Universiteit Leiden

# Master Computer Science

Optimising setups and gameplay for Conquest, a chess-like wargame, using Monte Carlo Tree Search


| | |
|---|---|
| Name: | Pim J. Bax, BSc |
| Student ID: | 1537040 |
| Date: | [17/04/2025] |
| Specialisation: | Artificial Intelligence |
| 1st supervisor: | Dr. Mike Preuss |
| 2nd supervisor: | Dr. Diego Pérez Liébana |


Master's Thesis in Computer Science


Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

# Contents

## Acknowledgements

I would like to thank Mike Preuss, for guiding me through the process of writing this thesis, as well as giving me a lot of tips on how to approach the research side of it, and I want to thank Diego Pérez-Liébana for being available as my second supervisor. I would also like to give a big thanks to James Goodman, who helped me a lot in answering my many questions when implementing the game in TAG.
I also want to thank Wout Gevaert, Mick Voogt and Annabelle Geluk, as well as both of my supervisors, for reading an earlier draft of this thesis and providing me with very valuable feedback. Naturally, any remaining errors in this thesis are my sole responsibility.

**Abstract**

As a part of the online game 'RuneScape 3', a chess-like wargame mini-game named 'Conquest' exists, which does not have well-established optimal play. This mini-game has some unique properties which make it very interesting to analyse, such as being a less complex wargame-like game. Because of this, analysing this game can help in devising approaches to analyse similar but more complex games. The reduced complexity stems from a smaller parameter space, a discretised playing area and only very minimal stochastic elements being featured in the game. Despite this, its search space is still larger than those of many other game types, meaning this paper explores ways to manage the large action space and produce effective strategies to optimise the different aspects of gameplay.

This paper creates multiple agents to play Conquest, and optimises their behaviour to achieve the highest win rates. It implements several strategies to reduce the size of the search tree, and implements two strategies to effectively traverse this search tree using Monte Carlo Tree Search (MCTS) in order to find the best moves. The two strategies implemented utilise First Play Urgency, and a variant of MCTS-Minimax. These MCTS agents are then tuned by evaluating it against a Random Search agent, and then letting the tuned agent play with different setup strategies. This results in finding which setups offer the best chance to defeat an opponent with an unknown setup of their own.

This paper contributes an exploration of different ways to handle large search spaces. It also provides insights into the strategies for the game Conquest, and demonstrates how this game can be analysed thoroughly, which can be used as a simplified analogue to more complex wargames.

# 1 Introduction

'Conquest' [1][2] is a mini-game, part of the Massively Multiplayer Online Role-Playing Game (MMORPG) 'RuneScape 3' [3]. It is best categorized as a wargame – a type of game that was originally designed as military tactics trainers, having since expanded into the table top game space. It also has several similarities with abstract strategy games such as chess. It was originally released on 25 August 2010, and aside from minor balancing tweaks[1] there have been no significant changes to the gameplay. Because a re-balancing update to the game is unlikely to happen at this point, we will direct our research towards optimising gameplay given the current rules and parameters of the game.

The game has several properties that make it an interesting research subject, especially in the context of optimising gameplay in wargames. The game is almost entirely deterministic, with just two randomized elements: the obstacles placed in the field before the start of the game, and a single command that applies to re-enable a random other command. This means that, other than this singular command, any action performed during the game will always have the same outcome, greatly simplifying analysis. Because stochastic action effects introduce a

[1] One of the game's troops had its health reduced by 100; `https://secure.runescape.com/m=news/charm-sprite-hunting`

greater variance in the rollout stage of a Monte Carlo Tree Search (MCTS) agent, the high determinicity of this game makes it well-suited to perform experiments on. Results from these experiments could then be extrapolated to be effective on other wargames as well. This paper will research ways to optimise strategy and tactics within Conquest, both in an effort to improve the game's strategies, and to show that Conquest can serve as a useful game for analysing games in the genre as a whole.

Because the overall gameplay structure is similar to other wargames, it can serve as a useful analogue to analysing games in the genre. Like Stratego, it is a majorly simplified version of a so-called 'miniature wargame', and features a separate setup phase that takes place before the main gameplay starts. Most modern miniature wargames involve many types of troops, which in turn have many different parameters, all of which influence the complexity of the game. Conquest simplifies these wargame aspects by reducing the number of troop types and the number of parameters each troop has, while still keeping the overall structure of player turns similar to that of most modern wargames. Because of this latter aspect of the game, it is more similar in many ways to modern miniature wargames than to Stratego, even though it is also played on a chequered board.

With the game consisting of a setup phase that is separate of the core gameplay, we will use two different approaches to optimise these phases. We will implement Conquest into a framework that can be used to analyse the game, and then optimise a MCTS agent to be as effective as possible at the core gameplay of the game. We then use this optimised agent in order to find which setups are effective against as many other setups as possible. In doing so, we will optimise both phases of the game separately.

Our work will contribute insights into gameplay optimisation for games with large action spaces. It will also show how the game Conquest is relatively easy to analyse, when compared to other wargames, meaning it can serve as an effective yet simple analogue to other wargames.

## 2 Research Questions

In this paper, we will attempt to optimise gameplay for Conquest, which is a game with a large search space, mostly deterministic gameplay, and a setup phase that takes place entirely before the core gameplay starts. By optimising this game, we hope to establish Conquest as a good analogue to the more complex games in its genre of miniature wargames, which can help optimise existing games, but it may also be useful in game balancing efforts for this type of game.

In order to optimise this game, we will first have to optimise the different parameters and test different techniques to find the combination of those which is best able to win games of Conquest. In order to avoid bias, we opted to use MCTS agents – possibly using action heuristics – instead of using rule-based agents to lead the agent towards useful moves. This MCTS agent needs to be able to handle action spaces consisting of upwards of 400 possible actions, with multiple actions per turn. This leads us to Research Question 1.

RQ 1. **What MCTS configuration performs well at the core gameplay of Conquest, a mostly deterministic game with a large action space?**

Finding an agent that can perform well is necessary to function as a competent baseline agent, in order to accurately assess the different setups. Such an agent will also be able to perform well on similar (mostly deterministic) games that have a large action space. By answering this question first, we then apply the results obtained from these experiments to answer our further research questions.

The main difficulty in finding the optimal strategy with MCTS is that the search space for each phase of the game is very large. For example, given 10 troops, a single turn may consist of 10 possible selection actions, each followed by up to $13 \times 13$ possible movement actions (if selecting a Scout), some of which can be followed by several possible attacking actions (up to 10, if the selected troop has a longer attacking range). Commands further expand this action space, by allowing up to 4 commands to be applied to one of 10 troops each, which may be applied at any point in the turn.

After finding a competent MCTS agent, we need to optimise the troop and command setups. Because there is a limit of 1000 points, 10 troops and 4 commands, there is only a finite number of setups possible. The first step to finding the best setups is calculating how many possible setups there are, and to calculate how many of these use a maximal amount of points. That is, the amount of setups where it is not possible to add another command or troop to the setup, either by lack of points, or by having filled up the available slots for troops and commands. This leads us to Research Question 2.

RQ 2. **How many possible setups exist in Conquest, and how many of those use a maximal amount of setup points?**

Answering this question will allow us to determine if it is possible to analyse all possible setups in this paper, or if it is something that has to be left for potential future research to analyse. For this paper, we ended up limiting ourselves to those listed on the Wiki (in particular, using the formations given in Appendix B), due to the total amount of setups being too large for us to analyse them all. However, we believe answering this question will be useful in facilitating further research into other strategies.

Because there has not been any prior research into the different strategies, we cannot be sure about the interactions between the different strategies listed on the Wiki. Although it is unlikely that there is a single optimal setup and game strategy, it is possible to determine which of the established setups is able to beat most of the other established setups. This leads us to Research Question 3.

RQ 3. **What setup strategies are most successful in general Conquest gameplay, without knowing the opponent's setup?**

By answering this question, we will develop ways to analyse the effectiveness of a large amount of strategies, and the interplay between them. We will apply the MCTS agent we developed for Research Question 1,

By allowing two identical agents to play against each other multiple times, each provided with a different setup, this should in general result in a win for the more successful setup in that given matchup. We will then execute this for every possible matchup, to find the overall best general setup. After this, we will also analyse the moves made by each of the agents, in order to determine the behaviour of the MCTS agents. This leads us to Research Question 4

RQ 4. **Which individual troops and commands are most effective in overall Conquest gameplay?**

Answering this question will help give some insight in effective strategies, as well as indicating which troops and commands are most effective. These results may be applied by players of the game in order to develop new strategies and setups that improve upon the current meta.

# 3 Conquest Gameplay

The game takes place on a 20 by 20 square board. Before starting a game, players need to set up a squad of up to ten troops, as well as selecting up to four commands. Troops function like individual pieces, similar to the pieces used in Chess or Stratego. However, unlike Stratego (and like Chess), the types of troops on the board are known by both players. Commands act like a set of special abilities to buff friendly troops, or debuff enemy troops, which have a cost and a cooldown period after being used. These commands are not made known to the enemy before they are used.

In the selection of troops, players are free to reuse the same troop type as many times as desired, up to the limit of 10 troops in total. The commands must be a selection of four unique commands. Troops and commands each have an associated cost, which comes into play both in setting up, and in the game itself. A total of 1000 points at most may be spent on the combination of troops and commands, and troops can be deployed anywhere on the first three rows of a player's side of the board. Further details about the setup selection that takes place before the core gameplay will be discussed in Section 3.4.

After creating a setup, players can challenge another player to a game, with neither player knowing their opponent's setup. The game takes place in turns, the structure of which will be elaborated on in Section 3.1. At the start of a game, the players decide the time limit per turn, which is set to 1 minute by default, but can be changed to a set time between 30 seconds and 3 minutes per turn. Upon ending a turn, the opponent player receives 25 command points. Notably, this means the first player starts out with 0 command points, but the second player starts out their first turn with 25 command points. These command points are very similar to setup points, with the values for both setup points and command points that are assigned to troops and commands being equal. However, players choosing not to spend setup points do not gain any extra command points at the start of the game.

## 3.1 Phases

The core gameplay takes place in turns, with four phases per turn: the **Selection**, **Movement**, **Combat** and **Rally** Phase. Some phases may be skipped, but once a phase has been ended, no actions from an earlier phase may be performed. At any time, any command may be used, provided the player has sufficient command points to activate a command. Some of these commands may affect the precise functioning of these phases, but the overall functioning of the phases is as follows.

**Selection Phase**

In the Selection Phase, a single friendly troop may be selected. This is the only troop that can be controlled during this turn, but commands can still be applied to other troops if desired.

**Movement Phase**

In the Movement Phase, the selected troop gets to move up to its maximum movement, with distances being calculated based on traversed Chebyshev distance (accounting for obstacles such as other troops). Multiple movement actions may be performed sequentially, but each movement action subtracts the moved distance in that action; taking a suboptimal route will not refund any movement points. Only a single troop can occupy a square at any time, meaning a movement may need to be routed around other troops. Troops may proceed to the next phase without spending any or all of their movement points.

**Combat Phase**

During the Combat Phase, the selected troop may attack another troop, given that the enemy troop is within their attack range (including diagonals). If the enemy troop survives the attack, and the attacking troop is within the enemy troop's own attack range, the enemy troop will get to attack back after the first attack has completed. If the enemy troop does not have the attack range to counter-attack, or if it died, the attacking troop will not suffer any damage. Only a single attack action may be performed, after which the Rally phase starts immediately.

**Rally Phase**

The Rally Phase is essentially a null phase; commands may still be applied, but there are no phase-specific actions that can take place.

## 3.2 Troops

Within the game, there are seven different types of troop. Each of them has a few core attributes, given in Table 1: point cost, movement, attack range, troop health, and finally damage output. Defeating a troop will award the enemy with command points equal to the troop cost, which can then be spent on commands.

| Name | Cost | Movement | Range | Health | Damage |
|---|---|---|---|---|---|
| **S**cout | 25 | 6 | 1 | 100 | 100 |
| **F**oot Soldier | 50 | 4 | 1 | 300 | 100 |
| **H**alberdier | 75 | 4 | 2 | 200 | 200 |
| **A**rcher | 100 | 3 | 6 | 100 | 100 |
| **M**age | 100 | 4 | 4 | 100 | 200 |
| **K**night | 100 | 4 | 1 | 300 | 300 |
| **C**hampion | 200 | 4 | 1 | 500 | 400 |

Table 1: Full list of available troops in Conquest, along with the different properties that apply to them. In this paper we may also refer to these troops by listing the first letter of their name, listed here in bold.

## 3.3 Commands

Aside from the troops, the game also features nine different commands, which have a more varied set of properties. The selection of commands that have been included in a setup may be used during the gameplay, provided the player using the command has sufficient command points available, and provided the command is not currently in its cooldown period (which starts the moment the command gets used). Table 2 shows these most fundamental properties, along with a short description.

| Name | Cost | Cooldown | Targets | Description |
|---|---|---|---|---|
| Battle Cry | 75 | 5 | Friendly | Gain +200 health and damage until next turn. |
| Stoicism | 75 | 5 | Friendly | Gain +400 health until next turn. |
| Regenerate | 150 | 4 | Friendly | Recover all lost health, ignoring health boosts. |
| Bombard | 200 | 2 | Enemy | Deal 100 damage. If the troop dies, no command points are awarded. |
| Winds of Fate | 150 | 2 | Neither | A random command on cooldown will reach the end of its cooldown. |
| Charge | 75 | 5 | Friendly | Movement is doubled for this turn. |
| Chastise | 50 | 3 | Enemy | Troop cannot be selected next turn. Cannot immobilise the last mobile troop. |
| Vigilance | 50 | 3 | Friendly | Counter-attack will happen before original attack, if within range. |
| Shield Wall | 50 | 5 | Friendly | All damage dealt to this troop is reduced to 100 for one turn. |

Table 2: A list of command descriptions that can be used in Conquest. Commands have an associated point cost, and after using one it cannot be used until after the cooldown period has elapsed.

Generally, applying commands can have a major effect on the course of a battle. For example, using Charge allows troops to out-manoeuvre a Scout, or using Stoicism gives any troop at least as much health as a Champion. Because of this, using commands effectively is a major part of the core Conquest gameplay. Making sure commands synergise with the rest of the setup, and using them effectively, are important factors in determining the outcome of a game.

## 3.4 Setting up

Before a game starts, a player will have the opportunity to spend 1000 points on a troop and command loadout they will be using during the game. The selected troops may be placed anywhere in the bottom three rows of the board. Figure 1 shows the default setup for new players, but this can be customised in any way possible, provided that the setup is limited to 10 troops, 4 commands, and a total of 1000 points spent.

While strategy during the game is very important to win, a proper setup also has a major effect on a player's winning chances. As an extreme example, although a player with 6 Archers, Regenerate, Bombard and Shield Wall would have spent all 1000 points, the bad synergy between their troops and commands will render this setup much less effective than for example a setup consisting of 10 Foot Soldiers, Bombard, Charge and Stoicism, despite costing only 850 points. Because both Regenerate and Shield Wall have no effect on troops with 100 health, this demonstrates that choosing an effective setup with synergies between commands and troops is an important aspect of the game.

### 3.5 Established strategy

The mini-game saw most active play during the years shortly after its release, with popularity dropping off after a few years. The official RuneScape Wiki documents a large list of popular strategies [4], but the page's edit history shows no substantive edits after July of 2017. This means that the game's meta strategies have not (publicly) evolved for a long time, but the large number of different strategies demonstrates that there are a large number of different approaches to building a setup. As can also be seen on the same page, many strategies list a specific set of weaknesses. This implies that there may be several cyclic dominances among the setups.

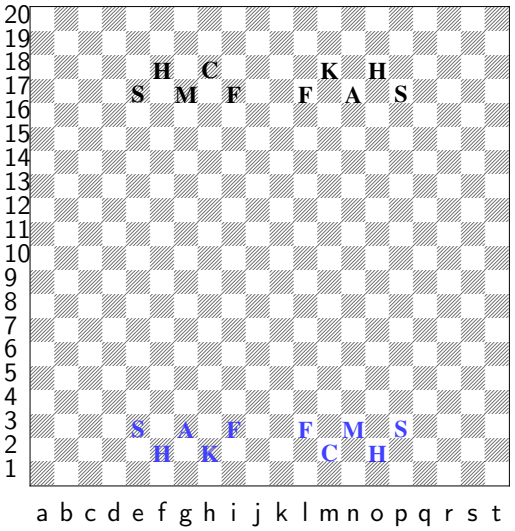


Figure 1: The default setup of troops for both players, with blue pieces representing player 0, and black pieces representing player 1. The default commands are Battle Cry, Charge and Chastise. This represents a balanced setup, which utilises every type of troop, along with some of the most commonly used commands.

Most strategies heavily rely on accurate usage of commands. For example, given enough command points and the right set of commands, a single Scout is able to use Charge to enter the enemy lines, then use Battle Cry to defeat a Knight in one hit, then use Shield Wall to avoid being killed by any enemies (even if the enemy uses Battle Cry of their own), then next turn they can kill an enemy Mage, use Stoicism to survive another turn, and then kill an enemy Archer in their third turn. Of course, this type of attack can be countered by using commands such as Chastise, or by launching a counteroffensive against the enemy using similar commands.

There are many other strategies that use synergies between commands to the player's advantage. Because of this, determining the optimal setup is far from elementary, and even given a particular enemy setup, it may not be easy to determine what is the best counter to it.

# 4 Literature

In this paper we will be taking an in-depth look at the game Conquest [2]. It has not seen any prior research, only having some established strategies laid out on the official Wiki's strategy guide [4]. However, research into games with similar gameplay elements can still prove useful in optimising for this game. In this paper, we will be utilizing Monte Carlo Tree Search [5], in order to simulate gameplay without relying on specific strategies. We will be exploring the use of tuning the First Play Urgency [6], as well as experimenting with other strategies to optimise gameplay, in particular using a MCTS-Minimax hybrid [7].

In order to perform our experiments, we utilised the Tabletop Games Framework (TAG) [8], which has been used to implement many other games for research purposes [9], and which simplifies implementing a MCTS agent that is able to play the game once programming the game itself is done.

## 4.1 Research relevance

Because Conquest exists in the intersection of wargames and abstract strategy games, research into optimising it can be partially extrapolated to both genres of games. In particular, it features only a small number of troops and commands, and has relatively few rules when compared to most other wargames, which makes it easier to analyse than other games in the genre. Compare for example the number of attributes in Table 1 and Table 2 with the number of attributes in [10, Table 1], which demonstrates that typical wargames feature many more parameters. Furthermore, with Conquest being played with a timer set to 1 minute per full turn by default[2, 'Gameplay' section], and the average game taking around 80 turns, the playtime of a full game is generally shorter than most other games in the Wargame category [11, Table 1]. Especially considering the game uses a per-turn timer, and not an overall timer that is similar to a chess clock, it is typical for players not to use the full allotted time. Instead, they typically end their turn early, after performing their chosen actions.

The game features a much smaller parameter space than most of the other games in its category, which drastically reduces the parameter space for balancing purposes. With only 7 troop types and 10 commands, and each of these troops having only 5 characteristic properties, it can be useful for benchmarking game balancing algorithms, or for developing automatic playtesting algorithms for wargames. As well as having a smaller parameter space, it also features grid-based discrete movements, as opposed to continuous playing surfaces featured in many other games. Since continuous search spaces require specialised approaches that are more complex than those that can be applied to discrete search spaces [12], Conquest can be useful as a simplified wargame analogue. This could prove useful in analysing more complex wargames such as Warhammer 40,000 [10], or many of the games analysed in [11].

Another advantage of Conquest is the lack of non-deterministic or stochastic elements. Only the obstacles placed at the start of the game and the application of the Winds of Fate command are randomized, with the rest of the game being fully deterministic. This means the game is much easier to analyse than games where moves are largely stochastic [13].

Furthermore, because the stochastic elements of this game have only a minor impact, this game could serve as a useful case study in comparing deterministic and stochastic behaviour in games. Papers such as [14] may benefit from comparing different types of behaviour for the singular stochastic element during the gameplay, being the Winds of Fate command. On the one hand, a semi-deterministic implementation of this command could be made, by using a pseudorandom number generated from a seed of the current game state. This ensures that, given the current game state, the outcome will always be identical. On the other hand, a more stochastic approach would be one which selects a random choice only at time of application of the command.

Another feature of Conquest, which is also present in several other wargames, is that it features a type of turn-based gameplay that involves a variable amount of moves per turn, with the average moves per turn being around 5. Techniques such as those used in Nested Monte Carlo Tree Search [15] could prove useful to improve the efficacy of tree search strategies. Applying hybrid strategies between MCTS and Minimax could also be a useful technique to improve gameplay [7], for example to ensure a full turn is evaluated before a first move is chosen. Improving the efficient handling of variable turn lengths can also benefit analysis of other genres of games that feature them, such as many deck building games.

Aside from these ways Conquest is relevant for research purposes, it can also be useful to look into developing a Conquest bot, for players to play against in absence of opponent players. Currently, in the game, there is no computer player to play against, so this may make a useful addition to it to make the game more appealing [16].

## 4.2 Prior research

Although the main focus for this paper is finding the most effective strategies within the game, we can still obtain useful insights from prior research into game balancing. Because research into autobalancing is more prevalent, this can help find more relevant research that can be applied to our research questions. It has been shown that AI can be used effectively in automatic game balancing [17]. Using the same approach as autobalancing, we can find the most powerful strategies, and instead of rebalancing them, apply them to be more successful at the game.

Comparing a large amount of different strategies, in order to determine which strategies are more successful than others, is an important aspect of game balancing [18]. Automatically determining the strongest choices is not only important in balancing the game before release

[19], it can also serve a significant role in finding the best strategies available in the game after it will no longer receive balancing updates. Since the troops have different properties that have been balanced through the unit costs, they will likely have asymmetric superiority among the individual troops, or at least between different possible setups, meaning that no simple optimal setup exists [20].

One way to determine the usefulness of a troop is to perform a balancing analysis and determine their point value [10], and then use those values as a heuristic to build the most valuable team possible within the point cap of 1000. Especially when there are multiple characteristics of game elements that need balancing at the same time, balance between these characteristics may be hard to achieve [21].

With the large action space, growing exponentially with more troops and commands being available, we have to ensure this exponentially growing search space for our MCTS agent remains manageable to analyse. Approaches such as [22] may prove useful in inspiring an effective approach to tackle this problem. We will also attempt to implement a way to merge states using Monte Carlo Graph Search (MCGS) [23] or merge identical game states using transposition tables [24], in order to reduce the size of the search tree.

# 5 Calculating the number of possible setups

We started out by determining the amount of setups that are possible, in order to find out the scope of a possible full analysis of all setups. In this paper, we limited our research to only the list of setups on the Wiki's strategy page [4], but by calculating this first, we we know what the scope of potential future research would be.

In order to determine the number of maximal setups possible, we first have to properly define what a maximal setup is. Given the set of troops $T$ and the set of commands $C$, a setup $S$ consisting of at most 10 troops and 4 commands is maximal if one of the following conditions is met:

- The total value of the selected components $val(S) = 1000$
- The troop count $troops(S) = 10$ and the command count is $commands(S) = 4$
- $commands(S) < 4$ and all commands $c \in C$ that are not part of setup $S$ have $val(c) > 1000 - val(S)$
  - Since the cost for a Scout is 25, and Setup Points are always spent in multiples of 25, any setup with fewer than 10 troops is not maximal, so the case $troops(S) < 10$ does not need to be considered.

If one of these conditions is met, no further additions to the setup can be made, making it a maximal setup. Given that adding something to a setup will always improve the setup's effectiveness, or keep it equal[2], it is never beneficial to play with a non-maximal setup.

We can answer Research Question 2 by using a bounded knapsack algorithm, with the bounds being 1000 Setup Points, 10 troops and 4 commands, where the commands can only be used once each. We can then take the list this generates, and filter it to meet at least one of the three conditions listed above. This will give us a full list of maximal setups, without considering the positioning of the individual troops. The code for this experiment is made available as a gist on GitHub[3].

Of course, each of these setups has a large number of possible locations each of the troops can be placed. With each troop having to be placed in the bottom three rows of the board, this gives 60 possible squares to place up to 10 troops. Although this greatly increases the number of unique setups, we will consider all possible troop arrangements to be equivalent. Because the board dimensions are 20 squares, with the bottom 3 rows allowing troop placement, this leaves $20 - 2 \cdot 3 = 14$ squares between the frontmost row of each initial setup. This means that a troop in its starting position can never be attacked by an enemy troop that starts the turn

[2] A non-maximal setup would be 10 Scouts and 3 commands. Adding the Regenerate command will make it maximal, while not improving the effectiveness.

[3] The code for the python script used to generate a full list of all possible setup permutations in Conquest, or to just compute the amount of those setups there are, can be found on `https://gist.github.com/Joeytje50/0dacbcf130bfeb8d683abfb6e7e0b5a8`.

in its starting position. Even a Scout, Archer or Mage that has Charge applied to them only have a total attack range of $2 \cdot 6 + 1 = 13$, $2 \cdot 3 + 6 = 12$ or $2 \cdot 4 + 4 = 12$ squares, respectively. Because of this, the initial placement of the troops is much less significant than the selection of the troop types.

The list of possible setups generated by our bounded knapsack algorithm resulted in a total of 1,714,865 setups, where the order and location of troops, as well as the order of commands, was not taken into account. If this is then filtered to include only maximal setups, this reduces to a total of 283,717 possible combinations. The number of setups that use up the full 1000 points players can spend is further reduced to 171,496.

Although the analysis of creating a setup was not within the scope of this paper, it can be useful in facilitating future research into finding new successful setups. Due to the large number of possible setups, a sophisticated setup creation method, such as Procedural Content Generation (PCG) or Reinforcement Learning could be necessary to generate the best overall setups out of the full list.

# 6 Implementation in TAG

To implement Conquest in a way that enables us to run consistent tests between the different agents, we used the TAG Framework [9]. This framework simplifies tuning agents, and running tournaments, as well as having a set of built-in MCTS agent settings.

We attempted to optimise gameplay on two levels. Firstly, we will attempt to find a configuration of a MCTS agent to play Conquest effectively. We will optimise the gameplay using the Default setup, given to all new players who have not configured their setup. This setup, shown in Figure 1, consists of at least one of every troop, as well as featuring the three most commonly selected commands, based on tallying the troops in the official Wiki's strategy guide [4] (this tally is also given in Table 6 below). Secondly, we will then use this agent to play games using a list of troop setups taken from the game's official Wiki.

Implementing the game in TAG involved implementing the overall board layout, and handling all types of actions available in each phase (Section 3.1), implementing the properties and effects of all types of troops (Section 3.2) and commands (Section 3.3). The implementation of Conquest into TAG has been requested to be merged into the main TAG repository on `https://github.com/GAIGResearch/TabletopGames/pull/323`.

After implementing the core gameplay of Conquest, the size of the search tree very quickly reaches an exponential growth of the search tree, making it infeasible to explore all reasonable moves. Because of that, we had to implement several methods to reduce the search space of this game, or more optimally traverse the tree. We will now discuss the ways we reduced the size of the search space in the subsections below.

## 6.1 Semi-deterministic Winds of Fate

A very minor optimisation we implemented was a change to the sole non-deterministic element of the game, which is the Winds of Fate command. We modified this to no longer use a fully random number, but instead we generated a pseudo-random number, by using a random number generator that got initialised to the current game state hash. This ensures that, given a game state, the result of applying this command will always be the same. While this does not reduce the size of the search tree in any significant way, it reduces the variance of the resulting child sub-tree, which improves the reliability of the results at lower amounts of state evaluations.

## 6.2 Single movement action

The first major optimisation we implemented to reduce the size of the search tree was to enforce a single movement action per turn. In the original Conquest game, players are allowed to move their troops any number of times, as long as the selected troop still has movement

points remaining. However, this massively increased the search space in the movement phase, while having no objective benefit to AI players.

Given a scout's ability to move up to 6 squares, the Movement Phase now consists of at most $13 \cdot 13 = 169$ actions. Without this strategy, $12 \cdot 12 = 144$ of these moves would have a movement point remaining. This means they would be able to move a second time. Furthermore, $11 \cdot 11 = 121$ of those moves would be able to perform yet another movement action. In short: not applying this strategy would increase the search space for the Movement Phase by $12! \approx 4.79 \cdot 10^8$ moves, in the case of a Scout with a fully free range of motion.

Although humans can make up their mind after moving their troops, for example wanting to move them one square left or right after making their move, an MCTS agent will simply use its internal playouts to determine the best place to put the troop, removing the need for multiple consecutive movement actions. Because of that, this optimisation should have no effect whatsoever on the efficacy of the MCTS agents.

### 6.3 Merge identical nodes

Another early approach to reduce the search space we implemented was to merge tree nodes that represent an identical state. Because it is possible to reach an identical state in different ways in some cases, this should reduce the search space. For example, applying Chastise to an enemy troop A and then attacking a different troop B is identical to attacking enemy troop B and then applying Chastise to troop A. Merging these tree nodes reduces the search space by quite a bit, but unfortunately this also increased overhead for each node creation, making this optimisation much less valuable in the end.

The way we implemented this strategy is similar to the MCGS implementation in [23], keeping track of identical nodes and linking to them when they occur elsewhere in the tree, instead of creating another instance of this identical node. The tree is now represented as a graph, due to the changes to the relations between the parent and child node. When traversing this graph, we assigned which parent node we arrived to a given node from. When backtracking, we would then traverse up the tree by following the saved parent node. This way, the correct parent node received backtracking updates, while still avoiding the extra work for evaluating the same game state multiple times separately.

While this strategy would have a significant effect if no other optimisation strategies would be applied, it has only a negligible effect when applied in conjunction with other strategies discussed in the rest of Section 6. Because merging nodes still requires the actions to be evaluated before the nodes can be merged, whereas our other strategies outright prevent creation of certain tree nodes, the other optimisations were also generally more efficient in computation time. Because of this, this strategy was not utilised in the final algorithm.

In future research, it may be useful to omit some of the other optimisations made, in favour of implementing a transposition table or MCGS method. This may ensure that the game that is being experimented with functions identically to the original game. However, because of the additional debugging complexity that came along with our implementation of these transposition tables, we decided to stick with our other optimisations instead. These other optimisations only had negligible effects on the freedom of the agents to make any sequence of moves that would be allowed in the original game, meaning the added complexity of an implementation of MCGS did not outweigh the benefits it provided in our case.

### 6.4 Trim bad moves

Some of the other optimisations we implemented into the game were to apply search tree pruning to trim some objectively bad combinations of moves within a single turn. For example, after applying the Charge command, it makes no sense to move within the default movement range, since this simply wastes command points and puts the command on cooldown. Also, applying Charge is only made to be possible on the currently selected troop, since it serves no purpose otherwise. Furthermore, applying Regenerate to a troop that is at full health, or applying Shield Wall to a troop that has only 100 health remaining, both have no effect, so these options also get optimised away.

This optimisation reduces the search space of a scout that has Charge applied to it by up to 169 moves. Given a playing field of $20 \cdot 20 = 400$, this can almost halve the size of the Movement Phase, in some cases. For the trimming of the Regenerate and Shield Wall commands, this only reduces the search space by 1 per 100-health troop for each Phase where commands can be applied, but even this may have a large effect, due to the exponential nature of the search tree and due to commands being available during multiple phases. While these optimisations do not have a very large effect on the search space when compared to others discussed in the rest of Section 6, their inclusion is still valuable, due to them having no effect at all on optimal gameplay.

### 6.5 Limit command usage window

In order to further reduce the search space, some commands were limited in which phases of the game they could be applied. For example, given the optimisation mentioned above, applying Charge is only sensible after deciding which troop to select, but before moving the troop. Applying this command in any other phase was therefore removed. Similarly, when applying any other command, it makes no difference whether it is applied in the Selection, Movement or Combat Phase, since they either affect the combat during this turn, or they affect the next player's turn. We therefore limited most command usage to the Movement or Rally Phase, reducing the search space. The Rally Phase was included because killing a troop can grant command points, which may then allow the player to activate a command they previously could not afford. Because it is possible for a turn to never have a Combat

Phase, namely if the selected troop does not use up all of its movement points, it was not feasible to limit command usage to this phase instead.

Due to the exponential nature of search trees, limiting commands to only specific moments in a turn also has exponential time savings associated with it. Take the Selection Phase for example, which consists of up to 10 selection actions. With a maximum of 4 commands which can each be applied to one of 10 troops, this would add 40 further actions to this phase. However, after executing any given command, this leaves the remaining 3 commands that can still be used, plus the original selection action. This can theoretically be repeated 4 times before all commands are on cooldown[4], after which one of the up to 10 selections must be made.

Given $c \leq 4$ available commands and a set of living friendly[5] troops $T$, every command can be applied to any of the $|T| \leq 10$ troops, a turn can start with a total of $C_c = c \cdot |T| \leq 40$ possible command actions. Take $A_{SP,c}$ the action space with commands of the Selection Phase, and $A_{SP} = |T|$ the base action space (without allowing commands in this phase). Because commands can typically be applied at any time and in any order, this means $A_{SP,c}$ consists of $|T|$ selection actions, plus $C_c$ command application actions, after which the Selection Phase still takes place with one fewer command available to the player. This means the action space for this phase can be calculated using Equation 1.

$$\begin{aligned} |A_{SP,c=4}| &= A_{SP} + C_4 \cdot |A_{SP,c=3}| \\ &= |T| + C_4 \cdot \Big( |T| + C_3 \cdot \big( |T| + C_2 \cdot (|T| + C_1 \cdot |T|) \big) \Big) \\ &= |T| \cdot \Big( 1 + C_4 \cdot \Big( 1 + C_3 \cdot \big( 1 + C_2 \cdot (1 + C_1) \big) \Big) \Big) \\ &\leq 10 \Big( 1 + 40 \cdot \Big( 1 + 30 \cdot \big( 1 + 20 \cdot (1 + 10) \big) \Big) \Big) \\ &\approx 2.65 \cdot 10^6 \end{aligned} \tag{1}$$

Every single one of these nodes would still have a distinct subtree below it. Although the nodes where some or all of the commands have already been used would have a slightly smaller subtree below it, pruning the tree from 2.6 million nodes in the Selection Phase down to just 10 is a massive improvement in efficiency.

The same reasoning applies to the Movement Phase, by simply taking Equation 1 and changing $A_{SP}$ to the base action space of the Movement Phase $A_{MP}$. Because troops generally have a significantly larger action space in the Movement Phase, the optimisation in this

[4]Note that Winds of Fate could allow a single command to be used twice in one turn, but because its application does not have a recipient, this means that there is only one instance of Winds of Fate, whereas there are up to 10 instances (1 per recipient troop) for all other commands. Because this does not significantly impact the search space, we do not take Winds of Fate into account in this calculation.

[5]For simplicity of reasoning, we will consider only commands that apply to friendly troops; an equivalent reasoning applies to commands that can be applied to enemy troops, meaning our resulting figures are identical to those resulting from a combination of command types.

phase is even more significant than it is for the Selection Phase. The optimisation depends on the movement range of the selected troop, though, so we cannot calculate a precise figure here.

### 6.6 Delay exploring command usage

The above optimisations were still not quite sufficient to optimise the agent that evaluated a full turn before deciding its move (see Section 7.1), we implemented a more drastic optimisation for that agent.

During the first analysis of a full turn to be made, commands were left out as possible moves altogether (with the exception of Charge, which we will cover later). Then, after a full sequence of moves has been found to be optimal for this turn, the moves are played out up to the Combat Phase. Once the Combat Phase is reached, an additional tree search is performed, where applying commands is considered. Because the Combat Phase generally has a very small search space, other than applying commands, this phase is very well suited to expand the search space. In the same vein, in the Rally Phase another such search is performed, in order to determine if commands need to be applied to improve the survivability of the player's troops.

While the above strategy works for any commands that affect either friendly troops' combat, or the enemy's ability to retaliate, it does not apply to the Charge command, because that has to be applied before the Movement phase. Because of this, we applied a different strategy to Charge. For it, we allowed using the command, but for subsequent movement actions, we only considered actions that would place the selected troop within attack range of an enemy troop. Charge is generally only useful when quickly moving into the enemy's territory and dealing damage or killing a troop. Because of that, it does not make much sense exploring the large amount of movement actions that would place the troop in some arbitrary location, not able to attack. This would typically be a waste of the command points to use Charge. Although it is possible some strategies may use Charge to flee from a large group of enemy troops, this is a rare occurrence, meaning that the cost of the time investment to analyse these many moves outweighs the overall benefit obtained from the small amount of times this yields beneficial results.

This command usage delay is effectively a strategy based on dynamic programming. Instead of considering the entire problem all at once – which is a side-effect of the agent needing to evaluate a full turn before deciding its first move – we first consider a simpler problem, and then evaluate the augmentation on only the best-performing action sequence. Because the FPU agent (Section 7.2) already prioritises exploitation of moves even before exploring all possible actions, we did not need to implement this for that agent. Implementing this optimisation for that agent would likely have a much smaller benefit.

The effect of this optimisation can be estimated by considering the reduction in the action space of the Combat and Rally Phases, which can once again be done using Equation 1. The base action space of the Rally Phase is just a single action, being to end the turn. For the Combat Phase, the base action space usually consists of the same action to end the turn, plus one or two attack actions (or sometimes more if troops are bunched up, or the selected troop has a longer attack range). This means that each of these phases gets optimised by a factor of around $10^5$ for this initial phase of the evaluation.

Despite this improvement though, it does add a constant overhead to most turn evaluations. After a sequence of moves has been found, the application of commands needs to be evaluated on both the Combat and Rally Phases. This re-uses the tree that was just used to evaluate the best moves without using commands, which means that the best move sequences are already fairly well established when applying the Upper Confidence Bound formula that is used in MCTS. Because of this, only these most effective move sequences will have the usage of commands explored, with the worse moves not being visited any more at this later stage of the tree search.

If the player has no command points available, no commands off cooldown, or if the optimisations discussed in Section 6.4 have already filtered out the available commands, this does not trigger an extra evaluation of command usage. This means that when there is no benefit to this optimisation, there is also no cost associated with it.

We do not have a precise figure for the overhead cost this generates, but because this evaluates the larger action space only on a much smaller set of tree nodes, we expect this to be orders of magnitude smaller than the savings generated by this optimisation. Although we did not generate precise figures, we did notice a significant improvement in computation time after implementing this optimisation.

## 6.7 Effectiveness of the optimisations

Although most of these optimisations do not necessarily turn the search space into a small one, the exponential nature of decision trees benefits from these significant reductions of the search space anyway. The following overview shows a rough estimate of the effectiveness of each optimisation strategy, as explained further in their respective subsections above. Note that we did not do precise calculations to reach the given optimisation figures, although the reasoning behind these figures can be found in their respective sections.

**Semi-deterministic Winds of Fate**

Optimisation: Remove randomness from the core gameplay.

Effectiveness: Negligible effects on the search tree size, but reduced variance of the resulting sub-trees after applying this command.

**Single movement action**

Optimisation: Omit repeated Movement actions in the same turn.

Effectiveness: Up to $4.79 \cdot 10^8$ fewer possible moves per turn.

**Merge identical nodes**

Optimisation: Use the same tree node to represent multiple instances of the same game state.

Effectiveness: No major effect, when considering overhead computation cost and diminished efficacy when combined with other strategies.

**Trim bad moves**

Optimisation: Avoid making move combinations that are objectively bad.

Effectiveness: Roughly $2\times$ smaller action space.

**Limit command usage window**

Optimisation: Prevent use of commands outside designated phases.

Effectiveness: On the order of a factor $10^6$ smaller action space for each phase where commands are not considered (multiplicative).

**Delay exploring command usage**

Optimisation: Delay the exploration of commands until a later point in the turn.

Effectiveness: Similar efficacy to typical Dynamic Programming efficacy. Optimisations on the order of a factor $10^5$ smaller action space for each phase where commands were not considered (multiplicative), in exchange for what is estimated to be a significantly smaller overhead computation cost.

Out of these, we implemented all mentioned optimisations except for the 'Merge identical nodes' one explained in Section 6.3. The 'Delay exploring command usage' optimisation mentioned in Section 6.6 was only applied to the agent that evaluates a full turn playout before deciding its first move, which is discussed in Section 7.1.

# 7 Optimising the MCTS agents

In order to optimise the MCTS agents, as described by Research Question 1, we implemented two strategies to efficiently handle the large search space that still remained after the optimisations made in Section 6.

We started by applying a relatively basic MCTS algorithm to the game, using a heuristic score for its evaluation function. The only heuristic used in this setup was the one to determine the relative value of the state. However, after some preliminary testing, it turns out this strategy generally does not work at all, resulting in agents that only barely outperform a random agent. Because of this, we attempted two different approaches to resolve this.

The first approach used was implementing a system that guarantees the entirety of a turn is evaluated before the first move of the turn is executed. After a sequence of moves is found which is determined to be the best, no further evaluations will be made in order to reduce the time spent per turn. This approach is supposed to increase the likelihood of a specific move to conform to some overall 'plan'. When observing the plain MCTS agent, it regularly made moves that did not seem to match any reasonable strategy for its previous move. For example, it might select one of the troops that would be able to move up and attack an enemy troop, but then decide to move in a completely different direction from that attackable troop. This also ensured a minimum level of exploitation of good moves, as opposed to very superficial exploration of all possible moves. This approach is further covered in Subsection 7.1.

After this, we also implemented a second approach that primarily made use of action heuristics. This sorted the possible moves at any point by a simple heuristic which of these moves is most likely to be beneficial. Along with this, we applied a First Play Urgency (FPU) to start exploiting the best moves, before exploring all possible moves first. Due to the very large action space at some points in the game (especially during the Movement Phase), it is necessary to focus on what moves are likely to be useful. This approach is further detailed in Subsection 7.2.

For both approaches we implemented a modification to the MCTS algorithm's tree depth criteria. Turns consist of multiple consecutive moves, with only one of the last moves of the turn (being the attacks that take place in the Combat Phase) affecting the heuristic score as evaluated by the algorithm in a positive way. Because of this, it is not as useful to evaluate the outcome of half a turn. More importantly, the use of commands during a turn may affect the heuristic negatively, meaning that ending a rollout halfway through a turn, and evaluating the game state at that point, may backpropagate a much worse evaluation score than it should get. Because of that, we changed the maximum tree depth criteria to only take into account the number of full turns, as opposed to taking an absolute tree depth parameter. Doing this ensures the evaluation at the end of a rollout, as well as the building of the MCTS tree, are explored up to the final state at the end of a turn, instead of halfway through it.

## 7.1 Full-turn playout

Because of the reasoning mentioned before, evaluating only partial turns does not provide as much useful information as evaluating a turn fully. As an approach to make use of this fact, we implemented a full-turn playout approach, which is a type of MCTS-Minimax hybrid [7]. Using this approach, MCTS was applied until the point where repeatedly selecting the best move in a position would lead to a state where it is the opponent's turn, as opposed to leading to a state where an unexpanded node is reached. Algorithm 1 shows the overall structure of this approach.

By evaluating a full turn first, and then applying the sequence of moves that led to the opponent's turn, we know the selected moves were chosen based on a MCTS that reached a minimum depth of a full turn, meaning the best move would be less short-sighted than when this approach is not used.

**Algorithm 1** Pseudocode for the full-turn evaluation. It ensures a turn is evaluated in its entirety, before deciding upon the first move to make. Subsequent moves within that turn do not get allotted further computation time, but instead re-use the evaluations found during this initial full-turn evaluation.

```
1:  function EVALUATEDFULLTURN(node)
2:      currentPlayer ← node.player
3:      while node.player = currentPlayer do
4:          node ← node.bestChildNode
5:          if node is unexplored then
6:              return false
7:          end if
8:      end while
9:      return true
10: end function
11: function EVALUATEFULLTURN(rootNode, baseBudget)
12:     budget ← baseBudget
13:     while budget > 0 do
14:         TREEPOLICY(budget)                          ▷ These three functions may
15:         ROLLOUT(budget)                             ▷ reduce the remaining budget
16:         BACKUP(budget)                              ▷ based on the budget type
17:         if budget ≤ 0 and not EVALUATEDFULLTURN(rootNode) then
18:             budget ← budget + baseBudget            ▷ Increase budget
19:         end if
20:     end while
21: end function
```

This approach turns the exploration budget into a flexible budget. The base budget provided can be exceeded, but because of this, it is also reasonable to set a budget to a much lower value than it is reasonable to evaluate the full turn. For example, if it is typical to evaluate the full turn in 2 seconds, it can be reasonable to set a base budget of 0.5 seconds. The algorithm will then attempt computation bursts of half seconds, after which it will check if a sensible sequence of moves has been found to finish this turn. This also means that it is possible for the agent to find a good sequence of moves after only half a second. This means that, for simpler turns, less time is used, while more complicated situations use up more time.

## 7.2 First Play Urgency

The main issue with using plain MCTS for Conquest is the fact the search space may become very large in some situations, especially in the Movement Phase of Scouts, or any troops with Charge applied to them. One approach to limit the amount of time spent on exploring all of these different options, is to start exploiting promising looking moves, before having explored all possible moves from a given state. This allows the agent to greatly improve the efficiency of move exploration.

By using an action heuristic that prioritizes certain moves which lead to attacking the opponent, this ensures the moves that are most likely the best get explored first. If these moves all have a relatively low score, this will lead the algorithm to keep exploring more other options. However, when a high-scoring move is found, there will be more focus on that move, even if there are still unexplored moves.

Because this approach relies heavily on a heuristic evaluation of all possible moves, the heuristic used is able to affect the results significantly. To ensure the strategy chosen does not rely too heavily on a rule-based approach, only a relatively simple heuristic was chosen. This heuristic was applied to the selection policy within the tree, as well as the rollout stage. Each phase has a different heuristic applied to it, as detailed in the list below.

**Selection Phase**

No heuristic implemented

**Movement Phase**

The highest priority is given to any movement action where the troop is moved into a position where it can attack an enemy troop. The same highest priority is also given to the application of the Charge or Winds of Fate commands, in order to ensure these commands are at least considered when deciding a move.

Any other movement actions are given medium priority, unless Charge has already been applied. Since the Charge command is generally only useful when charging into an attack, the movement actions that do not result in the troop being able to attack the opponent are given the lowest priority.

**Combat Phase**

Attacking any troop is given the highest priority. Applying commands is given medium priority, among which the application of a command onto the currently selected troop is given higher priority. Generally, the currently active troop will be the one that is most active in battle, meaning it is most likely in need of a boost from a command.

**Rally Phase**

The same heuristic for commands is applied from the Combat Phase.

## 7.3 Random Search

As a baseline, we implemented a Random Search agent, against which the performance of both of the main strategies (discussed before in Section 7.1 and 7.2) can be checked.

The way this was implemented was using a MCTS agent that has a maximum tree depth of a single move, with the move selection being uniform. The rollout is then set to a certain number of full turns, ensuring the same principle as mentioned in Section 7.1 applies: rollout is only terminated after a full turn is evaluated, and the same number of full turns are always evaluated for every rollout. The selection within the rollout is implemented with the same heuristic as detailed in Section 7.2.

This Random Search agent was used mainly for optimising the MCTS agents. By running the N-Tuple Bandit Evolutionary Algorithm (NTBEA) Optimisation which is implemented in the TAG framework, we performed a parameter search to optimise different MCTS parameters. This was done by varying the parameters for the MCTS agent in a matchup against a fixed RandomSearch agent, with RandomSearch functioning as a baseline. Aside from optimisation, the RandomSearch agent also functioned as a baseline in general, used for setting a baseline performance for any given Conquest setup.

## 7.4 Selecting the best MCTS agent

Before experimenting, we needed to tune the parameters of the MCTS agents, and select the best performing agent and answer Research Question 1. To do this, we tuned the FPU agent (Section 7.2), and compared it to the performance of the full-turn agent (Section 7.1). To tune the agents, we performed a MCTS parameter search using NTBEA (as mentioned in Section 7.3), playing against the RandomSearch agent. We considered the following parameters and values for both of the agents, except that the ‘First play urgency’ and ‘Re-use tree’ parameters were not applicable to the full-turn agent. The tree depth and rollout length were both defined in terms of full turns, not single moves, as was discussed at the start of Section 7.

- Maximum tree depth: 1, **3**, 5, 8, 10
- Rollout length: **4**, 6, 8, 10
- First play urgency: **10**, 100, 1000
- Branching factor $K$: **0.5**, 1.0, 1.4
- Re-use tree: **Yes** or no

After tuning both of the agents' MCTS parameters, we took the two best sets of parameters for each, and added the RandomSearch agent. This gave us a set of 5 agents in total (2 agents with 2 configurations each, plus the RandomSearch agent). These five were put into a tournament playing around 50 games each. The FPU agent with the highlighted parameters shown above ended up winning 58% of its games, with the second best parameter set of this agent winning 52% of its games. The two full-turn agents won 54% and 50% of their games, respectively. The RandomSearch agent only won 35% of its games. Based on this, we selected the agent that performed best, being the FPU agent, to perform our further experiments with.

We playtested this FPU agent ourselves, being a decently experienced player. Based on these tests, the FPU agent generally played less tactically than an experienced human does, but it very quickly punished every mistake made by their opponent. Based on our estimation, our MCTS agent could qualify as an experienced player overall. While it may be possible to improve upon this 'skill level' of the MCTS agent, we expect this level to be sufficient to be able to draw meaningful conclusions from its gameplay.

# 8 Experiments

In order to answer the research questions posed in Section 2, we set up multiple experiments. Because the experiments that determine the best strategy within the game rely on a well-functioning MCTS algorithm, we ensured the MCTS agent was optimised through the methods described in Section 7. After that was completed, we applied the results of those experiments to optimise the game's strategy, in order to answer Research Questions 3 and 4.

## 8.1 Optimising troop setups

The main experiment to be done for optimising the game is to optimise the setup players can choose. Because some setups are strong against certain other setups, finding a good strategy that will work against a lot of other strategies is generally desirable. In order to find the optimal troop setups, we entered all strategies listed on the Wiki's strategy page [4] into a big list, and ran a full Round Robin tournament between all possible setups.

This tournament consisted of 23,250 games in total, played between 31 different setups. To speed up the execution of this tournament, we expanded the TAG framework by implementing a multi-threading version of the Round Robin tournament code. By allowing the individual games to run in parallel, this significantly reduced the amount of real time needed to finish the tournament. The exact setups that were implemented for this tournament can be found in Appendix B.

With 31 different setups, it is possible to generate $\binom{31}{2} = 465$ pairs of setups, without taking into account the starting player. We then let every pair of setups play 50 games, where each setup is the starting player for exactly 25 out of the 50 games. Setups do not get evaluated against themselves, because the outcome of such a matchup does not provide any insights into the effectiveness of the strategy.

From this tournament, the resulting scores are shown in Figure 2. This shows that although there are some losing strategies, there are no strategies that are clearly dominant. However, although it is not a dominant win against all opponents, the 'IAmWarrior' setup (Setup B.29) manages to achieve a positive score against every opponent, out of 50 games played against each. This strategy makes use of three health-boosting or restoring commands, combined with the just three of troop with the most health, and no other troops. The use of these three commands in particular likely help it improve its survivability, without requiring more troops. The reduced amount of troops likely also aids the ability to calculate the best move, due to its comparatively smaller search space.

Furthermore, there are some cyclic dominances between the strategies; that is, there are structures like the game rock-paper-scissors, where three or more strategies each have one winning matchup and one losing matchup, which forms a cycle of winning strategies. Some

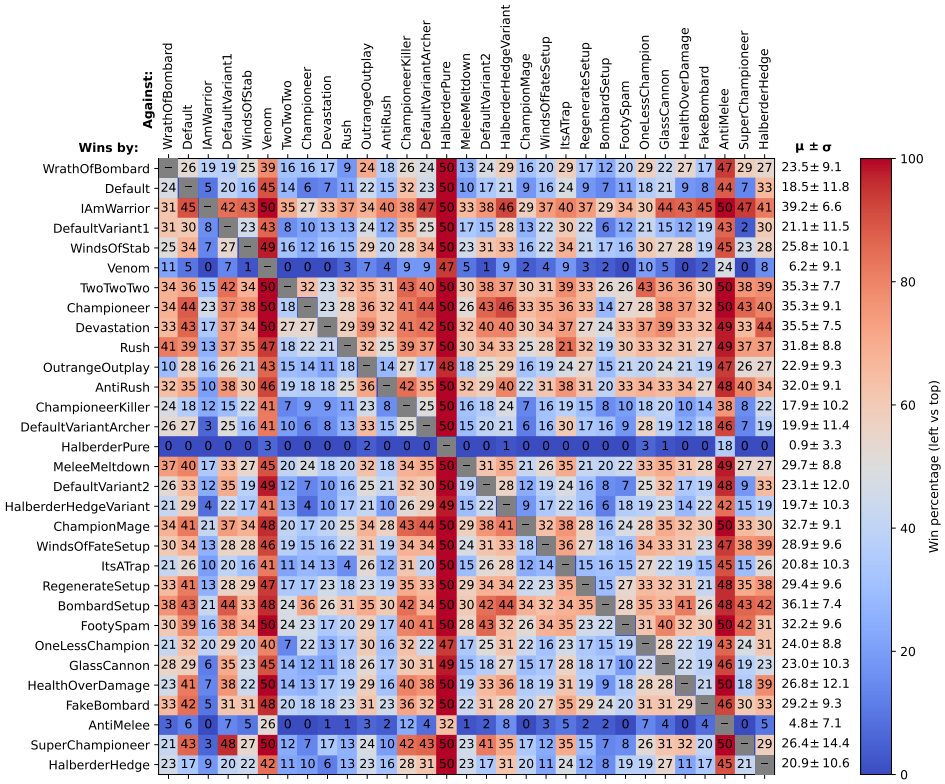


Figure 2: Results from a Round Robin tournament of 23,250 games between 31 setups. Each cell has the number of games won by the left setup, out of the 50 total games played against by the top setup. To the right, the mean and standard deviation of the amount of wins per opponent is also provided. This matrix shows that although there are some strategies that perform badly across the board, no strategies are clearly dominant against every other strategy.

of these cycles are shown in Figure 3, which is based on the data from Table A.1. For a matchup between two setup to count as a 'dominant' matchup, we chose a threshold of 29 out of 50 games that need to be won by one of the two setups. There were no cyclic dominances with a threshold of 30/50, and the number of cyclic dominances with a win rate of 28/50 already grew to 241 cycles. Given these criteria, we can see there are 11 cycles.

This figure shows that there are two independent cycles, with the bottom one relying solely on the 'DefaultVariant1' strategy winning against 'WrathOfBombard'. This means that the former is mainly used to counteract the latter, while losing against many other strategies (although there are still some strategies outside this cycle it is effective against; see Figure 2). The top cycle is much more interconnected, with the main five nodes forming a single large cycle, where two sub-cycles (being the leftmost three and the rightmost three nodes) are also present. These cycles do show that within the analysed setups, although this only occurs for a threshold of 29/50 or fewer wins, there are clear cyclic dominances present.

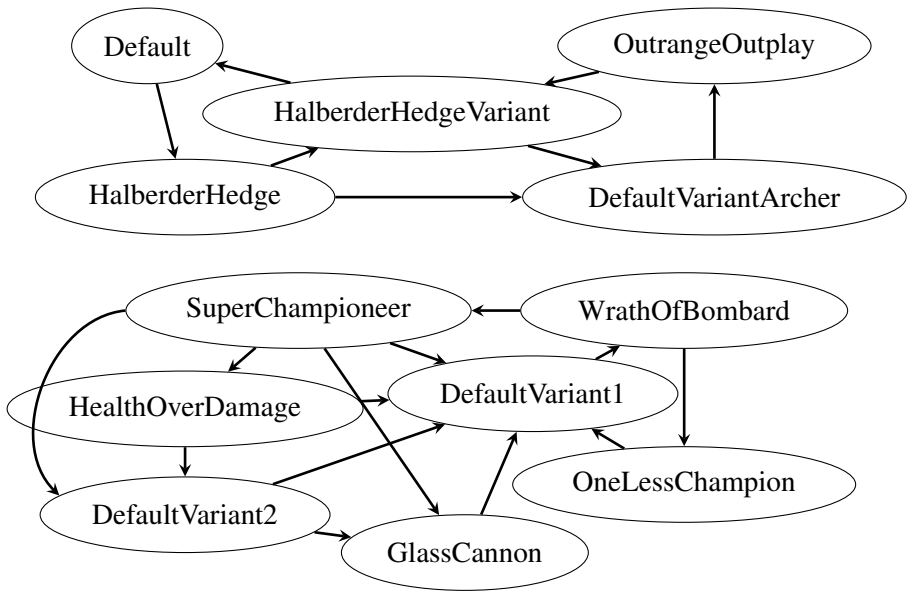


Figure 3: A graph showing the cyclic dominances in the big tournament. Two independent cycles exist, with the top one being more interconnected, showing that dominance of one strategy over another is clearly not a transitive property.

We estimate that a threshold of 29 out of 50 games is sufficient to clearly demonstrate a level of dominance, given that this requires the 'dominant' setup to win 8 more games than it loses. However, it can be interesting to also apply this approach to other games with similar setup stages. We were unable to find literature analysing cyclic dominances in wargame strategies, so we could not compare the strength of these cycles (that is, the threshold required to find cycles) to those of other games. Because the existence of cycles indicates that there is no singular dominant strategy [20], we expect the presence of strong cyclic dominances to be

an indication of the diversity of viable strategies available, which in turn could make for a better playing experience.

## 8.2 Identifying behaviour by the agents

Aside from running the full tournament, we also ran a smaller scale tournament of 4650 games, meaning a total of 10 games were played between each pair of two setups (with 150 games played in total by each setup). During these games, we logged every single move made, meaning we were able to analyse the behaviour of the agents quantitatively. This was used to track command usage, but in our second run we were also able to track more complex metrics.

We ran this experiment twice. In the first experiment, we were mainly able to track the usage of commands and the position of troops, but after obtaining the results from that run, we wanted to be able to track more detailed data about the agent behaviour. The second experiment featured more detail, also tracking the troop types that were selected, which troops they attacked, and whether this killed the enemy troop or not. Because this second experiment had some technical issues though, we will cover results from both experiments, as opposed to just using the more in-depth results.

Analysing the behaviour of the agents helps us to determine what types of strategy are more or less effective, when using our MCTS agent. This may help in developing strategies that may be more effective than the ones currently listed on the Wiki. The way this can be done is by finding powerful combinations of troops, commands, or combinations of both troops and commands.

### 8.2.1 Simple logging

In our first experiment of 4650 games, we mainly gathered statistics on the frequency of commands used. Commands that are not used as commonly either are less generally applicable, or our MCTS agent was not configured optimally to calculate the longer-term benefit from the use of these commands. The resulting command frequencies are shown in Table 3. This overview shows that, even though Battle Cry is the command featured in most setups, Chastise is the most commonly used command. And although Charge is generally used very frequently in gameplay by actual players, as seen by the frequency of its mentions in the Wiki's strategy guide [4], it is rarely used by the MCTS agent.

The average game lasted a total of $80 \pm 23.3$ turns, with each turn consisting of an average of $3.61 \pm 0.080$ moves. The shortest game lasted 24 turns (88 moves), with the longest game lasting 195 turns (696 moves).

| Name | Featured in setups | Used in tournament |
|---|---|---|
| Chastise | 17 | 51046 |
| Battle Cry | 29 | 49224 |
| Stoicism | 15 | 24779 |
| Vigilance | 7 | 21378 |
| Shield Wall | 10 | 20561 |
| Charge | 27 | 3594 |
| Bombard | 4 | 2292 |
| Winds of Fate | 2 | 2055 |
| Regenerate | 7 | 1545 |

Table 3: Statistics on command usage in a tournament of 4,650 games, between all 31 troop setups. It lists the number of setups that feature the command, along with the number of times the command was used throughout the tournament.

### 8.2.2 Extended logging

After running this experiment, we increased the detail of the logs generated by the agents. This means we were able to track which commands were used on which troops, as well as tracking which troops attack which, and whether this is a killing attack, or if it simply damages a part of the opponent's health.

We used this more detailed logging to analyse specific troops and commands more thoroughly, to be able to analyse which troops and commands can be considered more valuable, and which of them are generally less useful. In the following subsections, we will analyse a few aspects of the extended logging results, and explain the inferences we can make based on these observations, regarding which setup choices are generally more valuable, and which are less useful. This can aid in building a more optimal setup, which can be useful for potential future experiments. One way these findings can be applied to build new effective strategies is by looking at synergies between troops and commands, or considering overall efficacy of certain troops or commands.

#### Coding issues

Unfortunately, due to some coding issues, we encountered an error in the use of the Winds of Fate command, which meant that the two setups that used this command did not track any completed games at all. Any game that featured this command simply terminated after just a few turns, meaning the data from these games is unusable, and would skew the average length of games. Because of that, we disregarded any games that featured either of the setups that used Winds of Fate. For reference, the full overview of command usage is provided in Table A.2, although the data gathered in the prior run (as shown in Table 3) is more reliable for purposes of overall command usage statistics.

In this tournament, the Charge command was never used. After analysis of the code, which was reworked partially for the purposes of extended logging, we discovered this was caused

by the optimisations made to this command, which was discussed in Section 6.4. Because this did not affect the rest of the gameplay, we did not consider this a significant issue, meaning the games that feature this command can still be analysed further, albeit without any analysis about the usage of this command itself.

It seems during this run, the games took significantly longer to end. This is likely also due to an issue with the code. The average game, excluding those involving the command Winds of Fate, lasted a total of $527 \pm 242$ turns, with each turn consisting of an average of $3.13 \pm 0.076$ moves. The shortest game lasted 116 turns (387 moves), with the longest game lasting 2097 turns (6211 moves).

Although the data on command usage seems to be less reliable than hoped, we can still attempt to analyse which troops had certain commands applied to them, in order to determine if there are some expected patterns in the use of commands. We provided the raw command usage data in Table A.3. In this section, we will mainly analyse the processed data.

**Overall analysis**

Table 4 shows the relative spread of the recipients for each of the commands, indicating for each command which troops are most likely to have that command applied to them. This shows that in general Champions were most likely to have commands applied to them. However by calculating a relative metric, which is computed by taking the average command usage for that troop, and dividing it by the relative abundance of that troop occurring in the analysed list of setups (Appendix B), we can see that the Knight gets targeted with commands relatively commonly too. In this relative metric, Scouts significantly underperformed, likely due to the troop not being valuable enough to spend large amounts of command points on them. Based on this metric, the Knight and Champion are the most useful troops.

| **Command** | **S** | **F** | **H** | **A** | **M** | **K** | **C** | **Total** |
|---|---|---|---|---|---|---|---|---|
| Battle Cry | 21.6% | 2.7% | 16.0% | 4.9% | 5.7% | 9.12% | 40.0% | 100% |
| Stoicism | 19.2% | 2.9% | 17.7% | 4.4% | 4.7% | 11.2% | 39.9% | 100% |
| Regenerate | 0 | 0 | 0.3% | 0 | 0 | 0.4% | 99.3% | 100% |
| Bombard | 15.7% | 2.1% | 11.5% | 9.7% | 11.1% | 9.2% | 40.8% | 100% |
| Chastise | 21.7% | 3.3% | 18.1% | 5.3% | 6.9% | 9.5% | 35.1% | 100% |
| Vigilance | 10.4% | 2.4% | 6.9% | 7.0% | 13.0% | 11.9% | 48.6% | 100% |
| Shield Wall | 2.0% | 2.2% | 21.5% | 0.6% | 0.9% | 21.5% | 51.3% | 100% |
| **Average** | 16.8% | 2.7% | 15.7% | 4.7% | 6.4% | 11.3% | 42.3% | 100% |
| **Relative** | 16.8 | 41.3 | 36.5 | 26.5 | 29.8 | 80.6 | 94.3 | |

Table 4: Relative spread of troop recipients for each command. The 'Average' row is equal to the total amount of commands applied to that troop, divided by the total number of commands used. The 'Relative' row shows the 'Average', divided by: the troop Usage (Table 6) divided by the most commonly used troop (being 107 occurrences of the Scout). For example, Foot Soldiers are calculated as $2.7/(7/107) = 41.3$.

We further analysed the spread of commands that were applied to each of the troops, which is shown in Table 5. Here we can see that for the Foot Soldier, Halberdier and Mage, Chastise was the most commonly applied. The other four troop types most commonly had Battle Cry applied to them. Because Chastise is applied to enemy troops, and Battle Cry is applied to friendly troops, this indicates a balance between offensive and defensive tactics. Because Chastise is generally applied in response to an approach by the given troop, this command is less tactical than Battle Cry, which is generally applied in combination with a tactical advance. Because of that, we can infer that the troops to which Chastise is applied more often than Battle Cry, generally offer less tactical advantage than the troops that get to use Battle Cry more often. To make use of this distinction, it can be useful to experiment with building a setup that uses a combination of both tactical and less tactical troops. This is a possible explanation why the 'HalberderPure', which uses only Scouts and Halberdiers, performs significantly worse than most other setups.

This data also shows that Mages stand out for having Vigilance applied to them relatively often, especially compared to the Archer, which is a similar troop. This is surprising, given that the Archer is the troop with the longest attack range, meaning it is the only troop for which Vigilance is guaranteed to come into effect when the troop is attacked. However, the Mage receiving this command more commonly is likely due to its higher damage output, meaning that this (relatively cheap) command is a more effective deterrent when applied to Mages, whereas Archers benefit more from survivability using commands like Stoicism. This synergy between Vigilance and Mages can be used when building new setups.

| **Command** | **S** | **F** | **H** | **A** | **M** | **K** | **C** | **Average** |
|---|---|---|---|---|---|---|---|---|
| Battle Cry | 37.0% | 28.5% | 29.5% | 30.2% | 25.7% | 23.5% | 27.4% | 28.95% |
| Stoicism | 17.5% | 16.5% | 17.4% | 14.3% | 11.4% | 15.3% | 14.5% | 15.43% |
| Regenerate | 0 | 0 | 0.0% | 0 | 0 | 0.1% | 3.8% | 1.62% |
| Bombard | 0.67% | 0.55% | 0.53% | 1.5% | 1.2% | 0.6% | 0.70% | 0.723% |
| Chastise | 33.7% | 31.8% | 30.1% | 29.3% | 28.2% | 22.2% | 21.7% | 26.15% |
| Vigilance | 9.7% | 13.6% | 6.9% | 23.2% | 31.7% | 16.6% | 18.1% | 15.75% |
| Shield Wall | 1.3% | 9.1% | 15.6% | 1.5% | 1.6% | 21.7% | 13.8% | 11.37% |
| **Total** | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |

Table 5: Relative command usage for each troop. The 'Average' row is equal to the total number of times that command was used, divided by the total number of commands used. Values of exactly 0 indicate the command was not used on that troop at all.

### Troop usage

We also analysed the amount of times certain troops were selected. This gives an indication of which troops are used more often. Troops that were given a turn relatively often indicate these troops are considered more useful by the AI player. This may be useful information when trying to create new effective setups to play against the existing meta.

Table 6 shows the frequency of troop usage by type, as well as showing these values scaled by the frequency of that troop type occurring in the experiment, and scaled by the troop's value. Based on this data, Knights and Champions were clearly most commonly selected, scaled to the amount of that troop available. However, scaled further to their cost, the most effective troops turn out to be Scout, Foot Soldier and Knight. This despite only Scout being commonly used, with the other two troop types being less commonly selected in the strategies given by the game's Wiki.

| **Troop** | **S** | **F** | **H** | **A** | **M** | **K** | **C** |
|---|---|---|---|---|---|---|---|
| **Usage** | 107 | 7 | 46 | 19 | 23 | 15 | 48 |
| **Selected** | 140,540 | 23,016 | 127,180 | 43,941 | 50,507 | 75,264 | 332,373 |
| **Relative** | 1313.5 | 3288.00 | 2764.8 | 2312.7 | 2196.0 | 5017.6 | 6924.4 |
| **Per 25pt** | 1313.5 | 1644.00 | 921.6 | 578.2 | 548.9 | 1254.4 | 865.6 |

Table 6: Total troop counts per troop type, as analysed in the different setups shown in Appendix B. It also shows the amount of times said troop type was selected during our analysis, as well as the amount of times each troop was selected, divided by the number of times that troop occurs in the list of setups. Finally, the bottom row indicates the relative usage, further divided by the troop's cost, per 25 points. This indicates the usage relative to the troop's cost.

**Command usage**

Based on the tournament with extended logging, along with the troop usage counts from Table 6, we can see in 'Relative' row of Table 4 that the troop that got targeted by commands most commonly was the Champion. Given that the Champion is the most powerful unit, it makes sense that this unit would be targeted most by all different commands. The high rate of use of commands onto Knights given by this same table row does indicate that these troops may warrant more usage than there is in the current list of setups.

Despite being used less often than either the Mage or Archer, Knights had significantly more commands applied to them. Only for the Bombard command, Mages and Archers are targeted more commonly, which can be explained by the fact these troops do not have enough health to survive a single Bombard attack. Vigilance is more commonly applied to Mages only, which can be explained by their long-distance attacks, allowing them to utilise the command more effectively than a melee unit could. Other than those cases, Knights are more commonly targeted with commands. This further implies the Knight is a more valuable troop, being more worthwhile to spend command points on.

For the Shield Wall command, there is a significant difference in usage between the troops that have 100 health, versus the troops with more health. Because the use of this command on troops with 100 health remaining is not allowed in our implementation (for optimisation purposes discussed in Section 6.4), this means this command can only be applied after a health-boosting command was applied to it during the same turn. The relative rarity of Foot

Soldiers can partially explain why they are targeted with this command only as often as troops with 100 health, and the same reasoning explains why the Knight has as many Shield Walls applied as the Halberdier, despite having more health.

While most commands listed in Table 4 follow a fairly similar spread across different troops, there are some commands with clear deviations from the norm. Regenerate is the clearest outlier, being used almost exclusively on Champions. Because this is the troop with the largest health pool, its utility on this troop is clearly greatest. On top of that, the larger health pool allow them to survive with less than their full health for longer, meaning there are more opportunities to apply the command to it. Interestingly, despite having equal health to Knights and more health than a Halberdier, the Foot Soldier never had Regenerate applied to it. This may indicate that this troop is generally less important than other troops with larger health pools, although it could also be caused in part by Foot Soldiers being used scarcely in the tested setups, similar to the results for Shield Wall mentioned above.

Out of these commands, only Chastise and Bombard can be applied to enemy troops. Out of these, Bombard seems to be used more often on lower health troops, or on the valuable Champion (perhaps to weaken them to come within the health range where another Champion can instantly kill them). Chastise, on the other hand, has a more equal effect on all troops, simply disabling them for a single full turn. Because of this, the frequency of Chastise use can serve as a useful indicator of value, as considered by an opponent fighting against that troop. Comparing the use of Chastise to the average command frequency for that troop, using Table 5, we can see that specifically Foot Soldiers and Halberdiers have Chastise applied to them more often than any other command. This may indicate that these troops are generally considered more of a threat to the enemy than their relatively low cost implies.

**Attacking behaviour**

From our run with extended logging, we kept track of the moves made by each of the troops. In particular, we looked at the attacking moves made. Table 7 shows the tracked attacks. Interestingly, Scouts and Foot Soldiers have the lowest percentage of killing blows with their attacks, despite the risk of being counter-attacked because of their single-square attack range. The damage incurred from this counter-attack can be compensated for by the use of health-boosting commands, allowing them to attack other troops without getting damaged by the counter-attack.

Interestingly, the kill rate for both Champions and Knights is nearly 100%. This indicates that they generally do not attack other Champions, without either using Battle Cry, or having weakened the enemy beforehand by means of a prior attack or the use of Bombard. It also shows that they rarely attack enemies with Shield Wall applied to them. While this information is not as useful when determining a good setup, it can give insights into which type of moves are effective during gameplay.

| Defender→ ↓Attacker | S | F | H | A | M | K | C | Total | Kills | Kill rate |
|---|---|---|---|---|---|---|---|---|---|---|
| **Scout** | 969 | 61 | 361 | 107 | 223 | 179 | 790 | 2690 | 2042 | 75.9% |
| **Foot Soldier** | 123 | 12 | 81 | 14 | 28 | 23 | 90 | 371 | 283 | 76.3% |
| **Halberdier** | 1507 | 150 | 854 | 169 | 269 | 312 | 1127 | 4388 | 3819 | 87.0% |
| **Archer** | 1699 | 91 | 455 | 209 | 377 | 260 | 887 | 3978 | 3163 | 79.5% |
| **Mage** | 1234 | 78 | 867 | 95 | 203 | 244 | 845 | 3566 | 3142 | 88.1% |
| **Knight** | 296 | 60 | 214 | 43 | 66 | 112 | 353 | 1144 | 1130 | 98.8% |
| **Champion** | 1526 | 197 | 1240 | 253 | 391 | 689 | 1976 | 6272 | 6241 | 99.5% |

Table 7: Amount of attacks between any set of attackers and defenders, along with the total number attacks made by that troop. It also indicates which proportion of the attacks made by that troop were dealing enough damage to kill the attacked troop.

# 9 Conclusions

Based on the large number of possible setups, it is not feasible to analyse the full set of maximal troop setups that were generated in Section 5. Although in this paper we focused solely on the recommended setups given on the Wiki's strategy guide, expanding this to include all possible setups is not feasible. Applying a PCG strategy such as Reinforcement Learning or an Evolutionary Algorithm, such as the approach mentioned in [17], may make it feasible to construct more effective setups than those analysed in this paper. We were able to answer Research Question 2 though, by computing all possible permutations of setups. This resulted in 1.7 million total setups, out of which roughly 284 thousand are maximal setups.

Although a MCTS-Minimax hybrid such as the full-turn playout described in Section 7.1 may seem useful in this game, given that it involves multiple moves per turn, we can conclude from comparing its performance to a First Play Urgency approach that this does not perform better. Modifying the FPU in order to more quickly explore longer sequences of moves, instead of analysing more shallow move sequences, has a more successful effect. This answers Research Question 1. An implementation that modifies the First Play Urgency to prioritise exploitation over full exploration is a very effective way to quickly traverse a large search tree like that of Conquest.

The agent behaviour we analysed in order to answer Research Question 4 shows that, relative to their occurrence in the setups that were analysed (listed in Appendix B), the most frequently used troops were not necessarily the troops that were most prevalent troops. Both in the amount of times the troops were selected (Table 6), and in the amount of times commands were applied to them (Table 4), we can see that Knights score highly, indicating their overall high value. With these two metrics, we can also see that Halberdiers, Archers and Mages are under-utilised by the AI, relative to their prevalence in the Wiki's setups. This may indicate

that the AI values long-ranged attacks less than the human players that wrote the strategy guide do.

Given the data we collected, we think we provide several new insights into the game, making it possible to develop new strategies based on them. For example, setups which utilise the Knight and Foot Soldier more often can prove to be useful in practice, given that in our results these troops were considered more valuable than their scarce usage would imply.

In answering Research Question 3 we were able to find some setups that outright lost against the majority of the other setups, even losing all 50 of the evaluated games against several other setups. Although these results are very clear for the worst setups, the best setup is not just as clear. However, we were able to find some strategies that performed very well, with one strategy even winning over half of its games against every single opponent (Setup B.29).

Because there are cyclic dominances in the game, there generally should be no single best strategy [20], so Setup B.29 winning the majority of games in each matchup does not automatically mean this is the overall best strategy in the entire game. It does imply that there are likely to be currently undocumented strategies which could dominate this strategy. This means that further strategising, testing, or researching is needed to counteract this strategy.

The analysis of the usage of different troops and commands given in Section 8.2.2, yielded some interesting insights, which could be taken into account when building new setups. Overall, Knights turned out to be more valuable than their current prevalence suggests. Foot Soldiers also showed some potential, given that they were targeted by the Chastise command often, but more testing is required to definitively assess this troop, as many of our results on the Foot Soldier could easily be attributed to their relative scarcity in the list of setups we used.

The combination of the use of Mages and the Vigilance command seems to be a strong combination, based on our results. The prevalence of Champions in our list of setups also seems to be warranted, given the high performance of setups that prominently feature them, as well as many of the other metrics listed in Section 8.2. Both Champions and Knights performed very well overall, with both of these troops almost always killing their targets in a single hit, and both of these troops being selected by the MCTS agent very commonly to apply commands onto.

# 10 Future research

Based on our findings in this paper, we think further research into Conquest may be very beneficial both in improving strategies for the game itself, and for of its advantages over other similar games when it comes to computation, which helps improve the field as a whole.

## 10.1 Conquest-specific research

Future research into Conquest may attempt to determine if the placement of obstacles affects different setups unequally. The placement of obstacles may be more beneficial to the Ranger or Mage, but it could also be the case that the troops with greater health pool such as Knight, Foot Soldier and Champion benefit from obstacles more. Because this would increase the variance per game though, inserting more uncertainties, we did not attempt to answer this question in this paper.

Another point of interest could be the more deterministic variant of Winds of Fate we applied in our experiments. Because this command always behaved the same, given a certain game state, applying this command would result in a predictable outcome, meaning that the MCTS agents were able to explore the future outcome after using it. It would be interesting to see if changing it to be stochastic, as it is in the actual game, would affect the usage of the command. Because this command is already rarely used despite its predictable outcome, we suspect this would make the command even less viable than it was during our experiments, meaning that this likely does not have a significant effect on our conclusions, other than perhaps making the setups that use it slightly less effective.

This paper has focused entirely on optimising for the current rules of the game. However, future research could attempt to rebalance some of the attributes of the troops and commands in order to facilitate more different strategies in the game. Examples include changing the costs for individual troops or commands, or the specific properties such as their health or movement. Alternatively, the setup limitations of point total, troop count and command count could be rebalanced.

Further research could also be done by dynamically generating setups, in order to find strategies that have not been explored on the Wiki's strategy page yet. As discussed in Section 5, knowing the amount of possible setup permutations lays the groundwork to exploring the efficacy of new strategies that are currently unknown by players. An approach to generate effective strategies from this list could be to apply PCG approaches, in order to generate effective starting setups. An approach such as [25], which uses Reinforcement Learning to generate a balanced level design, could be applied to generate a setup which is as effective as possible. Approaches such as [26] could also be applied to generate effective team compositions, in order to generate effective setups.

With regard to the most effective MCTS agent, it may be interesting to attempt to combine the two main approaches described in Section 7. Using FPU in combination with a MCTS-Minimax hybrid approach may prove more effective than applying only one of the two approaches. In this report, we did not attempt to implement this approach, due to time constraints. Considering either approach improved significantly over plain MCTS agents, exploring such a combination further may prove to be a useful improvement over the current implementation, though.

## 10.2 General strategy game and wargame research

We expect Conquest to be a useful game to analyse the same concepts that exist in much more complex games in the wargame category. In future research, it would be useful to attempt to do more concrete research into the parallels that can be drawn between Conquest and other games in the genre, in order to establish more clearly what types of research can benefit from analysis of this game.

Conquest has proven to be a fairly simple game in the category nonetheless, meaning it can be very helpful for future research into this genre to apply novel approaches onto Conquest first, before attempting to apply them on more complex wargames. The relatively small set of parameters that go into the troops and commands of Conquest make it much easier to analyse different aspects of the game.

Furthermore, although Conquest has a relatively large search space in absolute terms, the fact its moves are discretized onto a grid has major advantages over many other wargames, since these require different approaches to deal with the continuous search spaces of their distance-based movements.

As discussed in Section 8.1, the presence of cyclic dominances may indicate a level of variety in viable strategies. Unfortunately, we were unable to find prior research into cyclic dominances in wargames, meaning we could not compare our findings to the literature. Future research into finding such cycles in different wargame strategies, in order to determine how well this metric can serve as a metric of variety among the viable strategies.

The overall approach we used to analyse Conquest may be effective at analysing other similar games as well. Of course it is always effective to reduce the action space and to prune the search tree, whenever this does not impact the gameplay itself, but the other strategies we used to analyse Conquest may be useful in future wargame research as well. Like the cyclic dominances mentioned earlier, the analysis of actions taken during a game which we performed in Section 8.2 can also prove very useful in finding new meta strategies for the game.

# A Supplementary tables

In this appendix we will provide some supplementary tables, which show additional information that could not concisely be shown in the main body of this paper.

| Cycle length | Cycle |
|---|---|
| 3 | WrathOfBombard → OneLessChampion → DefaultVariant1 |
| 3 | WrathOfBombard → SuperChampioneer → DefaultVariant1 |
| 4 | WrathOfBombard → SuperChampioneer → DefaultVariant2 → DefaultVariant1 |
| 5 | WrathOfBombard → SuperChampioneer → DefaultVariant2 → GlassCannon → DefaultVariant1 |
| 4 | WrathOfBombard → SuperChampioneer → GlassCannon → DefaultVariant1 |
| 4 | WrathOfBombard → SuperChampioneer → HealthOverDamage → DefaultVariant1 |
| 5 | WrathOfBombard → SuperChampioneer → HealthOverDamage → DefaultVariant2 → DefaultVariant1 |
| 6 | WrathOfBombard → SuperChampioneer → HealthOverDamage → DefaultVariant2 → GlassCannon → DefaultVariant1 |
| 5 | Default → HalberderHedge → DefaultVariantArcher → OutrangeOutplay → HalberderHedgeVariant |
| 3 | Default → HalberderHedge → HalberderHedgeVariant |
| 3 | HalberderHedgeVariant → DefaultVariantArcher → OutrangeOutplay |

Table A.1: The cyclic dominances that exist in the data for the big tournament. The relation $A \to B$ indicates $A$ dominates $B$ with at least 29 out of 50 of the games played between them being won by $A$.

| Name | Featured in setups | Used in tournament |
|---|---:|---:|
| Battle Cry | 29 | 91562 |
| Chastise | 17 | 82727 |
| Vigilance | 7 | 49818 |
| Stoicism | 15 | 48808 |
| Shield Wall | 10 | 35970 |
| Regenerate | 7 | 5128 |
| Bombard | 4 | 2287 |
| Charge | 27 | 0 |
| Winds of Fate | 2 | 0 |

Table A.2: Statistics on command usage in the extended tournament. It lists the number of setups that feature the command, along with the times the command was used in this tournament. Due to some issues with a subset of the commands, this data was not used for analysis.

| **Command** | **S** | **F** | **H** | **A** | **M** | **K** | **C** | **Total** |
|---|---|---|---|---|---|---|---|---|
| Battle Cry | 19732 | 2470 | 14673 | 4510 | 5199 | 8356 | 36622 | 91562 |
| Stoicism | 9349 | 1431 | 8660 | 2139 | 2318 | 5460 | 19451 | 48808 |
| Regenerate | 0 | 0 | 16 | 0 | 0 | 20 | 5092 | 5128 |
| Bombard | 358 | 48 | 263 | 221 | 253 | 211 | 933 | 2287 |
| Chastise | 17964 | 2760 | 15001 | 4370 | 5717 | 7894 | 29021 | 82727 |
| Vigilance | 5167 | 1181 | 3428 | 3467 | 6457 | 5923 | 24195 | 49818 |
| Shield Wall | 716 | 787 | 7751 | 222 | 321 | 7725 | 18448 | 35970 |
| **Total** | 53286 | 8677 | 49792 | 14929 | 20265 | 35589 | 133762 | 316300 |

Table A.3: The raw data to produce Tables 4 and 5. It shows the exact number of times a certain command was applied to a given troop.

# B List of used troop loadouts

This appendix lists all different setups used in our experiments, along with their name. These are all directly taken from the RuneScape Wiki's strategy page [4], or if no explicit loadout is given on the strategy page, it is an interpretation of the description given there. The Default setup, which is simply the game's default given to players who haven't played the game before, can be seen in Figure 1. The setup names are all taken from the Wiki, with alternate variants of strategies getting 'Variant' appended to their name.

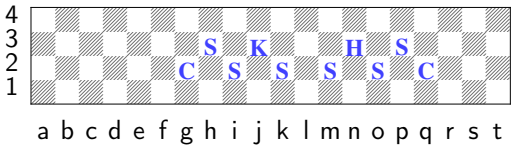


Figure B.1: *Rush* setup, using Battle Cry, Charge, Shield Wall, Stoicism.

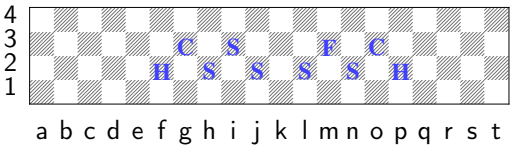


Figure B.2: *AntiRush* setup, using Charge, Stoicism, Chastise, Battle Cry.

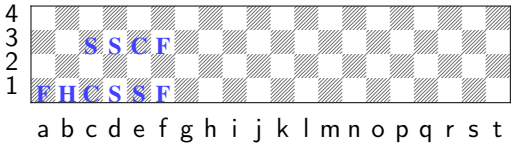


Figure B.3: *FootySpam* setup, using Charge, Battle Cry, Stoicism, Chastise.

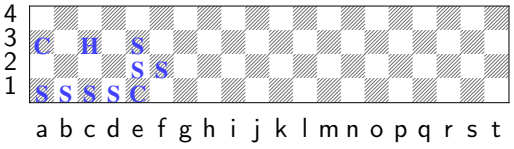


Figure B.4: *RegenerateSetup* setup, using Charge, Battle Cry, Chastise, Regenerate.

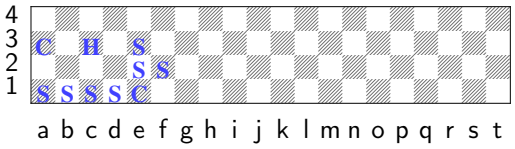


Figure B.5: *WindsOfFateSetup* setup, using Charge, Battle Cry, Chastise, Winds Of Fate.

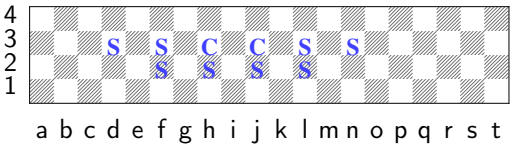


Figure B.6: *BombardSetup* setup, using Charge, Battle Cry, Chastise, Bombard.

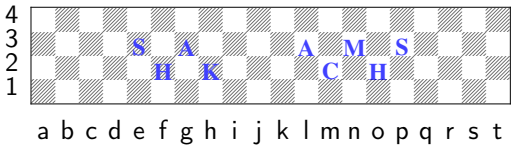


Figure B.7: *DefaultVariantArcher* setup, using Charge, Battle Cry, Chastise.

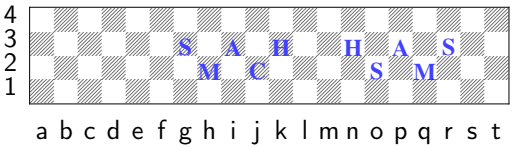


Figure B.8: *DefaultVariant1* setup, using Battle Cry, Chastise, Vigilance.

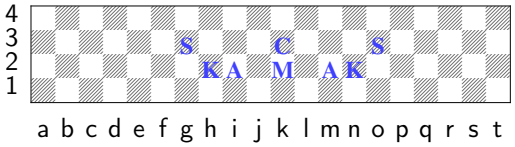


Figure B.9: *DefaultVariant2* setup, using Charge, Battle Cry, Chastise, Vigilance.

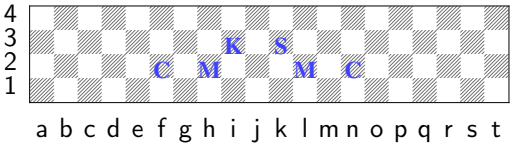


Figure B.10: *ChampionMage* setup, using Battle Cry, Vigilance, Regenerate.

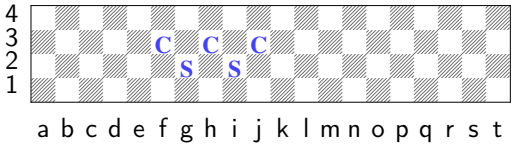


Figure B.11: *Championeer* setup, using Battle Cry, Charge, Regenerate, Shield Wall.

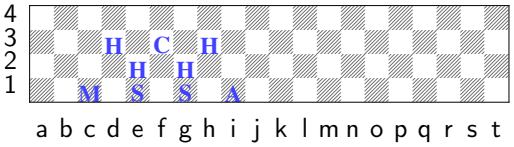


Figure B.12: *ChampioneerKiller* setup, using Charge, Battle Cry, Shield Wall, Chastise.

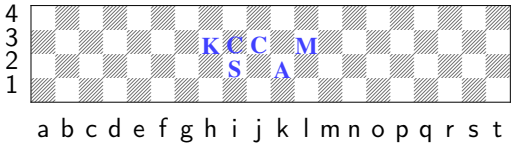


Figure B.13: *Devastation* setup, using Charge, Shield Wall, Battle Cry, Stoicism.

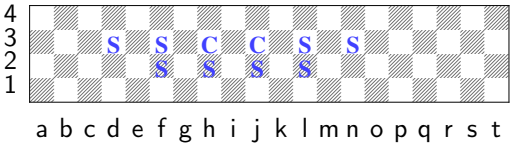


Figure B.14: *FakeBombard* setup, using Charge, Battle Cry, Regenerate, Stoicism.

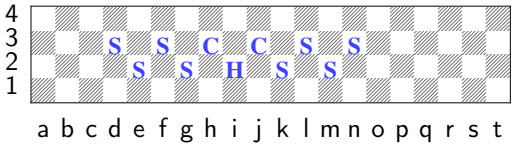


Figure B.15: *WindsOfStab* setup, using Charge, Battle Cry, Shield Wall, Winds Of Fate.

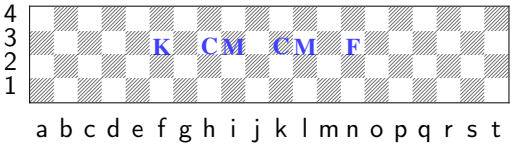


Figure B.16: *HealthOverDamage* setup, using Stoicism, Charge, Shield Wall, Vigilance.

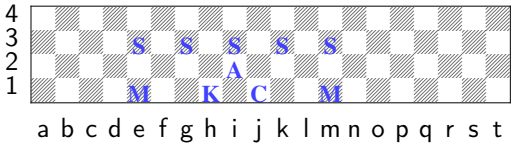


Figure B.17: *ItsATrap* setup, using Battle Cry, Stoicism, Charge, Shield Wall.

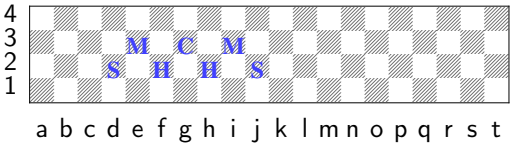


Figure B.18: *OneLessChampion* setup, using Bombard, Vigilance, Charge, Battle Cry.

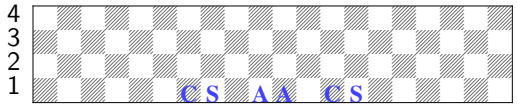


Figure B.19: *TwoTwoTwo* setup, using Chastise, Battle Cry, Stoicism, Regenerate.

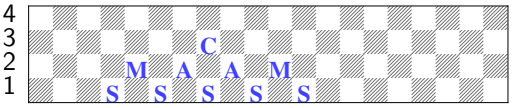


Figure B.20: *GlassCannon* setup, using Charge, Battle Cry, Stoicism, Chastise.

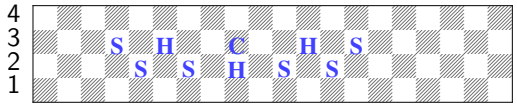


Figure B.21: *WrathOfBombard* setup, using Battle Cry, Charge, Stoicism, Bombard.

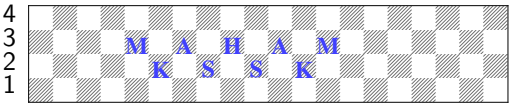


Figure B.22: *Venom* setup, using Stoicism, Battle Cry, Charge, Shield Wall.

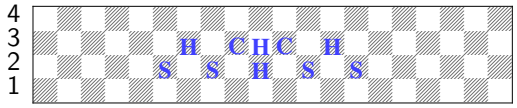


Figure B.23: *MeleeMeltdown* setup, using Battle Cry, Charge, Chastise, Shield Wall.

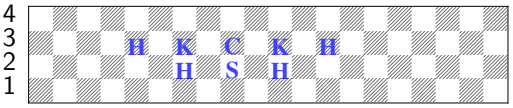


Figure B.24: *HalberderHedge* setup, using Battle Cry, Charge, Chastise, Stoicism.

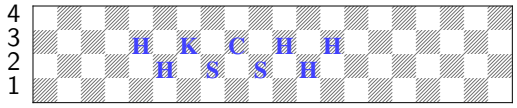


Figure B.25: *HalberderHedgeVariant* setup, using Battle Cry, Charge, Shield Wall, Stoicism.


Figure B.26: *OutrangeOutplay* setup, using Charge, Battle Cry, Chastise, Bombard.

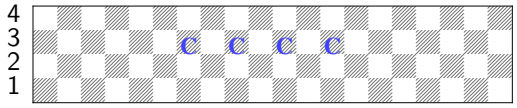


Figure B.27: *SuperChampioneer* setup, using Regenerate, Vigilance.

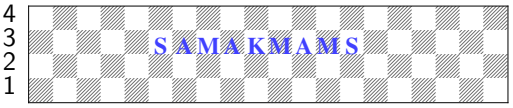


Figure B.28: *AntiMelee* setup, using Chastise, Battle Cry, Charge, Vigilance.

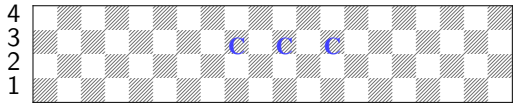


Figure B.29: *IAmWarrior* setup, using Charge, Stoicism, Regenerate, Battle Cry.

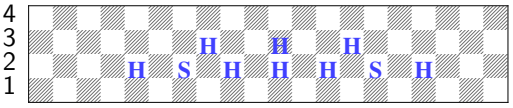


Figure B.30: *HalberderPure* setup, using Charge, Battle Cry, Chastise, Stoicism.

# References


[1] Ltd. Jagex. Conquest – news – runescape. https://secure.runescape.com/m=news/conquest. Accessed: 2025-04-15.

[2] The RuneScape Wiki. Conquest. https://runescape.wiki/w/Conquest. Accessed: 2025-04-15.

[3] Ltd. Jagex. Runescape. https://www.runescape.com/community. Accessed: 2025-04-15.

[4] The RuneScape Wiki. Strategies for conquest. https://runescape.wiki/w/Conquest/Strategies. Accessed: 2025-04-15.

[5] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] Yizao Wang and Sylvain Gelly. Modifications of uct and sequence-like simulations for monte-carlo go. In *2007 IEEE Symposium on Computational Intelligence and Games*, pages 175–182. IEEE, 2007.

[7] Hendrik Baier and Mark HM Winands. Mcts-minimax hybrids with state evaluations. *Journal of Artificial Intelligence Research*, 62:193–231, 2018.

[8] Raluca D. Gaina, Martin Balla, Alexander Dockhorn, Raul Montoliu, and Diego Perez-Liebana. TAG: A Tabletop Games Framework. In *Experimental AI in Games (EXAG), AIIDE 2020 Workshop*, 2020.

[9] Queen Mary University of London. Tag | tabletop games. https://tabletopgames.ai/. Accessed: 2025-04-15.

[10] George EM Long, Diego Perez-Liebana, and Spyridon Samothrakis. Balancing wargames through predicting unit point costs. In *2023 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2023.

[11] Edoardo Polimeno, Christian Catalano, Michele Scalera, and Marco Biagini. Hobby wargames: a preliminary survey. *Multimedia Tools and Applications*, pages 1–26, 2024.

[12] Kyowoon Lee, Sol-A Kim, Jaesik Choi, and Seong-Whan Lee. Deep reinforcement learning in continuous action spaces: a case study in the game of simulated curling. In *International conference on machine learning*, pages 2937–2946. PMLR, 2018.

[13] Stefan Freyr Gudmundsson and Yngvi Björnsson. Mcts: improved action selection techniques for deterministic games. In *Proceedings of the IJCAI-11 Workshop on General Game Playing (GIGA'11)*, pages 23–30, 2011.

[14] Kaan Unlu. Determinism versus stochasticity in the action economy of turn-based tactics games. 2023.

[15] Tristan Cazenave, Abdallah Saffidine, Michael Schofield, and Michael Thielscher. Nested monte carlo search for two-player games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[16] A. Fernández-Ares, A. M. Mora, J J. Merelo, P. García-Sánchez, and C. Fernandes. Optimizing player behavior in a real-time strategy game using evolutionary algorithms. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2017–2024, 2011.

[17] Vanessa Volz, Günter Rudolph, and Boris Naujoks. Demonstrating the feasibility of automatic game balancing. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 269–276, 2016.

[18] Fernando de Mesentier Silva, Rodrigo Canaan, Scott Lee, Matthew C. Fontaine, Julian Togelius, and Amy K. Hoover. Evolving the hearthstone meta. In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019.

[19] Daniel Hernandez, Charles Takashi Toyin Gbadamosi, James Goodman, and James Alfred Walker. Metagame autobalancing for competitive multiplayer games. In *2020 IEEE Conference on Games (CoG)*, pages 275–282. IEEE, 2020.

[20] Sebastian J Schreiber and Timothy P Killingback. Spatial heterogeneity promotes coexistence of rock–paper–scissors metacommunities. *Theoretical population biology*, 86:1–11, 2013.

[21] Haoyang Chen, Yasukuni Mori, and Ikuo Matsuba. Solving the balance problem of on-line role-playing games using evolutionary algorithms. 2012.

[22] Levi HS Lelis. Planning algorithms for zero-sum games with exponential action spaces: A unifying perspective. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 4892–4898, 2021.

[23] Edouard Leurent and Odalric-Ambrym Maillard. Monte-carlo graph search: the value of merging similar states. In Sinno Jialin Pan and Masashi Sugiyama, editors, *Proceedings of The 12th Asian Conference on Machine Learning*, volume 129 of *Proceedings of Machine Learning Research*, pages 577–592. PMLR, 18–20 Nov 2020.

[24] Benjamin E. Childs, James H. Brodeur, and Levente Kocsis. Transpositions and move groups in monte carlo tree search. In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 389–395, 2008.

[25] Florian Rupp, Manuel Eberhardinger, and Kai Eckert. Balancing of competitive two-player game levels with reinforcement learning. In *2023 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2023.

[26] Mike Preuss, Nicola Beume, Holger Danielsiek, Tobias Hein, Boris Naujoks, Nico Piatkowski, Raphael Stüer, Andreas Thom, and Simon Wessing. Towards intelligent team composition and maneuvering in real-time strategy games. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):82–98, 2010.