Investigating the methods used to create challenging AI opponents within RTS games

Written by Thomas Stevens

Enrolled Course: *Computer Games Design & Programming*

Student Number: *21012473*

School of Digital, Technologies and Arts

Contents

1.0 Abstract

Realtime strategy (RTS) games are a sub-genre of strategy games which are characterised by the fact that all decisions and movements happen in real-time, unlike their turn-based counterparts. Calculating many decisions at runtime however can make it difficult to engineer challenging AI. Therefore, this paper investigated the different techniques and development processes used by the games industry and academia, to determine which ones are the most optimal for creating more challenging AI opponents within RTS games. These methods were tested in a brand new RTS framework developed in Unreal Engine 5, utilising C++ for greater performance. This framework included both the core mechanics of an RTS game, but also a basic AI structured according to practices researched in the literature of this field. The methods researched were then implemented into the decision-making process of this basic AI. These were then tested against human participants to see from both data collected on match statistics, as well as participant opinion, on whether or not the AI they played was challenging. This paper found that methods such as dynamic difficulty scaling and the Lanchester combat prediction algorithm were deemed more challenging by participants and supporting data, rather than methods like allowing the AI to cheat.

2.0 Introduction

Artificial Intelligence (AI) is a major building block of the strategy game genre, this is because most titles have single-player modes, which requires a form of computer opponent for players to compete against.

In turn-based strategy games such as Sid Meier's Civilisation VI (Take-Two Interactive, 2016), players – both human and computer – must manage resources, construct units and plan attacks on their opponents. All these actions make up a players turn and after making a turn, the player progresses to the next turn by pressing a button. There is a loading buffer that takes a few seconds before letting the player take their next turn, the buffer duration varies depending on the number of AI factions in game. This buffer allows the AI ample time to evaluate the game state and make moves based on it. In turn this can lead to more challenging experiences as both players and their computer opponents can properly plan moves.

By stark contrast however, in real-time strategy games there is no luxury for human or computer players on having ample time to contemplate their moves. This is because everything runs in real-time at a constant refresh rate between 30-60 frames per second, depending on the graphical intensity of the game. Players must simultaneously deal with multiple tasks such as collecting resources, building units and managing potentially several battles on different front lines. This leads to the AI also being more complex than their turn-based counterparts as the AI must also be able to handle all these tasks and adapt to the constant changes the real-time setting will cause throughout a match.

The problem that will be investigated in this paper is how to make AI within RTS games more challenging. As previously mentioned, turn based strategy features a loading buffer allowing the AI ample time to make a move and think ahead like in Chess – a game where AI can already beat humans, such was the case when world champion Garry Kasparov lost to the chess engine deep blue in 1997 (Hoekenga, 2007). RTS games however do not have this luxury, so a large amount of computing power is needed to determine strategies at runtime. The larger the scale of a game, the more computing power is required to make decisions. An example on how large strategy games can be is Stellaris (Paradox Interactive, 2016). In this title a player must manage a galactic scale faction that has to colonise planets and systems to expand their territory and resource pool. As well as this, they must also build large military and civilian fleets to fight wars and construct infrastructure respectively. The matches taking place in an entire galaxy with up to 30 AI empires spread across up to 1000-star systems, which can take up a lot of processing power. As a result, to make their opponents appear more challenging, without developing more adaptive and potentially processer heavy AI; the game will either give the player or AI economic and research bonuses to help them progress quicker.

This approach does make opponents more challenging, as the AI will always have a larger starting force due to economic bonuses, but through resource optimisation, a skilled player can catch up and nullify these handicaps. The AI essentially loses its lead and presents little challenge past this point, as the strategies utilised are the same as at lower difficulties. Therefore, this paper will look into other ways to create a more dynamic and challenging AI opponent for players that provide a challenge throughout the entire duration of a game, not just the early to mid-game.

3.0 Aims and Objectives

The aim of this paper is to investigate what techniques and development processes can be utilised to develop more challenging AI opponents for players to compete against within RTS games. This will be achieved by studying techniques used by both the games industry and academics who specialize in this field of research. This research will culminate in an RTS prototype, that will feature AI systems developed using a combination of techniques. This prototype will then be play tested to see which systems were deemed the most challenging. Three main research questions will be considered when conducting this study:

- RQ1: What measures are used to measure the difficulty of AI in RTS games? Is it an AI that will always try to be as difficult as possible for the player to defeat? Or is it an AI that will present the player difficulty during a match but allow itself to be overcome if the player is skilled enough?

- RQ2: What are the optimal design philosophies for designing RTS AI? For instance, how should an RTS AI be structured and how can structuring affect the level of challenge an AI will pose to a player?

- RQ3: What are the optimal algorithms utilised for combat prediction, that are used to create a challenging AI system? Specific areas that will be investigated regarding this research question include:
    - Predictions Algorithms
    - Counter systems

4.0 Literature Review

4.1 Tactics for RTS games

4.1.1 The building blocks of RTS games

RTS games are a real-time war simulation where players select a faction and compete against other factions, which controlled either by other players in a multiplayer setting, or more typically AI controllers in single player. A faction in the context of RTS games is a controllable entity that has their own unique appearance for units and buildings. Some factions may also have unique units and mechanics which gives every faction their own unique strategy when playing. Despite these advantages however, developers must ensure that all the factions are balanced to ensure a fair game.

The main factor of importance in a real-time strategy game is the economy and resource management (Oluwafemi J, Akinde, 2014). Factions must be able to gather resources either passively through capturing resource structures such as in the skirmish mode of Star Wars: Empire at War (Petroglyph Games, 2006) or by harvesting resources utilising workers such as in Age of Empires (Ensemble Studios, 1997). These resources can be used for a variety of factors such as upgrading their technology, resulting in stronger units and defences, as well as the initial construction of those units and buildings.

The most common win condition of RTS games is to defeat the opposing factions within a game. Therefore, the battle system is a critical part of the infrastructure of RTS games which allows factions to attack each other's military units and buildings. As a result, RTS games have a wide variety of military units (Robertson, Watson, 2014) that factions can utilise, as the more potential unit types that are on offer, the greater the combinations of units that can be used to achieve a balanced fighting force (Oluwafemi J, Akinde, 2014). If all the units within an RTS game are balanced it will prevent the use of rush tactics that only involve one unit.

When it comes to the user interface (UI) design of an RTS game, they can be simple or extremely complex. Simpler user interfaces are more effective as they convey the information the player needs to know in a clear and concise manner. This makes it easier for the player to understand and makes the game less of a challenge as whole. Complex user interfaces are required for RTS games with a lot of game mechanics, as simpler user interfaces cannot handle all the information a player requires, however, developers run the risk of confusing the player and making the game have a steeper learning curve.

4.1.2 The role of an AI opponent

In real-time strategy games that possess a single player mode, the human player requires other factions to fight against. In the absence of other human players to control these factions, as is the case in multiplayer modes, an AI controller will step in and take control of an enemy faction. They will then attempt to play the role of a human player and order their units to gather resources and attack the players' forces with the aim of winning. However, winning is not the true goal of an AI opponent, as according to Davis' (Davis, 1999) first law of computer game AI: 'The goal of any AI is to lose the game.'

The reasoning behind this law is that attempts to make perfect AI opponents, that use all the optimal tactics, would be undesirable (Davis, 1999). This is because, despite players wanting a more challenging opponent, they would not want to lose constantly to one; rather they appreciate the challenge of an opponent who may defeat them a couple of times, but eventually they will be able to overcome the AI. This rewards the player with a great sense of accomplishment in the end, as they have beaten a challenging foe. So overall the role of an AI opponent within an RTS game is to challenge the player by creating an exciting ebb and flow of power, keeping the player engaged and

threatened throughout the course of the game, but in the end, if the player is skilled enough, will lose to them.

4.1.3 Strategies used in strategy games

In RTS games, both the player and AI can utilise various strategies to claim victory. Therefore, in every strategy game, there are three broad categories of playstyles that can be used. The first is the "Boom" strategy (referred to as economy building in this paper), this refers to prioritising faction upgrades and building up the economy of the faction in order to get the most powerful units first (Bycer, 2025).

The second is rushing or a rush-attack. This is a strategy that results in crafting an army with a large number of low-level units and attacking your opponent as soon as possible. The aim is to destroy key buildings before they are able to craft powerful units to defend (Li, 2008). This strategy if used correctly, can cripple or even defeat an opponent in the early game, although the user can be worse off if the attack is successfully defended.

The last common strategy is turtling, this can be used alongside economy building as the principle of turtling is to use a slow and cautious playstyle and focus on heavily defending a particular area (Deriglazov, 2018). In the context of RTS games, turtling is when one player focuses on defense only to shut down any incoming attack, with the goal to survive long enough to get more powerful units than their opponent to launch a counterattack.

These three exist within a rock, paper, scissors dynamic, as rush can be used to prevent people building their economy, economy building can be used to gain enough resources to overpower turtles, and turtling can be used to fend off rush attacks.

4.2 Structuring of AI systems in RTS games

4.2.1 Basic Structuring

AI systems within RTS games are structured by breaking the AI down into different managers with each one being responsible for a task a faction would be required to do (Scott, 2002). The number of managers utilised can be altered dependent on the game's requirements, however this approach always requires a civilisation manager. This manager is responsible for coordination between the lower-level managers and controls when a civilisation is required to upgrade their technology. Scott also describes 5 more lower-level managers when using this structure type:

- The unit manager controls unit recruitment and can have the option to keep track of the players unit count and try to train an equal number of units to the player for balanced combat.
- The build manager, which is responsible for the placement of structures in a game, coordinates with the unit manager and can be sent requests to build unit training buildings, so the unit manager can construct units. This module also handles terrain analysis when placing buildings as well, since buildings will have certain requirements on where they must be placed.
- The resource manager, which is responsible for tasking workers to gather resources in response to orders from both the unit and building managers.
- The research manager which controls the upgrading of the civilisation's technology tree. Upgrades are selected based on cost, effectiveness and the current technology level of the player.
- Lastly the combat manager which is responsible for directing military units to attack enemy civilisations. It coordinates with the unit manager to request more units, as well as keeping track of offensive and defensive duties via individual personnel managers.

4.2.2 Behaviour Trees and State Machines

As well as breaking down the AI into managers controlling different tasks, the code within these managers should also be discussed. A common approach to structuring any sort of AI system whether that be a non-player character (NPC) or an overarching AI, will be using either state machines or behaviour trees. Both methods offer a way for agents to make decisions autonomously depending on certain conditions or events, which is something the previously discussed manager system would rely on, due to their communication between each other.

A finite state machine (FSM) is a basic mathematical model of computation that consists of a set of states along with transitions and events to arbitrarily swap between said states. These were widely used in autonomous agents before the creation of behaviour trees (Colledanchise, Ogren, 2018). They feature a common structure and are easy to implement and understand. The downside with FSMs, however, is they lack modularity as adding and removing states gives rise to the issue of re-evaluating the entire process.

Behaviour trees were developed by the computer games industry as a tool to increase the modularity in control structures NPCs. Before their introduction, FSMs used to be the norm for NPC controls. Behaviour trees are a directed rooted tree comprised of root and leaf nodes, operations start at the root node and traverse through leaf nodes depending on a certain condition that needs to be met (Colledanchise, Ogren, 2018).

4.2.3 The Dark Reign Model

The method of breaking down an RTS AI into different modules is standard practice, at least in relevance to the games industry. Another example that uses this concept is The Dark Reign Model (Davis, 1999), a model used to develop strategy games by the games company Blizzard. It breaks down an AI into three main modules: an analysis module, resource allocation and high-level AI module. The analysis module breaks down strategic goals and ranks them in order of how high a priority a particular goal is, whereas the resource allocation module takes the goals from the analysis module and allocates troops to the goals. The high-level module changes the parameters that the former two modules use, dependent on the state of the game or specific scripted events. This approach to tackling the structure uses a lot less modules than the basic structure described previously, however utilises a way to change parameters in the high-level module to easily change how an AI may behave. Using an approach that allows easy modification of a systems behaviour will allow a developer to better tweak their designs based on feedback to create a more challenging opponent.

4.2.4 Data-Driven Design

Another approach of structuring an RTS AI, is by using data-driven design for the decision-making processes rather than behaviour trees or state-machines. This involves giving each control module a set of actions to choose from. The way the AI determines which one to choose, is by choosing the highest weighted action out of the selection. This approach is utilised by the AI systems in Stellaris (Bari, 2017). By utilising this approach, it cut down the amount of work the Stellaris programmers had to do and allows better balancing changes as the AI can be completely adjusted outside of the codebase. The way weights are determined are by triggers that are enacted either at the start of the game, after the AI has been randomly generated, or by in-game events. The weights are selected by either picking the highest weight outright or weighted randomisation. Weighted randomisation works by having a select number of weights, the weights are summed to get a total value, then each weight is converted to be a percentage value of the total. This percentage represents the likelihood of that weight being chosen. This approach to creating RTS AI can be useful especially in the context of

adding new mechanics to an expanding game, as it just involves adding new weights when AI new mechanics are created. It is also useful performance wise, as there is no tree-traversal through behaviour trees that needs to be done; so more graphically demanding games may favour from this approach to improve performance.

4.3 Methods to create more challenging AI

When it comes to creating AI opponents that are more challenging for a player to play against, both the games industry and academics have different approaches to tackling this problem. One main factor that contributes to this difference is the constant increase in graphical quality in newly released games. As graphically impressive games sell better (Buro, Furtak, 2004), more processing power is utilised for rendering rather than computing for AI calculations. Therefore, to give players who want a more challenging experience in RTS games, the industry will use shortcuts to get this result. Since the research side of the field however spend their time creating AI for older games like StarCraft (Blizzard Entertainment, 1998) or using simple simulations in software such as the Open Real-Time Strategy engine (ORTS); their methods result in more challenging AI without utilizing shortcuts.

Additionally, there is little demand for new AI techniques in the games industry. Current techniques are not viewed as an obstacle to create an AI that is challenging and fun to play against (Robertson, Watson, 2014). Studios are usually under severe time constraints when developing, so there is little incentive to create and test new AI methods that could result in more challenging opponents, as they are costly to create (Buro, Furtak, 2004). Considering this fact, it is no surprise that studios use the same techniques and resort to shortcut methods like cheating to enhance the difficulty of their games against AI opponents.

4.4 Methods utilised by the game industry

4.4.1 Cheating

Allowing their AI opponents to cheat is a method that is widespread throughout industry made RTS games (Robertson, Watson, 2014). This cheating however can be in a variety of forms that would allow an AI opponent to have an edge over the human player.

4.4.2 Cheating the fog of war

One strategy used in turn-based and real-time strategy games alike, is allowing AI opponents to see through the fog of war. The fog of war is a visibility limiter that prevents human players from seeing the whole map at once – which would give a great tactical advantage to any faction, whether human or AI. Due to the fog of war, human players can only see areas of the map properly if they have a unit or building stationed there. The number of tiles visible around a given building or unit is dependent on the individual's visibility range, which makes it particularly difficult to determine an opponent's strategy due to the incomplete information about an opponent's actions (Kabanza et al, 2010).

Cheating this data by removing the fog of war for AI opponents, can make them more challenging as they can watch their opponent's faction move and station forces in certain areas, therefore allowing the AI to launch a surprise attack on their opponent. Alternatively, they could see when the enemy faction is sending an army to attack and mount a defence to fend them off. While this certainly would make the AI more challenging, developers must be careful with this approach as it is difficult to hide from human players. Therefore, if the cheating is obvious the game will lose its fun factor (Davis, 1999).

Even though the fog of war is usually ignored by AI opponents in strategy games, it is also done for performance reasons. When dealing with AI and a fog of war, a developer would have to code in a memory system to the AI, so it can remember where it saw units before they disappeared into the fog and predict where they might end up if trying to attack them. This is a problem the developers of the Civilisation franchise had (Johnson, 2008), and it was decided to let the AI have information cheats and limit the amount they can use rather than code an AI compatible with the fog of war. This is because it would be too resource intensive and would affect game performance. So, allowing the AI to have information cheats by ignoring the fog of war can be an effective method of increasing RTS AI opponent difficulty. However, the amount of information an AI is allowed to gain must be limited to avoid it being too obvious and ruining the 'fun' a player may experience.

4.4.3 Cheating the economy

Another method of cheating used in strategy games, is cheating the economy of an AI opponent. In games such as the civilisation franchise, to increase the difficulty for the human player, the main thing that is done is reducing the cost of everything for the AI. With the hardest difficulty in the game Civilisation IV (Firaxis, 2005) being Deity, which gives every AI faction a 40% reduction in prices across the board (Johnson, 2008). This increases the difficulty for the human player as they must deal with AI opponents that will most of the time, have more units and higher technology than them. It is still possible to win through clever strategies but when one's opponents have armies twice the size of one's own, it is more of a challenge.

Aother clear example of a game that uses economic cheating is Stellaris (Paradox Development Studio, 2016). Rather than reduce the cost of every item however, the difficulty increase instead comes from maintenance costs and boosted resource production. In the game depending on how large your navy is, affects how much upkeep you pay – so large fleets that take up your entire naval capacity can be difficult to achieve without a good resource income. On its hardest difficulty – grand admiral – the AI opponents have their upkeep reduced by 40%. As well as this, resources generated

from colony populations are increased by 75% (Paradox Development Studio, 2016). These egregious bonuses will in the end allow the AI to not only have larger fleets than the human player but also be able to replenish them quickly with large resource pools. Once again, this does make the AI much more challenging, but it reduces the amount of viable tactics that can be used by the human player so can make it less fun since the cheating is so blatant (Davis, 1999).

4.4.4 Data-Driven Design

Although this topic was discussed in section 4.2, it will briefly be considered about in the context of how data-driven design can make more challenging AI opponents. In Stellaris, AI opponents have different traits that are randomly generated depending on what weights each trait has in the generation process. (Bari, 2017). These traits will alter the behaviour of the AI and can make your game more challenging as a result without adjusting the difficulty. For example, if the first AI faction the human player meets is a machine empire that has the trait determined exterminator (Paradox Development Studio, 2017), they will declare war immediately. This is because that trait will make the AI want to declare war and purge any faction that is made up of organic lifeforms. As a result, these randomised traits can make the game a lot more difficult if the human player start is next to a faction likely to declare an early war due to xenophobic ideology.

4.5 Methods utilised by academia

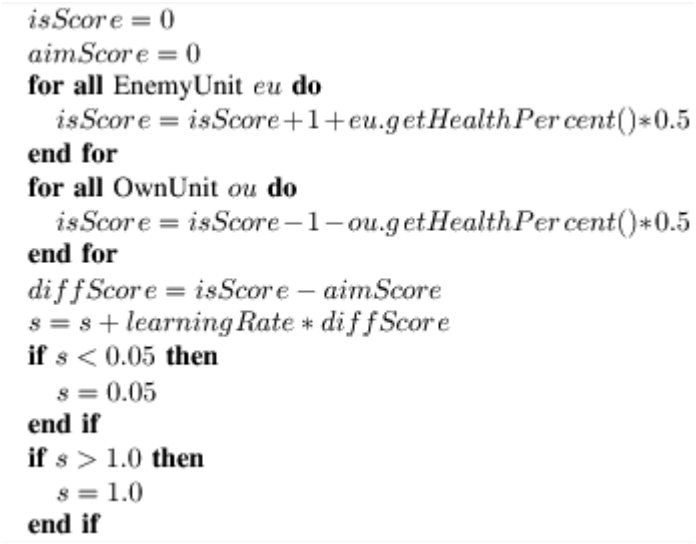
In terms of the research of improving RTS AI, there are many more methods that have been developed to create a more challenging opponent for the player. The testbeds that are used by the following methods are the RTS game StarCraft and simple RTS simulations using the Open Real-Time Strategy engine (Buro, 2003), referred to in this paper as the ORTS. The ORTS is an open-source real-time strategy game engine developed by the university of Alberta. It has an extensive scripting system that supports many different types of games. (Hagelbäck, Johansson, 2009).

4.5.1 Dynamic Difficulty Scaling

Although this is a concept utilised by the games industry under the term dynamic difficulty adjustment (DDA) and is used in a variety of titles, with the Crash Bandicoot series of games (Naughty Dog, 1996) being some the first to use it. However, with 84 research papers published between 2009 and 2018 (Zohaib, 2018), it is a field well explored by academics. Hence the placement of it within this section.

Dynamic difficulty scaling means that the difficulty of the game is adjusted during the game to suit the skills of the human player (Hagelbäck, Johansson, 2009). The purpose of this is to give the human player consistent challenge throughout the game, as usually their skill increases the more they play. This is opposed to static difficulty which is set by the player at the start of the game.

A further development on this concept is called: Rapidly Adaptive Game AI. This was an approach where the difficulty in the RTS game engine Spring (Spring Engine, 2007). is adapted at runtime utilising observations of the current game state to predict the likely outcome of a given game (Bakkes et al, 2008).



$isScore = 0$
$aimScore = 0$
for all EnemyUnit eu **do**
  $isScore = isScore + 1 + eu.getHealthPercent() * 0.5$
end for
for all OwnUnit ou **do**
  $isScore = isScore - 1 - ou.getHealthPercent() * 0.5$
end for
$diffScore = isScore - aimScore$
$s = s + learningRate * diffScore$
if $s < 0.05$ **then**
  $s = 0.05$
end if
if $s > 1.0$ **then**
  $s = 1.0$
end if

Figure 1: Equation used to determine the AI difficulty (Hagelbäck, Johansson, 2009)

The adaptive difficulty algorithm utilised by Hagelbäck and Johansson (Hagelbäck, Johansson, 2009) works by evaluating a score to determine how beatable an AI faction should be. This algorithm works by evaluating the relative strength of both the human and AI factions by utilizing the formula in figure 1 within a for loop to determine the total combined health of all units within a faction.

The difference in the unit health percentage of each unit from each faction is found and determines the evaluation score. A positive score means the human player is in the lead, whilst a negative means the AI has the higher unit strength. The aimScore is the value of strength the bot aims for, which is set to 0, so the aim is that both sides possess equal unit strength. A higher positive value for the aimScore however, will tell the bot to aim to always let the human player have a slight lead. Difficulty is determined therefore by the value of s, the higher the value of s is between 0 and 1, the more difficult the AI becomes.

Although the results from the study measured enjoyment and fun the test subjects experienced when combating the AI factions, they found that the AI with an aim-Score of 0.4 to start with that had the adaptive difficulty scaling functionality was the most fun to play against. This approach is a good way to develop challenging AI opponents that build up their difficulty as the game progresses. That way, the human player faces a constant challenge throughout the game and not just until they overpower the AI, such is the case in games with static difficulty.

4.5.2 The Lanchester Model

It is challenging for AI opponents to estimate combat outcomes of a battle accurately, attempting to do so by running simulations is a popular method but is very resource heavy. The Lanchester Model is an outcome evaluation model, based on Lanchester's attrition law (Lanchester, 1916), which takes into consideration forces made up of different unit types and the fact that units can enter a battle with any percentage of their maximum health remaining (Stanescu et al, 2017).

For modern or futuristic themed RTS games, most units are ranged or can target multiple enemies at once, as a result this model utilises Lanchester's Law of Modern Warfare, as this law is intended to apply to ranged combat, since it quantifies the value of the advantage of having a larger army. This law does have nothing to do with weapons range however, rather the rate at which new targets can be acquired, since ranged weapons can engage targets as fast as they can shoot and kill them. The aim of using this law is to, before engaging in combat, determine the army sizes and calculate the victorious faction and what the remaining army size of the victorious faction will be.

However, most RTS games feature large armies comprised of a mix of different unit types, all with different values for their health and damage output, that must be factored into the prediction equation. Suppose an army has two different unit types, referred to as G1 and G2 with effectiveness values of g1 and g2. To calculate the effectiveness of the mixed unit army, an equation must be used as seen in figure 2 that takes in the into account the total count and effectiveness of each portion of the army, divided by the total army size (Mackay, 2006).

$$g_{\text{ave}} = \frac{g_1 G_1 + g_2 G_2}{G}$$

Figure 2: Equation to calculate a mixed unit army's strength (Mackay, 2006)

To determine the effectiveness of a single unit, an evaluation of their strength is needed for each unit involved. These can be determined statically or through machine learning simulations. One method to determine a unit's strength without machine learning however, is an equation considering unit cost and health attributes, as seen in figure 3 where α represents unit strength.

$$\alpha_i = \frac{\text{Cost}(i)\text{HP}(i)}{\text{MaxHP}(i)}$$

Figure 3: Equation to find unit strength (Stanescu et al, 2017).

This approach can work but it lacks any attributes on a unit's attacking strength, the amount of damage per second a unit can inflict on an enemy target, and any defence modifiers a unit may possess. For example, some strategy games utilise armour values on their units which means any incoming damage is reduced by a certain percentage. Learning a unit's strength values through logistic regression is the other method tested, as it can use data from hundreds of battle simulations to determine how effective a unit is in a combat situation against different units.

Testing of this model was undertaken utilising UAlbertaBot – an open-source StarCraft AI (Churchill, 2013) – which runs combat simulations to decide if it should attack the enemy, based on if a win is predicted with the current units possessed. A simulation call in this procedure was replaced with a Lanchester model-based prediction and tested against the 6 top bots from the 2014 AIIDE StarCraft AI tournament, with 200 matches being played against each bot per tournament. Three tournaments were played in total, with the first using the UAlbertaBot default settings, the second using statically determined strength values and the third using learned values.

The average win rate of the default settings was 60%, using statically determined values it increased the win rate to 63.9% and the learned values had an average win rate of 69.7% (Stanescu et al, 2017). These results show the Lanchester model could be a viable alternative for combat prediction algorithms already used in RTS games, since the data conveys a stable increase in win rates. Leading to a potentially more challenging AI for human players to face, since the AI can effectively determine the optimal times to attack or retreat based upon the current game state, a feature some mainstream RTS games lack.

4.5.3 Monte Carlo Tree Search

One challenge faced when creating AI for modern strategy games, is dealing with the complexity of the game itself. With many potential moves that can be made, especially in the RTS genre when these must be performed in real-time. AI programmers can be faced with issues when trying to encode the way an AI chooses which action to perform.

Utilising search techniques present an interesting solution as it allows an AI to cycle through possible moves and perform the one deemed best, dependent on the current game-state. Search techniques like minimax iterate over all possible moves, evaluating the possible result of each, then returning the move deemed best. Unfortunately, these techniques do not work in strategy games, especially RTS games, as there are too many moves to explore within a reasonable time frame (Roelofs, 2017).

One search technique can be utilised however, due to its capability of handling complex amounts of data, this being the Monte-Carlo Tree Search (MCTS). MCTS relies on an intelligent tree search that balances exploration and exploitation. It performs random sampling in the form of simulations, then stores statistics of actions to create better educated choices in each iteration (Świechowski et al, 2023)

Each iteration of an MCTS process contains four phases – as seen in figure 4 – these are: selection, expansion, simulation and finally backpropagation. In the selection phase, while the state is found in the tree, the next action is chosen according to stored statistics in a way that balances between exploitation and exploration. In the expansion phase, when the game reaches the first state that cannot be found within the tree, the state is added as a new node. Therefore, adding a new node for each simulated game. In the simulation phase, actions for the rest of the game are selected at random; however, the weighting of actions does impact the level of play as if all actions are equally weighted, strategies will be suboptimal. Lastly, at the end of the simulated game, the backpropagation phase occurs. This updates each tree node that was traversed during the simulation and the visit counts are increased and win/loss ratio is modified according to the outcome (Chaslot et al, 2008).

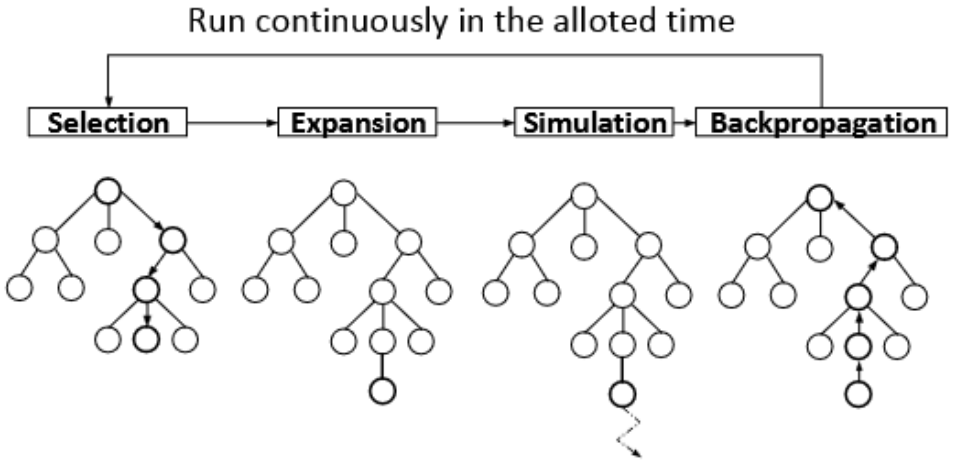


Figure 4: The Monte Carlo Tree Search Algorithm (Chaslot et al, 2008)

Despite MCTS being a better suited search technique for strategy games, complexity still presents issues for the algorithm; paired with the fact that it also struggles with problems that have a very narrow path of victory and success, makes it poor at tactical decision making (Roelofs, 2017).

Roelofs (Roelofs, 2017) found that these pitfalls can be rectified however with adjustments at each phase of MCTS, which led to an AI that players found more challenging as a result. During the expansion phase, by dividing moves into actions, MCTS can understand that move B, which is a slight variation of move A – a good move explored earlier – may be a good move as well. This improves performance also as instead of expanding over all possible moves at each expansion step, a certain action is selected within the current move and expand into the actions defined in its action set.

In the simulation phase, complexity can ruin performance. Building an abstraction of the conflict that needs to be simulated by defining a function that, given a setup, calculates a loss or victory, saves time in the simulation phase and speeds up the rate an AI can calculate good move.

Within the backpropagation phase, since MCTS has a poor track record for tactical decision making (Roelofs, 2017), utilising an evaluation function that severely punishes the AI for losses will result in MCTS displaying a paranoid behaviour as it will only select nodes that give the enemy little to no chance of winning. It also allows MCTS to move away from nodes that lead to negative reinforcement and allow it to explore nodes that have never encountered a loss, which may be moves not tried yet. This improves the tactical decision making of MCTS and can lead to a more challenging AI opponent as a result. Finally, the optimisation used to improve the selection phase, is an optimised variation of the insertion sort algorithm to cache the rank of each node and store which node can be selected next. This improves performance as the rank of each node does not need to be repeatedly recomputed.

In short, MCTS proves to be a useful search technique for games with larger data volumes that need to be analysed, so by using the optimisations presented by Roelofs (Roelofs, 2017) to increase performance of MCTS even more, whilst making up for the poor tactical decision making MCTS tends to have, results in an effective method to program challenging AI opponents.

4.5.4 Fuzzy Case Based Reasoning

Another approach to developing challenging AI opponents is with the use of Case-based reasoning (CBR). CBR is an approach to problem-solving and learning that can utilise knowledge from previously experienced situations. It is a concept inspired by the human skill to solve problems by generalising over previous observations in a restricted problem domain (Cadena, Garrido, 2011). An evolution of this concept is the combination of Fuzzy sets and Case-based reasoning (FCBR), with the Fuzzy set theory being a powerful method for mapping vague inputs to a precise output using linguistic rules. Since the real-world information is vague and partially true, it creates an environment that is fuzzy (Surucu et al, 2023).

A fuzzy set is the use of classical set theory with fuzzy logic applied to it. In classical set theory, a membership function is used to determine whether an object belongs to a set or not by assigning it a value, which is either 1 it belongs to the set, or 0, it does not. Using fuzzy logic however, the idea is to associate a number with each object indicating the degree to which it belongs to a set, rather than the typical Boolean value of classic set theory (Kamble, Rewaskar, 2020).

This combination of Fuzzy sets and Case-based reasoning was utilised to manage the tactical reasoning component of a StarCraft AI bot, as well as using CBR for managing the strategic reasoning of the bot. FCBR simplifies the process of knowledge representation, whilst enabling the knowledge acquisition practice using a case base. Using linguistic variables within the case base significantly reduces the large number of actions and objects that an RTS game possesses. Therefore, allowing FCBR to deal with the vast space of actions presented in an RTS game whilst incorporating human knowledge in the reasoning process. Furthermore, the tactical agent proposed in this method is based on a CBR approach and uses a fuzzy representation of cases, allowing the AI to deal with abstract and incomplete information. This abstract representation of the game state closely mimics human thinking

as human players tend to think in an approximate form, rather than exact. So fuzzy theory deals with this kind of reasoning (Cadena, Garrido, 2011). This human like thinking could potentially result in an AI opponent that feels more realistic and challenging to play against.

Rather than using a goal-based system, Cadena and Garrido used a case library formulated of cases composed of features that represent the state of the game itself as well as the actions executed within that state. To construct the case base, a replay of an experienced human player playing against the built in StarCraft AI is saved in a format that enables replay of the game. This replay is analysed by the Brood War Application Programming Interface or BWAPI (Kovarex, 2011), then, the features that represent the state of the game and strategic actions are added to the case base.

The results of this method when pitting this AI against the default StarCraft AI are promising. The tests conducted being split up into 3 sections: one using the same start positions as gathered in the replay the case base is based on; the second using the selected position for the bot, but a random position for the default StarCraft AI; the third having both bots start in random positions. Each test having 100 matches played. Across all tests the average win rate was 60% with the best win rate being the random start positions for both. This shows that the bot is already a more challenging opponent than the default StarCraft AI. If more than one replay, showing various expert human player tactics were used to construct the case base, this approach could yield a very challenging AI opponent for human players to face. Especially since this method results in an AI that has more human like thought processes.

5.0 Research Methodologies

To clearly understand how to create challenging AI opponents for players in the RTS game genre, an experiment will be conducted to test each of the methods discussed in the literature review independently of each other. The results of each method will be compared to determine which of the methods are beneficial to implement when attempting to create a challenging AI opponent.

Before any testing can be conducted, an RTS game is required as a testbed to input these methods into. Many platforms were considered, such as StarCraft and the ORTS, since both are prolifically used in the RTS AI research field (Robertson, Watson, 2014). However, the aim of this study is to determine the best methods for creating more challenging AI in any sort of RTS game, and since the majority of these have been designed around the aforementioned titles, it would draw more interesting results to test them in a completely new RTS game prototype. It would also test to see if the methods used by academics can be easily plugged into existing AI architecture of new RTS games.

5.1 RTS Development

To develop a RTS prototype to test these methods, a few factors will need to be considered such as the game engine to develop in; the programming language to utilise; the game mechanics of the RTS; the setting of the game and lastly the unit types.

The engine that was chosen for this project was Unreal Engine 5 (Epic Games, 2022), specifically version 5.4.4. The reason this engine was chosen was due to the extensive built in systems that would streamline the development of the project, allowing more time for the development of the AI itself. Features like the Navigation volumes would allow easy development of the unit movement, and the built-in blackboard and behaviour tree systems would speed up AI development. As for the programming language, despite Unreal Engine featuring the Blueprint scripting language, C++ was opted for as it allows more direct control over components, something that blueprint lacks. C++ also features the advantage of manual memory allocation, resulting in faster performance times in compiling and running the code. This faster performance will be beneficial due to the high volume of processes an RTS AI will need to run.

As for the mechanics and gameplay idea of the prototype this was heavily inspired by the space skirmish mode in the game Star Wars: Empire at War (Petroglyph Games, 2006). In the skirmish mode two factions on opposite ends of a space map start with a space station and handful of basic units. Players can only build units and upgrade technology level from this space station and if it is destroyed, they will lose. Resources are gained per second and credit mines can be captured around the map to increase the income per second. The prototype will use the same mechanics and design principles as it ensures matches played are short and fast paced. The one major difference will be rather than set in space, this prototype will have a naval theme and be set in an ocean. Lastly there will be five unit types for both the AI and players to choose from, each based on a class of naval ships. This variety ensures some level of strategy is used as each unit will have different strengths and weaknesses.

After the prototype has been developed, the enemy faction will require AI functionality so it can be controlled and fight the player – as is the case in every real-time strategy game. This AI will be developed utilising Unreal Engine's Behaviour Tree system, so it can manage the decisions it will need to make. The structure of the AI will be based upon a combined approach of the dark reign model (Davis, 1999) and the approach laid out in section 4.2.1, which involves breaking down the AI into different managers with each one being responsible for a specific faction task (Scott, 2002). These approaches seem to be used heavily by the games industry, so it is the logical choice to base this RTS AI around those principles.

When the AI has been completed, the next stage will be to create five identical levels, the difference between these levels will be slight changes within the code of the AI itself. Most methods discussed within the literature review are sets of equations swapped out in key points within the decision-making process of the AI. This means these methods can be injected into the AI's code, so that each level features a functionally different AI. Two levels will have no changes to the AI code, this being the first one to use as a baseline for the results, the other level will have changes made to the income of the enemy faction to test the industry method of cheating resources to make the AI more challenging (Johnson, 2008).

5.2 Data Types

For this study, since participants will be attempting to defeat AI opponents in a one versus one match. The data required from each participant will, state whether they beat the AI opponent, rate the bot's difficulty, and detail the strategy used. As a result, a mixed methods approach will be required since these questions will result in both qualitative and quantitative data. The use of quantitative data will allow the data to be clearer and more presentable; as well as give clear metrics on the participant's opinion on how challenging, on a linear scale, they thought the AI they played against was.

Raw numerical data will be useful for this study; however, the use of qualitative data will also be valuable, as it provides reasoning for their numerical ratings of each AI opponent they will face. This will provide a clearer image of what happened in each participant's battle against an AI opponent which will in turn allow a more informed analysis of the results.

5.3 Data Collection

For the data collection of this experiment, participants will be asked to play five levels, each level is the same map, it just features a different type of AI. The differences between the bots in each level will be abstracted from the participant and will just be labelled level 1-5. This is done to prevent bias, as knowing a bot is cheating may affect the players determination to beat it, therefore skewing the results. Once a player has completed a level, they will then be asked to fill in a questionnaire form corresponding to the level they have just played. Each questionnaire is identical, apart from the level 1 questionnaire which asks participants what skill level they have with RTS games. The questionnaires are separated like this to keep each form more condensed and easier to read. The questionnaire can be seen in Appendix 1. For the questions asking participants to rate the difficulty, adaptability or their skill, a 10-point version of the Likert Scale (Likert, 1932) is used. The Likert scale is a set of statements offered for a real or hypothetical situation that participants are asked to show their level of agreement with on a metric scale (Joshi et al, 2015). This version has ten options with 1 always being the lowest and 10 always being the highest rating. This scale was chosen over a traditional 5-point Likert scale as adjacent options are less radically different from each other. This larger spectrum of choices offers more independence to a participant to pick the option that they feel is correct rather, than an option that partially represents their current opinion (Joshi et al, 2015).

Data will also be collected in the form of metrics gathered throughout the course of the match. The data that will be tracked is how many ships of each class both the player, and AI opponent lost as well as the match duration. This data will be saved at the end of each match in a .csv file. One file will be created per level and the name of the file corresponds to the level played. These files will be collected and stored with the participant data for analysis.

5.4 Data Analysis

The data gathered from the questionnaires will be analysed in two ways. For the quantitative data, since it will all be numerical or Boolean in nature, it will be plotted into tables and graphs for the purpose of readability. For the questions that ask for a participant to rate something, averages will be calculated, and these average values will be utilised to draw overall conclusions on how challenging a bot was. The higher the average result, the more difficult that bot was perceived to be.

For the qualitative data, this will be analysed using a technique called thematic analysis, this is a research method used to identify and interpret patterns or themes in a data set by selecting out keywords within a participant's response and assigning them into codes. Using these codes, themes can be built. Participant's responses are then placed in a theme and the proportion of responses that fit into each theme can be analysed in a quantitative manner (Naeem et al, 2023). This technique will be done to create themes based on the strategy each player used and see the proportion of people who used each strategy on each opponent. This will help determine if a certain AI was weak/stronger against a particular strategy a participant may have used.

For the metrics gathered in each map on ship losses and match time, this will be used to gauge an idea of how challenging it was for a person in a particular match, and to see if this data matches the responses in the questionnaire. To gauge how challenging a particular match was the ship losses and match duration will be reviewed. For example, a match which went on for a long period of time and had both the AI and participant both lose a substantial number of ships, regardless of the victor, will show it was a challenging experience. On the other hand, a short match time with minimal losses on the player side with a player victory will mean the AI was too easy. Alternatively, if the same results are seen but the AI won, it was either too difficult for the participant and they got easily defeated, or the participant did not try to win.

5.5 Limitations

There are a few potential limitations to consider with this study. The main one being participants' experience level with the RTS genre of games. Although most participants will likely be games design and technology students, it does not mean that everyone will have experience playing RTS games before. Therefore, those with little experience are likely to find even the base AI difficult to beat. This could skew the results of the study, as even the basic AI could have a high loss rate.

Another limitation is the time taken for one participant to complete the study. RTS games are not known for their short game time, so participants may feel fatigue after playing a few of the levels. As a result, a surrender button will be unlocked after five minutes of game time has passed, so if the participant is not confident, they will be able to beat that AI they can forfeit the game. This feature, however, may encourage a higher loss rate with participants opting to give up even though the tide of the game may be about to turn in their favour.

One final limitation to consider is the number of methods being used in this study. Only 3 methods utilised by the industry and academia are being used, this being cheating, the Lanchester Model and Dynamic Difficulty scaling. There are many more methods that could be used in this study to gather a broader array of data to analyse the level of challenge each one brings to the player; however, due to time constraints and the previously discussed limitation of how long one run of the experiment may take, it is a limitation that cannot be changed.

6.0 Results and Findings

The study was conducted, in which 10 participants were asked to play through all five of the different level each containing a functionally different AI. They were asked to play through them in level order, complete a questionnaire after completing each match. The nature of the AI they were playing against was abstracted from them.

6.1 Participant skill level

The first set of results details the participants skill level. This was the first question asked after a participant completed the first match. Asking participants to rate their skill level with RTS games was to compare the win rates of both skilled and unskilled participants. As seen in figure 5, there was a fair distribution of skill levels amongst the participants, with 3 participants who were well versed in RTS games rating themselves a 7 or above. 3 moderately skilled participants rating themselves a 4-6 and the largest proportion of participants were low skilled rating themselves 3 or below. This distribution allowed for a good opportunity to see how participants of different skill levels faired against each bot.

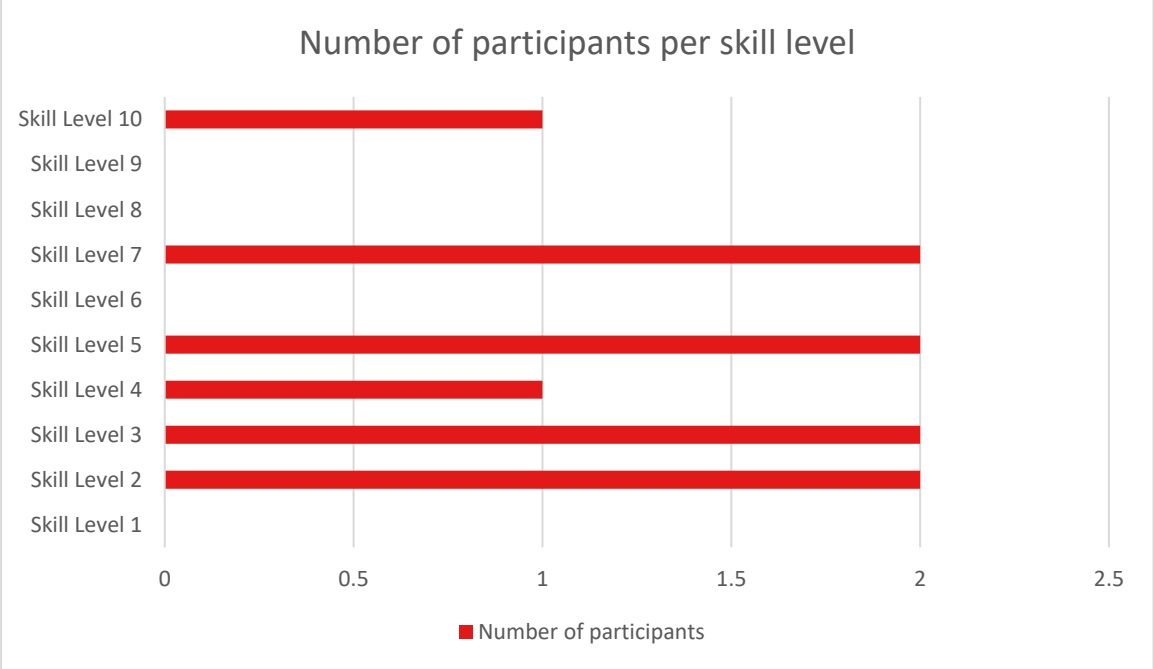


Figure 5: Chart showing the skill level of participants who took part in this study

6.2 Match statistics and questionnaire responses

Figure 6 shows the results gathered from a combination of questionnaire responses and save data collected at the end of each match. The save data included how many of each ship type was lost by both player and AI as well as the match duration. The ship class data is abstracted; however, the full stats breakdown is viewable in the appendix of this paper. Each data point is also a combined average of the results gathered by each of the 10 participants.

The first set of results in figure 6 is from the baseline AI. This AI provided a baseline for players to see how they handled the AI that was created without any modifications. It featured the longest average match duration of 791.3 seconds or 13.18 minutes. It also featured the result for both the highest average player and AI casualties.

The second set of results is from the AI with a constant 1.5x income multiplier. This had the longest match duration and highest player casualties outside the baseline. Despite having the lowest win rate

at 50% however, it was also rated the joint lowest in the difficulty category with an average score of 5.2.

The third set of results is from the AI that used the Lanchester combat prediction model as mentioned in section 4.5.2 by (Stanescu et al, 2017). This algorithm will decide whether to attack/retreat based on unit strength as opposed to who has the most units by raw numbers. Interestingly this features the highest AI casualty rate outside the baseline but also the lowest player casualty rate. As well as this it shares the joint lowest difficulty and the lowest match time. These are unexpected results considering the combat prediction algorithm proposed by (Stanescu et al, 2017) yielded better results than default StarCraft AI. However, that paper did not test this model against human participants.

The fourth set of results are from the AI using dynamic difficulty scaling (DDS). This AI was given the highest adaptability rating of all 5 with a score of 6.4/10. The win rate is also the same as the Lanchester model, however it features a higher difficulty rating.

The last set of results are from the AI that used both DDS and the Lanchester model. The results in this data set are most interesting as this AI not only has the highest win rate at 80%, but also the highest difficulty rating and the second highest adaptability score of 5.8 and 5.9 out of 10 respectively. Furthermore, this AI also featured the second highest match duration, player and AI casualty rates outside the baseline.

Average stats per match	Base AI	AI 1.5x Income Multiplier	Lanchester Model AI	DDS AI	DDS + Lanchester AI
Total Player Casualties	18.6	15.9	10.4	11.3	13
Total AI Casualties	29.9	21.6	23.5	22	22.2
Match Duration (seconds)	791.3	670.5	626	639.2	666.7
Difficulty Rating	5.3	5.2	5.2	5.5	5.8
Adaptability Rating	4.9	5.9	5.7	6.4	5.9
Win rate	60%	50%	70%	70%	80%

Figure 6: Table of results of average match statistics against each AI bot.

6.3 Thematic analysis of participant strategies

To quantify the data from the qualitative data obtained in this study, thematic analysis was undertaken to determine the proportion of participants who utilised specific strategies within each match. There are three strategies that are commonly used by RTS players, these economy building, turtling and rushing (Bycer, 2025).These were 3 of the 5 chosen themes used in this analysis. The other two were added based on codes, as during the coding stage of the analysis, a fair proportion of participants opted to use mid and late game pushes when they had stronger ships. The results table in figure 7 is a breakdown of how many participants used a specific strategy based on the codes in their responses. The total of each column is more than 10 however, as participants seem to use multiple strategies throughout the course of a game.

In figure 7, the first column shows the strategies used against the baseline AI. Here most participants built up their economy before a late game push once they had a greater resource output than the AI. It is also the bot that had the most uses of a late game push against it, which correlates with the long average match duration seen in figure 6.

The second column shows the strategies used against the cheating AI. Economy building is the main strategy utilised here since it is the only way to overcome the AI's 1.5x income multiplier. However, the number of late game pushes has dropped which correlates to the lower win rate this bot has against participants. So, it is likely more would have used this option if they did not lose to the AI In the early game. Turtling is also more prevalent with 2 participants using this strategy as opposed to 1 in the baseline, again likely due to them trying to defend against an early AI rush. Other than those changes the statistics stay the same as the baseline.

The third column represents the strategies used against the AI using the Lanchester combat prediction model. This sees the greatest number of participants using the economy building strategy with 7/10 utilising it. It also sees the lowest number of participants using turtling as no one use this strategy against this AI. Instead, a pattern of aggressiveness is seen as participants opted to rush or push in the mid to late game.

The fourth column shows strategies used against the AI using DDS. This shows the most even distribution of strategies out of all the matches, with 4 participants opting to economy build and push in the late game. But it also sees the greatest use of turtling with 3 participants using it. It also sees the joint highest usage of mid-game pushes with 3 participants using this strategy. This mix of strategies is likely due to the DDS giving a better ebb and flow of power between the participant and the AI. A point reinforced by one participant who stated in the feedback for the AI question: "They came at me then I fended them off then they came at me again and I fended them off till I could overpower them." Showing this bot forced players to use a variety of different strategies throughout the course of a match.

The last column shows the strategies used against the final AI that participants faced which was the AI using both DDS and the Lanchester model. Although there is a fair distribution of strategies, like the pure DDS AI. Participants leaned less towards turtling and more towards economy building and mid to late games pushes, which mirrors the results from the pure Lanchester model AI.

Strategy	Codes Included	Strategy Definition	Base AI	AI 1.5x Income Mulitplier	Lanchester Model AI	DDS AI	DDS + Lanchester AI
Economy Building	Capturing mines, saving money tech rushing	Taking gathered resources and investing them into obtaining more resources and a greater tech level at a quicker rate than your opponent. Then build a large army to overwhelm the opponent in the mid-late game.	6	6	7	4	5
Turtling	Defense	A slower version of economy building where you gradually develop units while investing resources. This helps form a gradual and later inpenetrable defense against the enemy forces rushing you.	1	2	0	3	2
Rush	Corvette spam	This strategy involves building a lot of early game ships at the start and pushing the enemy base immediately, with the aim to ensure a quick victory.	1	1	2	2	2
Mid game push	Mid level ship spam, swarm attacks	This strategy involves a combination of rushing and economy building. Instead of rushing immediately, a player will build their economy and tech to the mid-game point. After which they will build a fleet of mid-game ships and hope to beat the enemy in one attack.	2	2	2	3	3
Late game push	battleship spam	A specialised version of the economy building. Rather than build a mixed unit fleet, the endgame invovles building the most powerful unit and spamming them over and over till the enemy loses.	6	3	5	4	4

Figure 7: Thematic analysis of the strategies used by participants

7.0 Discussion and Analysis

7.1 Analysis of the AI developed

The results of this investigation provide an interesting insight into which methods used within the five AI systems created a challenging match for participants, and which ones did not. The first method used was allowing the AI to cheat by giving them a 1.5x income multiplier throughout the entire game. As previously discussed, the 50%-win rate shows that participants found this to be a difficult level with one stating in the question asking for feedback around the AI: "I would increase then development time for enemy AI ships to take longer, to give the player a better chance to strategise." Furthermore, the cheating of the AI appeared to be obvious to participants with another stating "I think it could be more balanced as it felt as though they had a lot more ships than me." As stated by (Davis, 1999) if the cheating is obvious, the game is not fun. Therefore, showing that allowing this AI to have such a severe economic advantage over the player did not create a challenging opponent for them, moreover a combatant they found difficult and not fun to play against which does not align with the aim of this study. As a result, showing the use of cheating is not necessarily a viable strategy to create challenging AI opponents in RTS games.

Following on, the next method that was utilised was the Lanchester model of combat prediction within RTS games proposed by (Stanescu et al, 2017). This AI saw both the lowest player casualty rate, highest AI casualty rate and lowest average match stats. On top of this it was also rated the lowest difficulty and the lowest adaptability outside of the baseline AI. Therefore, having the worst statistics out of all the other AI that was tested, conveys it did not do an effective job at creating a challenging match for the participants. This is in direct contrast to the study conducted by (Stanescu et al, 2017), which showed a modified StarCraft AI bot utilising this algorithm had a 60.8% average win rate against tournament winning StarCraft AI. One factor for this might be because this paper never tested against human opponents, however the more likely factor is that the paper uses logistic regression to learn unit strength values through simulated matches. This increased the win rate for the AI by 8.9% over the AI that did not use logistic regression. As a result, implementing this may have made the AI more challenging for participants, as the results show that it seemed the easiest and quickest to beat providing little challenge. This is supported by feedback from participants, for example one participant stated: "It felt like they were easier than level 2, but I did notice that they seemed to 'poke' me more, in that they sent out a feeler ship (a cheap one) and then quickly retreated."

The next AI to analyse was the one that used dynamic difficulty scaling (DDS). Based on pure data and feedback given by participants, it appeared to be the AI that performed the second best in providing a challenging experience for players, especially as one participant stated, "that was the most engaging the game had been." The longer average match time and the higher average casualties on both the participant and AI side suggests matches were more intense and no side was favoured. Furthermore, looking at the strategies used in figure 7, a combination of all types of strategies were used in high proportion, indicating participants felt the need to change plans based on the behaviour of the AI. This also seems to support the study by (Hagelbäck, Johansson, 2009) as the AI utilises the same adaptive difficulty equation with the values from the bot which performed the best in their respective study. This combination of results seeming to indicate that DDS is an effective strategy for creating challenging AI opponents, as although the win rate for the participant is 70%, the match data indicates that no match was easy, and participants were challenged throughout the course of the entire match. Further supported by feedback from one participant who stated: "They came at me then I fended them off then they came at me again and I fended them off till I could overpower them." This type of match fitting with Davis's (Davis, 1999) idea of a challenging opponent being one that threatens the player but, in the end, making them feel like they heroically overcame a superior enemy.

The final AI was a combination of both the Lanchester model as well as utilising DDS. This combination was used to see if combining methods would create an even more challenging AI for the player than using these methods alone. Judging purely on the data, this approach seemed to have worked as it features some favourable results. The highest win rate and second longest match times outside the baseline indicate that matches were long but allowed the player to win in the end. This is further supported by the fact that it received the highest difficulty rating, and second highest adaptability. This should not correlate with the high win rate unless the match was challenging for the participant, and they felt like they overcame a superior enemy. Furthermore, it featured a greater usage of strategies with half the players utilising economy building, likely to overcome the AI creating large forces after losing a push. This result is further supported by the feedback given with one participant stating: "I feel like this area was well balanced and great for a final level I have no other suggestions" and another stating: "The AI provided great challenge against my strategy and pushed into different playstyles like sending different units to the credit mines." These results are surprising considering that the Lanchester model alone provided little challenge for players. Therefore, combining it with DDS, which was the best performing until this match, managed to improve the level of challenge overall. It is even more surprising as, players tend to get better at a game, the more they play it. So, leaving this level to last and for it to still be the most challenging for players supports the fact that both these methods when used in conjunction with each other, create a challenging opponent for the player.

7.2 Limitations of the investigation

Despite the results seeming to be favourable towards the AI that was deemed challenging by other academics, there are some limitations to be discussed regarding the investigation. Including issues with the methodology, design features of the RTS framework and the conditions in which the results were obtained.

Regarding the adequacy of the methodology, for the most part the testing processes and data gathering methods appeared to be conducted well. The two main issues however are the sample size of participants and the bias around the result analysis. With the sample size, since only 10 participants were asked to take part in the study, it is not representative of the entire population. Therefore, there is a small chance that the results would be statistically significant. Furthermore only 30% of participants were somewhat skilled in playing RTS games, as a result although lower skilled players found it more of a challenge to play against the various AI. It may then mean that applying these methods to games played by hardcore RTS players would not work, since it would not be a challenge for them. On top of this asking participants to rate their own skill level in RTS games will skew the results with bias, as some players may rate themselves a higher or lower skill level than they actually are, meaning the results are not representative of their true skill level. For example, the participant who rated themselves a 10 on the question regarding their skill level, only ended up winning two out of the five matches, which is the same win rate as someone who only gave themselves a rating of 2.

In terms of bias over the result analysis, this refers to the thematic analysis conducted to quantify the data from the question asking what strategy participants used. As the coding and theming part of thematic analysis is heavily reliant on the reader's interpretation of a participant's response, it means if another party conducted the same analysis of the results, they could end up with different results entirely. Resulting in this data having some bias.

Another limitation of this investigation is regarding the design of the RTS framework used. If this test was conducted using an existing and polished piece of RTS software, such as StarCraft or the ORTS, there would be a minimal chance of bugs within the game affecting overall results. However, since the decision was made to construct a completely new RTS for the purpose of testing, it meant that this was more susceptible to bugs, which could potentially affect the data collected. Unfortunately, this did happen within the investigation as 3 participants noted in the feedback question, that, the AI moved to capture their credit mines but could not destroy them. This resulted in times where a few ships clustered the mines before getting ordered to attack the participant's shipyard. It is very likely had this bug not been present the results may have looked drastically different, potentially a lower win rate and greater difficulty ratings across all AI's.

The last limitation is the conditions in which the results were obtained. As the participants were university students, the majority of which were on the same courses, so since the study took place over the span of 3 days, participants could have conferred with each other about the study. For example, participants who had already taken part could have passed on strategies to defeat each AI to participants who were yet to take part in the study. Resulting in potentially skewed results from some participants as they had insight into how each AI operated.

8.0 Conclusion

This paper set to investigate what techniques and development processes could be used to develop more challenging AI opponents for RTS games. During the course of this investigation, extensive research was conducted in both the how to structure an AI, but also what algorithms and techniques could be used during the decision-making processes to enhance the difficulty of an AI without resorting to allowing the AI to cheat.

RQ1 was to discover what deemed an AI as challenging. The result turned out to be one that threatened the player throughout the course of a match and may beat them a couple of times, however in the end the player will manage to overcome the opponent and feal a great sense of accomplishment doing so. It should not be an AI that is impossible to beat as this will make players give up. RQ2 was to determine the optimal design philosophies for developing an RTS AI. The answer to this is to break down the different tasks an AI must perform into different managers, so each manager focuses on one task, but also allowing these managers to communicate with each other. This makes it a more manageable task to code an AI, as other functionality can be temporarily abstracted while developing a single manager, then at the end they can all be combined at the end to create a fully functional AI. RQ3 asked what the optimal algorithms used for combat prediction are. The research all pointed towards using a variant of the original Lanchester Laws of warfare, with the one used in this study being a specific adaptation of the law of modern warfare to be used in the context of an RTS game. Further steps could have been taken to incorporate logistic regression to learn strength values of units rather than creating them through data on the unit health and cost, however this would have taken the project out of the projected timespan, so the decision was made to cut this aspect. Despite the fact the projected results might have been better.

The aim of this study was to determine the techniques that can be used to develop more challenging AI opponents. The results of the study concluded that the AI used in level 4 and 5 were perceived by participants in qualitative results as the most challenging opponents, which is a fact the quantitative data reinforces. The AI in level 4 using dynamic difficulty scaling, whilst the AI in level 5 used a combination of dynamic difficulty scaling and the Lanchester model for combat prediction. Both of these also gained better verbal feedback in the questionnaire on how challenging it was compared to the AI that outright cheated due to an economic advantage. A fact that was also supported by the quantitative data collected. These results supporting the fact that, there are methods that work better than cheating to create a more challenging AI opponent for players to face.

Although the point this paper set out to prove was achieved, the fact that there was a bug in the RTS framework's code that prevented the AI from taking the participant's credit mines. Although this did not occur all the time, the fact that it was present means the results of this study are different from one where the bug did not exist. This is because participants would have less income when their mines get destroyed and could have made them more likely to lose because of this. Furthermore, the limited sample size of this study brings the question of if 100 participants were asked instead of 10, would the current trend in the results be the same, as it is hard to determine if the chosen participants were in fact representative of the entire population. Despite these shortcomings however, the study can still be deemed a success as these issues with the framework and sample size do not necessarily invalidate the results. Especially as even in fully polished and released games, players will find ways to exploit the AI and discover bugs the developers were not aware of, so it could be argued that the test is even more realistic since it allowed participants to attempt to exploit the AI, as they could in a pre-existing title.

9.0 Recommendations

Future work in this study could include creating AI opponents using the other methods discussed in the literature review. For instance, utilising Monte-Carlo tree search as opposed to basic tree searching algorithms, or the use of fuzzy case-based reasoning to help and AI learn, and problem solve based on previous experience during the current match. Another direction could be to utilise data-driven design, like in the game Stellaris. This allows designers to weight specific choices based on events within the game. Alternatively, the weight of choices could be tied to a dynamic difficulty scaled system that made the AI more likely to choose aggressive strategies when pressed into a corner by the player.

The approach that was taken with dynamic difficulty scaling in this study was to give the AI a boost or reduction in the income multiplier depending on the difficulty score which was calculated every 10 seconds. This equation could be adjusted to make the AI receive more severe bonuses and handicaps depending on the difficulty score to see if the results would be better than the ones gathered in this study.

If this study were to be conducted again however, some changes would need to be made for a more conclusive set of results to be obtained. The main change would be to spend more time on the development of the framework to ensure all core mechanics are bug free, therefore no bugs could potentially interfere with the results like they did in this study. Another change to be done, if this study was conducted again, would be to ensure all participants take part at the same time but in different spaces, therefore reducing the risk of conferring.

Furthermore, changes could be made in relation to the order that participants played the different levels. The current ordering was the basic AI, cheating, Lanchester, DDS then the DDS and Lanchester combination. Since all the levels and mechanics were the same and it was just the AI that changed, despite the difference in difficulty, participants would naturally improve their skill in the game as they progressed. So, performing the test again but getting participants to play the AI in reverse order to what was previously stated, could yield different results as they faced the proposed most challenging AI first and the basic AI last. Potentially flipping the results and getting more losses on the DDS and Lanchester model combination. In theory however, since the difficulty adjusts depending on the player's relative power to the AI, the AI should adjust for the new player and the results could be very similar.

10.0 Appendices

Appendix 10.1 Questionnaire form given to each participant

Dissertation Project Playtest Form Level 1 Responses

This is one of 5 Identical forms each relating to a specific level and the AI opponent within it. Please fill out all questions and give as detailed answers as you wish.

1. On a scale of 1 to 10 (1 being never played an RTS game and 10 being very skilled) How would you rate your skills in realtime strategy games? *

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

2. Did you beat the AI opponent in Level 1? *

- ◯ Yes
- ◯ No

3. If you won, what strategy did you use to win?

Enter your answer

4. On a scale of 1 to 10 (1 being very easy and 10 being impossible) how would you rate the difficulty of the AI opponent in Level 1? *

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

5. On a scale of 1 to 10 (1 being it did not adapt at all and 10 being it adapted to everything you did) how would you rate the ability of the AI opponent Level in 1 to adapt to your strategy during the match? *

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

6. What feedback would you give to improve the AI opponent in Level 1? *

Enter your answer

Appendix 10.2 Raw data gathered from participants who fought the baseline AI

Level 1 - Base AI	Partcipant 1	Partcipant 2	Partcipant 3	Partcipant 4	Partcipant 5	Partcipant 6	Partcipant 7	Partcipant 8	Partcipant 9	Partcipant 10	Average
Player Corvettes Lost:	5	29	3	5	4	3	9	12	32	3	10.5
Player Frigates Lost:	4	0	2	2	4	0	3	11	0	1	2.7
Player Destroyers Lost:	1	0	1	6	2	2	1	4	5	1	2.2
Player Cruisers Lost:	0	0	2	1	2	1	6	0	3	2	1.5
Player Battleships Lost:	0	0	0	0	0	0	3	0	10	1	1.3
Total Player Ships Lost:	10	29	8	14	12	6	22	27	50	8	18.6
AI Corvettes Lost:	4	10	12	11	34	23	25	37	42	11	20.9
AI Frigates Lost:	1	0	4	1	6	11	6	6	18	10	6.3
AI Destroyers Lost:	0	0	0	1	0	2	3	0	8	3	1.7
AI Cruisers Lost:	0	0	0	0	0	1	2	0	5	0	0.8
AI Battleships Lost:	0	0	0	0	0	0	0	0	2	0	0.2
Total AI Ships Lost:	5	10	16	13	40	37	36	43	75	24	29.9
Match Duration	405	735	505	634	833	722	992	864	1570	653	791.3
Difficulty Rating	9	3	6	6	3	7	7	2	8	2	5.3
Adaptability Rating	8	5	3	7	7	3	7	1	5	3	4.9
Won/Lost the match	Loss	Loss	Loss	Loss	Won	Won	Won	Won	Won	Won	W/L Ratio 6/4
Participant Skill Level	5	2	10	2	7	5	3	7	4	3	

Appendix 10.3 Raw data gathered from participants who fought the cheating AI

Level 2 - Cheating AI	Partcipant 1	Partcipant 2	Partcipant 3	Partcipant 4	Partcipant 5	Partcipant 6	Partcipant 7	Partcipant 8	Partcipant 9	Partcipant 10	Average
Player Corvettes Lost:	8	4	6	9	4	26	11	15	8	7	9.8
Player Frigates Lost:	0	6	1	5	3	0	0	0	3	2	2
Player Destroyers Lost:	4	2	5	5	6	0	0	0	3	2	2.5
Player Cruisers Lost:	0	0	1	1	1	0	1	1	3	0	0.8
Player Battleships Lost:	0	0	6	0	0	0	0	0	0	0	0.6
Total Player Ships Lost:	12	12	19	20	14	26	12	16	17	11	15.9
AI Corvettes Lost:	3	3	15	6	6	44	14	45	23	4	16.3
AI Frigates Lost:	0	1	8	0	6	0	2	0	9	0	2.6
AI Destroyers Lost:	0	0	5	4	5	0	2	0	5	0	2.1
AI Cruisers Lost:	0	0	1	0	1	0	0	0	2	0	0.4
AI Battleships Lost:	0	0	0	0	0	0	1	0	1	0	0.2
Total AI Ships Lost:	3	4	29	10	18	44	19	45	40	4	21.6
Match Duration	356	457	864	830	875	697	647	884	776	319	670.5
Difficulty Rating	8	6	2	6	4	7	4	2	4	9	5.2
Adaptability Rating	9	9	7	6	4	3	5	4	4	8	5.9
Won/Lost the match	Loss	Loss	Loss	Loss	Won	Won	Won	Won	Won	Loss	W/L Ratio 5/5
Participant Skill Level	5	2	10	2	7	5	3	7	4	3	

Appendix 10.4 Raw data gathered from participants who fought the Lanchester model AI

Level 3 - Lanchester Model	Partcipant 1	Partcipant 2	Partcipant 3	Partcipant 4	Partcipant 5	Partcipant 6	Partcipant 7	Partcipant 8	Partcipant 9	Partcipant 10	Average
Player Corvettes Lost:	21	2	3	2	4	21	5	8	0	3	6.9
Player Frigates Lost:	1	2	2	2	2	0	0	0	2	1	1.2
Player Destroyers Lost:	1	9	2	6	2	0	0	0	0	0	2
Player Cruisers Lost:	0	0	0	0	1	0	0	0	0	1	0.1
Player Battleships Lost:	0	0	0	0	0	0	0	1	0	0	0.1
Total Player Ships Lost:	23	13	7	10	9	21	5	9	2	5	10.4
AI Corvettes Lost:	9	14	19	11	25	23	20	27	14	16	17.8
AI Frigates Lost:	3	1	2	1	10	2	12	3	5	9	4.8
AI Destroyers Lost:	0	0	0	0	2	0	0	0	4	2	0.8
AI Cruisers Lost:	0	0	0	0	1	0	0	0	0	0	0.1
AI Battleships Lost:	0	0	0	0	0	0	0	0	0	0	0
Total AI Ships Lost:	12	15	21	12	38	25	32	30	23	27	23.5
Match Duration	535	577	661	594	772	732	678	638	528	545	626
Difficulty Rating	7	9	6	7	3	6	4	3	1	6	5.2
Adaptability Rating	9	10	4	7	5	6	5	2	3	6	5.7
Won/Lost the match	Loss	Loss	Won	Loss	Won	Won	Won	Won	Won	Won	W/L Ratio 7/3
Participant Skill Level	5	2	10	2	7	5	3	7	4	3	

Appendix 10.5 Raw data gathered from participants who fought the DDS AI

Level 4 - DDS	Partcipant 1	Partcipant 2	Partcipant 3	Partcipant 4	Partcipant 5	Partcipant 6	Partcipant 7	Partcipant 8	Partcipant 9	Partcipant 10	Average
Player Corvettes Lost:	8	1	15	3	0	5	7	6	4	4	5.3
Player Frigates Lost:	9	0	0	0	0	1	0	2	7	1	2
Player Destroyers Lost:	0	5	0	12	1	0	0	0	1	4	1.9
Player Cruisers Lost:	0	0	0	0	1	0	0	0	1	1	0.2
Player Battleships Lost:	0	0	0	0	0	0	0	1	0	0	0.1
Total Player Ships Lost:	17	6	15	15	2	6	7	9	26	10	11.3
AI Corvettes Lost:	8	5	14	14	13	17	24	22	26	20	16.3
AI Frigates Lost:	0	1	0	1	12	8	0	0	4	6	3.2
AI Destroyers Lost:	0	0	0	1	5	4	0	0	3	6	1.9
AI Cruisers Lost:	0	0	0	0	0	0	0	0	3	0	0.3
AI Battleships Lost:	0	0	0	0	0	2	0	0	0	0	0.2
Total AI Ships Lost:	8	7	14	16	30	31	24	22	36	32	22
Match Duration	487	468	448	887	664	647	748	664	686	693	639.2
Difficulty Rating	8	9	2	6	3	6	4	4	6	7	5.5
Adaptability Rating	8	10	7	4	5	6	6	4	6	8	6.4
Won/Lost the match	Loss	Loss	Loss	Won	Won	Won	Won	Won	Won	Won	W/L Ratio 7/3
Participant Skill Level	5	2	10	2	7	5	3	7	4	3	

Appendix 10.6 Raw data gathered from participants who fought against the DDS + Lanchester Model AI

Level 5 - Lanchester + DDS	Partcipant 1	Partcipant 2	Partcipant 3	Partcipant 4	Partcipant 5	Partcipant 6	Partcipant 7	Partcipant 8	Partcipant 9	Partcipant 10	Average
Player Corvettes Lost:	6	2	4	7	1	4	12	8	9	4	5.7
Player Frigates Lost:	5	0	0	0	0	3	0	2	17	1	2.8
Player Destroyers Lost:	2	5	1	0	2	0	0	0	0	1	1
Player Cruisers Lost:	1	0	0	16	0	0	0	2	0	3	1.9
Player Battleships Lost:	0	0	0	9	0	0	0	0	1	2	1
Total Player Ships Lost:	14	7	5	32	3	7	12	12	27	11	13
AI Corvettes Lost:	3	3	7	28	17	16	13	19	20	19	14.5
AI Frigates Lost:	1	1	0	14	5	1	6	0	13	7	4.8
AI Destroyers Lost:	0	0	0	8	0	0	0	0	4	4	1.6
AI Cruisers Lost:	0	0	0	3	0	0	0	0	2	1	0.6
AI Battleships Lost:	0	0	0	5	0	0	0	0	1	1	0.7
Total AI Ships Lost:	4	4	7	58	22	17	19	19	40	32	22.2
Match Duration	478	434	472	1247	539	588	624	716	896	673	666.7
Difficulty Rating	9	9	4	7	2	2	6	4	7	8	5.8
Adaptability Rating	10	10	7	5	3	2	7	3	5	7	5.9
Won/Lost the match	Loss	Loss	Won	Won	Won	Won	Won	Won	Won	Won	W/L Ratio 8/2
Participant Skill Level	5	2	10	2	7	5	3	7	4	3	

11.0 References

Ayangbekun, O.J. and Akinde, I.O., 2014. Development of a real-time strategy game. *Asian Journal of Computer and Information Systems*, *2*(4).

Bakkes, S., Spronck, P. and van Den Herik, J., 2008, December. Rapid adaptation of video game AI. In *2008 IEEE Symposium On Computational Intelligence and Games* (pp. 79-86). IEEE.

Bari, M., 2017. Creating Complex AI Behaviour in Stellaris through Data-Driven Design. In *Game Developer Conference*

Buro, M., 2003, Open Real-Time Strategy [Game Engine]. ORTS.

Buro, M. and Furtak, T.M., 2004, May. RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)* (Vol. 6370, pp. 1-8).

Bycer, J., 2025. *Game Design Deep Dive: Real-Time Strategy*. CRC Press.

Cadena, P. and Garrido, L., 2011. Fuzzy case-based reasoning for managing strategic and tactical reasoning in starcraft. In *Advances in Artificial Intelligence: 10th Mexican International Conference on Artificial Intelligence, MICAI 2011, Puebla, Mexico, November 26-December 4, 2011, Proceedings, Part I 10* (pp. 113-124). Springer Berlin Heidelberg.

Chaslot, G., Bakkes, S., Szita, I. and Spronck, P., 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (Vol. 4, No. 1, pp. 216-217).

Churchill, D., 2013. *UAlbertaBot* [Custom Written AI] Churchill, D.

Colledanchise, M. and Ögren, P., 2018. *Behavior trees in robotics and AI: An introduction*. CRC Press.

Davis, I.L., 1999. Strategies for strategy game AI. In *AAAI 1999 Spring Symposium on Artificial Intelligence and Computer Games* (pp. 24-27).

Deriglazov, A., 2018. The Pleasure of Turtling: Having Fun the Wrong Way. In *Proceedings of the 12th International Philosophy of Computer Games Conference, Copenhagen (Denmark)*.

Ensemble Studios., 1997. *Age of Empires* [Video Game]. Xbox Game Studios.

Epic Games., 2022. *Unreal Engine 5* [Game Engine]. Epic Games.

Firaxis Games., 2005. *Civilisation IV* [Video Game]. 2K Games.

Hagelback, J. and Johansson, S.J., 2009, September. Measuring player experience on runtime dynamic difficulty scaling in an RTS game. In *2009 IEEE Symposium on Computational Intelligence and Games* (pp. 46-52). IEEE.

Hoekenga, B.C., 2007. *Mind over machine: what Deep Blue taught us about chess, artificial intelligence, and the human spirit* (Doctoral dissertation, Massachusetts Institute of Technology).

Johnson, S., 2008. Playing to lose: AI and civilization. In *Game Developer Conference*.

Joshi, A., Kale, S., Chandel, S. and Pal, D.K., 2015. Likert scale: Explored and explained. *British journal of applied science & technology*, *7*(4), pp.396-403.

Kabanza, F., Bellefeuille, P., Bisson, F., Benaskeur, A.R. and Irandoust, H., 2010, July. Opponent behaviour recognition for real-time strategy games. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Kamble, A.J. and Rewaskar, R.P., 2020. Soft computing-fuzzy logic: an overview. *International Journal of Fuzzy Mathematical Archive*, *18*(1), pp.45-52.

Kovarex., 2011. *Brood War Application Programming Interface* [AI Creation Framework]. Kovarex.

Lanchester, F.W., 1916. Aircraft in Warfare: The Dawn of the Fourth Arm. *Constable and company limited*.

Li, W., 2008. *Finding optimal rush attacks in real Time strategy (RTS) games* (Master's thesis, Universitetet i Agder/Agder University).

Likert, R., 1932. A technique for the measurement of attitudes. *Archives of Psychology*.

MacKay, N.J., 2006. Lanchester combat models. *arXiv preprint math/0606300*.

Metzen, C. Phinney, J., 1998. *StarCraft* [Video Game]. Blizzard Entertainment

Naeem, M., Ozuem, W., Howell, K. and Ranfagni, S., 2023. A step-by-step process of thematic analysis to develop a conceptual model in qualitative research. *International Journal of Qualitative Methods*, *22*, p.16094069231205789.

Naughty Dog., 1996. *Crash Bandicoot* [Video Game]. Sony Computer Entertainment.

Paradox Development Studio., 2016. *Stellaris* [Video Game]. Paradox Interactive.

Paradox Development Studio., 2017. *Stellaris: Synthetic Dawn* [Video Game DLC]. Paradox Interactive.

Petroglyph Games., 2006. *Star Wars: Empire at War* [Video Game]. LucasArts.

Robertson, G. and Watson, I., 2014. A review of real-time strategy game AI. *Ai Magazine*, *35*(4), pp.75-104.

Roelofs, G.J., 2017. Pitfalls and solutions when using monte carlo tree search for strategy and tactical games. In *Game AI Pro 3* (pp. 343-354). AK Peters/CRC Press.

Scott, B., 2002. Architecting a game ai. *AI Game Programming Wisdom*, *1*.

Stanescu, M., Barriga, N.A. and Buro, M., 2017. Combat outcome prediction for real-time strategy games. In *Game AI Pro 3* (pp. 301-308). AK Peters/CRC Press.

Surucu, O., Gadsden, S.A. and Yawney, J., 2023. Condition monitoring using machine learning: A review of theory, applications, and recent advances. *Expert Systems with Applications*, *221*, p.119738.
Świechowski, M., Godlewski, K., Sawicki, B. and Mańdziuk, J., 2023. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, *56*(3), pp.2497-2562.

Take-Two Interactive., 2016. *Sid Meier's Civilisation VI* [Video Game]. 2K Games.

The Spring Community., 2007. *Spring Engine* [Game Engine]. The Spring Community.

Zohaib, M., 2018. Dynamic difficulty adjustment (DDA) in computer games: A review. *Advances in Human-Computer Interaction*, *2018*(1), p.5681652.