.3 – Case Studies

Despite the lower audience demographic present for strategy games, as opposed to other genres, the strategy genre still holds an expansive library of games that can be treated as case studies for development. One of the more notable titles being the '*Fire Emblem'* franchise that bears resemblance to the theme and gameplay that is intended.

Being a critically acclaimed series of the strategy genre that has existed since 1990 [8], Fire Emblem serves as a good reference point for traditional grid turn-based games. Many gameplay concepts can be derived from this series, being one of the leading trailblazers of the grid styled turn system. For example: *'Terrain'* has served as a staple since the games' founding and has been iterated on throughout every new game – <u>*'Terrain'* serves as a strategic element</u> that can dynamically influence the flow of a decision making, whether deciding on defensive choke points for allies, or avenues of aggressive attacks. Naturally this a point where inspiration can be drawn upon, with decision making being a skill that is favoured for the project.

Other considerations that can be derived are primarily the weapon triangle the game features, as well as balancing. Similarly to *'Terrain'*, the weapon triangle serves as a critical factor in decision making - functioning similarly to a game of rock-paper-scissors which determines the effectiveness and impact of actions performed.

<u>Balance</u> on the other hand determines the exploitability of game systems, for a single player game like Fire Emblem, this is less of a concern due to it being less competitive in nature. However, for this project, balance can be a key point of interest, as multiplayer games are inherently competitive, especially for a game that uses chess as a point of reference. META (most effective tactics available) is a term commonly used in games, where victory is dictated by choosing the most efficient method to play the game that were established due to poor game design.

To put it into perspective Three Houses (one of Fire Emblems recent games) was jokingly called 'Flier Emblem' [9] by a group of players due to the poor balancing of aerial units (fliers) that were regarded as META. Fliers have the boon of high mobility, high evasion, and unrestricted movement of impassable 'Terrain', and essentially rendered every non-flying unit obsolete. High mobility paired with unrestricted movement essentially allowed these units to blitz through enemy lines much better than any grounded unit could, and paired with high evasive stats, these units were essentially untouchable even in enemy lines, invalidating defensive unit archetypes. The only drawback is the low offensive stat, but even this can be offset with equipment. This along with other similar scenarios can serve as a good reference point for unit design of the project, regarding rules of behaviour.

'Advanced Wars' [10], is a kindred spirit of Fire Emblem, making it a good stimulus for deriving ideas from. Buildings, unit generation, and resource management, are interconnected systems, that distinguish this series from Fire Emblem. War is the main theme here, where players are expected to utilise their existing resources to annex territory of their enemy, from this - forces can be accumulated by mass production of heavy weaponry to subjugate the opposition. This game excels at developing strategic thinking so that players can control the board state, and resource management and unit generation are the two main components that drive this – both of which are articles of consideration for the scope of the project.

Something to point out is the clever use of cute visuals that reduces the impact of violence to the point that the age rating is only 7+. This highlights the effectiveness of graphics as a tool for improving public reception as; something to investigate (see *Figure 1* as a reference).

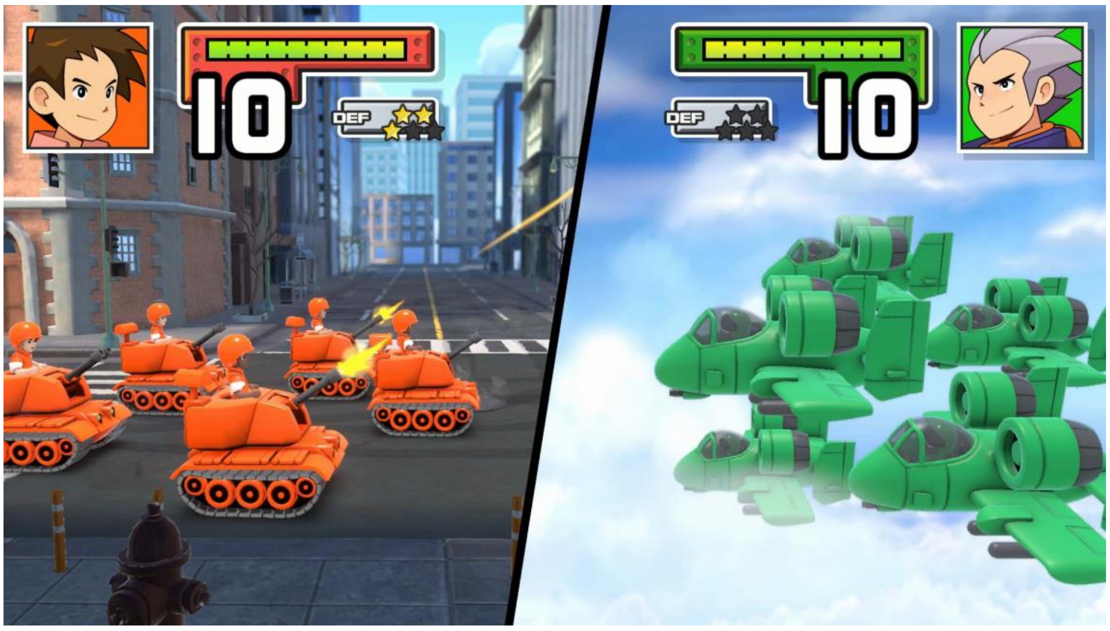
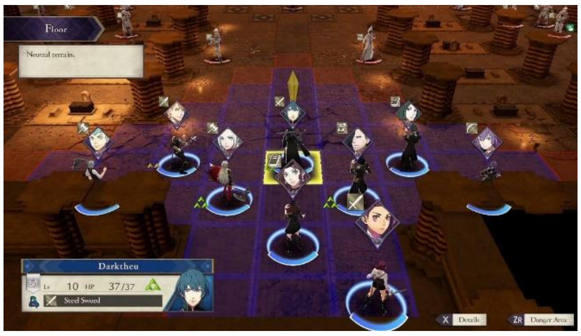
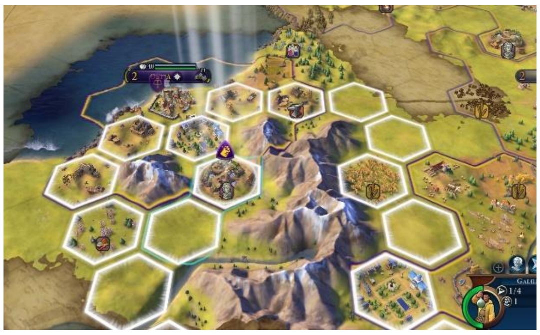


Figure 1 – Advanced Wars Re-boot combat visuals. [10]

If both previously mentioned case studies fall under the scope of a *traditional basic grid style turn-system*, 'Civilization' (commonly known as Civ) is undoubtedly unconventional and complex for the genre. As the name implies, Civ revolves around advancing civilization, what the player does with this civilization is entirely up to them, whether they perform crusades to unify the world power, form religious cults to gather faith for dubious purposes, or advancing technology to the point that AI takes over the world. The unique selling point is the freedom players are given, from choosing factions, deciding historical figures as leaders, to a technology tree. Civ offers a plethora of win-conditions and heightens replay ability through a vast list of decisions the players can make.



Square Grid System from Fire Emblem: Three Houses. [11]



Hexagonal Grid System of Civilization VI [12]

Figure 2 - Side-by-side visual comparison of Grid systems for Fire Emblem and Civilization VI.

Being different from most games of the genre, Civ naturally has a high bar to entry due to the complexity of the game – learnability and understandability are critical component to player retention. However, it's this uniqueness that represents a broader perspective of ideas making the game stand out as a case study. Even the grid system is completely different (*Figure 2*).

3 - Design Brief

This section serves as a brief overview details the fundamental concept of the game that was intended to be developed during the duration of the deadline.

3.1 - Description

An interactive 2-player turn-based strategy game built on Unity Engine. Based on a medieval fantasy setting, the two players command opposing factions in a war styled scenario to capture the enemy 'Fortress'. Formulating strategies, players engage in a dynamic battle of wits in controlling their forces, whether it be advancing on enemy terrain, fortifying their position, engaging in combat, or taking strategic positions.

3.2 - Gameplay Overview

Similar to chess, two players command opposing forces to capture a strategic piece as the win condition – the Headquarters (Fortress). Players do this by rotating turns and issuing commands to their possessed units within the fixed allotted time interval to maintain pressure and eliminate idleness in between matches. However, unlike in chess, the scale is much grander, with both the environment and the available pieces being much more interactive. Players are granted much more freedom in their actions during their turns, with units being more flexible – having no pre-defined movement pattern, nor being restricted to only commanding 16 pieces that cannot be replenished.

The gameboard is set on a set of pre-generated maps containing various environmental variables known as *'Terrain'* and various capturable buildings referred to as *'Structures'*, with the respective headquarters of the players being one of them. From these *'Structures'* players can purchase units using micro-managed resources that is replenished every turn. These units can be controlled to traverse the *'Map'* and engage with enemy controlled units within combat range. The end goal is to capture the enemy *'Fortress'* by attacking it your units to deplete its health, whilst defending your own. Players are expected to formulate their own strategies whilst counteracting the schemes of their opponent in an interactive and dynamic battlefield that facilitates war.

3.3 – Unique Selling Point

The premise of the game is to present an entertaining environment to refine ones' problem-solving skills, and as the spiritual successor of chess, this aims to deliver. Unit variants, advanced combat systems, and resource management, are some of many examples that can be expected as part of the scope for the project. These features work in tandem to enhance informed decision making, with entertainment as a secondary focus.

What allows this game to stand out is the combination of various grid turn-based strategy game features used to enhance the educational value of this game. Many of the mentioned features alone are not unique, however packaging these components as system will provide a new experience for strategic thinking. However, unlike in Civ, the intended product will sustain simplicity to allow for a lower entry point for beginners. But with a difficult to master system paired with competitive multiplayer, that will retain more advanced/'hardcore' players.

3.4 – Feasibility Study

After deliberation, it can be assessed that technical and economic issues should not be present within the development. The purpose of this project is not to develop a AAA game, but one that fulfils the conditions of developing educational skills. Neither a large budget nor power device is necessary. Whilst a budget may serve to enhance aesthetics and audio, these features fall under the category of quality of life rather than gameplay, which is the main focal point. Unity is a free software that provides the sufficient tools needed for game development, with assets readily available.

The most prominent question and greatest concern involved with this project is – *'will there be enough time to develop it?'* This is a reasonable question to ask due to the uncertainties involved with time. Purely based on the design brief, the fundamental components that make up this game can be broken down as the following.

- Multiplayer networking.
- A turn-based system.
- Units that act as chess pieces.
- A grid system.
- Win conditions.

Whilst additional functionality such as *'Terrain'* was mentioned, on an abstract level, these five systems alone are enough to serve as the foundation of the game.

With this information, a deduction can be made that the development of this game is indeed viable. This conclusion was made after summarising the core components that needs to be developed with the intention of the game.

An indie game was suggested to take anywhere between a few days to several years to develop [13], whilst this doesn't inspire confidence, it does suggest that it is possible to develop a game within the span of 7 months as an upper limit. The proposed game that is being developed is for skill nurturing as selling point, with entertainment as a secondary factor.

Naturally this gives more leeway in regard to graphics and sound which are one the more prominent time sinks of the game development lifecycle.

With an abundance of online tutorials and guides to serve as a reference point for development. It's reasonable to assume that a system can be developed consisting of the 5 mentioned components, within the span of 1-3 months if factoring the removal of excessive gold platin, and instead prioritising the functional aspects.

4 – Software Development

This section specifies the processes involved with the development for the current build of this game. Justifications and improvements will be stated for certain decisions that were made during development.

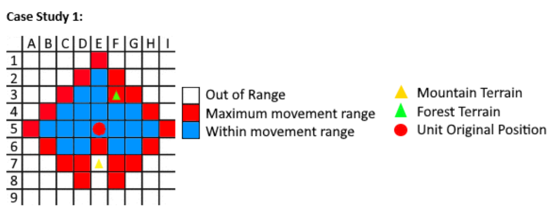
4.1 – Planning

With the initial concept of the game finalised, a more in-depth plan is required to iron out the more technical details regarding implementation. Section 3.2 outlines features that would potentially serve as components for the system, but a clearer description is necessary for implementation. For example, 'Terrain' and 'Structures' have been mentioned but based on description their purposes are very abstract and left for interpretation - What exactly are they, what advantages do they bring to the game, and how do they influence the player experience? These questions need to be addressed and fleshed out not only for player understandability, but also technical implementation.

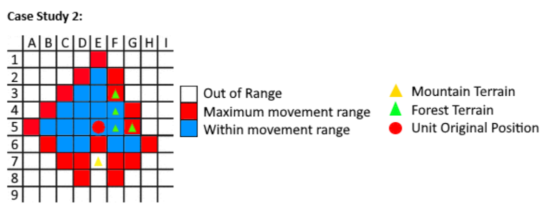
4.1.1 - Scope

The Scope (or Design Plan) documents the intended directions available for the project to work towards, this includes the technical outlining of features, how they influence the system, and how users may interact with them.

The Scope serves as a general guideline rather than an instruction book, documenting behaviours and rules certain systems follow, this aids in the actual implementation of the project in terms of deciding how game logic would operator. Case studies 1 & 2 of the Design Plan document serve are an example of this, where logic involved with a Units' movement is depicted using description and images to describe both the fundamental behaviour, as well as possible scenarios that could influence change.



Case Study 1 – Movement example.



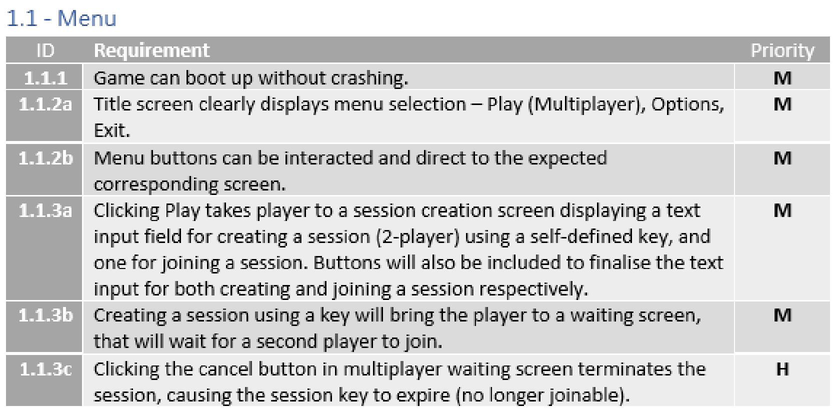
Case Study 2 – Movement example inhibited by 'Forests'.

Figure 3 - Side-by-side comparison of case studies 1 and 2 found in the Design Plan document.

4.1.2 – Requirements

Following from the Scope are the requirements. These are a listing of logical components that were that were identified when breaking down various features described in the Scope. Requirements are essentially building blocks that compose the system, and they aid in the development by serving as a pseudo checklist for what needs to be implemented, for the system to function as described. These can be functional, which are composed of functional elements of the system, and non-functional, which are centred around optimization for user experience.

Having a visual list of tasks that need to be completed, allows the developer insight on what to work on next, enhancing overall productivity and streamlining workflow. Additionally, the list of requirements can aid in mitigating a major risk involved with this project, which is uncertainty of time. In the Requirements Document, the requirements have been categorised with priority metrics. These arbitrary values were assigned based on the relevancy and importance they hold on to the project.



1.1 - Menu

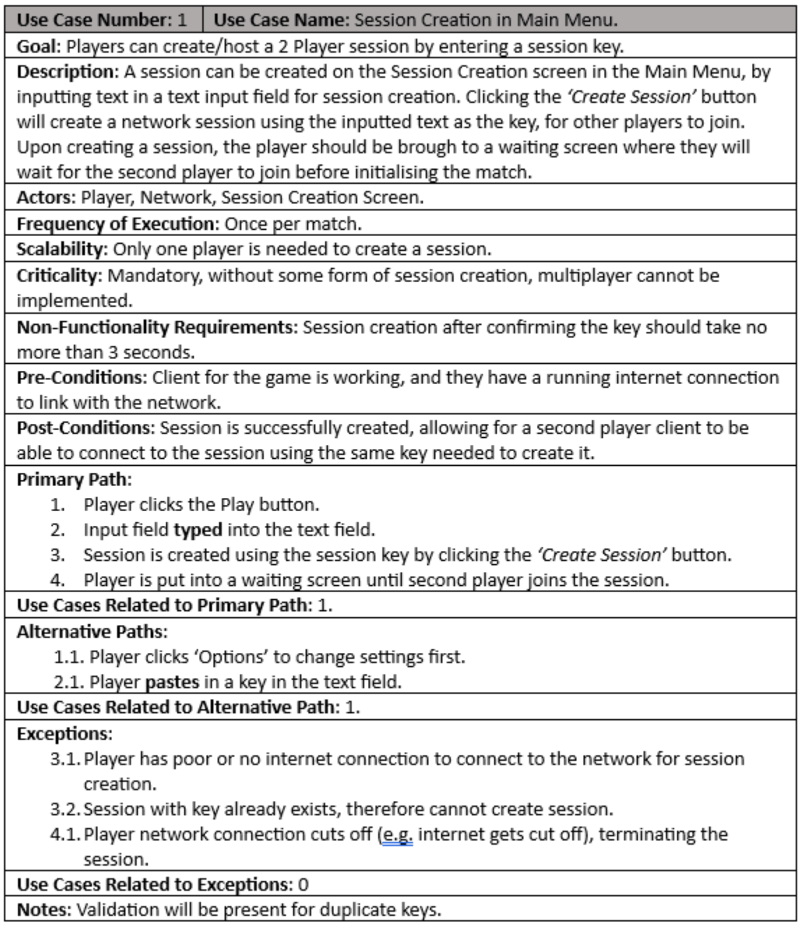
ID	Requirement	Priority
1.1.1	Game can boot up without crashing.	**M**
1.1.2a	Title screen clearly displays menu selection – Play (Multiplayer), Options, Exit.	**M**
1.1.2b	Menu buttons can be interacted and direct to the expected corresponding screen.	**M**
1.1.3a	Clicking Play takes player to a session creation screen displaying a text input field for creating a session (2-player) using a self-defined key, and one for joining a session. Buttons will also be included to finalise the text input for both creating and joining a session respectively.	**M**
1.1.3b	Creating a session using a key will bring the player to a waiting screen, that will wait for a second player to join.	**M**
1.1.3c	Clicking the cancel button in multiplayer waiting screen terminates the session, causing the session key to expire (no longer joinable).	**H**

Figure 4 - A screenshot of some requirements for the Menu of the requirements document.

In *Figure 4,* the requirements lists the requirement ID, a synopsis of what the requirement, as well as a priority scale. This was structured in a way that is easily identifiable, concise, and measured by importance. The priority scale exists based on what is considered **mandatory**, **high priority**, or **low priority** – and what this enabled is determining what implementations should be prioritised when developing the game allowing the developer to focus more on integral functionality needed for the game to work, rather than gold plating and polish – been more effective time management for development. For example: a game that cannot boot up or is constantly crashing is useless to users as they will be able to fulfil the intended purpose of using the software, hence the priority is **M** (mandatory). Conversely, a cancel button is allocated as **H** (high priority) rather than **M**, due to the alternative solutions that players can overcome this issue, such as closing the application, whilst this is not an ideal case, a solution exists without having to code it in. However, this obviously inconveniences the player, and is not considered a difficult feature to implement, hence has a designation of **H** rather than **L** (low priority).

4.1.3 – Use Cases

User interactions are something to always consider when developing software, as users will not always follow the intended route when interacting with the system. This is especially the case for software (such as a game) that grants a higher degree of freedom in terms of interaction when there is no specified path or tutorial on how a player should play.



Use Case Number: 1	**Use Case Name:** Session Creation in Main Menu.
Goal: Players can create/host a 2 Player session by entering a session key.	
Description: A session can be created on the Session Creation screen in the Main Menu, by inputting text in a text input field for session creation. Clicking the *'Create Session'* button will create a network session using the inputted text as the key, for other players to join. Upon creating a session, the player should be brough to a waiting screen where they will wait for the second player to join before initialising the match.	
Actors: Player, Network, Session Creation Screen.	
Frequency of Execution: Once per match.	
Scalability: Only one player is needed to create a session.	
Criticality: Mandatory, without some form of session creation, multiplayer cannot be implemented.	
Non-Functionality Requirements: Session creation after confirming the key should take no more than 3 seconds.	
Pre-Conditions: Client for the game is working, and they have a running internet connection to link with the network.	
Post-Conditions: Session is successfully created, allowing for a second player client to be able to connect to the session using the same key needed to create it.	
Primary Path: 1. Player clicks the Play button. 2. Input field **typed** into the text field. 3. Session is created using the session key by clicking the *'Create Session'* button. 4. Player is put into a waiting screen until second player joins the session.	
Use Cases Related to Primary Path: 1.	
Alternative Paths: 1.1. Player clicks 'Options' to change settings first. 2.1. Player **pastes** in a key in the text field.	
Use Cases Related to Alternative Path: 1.	
Exceptions: 3.1. Player has poor or no internet connection to connect to the network for session creation. 3.2. Session with key already exists, therefore cannot create session. 4.1. Player network connection cuts off (e.g. internet gets cut off), terminating the session.	
Use Cases Related to Exceptions: 0	
Notes: Validation will be present for duplicate keys.	

Figure 5 - Visual reference of a Use Case Description 1 found in the Use Case Description document.

Use Case Descriptions serve to facilitate and document how a user would interact with the system allowing the developer to produce solutions for various scenarios that would play out. In *Figure 5*, a primary path is documented alongside, alternative paths, and exceptions. The primary path follows the intended route of reaching a particular goal in this scenario is Session Creation. From the primary path, alternatives paths branch out depicting scenarios where the user does not follow the intended route of a goal. For instance, a developer cannot predict when a user may suddenly lose internet connection during a created session, thus creating an exception, therefore a solution should be proposed to address this by appending it into the requirements. This could be terminating the session when internet is no longer detected by the host and producing a corresponding error message indicating the problem to the user.

1.1.3d	Session is terminated when session host loses internet connection and displays an error message.	H

Figure 6 - Appended requirements based of Use Case Number 1 exception 4.1.

4.2 – Software Quality Processes

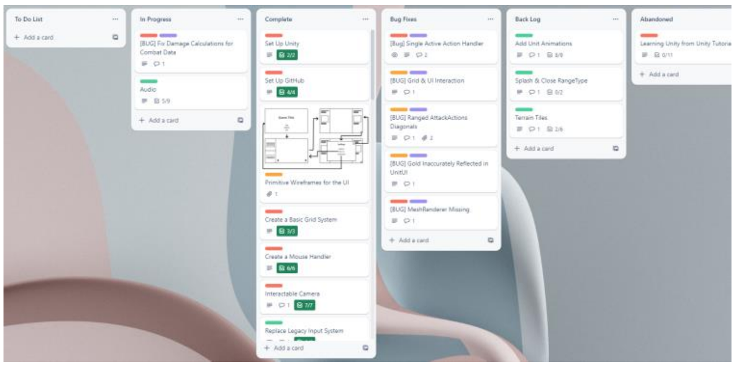
Quality correlates to the reception a product receives from its users. Whilst the plan addresses the functionality needed for the product, a good execution of the plan is needed to follow up, otherwise the workflow can be disrupted due to internal or external influences.

4.2.1 – Trello

Trello is a tool that was mentioned in section 1.7, regard how it would be employed in software methodology approach. Expanding on this, Trello serves as a virtual checklist to organise tasks that were generated from the requirements, making it an effective task management tool in comparison to a simple requirements checklist being much more dynamic and flexible. To put it into perspective, 6 categories are defined on the current Trello board for:

- To Do List
- In Progress
- Complete
- Bug Fixes
- Back Log
- Abandoned

These categories interact with one another as the development proceeds, where tasks may move from one category to another as progress advances.



To Do List
+ Add a card

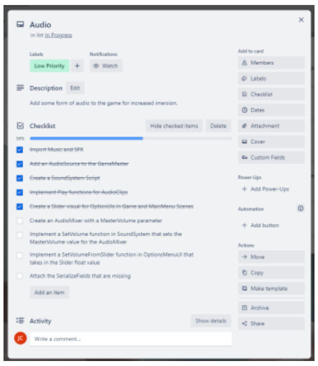
In Progress
[BUG] Fix Damage Calculations for Combat Data 1
Audio 5/9
+ Add a card

Complete
Set Up Unity 2/2
Set Up GitHub 4/4
Primitive Wireframes for the UI 1
Create a Basic Grid System 3/3
Create a Mouse Handler 6/6
Interactable Camera 1 7/7
Replace Legacy Input System
+ Add a card

Bug Fixes
[Bug] Single Active Action Handler 2
[BUG] Grid & UI Interaction 1
[BUG] Ranged AttackActions Diagonals 1 2
[BUG] Gold Inaccurately Reflected in UnitUI 1
[BUG] MeshRenderer Missing 1
+ Add a card

Back Log
Add Unit Animations 1 0/9
Splash & Close RangeType 1 0/2
Terrain Tiles 1 2/6
+ Add a card

Abandoned
Learning Unity from Unity Tutoria
0/11
+ Add a card



Audio
in list In Progress

Labels: Low Priority
Notifications: Watch

Description Edit
Add some form of audio to the game for increased imersion.

Checklist
56%
- [x] ~~Import Music and SFX~~
- [x] ~~Add an AudioSource to the GameMaster~~
- [x] ~~Create a SoundSystem Script~~
- [x] ~~Implement Play functions for AudioClips~~
- [x] ~~Create a Slider visual for OptionUIs in Game and MainMenu Scenes~~
- [ ] Create an AudioMixer with a MasterVolume parameter
- [ ] Implement a SetVolume function in SoundSystem that sets the MasterVolume value for the AudioMixer
- [ ] Implement a SetVolumeFromSlider function in OptionsMenuUI that takes in the Slider float value
- [ ] Attach the SerializeFields that are missing

Add an item

Activity
Write a comment...

Overview of the Trello board for the project. *Audio task description.*

Figure 7 - Visual reference of the Trello board for this project during the final stages of the current build.

This is demonstrated in *Figure 7,* where Audio (functionality) was moved to ‘In Progress’ after starting progress towards the task, in this case being ‘Import Music and SFX’. Each task card carries descriptions of sub activities that need to be completed before a task is ‘Complete’, whereby finishing the task – this provides a more in-depth technical description of specific implementations that need to occur during development. Another thing to note is the labels, which refer back to the priority scale in the Requirements – this serves as a visual indicator for tasks with higher precede for implementation due to their impact on the final product. For example, a Grid System has a higher priority than Audio due to it being a fundamental component of the game in comparison to a quality of life update, hence why it was marked with higher priority and thus completed at a much earlier date.

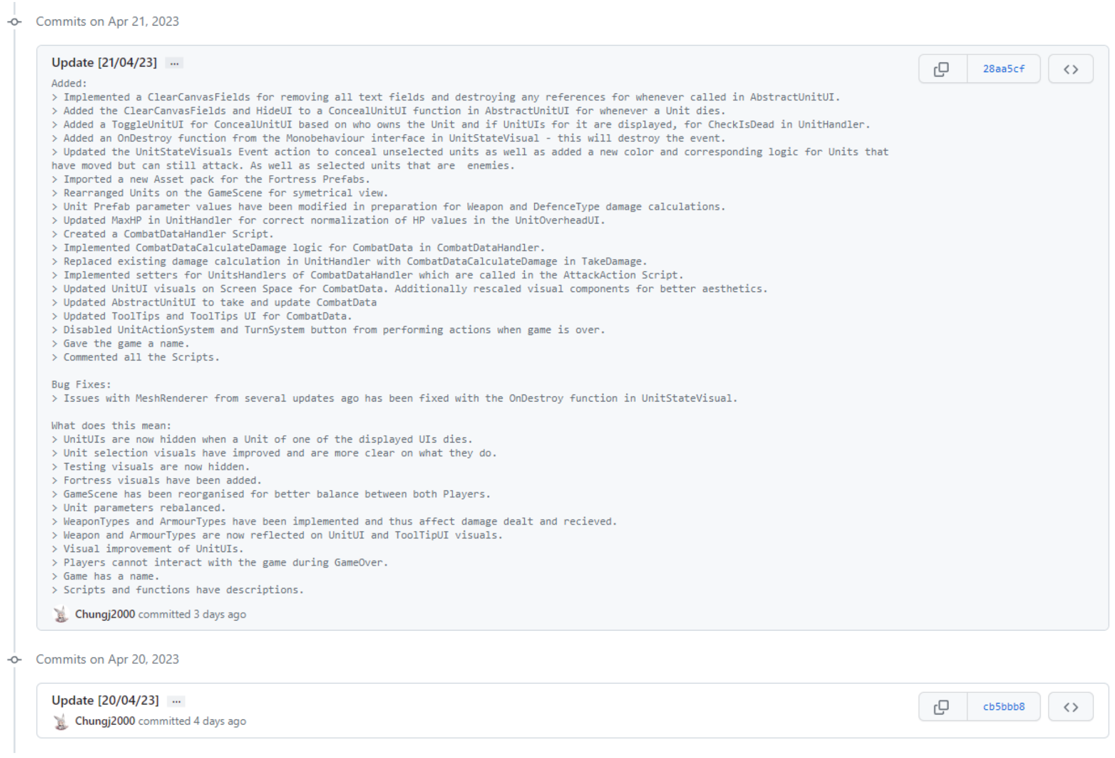
The ‘Bug Fixes’, ‘Back Log’, and ‘Abandoned’, categories are self-explanatory being for: debugging, low priority tasks that can be completed later, and tasks that were not worth completing due to one reason or another.

4.2.2 – GitHub

Version Control is mandatory for large projects such as game development. Losing hours’ worth of progress due to corrupted files is not a light matter, especially with a tight deadline.

> *‘At a high level, GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code.’* [14]

This description alone justifies the employment of GitHub, or at the very least worth considering some form of version control. Being able to store large quantities of code on a VPC (virtual private cloud), allows for relief of developers knowing that they’re able to recover any lost code due to uncertainties that may occur during development. Being a version control software provides additional benefits in the form of reverting changes.



Commits on Apr 21, 2023

Update [21/04/23] ...

28aa5cf

```
Added:
> Implemented a ClearCanvasFields for removing all text fields and destroying any references for whenever called in AbstractUnitUI.
> Added the ClearCanvasFields and HideUI to a ConcealUnitUI function in AbstractUnitUI for whenever a Unit dies.
> Added a ToggleUnitUI for ConcealUnitUI based on who owns the Unit and if UnitUIs for it are displayed, for CheckIsDead in UnitHandler.
> Added an OnDestroy function from the Monobehaviour interface in UnitStateVisual - this will destroy the event.
> Updated the UnitStateVisuals Event action to conceal unselected units as well as added a new color and corresponding logic for Units that
have moved but can still attack. As well as selected units that are  enemies.
> Imported a new Asset pack for the Fortress Prefabs.
> Rearranged Units on the GameScene for symetrical view.
> Unit Prefab parameter values have been modified in preparation for Weapon and DefenceType damage calculations.
> Updated MaxHP in UnitHandler for correct normalization of HP values in the UnitOverheadUI.
> Created a CombatDataHandler Script.
> Implemented CombatDataCalculateDamage logic for CombatData in CombatDataHandler.
> Replaced existing damage calculation in UnitHandler with CombatDataCalculateDamage in TakeDamage.
> Implemented setters for UnitsHandlers of CombatDataHandler which are called in the AttackAction Script.
> Updated UnitUI visuals on Screen Space for CombatData. Additionally rescaled visual components for better aesthetics.
> Updated AbstractUnitUI to take and update CombatData
> Updated ToolTips and ToolTips UI for CombatData.
> Disabled UnitActionSystem and TurnSystem button from performing actions when game is over.
> Gave the game a name.
> Commented all the Scripts.

Bug Fixes:
> Issues with MeshRenderer from several updates ago has been fixed with the OnDestroy function in UnitStateVisual.

What does this mean:
> UnitUIs are now hidden when a Unit of one of the displayed UIs dies.
> Unit selection visuals have improved and are more clear on what they do.
> Testing visuals are now hidden.
> Fortress visuals have been added.
> GameScene has been reorganised for better balance between both Players.
> Unit parameters rebalanced.
> WeaponTypes and ArmourTypes have been implemented and thus affect damage dealt and recieved.
> Weapon and ArmourTypes are now reflected on UnitUI and ToolTipUI visuals.
> Visual improvement of UnitUIs.
> Players cannot interact with the game during GameOver.
> Game has a name.
> Scripts and functions have descriptions.
```

Chungj2000 committed 3 days ago

Commits on Apr 20, 2023

Update [20/04/23] ...

cb5bbb8

Chungj2000 committed 4 days ago

Figure 8 - Commit log for 21/04/23 of the project GitHub repository.

Commits are the updates that are submitted to GitHub repository, and they serve as 'save states' of the submitted code. As save state, these can be 'loaded' allowing for changes to be reverted, giving developers a sense of ease that they can refactor or implement code, and being able to revert these changes if they're undesirable such as breaking existing functionality.

Proper documentation of commits is good practice, as they provide developers a clear understand the contents of each commit, making it easier to identify where they should revert to if necessary.

4.3 – Planning the Implementation

The final stages of preparations end with conceptualising the implementation of the finished product. This involves creating a detailed plan outlining how the physical implementation would take place. This includes how objects are defined, how these objects interact with one another to form a system, and how the user interacts with the system to achieve an intended goal.

Diagrams and descriptions are recommended to determine how a system should be built, as this gives a more comprehensible direction towards structural development of the product, making it good practice towards software quality (quality assurance). These plans do not need to be descriptive however, an abstract guide is advised at the minimum to deliver a functional product for the end-user, as it helps determine any structural or optimization issues present with the abstract concept. Naturally decisions made for implementation should adhere to the prior plans that were made as it simplifies many design decisions that would occur.

4.3.1 – Class Diagrams

4.3.1.1 - Unit

Class Diagrams are visual depictions of components present within the system – they also vaguely describe the relationships between said objects by indicating dependencies, compositions, aggregations etc.

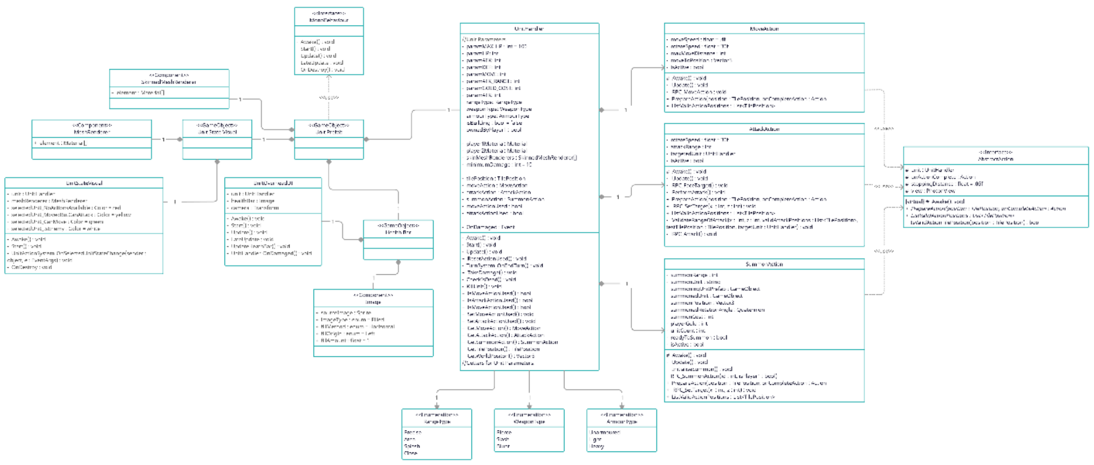


Figure 9 - Class Diagram for defining a 'Unit' in the game.
Please see appendix link if it is too small to zoom in.

The **Unit (Prefab)** objects within the game is defined in *Figure 9* - this describes the concept of what a Unit is; and is comprised of. A **GameObject** is a tangible representation of an object within a game for Unity, here a **Unit Prefab** would represent to controllable Units players are expected to interact with to perform actions.

A Unit is composed of 3 main components:

- **UnitHandler** – Responsible for handing Unit logic.
- **Unit State Visual** – Visual UI component for displaying what Unit is selected and what actions are available.
- **Health Bar** – Visual UI component responsible for displaying a Unit parameter of HP on the screen.

Leaving the UIs aside, a **UnitHandler** is responsible for storing all data that defines a Unit:

- Parameter Values (stats).
- Ownership of the Unit.
- Position of the Unit (on the Grid).
- Actions available.

Now then, why was so many variables and functions compiled into a single Script despite it affecting readability? By defining all relevant data of a single entity into a single script, the code becomes more *cohesive*, this increases ironically readability AND understandability of code as the developer doesn't need to browse through several class (Scripts) files to identify where a certain parameter is hidden within the **GameObject** to perform data manipulation.

Actions of a Unit are defined in the Action classes, which are children of an **AbstractAction** class. **AbstractAction** as abstract class that utilises the factory design pattern as shown from the functions in **MoveAction** and **AbstractAction**. This makes the code more maintainable to an extent as it allows for additional Action classes (*e.g. InteractAction*) to be defined with less effort involved by bundling fundamental functions and variables that constitutes an Action. Being an abstract class also reducing the coupling involved with code, as Action functions are less dependent on their parent.

4.3.1.2 – Grid Action

Asides from Units, players will also be interacting with the environment by performing actions over a GridSystem (*Figure 10*).

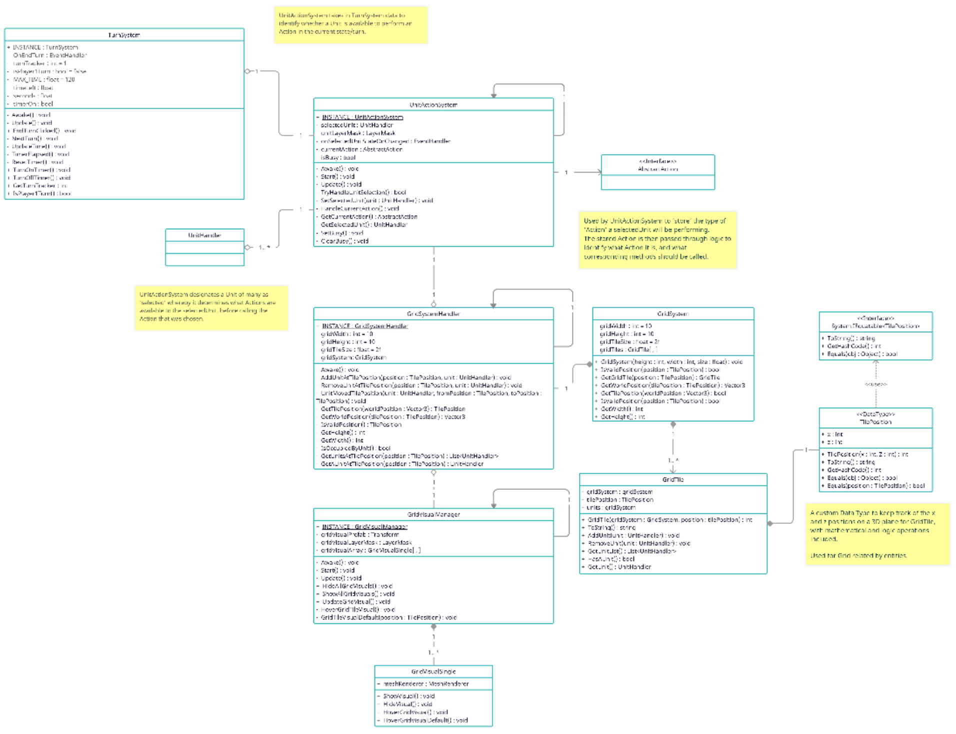


*Figure 10 - Class Diagram for defining Actions performed in the **'GridSystem'** of game, and the relationships involved.*
Please see appendix link if it is too small to zoom in.

The '**UnitActionSystem**' is responsible for passing in a unit reference which is then processed with validation logic to identify what the unit should do given the current context. **UnitActionSystem** uses the data exposed in the '**GridSystemHandler**', to determine available points of action for the unit. It also takes in data from a '**TurnSystem'** to determine whether the player can currently perform actions, based on whose turn it currently is.

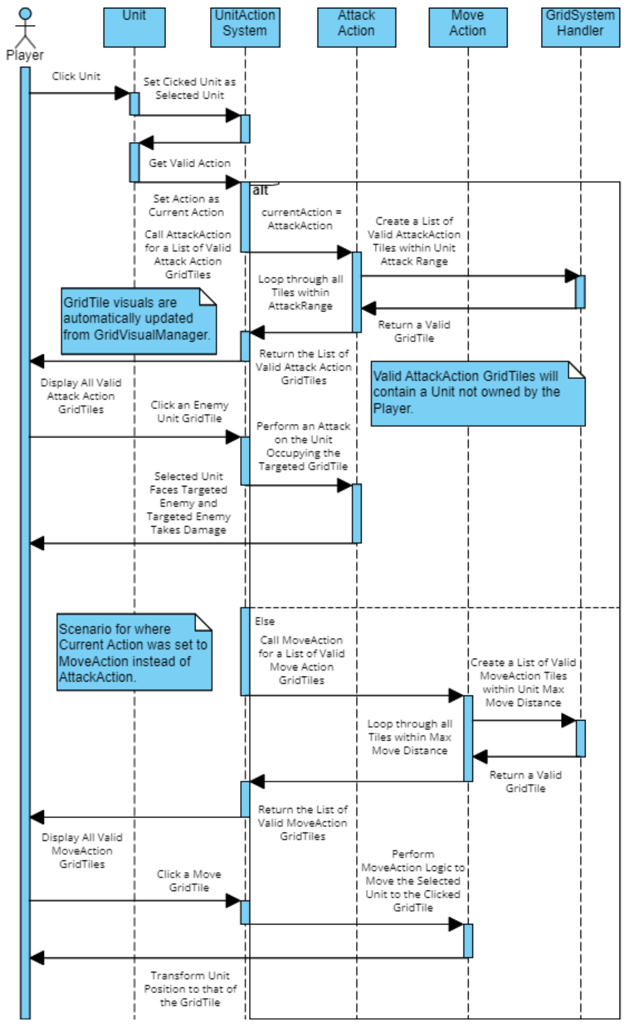
A **GridSystemHandler** is composed of a single '**GridSystem'**, the entity responsible for storing data relevant to a grid. The handler is primarily responsible for data manipulation involved with a **GridSystem**, making it cohesive, as functionality and data storage as separately defined. Within a **GridSystem**, consists of a 2D array of tiles ('**GridTiles'**), virtual **GameObjects** that are created in the game space to represent the tiles of a grid (these objects are visualised from the **GridVisualManager**). A **GridTile** contains logic for storing a unit reference that is occupying its position, and where the position is in the form of the custom data type **TilePosition** (*x*, and *z*, positions on a 3D space).

4.3.1.3 – Summary

The summarise, the **UnitHandler** contains all relevant data needed to execute given actions, and this can be accessed from the **Unit Prefab**. Said prefab passes the data through a **UnitActionSystem** to validate what actions are available. For example, can the unit currently attack, is it even the players' turn, and is the unit even selected in the first place? Valid actions are then processed by acquiring the **GridSystem** data to identify valid **GridTiles** for actions to occur once again applying validation logic in the **UnitActionSystem** with the identified action of the unit *(e.g., is there an enemy unit occupying **GridTile[x, z]**?)*.

4.3.2 – Sequence Diagram

Sequence Diagrams represent the flow of data and operations that are carried out through interactions. This serves as a visual aid to addressing how classes should interact with one another for achieving a specified purpose which aids in the design.



Player
Unit
UnitAction System
Attack Action
Move Action
GridSystem Handler
Click Unit
Set Cicked Unit as Selected Unit
Get Valid Action
alt
Set Action as Current Action
currentAction = AttackAction
Call AttackAction for a List of Valid Attack Action GridTiles
Create a List of Valid AttackAction Tiles within Unit Attack Range
Loop through all Tiles within AttackRange
Return a Valid GridTile
GridTile visuals are automatically updated from GridVisualManager.
Return the List of Valid Attack Action GridTiles
Valid AttackAction GridTiles will contain a Unit not owned by the Player.
Display All Valid Attack Action GridTiles
Click an Enemy Unit GridTile
Perform an Attack on the Unit Occupying the Targeted GridTile
Selected Unit Faces Targeted Enemy and Targeted Enemy Takes Damage
Else
Scenario for where Current Action was set to MoveAction instead of AttackAction.
Call MoveAction for a List of Valid Move Action GridTiles
Create a List of Valid MoveAction Tiles within Unit Max Move Distance
Loop through all Tiles within Max Move Distance
Return a Valid GridTile
Return the List of Valid MoveAction GridTiles
Display All Valid MoveAction GridTiles
Perform MoveAction Logic to Move the Selected Unit to the Clicked GridTile
Click a Move GridTile
Transform Unit Position to that of the GridTile

Figure 11 - Sequence Diagram depicting a player interacting with a unit, and the processes involved with action handling.

In the diagram actions are initiated as soon as a player selects a unit by clicking it. From here, what action the unit performs is decided based on what action the **UnitActionSystem** passed – **Move**, or **Attack**. Being children of the **AbstractAction** class, both actions are handled very similarly with minor discrepancies such as the range at which an action can be performed from or to *(attack range VS max move distance)*. Following the route of an attack, a list of valid **GridTiles** are pulled from the **GridSystemHandler**, and passed to the **AttackAction** of the unit. Players are then subsequently expected to click one of the displayed **GridTiles** on the screen which would then call the **AttackAction** to proceed with the attack using the chosen **GridTile**.

4.3.3 – Activity Diagram

Sequence Diagrams are inherently difficult to describe in detail due to the layout making them very abstract in general, so to supplement the sequence of interactions - an Activity Diagram was also created.

Activity Diagrams describes the flow of interactions in the form of a flowchart, this enables a higher degree of description for defining alternative paths of interaction logic, such as validation that is present in the Sequence Diagram.

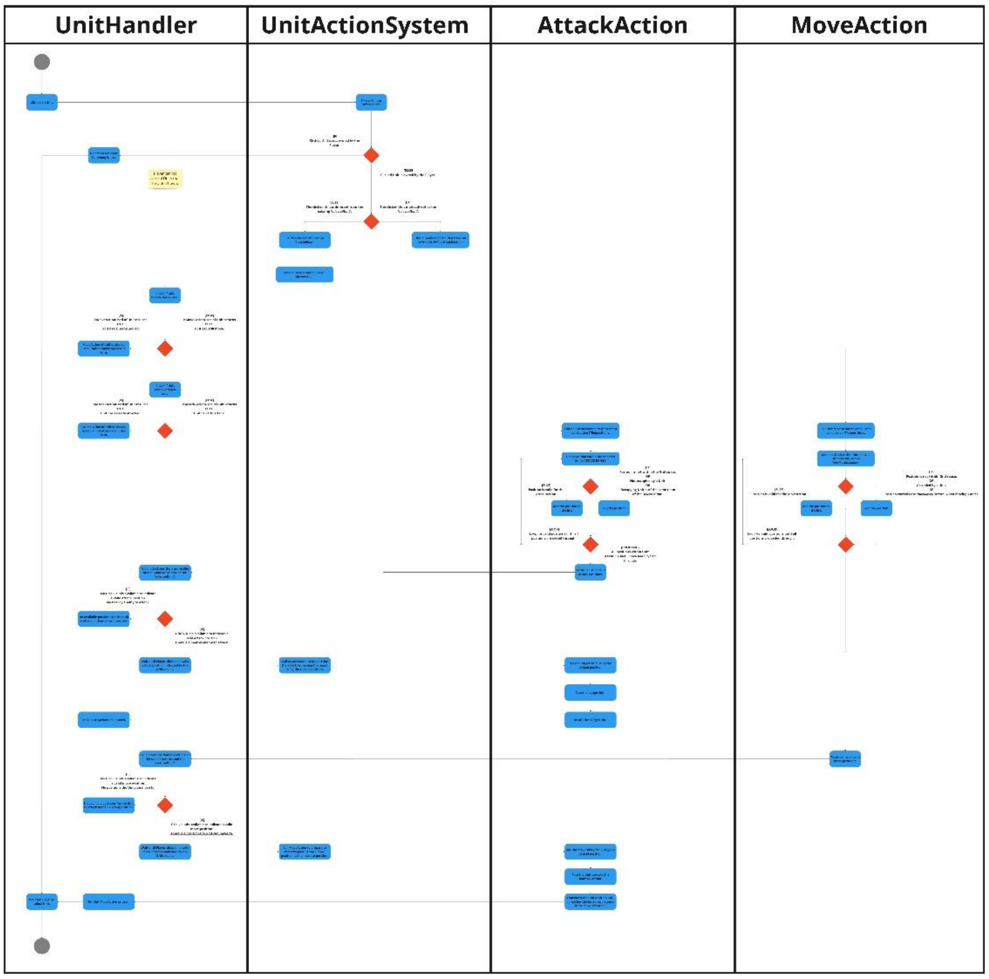


Figure 12 - Activity Diagram documenting alternative paths generated from validation logic.
Please see Appendix for image link.

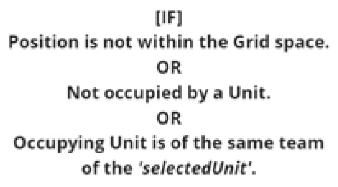
The initial interaction once again begins with unit selection; however this is shortly followed by the validation of whether the selected unit is owned by the player or not. Players should not be able to control units of the other player from a gameplay perspective due to the chaos this causes *(e.g. sabotaging)* and in such cases, the interaction immediately ends.

For this project, move and attack actions are treated as separate entities, meaning a unit cannot move and attack at the **SAME** time, but can move and **THEN** attack. The reasoning behind this is that it's simpler and quicker to implement this way, despite being more inconvenient for the player – *time is still a critical factor*.

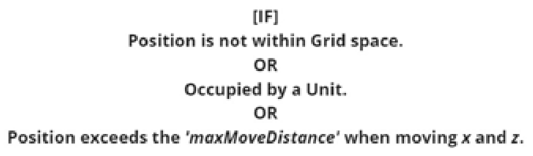
A move action is initiated simply by selecting a unit; however, an attack action would require the player to reselect the **same** unit. *Why not add a UI?* There's limited screen estate that I have other intentions for, having a cluttered/noisy UI can ruin user experience. Based on whether the selected unit is a newly selected unit, the attack or move action is determined.

Subsequently, additional validation logic is once again used to determine the execution of the action – this case being if a unit has already performed their action. A single unit performing too many actions can disrupt the game balance, hence why many games of the same genre follow the simple rule of one type of action per unit. This formula will also be applied here, meaning *one move*, and *one attack* per unit. If the action is already consumed, the interaction stops here, as the unit cannot perform anymore actions of the currently selected action.

As described in the Sequence Diagram both actions follow relatively similar logic once an action is described, which is to loop through a list of valid positions of the designated action. **AttackAction** uses the unit **ATK_RANGE** parameter, whilst **MoveAction** uses **MOVE** parameter as the maximum distance a unit can move at a given time. These positions are validated with the conditions of *Figure 13*.


[IF]
Position is not within the Grid space.
OR
Not occupied by a Unit.
OR
Occupying Unit is of the same team
of the *'selectedUnit'*.

AttackAction position validation.


[IF]
Position is not within Grid space.
OR
Occupied by a Unit.
OR
Position exceeds the *'maxMoveDistance'* when moving *x* and *z*.

MoveAction position validation

Figure 13 - Side-by-side comparison of validation for Attack & MoveAction.

Afterwards, once all positions have been validated the list is returned with each valid tile being visible on the screen. If no Grid visuals are visible this indicates that the current action cannot be performed in the current context, for example there is no nearby enemy within the units' attack range. In such cases, the interaction finishes here.

If there are visible Grid visuals present on the screen, the player is expected to interact with said visuals in order to designated where they wish to perform the action. For move, this would be facing the selected position and then moving to it, but for attack, this is facing the selected position before executing an attack on the unit who occupies the tile.

Finally, after the completion of the action, the action will be set as used, rendering it unable to perform the same action on the current turn.

4.4 – Development Process

With all plans finalised, the development can begin.

Due to the amount of code present, this section briefly outlines any notable points of development.

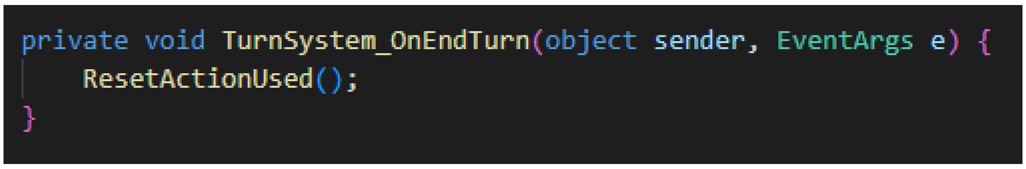
4.4.1 – Coding Practices

With such a large project, coding can become quite overwhelming, thus good practices are needed to help with the maintainability and readability of the code for the developer, and potentially other developers who could join in the future.

Obvious conventions that are employed are appropriate naming conventions.

- All variables use camel case (*Figure 15*).
- Functions are pascal case.
- Specific functions such as events are labelled based on what is calling the event. (*Figure 14*)

Naming conventions help with consistency which in term improves readability.

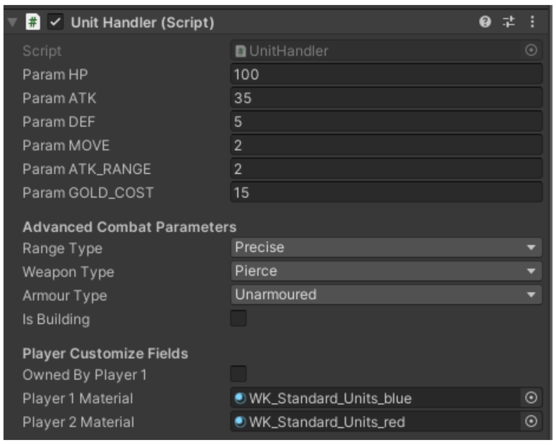


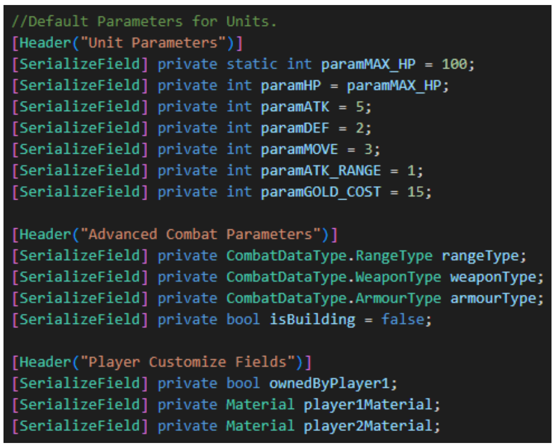
```
private void TurnSystem_OnEndTurn(object sender, EventArgs e) {
    ResetActionUsed();
}
```

Figure 14 - Event call from the TurnSystem, whenever the turn is ended, this resets the actions of all units so that they can perform actions again.

Additionally, all variables have been set to private wherever possible. This prevents functions that shouldn't have access to them to manipulate or change data directly, which reduces the likelihood of bugs occurring from manipulating data that shouldn't be manipulated.

SerializeField, is a special attribute of Unity where parameters of variables can be directly defined in the editor rather than the Script. What this allows is the directly manipulation of variables for specific objects. How this is employed, is reducing the need for factory design patterns for certain objects, such as units (*Figure 15*). This reduces the effort needed in specifically writing code for unique units, and instead can be done by directly creating a unit in the editor and manipulating the values through that.


```
Unit Handler (Script)
Script                UnitHandler
Param HP              100
Param ATK             35
Param DEF             5
Param MOVE            2
Param ATK_RANGE       2
Param GOLD_COST       15

Advanced Combat Parameters
Range Type            Precise
Weapon Type           Pierce
Armour Type           Unarmoured
Is Building

Player Customize Fields
Owned By Player 1
Player 1 Material     WK_Standard_Units_blue
Player 2 Material     WK_Standard_Units_red
```

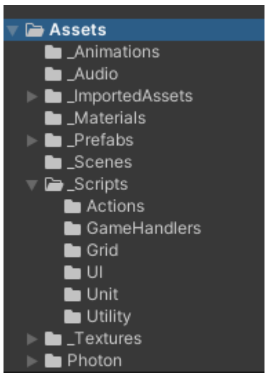

```
//Default Parameters for Units.
[Header("Unit Parameters")]
[SerializeField] private static int paramMAX_HP = 100;
[SerializeField] private int paramHP = paramMAX_HP;
[SerializeField] private int paramATK = 5;
[SerializeField] private int paramDEF = 2;
[SerializeField] private int paramMOVE = 3;
[SerializeField] private int paramATK_RANGE = 1;
[SerializeField] private int paramGOLD_COST = 15;

[Header("Advanced Combat Parameters")]
[SerializeField] private CombatDataType.RangeType rangeType;
[SerializeField] private CombatDataType.WeaponType weaponType;
[SerializeField] private CombatDataType.ArmourType armourType;
[SerializeField] private bool isBuilding = false;

[Header("Player Customize Fields")]
[SerializeField] private bool ownedByPlayer1;
[SerializeField] private Material player1Material;
[SerializeField] private Material player2Material;
```

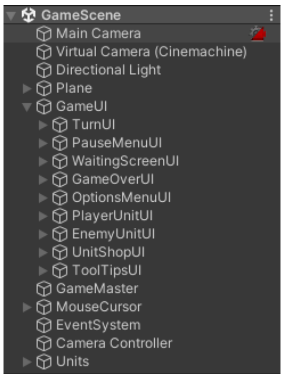
*Unity Editor fields for **UnitHandler*** *__UnitHandler__ script*

*Figure 15 – Usage of **SerializeField** for **UnitHandler** script.*

4.4.2 – Hierarchies

All project files are stored in a way that they can be easily identified through clearly defined hierarchies and structural organisation. This reduces the effort needed to find specific scripts, or game objects (*Figure 16*).


```
Assets
  _Animations
  _Audio
  _ImportedAssets
  _Materials
  _Prefabs
  _Scenes
  _Scripts
    Actions
    GameHandlers
    Grid
    UI
    Unit
    Utility
  _Textures
  Photon
```


```
GameScene
  Main Camera
  Virtual Camera (Cinemachine)
  Directional Light
  Plane
  GameUI
    TurnUI
    PauseMenuUI
    WaitingScreenUI
    GameOverUI
    OptionsMenuUI
    PlayerUnitUI
    EnemyUnitUI
    UnitShopUI
    ToolTipsUI
  GameMaster
  MouseCursor
  EventSystem
  Camera Controller
  Units
```

Unity Editor project folders for assets and scripts. *Unit Editor GameScene GameObjects.*

Figure 16 – Hierarchical structure of the Unity Editor.

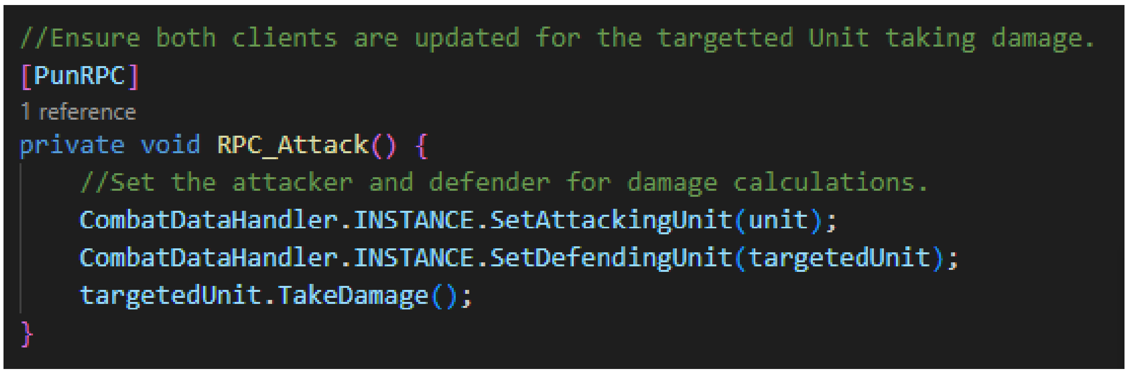
4.4.3 – Frameworks

Three frameworks were utilised in the development of this project. *Cinemachine*, *InputActionSystem*, and *Photon Unity Network (PUN)*.

Cinemachine and InputActionSystem are in-built frameworks that are readily importable from Unity. They allow for easier camera manipulation and player input handling respective, reducing the amount of code needed to directly manipulate and control said functionalities.

PUN on the other hand is the recommended networking framework that is most commonly employed with Unity development. It's a free framework that provides simplicity in implementing multiplayer functionalities. This includes room handling, remote procedural calls (RPC) for client updates, and RaiseEvents.

Room handling is responsible for the creation of game sessions between two players, using keys. This allows for a simple means of networking, where plays can create, join, or leave a session whenever desired.



```
//Ensure both clients are updated for the targetted Unit taking damage.
[PunRPC]
1 reference
private void RPC_Attack() {
    //Set the attacker and defender for damage calculations.
    CombatDataHandler.INSTANCE.SetAttackingUnit(unit);
    CombatDataHandler.INSTANCE.SetDefendingUnit(targetedUnit);
    targetedUnit.TakeDamage();
}
```

Figure 17 – RPC_Attack function that updates both clients for attack execution.

RPCs also play a major role in ensure that both clients up-to-date. In order for a true multiplayer experience, data must be shared between clients so that it becomes an interactive experience between both players. These calls can be found in win conditions, and actions of a unit (*Figure 17*).

4.5 – Testing

Testing is conducted throughout the development process. Having a testing pipeline concurrent to development will maintains a steady workflow, ensuring that a steady pacing of testing is conducted throughout the various cycles of development as intended from an Agile approach of development.

The intended methodologies that will be employed in testing will primarily consist of unit testing, and integration testing.

Automated unit testing will ensure that basic functions and components that are developed perform as intended on a fundamental level from the developer's perspective, this will involve validation of data types, expected from data manipulation, and so forth. How this will be done is using debug logs which is Unity's equivalent of console outputs from log messages in Android Studio, or Junit testing to a lesser extent. Debug logs will provide a clear indication of a processed value which can then be compared the expected value to gauge success. For example, an integer value is processed in a function with the expected outcome of 10, but instead returns 9 in the console, which clearly suggests that the programmed logic of the tested function is flawed in some way.

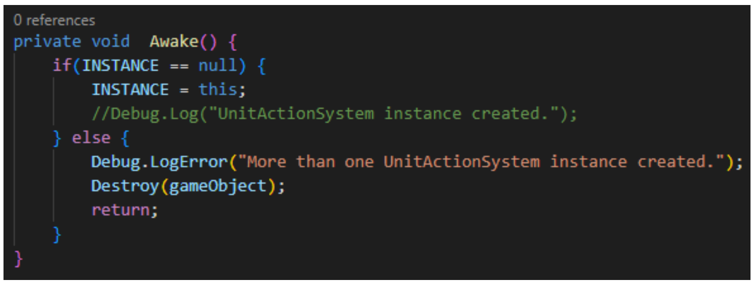

```
0 references
private void  Awake() {
    if(INSTANCE == null) {
        INSTANCE = this;
        //Debug.Log("UnitActionSystem instance created.");
    } else {
        Debug.LogError("More than one UnitActionSystem instance created.");
        Destroy(gameObject);
        return;
    }
}
```

Figure 18 – Debug Logs of the UnitActionSystem Script Awake function for the Singleton Pattern.

What can be identified in *Figure 18* is the use of two debug logs when verifying whether the INSTANCE of the singleton pattern is null or not. In the case of it being null, a debug log will be outputted into the console clearly indicating that the INSTANCE was correctly created during runtime. Whereas in the case that it is not null, an error will be declared in the console suggesting that there is an issue with the singleton pattern having another instance declared somewhere. These serve as a clear indicator for how a function handles data and gives insight for developers whether further attention is required in terms of development or bug fixing.

Excessive debug logs in the console affects performance so they ***must*** be commented out in the final build, and uncommented out when testing which can be tedious. However, what it provides is the return of data where applied. They're used liberally throughout the project and ensures that a constant stream of data is outputted in the console log to determine if the expected output is being produced at any given section of the code.

Integration testing on the other hand identifies whether multiple components correctly interact with one another when performing specific activities. Ideally a test plan should be created, with various scenarios or tasks noted down which will be performed in a live test of the game. Each of the test plan activities will record the actual outcome, which will subsequently be compared with the expected outcome to determine the success or failure of said activity. This helps identify whether functional requirements have been met, for example whether clicking the end turn button properly transitions to the next turn.

Both unit and integration testing primarily fall under white box testing, as the testing will be conducted with developer knowledge on the internal systems and structure for bug fixes to occur. On the other hand, black box testing is also necessary for software quality, as end-users are ultimately those who will be using the product, and they may not necessarily follow and perform activities that are expected or intended by the developer.

System testing is the chosen methodology for black box testing where the game will be tested by users not involved with the development process of the game. This will give insight on what to expect from player actions and serves as feedback on what needs to be developed further, as well identify any strengths the product has that may later be capitalized on. Unlike with the white box testing methods, system testing will not be conducted consistently throughout development but rather at critical development points such as the minimum viable product or a complete prototype, as white box testing is both time consuming to manage and organize.

Note: Unfortunately, due to time constraints the actual system/usability testing of the system was not carried out upon writing the report.

5 – Evaluation

This section briefly outlines the most recently build of the game, regarding the functionalities, as well areas of improvement that can be addressed and should be reviewed in the future. As well as the processes involved with development.

If you would like a more descriptive definition of the current game please see the Demo, or GitHub repository, as this is just an overall summary.

5.1 – Current Build

Decisive Battlefront is the name of the product after an arduous development journey of a months' worth of time solely on Unity development.

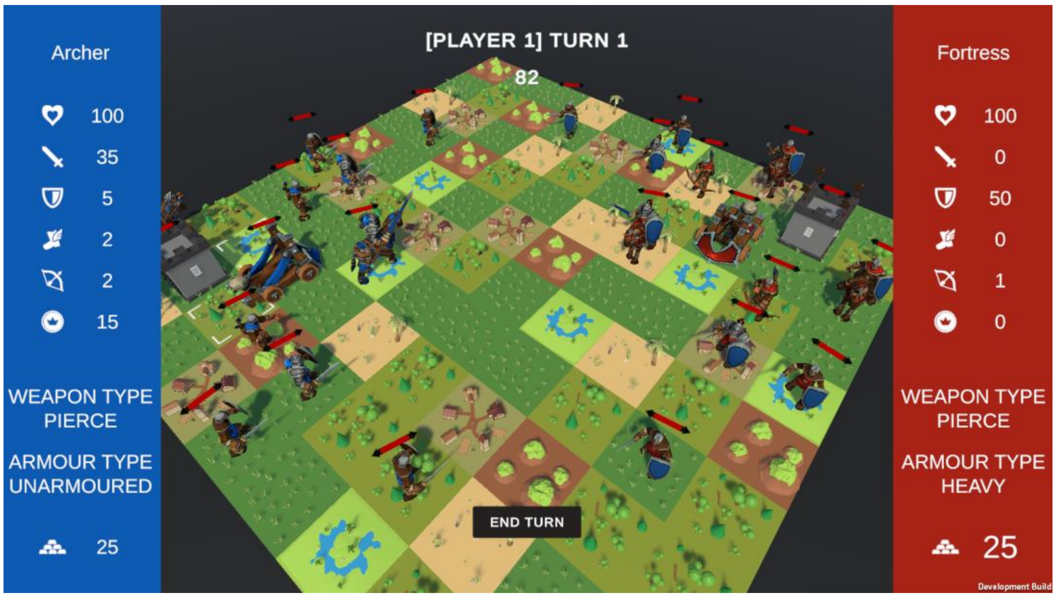


*Figure 19 – **GameScene** of the product with both player connected to the session.*

What can be expected of the current build are the following:

- Multiplayer session creation, that can be joined over a network.
- Grid turn-based system for unit interaction.
- An in-game pause menu.
- Working options menu.
- Win conditions through:
    - A player giving up.
    - Destroying the enemy Fortress.
    - Defeating all enemy that is not a Fortress.
- 6 variants of units:
    - **Militia** - Standard grounded unit.
    - **Archer** - Standard ranged unit.
    - **Light Cavalry** – Mounted unit with high mobility but low defense.

    - **Knight** – High defensive ground unit, with low mobility.
    - **Catapult** – Long ranged siege unit with poor mobility.
    - **Heavy Knight** – High defensive mounted unit, but lower mobility than its light counterpart.
- Resource management via passive generation every turn.
- Unit purchasing via a shop menu.
- UIs for unit parameters.
- Tooltips for unit parameters.
- A derivative of Fire Emblem's '*weapon triangle'* using Weapon Type and Armour Type as substitutes.
- Audio.

As promised, the game was delivered with the intended functionality of a turn-based system, that supports multiplayer. Adding onto this are features that would enhance gameplay in a manner that would aid in developing problem solving skills such as: critical thinking, and risk taking. Some of these include the resource management and unit purchasing pair found in Advanced Wars, as well as the pseudo 'weapon triangle' that were explained in the case studies. Players are expected to manipulate said systems to drag out an advantageous position, or to reverse a disadvantageous situation through strategic thinking. For example, blitzing the enemy Fortress with a catapult that deals an additional 50% damage *(due to an effective attack from Blunt Vs Heavy)*, or controlling the board with an overwhelming horde of units through unit purchasing.

5.2 – What can be Improved?

Despite having delivered the product within the deadline, there is still areas that can be touched upon to enhance the overall experience of the game. In terms of functionality, there area holes left within the Scope that could be added – this included 'Terrain' which was emphasised in the Design Brief, as well as 'Structures' which could working in tangent with resource management, and unit purchasing, to facilitate a more engaging experience of conquest that Advanced Wars demonstrates.

Processes that should be taken into consideration is the commits of GitHub. Frankly speaking, commits should be more frequent than just one per day, this is exemplified when taking the amount of functionality implemented in each commit *(Figure 8)*. These commits could easily be broken down and submitted separately throughout the development of the day, which is a more ideal scenario as it ties into the point of reverting changes. Having more commits allows for specified save points that can be reverted back and reduces lost progress from long durations of coding that may have resulted in failure.

Refactoring is another topic to take into consideration especially when factoring optimisation (admitted I never got around to non-functional requirements due to their low priority) and coupling of classes. *Figure 10* showcases three classes with singleton patterns: **UnitAction**, **GridVisualManager**, and **GridSystemHandler**. Singletons increases dependencies increasing the coupling. From a maintainability point of view, this is bad, as changing one piece of code in any of the mentioned classes can affect multiple classes.

5.3 – Conclusion

Overall, the development process can be considered a success in terms of building experience in game development in Unity, as well as acquiring knowledge on game theory. Due to time constraints which have been present throughout the project, a usability test could not be conducted that was mentioned in section 1.6, which would have helped gauge the success of the project in a more accurate manner.

Regarding development, and planning, the process was streamlined and linear due to having clear objective set in mind, when implementing. This allowed for many more features to be developed, tested, and implemented into the final build. Due to good development practices, this project can easily be built upon not just by me, but other developers, and is something to investigate in the future, when considering that the Scope still has more in store.

6 – References

Belli .B, 2020. National survey: Students' feelings about high school are mostly negative. [Online] Available at: https://news.yale.edu/2020/01/30/national-survey-students-feelings-about-high-school-are-mostly-negative [Accessed 22 April 2021]. [1]

Howarth . J, 2023. 50+ Amazing Video Game Industry Statistics (2023). [Online] https://explodingtopics.com/blog/video-game-stats [Accessed 22 April 2021]. [2]

Gilsenan. K, 2021. The next gen: getting to know kids' relationship with video games. [Online] Available at: https://blog.gwi.com/chart-of-the-week/kids-relationship-with-video-games/ [Accessed 22 April 2021]. [3]

Y PULSE, 2021. Gen Z & Millennials' Top 15 Hobbies Right Now. [Online] Available at: https://www.ypulse.com/article/2021/05/10/gen-z-millennials-top-15-hobbies-right-now/#:~:text=Eighty%2Deight%20percent%20of%20teens,%2C%20or%20drawing%20(32%25). [Accessed 22 April 2021]. [4]

Clement. J, 2018. Video game sales in the United States in 2018, by genre. [Online] Available at: https://www.statista.com/statistics/189592/breakdown-of-us-video-game-sales-2009-by-genre/ [Accessed 22 April 2021]. [5]

Your Dictionary, N/A. Strategy Game Definition. [Online] Available at: https://www.yourdictionary.com/strategy-game [Accessed 22 April 2021]. [6]

JDS Strategists, 2021. WHAT IS STRATEGY?. [Online] https://www.gameinformer.com/classic/2019/07/25/a-brief-history-of-the-fire-emblem-series. [Accessed 22 April 2021]. [7]

Hillard .K, 2019. A Brief History Of The Fire Emblem Series. [Online] https://jdsstrategists.co.uk/our-thoughts/posts/what-is-strategy/. [Accessed 22 April 2021]. [8]

Charcaolswift, 2020. Fire Emblem is Flier Emblem. [Online] https://gamefaqs.gamespot.com/boards/204445-fire-emblem-three-houses/78497236. [Accessed 22 April 2021]. [9]

Barder .O, 2023. 'Advance Wars 1+2: Re-Boot Camp' Review: Cute Carnage Prevails. [Online] https://www.forbes.com/sites/games/2023/04/23/advance-wars-12-re-boot-camp-review-cute-carnage-prevails/?sh=5461a85f12db. [Accessed 22 April 2021]. [10]

Stein. S, 2019. Fire Emblem: Three Houses review: The Switch's summer romance. [Online] https://www.cnet.com/tech/gaming/fire-emblem-three-houses-review-the-switchs-summer-romance/. [Accessed 22 April 2021]. [11]

Andrewsmd58, 2019. 250 science per adjacent mountain tile? Yes please.. [Online] https://www.reddit.com/r/civ/comments/7tll61/250_science_per_adjacent_mountain_tile_yes_please/ [Accessed 22 April 2021]. [12]

Bramble . J, 2022. HOW LONG DOES IT TAKE TO MAKE A VIDEO GAME?. [Online] https://gamemaker.io/en/blog/how-long-to-make-a-game#:~:text=Creating%20an%20indie%20game%20can,Cuphead. [Accessed 22 April 2021]. [13]

Kinsta, What Is GitHub? A Beginner's Introduction to GitHub [Online] https://kinsta.com/knowledgebase/what-is-github/ [Accessed 22 April 2021]. [14]

7 – Appendix

GitHub Repository
https://github.com/Chungj2000/CS3FYP

Trello
https://trello.com/invite/b/4s4Zh5gV/ATTIa21ed244f89fd81309a597bfa92918588C7702CD/final-year-project

CS3IP Unit Action Handling Sequence Diagram
https://drive.google.com/file/d/17us0v2ZiXSNeYo_BR-JynixZtEXtoMag/view?usp=share_link

CS3IP Grid Action Class Diagram
https://drive.google.com/file/d/1xeeawsne5r8p5okQIGleLi1N4W9AJcVx/view?usp=share_link

CS3IP Unit Action Activity Diagram
https://drive.google.com/file/d/1R4kALw8bCMTYBXGv5SgrHDSyBdKMJfZQ/view?usp=share_link

CS3IP Unit Class Diagram
https://drive.google.com/file/d/1OZiwQlCdNiUDWrVoFE1lakoWhZEHFnDN/view?usp=share_link

CS3IP Project Definition
https://docs.google.com/document/d/11g79lD4U1PWUILoYpJYkCcfq3XenNaZv/edit?usp=share_link&ouid=110924098538696911211&rtpof=true&sd=true

CS3IP Design Plan
https://docs.google.com/document/d/15BlQVQb4QDDVLRRG5eDenSd25sQSNx3R/edit?usp=share_link&ouid=110924098538696911211&rtpof=true&sd=true

CS3IP Requirement Document
https://docs.google.com/document/d/1Yo8SYZf_ltpbJNooh187VBH4EfdYKthA/edit?usp=share_link&ouid=110924098538696911211&rtpof=true&sd=true

CS3IP Test Plans
https://docs.google.com/document/d/1-WSzqLis6_J-RtFfR8efbWX5UxGg87Au/edit?usp=share_link&ouid=110924098538696911211&rtpof=true&sd=true

CS3IP Use Case Description
https://docs.google.com/document/d/1qhno00rRPphd78j9TX2x8t5aqehYMokD/edit?usp=share_link&ouid=110924098538696911211&rtpof=true&sd=true