University of Derby
School of Computing & Mathematics

A project completed as part of the requirements for the
BSc (Hons) Computer Games Programming

entitiled

INVESTIGATING APPROACHES TO AI FOR TRUST-BASED, MULTI-AGENT BOARD GAMES WITH IMPERFECT INFORMATION WITH DON ESKRIDGE'S "THE RESISTANCE"

by
Daniel P. Taylor
dtconfect@gmail.com

in the years
2013 - 2014

Abstract

The challenge of game-playing artificial intelligence for classic board and card games such as Chess, Backgammon, Go, and Poker, has for a long time received much attention from researchers, resulting in well developed theory, capable AI agents, and a variety established techniques. Modern board games, a loose category of board or card games developed in relatively recent years, expand upon this challenge by introducing additional opponents, or problems such as variable initial setup, non-determinism, and imperfect information. Such games have received comparatively little interest from AI researchers despite potentially providing a stepping stone between classical board games and more complex domains such as computer games. In this paper we present the hidden identities game, "The Resistance", created by Don Eskridge, for which there is yet no developed theory, and no substantial published work. With the aim of informing further work on "The Resistance", we provide an overview of the game and the mechanics which make it an interesting subject, a review of related games and AI techniques, descriptive analysis of existing agent implementations, and original investigation focussing on the analysis of "suspicion inaccuracies" and opponent modelling techniques. We conclude with a summary of our analysis and specific suggestions for the direction of future research.

Contents

List of Figures

1 Introduction

"The Resistance" is a modern, hidden identities board game for five to ten players where teams are elected to participate in up to five, highly simplified missions. Mission success is simply a matter of each player on the team choosing it, but a small number of players are in fact spies who seek to sabotage these missions.

Human interest in this apparently simple game often comes from social interactions involving deception and manipulation. However, "The Resistance" also poses an interesting challenge for artificial intelligence (AI) due to the short game length, the important role of trust between players, and the inability in some circumstances to determine who did what. Development of game theory around "The Resistance" is another problem, as the game differs significantly from games like Chess or Go, which feature pieces positioned within some physical space, and also from Poker, due to the lack of random mechanics such as card draw, and the importance of logical deduction.

Previous work on "The Resistance" is scarce, being limited almost exclusively to that conducted by the "AI for modern Board Games" workgroup at the Dagstuhl Seminar 12191, "Artificial and Computational Intelligence in Games" [Spronk et al., 2012], and a subsequent competition hosted by AiGameDev.com [AiGameDev.com Team, 2012]. For these works a computer implementation of the game and a number of AI players, also known as "agents" or "bots", were created. However, there is yet no published work which seeks to provide a substantial investigation and analysis of the game.

Therefore, as the first formal work of this sort, the primary objective of this paper is to help direct and guide future development by providing a general overview of important concepts in the game, related games and techniques, and the current state of work on "The Resistance".

Further contributions include modifications to the competition environment which may help to determine the viability of learning agents for play against human opponents, scripts for logging and graphing additional data to aid analysis of bots' behaviour, and an investigation into the effectiveness of expert rules and opponent modelling techniques.

1.1 Organisation

Section 2 begins with a more detailed description of the subject game, including a thorough description of the rules and procedure of play, the features which distinguish it from other games, and a few preliminary considerations from the game theoretic perspective. This is followed by a literature review (Section 3) which seeks to explore related board games, techniques in game-playing AI, and to draw comparisons to "The Resistance".

In Section 4 description and analysis of existing agents is presented to give an understanding of the current state of the art and aid in understanding later discussion. Next we provide justification for our focus in the rest of the work before detailing the implementations we contribute. These contributions are the modified competition environment, scripts for logging and graphing "suspicion inaccuracies" (utilized heavily in later analysis), and several AI players used in our experiments. Discussion of our opponent modelling agent necessitates further consideration of the game and opponent behaviour. Finally, we describe the experiments conducted.

Section 5 presents the results of these experiments in the form of graphs and competition results, which are then analysed to a much greater extent in Section 6, providing insights into the nature of the game and effectiveness of certain techniques.

Our work concludes in Section 7, which summarises the current state of AI in "The Resistance", the insights gained from our experiments and analysis, and makes suggestions for directions of future work.

1.2 Resources

At the time of writing the Python framework resulting from the Dagstuhul workgroup and Vienna competition, and all of the existing bots discussed in this paper, are available online, from a public GitHub repository, at: `https://github.com/aigamedev/resistance`.

Investigation and implementation discussed in this paper builds upon upon this code base, and we endeavour to make our contributions equally accessible. Initially we will commit these contributions to a fork, available online at: `https://github.com/dtConfect/resistance` [1].

Further discussion on AI for "The Resistance", focussing on the AiGameDev.com competition, can be found on the public mailing list, "THE RESISTANCE AI", online at: `https://groups.google.com/forum/#!forum/resistance-ai`.

1.3 Acknowledgements

I wish to thank my supervisor, Dr. Tommy Thompson, for his initial interest in this project, for directing my attention to the original works upon which it is built, and for his support and guidance throughout.

I would also like to thank all of the contributors of the original work, including the members of the "AI for Modern Board Games" workgroup at the Dagstuhl Seminar 12191, "Artificial and Computational Intelligence in Games" [Spronk et al., 2012], the organisers and entrants of the competition organised by AiGameDev.com [AiGameDev.com Team, 2012].

Further thanks is owed to Alex J. Champandard for his interest and advice, for supplying updates to the original codebase as well as the bots from the Dagstuhl workgroup; and to Pieter Spronck for his interest and advice.

[1] Assume that any file or path referred to in our Methodology, or elsewhere, refers to the contents of this repository.

2 The Resistance

This section provides a brief overview of the game and its rules so that the following discussion may be fully understood by those not already familiar. We end by discussing some of the features of "The Resistance" which make it an interesting subject for research.

"The Resistance" is a modern board game created by Don Eskridge with a focus on hidden identities and deductive reasoning. Players are each assigned a role as either a spy or resistance member, and the game is played to a maximum of 5 turns, or "missions", with victory awarded to the first group (Spies or Resistance) to turn 3 missions to their favour. Spies require missions to fail and are allowed to secretly identify one-another at the beginning of the game, while resistance members require them to succeed and must try to work out who they can trust in order to make more informed decisions.

Variations such as Avalon, the inclusion of additional roles, or "plot cards" can add additional depth, and some features such as the number of spies or participants in each mission vary based on the size of the game (number of players, 5-10). For this research, we focus on the simplest game mode: 5 players, consisting of 2 spies and 3 resistance, with no additional roles or plot cards. This was the focus of previous work we will be using as a baseline, and provides ample depth.

2.1 Gameplay

At each turn the player with the "leadership token" selects, from all players (including himself), a number of players to go on a mission. The size of this team is determined by the mission number (for a 5 player game, for each mission in turn, the team sizes are 2, 3, 2, 3, 3). The leadership token moves to the player on his left, and each player participates in a public vote, specifying whether they approve or reject the mission. If the number of approvals is greater than the number of rejections the mission goes ahead, otherwise the process repeats with the player now in possession of the leadership token selecting the team. If 5 team selections are rejected in a single turn, victory for the whole game is awarded to the Spies.

Missions are a simple affair where each player from the approved team secretly submits a "success" or "fail" Card. While resistance members must submit a success card, spies may select either success or failure. These are shuffled before the outcome is revealed, preventing trivial identification of spies in most cases. A mission fails if even a single fail card was submitted.

Open discussion is allowed at all stages of the game, but it is advantageous for spies to pretend that they are resistance members, and players may not reveal their role cards, which are the only material evidence of their role.

2.2 Analysis

Perhaps the most obvious aspect which differentiates "The Resistance" from other boardgames is the strong focus on the other players rather than the mechanics. Parallels could be drawn to Poker, where an opponent's objectives are known, his actions fully observable, but his state - the cards in his hand - not known. However, "The Resistance" allows for logical deduction of players' identities (and thereby objectives), and while it does feature some element of chance in interaction with players of uncertain objectives, it is also deterministic - there is no random element such as a dice roll or card draw. Due to this, observant resistance members can significantly raise their chances of victory by identifying their enemies, to the point (in some cases) where it is possible to

guarantee victory. In Poker, regardless of observation, there will always be some chance of failure due to card draw. This greater availability of data supports the exploration of a broader range of approaches.

The nature of the partial observability of player actions on a mission is another distinguishing feature which, to our knowledge, is unique to "The Resistance": Given a three-man mission with one sabotage, for example, we know exactly how many of which action were taken, but we usually do not know who did what. This differs from Poker, where we always know who did what, but do not know for certain why. Similar hidden identity games such as Mafia, or "Werewolf", come close to this, but as no teams are selected in those games the treachery is near-completely unobserved and players must depend almost entirely on social cues.

The significance of the other players in the game complicates not only the gameplay, but also the analysis of player performance. In other games we sometimes encounter the issue of strategies "overfitting" to certain opponents, and failing against strategies we would normally consider weaker. Here, players are required to cooperate with each other in some sense and are dependent, especially in the case of resistance members, on their teammates: Being paired with two incompetent resistance members may in fact render the game unwinnable should they opt to vote positively for teams obviously containing spies, and there is no mechanic allowing one to "strong-arm" the decision, especially in environments which cut out the social aspect of the game, such as the implementation discussed in this paper. Furthermore, we believe the game differs enough between the spy and resistance perspectives that playing strength in one role may give no indication of playing strength in the other. For these reasons an individual's play strength may not be accurately reflected in the outcome of a game, which may be detrimental to experimentation , but we believe that this difficulty can be overcome by controlling the player set and running a large number of tests.

When considering games in the context of game theory or artificial intelligence works we often discuss their complexity in terms of the state space - the number of possible configurations of all parameters making up a single state of the game, for example the positions of pieces in chess - and properties of the game tree (discussed in Section 3.4). Though we made some preliminary attempts to calculate such measures for "The Resistance" there are a number of features which make this a surprisingly difficult task considering the simple rule set; we might even suggest that consideration of the state variables and estimation of the complexity of "The Resistance" would merit a study all of its own.

If we consider the game to consist of three significant phases (selection, voting and mission execution), with all simultaneous player actions (voting, mission execution) collated into single actions, we may begin by estimating the "depth" of the game tree - the number of states we will pass through in a single, typical game. The game may last for three-to-five missions, each consisting of one-to-five pairs of selection and voting phases, and one mission execution phase, giving us a minimum depth of $(1 + (2 \times 1)) \times 3 = 9$, and a maximum depth of $(1 + (2 \times 5)) \times 5 = 55$.

To get any further we must consider what variables the game's state consists of, and what granularity to represent actions at. Do we care, for example, about the 2^5 possible voting permutations for individual players, or just whether the vote passed or failed? We may even argue that with the history of a game being so important, data from past votes and missions should be present in the state. If not, this data will still be represented in the game tree, which grows exponentially over the course of selection and voting phases, with terminal states present at various depths. Missions four and five sometimes won't be reached due to early spy or resistance domination. What's more, possibility of mission outcomes is dependant upon the identities of players on the team since resistance members cannot sabotage, so some mission failure branches will not be valid.

3 Literature Review

This section presents background information, information on techniques applied in this research, related research, and analysis thereof. Topics covered include previous work on "The Resistance" and other modern board games, Monte-Carlo Tree-Search, Opponent Modelling techniques integrated with Monte-Carlo Tree-Search, applications of these techniques to the games discussed, and how these problems differ from "The Resistance".

3.1 AI for Modern Board Games

The interest of Artificial Intelligence (AI) researchers in modern board games is highlighted in [Spronk et al., 2012] and [Chaslot et al., 2008], which suggest that they may be helpful in bridging the gap between research in classic board games such as Chess, Checkers, or Go, and modern computer games.

The complications over classic board games which are common in modern board games include variable initial setup, non-deterministic elements, imperfect information, and the presence of more than one opponent. Non-determinism refers to the idea that given a particular state of the game and a particular action taken at that state, the resultant state may not always be the same due some stochastic (random) element such as a dice roll or shuffled deck. Imperfect information games are those where the entire state of the game is not always apparent to every player, such as in card games where there are often face-down cards and players may also hold a "hand" of cards.

All of these aspects are present in "The Resistance" with the exception of non-determinism (discounting the actions of other players), and each contributes in some way to disrupting the effectiveness of techniques often applied to classic board games, such as tree-search techniques (Section 3.4) and opening books - the latter referring to the use of established, strong opening moves, as are often described in Chess literature.

Indeed, the lack of application of such techniques (especially tree-searches) to modern computer games is part of the stated motivation of [Spronk et al., 2012], which covers the inception of the "The Resistance" framework used for this research. [Szita et al., 2009] and [Chaslot et al., 2008] indicate that Monte-Carlo Tree-Search (Section 3.5), a technique which constructs and learns to exploit a game tree during play with random exploration weighted by past successes and failures, had previously been applied to the modern board game Settlers of Catan, and a real-time strategy (RTS) computer game. We may therefore consider that a sensible adaptation of tree-search techniques to this problem has already been tried, but we believe that the nature of the hidden information and interaction between players in "The Resistance" (Section 2.2) supports further investigation.

The "The Resistance" framework created during the Dagstuhl Seminar 12191, "Artificial and Computational Intelligence in Games", [Spronk et al., 2012] was later re-used for a competition hosted by AiGameDev.com, open to public submission of AI players, also known as "agents" or "bots", and results were announced at the Vienna Game/AI Conference 2012 organised by AiGameDev.com and the University of Vienna [AiGameDev.com Team, 2012]. Analysis of the methods used by a number of existing agents is carried out in Section 4.1, however none of the agents created for Dagstuhl, nor the 2012 competition, nor any of the others included with the framework make any notable attempt to apply techniques from classic board games - not even tree-search techniques. Furthermore, no other documents appear to have been published regarding AI for "The Resistance", so we must gather our own data for analysis, and turn to material on other games and research for inspiration. In the following sections we highlight several games of

relevance to "The Resistance" and the techniques explored in this paper, before moving on to the techniques themselves.

3.1.1 Go!

Go (known also as Igo, Weiqi, and Baduk) would not usually be considered a modern board game; it is a classic, 2-player, deterministic, perfect information game [Spronk et al., 2012] with a fixed initial setup (empty board), and has long been a subject of study for players and AI researchers alike.

However, when we compare Go to other classic board games such as Chess, while the rules appear simpler it is clearly distinguished by a larger state space [Wu and Baldi, 2007], game tree, and much higher branching factor (game trees are discussed in Section 3.4); each results from its larger board size (9x9 to 19x19), and the ability to place a piece almost anywhere on it during the early turns.

These problems render techniques often applied to Chess and other games, such as exhaustive tree-searches and alpha-beta pruning (an optimised tree-search algorithm which assumes opponents will always make the most profitable move), highly impractical. For this reason, much successful research has been conducted on Go which instead focusses on techniques that are also of interest when considering modern board games, such as Monte-Carlo Tree-Search [Gusmao and Raiko, 2012] . In fact, Go was the subject of the first successful application of Monte-Carlo Tree-Search techniques to human boardgames [Coulom, 2006]. Although the only apparent similarity between Go and "The Resistance" is their deceptive simplicity, Go literature serves as a rich resource for information on this technique, amongst others.

3.1.2 Settlers of Catan

Settlers of Catan, henceforth SoC, is another example of a modern board game, though significantly different and more complex than "The Resistance". The variety of initial setup is greater in SoC as the board itself is randomly assembled from 19 hexagonal tiles, forming a larger hexagon. Like "The Resistance", SoC allows for more than two players and imperfect information is a factor, though here it is the "development cards" held by a player which are unknown, rather than his long-term goal. Social interactions also differ, in allowing for the exchange of resources between players to mutual benefit, whereas the primary purpose of social interactions in "The Resistance" is to affect others' opinions of yourself, or each other. Finally, Soc introduces an additional stochastic element in the form of dice rolls. As with Go, similarities to "The Resistance" are limited, but SoC is likely the most researched example of a modern board game, and at least provides an example of techniques applied to games with more than two players.

SoC is older than "The Resistance" and has been the subject of substantial research, applying various AI approaches including Reinforcement Learning [Pfeiffer, 2004] (agents attempt to optimise for immediate rewards given after each action), Multi-agent Systems [Branca and Johansson, 2007; Saleem and Natiq, 2008] and, most recently, Monte-Carlo Tree-Search [Szita et al., 2009]. Several computer implementations of SoC exist, including JSettlers [Monin, 2013], which seems to be popular amongst researchers.

Of these papers, the Monte-Carlo Tree-Search approach [Szita et al., 2009] is the only one to claim results which are competitive against human opponents, though sufficient tests were not conducted to confirm statistical significance.

The Multi-agent Systems are also an interesting approach, where a main mediator agent evaluates the suggested actions of a collection of independent agents who each focus on different aspects of the state and action space. This precise approach may be more suited to a complex game such as SoC; perhaps a similar system could instead comprise bots who consider different overall approaches in a simpler game like "The Resistance", but it may be difficult to get these agents to compliment one another.

Finally, it is worth noting the comments on self-play and co-evolution made in [Pollack and Blair, 1998, p. 14], and referenced in our Reinforcement Learning example, [Pfeiffer, 2004, p. 2]: while this approach seems justified in SoC due to the greater variety of initial setup and stochastic elements introduced by the dice roll, application to "The Resistance" without further careful consideration is likely to result in poor exploration of the state space.

3.1.3 Poker

Poker may not usually be lumped into the loose category of modern board games as it is in [Spronk et al., 2012], but it does satisfy several of the criteria. Depending on the particular variety and house rules, it can be described as a non-deterministic game with imperfect information (both stemming from players' hidden hands which are drawn from a shuffled deck), and more than one opponent.

In fact Poker bears the greatest resemblance to "The Resistance" of all the games mentioned thus far due to the prominent role played by the hidden information; Go features no hidden information, while that of SoC is just one of many more, equally important mechanics.

To attempt to recount here the whole history of AI research in Poker would be a substantial and unnecessary endeavour. Of great interest however, are those who have attempted to apply Opponent Modelling, using statistical information to identify opponents' strategies and even to predict their hands [Billings et al., 2006; Ponsen et al., 2008; Southey et al., 2005]. Some examples even attempt to integrate Opponent Modelling and Monte-Carlo Tree Search techniques [Gerritsen, 2010; Ponsen et al., 2010; Schweizer et al., 2009; van der Kleij, 2010]. Our interest in this should be clear: since "The Resistance" has a smaller state-space than Poker (there are no playing cards and other players will have one of two primary objectives) and player roles are revealed at the end of each game (allowing for delayed analysis of opponent decisions with perfect information), it may be a simple task to apply this technique. The differing nature of Poker and "The Resistance" in the context of Opponent Modelling techniques is further discussed in Sections 3.6 and 3.7.

3.2 Expected Value Calculations

The expected value of a random variable or probability distribution is the average of all possible values, weighted by their probabilities. Note that as it is an average, the probability of a sample equalling the expected value may in fact be very low, or even zero.

For discrete random variables such as dice rolls and coin flips the calculation of the expected value is analogous to the mean average calculation of a sample: rather than multiply each value by its frequency and divide by the sample size before summing them, we simply multiply each value by its probability before summing them. In properly represented probabilities, the "sample size" - the total of the frequencies - is 1 (one), so the division is unnecessary. For continuous random variables such as temperature, rainfall, or the length of a fish, the expected value may be calculated as the integral of the distribution's probability density function (with infinite limits).

For example, a fair, six-sided dice, where the sides are valued 1-6 (one to six), will have an expected value of $1 \times 1/6 + 2 \times 1/6 + 3 \times 1/6 + 4 \times 1/6 + 5 \times 1/6 + 6 \times 1/6 = 3.5$. Alternatively, a weighted dice, preferring sixes, may have an expected value of $1 \times 1/10 + 2 \times 1/10 + 3 \times 1/10 + 4 \times 1/10 + 5 \times 1/10 + 6 \times 5/10 = 4.5$.

In games and decision theory the expected value of is great importance when selecting from actions with uncertain risk or reward, such as in the expectimax search, a minimax variant considering uncertainty, discussed in [Billings et al., 2006]). The significance of this should become more apparent during our discussion of Monte-Carlo Methods (Section 3.3) and Monte-Carlo Tree-Search (Section 3.5). The expected value also finds use in more involved statistical analysis such as Opponent Modelling techniques, as discussed in Section 3.6.

3.3 Monte-Carlo Methods

3.3.1 Overview

Monte-Carlo is a loose categorization of approaches and algorithms which can be used to obtain estimations and predictions for scientific and mathematical problems that may be infeasible, or even impossible to solve analytically or exhaustively. Monte-Carlo methods vary greatly based on their applications, but their defining feature is the use of repeated random sampling of a distribution, which may be, for example, a probability distribution, the domain of a function, or some more abstract representation of data, such as a two-dimensional graph.

The acquisition of each sample result may be referred to as an experiment, and according to the law of large numbers [Graham and Talay, 2013], the accuracy of the overall result should improve as the number of experiments is increased; specifically, the average of the results should approach the expected value (Section 3.2) of the distribution in question as the number of experiments approaches infinity. Sampling may be performed as either a physical experiment or by computer simulation, and while randomness is a key concept, some inputs may be controlled. If possible, the actual process or function of interest may be utilized in the experiment rather than some approximate simulation.

Monte-Carlo methods are versatile, and have found a great many uses in the fields of mathematics, natural science, and even real-time computer graphics.

Given method to attain a value from an unknown probability distribution d, for example, it is possible to estimate the expected value of the distribution itself through Monte-Carlo methodology. To estimate the expected value we would simple take n random values $(r_1, ..., r_n)$from the distribution and calculate the mean average as in Figure 1. When analysing some process samples should instead be taken from its domain used as inputs, and the results of each collated.

$$ev(d) \approx \frac{1}{n} \sum_{i=1}^{n} r_i$$

Figure 1: Expected value formula for Monte-Carlo methods.

As a simple, tangible example, consider a two-dimensional shape, s_1, of unknown dimensions. The area of this shape may be estimated by firstly placing it inside of a larger shape, s_2, of known area, then taking a large number of random points, uniformly distributed within the larger shape

[Hiob, 2007]. After evaluating which points fall within the smaller shape an estimate of its area can be calculated as in Figure 2. A similar methodology could be applied to calculate the volume of a three-dimensional shape, but this is less easy to envision as a physical experiment than the two-dimensional, where, for example, we might attempt to scatter grains of rice evenly over a diagram and count them by hand.

$$area(s_1) \approx \frac{\text{sampled points in } s_1}{\text{total sampled points}} \times area(s_2)$$

Figure 2: Monte-Carlo estimation of the area of a shape.

Simpler still, consider the flipping of a fair coin, that being a coin with equal chance of landing on heads or tails. If we allocate heads a value of 1 and tails a value of 0 (zero), with each having a probability of 0.5, then the expected value of a coin-flip can be expressed as $1.0 \times 0.5 + 0.0 \times 0.5 = 0.5$. The expected value, which is also the probability of landing a heads in this example, could be estimated by performing a large number of coin-flips either physically or by simple computer simulation (random number generation, not a simulation of the physics), and averaging the results. This is synonymous to the example in Figure 1.

3.3.2 In Board Games

[Brügmann, 1993] describes early attempts to utilise Monte-Carlo methods for AI in the game Go, but it seems that no great interest in such applications was stirred up until thirteen years later, when [Coulom, 2006] introduced Monte-Carlo Tree-Searches (Section 3.5) in the context of AI for that same game.

3.4 Tree-Search

3.4.1 Overview

Game trees are a method of representing games and similar decision processes as directed graphs. A complete game tree will contain all valid (according to the game's rules) states reachable from the initial state, and all actions available at each of these states; states are represented by nodes, while edges represent the actions which allow movement from one particular state to another. The actions branching from each individual state may represent the result of a player decision, a stochastic process, or some other process. Any complete playthrough, connecting the initial state (root node) to a terminal state (leaf node) may be represented by a decision tree, which is a subtree of the complete game tree.

Game trees may be used to describe a game's complexity, for example, simply by counting the number of different decision trees that are possible - that being the number of leaf nodes present. However, the complete tree is likely to be difficult to compute because we must consider the rules of the subject game. Furthermore, although the rules and initial setup restrict the states in the game tree to a subset of those in the full state space - the collection of all unique configurations of the variables which make up a state - equivalent states (identical, mirrored, rotated, etc.) accessible through various routes will be duplicated, so the game tree may be larger than the state space. We

may also consider the "branching factor" of the game tree, which is the number of children, or average number of children, branching from each node.

These measures of complexity are useful in intuitively identifying the limitations of tree-search techniques when employed by game-playing AI to select the best action at a given state. In simple, deterministic, two-player, zero-sum games, such as Noughts and Crosses, the subtree beneath the current state will always be fairly small, and therefore easy to search by properly applying evaluation rules like Minimax, which attempts to value a node by calculating the terminal value of the game, should each player choose actions which minimize his opponents maximum immediate gain, or maximize his own minimum immediate gain [Strong, 2011]. Such searches may vary in the extent to which they explore the depth (i.e. whether they search the tree to a terminal condition or not) and breadth (i.e. whether they explore every branching action at every node) of the tree, and may evaluate a decision tree, or partial decision tree, based on some heuristic function, the value of the terminating node, or another method. In order to optimise this process more advanced tree-search algorithms such as Alpha-Beta Pruning can be applied to limit exploration of poor moves, and the game tree itself can even be simplified [Burch and Holte, 2011], such as by combining equivalent states, or sequences of actions which lead to equivalent states .

3.4.2 In Board Games

But even with such optimisations, as we move into more complex games like Chess and Go, or those that introduce stochastic elements, hidden information, and greater numbers of players, such as Settlers of Catan, Poker, Backgammon, and "The Resistance", the application of such algorithms often becomes infeasible. In many cases this is simply due to the increased processing time required to search a larger tree, but in others the assumptions made by the algorithm do not hold, and significant alterations are required.

For example, a traditional Minimax algorithm is not a good fit for "The Resistance" for a number of reasons: While we may still consider "The Resistance" a zero-sum game if we look only at the victory/defeat of each side (Resistance and Spies) as a whole, the usual alternation of min and max nodes does not apply due to the additional players, and uneven, random division of roles. Furthermore, there are numerous turn types (selection, voting, mission), some of which reveal actions from multiple players simultaneously. A player also can never be assumed to be acting to minimise his opponent's maximum gain nor to maximize his own minimum gain. This is not only because the sub-goal of gaining his opponents' trust may interfere, but, most interestingly, because we may not know his role, and he may not know the roles of others. If we do not know a player's role, we do not know what maximising his own minimum gain means - we do not know if his decision node is a min or a max.

3.5 Monte-Carlo Tree-Search

3.5.1 Overview

Monte-Carlo Tree-Search (MCTS), as the name suggests, incorporates the idea of repeated random sampling, the core concept of Mote-Carlo methods, into a tree search algorithm which does not require a complete game tree in order to make informed decisions. Instead, the game tree is constructed over the course of play from the results of numerous playouts where actions at decision nodes for the controlled player are selected at random.

A typical MCTS algorithm can be described in essence by four steps (Figure 3): First, the "selection" of child nodes from the root until a non-terminal root node is located; second, "expansion" of the tree by creating one or more children, representing states reachable from the selected node; third, conduct a random playout, or "simulation", from one or more of the created nodes; finally, update the expected values of all the nodes above the selected node via "backpropagation" of the reward.

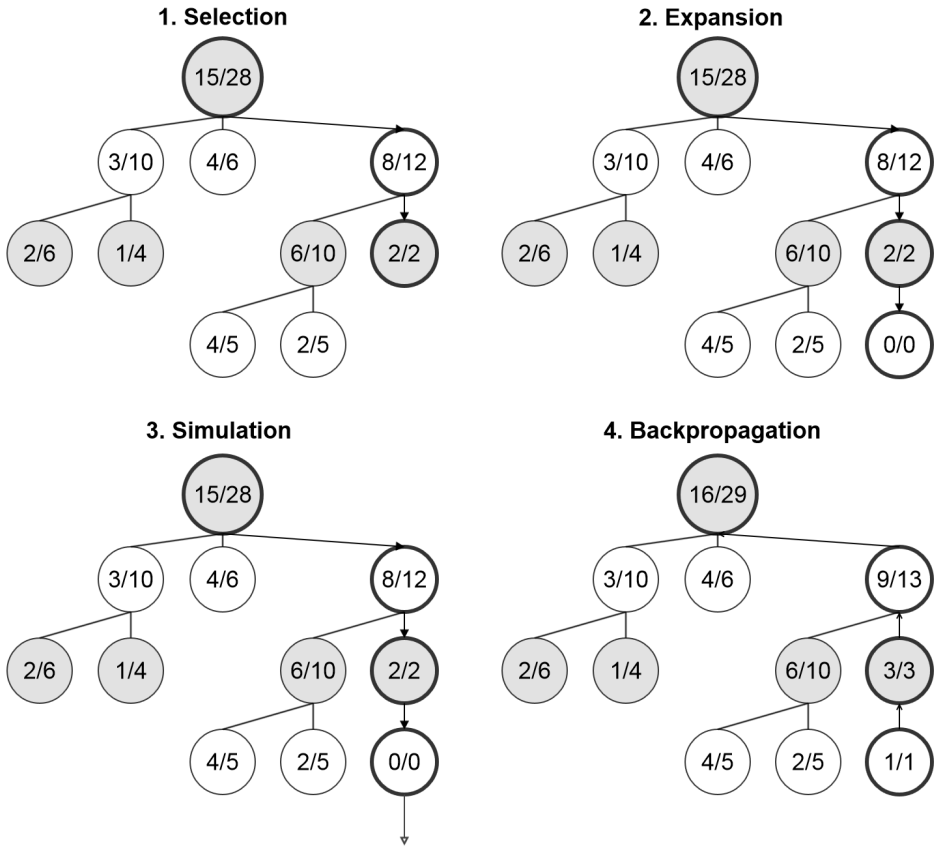

Figure 3: Four stages of Monte-Carlo Tree-Search

Playouts are full games played from a given node to a leaf node which represents a terminal state of the game. As in a regular game tree, terminal nodes are allocated some value, which in games is often 0 (zero) for a loss, or 1 for a win, but in MCTS this value is backpropogated to all nodes on the path from the terminal node to the root node. If each node maintains a sum s and count c of all values that reach it via this backpropogation, it is then trivial (s/c) to calculate a value which approaches the expected value of the state it represents as c approaches infinity. These values can then be used to decide which actions are most likely to be beneficial without searching the full depth and breadth of the tree, and without using any explicitly defined evaluation functions [Browne et al., 2012, p. 9]. The former is what makes MCTS useful for games with a large game tree, assuming our partial tree is based on sufficient exploration; the latter allows the application of MCTS to games without a developed theory, as the naive implementation requires no domain knowledge - only that the game's rules be defined. Domain knowledge and heuristics can, however,

be used to influence action selection [Szita et al., 2009].

Exploration of the tree is conducted by running multiple simulations to terminal states whenever the controlled player must make a decision, before making an actual decision based on the expected value of each action. During simulations actions at decision nodes for the controlled player are random. Therefore, early play strength is likely to be weak as evaluated moves we can choose from will be few, but as more games are played, we are able to make more informed decisions. Commonly successful actions will be explored disproportionately to commonly unsuccessful moves due to the actual action selection.

This can lead to the problem of local maxima[2] often encountered in computer learning, and so much effort has been expended in investigating exploration/exploitation algorithms which attempt to balance the exploitation of known good actions, and the exploration of less-explored actions. In-depth discussions of exploration/exploitation algorithms can be found in [Gusmao and Raiko, 2012] and [Gelly and Wang, 2006].

One possible complication of these simulations arises in action selection at decision nodes for other agents. We might simply select opponent moves at random, as we do our own. Another solution is to assume that the agent will take the action with the greatest expected value for itself, based on our own understanding of the game. In a zero-sum, two-player game, this would be trivial, as the branch with the lowest expected value for our player should hold the highest expected value for the other, but this is quickly complicated by the inclusion of multiple agents, imperfect information, stochastic elements, etc. To that end, our simulation of opponent behaviours may be improved by the integration of Opponent Modelling techniques (Section 3.6), as is discussed in Section 3.7.

3.5.2 In Board Games

Since their successful application to Go in 2006 [Coulom, 2006] MCTS techniques have been utilised to create successful AI players for a variety of games including classic, two-player, deterministic games like Go, more complex modern board games, and even video games [Chaslot et al., 2008]. The numerous publications following this which continue to focus on MCTS in Go attempt to analyse its success in order to extend it to other problems, and also to enhance the performance of MCTS in Go, often by employing exploration/exploitation algorithms.

[Gusmao and Raiko, 2012] suggests three main properties of Go which make it a good fit for MCTS (specifically, the UCT tree algorithm based on UCB, an exploration/exploitation algorithm), namely: the action space - the number of moves available at any decision node - is large and grows exponentially with the depth of the tree, rendering exhaustive searches infeasible; the evaluation of non-terminal states is difficult, but MCTS only has to evaluate terminal states; loops are forbidden by the rules and the game is deterministic, so a tree structure fits well. Each of these points could be made for "The Resistance" too, though it is the hidden information and dependence on other players' actions which renders the evaluation of states difficult, and the large action space (resulting mostly from the potential 10 combinations of players at each selection turn, and potential 5 selection, 5 voting turns per mission) is perhaps less extreme if modelled correctly.

[Gelly and Wang, 2006] points out that UCT is supposed to explore all possible actions at a particular state before considering exploitation via UCB, which renders it still unsuitable for games with a large action space, such as Go, though the paper goes on to describe a simple alteration to

[2] A local maxima is a point within the solution space of a problem which may represent the most optimal of a set of similar solutions, but is not necessarily the global maxima - that being the optimal general solution to a problem.

the algorithm that deals with this. This paper also highlights the potential parallelization of MCTS during the simulation phase, though we must take care as the behaviour of UCT may differ under parallelization.

As was previously mentioned (Section 3.1.2), Settlers of Catan too has seen a MCTS implementation [Szita et al., 2009]. This paper details the implementation of a working AI which was able to beat previous hand-coded agents, and provide a "reasonably strong" opponent for humans. Slight alterations were made to the game to make this first foray into MCTS SoC play a little easier (development cards were made visible to all players, and the agents will not participate in or initiate trades). What remains is still a strong demonstration of MCTS capabilities in a complex, multi-agent, stochastic environment with a variable initial state, and numerous methods of attaining victory, but, as a result of these modifications, does not support the idea of MCTS in a game heavily dependant on imperfect knowledge.

The SoC paper also provides excellent examples of how MCTS may be guided by domain knowledge through two subtly different methods. The first method injects the domain knowledge into the Monte-Carlo simulations by adding heuristics which affect the sampling of actions by weighting specific actions such as building a city or settlement, playing a development card, and so on. In practice this was seen to lower performance, though precise reasons for this were not attained. The second method injects the domain knowledge into the MCTS by allocating "virtual wins" to immediate actions which are thought to be a priority, effectively raising the expected value of those nodes without affecting the expected value of those above them. This was seen to raise the playing strength of the agent. These results might seem unintuitive, since one would assume that the former method, which merely guides the Monte-Carlo exploration, would be less detrimental than the latter, which actively lies about the ratio of successful branches beneath certain nodes, misguiding actual decisions.

The translation of these methods to the domain of "The Resistance" is not immediately apparent as these methods both force an emphasis on actions with predictable consequences which humans have pre-determined to be strong. In "The Resistance" the action-space is smaller, and for the vast majority of cases, unless both spies are known, the outcome of actions are highly dependant on the actions of the other players, so specifying domain knowledge in this manner would be difficult. Obvious decisions, such as not voting in approval of a team of three which does not contain yourself, or always sabotaging a mission if it'll result in immediate spy victory, can easily be hand-coded as special cases which overrule MCTS decisions.

A comparable subject would be poker, which also suffers from unpredictability in the outcome of actions. Most of the research on MCTS in Poker seems to involve some degree of Opponent Modelling, and so is further discussed in Section 3.7.

3.6 Opponent Modelling

3.6.1 Overview

Opponent Modelling (OM) refers to the application of statistical and probabilistic models to adversarial, multi-agent environments such as board games and card games in order to identify opponent strategies, preferences, biases, and goals. Usually the goal of this modelling is to enable another agent to select strategies which counter his opponents', or to otherwise exploit their weaknesses. Popular subjects of research into Opponent Modelling techniques are Real-Time Strategy (RTS) computer games, and Poker variants. Recently OM has seen application alongside Monte-Carlo

Tree-Search, again mostly in Poker (Section 3.7).

RTS examples such as [Schadd, 2007], [Avontuur, 2010] and [Fjell, 2012], which focus on the games "Total Annihilation: Spring", "Wargus", and "Starcraft" respectively, often limit their application of OM to the classification of players into pre-defined categories for which known counters may be deployed in response. Figure 4 shows how [Schadd, 2007] categorized players strategies hierarchically based on their OM data. Given an opponent with an aggressive, Tank-heavy strategy, for instance, we may deploy airplanes in response.

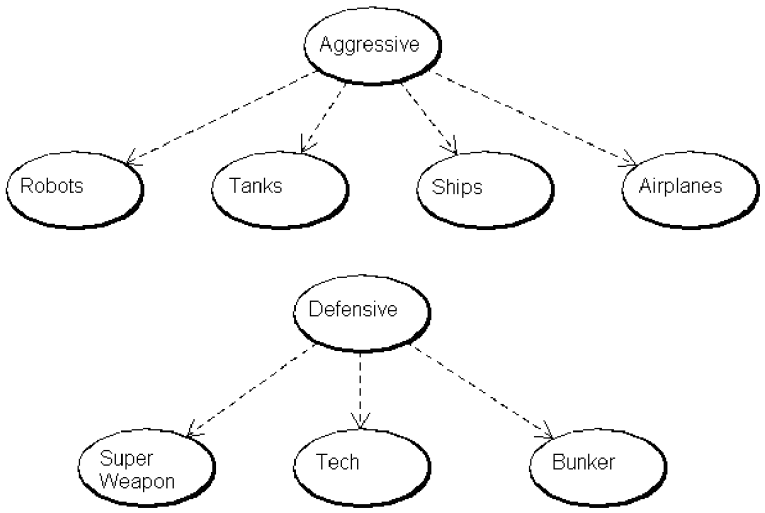

Figure 4: Hierarchical player categorization in "Total Annihilation: Spring", from [Schadd, 2007].

Beneath, and outside of this, OM implementations generally work in a similar manner: The opponent's state and/or actions taken at a state are observed, usually at discrete intervals (certain stages of the game in turn-based environments, or in the example of [Schadd, 2007], every five seconds). The variety of data recorded by this observation usually is tailored to the implementation and refined manually. Data may be immediately applied to modify some statistical representation of the opponent's past behaviour, or it may be cached in its raw form and applied later if necessary. For example, the frequency of a Rock-Paper-Scissors player selecting the scissors may be modified immediately after observation by increasing the variable representing scissor-selection count by one. Conversely, the frequency of a particular player of "The Resistance" voting in support of a team which does not contain any spies, when he himself is a spy, cannot be updated until the end of the current game (assuming the OM agent is not also a spy) when the role of each player is revealed. In this situation, we must instead record every team proposed and every vote of that player so that we can update our model when the game ends.

[Billings et al., 2006] provides a good example of what data might be recorded for OM in the context of Poker, during the discussion on why multiple abstraction granularities were used. Their data includes the frequency of opponent action selections at each decision node in the game tree, the frequency of an opponent making a particular number of bets and raises in a single hand, the total number of raises by both the opponent and the bot itself (presumably averaged over all encounters thus far), and the final pot size (again, presumably averaged over all encounters). As to the multiple abstraction levels, the stated reasoning was that this would allow for faster learning, based on a

larger number of inferences over a short time, and "allow us to generalize the observations we have made, and apply that knowledge to other related situations".

Another topic of significance touched on in [Billings et al., 2006] is what to do when we have yet to accumulate any observations relevant to an opponent or situation. Their solution was to initialise the model with default data derived similarly to a Nash equilibrium strategy - a situation in which if no other player changes their strategy no player can profit from changing his own, and so each player maintains a constant strategy. [Ponsen et al., 2008], referencing [Billings et al., 2006], approaches this problem in a different manner, by pre-learning a prior probability distribution (prior, for short) from a large number of observed games, and then learning some relational function for each opponent faced which adapts the prior to that particular opponent. This is similar to [Southey et al., 2005], though no direct reference is made. Here, a prior is derived for a simplified, tractable version of poker, "Leduc Hold 'Em", for proof of concept. Then, for the intractable problem of Texas Hold 'Em, "informed priors" defined by human experts were used.

[Southey et al., 2005] also covers several methods for computing a response to observed behaviour, though these may be limited only to Bayes-based OM implementations.

3.6.2 In Board Games

The prevalence of OM in Poker seems to have been necessitated by the prominence of partial observability in the game, which limits our ability to analyse the best moves given a particular state. Opponent Modelling is therefore used not only to try and predict a player's actions at a given state, but to try and make assumptions about the players' hands by comparing their current behaviour to their past behaviours. The tasks of inferring a player's hand strength in poker and inferring a player's role/objective in "The Resistance" would seem to be analogous, but it is important to note how the hidden information differs if we are to consider OM techniques in this context:

1. In "The Resistance", player's actions are not always fully observable as they are in Poker. For example, a sabotage in a three-man mission prevents us from recording with certainty the actions of each of those players at the current state, which is a significant inconvenience when trying to compare their behaviours to an opponent model.

2. In Poker a player may fold, whereafter his state for that entire hand will not be revealed, preventing proper recording of his past behaviours. In "The Resistance" roles are always revealed at the end of a game, so behaviours can be recorded retrospectively, just as when the cards are revealed in Poker. There is also the possibility of resolving the partial observability of actions described in point 1 by identifying which player was actually capable of sabotage, though some uncertainty remains if there were two spies on a team but only one sabotage.

3. In "The Resistance" players may have knowledge of the others' states which we do not. For example, if a Resistance player was on a mission with a single Spy who sabotaged, that Resistance player will know for certain the identity of one spy, but others may not. Modelling such intricacies should be possible, but is quite complicated and requires additional human consideration to design.

OM is the only technique discussed so far which has already been apply to some degree in "The Resistance", as is discussed in Section 4.1 (see Clymily, Magi, and Sceptic). Here, for the most part, OM appears to have been utilized to solve the categorization problem of whether each player

is a spy or resistance member. This is the most obvious application of OM to "The Resistance", and with only two categories, a simpler state, and less hidden information, should be a far simpler task than that of categorizing opponent strategies in RTS games. The complicating factor here over RTS games, however, is that we must also learn what behaviour is symptomatic of each category. Furthermore, while we can model the frequency of spies sabotaging missions they're on, for example, with near-perfect accuracy due to the end-game role reveal, the partial observability of sabotage actions in the current game complicates comparison to the model.

One bot from the Dagstuhl seminar also used OM to some extent: While having only fixed rules for which actions raise suspicion of a player being a spy in a particular game, Trusty records the accuracy of its suspicions when identities are revealed. In this way over numerous games the bot gets an impression of how well its fixed model works to identify particular opponents' roles, and can adjust its decisions based on this confidence. This technique could be applied to other OM bots, which would also allow us to graph the accuracy of their models over time. Clymily (Section 4.1.14) also gives some small indication of how we may apply OM techniques in "The Resistance" which seek to predict the actions of other players, as is the focus of the Poker papers discussed.

Although [Billings et al., 2006], [Southey et al., 2005] and [Ponsen et al., 2008] focus on heads-up (two-player) Poker, the principle of Opponent Modelling, that it concentrates on analysing opponent behaviours rather than the game itself, means that their methods will likely transfer more easily to larger games than game-theoretic solutions might. Of these papers [Billings et al., 2006] demonstrates its success most clearly, providing statistics from an experiment in which the subject, Vexbox, won every match played by a large margin, and claiming that it was even competitive against humans.

[Billings et al., 2006] also neatly highlights some of the major challenges in Poker, which map reasonably well to "The Resistance":

1. "Learning must be rapid (within 100 games, preferably fewer). Matches with human players do not last for many thousands of hands..." This is certainly true of "The Resistance"", and is a concern we have about the model of competition used in [AiGameDev.com Team, 2012], as is discussed in Section 4.4.

2. "Strong players change their playing style during a session; a fixed style is predictable and exploitable." This we are sceptical of in "The Resistance" due to the cooperative elements of the game. Change-up of strategies is likely to lead to confusion and mistrust, which only benefits the spies.

3. "There is only partial feedback on opponent decisions. When a player folds, their cards are not revealed."

With rapid adaptation in mind, we may consider employing multiple abstraction levels, as in [Billings et al., 2006]. The progression of granularities explored therein does not map well to "The Resistance" (due to the lack of bets, raises, pot, etc.), and this system could become hideously complicated if we were to begin considering voting style as spy, as resistance, as either role, different game stages, and so on. Perhaps there may be merit , for now, to exploring individual granularities though; for example, consider learning our spy and resistance categorization models from all players, rather than maintaining separate versions for each.

We may also consider introducing some sort of prior, as in [Southey et al., 2005] and [Ponsen et al., 2008]. These could take the form of standard behaviours expected from mid-level spy and

resistance players, which could be used to initialize the categorization models for new opponents. Since "The Resistance" has yet to undergo much game-theoretic analysis these priors would likely be pre-learned by observing a large number of games [Ponsen et al., 2008], or through self-play, as suggested in [Billings et al., 2006].

3.7 Integrating Opponent Modelling with Monte-Carlo Tree-Search

More recent research shows an interest in utilizing Opponent Modelling techniques alongside Monte-Carlo Tree-Search,especially in the domain of AI Poker. The key factor differentiating these methods from the OM methods discussed previously appears to be that rather than attempt to directly predict an opponent's play and hand strength so as to calculate an appropriate response, the models are applied to improve the accuracy of the Monte-Carlo simulation.

This is most evident in work on the game "The Octagon Theory" [Fernandes, 2012], which suggests that (especially in a game with such a large branching factor) "a fully-random playout is very unlikely to mirror an actual playout (i.e. a sequence of moves that would actually be performed by an intelligent player)". The problem then, is that not only would pure MCTS waste time running irrelevant simulations, but the results of those simulations will be considered with as much weight as those of more realistic simulations. This may intuitively be seen to be less of a problem in games with a smaller action space, such as Poker. Where this work focusses on taming one useful method (MCTS) through the use of another (OM), the works on Poker are purely about trying to combine the two useful methods.

Two clearly related works, [Ponsen et al., 2010] and [Gerritsen, 2010], expand upon the earlier work of [Ponsen et al., 2008] discussed as part of Section 3.6. Therein the matters of OM and approaches discussed are similar (pre-learned, generalized prior distributions and differentiating functions learned for each opponent), though the focus is on the suitability of MCTS to the problem of AI Poker, as well as the method and impact of guiding the simulation phase with information garnered from models of opponents' behaviours.

In these works the integration of OM is fairly non-intrusive. Models are used only to guide the simulation phase: first by weighting the random selection of opponents' private cards at the start of each simulation iteration based on the hand strength presumed by comparing their behaviour this hand to their models; and second, by weighting the random selection of actions at opponent decision nodes during a simulation based on their models, the visible game state, and the cards sampled for the current simulation. Decisions at the OM&MCTS bot's own nodes in the simulation are governed by the UCT tree algorithm. [Gerritsen, 2010] also introduces a balance parameter B, which governs the influence of OM and UCT decisions at opponent decision nodes, though it is left at $B = 1$ (no influence from UCT) for the duration of the experiment.

While MCTS is demonstrated to be effective and further enhanced by the guidance of OM systems in this manner, no indication is given of how the resulting combination of the two compares to typical OM bots such as those discussed in Section 3.6.

The final work we will discuss here is [Schweizer et al., 2009]. Similar methods appear to be employed here for allocating players' private cards and predicting actions, but the paper contains great detail on the implementation and considerations. Of particular interest is that the successes of the agent produced are attributed to its ability to exploit weak opponents. This ability would appear to be the result of aggressive policies adopted when an opponent's model passes certain expert-defined thresholds. While this may be less impressive than had the behaviour emerged by itself, it may yet having some bearing in "The Resistance".

4 Methodology

This section covers the details of our contributions: First, we provide descriptions and analysis of a number of existing agents, which gives an indication of the current state of AI in "The Resistance" and provides context for the rest of our work. Second, we highlight a number of expert rules that have been identified in existing agents. Third, we discuss the focus of this work with reference to the various possible directions covered in the literature review (Section 3). Next, we describe the tools we have implemented to aid in the analysis of agents' performance in "The Resistance", including the "burst" competition model, "suspicion inaccuracy" logging, graphing, and a few focussed bots. Finally, we provide an overview of our experiments, the results of which are discussed in the following sections.

4.1 Previous Agents For The Resistance

Along with the Python implementation of the game (and various related scripts), the public repository for the existing work (Section 1.2) also contains a large number of agents, including those created by the original workgroup, the 2012 Vienna competition entrants, and others implemented by Alex J. Champandard.

Not only does this provide a group of bots for experimentation that is perhaps more diverse than we could have created for ourselves, but by examining these we may determine what techniques have already been attempted, and gain some idea of what popular plays exist that are thought to be strong.

Before moving forwards, we will briefly describe the tournament procedure of the Vienna conference competition [AiGameDev.com Team, 2012] which a number of these agents were designed for, as we will ocassionally reference their performance therein.

The first round of the competition consisted of ranking each entrant by their win percentages after playing 12,000 games against combinations of the beginner and intermediate AIs RuleFollower, Jammer, Statistician, LogicalBot, and Bounder (described in the following sections). The second round saw the bots with the top five win percentages from the first round compete in against each other in every possible configuration. The final part of the competition was a direct elimination tournament including all entrants, beginners, and intermediate bots, where for each round 25,000 games were played, and the bot with the lowest win percentage was eliminated. In the final round of the tournament the remaining five bots are simply ranked by win percentage in games amongst themselves.

4.1.1 Paranoid, RandomBot, Deceiver, RuleFollower and Jammer (beginners.py)

These are simple, not necessarily competent agents implemented by Alex J. Champandard as examples for the Dagstuhl workgroup and Vienna conference. We will describe them briefly as some will appear in later discussions.

Paranoid appears to be a very weak player that will frequently vote against, and makes no attempt to select good teams. He will select a team consisting of himself and a random sample of other players, and will only vote positively for teams which he is a part of.

Hippie is almost the opposite of Paranoid. Again, his selected teams consist of himself and a random sample of other players, but he will vote positively for every proposed team.

RandomBot does what it says on the tin: all decisions are random, there is no attempt even to include himself in selected teams.

Deceiver features basic expert rules for avoiding discovery as a spy, which may also serve him well as resistance. He will always approve a mission on the fifth voting attempt, reject teams featuring both spies (if he is a spy), and reject three-man teams not featuring himself.

RuleFollower is almost identical to Deceiver, differing only in that when it is a spy it will approve of any mission featuring at least one spy, rather than reject those featuring both.

Jammer plays poorly as resistance, selecting completely at random and approving any proposed team. As a spy he behaves in opposition to what one would presume would be good practice: he selects both himself and the other spy for missions and, according to the comments, plays against common wisdom for sabotaging when both spies are on the team.

4.1.2 Trusty (trusty.py)

Trusty is not alone, even amongst the other Dagstuhl bots, in maintaining suspicion scores and behavioural statistics for each player over individual games. Bots such as this typically feature hand-coded conditions for calculating suspicion scores and making decisions based on them. Scores are wiped after each game as roles and seating configurations are reallocated. It could be said that such methods attempt to classify players into one of two categories (spy or resistance) based on fixed models implicitly defined by the conditions, statistics and suspicion score adjustments, similarly to techniques discussed in Section 3.6.

The Opponent Modelling is taken a little further in Trusty, however, which is the only bot from Dagstuhl to properly utilize global variables for maintaining data on players over all games it plays in a competition (the competition implementation creates a new instance of each bot for every game played, wiping local data). Data maintained in this manner is simply an average of how accurately the fixed model has identified each player (by name) over past games. When suspicion scores are calculated for players based on the fixed model, they are multiplied by this accuracy score. The accuracy score is calculated by accumulating the average difference between the suspicion score and identity (where $spy = 1.0$ and $resistance = 0.0$ at the end of each game played. A more advanced implementation of this may define multiple fixed models, more explicitly, maintain accuracy scores for each player, for each model, and base suspicion on a combination of all models and accuracies. This would then be comparable to methods employed for OM in RTS games (Section 3.6) [Avontuur, 2010; Fjell, 2012; Schadd, 2007].

Trusty also features a few of the "expert rules" which we have seen to be common in both simple and more advanced bots. For example: as a spy, Trusty will vote positively for any mission featuring at least one spy; as resistance, Trusty will vote negatively for any three-man mission not featuring himself (as this team is guaranteed to contain a spy). These rules usually take precedence over other systems, and often intuitively appear strong, which is supported by their popularity. We will reserve further discussion of such rules for a later section (4.2) where possible.

4.1.3 LogicalBot (aigd.py)

LogicalBot maintains a collection of "taboo" teams which is empty at the start of a game and accumulates any team which has been shown to contain at least one spy (teams that have previously failed a mission). LogicalBot will then vote negatively for any proposed team if there is a team in the taboo collection which is a subset of it (assuming certain expert rules do not take precedence).

Logic is also applied to determine when players are certainly spies, and teams containing such players will also be voted against.

When he is the leader, and a resistance member, LogicalBot will reuse the previous team if it was successful, though he will also ensure that he is part of the team himself. Otherwise he will attempt to assemble a team which contains himself and no known spies (note that knowing this, a player may, in some cases, exploit it).

This is a fairly simple implementation which depends upon logical deduction of spy identities, though part of it, the taboo teams list, is a fairly abstract manner of doing so. A similar mechanic can be seen in Bounder, Rebounder, and several other bots which focus on eliminating possible configurations of player roles, rather than eliminating unsafe team selections.

4.1.4 Statistician (aigd.py)

Statistician might be the first example of an active OM bot in "The Resistance", maintaining global statistics on the behaviour of each opponent bot (by name), including their frequency of voting for teams containing and not containing spies, of selecting teams containing spies, of selecting teams containing themselves (each as spy and resistance, separately), and of sabotaging missions (as a spy).

Statistician also maintains local statistics, which is merely a suspicion score for each player; although global statistics are recorded for all players across all games played, only those relating to voting behaviour are used to modify suspicion scores. There is no use of expert rules in this bot, only logic based on the suspicion scores.

Though this bot serves as a proof of concept, we are not convinced by the global statistics that are recorded. Frequencies such as voting for spy teams, voting for resistance teams, selection thereof, etc, will likely only be useful if the opponent modelled is at least moderately effective when playing resistance, since poor players' frequencies may not differ significantly between roles. For an extreme example, refer to Hippie and Jammer (Section 4.1.1), which always approve teams, leading to a 100% frequency of voting positively for spy teams in both roles.

4.1.5 Bounder (intermediate.py)

Bounder appears to have logic designed for playing resistance, but no separate cases for spy play beyond sabotaging missions (which he does at every opportunity). As seems fairly common in the example bots not utilized in the competition, this bot focusses on one mechanic and does not even include any expert rules. The focus here is logical deduction of spy identities by tracking all possible configurations of player roles and eliminating them based on the outcomes of missions. Similar techniques are used by a number of bots, and though they may be described instead as tracking all possible pairs of players who could be spies, there is no real distinction.

This implementation begins with two collections, "optimistic" and "pessimistic", each containing every possible unique permutation of [True, True, False, False], where True identifies a spy, and each index corresponds to that in a list of the other four players. Bounder always considers himself a resistance member. Configurations (permutations) are removed after each mission. The optimistic collection assumes that spies will sabotage at every opportunity, so it discards any configuration which would indicate that the number of spies on the mission did not match the number of sabotages. The pessimistic collection allows for the idea that not every spy on a mission must sabotage, so it only discards configurations which indicate the number of spies on the mission would be

less than the number of sabotages; these configurations are impossible. Therefore, optimistic will eliminate configurations quickly and inaccurately, possibly becoming empty, while pessimistic will eliminate them slowly, and safely, being reduced at minimum to the configuration which matches reality.

Selection and voting depends upon the configurations present in these collections, with pessimistic only being considered once optimistic is empty. The application of these two collections seems sensible here, as the safe logical deduction of pessimistic may operate far too slowly in some cases to be useful within three-to-five missions, but provides a solid backup when optimistic is shown to have been overzealous. We are uncertain, however, of how quickly the optimistic collection might empty after a spy team member allows his mission to succeed. Perhaps decisions should consider both collections, rather than only using pessimistic as a fallback.

4.1.6 Rebounder (rebounder.py)

Rebounder, a competition entrant written by Peter Cowling (upon whose design Alex J. Champandard's implementation of Bounder was based), introduces two additional configuration collections, logic for spy play, and minimal expert rules.

The "strong_configs" and "weak_configs" collections correspond directly to the optimistic and pessimistic collections of Bounder, respectively. The two new collections, "vote_configs" and "select_configs" are only utilized when Rebounder selects teams as the leader. vote_configs is updated after voting, discarding any configuration which would indicate that a spy has just voted against a team containing spies, for a team not containing spies, or for the fifth voting attempt of a mission. The select_configs collection is updated after team selection, discarding any collection which would indicate that the leader is not a spy, has selected a team not containing himself, or has selected a team containing zero or more than one spy. Teams selected as leader consist of players randomly sampled a configuration randomly drawn from an ordered, non-empty intersection of all four configuration collections (order: weak, strong, select, vote), where any intersection which would result in an empty collection is skipped.

While the elimination process of vote_configs - assuming spies will vote in approval of teams with spies on them - seems solid (if exploitable), that of select_configs is a little odd. Why would we eliminate any configuration where the leader is not a spy?

Nevertheless, Rebounder demonstrated strong performance in the Vienna conference competition, where it placed second in the first round, third in the second round, and fourth in the final of the direct elimination tournament.

4.1.7 Invalidator (invalidator.py)

Invalidator also works very similarly to Bounder, but includes a number of expert rules. The main difference though is that Invalidator does not remove configurations from its collection based on the outcome of missions, but instead maintains "invalidation" scores for each configuration which are adjusted based on the outcomes of selection, voting, and mission execution.

For example, after voting, any configuration which would indicate that the team contained a spy has its invalidation score raised by 1.0 for every player that the configuration considers to be a spy who also voted against the team.

4.1.8 GrumpyBot (grumpy.py)

Although Grumpy performed poorly in the Vienna competition, we believe it worth highlighting for its unique approach. Grumpy is the only bot to have attempted any sort of lookup technique, the poor performance in the competition giving no indication of the success or failure of this approach since the bot appears to have been submitted in an incomplete state.

Grumpy defines two strings of integers, and indexes these by the current mission, voting attempt, and game phase to decide what to do. The contents of these strings, however, do not seem compatible with the implementation, and it seems that more strings are required. It is unclear whether these strings were generated or hand-crafted. We are also concerned that the current mission, voting attempt, and game phase may be too simplistic a representation of the game state. Perhaps the intention was to try and compute an optimal set of moves for "The Resistance", or a Nash Equilibrium (previously mentioned in Section 3.6); this could be an interesting topic for future research.

4.1.9 PandSBot (PandSBot.py)

PandSBot was perhaps the strongest competitor in the Vienna competition, placing third in the rankings against beginners and intermediates, first in the competition amongst the top five from the first round, and first in the final round of the direct elimination tournament.

This is a bot which focusses on single-game suspicion tracking for players and pairs, with a few common expert rules thrown in for voting and sabotage. We will not attempt a full analysis of the suspicion score modification here, but are compelled to include PandSBot in this discussion to highlight how strong an AI may be achieved in "The Resistance" merely by examining player behaviours over the course of a single game. The Vienna competition results also help to demonstrate that an agent which performs well against advanced agents may not perform so strongly against simpler opponents.

4.1.10 GarboA (garboa.py)

GarboA works similarly to PandSBot, although it maintains suspicion scores only for individual players, and not pairs. Another apparent distinction when looking over the code is that PandSBot appears to make its suspicion score adjustments based on probabilistic calculations, while GarboA adds and subtracts arbitrary, hard-coded integers, such as $+25$ for each player on a team of three with one saboteur.

One or both of these differences likely explains why GarboA did not perform as strongly as PandSBot in the Vienna competition, although its performance was still quite strong, placing fourth in the third and fourth rounds, and narrowly missing the final round of the direct elimination tournament.

One interesting case in GarboA's voting method, utilizing its suspicion scores, is to vote against any mission which pairs itself with the second most suspicious player. If that player is indeed a spy, then this should be an effective way to avoid accruing suspicion in the eyes of other players - advantageous whichever role we're playing.

4.1.11 Bot5Players (dmq.py)

Yet another agent which depends only on expert rules and suspicion tracking based on players' behaviour in each, individual game, is Bot5Players. This agent performed similarly to GarboA in the Vienna competition, notably showing the strongest performance in the first round.

Bot5Players is most interesting, however, for its modular implementation: Expert rules and suspicion score adjustments are split into numerous "behaviour" objects, each responding to a particular circumstance. Behaviours are stored in a number lists, sorted by priority, and are looped over in this order when updated. For decision behaviours, a boolean is returned indicating whether the agent should take each behaviour's decision, or try the next one in the priority list.

This approach allows for easy experimentation with different rules and tactics by adding, removing, and changing the priority of behaviours. Behaviour inclusion and priority could even be optimized through automated procedures such as reinforcement learning [Pfeiffer, 2004], or co-evolution [Pollack and Blair, 1998].

4.1.12 Magi (mp.py)

Although Magi performed poorly in the first round of the Vienna competition it was one of three similar bots (Magi, ScepticBot, and Clymily) which reached the final round of the direct elimination tournament, where Magi placed last (fifth place). Each of these bots uses a combination of active opponent modelling over multiple games (with a separate model per opponent, per role), pairs/configuration tracking, and expert rules; the specifics of each implementation and the balance of their consideration varies.

Similarly to Bot5Players's modular behaviours, Magi defines behaviours tracked in its opponent model as a seperate objects, capable of maintaining frequencies of actions and comparing current behaviour to the current model. Behaviours tracked include:

1. Frequency of selecting teams featuring spies.
2. Frequency of selecting teams featuring subject.
3. Frequency of voting for teams featuring spies.
4. Frequency of voting for teams featuring both spies.
5. Frequency of voting for successful teams.
6. Frequency of voting for teams of three not featuring subject.
7. Frequency of sabotaging when the subject is a spy on a mission.

Magi maintains a probability (suspicion) score for each configuration of player roles based on the behaviour of all players compared to their models, similarly to how Invalidator maintains invalidation scores for each (Section 4.1.7).

4.1.13 ScepticBot (sceptic.py)

The second of three opponent modelling agents described here which made it to the final round of the direct elimination tournament in the Vienna competition. ScepticBot performed a little better than Magi in the first round of the, but failed to make it into the top five which competed in round two. However, in the final round of the direct elimination tournament, ScepticBot placed second.

ScepticBot differs from Magi in that it simply eliminates impossible configurations from its role configuration collection based upon mission sabotages, similarly to Bounder's pessimistic collection, and maintains individual suspicion scores for each opponent based on their own actions. An opponent's final suspicion score is then a result of the suspicion generated by its own actions, and the fraction of remaining configurations which indicate it is a spy, though suspicion from the bot's own actions have twice the weight of the so called spy ratio.

Behaviours tracked for ScepticBot's opponent models include:

1. Frequency of selecting teams with its partner on when the subject is a spy.
2. Frequency of voting against the first mission.
3. Frequency of voting for teams featuring spies.
4. Frequency of voting for teams of three not featuring subject.
5. Frequency of sabotaging when the subject is a spy on a mission.

4.1.14 Clymily (clymily.py)

Clymily is perhaps the most complicated of the OM bots which performed well in the Vienna competition, featuring not only OM, configuration tracking, and expert rules, but also a subsumption framework[3], and, it appears, taboo team tracking similar to that of LogicalBot. Clymily also saves its models for each player in external files so that it may utilize them not only in later games in a competition, but also in future competitions (though this may not be allowed, or useful since the same opponents may not be encountered again). Clymily also appears to have a significant number of functions for predicting future opponent actions based on its models of their previous behaviour.

Although not all are used, there are some interesting functions to be found within Clymily's code, indicating potential tactics we have not seen considered elsewhere. For example, the function "_inHammerPos" returns a boolean indicating whether, if all votes fail until Clymily becomes leader, it will be in the position of proposing the fifth team, which any reasonable resistance member (or cautious spy) will be forced to accept. Clymily does not make use of this, and while it is unlikely to be useful knowledge during the first or second voting attempt, it might be a consideration for the third or fourth.

One of these functions which is used, "_myVoteIrrelevant", estimates, based on the opponent models, the probability that Clymily's vote will not sway the outcome of a voting round due to the manner in which other bots are expected to vote. This is used to allow Clymily, when a spy, to approve teams it otherwise would not, in order to avoid incriminating itself.

[3] A subsumption architecture is one in which the agent makes decisions based upon hierarchy of behaviours, similarly to the multi-agent systems referenced in Section 3.1.2 [Branca and Johansson, 2007; Saleem and Natiq, 2008].

4.1.15 KreuterBot (dkreuter.py)

KreuterBot, the only other OM entrant to the Vienna conference competition performed poorly by comparison to the others, placing behind Magi in the first round, and being eliminated third in the direct elimination tournament. It appears that KreuterBot attempts use its opponent models and some expert rules to predict opponent actions, makes decisions based on these predictions, and maintains statistics for their accuracy. Unlike Magi, ScepticBot, and Clymily, KreuterBot does not track possible role configurations.

4.2 Common Expert Rules

We have found that many existing agents employ a number of expert rules, particularly in the voting stages. These specify what actions to take given certain conditions and are usually based on some intuitive, human reasoning about what appears suspicious, and what can never be, or always will be profitable. Even in bots with more complex systems in place, such as suspicion scoring, configuration tracking, logical deduction, or opponent modelling, such rules are often considered first and just one is enough to make a decision. The order of these conditions is therefore important, and may vary between agents. This importance is highlighted in "Bot5Playes" (dmq.py), which maintains priority lists for the rules tested at each game phase. Here we will discuss some of the more common and/or interesting expert rules employed.

As leader, most agents ensure that they select themselves as part of a mission team. For a team of two, assuming the leader is a resistance member, this gives an improved probability of selecting a team with no spies on it ($2/4 = 0.5$, rather than $3/5 \times 2/4 = 0.3$). For a team of three, however, assuming the leader is a resistance member, this is the only way to guarantee there is no spy on the mission. It is therefore very risky for resistance members to approve a three-man team not featuring themselves, and doing so as a spy would raise a lot of suspicion amongst observant players; a large number of agents (GarboA, Magi, PandSBot, ScepticBot...) will reject any such team.

In the rules of "The Resistance" (Section 2.1) after five failed voting attempts in a row victory is immediately awarded to the spies. Therefore there is no advantage for a resistance player in rejecting the fifth voting attempt for a mission under any circumstances. Many bots (Deceiver, Bot5Players, ScepticBot...) embrace this as an early approval case in voting, regardless of role. Others do so but still vote negatively if they are spies (Rebounder, PandSBot, RuleFollower...). We consider the latter of highly questionable merit, as given three competent resistance members the vote can never fail and such actions would appear highly incriminating.

There is no apparent advantage to players in either role rejecting the first voting attempt for the first mission either, since no information is known (to the resistance, at least) about player roles. We would suggest that even subsequent voting attempts for the first mission should also be approved, since any information gleaned from voting this far is likely down to incompetence rather than malice. At least two agents (GarboA, ScepticBot) respect this policy.

Several policies can be seen for sabotaging when both spies are on the team. Some bots simply will not sabotage in this situation (GarboA, Magi, Invalidator), others sabotage based on who is the leader (Opeth), while others do so based on table position index (Rebounder, SoreLooser [sic]). In a five-player game, there is no advantage for spies in double-sabotaging a mission, and doing so would aid more advanced bots in narrowing down spy candidates, so it should be avoided. However, there can be no global best policy for determining whether to sabotage in this situation as it is dependant on your partner's policy. We might prefer the leadership policy as the saboteur

may then differ each time, but to other players this difference will not be apparent.

Spies may also attempt to avoid such situations by selecting teams composed of themselves and non-spy players, and by rejecting missions featuring both spy players, but for the most part existing agents simply accept teams featuring at least one spy (Deceiver, Bot5Players, GarboA, Magi, PandSBot...).

When spies have already sabotaged two missions there is no disadvantage in sabotaging the third mission when possible as this would win the game for them. Several bots have conditionals for this (Bot5Players, Invalidator, ScepticBot...). In this position, spies may also opt to approve any mission with at least one spy on the team, though these players may be rendered somewhat suspicious by their enthusiasm.

4.3 Focus

In Section 3 we discussed a number of other board games, techniques which have been applied to them in previous research, and how these games and techniques relate to "The Resistance". In Section 4.1 we described the approaches of a number of existing AI agents which play "The Resistance". This puts us at last in a good position to consider possible directions for future work, including our own.

First, as mentioned in Section 3.1, nobody has yet, to our knowledge, attempted to apply tree-search techniques (Section 3.4) to this game. We previously dismissed simple algorithms such as Minimax due to the complexities incurred by the multiple game phases, large number of players with unknown objectives, and high branching factor. However, a related technique, Monte-Carlo Tree-Search (MCTS) 3.5, has been shown to be effective in a number of related games including Go [Gusmao and Raiko, 2012], Settlers of Catan [Szita et al., 2009], and Poker variants.

While it may be tempting to begin with research into naive MCTS implementations for "The Resistance", we believe that proper injection of domain knowledge, as discussed in [Szita et al., 2009] will be critical in a game so dependant upon the identity of your fellow players, as evidenced by the strong performance of bots such as Clymily, Sceptic, and PandSBot, which each apply numerous techniques to try and achieve this. While [Szita et al., 2009] identifies a good method for injecting domain knowledge into MCTS based on expert rules, most Poker papers featuring MCTS also involve Opponent Modelling (OM) techniques (Section 3.6), and should provide a good reference for how such dynamic measurements, even suspicion scores and likelihood of a player being a spy based on configuration tracking, may be used to influence MCTS. Nevertheless, it may yet be too early to begin exploring the integration of MCTS and OM in "The Resistance", as each method is somewhat complex, and neither has been sufficiently, individually explored in this context.

Further investigation into OM techniques is supported by the success of the existing agents Clymily, Sceptic, and Magi. However, each of these agents also employs expert rules, configuration tracking, and deductive logic specific to "The Resistance" (in fact, specific to the simplest, five-player variant), and so it is unclear how much OM contributes to their strength. The fact that the strongest bot in the Vienna competition, PandSBot, performed so well using only suspicion scoring for individual players and role configurations over individual games, similarly to the other aspects of the OM agents, gives us further reason to be sceptical of the effectiveness of OM in "The Resistance". Comparisons of "The Resistance" to Poker, where OM seems to be a popular and effective technique, are also mired by differences in the natures of their imperfect information. Still, we believe that there is room for further development in this area by examining the different types

of behaviour we might track in opponent models, considering the fact that partial observability of actions complicates the fitting of players to models, and by understanding that opponents too are operating with imperfect information.

Finally, we wish to highlight the possibility of employing computer learning techniques to optimise the decision making, suspicion score modifications, and behaviours tracked by OM agents through techniques such as reinforcement learning [Pfeiffer, 2004] and co-evolution [Pollack and Blair, 1998]. Bots such as Bot5Players and Magi give good examples of modular structures which could be used in implementations of these techniques. As previously suggested (Section 4.1.8), computer learning could also be applied to a lookup implementation, similar to GrumpyBot's, in order to try and estimate an optimal strategy or Nash Equilibrium. Consideration must be made, however, to ensure adequate exploration of the state space in co-evolution, as highlighted in Section 3.1.2.

This is likely to be the first formal work on "The Resistance", and aims to provide a general overview of the game, the current state of AI research with regards to it, and related games and techniques, in order to help guide future development. Therefore, we avoid straying into new territory such as MCTS or computer learning techniques, for which there is little existing knowledge upon which to build. Instead we focus on implementing tools which we believe will be useful in examining the performance of agents in the existing framework, and use these to investigate the value of OM in "The Resistance", a technique which for which there is already some, uncertain, indication of worth.

4.4 Competition Model

The agents Clymily, Sceptic, Magi and KreuterBot, described in Sections 4.1.14, 4.1.13 and 4.1.12 respectively, each apply Opponent Modelling techniques (Section 3.6) wherein the behaviour of individual opponents is modelled over the course of all of a competition's games in which they themselves participate. In fact, Clymily stores this data in a file so that it can be reloaded and utilized even in a separate competition execution. Typically such systems require a long learning phase before they gain an advantage. In [Billings et al., 2006, p. 10] the authors describe the performance of the OM, Tree-Search based bot, Vexbot, in two-player Limit Texas Hold'em Poker. To learn to exploit the previous best program, Sparbot, they claim, required several thousand hands. They also say that this was much longer than was needed for lesser bots, and that in matches against human players a significant increase in performance could consistently be seen after only 200-400 hands. How this translates to The Resistance is as yet unclear.

4.4.1 Original Competition Model

In previous work on The Resistance a given collection of bots would compete in a competition usually consisting of ten-thousand or so games. This is not necessarily to allow for online learning techniques such as these, which may be less effective with a smaller number of games, but rather for statistical significance. With smaller numbers the resultant statistics could vary more from run to run based on the random behaviour of one or more competitors, and may also be unfairly affected by the inability to run a balanced, comprehensive set of permutations. In fact, even in situations where the number of games requested is greater than the number required for a comprehensive set of permutations, but not a multiple of it, the statistics will be affected by those (randomly selected) permutations which are processed more than once, though to a lesser degree than were some omit-

ted entirely. Therefore, the first modification made to the original competition code is simply to output information about the requested competition which may aid in refining experiments. This information includes the number of game required to cover all unique permutations possible with a given set of players, the fraction (or multiple) of these that can be played in the requested number of games, and warnings, such as when this fraction is not a whole number.

The number of unique permutations p possible in a competition is a function of the number of competitors k and the number of players per game g:

$$p = \frac{k!}{(k-g)!} \times \frac{g!}{s!r!}$$

Figure 5: General formula for the number of "The Resistance" game permutations given any number of competitors and game size.

where the number of Spies s and Resistance r are determined by g as per the game's rules. Since we only deal with five-player games, where $s = 2$ and $r = 3$, we can simplify this by evaluating the term relating to unique role (Spy or Resistance) permutations per seating permutation:

$$p = \frac{k!}{(k-5)!} \times 10$$

Figure 6: Formula for the number of "The Resistance" gane permutations given any number of competitors and game size of 5.

So, for example, a competition of 3,600 games with 5 competitors would consist of each possible one of the 1,200 seating and role permutations 3 times; a competition of 3,600 games with 6 competitors would consist of only half of the possible 7,200 permutations of seating and roles.

4.4.2 Burst Competition Model

Should we seek in the future to develop bots which are competent against human players, the original competition model provides too lenient an environment by allowing for behaviours which rely on experience with specific opponents accrued over the course of thousands of games to perform well. An individual human is unlikely to willingly participate in enough games for such a bot to learn to exploit his behaviour, and even then he may arbitrarily change his behaviour at any moment. However, if we reduce the number of games played significantly enough to plausibly accommodate human play, say 10-20 games, we lose statistical significance due to non-deterministic bot behaviours, and due to running a set of games not representative of the possible permutations with those competitors.

Therefore, we adjust the competition model in a manner which we believe will encourage the development of rapidly adapting behaviours or strong, generalized rule-sets, while still maintaining statistical significance by running a large number of games. While human participation in this model remains impractical, it should give a stronger indication of which bots are most suited for human opponents.

The core of these adjustments is the periodic obfuscation of bot names. In burstcompetition.py burstSize b can be supplied as an argument, as can the requested number of games. Every b games, a bot receives a new, unique integer identifier by which other bots can recognise it [4]. The bot names previously used to track behaviours over all games are no longer available, so the learning phase for individual opponent behaviours is restricted to b games.

In this way we are still able to run a high, statistically significant set of games and ensure that all possible permutations are considered equally, though the enumeration differs slightly. The permutations of each combination of competitors are shuffled before being formed into bursts to ensure that there is no pattern in each burst, and the collection of all bursts for all combinations of competitors is shuffled before processing to ensure there is no pattern there either. In this way, every subsequent b games are guaranteed to feature the same bots, but nothing is guaranteed beyond that.

To better represent the enumeration of these games, the formula for Figure 5 can be restated as in Figure 7. Note that the last term, representing unique role permutations per seating, remains the same, and the other terms are easily seen to be equivalent, so the simplification when g is known does not change.

$$p = \frac{k!}{g!(k-g)!} \times g! \times \frac{g!}{s!r!}$$

Figure 7: Formulaic representation of the enumeration of "The Resistance" game permutations in the "burst" competition model.

4.5 Suspicion Inaccuracy

In Section 4.1.2 we discussed the technique of estimating the accuracy with which an opponent model matches an opponent's behaviour by comparing resultant suspicion scores to players' actual identities when these are revealed at the end of each game.

We wish to explore this as a method of assessing bot performance separately from the victory conditions of the game, and of examining the effectiveness of active opponent modelling systems in "The Resistance". While this method is not exclusively restricted to bots which conduct explicit, active modelling of opponents, it requires that the subject maintains suspicion scores for its opponents within some finite range (most simply 0.0 to 1.0). If l is the lower bound of this range, and u is the upper, suspicion inaccuracy[5] i can then be calculated as $i = \frac{suspicion-l}{u-l}$ for resistance members, and $i = 1 - \frac{suspicion-l}{u-l}$ for spies.

We were able to identify three existing bots which meet this requirement: Magi, ScepticBot, and Clymily - all strong competitors from the Vienna competition, all OM agents, and all employing similar logical deduction techniques via tracking potential role configurations. Note, that the suspicion scores maintained by these bots are a result of both the OM and logical deduction, but do not give any bearing on the expert rules (Section 4.2) employed. Later (Section 4.6.2) we discuss our implementation of a pure OM agent, Stalker, which also meets these requirements.

[4] A pseudonym generator is also implemented to aid human developers and players in identifying bots, but this has no impact on bot performance.

[5] Our decision to observe inaccuracy rather than accuracy is arbitrary, unimportant, and a matter of preference. It simply means that 0.0 is the best case, rather than 1.0.

This method should liberate bots, at least somewhat, from being judged by the performance of their comrades, as the observation and logical deduction capability of an individual is not so closely tied to its teammates' as the overall play capability is (Section 2.2). Incidentally, it may also enable investigation into the importance of each round in terms of accuracy gained accrued, though this may be skewed by reckless spy behaviour when early victories are available in missions three and four.

4.5.1 Logging

To facilitate recording of inaccuracies during play we have implemented a collection of classes (suspAcc.py) and modifications of our subject bots (bots\1\confect-mod\) which utilize these. Each observer bot records its suspicion scores after each mission, after any adjustments based on observation have been applied. At the end of a game, once roles are revealed, these are converted to inaccuracies and logged to a file. A completed log file will contain a record for every game played in a single competition, including a list of other players' names, those who are spies, the suspicion and inaccuracy scores for each player after each mission in the game. Figure 8 gives an example of data recorded over several games.

```
!
Stalker suspicion inaccuracy log
-
others: 1-Bot5Players.2-PandSBot.3-GarboA.4-Rebounder
spies: 3-GarboA.4-Rebounder
n:Bot5Players;s:0.472222;r:0;i:0.472222;
n:Bot5Players;s:0.479167;r:0;i:0.479167;
n:Bot5Players;s:0.483333;r:0;i:0.483333;
,
n:PandSBot;s:0.750000;r:0;i:0.750000;
n:PandSBot;s:0.750000;r:0;i:0.750000;
n:PandSBot;s:0.650000;r:0;i:0.650000;
,
n:GarboA;s:0.666667;r:1;i:0.333333;
n:GarboA;s:0.694444;r:1;i:0.305556;
n:GarboA;s:0.708333;r:1;i:0.291667;
,
n:Rebounder;s:0.458333;r:1;i:0.541667;
n:Rebounder;s:0.458333;r:1;i:0.541667;
n:Rebounder;s:0.450000;r:1;i:0.550000;
-
others: 1-PandSBot.2-Rebounder.3-Bot5Players.5-GarboA
spies: 2-Rebounder.4-Stalker
SPYGAME
-
```

Figure 8: Example suspicion inaccuracy score logging output from suspAcc.py.

We ignore missions where the observer is a spy as some systems may immediately set perfectly accurate suspicion scores for these missions, while others may not bother, and others may proceed as if they were a resistance member, perhaps using these scores to aid in their deception; this would skew results. The mission is still logged in sequence, but it is marked as a spy game, and statistics are not recorded.

The implementation is simple and suboptimal, including some duplicate data such as repetitions of player names. A file generated by a single observer for a competition of five bots over

12,000 games is approximately five megabytes. Further optimization is neglected in order to avoid introducing subtle errors that could ruin our results, but much mileage could be gained by optimising the data. Storing the inaccuracies is, in fact, worthless, as these can be recalculated from the suspicion scores and the players' roles.

Finally, name deobfuscation was implemented to allow this system to work with our "burst" competition model (Section 4.4.2).

4.5.2 Graphing

Browsing over 12,000 games' worth of numerical output is probably the most tedious and least intuitive method of examining the output from the logging described above.

To aid in examining these results we process them into graph form. What was seen to be the easiest, most flexible and reusable method of doing so was to write a python module (suspAccPlotter.py), which utilizes matplotlib[6]. Our module was intended to allow for extension to produce different types of graph, though we only got as far as implementing one, an example of which is given in Figure 9.

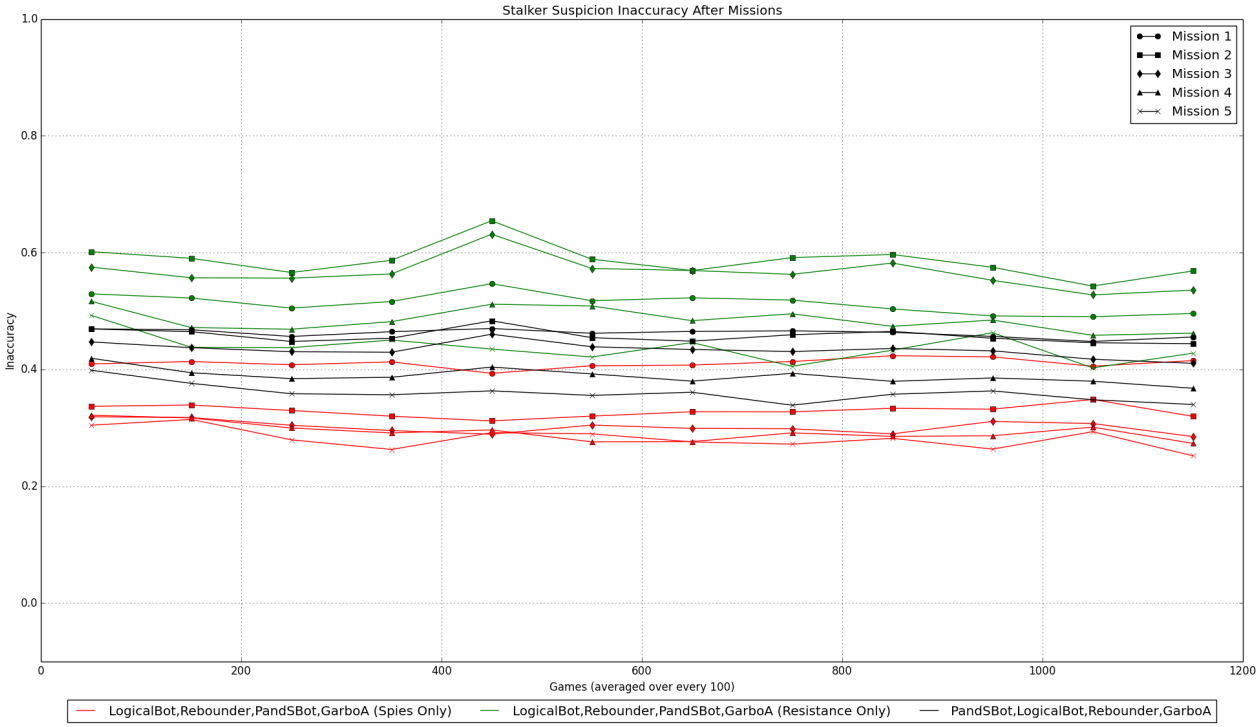

Figure 9: Example graph output from suspAccPlotter.py.

Legends indicate the missions (denoted by marker type) and average group (denoted by line and marker colour) to which each plot corresponds. Black is reserved for the average over all players, which is produced by default. Parameters allow the user to specify groups of players to average over (we can specify a group containing an individual), groups to average over only spy

[6]matplotlib is a 2D plotting library, available, at the time of writing, online at `http://matplotlib.org/`

or resistance games for, and the number of games over which to average . Points are plotted at the centre of the set of games averaged (x-axis).

Note that the current plotting implementation is only designed to handle competitions with five competitors, as the competitors in each game of such a competition will be identical.

4.6 Our bots

4.6.1 Ruru (Expert Rules Only)

While getting to grips with the framework, prior to much of the investigation discussed thus far, we began work on a bot which makes decisions purely based on expert rules. A little extra effort has since been put in to extend this implementation with additional expert rules found in existing bot implementations (Section 4.2, and variants based on our own insights. Logical deduction carried out by Ruru is minimal: If it is a resistance member, both players on a team of two featuring two saboteurs are certainly spies, and future teams containing these players will be rejected, assuming non of the expert rules are triggered.

Had we opted to conduct a detailed investigation into the power of expert rules in "The Resistance", it may have been beneficial to implement a prioritised behaviour structure similar to that of PandSBot, which would have made it practical to gather data on different configurations. Nevertheless, running Ruru against beginners and more advanced opponents may give some insight into the general effectiveness of expert rules.

The expert rules employed by Ruru, in order of consideration, are as follows:

1. (Selection) Include itself when selecting teams of three.
2. (Selection) (Spy) Include one spy when selecting teams (always self).
3. (Voting) Approve missions on the fifth voting attempt.
4. (Voting) Approve any voting attempt in the first mission.
5. (Voting) Approve missions where self is the leader.
6. (Voting) (Spy) Approve missions with at least one spy if spies only need one mission to win.
7. (Voting) Reject teams of three not featuring self.
8. (Voting) (Spy) Reject missions where the entire team is spies.
9. (Voting) (Spy) Reject missions with zero or both spies on the team.
10. (Voting) (Spy) Approve missions with at least one spy on the team.
11. (Voting) Reject missions with known spies on the team.
12. (Voting) Approve any mission.
13. (Sabotage) (Spy) Sabotage any mission which will win the game for spies.
14. (Sabotage) (Spy) Don't sabotage if every player on the mission is a spy.

15. (Sabotage) (Spy) Sabotage any mission where self is the leader.

16. (Sabotage) (Spy) Sabotage if self is the only spy on the mission.

4.6.2 Stalker (Opponent Modelling)

In Sections 4.1 and 4.3 we described how the impact of OM on the performance of several existing bots, namely Magi, Clymily, and ScepticBot, is unclear, due to the application of various other powerful techniques such as role configuration tracking. In Section 4.5 we pointed out how our implementation of suspicion inaccuracy logging will also fail to capture the performance purely of OM, as suspicion scores for these bots result from a combination of the applied techniques.

It was easier to build our own OM bot from the ground up than to try and strip all of the non-OM components from the unfamiliar, complex code of one of the existing implementations. Therefore, we created Stalker (Section 4.6.2), which is capable of reporting individual suspicion scores, per opponent, based only on how well they fit models built from their previous behaviour as a spy or resistance member.

We believe that the problem of acting upon the reported suspicion scores should be viewed as one separate from their calculation, as poor performance given certain information is not necessarily indicative of the quality of that information. In this work we make no attempt to investigate responses to OM data, only to investigate the accuracy of the data itself. Therefore, Stalker simply inherits Ruru's implementation for decision making, which ignores the OM data, and we assess Stalker's performance based purely on the reported suspicion scores.

Stalker is implemented in a modular fashion, similar to that of Magi, where each behaviour modelled is represented by a frequency, and an instance of a class capable of updating that frequency based on data from a game just completed (with roles revealed), also capable of evaluating how well a player's behaviour, in data representing the current game thus far executed, matches that frequency. Stalker maintains a copy of each frequency and behaviour class for each behaviour, in each role model (spy or resistance member), for each player (by bot name).

Fitting of a player's behaviour in the current game to a particular frequency is calculated by sampling $f = frequency$ for each action in the game in line with the behaviour (complimentary), $f = 1.0 - frequency$ for each action in the game opposing the behaviour (contradictory), and then averaging these samples. This is a simple, generalized method which seems to work for a large number of behaviours. Fitting of a player's behaviour to an entire model (e.g. spy or resistance member model, consisting of many behaviours) can then be calculated as the average of the fittings for each behaviour frequency. Finally, a player's suspicion score may be calculated as the sum of the fitting to the spy model s and one minus the fitting to the resistance model r, applying coefficients to each term such that they add up to 1.0, or whatever the preferred range. Stalker simply weights each model 0.5, as in Figure 10.

$$suspicion = ((1.0 - r) \times 0.5) + (s \times 0.5))$$

Figure 10: Suspicion calculation from Stalker bot, based on fitting scores for spy and resistance member models.

During investigation of the behaviours tracked by existing OM bots and implementation of

our own, several distinct categories of behaviour became apparent which may require differing approaches for modelling, fitting, and may be problematic:

1. Perfect Information Behaviours: Often relating to expert rules, these are behaviours we can fully observe, and model with certainty, where the subject has full knowledge of the parameters considered, and full control over the outcome of his action. An example might be the frequency with which an agent selects teams featuring itself, votes for teams featuring itself, or votes for three-man teams not including itself.

2. Competence Based ("Sketchy") Behaviours: These are behaviours where, if it is a resistance member, the subject must make a decision without knowing the relevant parameters. These behaviours are often related to the success or failure of missions, for example, the frequency with which an agent votes for a team which then fails a mission. Fitting to such behaviours may be unproductive in early rounds, or when faced with bots which do not play a competent resistance member, as actions which fit the spy model may instead be the result of lack of information or playing ability. At least, however, the resistance model should come to reflect this, countering over-suspicion of poor players so that modelling such behaviours is unlikely to be detrimental.

3. Partially Observable Action Behaviours: These are behaviours which we cannot fully observe during the course of a game; specifically, the act of sabotaging or not sabotaging a mission. Although we can easily identify who could not have sabotaged a mission at the end of a game when identities are known, allowing us to construct accurate models of opponent behaviour, comparison of behaviour in the current game to these models is not trivial, as there is uncertainty as to who was responsible.

4. Twice Partially Observable Action Behaviours: This is not a significantly different consideration than category 3, at least in five player games. This describes the case where there are two spies and a resistance member on a mission, and one of the spies sabotages. Unless we are one of those spies, we cannot, even in retrospect with roles revealed at the end of the game, know for certain which spy sabotaged the mission.

The last two categories are where we encounter the partial observability of actions, one of the features which distinguishes the challenge of "The Resistance" from that of Poker. To clarify why this is such a problem, we shall consider it in the context of our simple, generalized behaviour fitting model:

Assume the frequency of a spy agent sabotaging will tend to be 1.0, or thereabouts (we need not track frequency of sabotaging as a resistance member, as the game rules do not allow resistance members to sabotage). Given a three-man mission, how then do we record each team member's action as complimentary or contradictory to this behaviour? If we were to split the suspicion equally by recording each player's action as one complimentary and two contradictory ($^{1.0+0.0+0.0}/_{3.0} = 0.\dot{3}$), this would result in our suspicion of each team member falling, rather than rising.

Of the existing bots, ScepticBot is the only one to account for the models' sabotage frequencies when modifying suspicion scores based on mission results, and then does so only in the fully observable case - when there is no sabotage. Otherwise, when there is a sabotage, ScepticBot modifies suspicion scores by comparing players' votes for the team, to their frequencies of voting for spy teams in past games - a category 2, competence based behaviour. Our implementation is

not specifically limited to calculating fitting scores based upon complimentary and contradictory actions, but further consideration is required in order to achieve an elegant solution, and we will not pursue it further in this work.

The implementation of Stalker tested in this work then, models only category 1 and 2 behaviours. The category 1, perfect information behaviours, are the following:

1. selectsTeamFeaturingSelf: Frequency of subject selecting teams featuring itself.
2. votesForTeamFeaturingSelf: Frequency of subject voting for teams featuring itself.
3. votesForTeamNotFeaturingSelf: Frequency of subject voting for teams not featuring itself.
4. votesForTeam3NotFeaturingSelf: Frequency of subject voting for three-man teams not featuring itself.
5. votesAgainstTeamOnFifthAttempt: Frequency of subject voting against teams on the fifth voting attempt.

Behaviours 4 and 5 above are suspected to be powerful, because they are behaviours which have no advantage for resistance members, and many bots include expert rules to account for this. Such rules may, however, be overruled by spy-specific rules, which would quickly cause a discrepancy between spy and resistance model frequencies that would have a big impact on fitting scores.

The category 2, competence based behaviours, are the following:

1. selectsSuccessfulTeam: Frequency of subject selecting a successful team.
2. votesForSuccessfulTeam: Frequency of subject voting for a successful team.
3. votesForUnsuccessfulTeam: Frequency of subject voting for an unsuccessful team.
4. votesForTeamWithTwoSabotages: Frequency of subject voting for a team with two sabotages.
5. teamOnIsUnsuccessful: Frequency of teams featuring the subject being successful.

Although we have expressed concerns about these (even as early as our analysis of Statistician in Section 4.1.4), it is completely possible that the biases of spies will create a significant statistical distinction in the models, despite any noise caused by misinformation and incompetence on the part of the resistance members. We suspect, for example, that spies will have a bias against behaviour 4, as they will attempt to avoid risking incrimination via double sabotage by rejecting such teams.

Behaviour 5 is an attempt to compensate for neglecting category 3 and 4, partially observable, behaviours. This might allow the OM agent to "learn" the correct amount of suspicion to allocate for presence on failed missions.

4.6.3 Nosey (Grouped Opponent Modelling)

In Section 3.6.2, we briefly explored the idea of OM abstraction levels (granularities), with reference to [Billings et al., 2006], as a method of speeding up adaptation to opponent strategies. We suggested therein that we may initially consider trying out a very coarse granularity, giving the example of maintaining only one opponent model for each role (spy or resistance member), based on the behaviour of all opponents.

Allowing for this was a trivial consideration when implementing Stalker, which our grouped OM bot, Nosey, inherits from, and also has the interesting effect of potentially overcoming the hurdles introduced by our burst competition model (Section 4.4.2) in a manner which seems quite fair and compatible even with human play.

4.7 Experiments

4.7.1 Opponent Groupings

For the tests conducted in this work we define two groups of four opponents which the bots of interest compete in competitions with. To ensure that our subjects do not interfere with each other's performance, each enters into a separate competition against each grouping, bringing the total number of competitors in each competition to five. To ensure the variety of permutations played is representative, each competition runs for a number of games which is a multiple of 1,200 games (Section 4.4); we selected 12,000 games per competition, similar to the amounts used in the Vienna conference competition. Although we expect each experiment to be statistically significant in its own right, due to the large number of games played, each is repeated twice in order to confirm stability in the results.

The first opponent grouping, the beginners, consists of four of the simplest existing agents, all found in the "beginners.py" script:

1. Jammer
2. Deceiver
3. Hippie
4. Paranoid

These are very simple opponents which even a basic, rule following bot like Ruru should be able to outperform. They may, however, pose some difficulty to OM bots, since only Deceiver and Jammer feature behaviours which will be affected by their own role. While Clymily, Magi, and ScepticBot should be able to handle this due to the other techniques employed, our own pure OM agents, Stalker and Nosey, will likely fail.

The second opponent grouping, the advanced agents, consist of four of the strongest competitors from the direct elimination tournament of the Vienna competition. We avoid including our subjects in this, and would also have avoided including other strong OM agents if any were present, due to the possible impact on our results of two opponents learning from one another. Our advanced grouping therefore consists of:

1. PandSBot
2. GarboA
3. Rebounder
4. Bot5Players

We expect that OM agents will be able to model these quite effectively, since their more complex logic will lead to different action frequencies in different roles. Ruru, however, is expected to perform poorly.

4.7.2 Expert Rules Performance

To test the effectiveness of our expert rules bot, Ruru, we assess its performance against both beginner and advanced opponent groupings in terms of win percentage, compared to that of another, much simpler rule based bot, RuleFollower (beginners.py), and a simple bot which combines expert rules and some logical deduction, LogicalBot (aigd.py).

4.7.3 Opponent Modelling Inaccuracies

Each of the existing OM agents which we have modified to log their suspicion inaccuracies (Clymily, ScepticBot, and Magi) is tested against both the beginner and advanced groups, recording the suspicion inaccuracies, win percentages, and other data output by competition.py.

For Clymily, where we see any sign of slow adaptation, an extra set of tests is conducted by running each experiment one additional time without deleting the external file which stores its opponent model, so that we might examine the effect the knowledge which is carried over.

We also test our pure OM agent, Stalker, against both opponent groupings, recording only the suspicion inaccuracies, since Stalker's actual decisions are identical to Ruru's, which we test separately. The modular nature of Stalker's modelled behaviours allows us to easily test the effectiveness of different sets, or even individual behaviours; for this work, we limit our experiments to five configurations we believe will be of interest:

1. All perfect information and competence based behaviours described in Section 4.6.2.
2. Only the perfect information behaviours described in Section 4.6.2.
3. Only the competence based behaviours described in Section 4.6.2.
4. All perfect information and competence based behaviours, excluding teamOnIsUnsuccessful.
5. Only teamOnIsUnsuccessful.

We single out teamOnIsUnsuccessful because of its relation to the category 3 and 4, partially observable action, behaviours, and because we highly suspicious of its usefulness.

Finally, we test our grouped OM agent, Nosey, against both opponent groupings. Configurations of Nosey tested are informed by the results of Stalker's tests, and early tests with Nosey.

4.7.4 Burst Competition Impact

The final set of experiments conducted aim to test the impact of our burst competition model. Bots, opponent groupings, and configurations (of Stalker and/or Nosey) tested with the burst competition model are be those which demonstrate some evidence of gradual adaptation over the course of games played for other tests.

We also investigate the effect of the burst competition model on the final round of the direct elimination tournament from the Vienna conference competition, which featured three OM agents. We first reproduce the conditions of the original competition and confirm that our results corroborate theirs. This means running a single, 25,000 game competition between PandSbot, ScepticBot, Clymily, Rebounder, and Magi. Next, we switch over to the burst competition model, leaving all other parameters the same but testing with a small burst size.

5 Results

In this section we present results of the experiments previously described in Section 4.7. Further examination and analysis of these results is conducted in Section 6. Some uninteresting figures are omitted to avoid clutter, and can be found instead in Appendix A.

5.1 Reading Competition Output

To aid in understanding of competition result listings, such as Figures 11 and 12, we will first describe the meaning of each column:

The first (leftmost) column beneath SPIES, RESISTANCE, and TOTAl, lists the win percentage for each competitor, in each role. Other columns are specific to the roles whose header they fall under. The second column beneath SPIES lists the percentage of votes by resistance players, regarding teams featuring the bot in question as a spy, which were positive. The second and third columns beneath RESISTANCE list, respectively, the percentage of teams featuring no spies which the bot in question voted positively for, and the percentage of teams featuring spies which the bot in question voted negatively for. The final column under SPIES lists the percentage of teams put forth by resistance players which featured the bot in question, while the final column under RESISTANCE lists the percentage of teams put forth by the bot in question, as resistance, featuring no spies.

5.2 A Note On Interpreting Graphs

Graphs produced from suspicion inaccuracy data feature varying degrees of noise, likely resulting from a combination of non-determinism in some bots' logic, the random order in which permutations are played, and the instability of opponent models. The majority of graphs presented here are smoothed out by averaging over sets of 500 games, selected as a compromise between readability and accuracy. Therefore, we advise cautious interpretation of these graphs, and will of course bear this in mind in our own discussion. Further discussion on the noise in this data and its impact on our results is presented in Section 6.2.1.

5.3 Expert Rules Performance

5.3.1 Against Beginners

Figures 11, 12, and 13 show details of the performance of three agents (Ruru, RuleFollower, and LogicalBot, respectively) against the beginners opponent grouping.

Our own bot, Ruru, is outperformed by a small margin by both RuleFollower and LogicalBot, despite implementing a far greater number of expert rules than RuleFollower which we believed would be effective. LogicalBot's greater performance may be attributed to better performance as a resistance member. This is likely a result of the logical deduction conducted.

Disproportionate win percentages of spies and resistance members in these results is likely a result of HippieBot's and Jammer's presence, as these two will approve any mission in voting. As we previously discussed (Section 2.2), being teamed up with incompetent players can be a huge disadvantage for resistance members.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  Paranoid          97.0%          85.5%            41.8%
  Hippie            96.6%          63.3%            41.6%
  Ruru              94.9%          71.0%            41.8%
  Deceiver          90.8%          70.4%            42.8%
  Jammer            90.1%          62.1%            38.8%
RESISTANCE                        (vote,           select)
  Paranoid           8.2%          36.9% 83.5%      32.0%
  Hippie             7.9%          100.0%  0.0%     33.5%
  Ruru               6.8%          100.0% 27.2%     26.4%
  Deceiver           4.1%          100.0% 23.5%     33.9%
  Jammer             3.6%          100.0%  0.0%     19.7%
TOTAL
  Paranoid          43.7% (e=0.89 n=12000)
  Hippie            43.4% (e=0.89 n=12000)
  Ruru              42.0% (e=0.88 n=12000)
  Deceiver          38.8% (e=0.87 n=12000)
  Jammer            38.2% (e=0.87 n=12000)
```

Figure 11: Performance of Ruru against beginners opponent grouping.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  RuleFollower      98.6%          71.1%            40.3%
  Paranoid          97.0%          86.1%            41.1%
  Hippie            96.5%          62.9%            40.0%
  Jammer            93.3%          61.9%            36.5%
  Deceiver          91.6%          70.0%            41.3%
RESISTANCE                        (vote,           select)
  RuleFollower       6.7%          100.0% 26.0%     33.9%
  Paranoid           5.7%          35.8% 83.9%      33.6%
  Hippie             5.3%          100.0%  0.0%     33.5%
  Jammer             3.2%          100.0%  0.0%     19.2%
  Deceiver           2.1%          100.0% 23.9%     34.5%
TOTAL
  RuleFollower      43.5% (e=0.89 n=12000)
  Paranoid          42.2% (e=0.88 n=12000)
  Hippie            41.8% (e=0.88 n=12000)
  Jammer            39.3% (e=0.87 n=12000)
  Deceiver          37.9% (e=0.87 n=12000)
```

Figure 12: Performance of RuleFollower against beginners opponent grouping.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                          (voted,          selected)
  LogicalBot       98.4%        71.5%            40.1%
  Hippie           93.7%        59.3%            36.7%
  Paranoid         93.6%        81.0%            36.6%
  Jammer           90.4%        59.3%            35.6%
  Deceiver         87.6%        68.4%            42.4%
RESISTANCE                     (vote,           select)
  LogicalBot       11.0%        100.0% 39.4%     44.8%
  Hippie            7.8%        100.0%  0.0%     33.4%
  Paranoid          7.8%        34.6% 83.6%      34.0%
  Jammer            5.7%        100.0%  0.0%     19.3%
  Deceiver          3.8%        100.0% 22.9%     33.9%
TOTAL
  LogicalBot       46.0% (e=0.89 n=12000)
  Hippie           42.2% (e=0.88 n=12000)
  Paranoid         42.2% (e=0.88 n=12000)
  Jammer           39.6% (e=0.87 n=12000)
  Deceiver         37.3% (e=0.87 n=12000)
```

Figure 13: Performance of LogicalBot against beginners opponent grouping.

5.3.2 Against Advanced

Figures 14, 15, and 16 show details of the performance of three agents (Ruru, RuleFollower, and LogicalBot, respectively) against the advanced opponent grouping.

Here, these simple bots are all shown to be ineffective against more complex agents, placing last in every category, unsurprisingly. However, Ruru is no longer outclassed by RuleFollower as it was in Section 5.3.1, making up for poor performance as a resistance member with significantly stronger spy performance than RuleFollower, likely as a result of being selected, and approved in voting for a greater number of teams. In fact, in the spy category, Ruru even outperforms the logical deduction and expert rules based LogicalBot, with a considerable difference in selection rate.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  PandSBot          78.6%          57.3%            28.6%
  Rebounder         74.9%          49.2%            24.7%
  Bot5Players       73.1%          55.8%            24.3%
  GarboA            68.8%          58.2%            23.2%
  Ruru              44.7%          40.3%            21.0%
RESISTANCE                      (vote,           select)
  PandSBot          39.0%          89.0% 66.4%      70.3%
  Rebounder         36.6%          99.1% 51.7%      66.3%
  Bot5Players       35.4%          96.0% 47.2%      60.1%
  GarboA            32.5%          82.6% 56.4%      56.6%
  Ruru              16.5%          100.0% 18.0%     22.0%
TOTAL
  PandSBot          54.8% (e=0.89 n=12000)
  Rebounder         51.9% (e=0.89 n=12000)
  Bot5Players       50.5% (e=0.89 n=12000)
  GarboA            47.0% (e=0.89 n=12000)
  Ruru              27.8% (e=0.80 n=12000)
```

Figure 14: Performance of Ruru against advanced opponent grouping.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  PandSBot          73.7%          56.6%            25.0%
  Rebounder         71.7%          48.4%            22.4%
  Bot5Players       65.3%          53.2%            21.1%
  GarboA            62.2%          55.1%            20.5%
  RuleFollower      36.6%          38.2%            13.6%
RESISTANCE                      (vote,           select)
  PandSBot          46.0%          91.5% 66.4%      74.6%
  Rebounder         44.6%          98.1% 52.7%      71.4%
  Bot5Players       40.4%          95.7% 47.7%      67.1%
  GarboA            38.3%          86.4% 56.0%      63.5%
  RuleFollower      21.2%          100.0% 19.1%     31.7%
TOTAL
  PandSBot          57.1% (e=0.89 n=12000)
  Rebounder         55.4% (e=0.89 n=12000)
  Bot5Players       50.3% (e=0.89 n=12000)
  GarboA            47.9% (e=0.89 n=12000)
  RuleFollower      27.4% (e=0.80 n=12000)
```

Figure 15: Performance of RuleFollower against advanced opponent grouping.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  PandSBot          66.4%          57.2%            22.1%
  Rebounder         58.1%          46.7%            18.6%
  GarboA            51.2%          57.9%            18.1%
  Bot5Players       50.0%          56.7%            18.1%
  LogicalBot        38.3%          39.6%            13.5%
RESISTANCE                      (vote,           select)
  PandSBot          56.3%          93.2% 66.6%      77.6%
  Rebounder         50.8%          98.6% 50.5%      71.3%
  GarboA            46.1%          85.9% 54.6%      63.4%
  Bot5Players       45.3%          95.6% 46.5%      66.1%
  LogicalBot        37.5%          100.0% 27.7%     49.2%
TOTAL
  PandSBot          60.3% (e=0.88 n=12000)
  Rebounder         53.7% (e=0.89 n=12000)
  GarboA            48.2% (e=0.89 n=12000)
  Bot5Players       47.2% (e=0.89 n=12000)
  LogicalBot        37.8% (e=0.87 n=12000)
```

Figure 16: Performance of LogicalBot against advanced opponent grouping.

5.4 Previous Opponent Modelling Inaccuracies

5.4.1 Clymily

Figure 17 shows the suspicion inaccuracy graphs of Clymily when faced with the beginner and advanced opponent groupings, in turn. While it seems that from the first mission Clymily gets more accurate information about the beginners than the advanced agents, subsequent missions show much more accurate identification of roles in the games against advanced agents, reaching near certainty of the correct roles by the fourth and fifth.

There also appears to be a downward trend in inaccuracies over the course of the 12,000 games, against both groupings, and perhaps even a vague curve which is most distinct in the beginners graph. Even bearing in mind that the same graph with shorter intervals for game averaging might reveal significant noise (Section 5.2), this suggests some learning is taking place, and is beneficial. The only system of Clymily which could be responsible for this is its OM.

Since the evidence for this is strongest in the beginners graph, we tried Clymily again against the beginners opponent grouping, with the benefit of its OM data stored from the previous competition. Figure 18 shows the results of this competition. Plots for each mission begin at points comparable to their ending positions in the previous competition, and demonstrate no downward trend, which we believe indicates that Clymily's models previously reached their optimal representation, and that it continues to benefit from this.

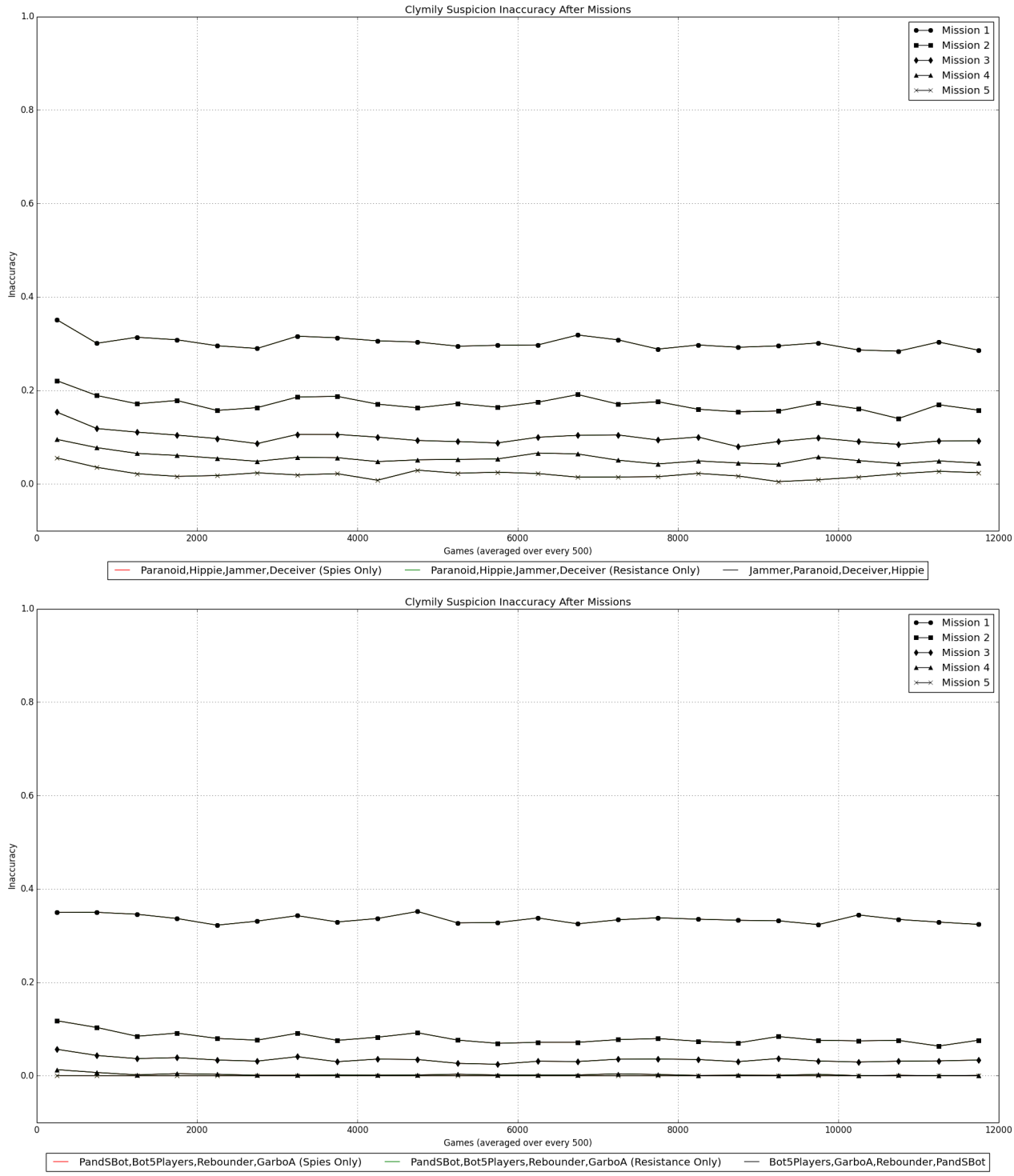

Figure 17: Suspicion inaccuracies of Clymily vs. beginner and advanced groupings, averaged over every 500 games.

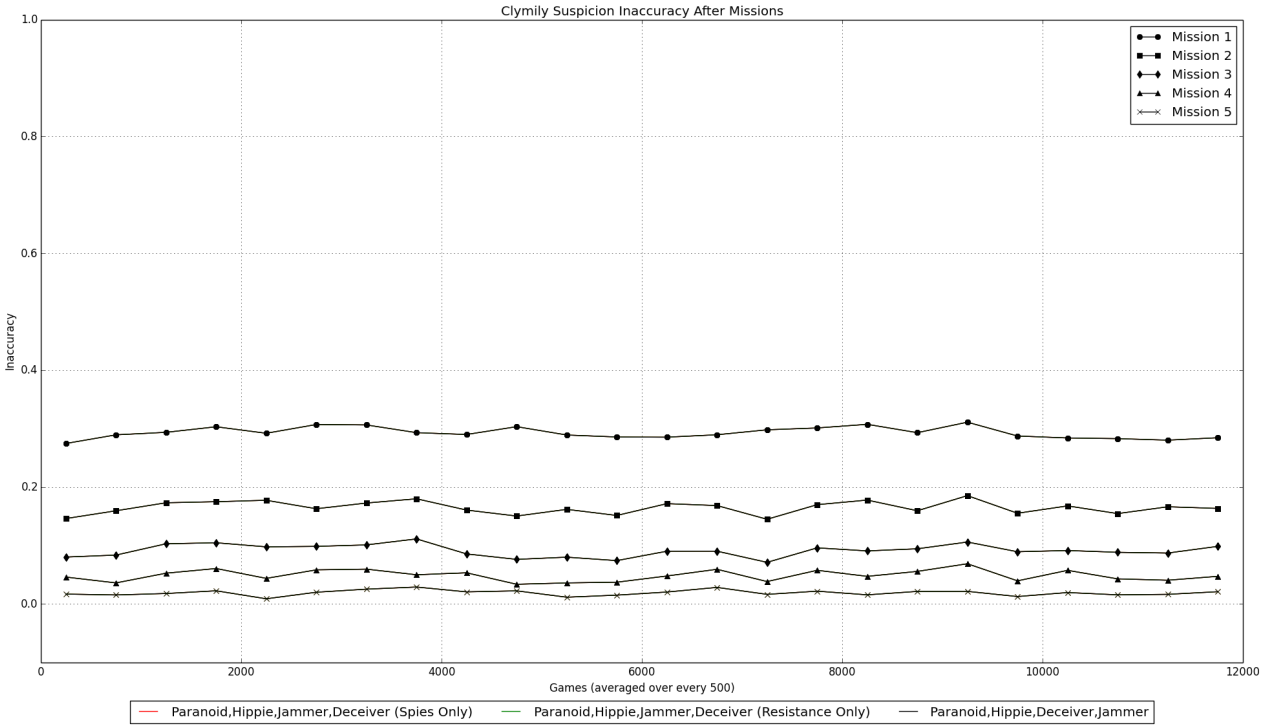

Figure 18: Suspicion inaccuracies of Clymily vs. beginners, using previous data, averaged over every 500 games.

5.4.2 ScepticBot

Figure 19 shows the suspicion inaccuracy graphs of ScepticBot when faced with the beginner and advanced opponent groupings, in turn. Unlike Clymily's graphs (Figure 17), there is no notable distinction between the performance against beginners and advanced agents, and no sign of any learning over the course of the 12,000 games. The presence of other techniques in ScepticBot affecting the suspicion scores prevents us from knowing what the impact of the OM is, but at least there is no evidence that it is detrimental.

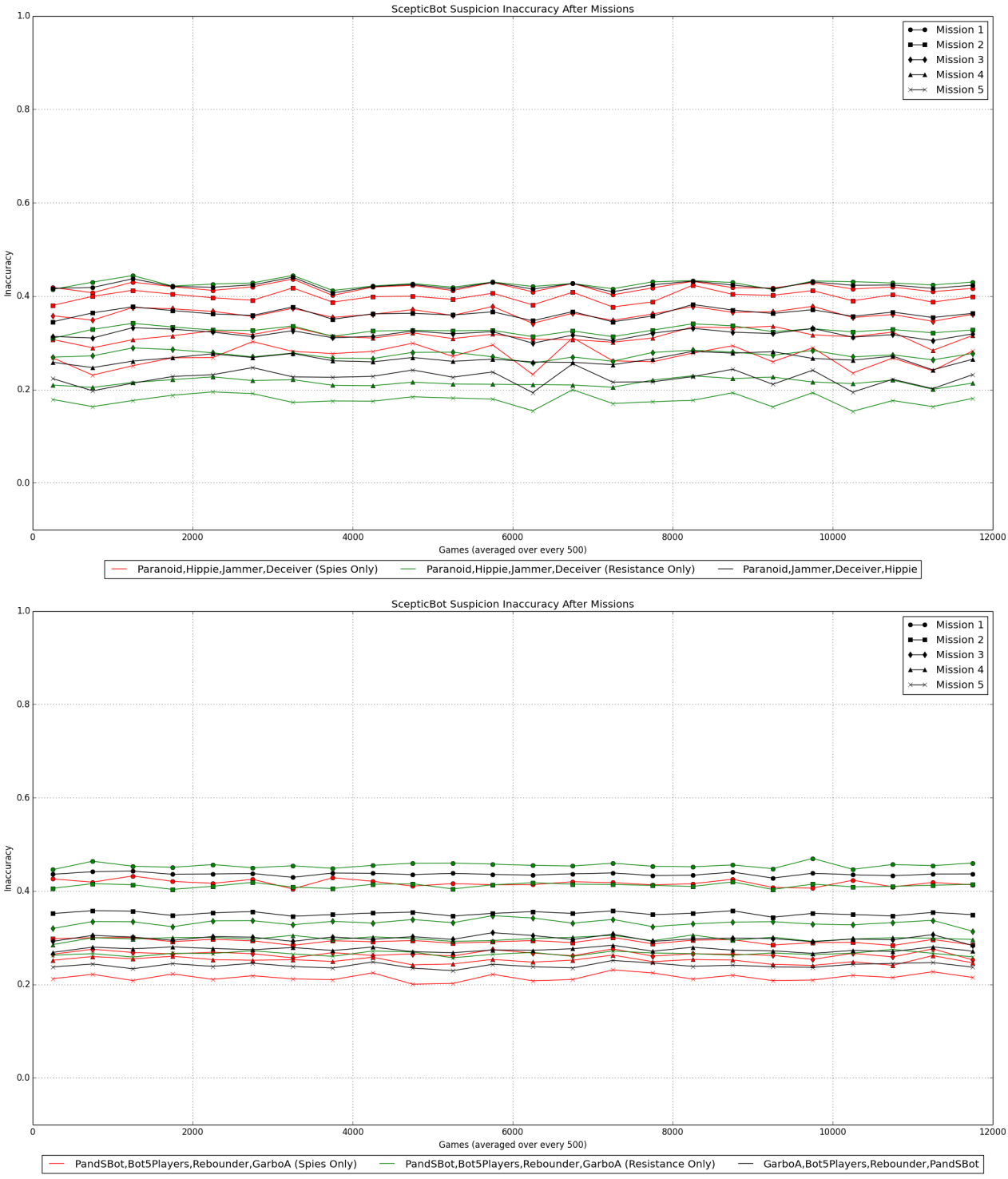

Figure 19: Suspicion inaccuracies of ScepticBot vs. beginner and advanced groupings, averaged over every 500 games.

5.4.3 Magi

Figure 21 shows the suspicion inaccuracy graphs of Magi when faced with the beginner and advanced opponent groupings, in turn. Here there is a stark contrast between performance against the two groupings. Inaccuracies for advanced opponents show a similar progression between missions to those of Clymily, though there is no sign of adaptation between games. Against beginners Magi continues to identify the resistance members well, but struggles with spies, where inaccuracies increase significantly from one mission to the next. Again, it is not clear what part of this behaviour can be attributed to Magi's OM, and what part is the result of its other techniques.

Here we include one additional graph, demonstrating the inaccuracies of Magi's suspicion scores for Deceiver in the beginners competition, split by role (Figure 20). This pattern is the same, more or less, for each of the beginner opponents, with spy inaccuracies generally increasing between missions.

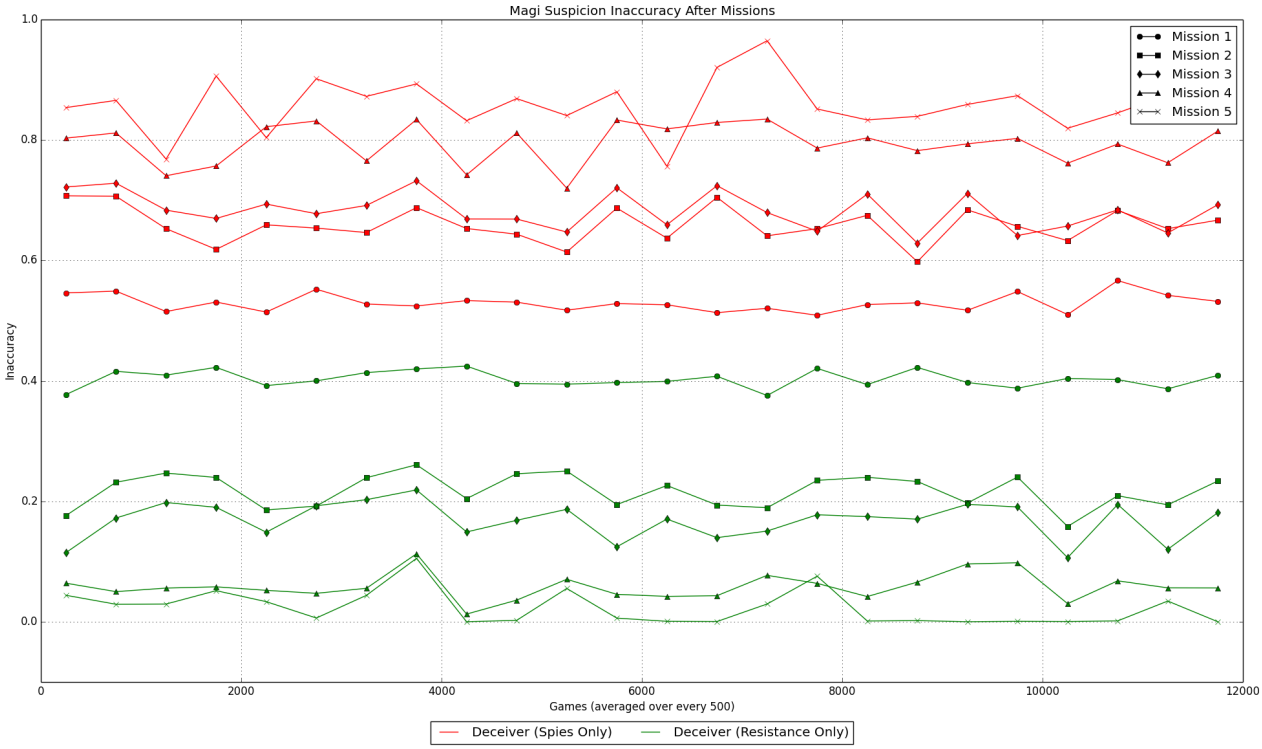

Figure 20: Suspicion inaccuracies of Magi vs. Deceiver, averaged over every 500 games.

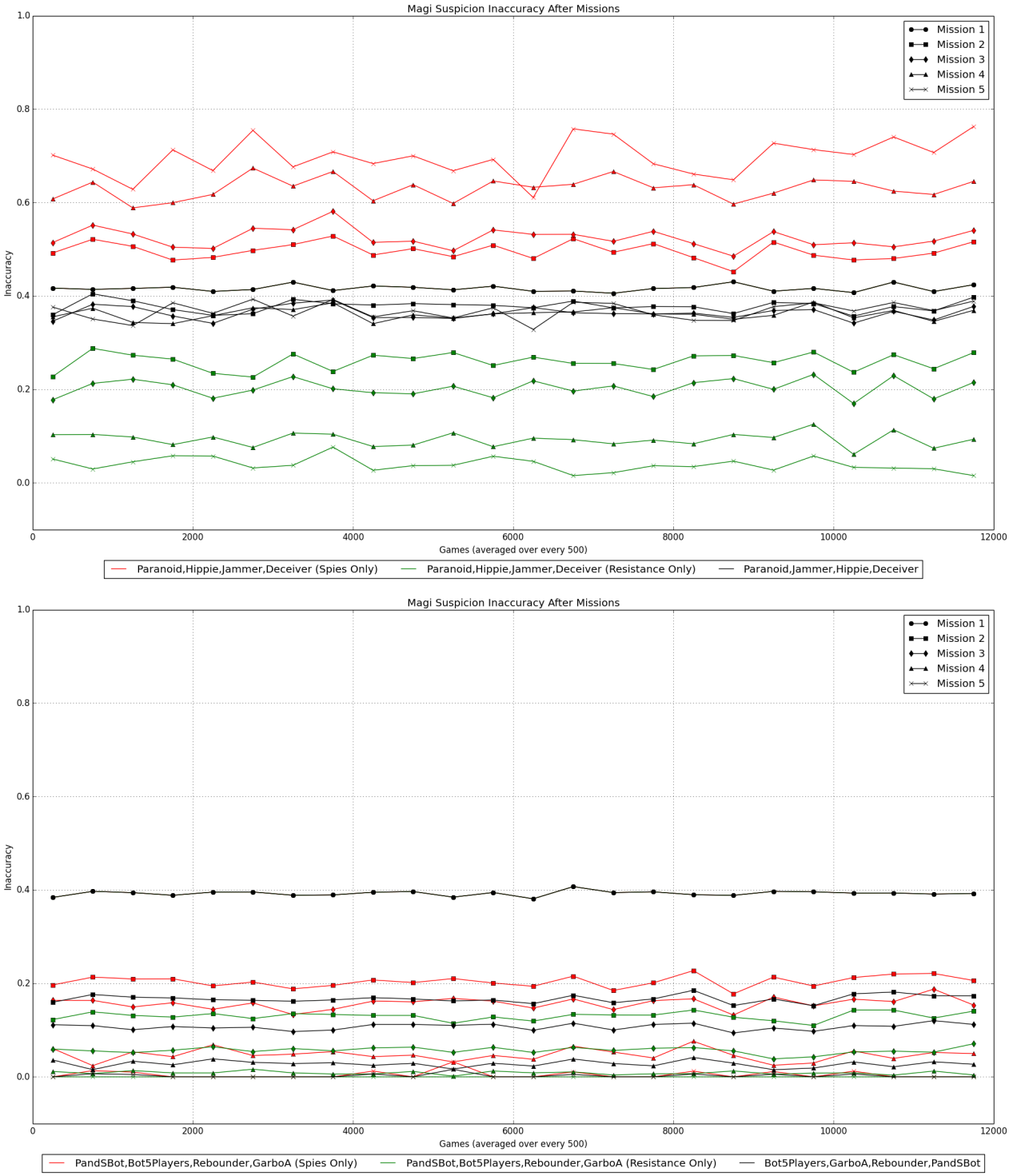

Figure 21: Suspicion inaccuracies of Magi vs. beginner and advanced groupings, averaged over every 500 games.

5.5 Pure Opponent Modelling Inaccuracies (Stalker)

5.5.1 Against Beginners

Figure 22 shows the suspicion inaccuracy graph of Stalker, configured to model all behaviours described in Section 4.6.2, when faced with the beginners opponent grouping.

We omit graphs for other Stalker configurations against the beginners opponent grouping as, averaged over 500 games, each presents as a fairly straight line, with all missions clustered just below 0.5. 0.5 is the reading we expect when an opponent fits equally well to both its spy and resistance member models, or when subsequent games vary evenly between high and low accuracy, so this shows minimal effectiveness of opponent modelling against the beginners.

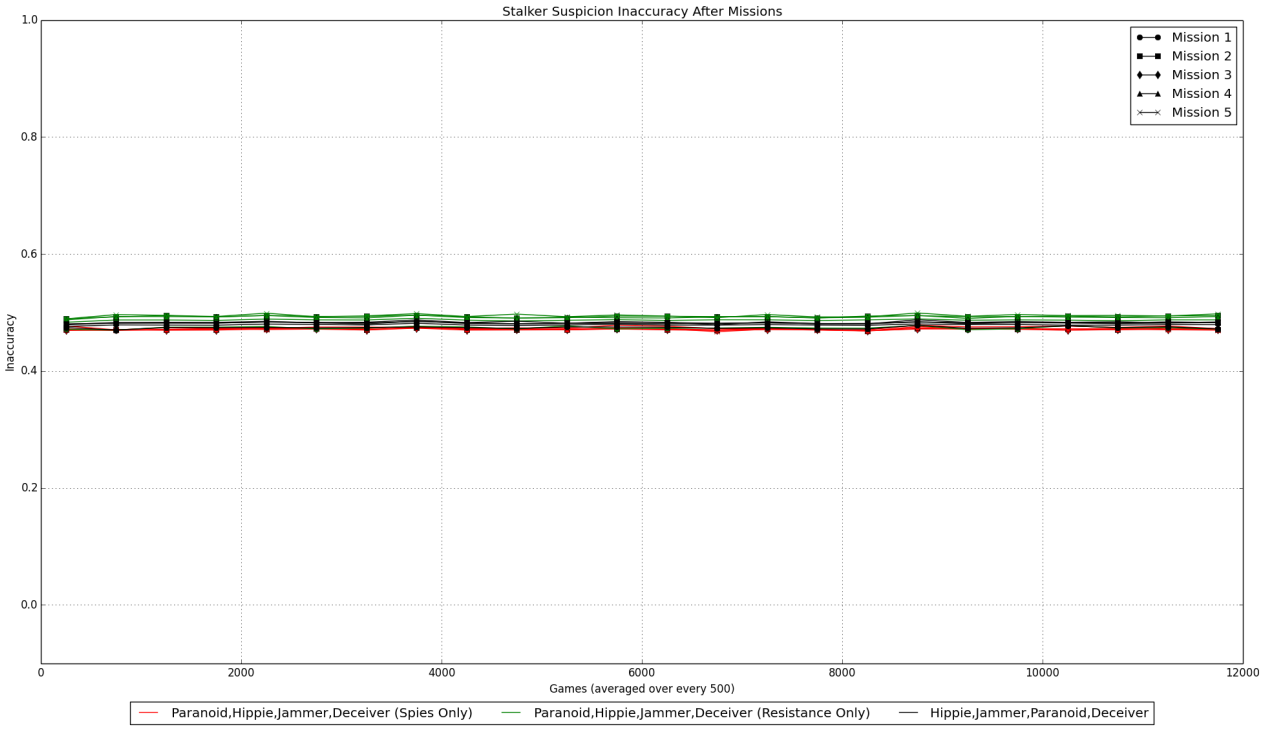

Figure 22: Suspicion inaccuracies of Stalker (all behaviours) vs. beginners, averaged over every 500 games.

5.5.2 All Behaviours

Figure 23 shows the suspicion inaccuracy graph of Stalker, configured to model all behaviours described in Section 4.6.2, when faced with the advanced opponent grouping.

This first experiment with our pure OM agent against the advanced grouping is promising, demonstrating a clear reduction in suspicion inaccuracies after each mission. Though not as strong as ScepticBot (Figure 19), and much weaker than Clymily (Figure 17) or Magi (Figure 21), this at last gives us evidence that OM is somewhat effective at identifying player roles in "The Resistance", without the support of any other techniques.

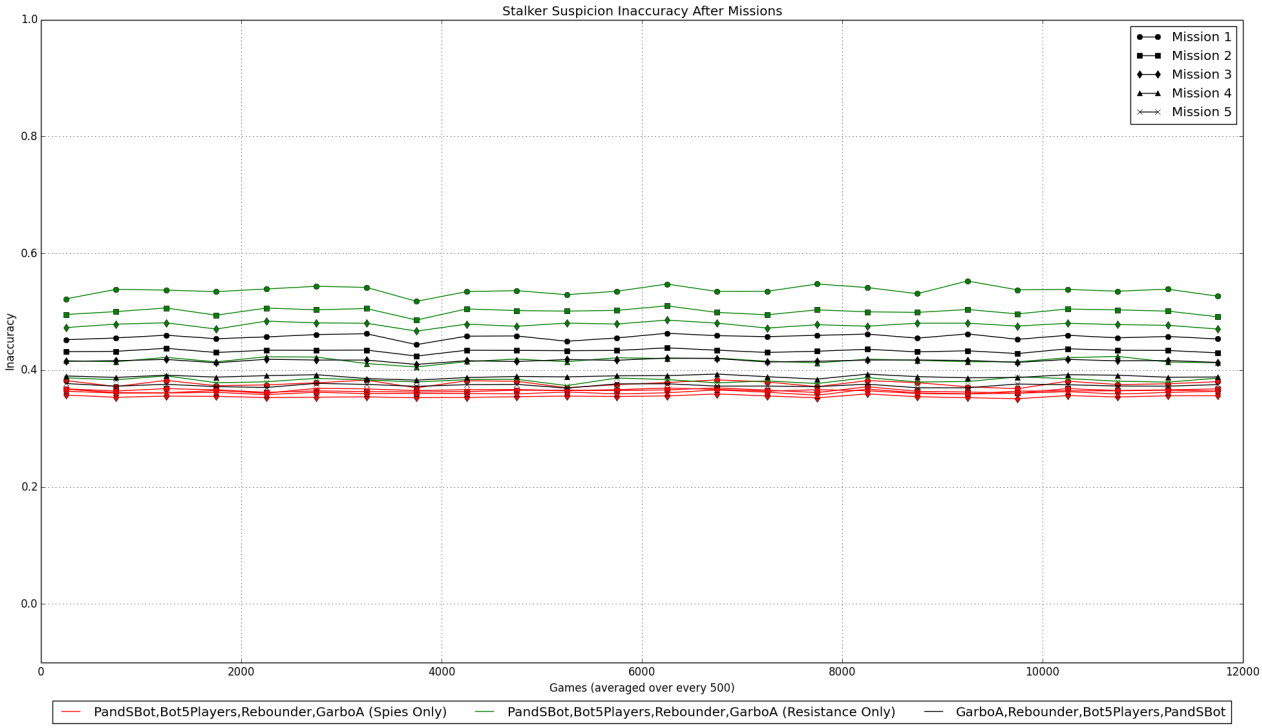

Figure 23: Suspicion inaccuracies of Stalker (all behaviours) vs. advanced, averaged over every 500 games.

5.5.3 Only Perfect Information Behaviours

Figure 24 shows the suspicion inaccuracy graph of Stalker, configured to model only the perfect information behaviours described in Section 4.6.2, when faced with the advanced opponent grouping.

These results are surprising, and somewhat disappointing, as we expected that decisions made by modelled players with complete control over the parameters, such as whether to select a team featuring themselves, whether to vote against a team on the fifth voting attempt, and so on, would demonstrate the greatest distinction between behaviour as a spy, and as a resistance member. Instead, we see results comparable to those of Stalker against the beginners opponent grouping (Figure 22), with missions clustered closely together around 0.5.

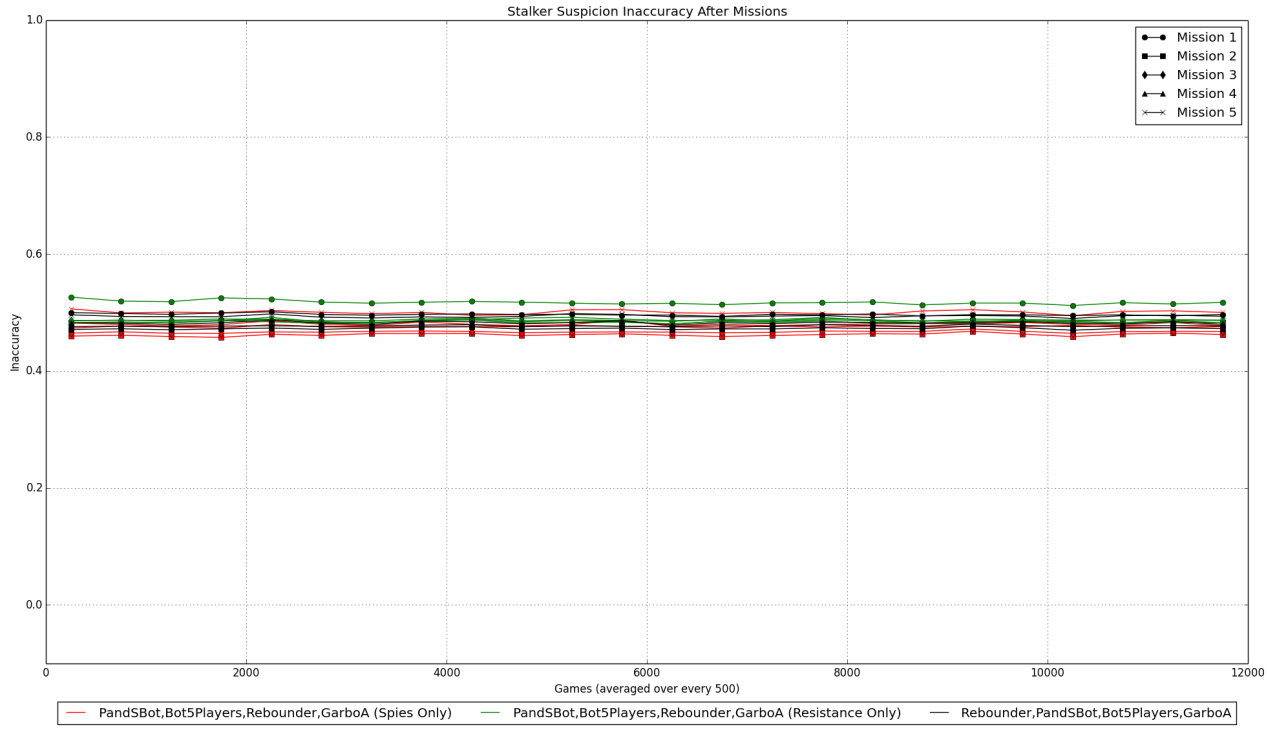

Figure 24: Suspicion inaccuracies of Stalker (only perfect information) vs. advanced, averaged over every 500 games.

5.5.4 Only Competence Based Behaviours

Figure 25 shows the suspicion inaccuracy graph of Stalker, configured to model only the competence based behaviours described in Section 4.6.2, when faced with the advanced opponent grouping.

Despite a large discrepancy in how well this configuration identifies spies compared to resistance members, this appears to represent a great overall improvement over the configuration using all behaviours (Figure 23). This configuration demonstrates perhaps the strongest performance of any Stalker configuration tested, comparable to that of the existing OM agent ScepticBot (Figure 19), which also employs more complex techniques. Identification of resistance members is, however, quite weak, leading to higher overall inaccuracies in missions one, two and three, which are critical to preventing an early spy victory.

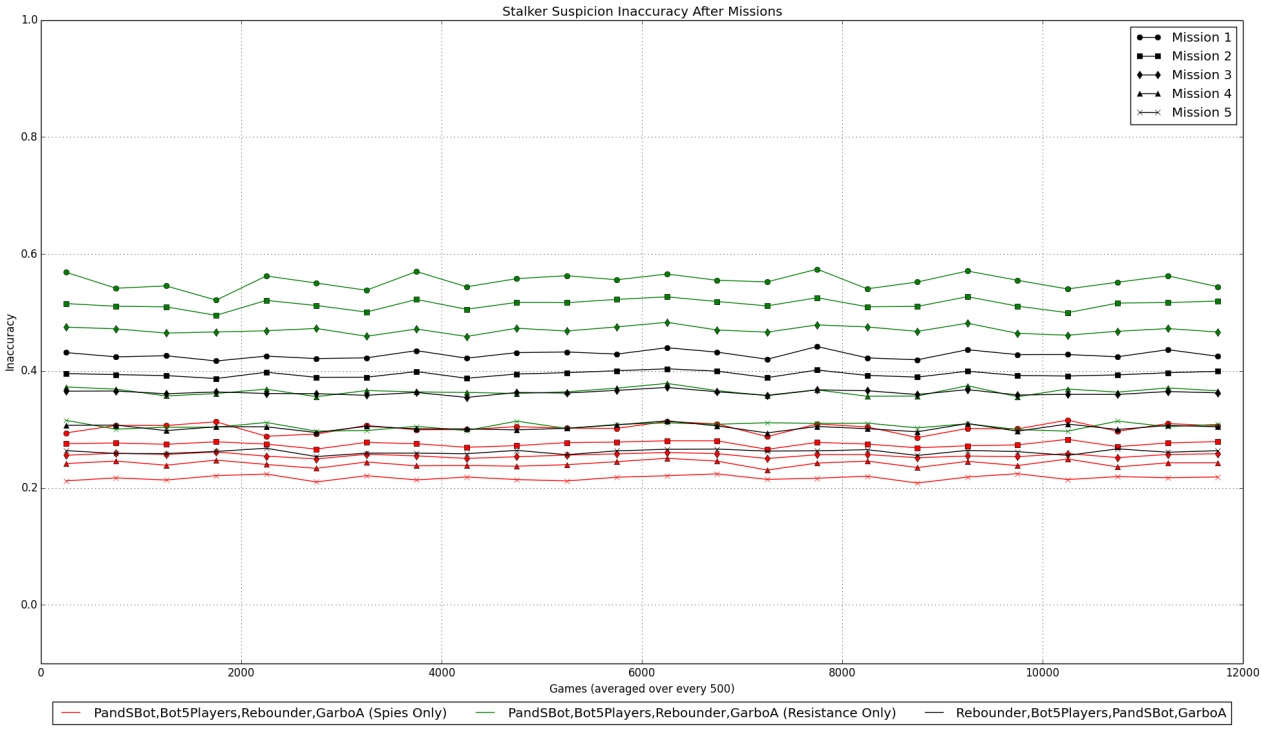

Figure 25: Suspicion inaccuracies of Stalker (only competence based) vs. advanced, averaged over every 500 games.

5.5.5 All Behaviours Except teamOnIsUnsuccessful

Figure 26 shows the suspicion inaccuracy graph of Stalker, configured to model all behaviours described in Section 4.6.2 apart from teamOnIsUnsuccessful, when faced with the advanced opponent grouping.

Despite our concerns about the "teamOnIsUnsuccessful" behaviour, removing it from the modelled behaviours causes a slight increase in suspicion inaccuracies when compared to our experiment with all behaviours (Figure 23). Although resistance member inaccuracies appear lower than in the all behaviours configuration, the overall result is high inaccuracy due to lessened ability to identify spies.

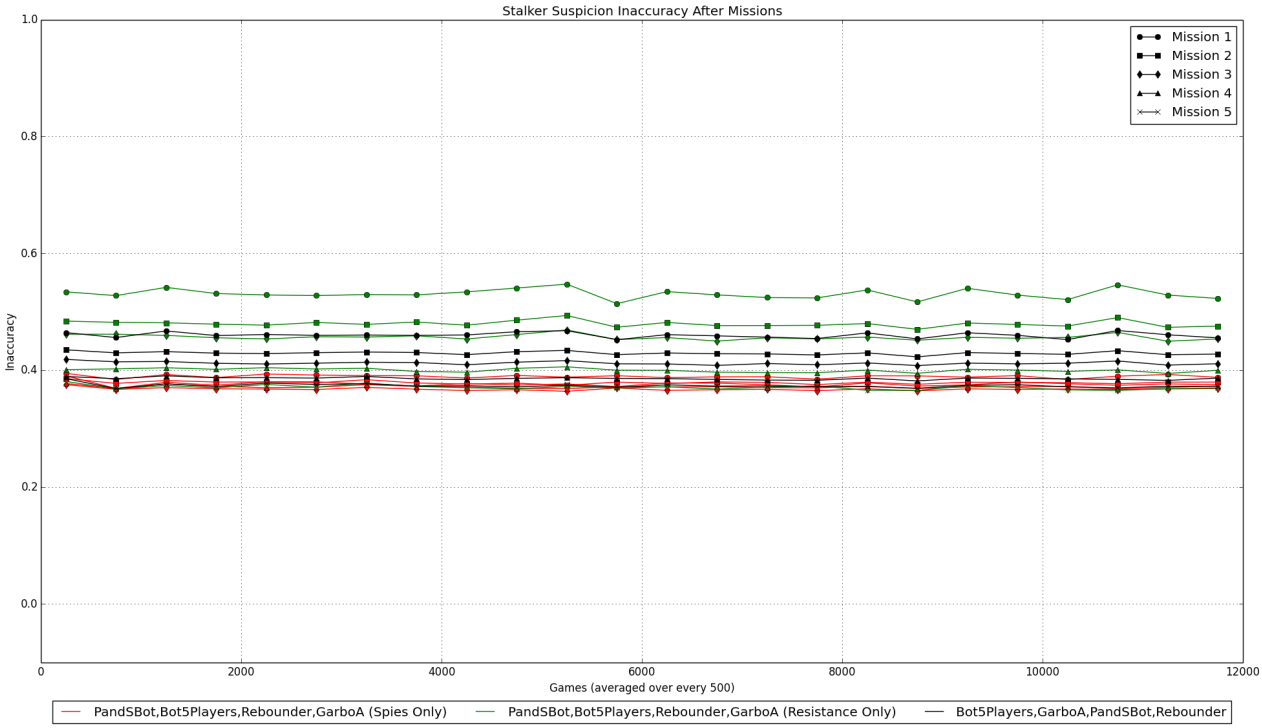

Figure 26: Suspicion inaccuracies of Stalker (all behaviours except teamOnIsUnsuccessful) vs. advanced, averaged over every 500 games.

5.5.6 Only teamOnIsUnsuccessful

Figure 27 shows the suspicion inaccuracy graph of Stalker, configured to model only the teamOnIsUnsuccessful behaviour, when faced with the advanced opponent grouping.

This configuration shows the second best overall performance of any Stalker configuration tests (behind the competence based configuration, Figure 25), despite having the highest inaccuracies for resistance players.

While it is uncertain, given such a vast difference between spy and resistance member inaccuracies, whether the suspicion scores in each individual game would be useful, the fact that the overall outcome of this behaviour appears positive is quite encouraging, especially considering it represents partially observable actions which are even, to some extent, out of the control of the individual modelled.

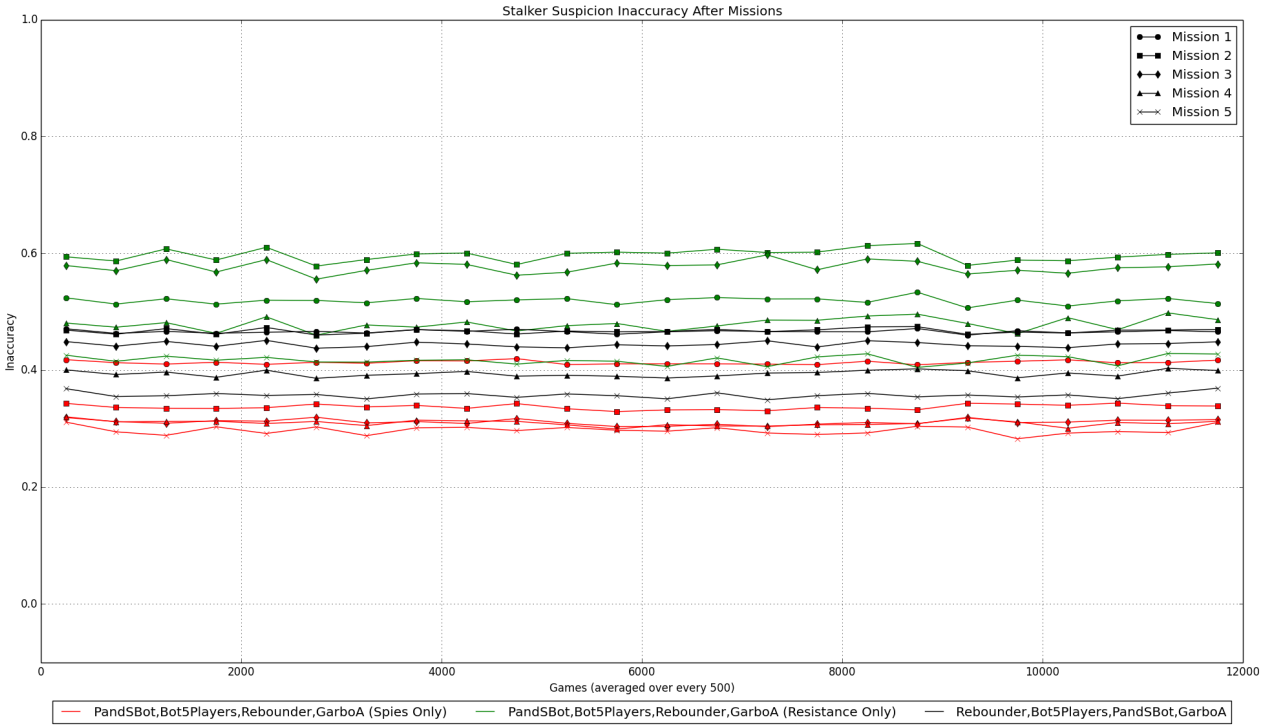

Figure 27: Suspicion inaccuracies of Stalker (only teamOnIsUnsuccessful) vs. advanced, averaged over every 500 games.

5.6 Grouped Opponent Modelling Inaccuracies (Nosey)

5.6.1 Against Beginners

Figure 28 shows the suspicion inaccuracy graph of Nosey, configured to model all behaviours described in Section 4.6.2, when faced with the beginners opponent grouping.

As with Stalker (Section 5.5.1), we omit graphs for other Nosey configurations against the beginner opponent group, as each displays more or less the same outcome, with no indication of OM being effective against beginners.

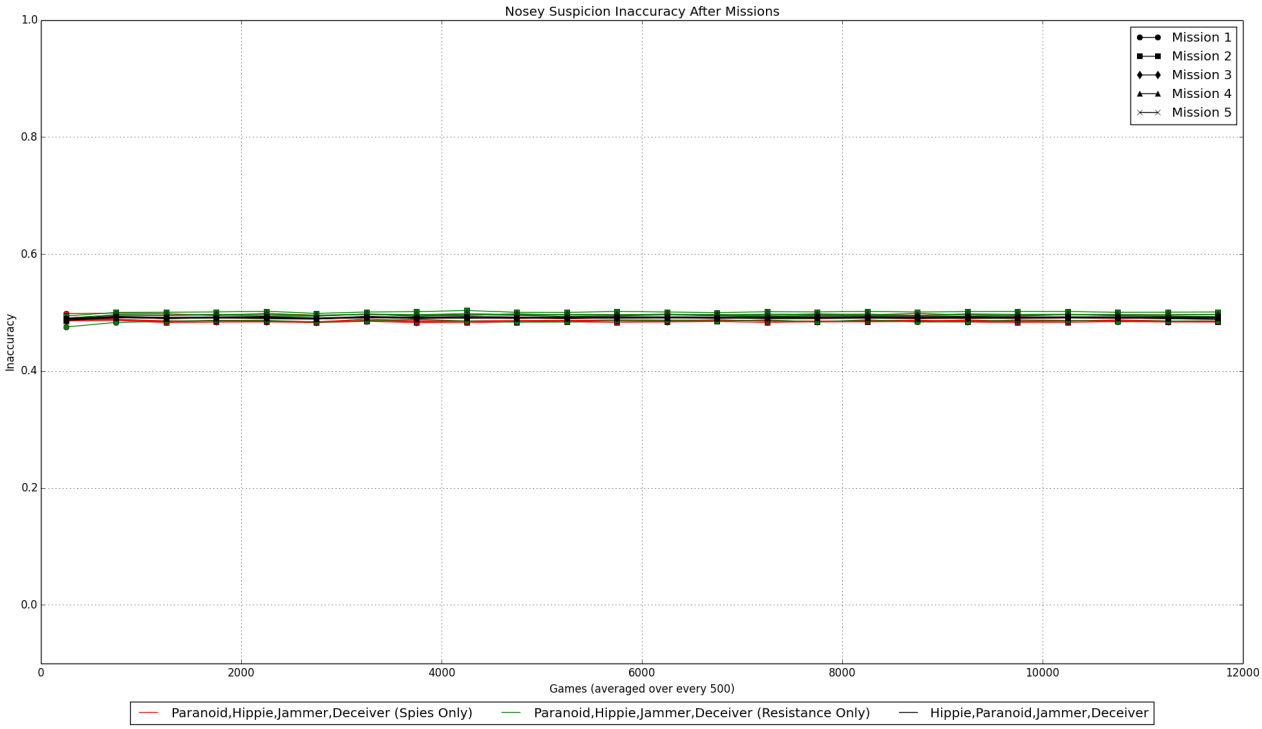

Figure 28: Suspicion inaccuracies of Nosey (all behaviours) vs. beginners, averaged over every 500 games.

5.6.2 Against Advanced

Figure 29 shows the suspicion inaccuracy graph of Nosey, configured to model all behaviours described in Section 4.6.2, when faced with the advanced opponent grouping.

Although we ran experiments using four configurations (all behaviours, perfect information behaviours, competence based behaviours, and teamOnIsUnsuccessful), we will include only the first of the graphs here as an example, as in each case performance is indistinguishable from that of Stalker with the same configuration and opponent grouping (Section 5.5).

While this seems to support the idea of using coarser granularities for OM in "The Resistance", it is possible that further testing with more varied opponent groupings may interfere with the strong performance seen here.

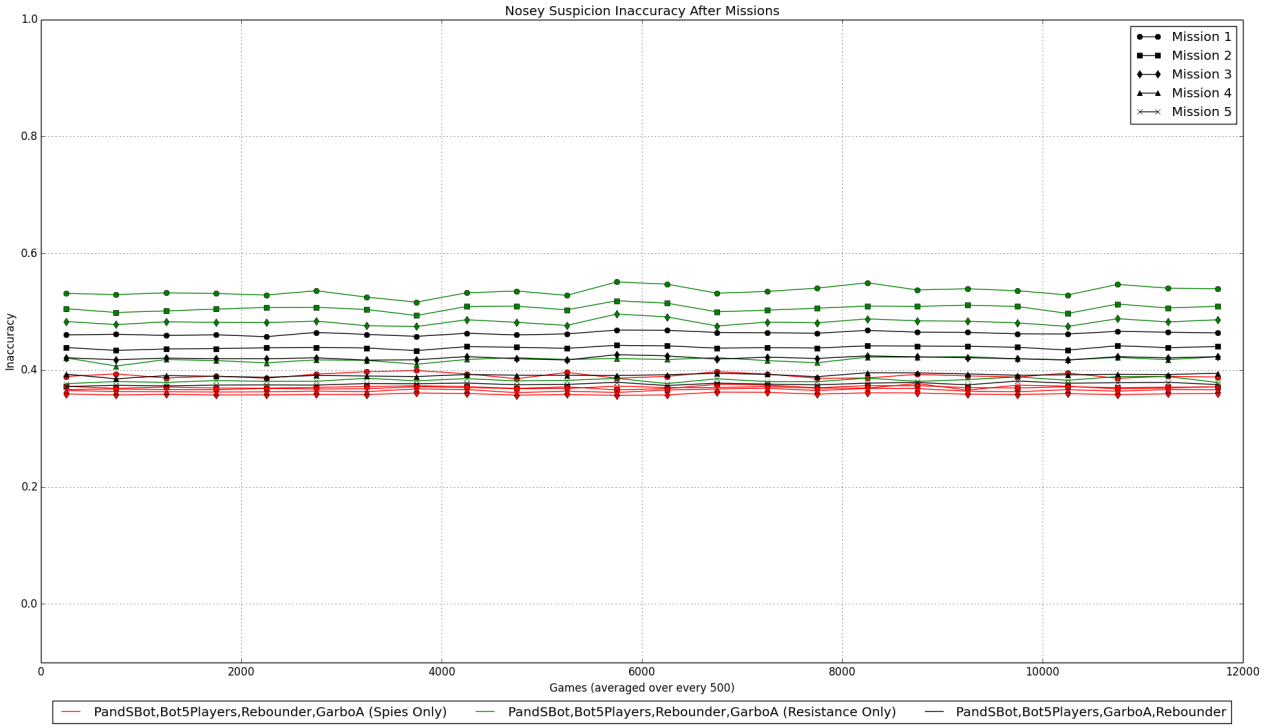

Figure 29: Suspicion inaccuracies of Nosey (all behaviours) vs. advanced, averaged over every 500 games.

5.7 Burst Competition Impact

5.7.1 Impact on Slowly Adapting Agents

From the results featured in Section 5.4 we were only able to observe signs of gradual adaptation to opponent strategies in one agent: Clymily. We also saw no sign of gradual adaptation in our own pure OM agents, Stalker and Nosey (Sections 5.5 and 5.6). Therefore we focus our attention on Clymily for this part of the experimentation.

Since the strongest evidence of gradual adaptation was seen in Clymily's inaccuracy scores against the beginners opponent grouping, we ran further competitions with this grouping, with burst sizes of 1,200 (the highest possible), and 100. The steeper, almost curved, section at the beginning of the original graph (Figure 17) lasts for around 2,000 games, so we expect a visible difference even at the highest burst size possible.

Figure 30 shows the suspicion inaccuracy graphs for burst sizes 1,200 and 100. Note that for burst size 1,200 we averaged over 600 games rather than 500, aligning the average groupings to half of the burst size in order to more accurately capture its effects. We therefore see an alternating pattern where every first point plotted represents the first half of a burst, with high inaccuracies due to starting new opponent models, and every second point plotted represents the second half of a burst, with lower inaccuracies due to utilization of the existing opponent models. The graph for burst size 100, on the other hand, appears as a roughly straight line when viewed at this scale, as models are started afresh at intervals too regular to be apparent.

Finally, after observing the effect of the burst competition model on suspicion inaccuracies against beginners, we ran one further experiment, Clymily against the advanced opponent grouping,

burst size 100, to allow us to gauge the impact of this limitation on Clymily's performance in terms of the competition outcome (this is difficult with the beginners grouping, where Clymily's other logic ensures it still dominates).

The output of this competition and that of the original advanced grouping competition, graphed in Figure 17, can be seen in Figure 31. In the burst competition Clymily displays weaker performance as a resistance member, voting negatively for a smaller percentage of teams featuring spies, and making poorer selections when it is the leader. Overall performance is, however, better, due to stronger spy play.

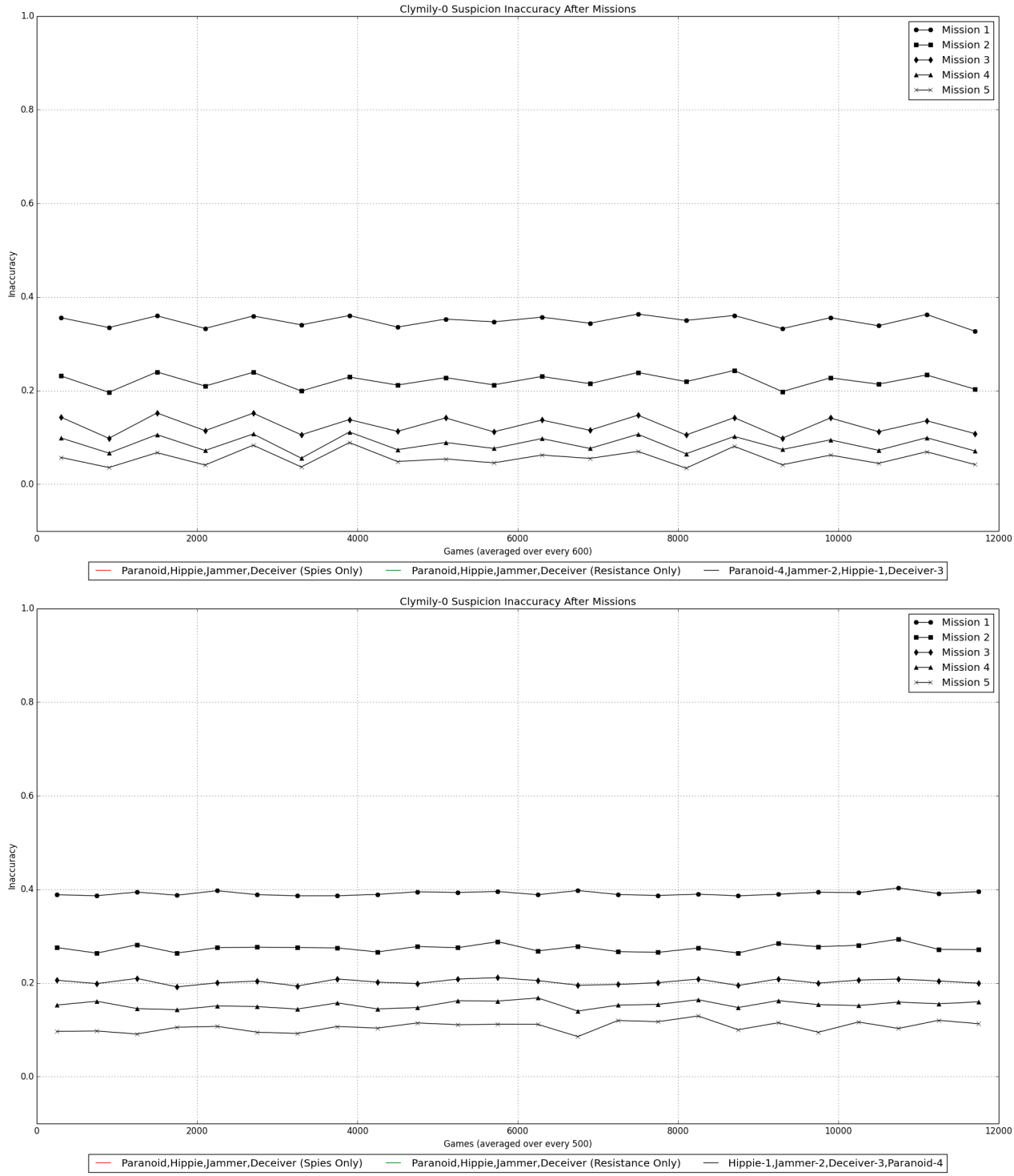

Figure 30: Suspicion inaccuracies of Clymily vs. beginners grouping, in burst competitions, burst sizes 1,200 and 100.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  PandSBot          45.2%          35.6%           15.2%
  Clymily           39.4%          40.9%           14.1%
  Rebounder         37.6%          26.7%           13.7%
  GarboA            35.3%          41.4%           13.3%
  Bot5Players       26.9%          36.3%           12.7%
RESISTANCE                      (vote,           select)
  PandSBot          68.7%          91.6% 68.8%     77.9%
  Clymily           64.8%          73.7% 93.6%     80.9%
  Rebounder         63.6%          98.8% 52.2%     75.4%
  GarboA            62.1%          86.9% 57.5%     66.8%
  Bot5Players       56.4%          94.8% 47.4%     67.5%
TOTAL
  PandSBot          59.3% (e=0.88 n=12000)
  Clymily           54.6% (e=0.89 n=12000)
  Rebounder         53.2% (e=0.89 n=12000)
  GarboA            51.4% (e=0.89 n=12000)
  Bot5Players       44.6% (e=0.89 n=12000)
```

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

Requested burst size: 100
Bursts played count: 120

SPIES                           (voted,          selected)
  PandSBot-1        59.4%          42.0%           25.1%
  Clymily-0         54.8%          63.0%            0.7%
  Rebounder-3       49.1%          30.3%           20.7%
  GarboA-2          45.3%          41.3%           16.6%
  Bot5Players-4     26.5%          43.9%           16.5%
RESISTANCE                      (vote,           select)
  PandSBot-1        61.2%          92.7% 54.7%     63.4%
  Clymily-0         58.2%          71.1% 83.5%     67.2%
  Rebounder-3       54.4%          99.4% 47.8%     90.6%
  GarboA-2          51.8%          86.2% 54.7%     64.8%
  Bot5Players-4     39.3%          99.9% 39.8%     85.8%
TOTAL
  PandSBot-1        60.5% (e=0.87 n=12000)
  Clymily-0         56.8% (e=0.89 n=12000)
  Rebounder-3       52.3% (e=0.89 n=12000)
  GarboA-2          49.2% (e=0.89 n=12000)
  Bot5Players-4     34.2% (e=0.85 n=12000)
```

Figure 31: Performance of Clymily against advanced opponent grouping, first in a regular competition, then in a burst competition, burst size 100.

5.7.2 Impact on the Vienna Results

Figure 32 shows the results from our reproduction of the final round of the Vienna competition's direct elimination tournament, along with the results from a burst competition, burst size 100, which was otherwise the same setup. Our reproduction of the Vienna competition results seems solid, with the win percentages under "TOTAL" being roughly equal to those found in [AiGameDev.com Team, 2012], and the order matching perfectly.

While the burst competition results differ significantly from the originals, the only competitors to have changed order (in all three categories, it happens) are Magi and Rebounder, with the OM agent Magi taking the higher rankings. Given the limitations we imposed on the OM agents in the burst competition - restricting the assembly of opponent models to 100 games - we had expected to see decreased performance in the OM agents, not the expert rule and configuration tracking based Rebounder, which should not be directly affected. The other non-OM agent, PandSBot also shows a greater difference in performance than the OM agents - it performs more strongly, as we might have expected if the OM agents were disadvantaged by the change.

```
Requested games: 25000
Comprehensive size: 1200
Comprehensive sets played: 20.833333

SPIES                           (voted,          selected)
  PandSBot          31.7%          31.4%            14.6%
  ScepticBot        26.3%          32.6%            13.9%
  Clymily           24.6%          36.0%            11.8%
  Rebounder         22.1%          26.2%            13.6%
  Magi              20.5%          26.8%            13.6%
RESISTANCE                      (vote,           select)
  PandSBot          79.4%          90.0% 67.3%      76.7%
  ScepticBot        75.8%          90.4% 69.9%      74.9%
  Clymily           74.7%          74.2% 93.1%      83.3%
  Rebounder         73.0%          98.9% 54.1%      71.6%
  Magi              71.9%          90.0% 57.7%      66.5%
TOTAL
  PandSBot          60.3% (e=0.61 n=25000)
  ScepticBot        56.0% (e=0.62 n=25000)
  Clymily           54.7% (e=0.62 n=25000)
  Rebounder         52.6% (e=0.62 n=25000)
  Magi              51.4% (e=0.62 n=25000)
```

```
Requested games: 25000
Comprehensive size: 1200
Comprehensive sets played: 20.833333

Requested burst size: 100
Bursts played count: 250

SPIES                           (voted,          selected)
  PandSBot-0        55.8%          48.7%             6.3%
  ScepticBot-1      33.1%          35.3%            34.6%
  Clymily-2         24.9%          13.0%            10.8%
  Magi-4            14.3%          16.3%             8.6%
  Rebounder-3        7.0%          12.0%            15.2%
RESISTANCE                      (vote,           select)
  PandSBot-0        92.2%          95.2% 84.6%      74.1%
  ScepticBot-1      77.1%          88.8% 67.5%      45.9%
  Clymily-2         71.6%          73.3% 97.6%      78.8%
  Magi-4            64.5%          89.7% 58.7%      80.8%
  Rebounder-3       59.7%          98.8% 52.9%      85.7%
TOTAL
  PandSBot-0        77.6% (e=0.52 n=25000)
  ScepticBot-1      59.5% (e=0.61 n=25000)
  Clymily-2         52.9% (e=0.62 n=25000)
  Magi-4            44.4% (e=0.62 n=25000)
  Rebounder-3       38.6% (e=0.60 n=25000)
```

Figure 32: Results from reproduction of the Vienna competition direct elimination, final round, then from the same setup with burst size 100.

6 Analysis and Evaluation

In this section we provide further exploration of the experimental results presented in Section 5, including considerations of the reasons behind those results, what conclusions we may draw from them, and additional figures for clarification.

6.1 Expert Rules Performance

While our own expert rule based bot, Ruru, only came third in the beginners match, the existing expert rules bot, Rule Follower, at least demonstrates that it is possible to beat such opponents with expert rules alone, despite using a much smaller number of rules (Section 5.3.1). Against the advanced grouping, although it is clear that expert rules are not enough for a strong competitor, Ruru is seen to be marginally stronger than RuleFollower due to greatly stronger spy play (Section 5.3.2).

Though we cannot be sure without running further experiments, it could be that certain expert rules do not function well against agents which play with low competence because the moves they make do not correspond with the behaviours those rules are designed to exploit or counter. By looking over Ruru's rules we are unable intuitively identify any examples of this.

What we can see though, is that Ruru has a number of rules intended to avoid accruing suspicion when it is a spy, such as approving missions on the fifth voting attempt, rejecting teams featuring two spies, and not sabotaging if every member of the team is a spy. Such rules are likely to have no benefit against the beginners opponent grouping, as none of those agents pay any attention to their opponents' behaviours, but may be useful against the more advanced opponents which allocate suspicion to actions and conduct logical deduction. RuleFollower has no such rules, which may explain why Ruru overtakes RuleFollower in spy performance against the advanced opponent grouping.

The poor performance of the beginner agent Deceiver in these experiments, when compared to that of RuleFollower, gives us an intuitive example, as the only difference between these two is that Deceiver will reject teams featuring both spies, while RuleFollower will approve them (Section 4.1.1). In the beginner competitions, a spy has nothing to gain by rejecting teams featuring both spies, as the other agents wouldn't notice even if both spies sabotaged, but if the vote fails, the opportunity to do so may be lost.

6.2 Opponent Modelling

6.2.1 Suspicion Inaccuracy Noise

Before further discussion of the suspicion inaccuracy gathered, will first address the issue of the noise present in this data. While we previously alluded to the possible causes of this (Section 5.2), we have not yet given an adequate assessment of the impact of this noise on our results.

To see this noise we must generate graphs of suspicion inaccuracy data which average over smaller sets of games, or even plot individual games. The worth of looking at individual games is questionable, as the random order in which games are played, non-determinism in agents' logic, and other circumstances affecting the observer are not known for any specific game. This is, after all, why we run such a large number of games in the first place, but we cannot get an accurate idea of the consistency of the observer's performance, game-by-game, without looking so closely.

Another question we might ask is how this noise impacts on our assessment of any bot as "learning" or "adapting" made based upon the graphs smoothed by averaging. While we believe that even an increase in average performance over the course of games played is evidence enough of learning in, for instance, the case of Clymily (Figure 17), there is also evidence of this at the game-by-game level: Figure 33 shows a zoomed subset of the data from two ineffective Stalker configurations (modelling all behaviours vs. beginners, and modelling only perfect information behaviours vs. advanced), with each demonstrating a short learning phase that starts with a larger variance of inaccuracies before settling into a smaller range, then maintained for the rest of the competition. Evidence of adaptation is also visible in data from Clymily's competition with beginners (Figure 35), where we can see that initially the inaccuracies are similar to those of Stalker's ineffective configurations, but quickly (after about 60 games) reach a point where a large portion of games are able to achieve perfect inaccuracy (0.0).

We might suggest that the cause of the noise is not purely the game-by-game circumstances and non-deterministic behaviours of agents, but also instability in the modelling due to over-learning from each action sampled. Initial, advantageous adaptation certainly seems to occur quickly, however if instability in the opponent model was caused by over-learning from each action, we might expect to see a reduction in noise over the course of the competition; Figure 34 shows no sign of this. Instability in the opponent model may also be caused by modelling behaviour which is non-deterministic or determined by parameters not corresponding to those modelled, essentially misinterpreting the reasons for an opponent's decision. This may explain the noise seen in the beginners competition of Figure 33, as these simple bots do not feature expert rules for all of the perfect information behaviours modelled by Stalker.

It is also important, though perhaps trivial, to note that the variance of values encountered in the successful example presented in Figure 35 is greater than that of the ineffective examples presented in Figure 33. However, the lower bound of this range is typically far lower in successful examples, while the upper bound shows little difference. We highlight this in order to draw attention to the fact that a greater range of values does not necessarily indicate poorer overall performance.

Furthermore, in successful examples such as Figure 35, higher inaccuracy values typically belong to missions 1, 2 and 3 (to a lesser extent), while in ineffective examples, such as Figure 33, we frequently see points for missions 4 and 5 peeking above these.

In closing, it appears that although noise may be a topic of worth for future investigation, maybe even providing some insights into bot performance, it does not appear to invalidate evidence of learning in data graphed over larger averages, and does not (for now at least) change our view of which bots demonstrated stronger performance than others.

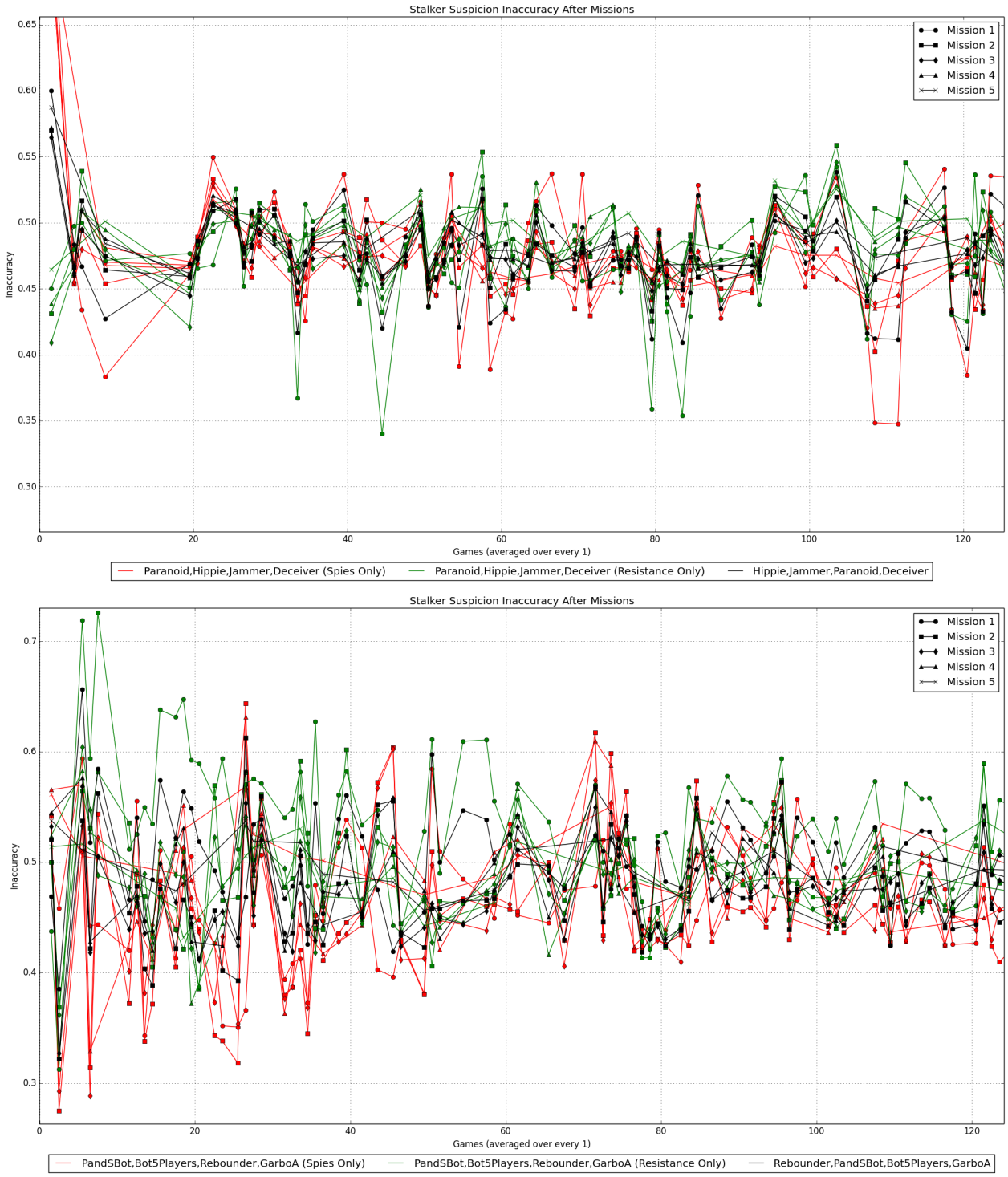

Figure 33: Suspicion inaccuracies of Stalker (all behaviours) vs. beginners and (only perfect information) vs. advanced groupings, zoomed.

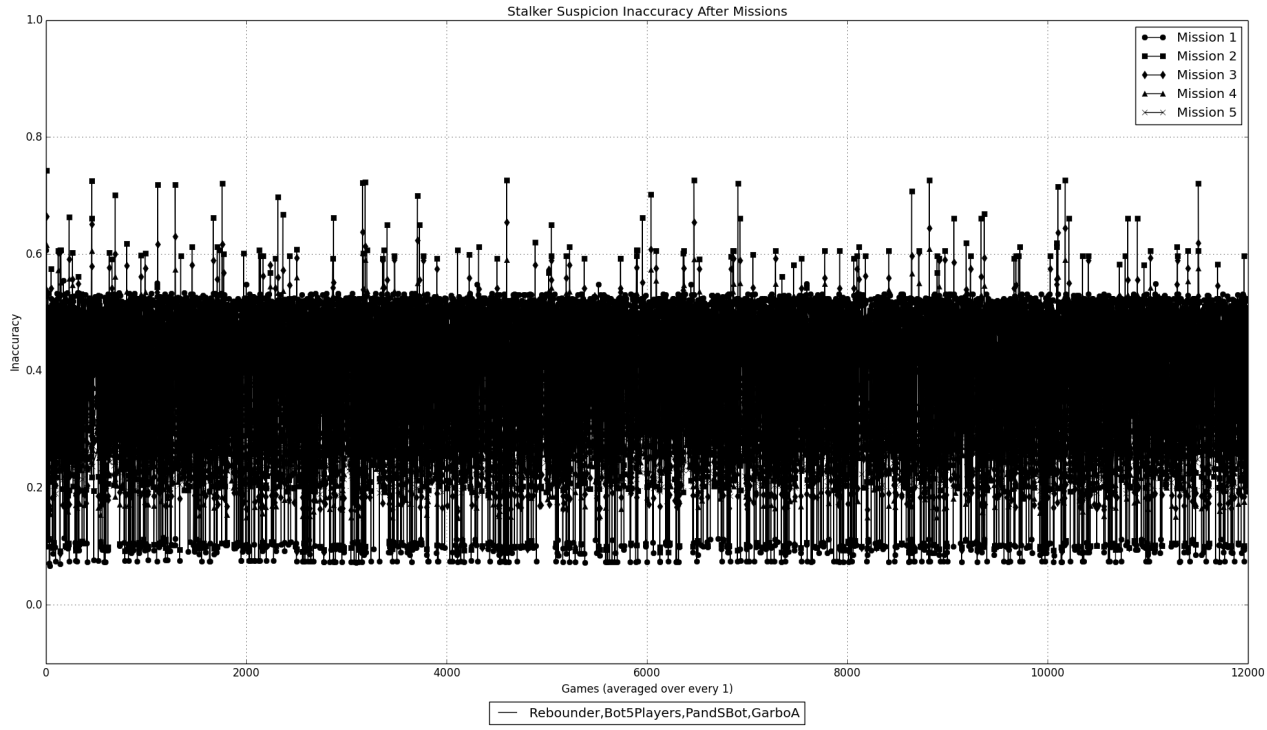

Figure 34: Suspicion inaccuracies of Stalker (only competence based) vs. advanced.

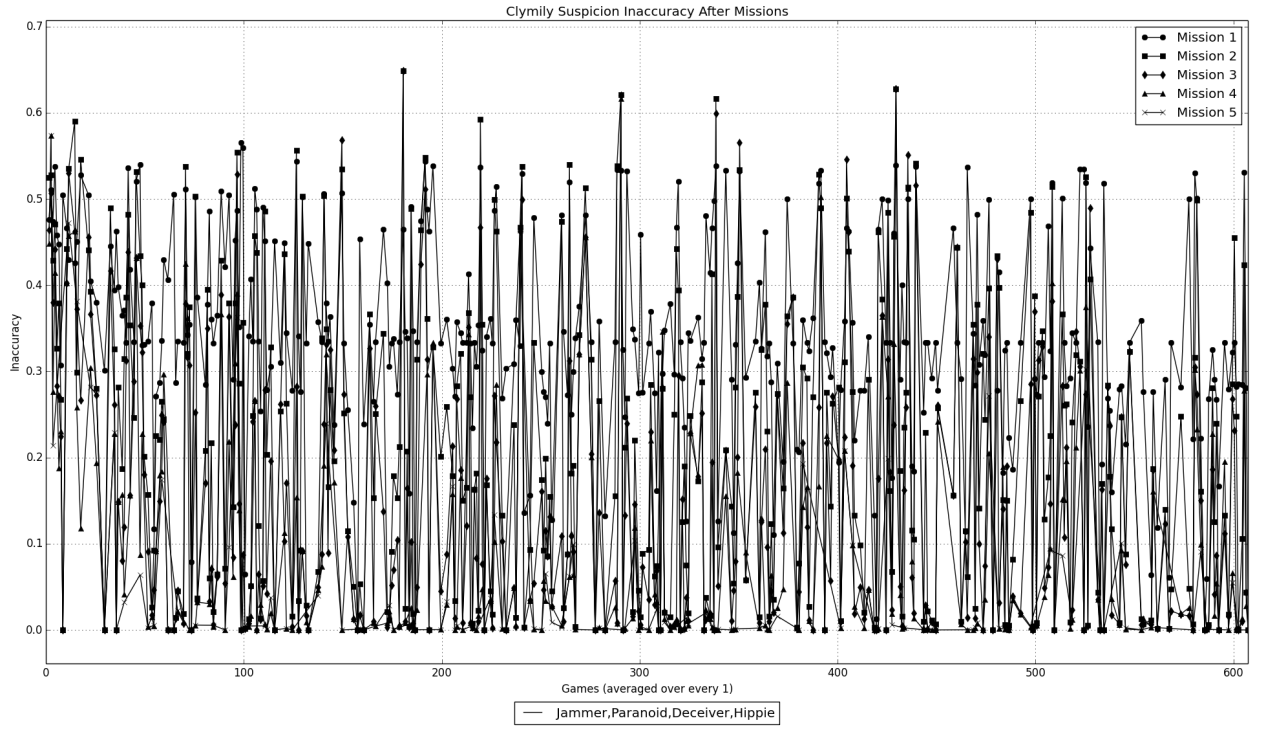

Figure 35: Suspicion inaccuracies of Clymily vs. beginners, zoomed.

6.2.2 Previous Opponent Modelling Inaccuracies

The results of the experiments presented in Section 5.4 do not give a strong indication of OM effectiveness for the existing agents Clymily, Magi, and ScepticBot, due to the presence of other systems which contribute to the suspicion scores calculated.

The only agent (of these) which we can say for certain benefits from its OM, at least in terms of suspicion score allocation, is Clymily, as we are able to observe evidence of gradual adaptation over the course of the competition (Figure 17). Further experiments demonstrated that the usefulness of learned opponent models in Clymily carries over to later competitions against the same opponents (Figure 18) due to model serialization, and can be disrupted by having Clymily compete in our burst competition model, where recognition of bot identities is restricted to an arbitrary number of games (Figure 31). Because of this, Clymily is able to achieve average inaccuracy scores which are superior to those of ScepticBot and Magi in competitions with both beginners and advanced opponent groupings.

By comparing the inaccuracies for these agents in competitions with the advanced grouping (Figures 17, 19, and 21) we might presume that Clymily would perform best of the three in competitions, closely followed by Magi, and then ScepticBot. However, as we mentioned in Section 4.6.2, we believe that the problem of acting upon suspicion scores is separate from that of calculating them, and so success or failure in one is not necessarily indicative of success or failure in the other. It is therefore unsurprising that, despite presumptions based on suspicion inaccuracy alone contradicting the results of the Vienna competition [AiGameDev.com Team, 2012], the performance of the bots themselves in these experiments (Figures 36, 37, and 38) corroborates them, with Clymily and ScepticBot almost equal, and Magi a short distance behind.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  PandSBot          51.9%          43.9%           19.7%
  ScepticBot        45.4%          41.5%           18.1%
  Rebounder         38.6%          34.4%           15.3%
  GarboA            38.0%          51.7%           14.0%
  Bot5Players       35.0%          47.2%           15.0%
RESISTANCE                      (vote,           select)
  PandSBot          65.0%          91.4% 66.8%      76.4%
  ScepticBot        60.7%          88.7% 66.4%      73.4%
  Rebounder         56.1%          98.7% 51.5%      69.3%
  GarboA            55.7%          83.1% 54.8%      60.2%
  Bot5Players       53.7%          95.3% 47.7%      63.4%
TOTAL
  PandSBot          59.7% (e=0.88 n=12000)
  ScepticBot        54.6% (e=0.89 n=12000)
  Rebounder         49.1% (e=0.89 n=12000)
  GarboA            48.6% (e=0.89 n=12000)
  Bot5Players       46.2% (e=0.89 n=12000)
```

Figure 36: Performance of ScepticBot against advanced opponent grouping.

Another idea supporting the separation of suspicion calculation and application problems is that the specific magnitude of suspicion does not matter. Given a situation where the inaccuracy of the suspicion score for every player is below 0.5, the suspicion scores for spy players are guaranteed

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  PandSBot          45.2%          35.6%            15.2%
  Clymily           39.4%          40.9%            14.1%
  Rebounder         37.6%          26.7%            13.7%
  GarboA            35.3%          41.4%            13.3%
  Bot5Players       26.9%          36.3%            12.7%
RESISTANCE                      (vote,           select)
  PandSBot          68.7%          91.6% 68.8%      77.9%
  Clymily           64.8%          73.7% 93.6%      80.9%
  Rebounder         63.6%          98.8% 52.2%      75.4%
  GarboA            62.1%          86.9% 57.5%      66.8%
  Bot5Players       56.4%          94.8% 47.4%      67.5%
TOTAL
  PandSBot          59.3% (e=0.88 n=12000)
  Clymily           54.6% (e=0.89 n=12000)
  Rebounder         53.2% (e=0.89 n=12000)
  GarboA            51.4% (e=0.89 n=12000)
  Bot5Players       44.6% (e=0.89 n=12000)
```

Figure 37: Performance of Clymily against advanced opponent grouping.

```
Requested games: 12000
Comprehensive size: 1200
Comprehensive sets played: 10.000000

SPIES                           (voted,          selected)
  PandSBot          55.7%          52.1%            23.1%
  Magi              42.4%          38.6%            22.0%
  Rebounder         41.2%          39.6%            17.0%
  GarboA            35.9%          49.3%            11.4%
  Bot5Players       27.5%          49.5%            10.2%
RESISTANCE                      (vote,           select)
  PandSBot          69.6%          90.6% 66.9%      77.6%
  Magi              60.7%          91.6% 55.8%      66.5%
  Rebounder         59.9%          99.4% 49.4%      70.4%
  GarboA            56.4%          88.2% 55.6%      64.0%
  Bot5Players       50.8%          95.7% 46.0%      64.4%
TOTAL
  PandSBot          64.0% (e=0.86 n=12000)
  Magi              53.4% (e=0.89 n=12000)
  Rebounder         52.4% (e=0.89 n=12000)
  GarboA            48.2% (e=0.89 n=12000)
  Bot5Players       41.5% (e=0.88 n=12000)
```

Figure 38: Performance of Magi against advanced opponent grouping.

to be higher than those of the resistance members, so basing decisions on list of players sorted by their suspicion scores will be as successful with inaccuracies of 0.46 as it would with inaccuracies of 0.2, etc. This may explain why ScepticBot is able to outperform Magi, despite having higher average inaccuracies (being "less certain", if we may); both Clymily and ScepticBot make use of player lists sorted by suspicion score, most notably for their team selections which both appear stronger than Magi's in the competition output.

Finally we wish to draw a connection between Magi's poor performance against our beginners

opponent grouping, and its poor performance in the first round of the Vienna competition, which involved beginner (RuleFollower, Jammer) and Intermediate (Statistician, LogicalBot, Bounder) agents. Figures 20 and 21 demonstrate average spy inaccuracies which increase over mission progression, passing above 0.5, although low inaccuracies for resistance members pull the average down to around 0.4. With regards to the previous discussion, these figures impress that it is not enough for the average inaccuracy for all players to be beneath 0.5; inaccuracy for all opponents must be beneath 0.5 to guarantee correct ordering.

6.2.3 Effective Behaviours for Modelling

Section 5.5 presents data on the suspicion inaccuracies of our pure OM agent, Stalker. While we tested a number of configurations (detailed in Section 4.7.3) and sought to group similar behaviours together into categories (Section 4.6.2), it is possible that each configuration's performance is not representative of every behaviour it includes. Our implemented behaviours are also not a comprehensive representation of those it is possible to model. Therefore, we would encourage further investigation into modelled behaviours, possibly even examining the usefulness of behaviours one-by-one. Closer inspection of the models generated rather than the suspicion scores generated by fitting opponents too them may also prove useful.

Using the average inaccuracies over all players (black) as our measure, regarding the graphs presented for Stalker's performance against the advanced opponent grouping (Figures 23 to 27), we find the results of these experiments quite surprising. Results from the experiment which modelled opponents based only on the perfect information behaviours suggest that this configuration performs about as poorly against the advanced opponent grouping as every configuration did against the beginners.

On the other hand, the configuration which models based only on the competence based behaviours produces inaccuracies comparable to those of ScepticBot, despite our concerns that these methods would allocate too much suspicion to resistance members for poor decisions made due to incompetence or lack of information. The next best performance to this, perhaps most surprisingly, is the configuration which models opponents based only on the "teamOnIsUnsuccessful" behaviour, which although we classify as a competence based behaviour in fact represents a compromise for the difficulties of modelling category 3, partially observable action behaviours. The compromise is made by essentially recording a sabotage action for every player in the team of a sabotaged mission, both for the current game history and the model frequency. This will likely cause "teamOnIsUnsuccessful" to allocate too much suspicion to unfortunate resistance members, similarly to the other competence based behaviours.

Figures 25 and 26 support our concerns by presenting significantly higher inaccuracies for resistance members than spies. In fact, when inaccuracies for spies and resistance members are almost mirrored around 0.5 in this way, this suggests that the agent is unintelligently allocating lots of suspicion to everyone. Referring back to our discussion in Section 6.2.2, we might make a further argument that these two configurations are not as strong as they first appear, as having such high resistance member suspicion inaccuracies allows suspicion scores for resistance members to be higher than those for spies, making correct decision making more difficult. Magi's graph for the beginners opponent grouping (Figure 21) shows the opposite problem, with low suspicions being allocated to both resistance members and spies.

With this consideration in mind, if we look for the configuration which has the lowest inaccuracies over all players while maintaining inaccuracies for spies and resistance members which

are mostly beneath 0.5, then perhaps the configuration modelling all behaviours apart from "teamOnIsUnsuccessful" appears to be the strongest. We would also suggest, however, that this only appears so because the inclusion of the perfect information behaviours results, all around 0.5, pulls the averages in towards the middle, where resistance inaccuracies were otherwise a shorter distance above than spy inaccuracies were below; this is not, therefore, evidence that modelling these perfect information behaviours is, in itself, useful.

The apparent ineffectiveness of modelling perfect information behaviours is puzzling. Since these behaviours often relate to expert rules catering to specific scenarios, such as rejecting teams of three not featuring yourself, perhaps the problem is that these expert rules do not come into play as often as the competence based behaviours. A better application of these frequencies then may be in prediction of opponent actions, as Clymily does, and identification of exploitable opponents, similarly to [Schweizer et al., 2009].

Finally, we should consider the ineffectiveness of Stalker against the beginners opponent grouping. For this it is tempting to refer back to our suggestion that instability in opponent models may be caused by modelling behaviours which are determined, in the subject agent's logic, by parameters not corresponding to those modelled (Section 6.2.1). However, Clymily and ScepticBot were able to maintain strong performance against the beginners grouping, with Clymily even displaying a stronger indication of gradual adaptation than against the advanced grouping. ScepticBot is not a suitable comparison as there is no clear indication of the benefit of its OM amongst the other systems, and differences from Clymily's modelled behaviours are difficult to assess due to its complex implementation.

6.2.4 Effectiveness of Grouped Opponent Modelling

While data presented in Section 5.6 suggests that our grouped OM agent, Nosey, was just as effective as our pure OM agent, the data from each set of experiments is not clear enough to judge whether grouped opponent models allow for faster learning.

Still, we believe that merely matching the performance of per opponent OM is enough to warrant further investigation, perhaps beginning with testing Nosey against a more varied group of opponents, and testing transferral of knowledge from group of opponents to another. Clymily too may yeild interesting results if modified to build opponent models which are generalized over all opponents, since Clymily has shown the clearest signs of learning out of all of our experiments (Section 6.2.2).

6.3 Burst Competition Impact

6.3.1 Impact on Clymily

In Section 5.7.1 we presented data on Clymily's performance in an environment where the number of games allowed for the construction of opponent models is limited.

Despite a clear change in the inaccuracy graphs, and significantly poorer play as a resistance member, the overall win percentages and rankings for these competitions were not effected by this environment. Clymily was able to cancel out the decrease in resistance member performance with an increase in spy performance for which we do not know the reason.

6.3.2 Impact on Vienna

In Section 5.7.2 we presented the results from our reproduction of the final round of the Vienna competition's direct elimination tournament and compared these to results from the same setup in a burst competition with burst size 100. Our reproduction seems solid, however the effects of the burst competition are once more difficult to interpret. Although PandSBot's improved performance may be a result of weaker performance by the OM agents, the only change in rankings shows an OM agent overtaking a non-OM agent.

There could be any number of possible reasons for this, and overall we feel that the number of variables in this experiment is far too high to draw any conclusions. When introducing a limit like this which is expected to impact on the performance of three out of five competitors, it is understandable that the results are difficult to interpret. This will likely be a common problem when investigating "The Resistance", due to the large number of players and strong focus on the opponents, as discussed in Section 2.2, and evidenced by our results, where agent performance is shown to vary greatly based on the opponents faced.

6.3.3 Significance

One possible interpretation of the small, unpredictable changes seen in OM agents' performance when entered into the burst competition model is that the OM is not a strong factor in their overall performance, and instead they are carried by the other systems in place.

This would seem all the more likely if the large amount of noise encountered in our suspicion inaccuracy graphs, discussed in Section 6.2.1, is a result of instability in the opponent models or fittings to them.

So that we can better identify the cause of this noise, further experiments should be run with an agent which combines OM and other techniques, then with that same agent's OM and other techniques tested separately, making sure that each configuration produces comparable suspicion scores.

7 Conclusion

In this paper we have introduced the modern board game "The Resistance" as a challenging environment for research into game playing AI. We have provided an overview of the game's rules, highlighting areas which differentiate it from other board games, modern or classical, and identifying those aspects which make it an interesting problem (Section 2).

A detailed exploration of current AI techniques was presented in the literature review (Section 3), using related games such as Go, Settlers of Catan, and Poker as a reference point, and drawing comparisons to these games with consideration of how techniques may transfer to "The Resistance".

We then conducted an analysis of a number of existing agents (Section 4.1), identifying common expert rules and other techniques employed. Amongst the current collection there are a number of strong competitors, and though the types of techniques applied are few, the implementations are quite varied. Certain techniques clearly dominate, however. All five agents present in the final round of the Vienna conference competition [AiGameDev.com Team, 2012] feature reasoning based on the tracking of possible configurations of player roles. Three of these agents also use opponent modelling techniques, however the first place bot does not do so, and we are unable to confirm the value of the opponent modelling in the presence of the other techniques employed.

Our preliminary investigation into expert rules in "The Resistance" is not thorough, but analysis of the results (Section 6.1) demonstrates that while expert rules can be effective against simple opponents the presence certain rules may interfere with an agent's ability to exploit them. This is due, for example, to some rules passing up opportunities for sabotage which may cause other agents to be suspicious of them, even when the opponents they face do not pay any attention to mission results. There is some evidence, however (Section 6.1) that such caution is effective in play against more advanced opponents. For this reason we do not believe that competitions against simple opponents are an effective method of assessing the overall value of a rule, nor competence of an agent in general.

While we identified a number of techniques which might be applied to "The Resistance" in our literature review, and further ideas based on analysis of existing agents, we decided to focus on analysing the usefulness of opponent modelling (Section 3.6), a technique already attempted in "The Resistance" by several bots. To facilitate this, we contribute a number of original implementations, including a modular, opponent modelling agent (Section 4.6.2), grouped opponent modelling variant (Section 4.6.3), and scripts for logging and graphing "suspicion inaccuracy" scores (Section 4.5).

Implementation considerations (Section 4.6.2) and experimental results (Section 5.5) demonstrate the importance of one of the biggest challenges in "The Resistance": allocating trust or suspicion amongst agents whose own knowledge of the game state may not be complete (based on what we call "competence based" behaviours) and whose actions are not fully observable. Figures 25 and 27, seem, at a glance, to indicate strong performance, but also demonstrate the tendency of modelling such behaviours to allocate too much suspicion to resistance members, which may be detrimental.

The results of our suspicion inaccuracy graphing seem surprising in some cases, seeming to indicate stronger performance from modelling competence based and partially observable action behaviours than those based on expert rules (Section 6.2.3), but they are also not trivial to interpret. In our analysis we discussed the possible interpretations of noise found in these results (Section 6.2.1), and although we conclude that its presence does not necessarily invalidate interpretations

of graphs averaged over large numbers of games, it is still worth bearing in mind. Overall, we believe that examining suspicion inaccuracy scores has allowed for interesting insights into agent performance and the game itself, though it is yet difficult to draw any solid conclusions from the results.

Finally, we examined the impact of restricted modelling time for OM agents by modifying the original competition model to generate new names for each participating agent at arbitrary intervals (Sections 4.4.2, 5.7 and 6.3). The burst competition model has not proved especially useful in our investigation; while the impact on signs of learning in suspicion inaccuracy graphs is clear and predictable (Figure 30), the impact on competition results both in Clymily's case, and on the outcome of our Vienna competition reproduction (Section 5.7.2) is not clear. We have suggested that the reason for this may be that the usefulness of OM is not a strong factor in the performance of existing agents (Section 6.3.3). If this is the case then burst competitions may yet prove useful in future work, possibly in assessing agents' capacity for rapid adaptation.

7.1 Directions For Future Work

In Section 4.3 we highlighted a number of possible directions for future work based on our analysis of existing agents and techniques explored in the literature review:

Monte-Carlo Tree-Search was identified as a possible avenue for exploring game tree based solutions in "The Resistance" since it has shown to be effective in other complex board games (Go [Coulom, 2006]), games involving multiple opponents (Settlers of Catan [Szita et al., 2009]), and games involving imperfect information (Poker, usually integrated with OM techniques). For MCTS to work in "The Resistance" we believe that injection of domain knowledge will be critical due to the importance of other players' identities. While it is possible to use OM data for this [Gerritsen, 2010; Ponsen et al., 2010; Schweizer et al., 2009; van der Kleij, 2010], for simplicity and clarity of results we would recommend first trying MCTS with values from hand-coded suspicion calculations injected as domain knowledge.

Inspired by the modular structures of agents such as Magi and Bot5Players, we also suggested that future research may attempt to apply computer learning techniques, such as reinforcement learning [Pfeiffer, 2004] or co-evolution [Pollack and Blair, 1998], to "The Resistance" in order to determine the most effective suspicion score modifications, expert rules, or even behaviours modelled by OM agents.

From the results of our own work we can identify yet more posabilities for future research:

Additional work is required to determine the relationship between the accuracy of suspicion scores, calculated through OM or other techniques, and strong decision making. This may involve running experiments with varying decision logic or artificial modification of the suspicion scores to varying degrees. The former method would also help to verify suggestions made in Section 6.2.2 about the power of decisions based on lists of players ordered by suspicion score when inaccuracies for all players are below 0.5.

Interpretation of the inaccuracy graphs themselves may even be worthy of investigation - or perhaps development of better techniques for observing OM effectiveness. Further analysis of the inaccuracies and noise present, considering the mean and standard deviation of each group of games averaged over, may allow for more confident analysis of suspicion inaccuracy results. We should also seek to identify the cause of this noise - specifically, whether it is unique to OM agents or also a factor in other techniques. Whether this noise is a result of OM instability, or just a component of the game caused by random initial setup and particular circumstances, the development of methods

which reduce it may lead to overall better performance.

Putting aside the relation of suspicion inaccuracy to actual performance once more, a thorough investigation of behaviours and granularities modelled by OM techniques may also be justified: While we tested a number of configurations of Stalker and drew general conclusions about what categories of behaviours seemed effective, these conclusions may not hold for specific behaviours within those configurations, especially those we have not implemented. Other works may seek to assess behaviours one by one, rather than in categories. Also, although we showed that modelling at a coarse granularity - all opponents sharing one model - seems effective, this requires confirmation by testing with a more varied selection of opponents, after which we may consider investigating the effectiveness of a model of one group of opponents when transferred to another.

Finally, though perhaps more valuable against simple bot implementations rather than human players, future work might consider the identification of exploitable behaviours in opponents, similarly to [Schweizer et al., 2009]. For example, identifying bots which do not conduct any logical deduction or suspicion allocation, such as those in our beginners opponent grouping, would allow our agent to be less cautious in its voting and sabotaging as a spy.

References

AiGameDev.com Team [2012], 'The resistance ai competition', http://files.aigamedev.com/coverage/GAIC12_AiGameDevResistanceCompetition.pdf.

Avontuur, T. [2010], 'Opponent modelling in wargus', http://arno.uvt.nl/show.cgi?fid=113484.

Billings, D., Davidson, A., Schauenberg, T., Burch, N., Bowling, M., Holte, R., Schaeffer, J. and Szafron, D. [2006], 'Game tree search with adaptation in stochastic imperfect information games (paper)', http://webdocs.cs.ualberta.ca/~jonathan/publications/ai_publications/cg04poker.pdf.

Branca, L. and Johansson, S. J. [2007], 'Using multi-agent system technologies in settlers of catan bots', http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.130.9829&rep=rep1&type=pdf.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S. [2012], A survey of monte carlo tree search methods, *in* 'IEEE TRANSACTIONS ON COMPUTATIONAL INTELIGENCE AND AI IN GAMES, VOL 4, NO. 1'.

Brügmann, B. [1993], 'Monte carlo go'.

Burch, N. and Holte, R. C. [2011], Automatic move pruning in general single-player games, *in* 'In: Proceedings, The Fourth International Symphosium on Combinatorial Search', AAAI Press.

Chaslot, C., Bakkes, S., Szita, I. and Spronck, P. [2008], *Monte-Carlo Tree Search: A New Framework for Game AI*, BNAIC.

Coulom, R. [2006], Efficient selectivity and backup operators in monte-carlo tree search, *in* 'In: Proceedings Computers and Games 2006', Springer-Verlag.

Fernandes, H. M. P. L. [2012], 'Guiding monte carlo tree search simulations through bayesian opponent modeling in the octagon theory', http://www.gamedev.net/index.php?app=core&module=attach&section=attach&attach_id=17852.

Fjell, M. S. [2012], 'Opponent modeling and strategic reasoning in the real-time strategy game starcraft', http://urn.kb.se/resolve?urn=urn:nbn:no:ntnu:diva-18449.

Gelly, S. and Wang, Y. [2006], Exploration exploitation in go: Uct for monte-carlo go, *in* 'Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop'.

Gerritsen, G. [2010], 'Combining monte-carlo tree search and opponent modelling in poker'.

Graham, C. and Talay, D. [2013], 'Strong law of large numbers and monte carlo methods'.

Gusmao, A. and Raiko, T. [2012], 'Towards generalizing the success of monte-carlo tree search beyond the game of go', http://users.ics.aalto.fi/praiko/papers/ecai2012gusmao.pdf.

Hiob, E. [2007], 'Finding areas using the monte carlo method', http://commons.bcit.ca/math/entertainment/monte/.

Monin, J. D. [2013], 'Jsettlers2 - java settlers of catan', http://sourceforge.net/projects/jsettlers2/.

Pfeiffer, M. [2004], 'Reinforcement learning of strategies for settlers of catan', http://www.igi.tugraz.at/pfeiffer/documents/LAG-37_pfeiffer_2004.pdf.

Pollack, J. B. and Blair, A. D. [1998], 'Co-evolution in the successful learning of backgammon strategy', http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.9215&rep=rep1&type=pdf.

Ponsen, M., Gerritsen, G. and Chaslot, G. [2010], 'Integrating opponent models with monte-carlo tree search in poker'.

Ponsen, M., Ramon, J., Croonenborghs, T., Driessens, K. and Tuyls, K. [2008], 'Bayes-relational learning of opponent models from incomplete information in no-limit poker'.

Saleem, H. and Natiq, R. R. [2008], 'A multi-agent player for settlers of catan - a mas player in game playing', http://www.bth.se/fou/cuppsats.nsf/all/5182dee54d5efc33c1257559004f8dbc/$file/SOC_Final_Report.pdf.

Schadd, F. [2007], 'Hierarchical opponent models for real-time strategy games'.

Schweizer, I., Panitzek, K., Park, S.-H. and Furnkranz, J. [2009], 'An exploitative monte-carlo poker agent'.

Southey, F., Bowling, M., Larson, B., Piccione, C., Burch, N., Billings, D. and Rayner, C. [2005], 'Bayes' bluff: Opponent modelling in poker'.

Spronk, P., Cowling, P., Champandard, A., Lanzi, P. L. and Paiva, A. [2012], *AI for Modern Board Games*, Dagstuhl Publishing, Dagstuhl, Germany.

Strong, G. [2011], 'The minimax algorithm', https://www.cs.tcd.ie/Glenn.Strong/3d5/minimax-notes.pdf.

Szita, I., Chaslot, G. and Spronck, P. [2009], *Monte-Carlo Tree Search in Settlers of Catan*, Springer, Pamplona, Spain.

van der Kleij, A. [2010], 'Monte carlo tree search and opponent modeling through player clustering in no-limit texas hold'em poker'.

Wu, L. and Baldi, P. [2007], *A Scalable Machine Learning Approach to Go*, MIT Press.

Appendices

A Additional Suspicion Inaccuracy Graphs

In this appendix, for completion, we present suspicion inaccuracy graphs previously omitted from Section 5 to avoid cluttering it with data which would not add anything to the discussion.

A.1 Stalker vs. Beginners

These graphs represent the suspicion inaccuracy scores for various configurations of our pure opponent modelling agent, Stalker, against the beginners opponent grouping.

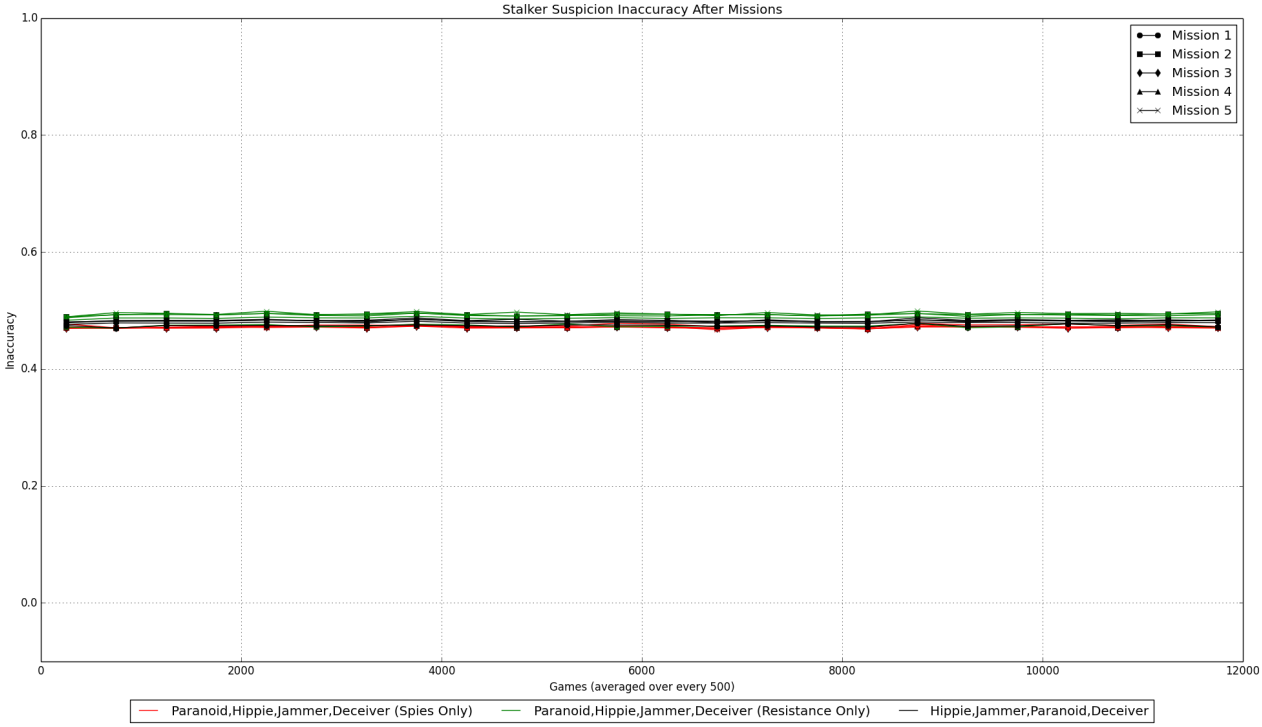

Figure A.1: Configuration: All behaviours

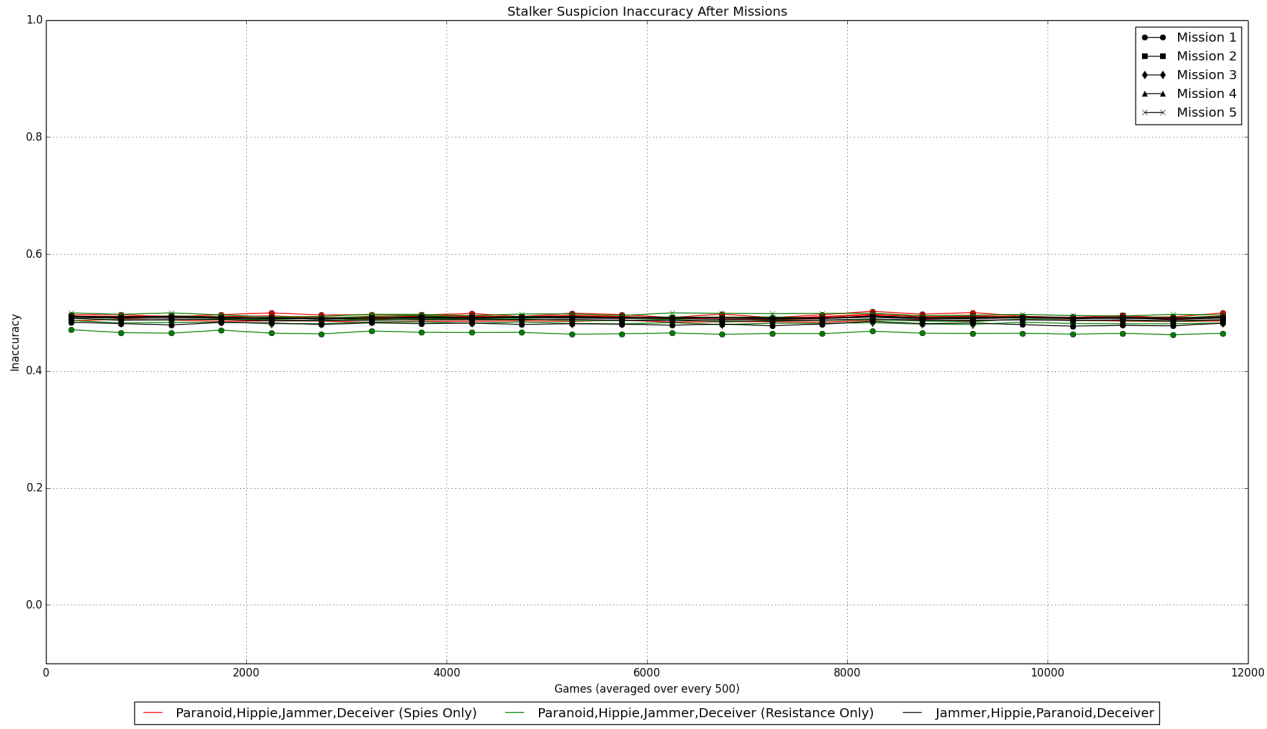

Figure A.2: Configuration: Only Perfect Information Behaviours

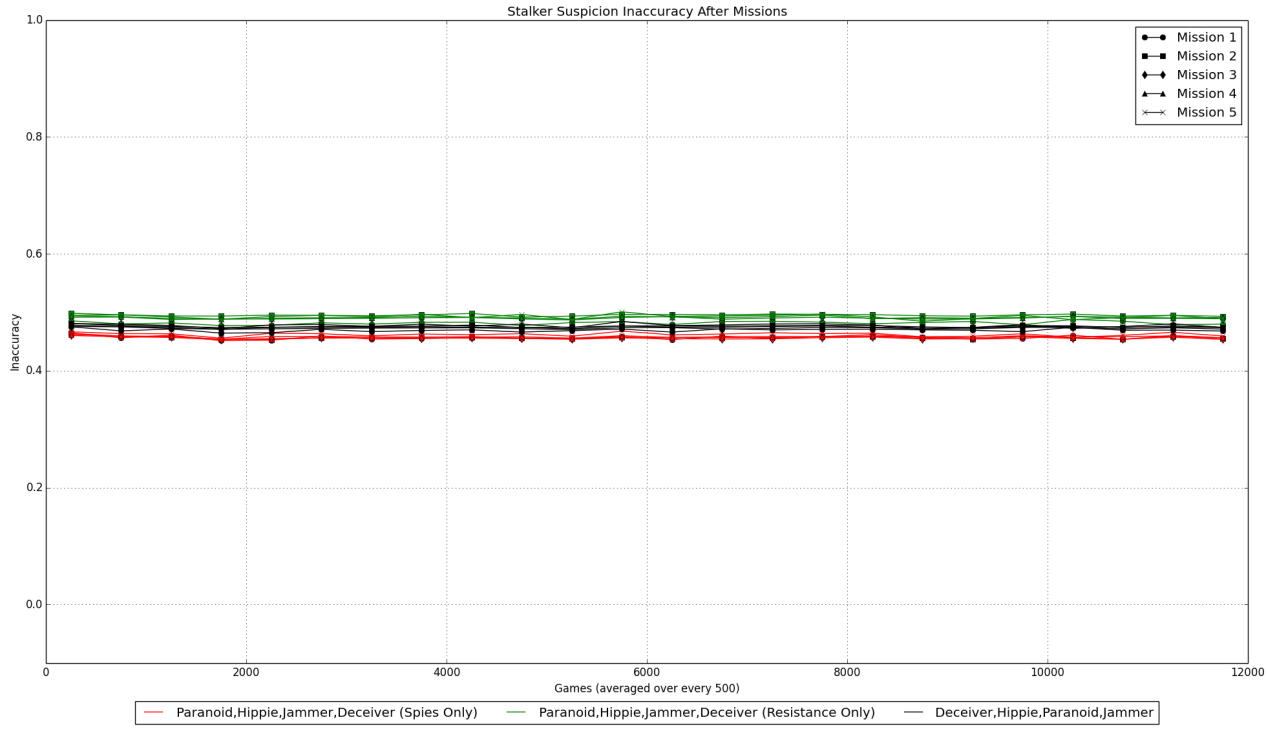

Figure A.3: Configuration: Only Competence Based Behaviours

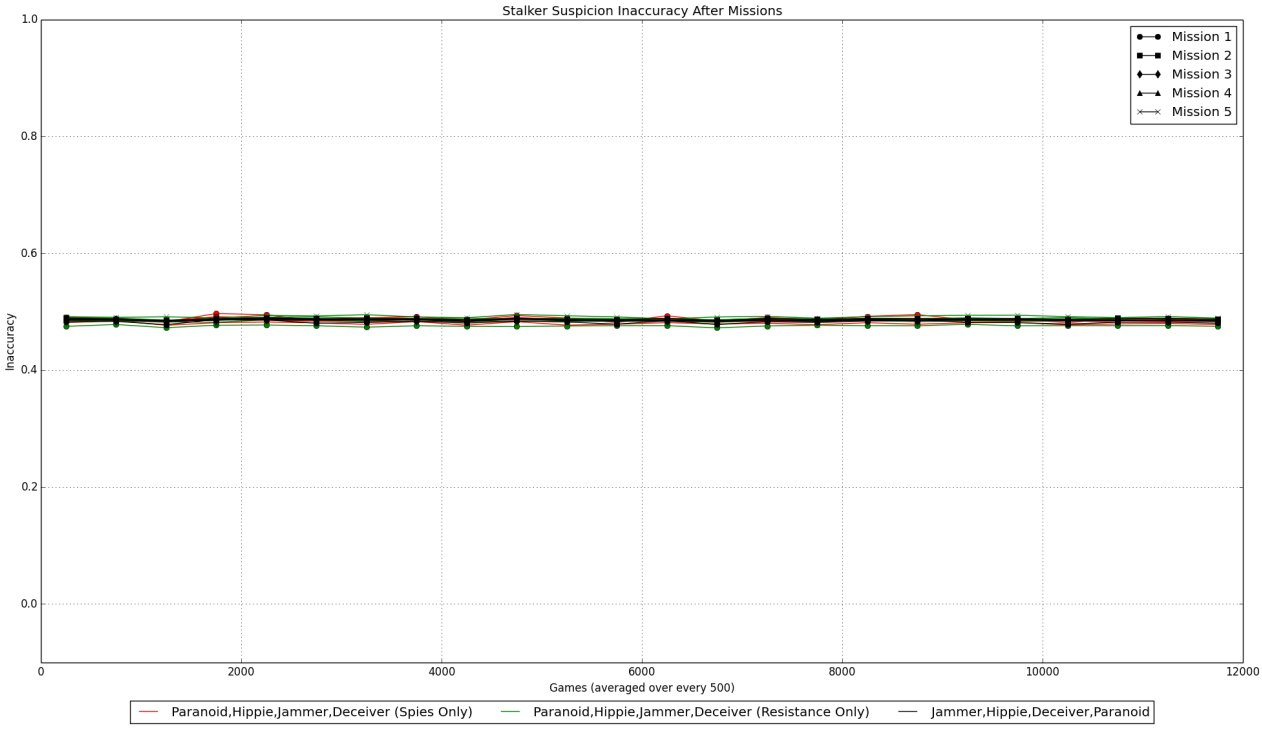

Figure A.4: Configuration: All Behaviours Except teamOnIsUnsuccessful

A.2 Nosey vs. Beginners

These graphs represent the suspicion inaccuracy scores for various configurations of our grouped opponent modelling agent, Nosey, against the beginners opponent grouping.

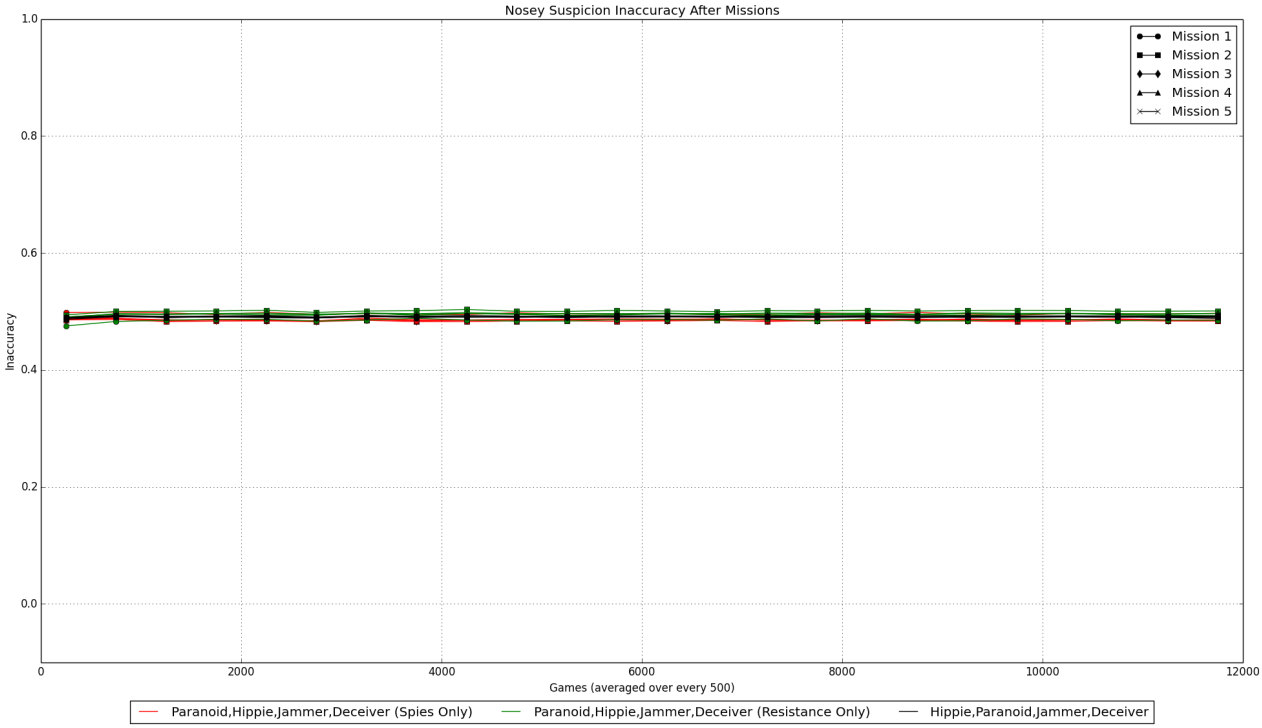

Figure A.5: Configuration: All behaviours

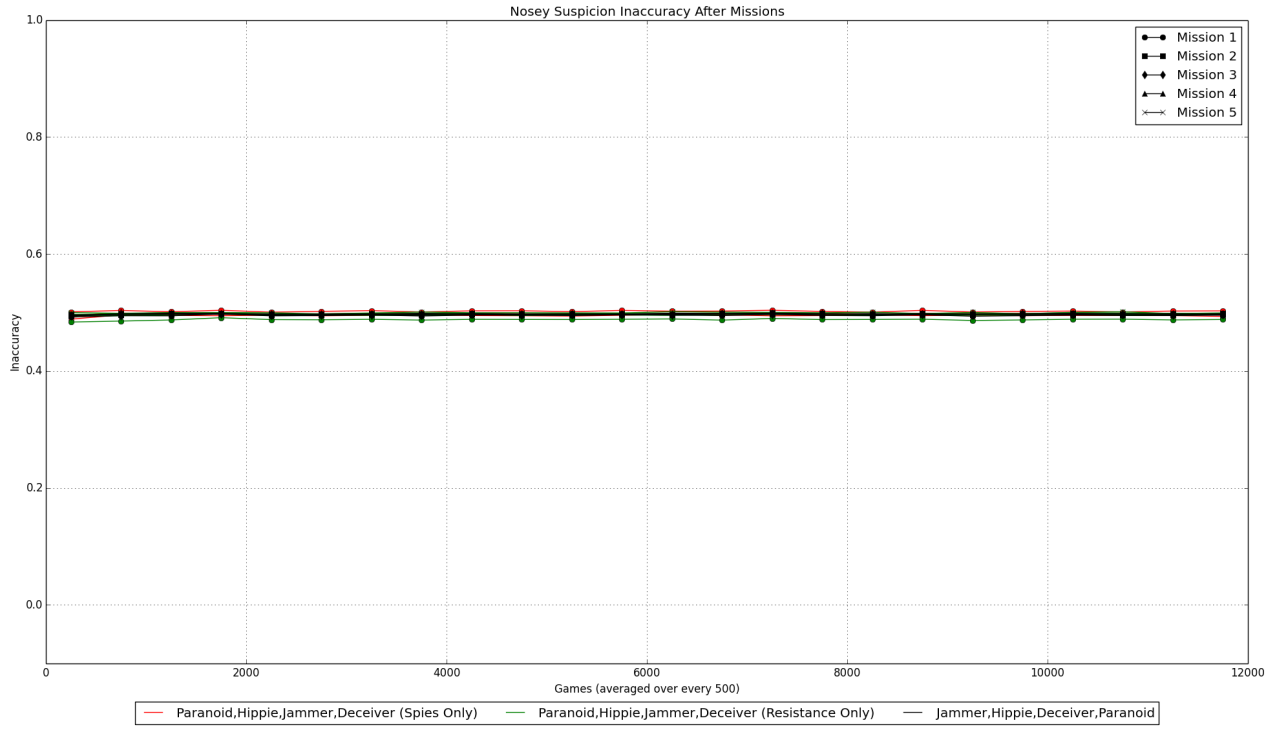

Figure A.6: Configuration: Only Perfect Information Behaviours

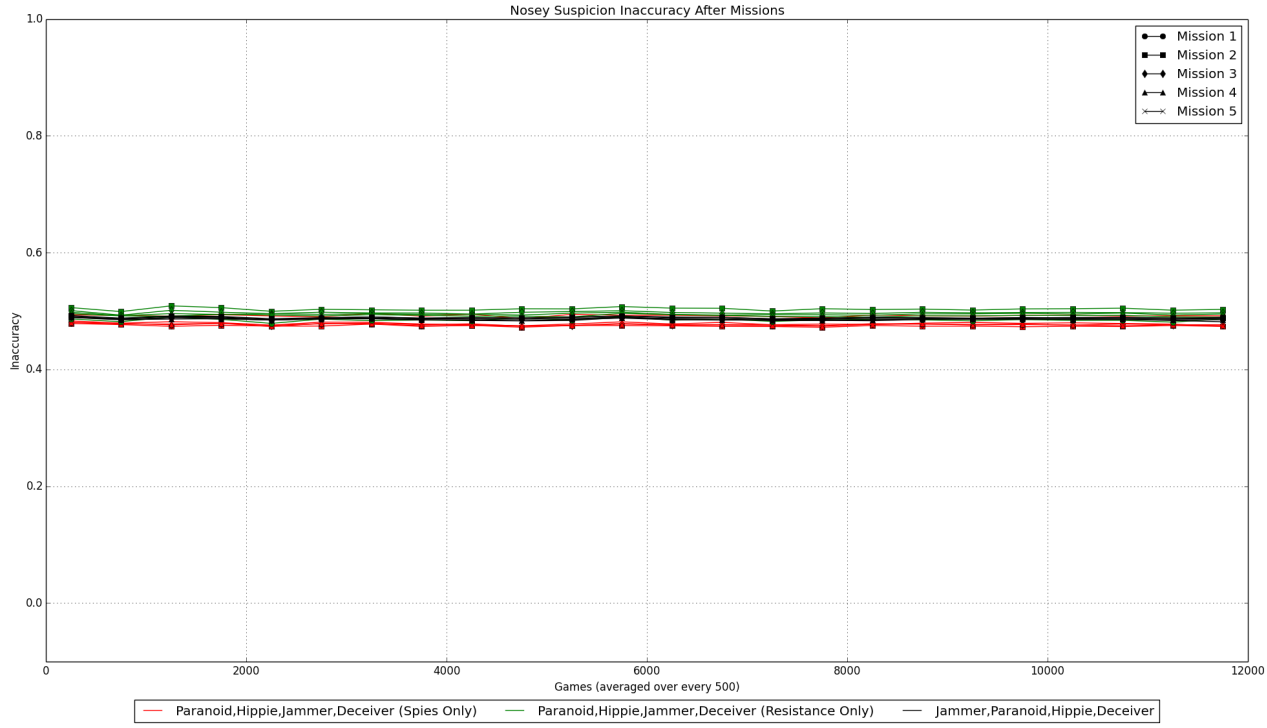

Figure A.7: Configuration: Only Competence Based Behaviours

A.3 Nosey vs. Advanced

These graphs represent the suspicion inaccuracy scores for various configurations of our grouped opponent modelling agent, Nosey, against the advanced opponent grouping.

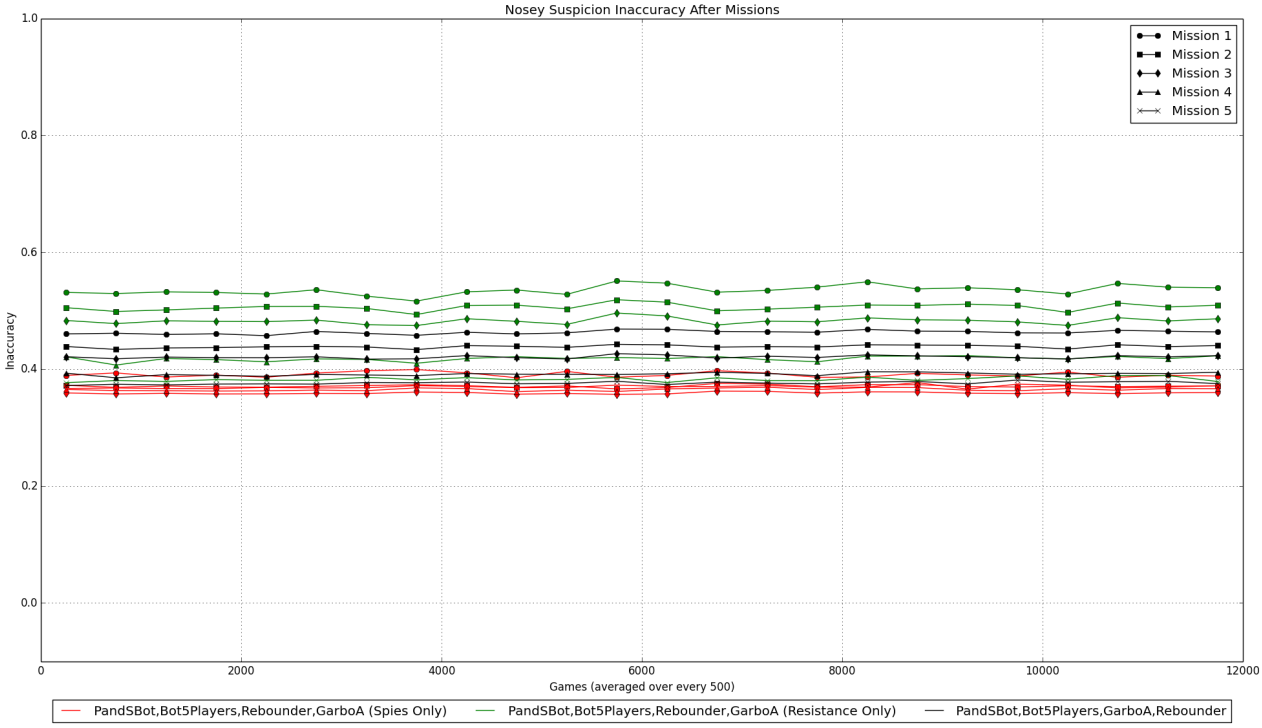

Figure A.8: Configuration: All behaviours

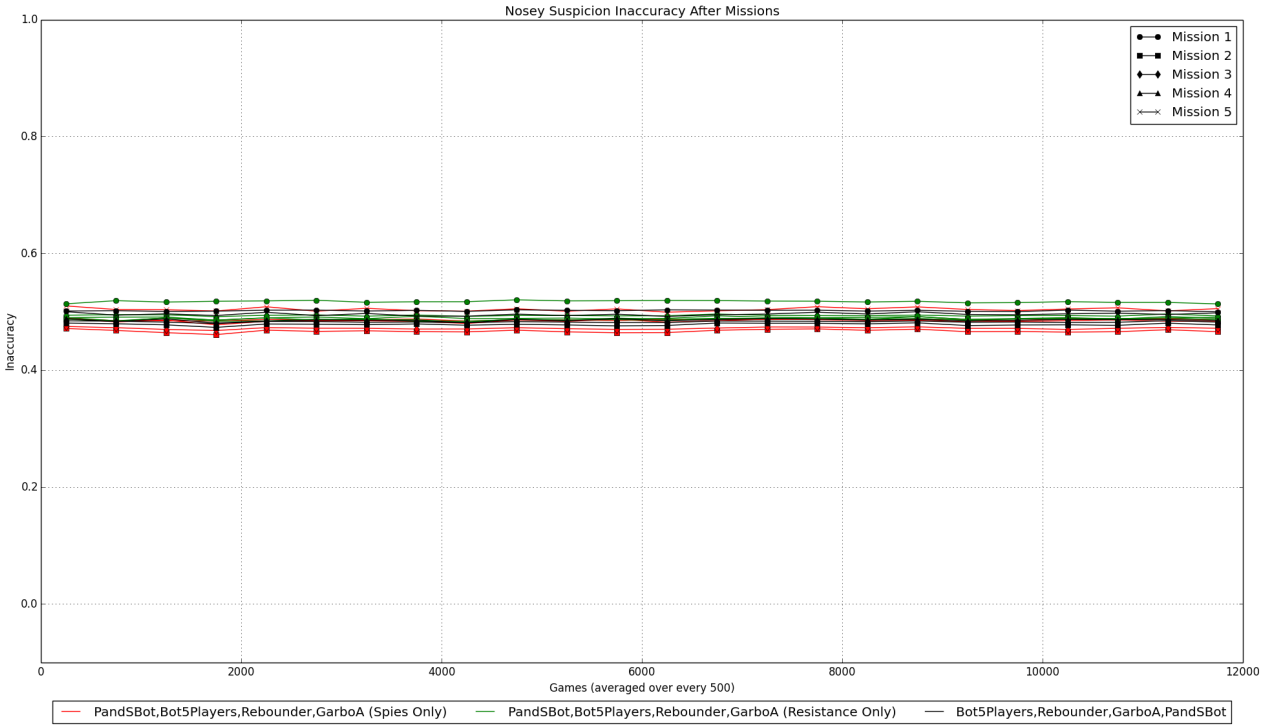

Figure A.9: Configuration: Only Perfect Information Behaviours

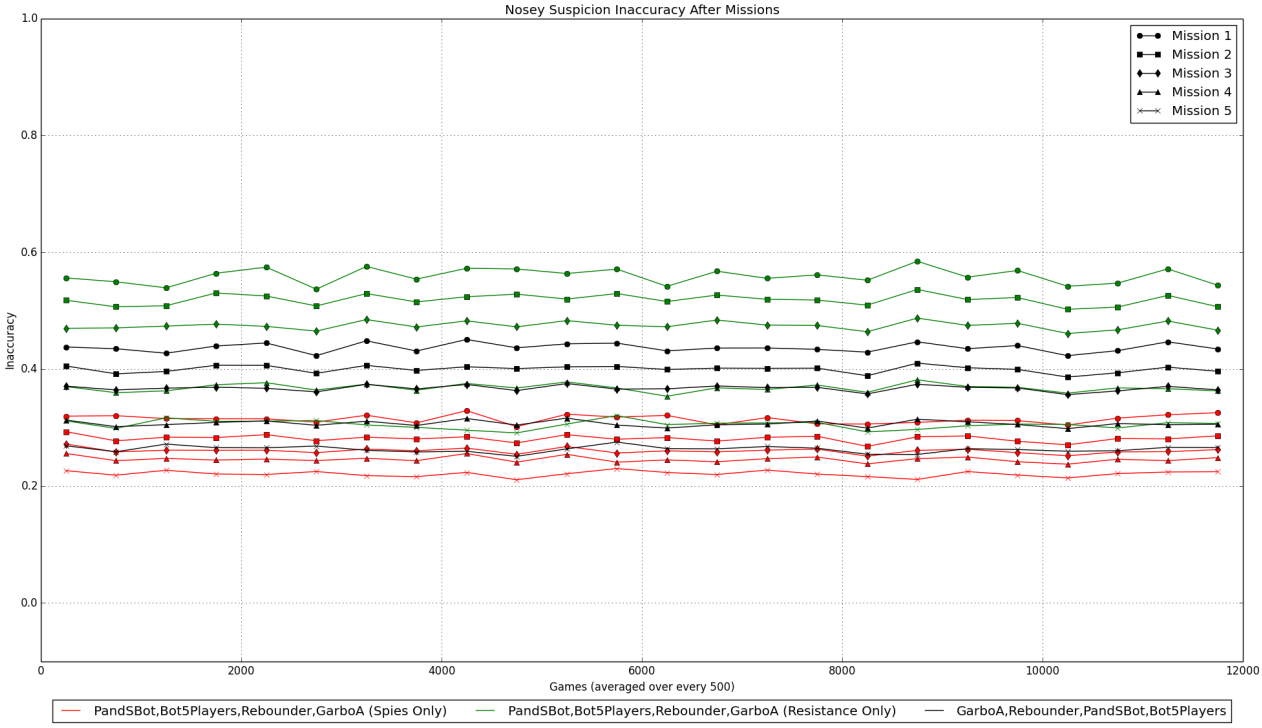

Figure A.10: Configuration: Only Competence Based Behaviours

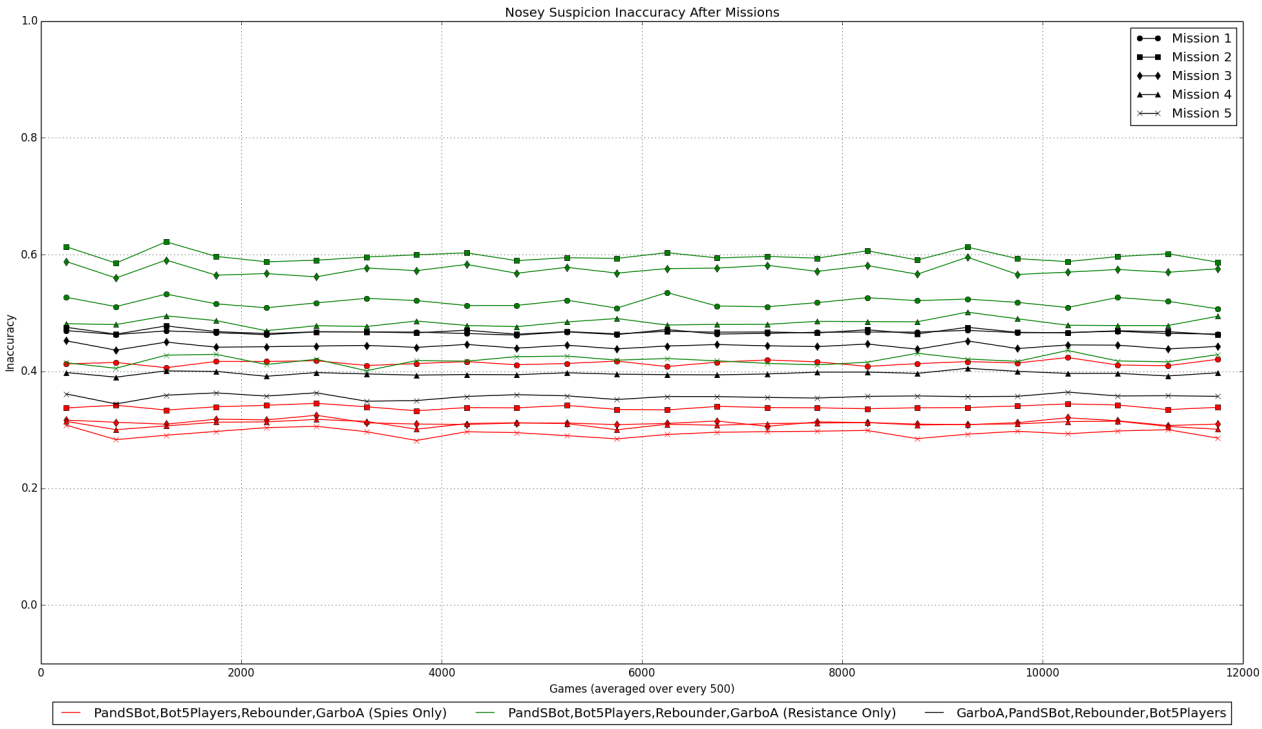

Figure A.11: Configuration: Only teamOnIsUnsuccessful