Does it matter how well I know what you're thinking? Opponent Modelling in an RTS game

James Goodman and Simon Lucas
Game AI Research Group
School of Electronic Engineering and Computer Science
Queen Mary University of London
Email: {james.goodman|simon.lucas}@qmul.ac.uk

***Abstract*—Opponent Modelling tries to predict the future actions of opponents, and is required to perform well in multi-player games. There is a deep literature on learning an opponent model, but much less on how accurate such models must be to be useful. We investigate the sensitivity of Monte Carlo Tree Search (MCTS) and a Rolling Horizon Evolutionary Algorithm (RHEA) to the accuracy of their modelling of the opponent in a simple Real-Time Strategy game. We find that in this domain RHEA is much more sensitive to the accuracy of an opponent model than MCTS. MCTS generally does better even with an inaccurate model, while this will degrade RHEA's performance. We show that faced with an unknown opponent and a low computational budget it is better not to use any explicit model with RHEA, and to model the opponent's actions within the tree as part of the MCTS algorithm.**

***Index Terms*—Opponent Modelling, Real-Time Strategy, Statistical Forward Planning, Evolutionary Algorithms, Monte Carlo Tree Search**

I. Introduction

When playing a game, or acting in any environment in which other players are present, we would intuitively expect that knowing what the other players are going to do will be helpful in deciding what actions to take. This is true whether their actions are taken in response to our actions, or simply as they interact with the environment regardless of what we decide to do. We need an *opponent model*.

It is also intuitive that if our opponent model is inaccurate, we will do less well. For example if we think our opponent will attack if they have a 3:1 advantage, but in fact they will only attack at 10:1 odds then we will likely play more defensively than optimal. Conversely, if they will actually attack at 2:1 odds then the defences we expect to inhibit an assault will fail to do so.

In a 2-player zero sum game we can theoretically fall-back on the concept of a Nash Equilibrium, which assumes a perfectly rational opponent, although this also reduces our potential to exploit a sub-rational agent. Calculating a Nash Equilibrium strategy may be straightforward in a simple normal-form game, but usually becomes intractable in an extensive form one [13]. Although there have been major successes here, as with the Counterfactual Regret Minimisation algorithms that approximate a Nash Equilibrium strategy in Texas Hold'em poker, these require very large amounts of pre-processing and domain-specific reductions of the state-space [16].

Real Time Strategy (RTS) games have very large branching factors and the potential for simultaneous actions by both players and also by different units of one player. This makes calculation of a Nash Equilibrium for every move infeasible. Any opponent model also requires some computational time. The more sophisticated our model, the less time we have to make our own decision. On the assumption that in most applications there will only be a limited computational budget available, we also need to trade-off between these. It may be that an inaccurate but cheap opponent model provides better results in actual play than a perfectly accurate but expensive one. At the lowest computational limit we have a 'DoNothing' opponent model that never does anything. This computational consideration is important in commercial games, in which we generally cannot leave the human player(s) waiting while the AI calculates its move, and CPU cycles have to be shared with such distractions as graphics, sound and physics engines.

In this paper we investigate the impact of varying the opponent model in a simple RTS-style game, Ground War, developed as a test-bed for ground-based military simulations. We investigate two popular Statistical Forward Planning algorithms that make use of a forward model to plan the next action; Monte Carlo Tree Search (MCTS) and Rolling Horizon Evolutionary Algorithms (RHEA). In each case we ask whether a simple opponent model with minimal computational budget can robustly improve the quality of play, and hence the verisimilitude of simulation results. By 'robustly' we specifically mean that the opponent model should be helpful against a variety of different actual opponents, and not just those for which the model is perfectly accurate. Work with statistical forward planning using a learned model in the Game of Life and Sokoban has shown that the learned environmental model need to be quite accurate to be at all useful [10], [20]. On the other hand recent work emphasised that learned forward models could lead to better outcomes if the model was trained to model what was relevant to gaining reward as opposed to trying to model the full state transitions [30].

We wish to determine how opponent model fidelity affects the game playing performance of statistical forward planning algorithms. We show that MCTS is more robust to using an incorrect opponent model than RHEA.

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

II. Background

A. Opponent Modelling

Opponent Modelling is where an agent A models what other agents in the environment will do in response to stimuli, where such stimuli include A's own actions. While in this work it is accurate to refer to 'opponent' models, more generally these are 'other agent' models, and are just as important in co-operative and semi-cooperative domains.

A few broad categories of opponent model can be distinguished, and see [1], [35] for more detailed surveys:

1) Game Theoretic. One approach is to assume the opponent is perfectly rational, and seek to find a Nash Equilibrium strategy (or approximation to), for example using counterfactual regret in Poker [16], [37]. This guarantees that we cannot be exploited by the other side, but can be very computationally demanding even when tractable.
2) Theory of Mind. Broadly this covers any approach which assumes that the other agents are also modelling us, and that to model their actions successfully it is necessary to also model their model of us (with a theoretically infinite recursion). Examples are the nested cognitive hierarchy of [3], [32], or the Recursive Modelling Method of [14]
3) Own policy. We assume the other agent is using the same algorithm that we are. A good example is MCTS where we model the other player's actions in the tree [2]. Classic minimax-search through the game-tree is another. In this case we can also use an estimate of the evaluation function that the opponent is using, which may be different to the one we use, especially if the game is asymmetric in any way [4], [35].
4) Heuristic. A hand-crafted 'expert' policy that specifies the action to take in any situation has the advantage of reducing computational overhead, but requires domain knowledge and will likely be able to represent a less flexible policy-space compared to the previous categories, especially if the heuristic is kept simple, both in terms of coding investment and time to execute. For example a heuristic policy used as a correctly-specified opponent model is found to significantly improve performance in MCTS [36]. This Heuristic can also be learned off-line from play-traces of human games, and/or be used to augment another approach, for example as a leaf evaluation function in tree search [23], [29].
5) Adaptive Model. A policy can be learned/adapted from an opponents observed moves as the game progresses. Approaches include Fictitious Play, or Bayesian updates over distinct heuristics [1], [31].
6) Environmental. In Multi-agent Reinforcement Learning the policy of actions taken by agents is subsumed as part of the environment [34]. This is not quite the same as no opponent model, as the opponent does act, but this is seen as a change to the environment and implicitly incorporated into the resultant learned policy.

As our objective here is to model our opponent with minimal computational overhead we use a Heuristic approach here, plus in the case of MCTS we use an 'Own Policy' approach that assumes the opponent is also using MCTS and model their actions in the same tree used for our own actions.

There is a deep literature learning an opponent model, but much less on the impact of getting this model wrong, or even slightly inaccurate [1]. In adaptive approaches this should be mitigated as the opponent model learns during play from actual opponent moves. In approaches that use a fixed heuristic or an off-line learned policy, then this risk of poor performance is greater.

Similar work to ours was considered when looking for robust play against unknown opponents in the game of Spades [33]. That found that a correct opponent model (using three exemplars on trick-play) is best, but can lead to major losses if mis-specified. A soft-max combination of the three exemplars was more robust against an unknown opponent. Their results also show that an incorrect model can sometimes help if it confuses the opponent by causing you to play contrary to what their correct model of you predicts.

B. Statistical Forward Planning

Monte Carlo Tree Search (MCTS) [2], [5], [9] has been used successfully in many games and other planning environments. It is an anytime algorithm that uses an available time budget to search the forward game tree from the current state. On each iteration four steps are followed:

1) Selection. Select an action to take from the current state. If all actions have been selected at least once then the best one is picked using the Upper Confidence for Trees equation [17]: $J(a) = Q(a) + C\sqrt{\frac{\log(N)}{n(a)}}$ The action a with largest $J(a)$ is selected at state. N is the total number of visits (iterations through) the state; $n(a)$ is the number of those visits that then took action$a$; $Q(a)$ is the mean score for all visits to the state that took action a; C is a parameter that controls the trade-off between exploitation, and exploration choosing actions with few visits so far. This step is repeated down the tree of game states until a state is reached with previously untried actions.
2) Expansion. Pick one of the untried actions at random, and expand this, creating a new state in the game tree.
3) Simulation. From the expanded state, complete a simulation to obtain a final score. In Ground Wars this does not run to the end of the game, but for 100 time-steps.
4) Back-propagation. Back-propagate this final score up the tree. Each state records the mean score of all iterations that take a given action from that state as $Q(a)$ that will affect future Selection steps. Once the available time budget has been used up the action at the root state with either the highest score or most visits is executed in the actual game environment.

A transposition-table approach is used in the tree with the node defined by the full visible state [7], rather than an Open Loop

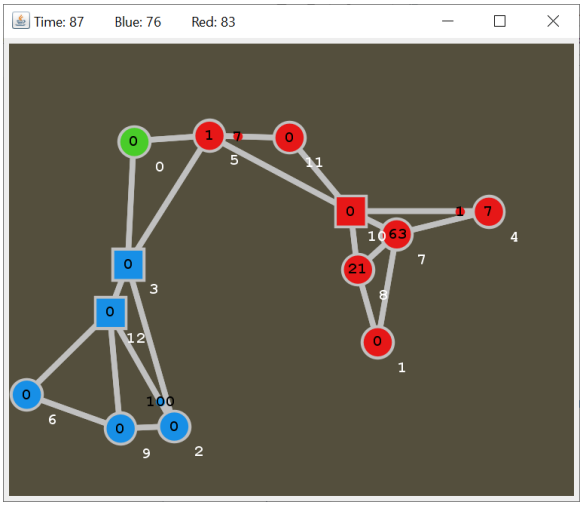

Fig. 1. Illustration of a Ground War game in progress between Blue and Red forces. The green node is neutral and not yet occupied by either side. Note the full Blue force of strength 100 is moving from Node 2 to Node 12 (node numbers in white).

approach in which the current state is defined by the action sequence taken to reach it [28].

Rolling Horizon Evolutionary Algorithms (RHEA) [27] evolve a sequence of actions (i.e. a plan) by iterating until the available time budget is exhausted (like MCTS it is anytime):

1) Generate a starting plan.
2) Randomly mutate the current 'best' plan.
3) Execute the mutated plan using the forward model and calculate a final score for the end state achieved.
4) If this is better than the previous 'best' plan, then replace the 'best' plan and repeat from step 2.

Once the time budget is exhausted, the first action in the current best plan is executed in the real environment. In all cases the score used in Ground War is the material advantage over the other side calculated as 5 points per occupied node, and 1 point per surviving force unit (see Section III).

III. Ground War

The digital simulation used for this project is 'Ground War' (developed on the foundations of the pre-existing 'Planet Wars' [19]). This aims to provide a simple, abstract simulation of ground combat with an arc and node map, and with forces able to move from node to node only along connecting arcs as shown in Figure 1. Key abstractions are:

1) No modelling of specific units. Each force has a size, being a simple numeric value. Each side (Blue and Red) has distinct attributes which apply to all forces of that colour.
2) Combat is resolved instantaneously by Lanchester's Laws, with one force removed in any battle [11], [15].
3) Once an order has been issued it cannot be cancelled and must be executed in full.

On top of these abstractions, additional functionality has been built of specific potential interest for wargame simulations to model the impact of fatigue, Command and Control (C2) constraints, Fog of War and Rules of Engagement. Only part of the C2 constraints are relevant to this study, and the remaining functionality is switched off for these experiments.

To represent the cognitive capacity of a human commander and the need to propagate orders to the units concerned a C2 parameter controls a minimum time (in game ticks) that must elapse between orders.

At any point in time (subject to these C2 constraints) either side may give an order to a force at any node to move to any connected node, and they will then start moving at their speed along the relevant arc. This may be for any percentage of the force currently in the node. For example if Blue has a force of size 45 in a node, then any number up to 45 may be ordered to move. This will split the force, with the remainder staying in the node as a 'garrison'.

The Ground War environment is similar in some respects to μRTS research environment for RTS games [26]. It does not have specific unit types, or economic aspects, but has many units that may be given orders simultaneously, and runs in continuous time. These both require adaptations for statistical forward planning from previous work which often has a single agent choosing a single action from a relatively small list at each time step [12], [21], [27].

A. Continuous Time

An order can be issued at any time with constraints imposed by the C2 parameterisation. Discrete time steps, or 'ticks', still exist in the simulation, but there is no requirement for an order to be selected for each one. This speeds up the simulation, as the computational overhead of planning is only required on a small subset of ticks. An order can be one of the following:

- LaunchExpedition(X, A, B). Send a force of size X from node A to node B.
- Wait(T). Wait for T ticks. The game forces this if no action may currently be taken due to the C2 constraints.

If the visible situation changes, then a Wait will be interrupted. For example if Red starts moving a force visible to Blue while Blue is Waiting, then Blue will immediately make a new decision in reaction (subject to the C2 constraint). This enables Blue to take a long Wait action until their currently moving forces reach their destinations without losing the ability to react to Red.

B. Action Space

RHEA evolves a genome that is then translated into a sequence of actions that can be taken in the environment. In contrast MCTS requires a list of actions that are valid in the current state. For RHEA a genome is a random number in base 10 (for example 35190438313924). This is converted to an action sequence as follows:

1) The first digits define A, the node from which an expedition should be launched. If there are fewer than 10 nodes only the first digit is used; if between 10 and 99 then the first two digits are used, and so on. If A is invalid due to this node not being under the control of

the player, or having no garrison, then a Wait order is the default.

2) The next digit defines B, the destination. '0' will use the first arc from A, '1' will use the second arc, and so on. (If a map has more than 10 arcs from any node, then two digits will be used here.)
3) The next digit defines the proportion of the current garrison force to be sent. This is defined in increments of 10%, with '0' being 10% and '9' being 100%.
4) The next digit defines how long to wait after taking the action. This is equal to d^2 ticks, where d is the digit value. The C2 constraints apply to set a minimum wait.
5) The above steps are then repeated for the next digits in the genome to generate the following action.

For MCTS we select 20 random actions at each node as the action space when the node is first visited. These are generated from random numeric sequences as above for RHEA until 20 distinct actions are found. This is a crude form of Action Abstraction [8], [24], reducing a very large number of possible actions to a small set tractable for forward planning.

There are more sophisticated methods by which the available actions for MCTS could be obtained, such as progressive unpruning or combinatorial multi-armed bandits (CMAB) [6], [25]. This approach is used to keep the action space as close as possible to that used for RHEA, and is sufficient to enable the comparison of opponent model behaviour that is our focus.

IV. Experiments

We investigate two types of opponent model here:

- Own Policy. In the case of MCTS we can model the opponent's actions using the same algorithm. For each forward simulation in planning we construct a tree for the opponent at the same time, and make their decisions using this. For each simulation we add one node to each tree. For RHEA no 'Own Policy' variant is tried; a Rolling Horizon Coevolutionary Algorithm [18] would be possible, but was found to perform poorly in preliminary experiments.
- Heuristic. A simple heuristic agent was hand-written which took three parameters:
  - Offence between 1 and 10. The numerical odds required for any Attack to be launched on a node.
  - Defence between 0 and 5. The numerical odds required to not Withdraw when Attacked.
  - Actions. An ordered list of Attack, Withdraw, Reinforce and Redeploy actions (in any order, including omissions). When considering a move, the Heuristic will run through this list and execute the first action that is valid.

    Attack is valid if a LaunchExpedition order against an enemy node exists that meets or exceeds the 'Offence' parameter odds.

    Withdraw is valid if an enemy attack is inbound on an owned node with fewer defenders than the 'Defence' parameter dictates, and a LaunchExpedition will retreat the defenders to an unthreatened node.

    Reinforce is valid if the enemy *could* launch an attack on a node that would provoke a Withdraw action. A LaunchExpedition will be executed to send reinforcements from an unthreatened node.

    Redeploy is valid if an unthreatened node can send reinforcements to another node that could then launch an Attack action (i.e. an offensive version of 'Reinforce').

A set of potentially useful Heuristic agents were obtained using the NTBEA optimisation algorithm [22] over these three parameters. H0 was handcrafted as a starting point. H1 was then optimised to be able to beat H0. H2 was optimised to play well against RHEA and then H4 against RHEA that used H2 as an opponent model; H3 against MCTS and H5 against MCTS that used H3 as an opponent model. The attributes of the resultant Heuristics are listed in Table I. H1 is a very aggressive player, attacking as long as it has any numerical advantage, while H2-4 will only attack if they have a 10:1 superiority. There is little difference between H2 and H3, optimised against RHEA and MCTS respectively; but surprisingly large differences between H4 and H5, optimised against RHEA+H2 and MCTS+H3.

In all cases RHEA and MCTS were given a budget of 50 generations/iterations per decision. RHEA used a (1+1) EA with one genome generated per generation with a mutation probability of 0.5, a length of up to 4 actions lasting 100 ticks and a discount factor of 0.999. MCTS used a C of 3, a discount factor of 0.999 and a rollout length of up to 100 ticks. These parameters were found using NTBEA optimisation with equal time allocated to RHEA and MCTS [22].

With 50 iterations RHEA takes an average of 0.8ms per decision and MCTS twice as long at 1.6ms. When an opponent model of any type is used by an algorithm these times approximately double. This is because the Ground War forward model does not calculate for each 'tick' in turn, but recalculates the game state only in a tick where an action is executed; either at initiation or when a two Forces meet in battle. As a result the computational time required is roughly proportional to the number of player decisions taken and not the number of game ticks that elapse. The Heuristics take about 800ns to make a decision, so do not impact on the time. It should be stressed that we are not comparing the performance of RHEA and MCTS as such, but how each algorithm makes use of an (in)accurate opponent model. The net play level of these algorithms with 50 iterations and 1-2ms per move is subjectively at a 'good novice' level appropriate for a simple Game AI; a human player can defeat them once they understand the game. Two sets of experiments were conducted using these RHEA, MCTS and Heuristic agents.

A. OM Accuracy experiments

RHEA and MCTS were tested against a fixed H3 opponent, with an opponent model based on H3 but Offence varied over all integers between 1 and 10, and Defence varied over the range 0.5 to 5.0 at 0.5 intervals. This gives 100 different

Name	Against	Offence	Defence	Actions
H0		3	1.2	W, A
H1	H0	1	0.5	RD, A, W, RF
H2	RHEA	10	1.5	RD, A, W, RF
H3	MCTS	10	1.0	RD, W, A, RF
H4	RHEA+H2	10	1.2	A, W, RF, RD
H5	MCTS+H3	3	0.5	W, RD, A, RF

TABLE I

HEURISTIC AGENTS USED. 'AGAINST' IS THE TARGET AGAINST WHICH EACH WAS OPTIMISED. THE ACTIONS ARE LISTED IN ORDER; **A**TTACK, **W**ITHDRAW, **R**EIN**F**ORCE AND **R**E**D**EPLOY. RHEA+H? MEANS RHEA USED WITH AN OPPONENT MODEL OF H?, WHERE ? $\in \{0, 1, 2, 3, 4, 5\}$

Heuristic opponents for each match-up, and 2000 games on random maps were run against each.

These experiments were then reversed with a fixed H3 or H0 opponent model, and the actual opponent being varied on the same basis. The objective of all these experiments was to see if an opponent model only provides benefit against accurately modelled opponents, and how far this benefit extends. The H3 and H0 agents from Table I were selected to vary both Offence and the available Actions.

All random maps had 8-10 nodes, and each side started with control of a single random node with a force of 100 units.

B. OM Efficacy experiments

Two round robin tournaments were run. The first includes all six Heuristic agents in Table I, plus RHEA using several of these as opponent models, as well as RHEA without an opponent model (i.e. a "Do Nothing" opponent model) and RHEA with an opponent model that took random actions. The second includes all Heuristic agents, and otherwise replaces RHEA with MCTS. It also includes an MCTS agent that uses MCTS to model the actions of the opponent; this does not increase the number of calls to the forward model so is 'free' if this is the rate-limiting step. These experiments investigate whether simple opponent models with a low computational overhead can robustly help against more realistic opponents that are outside the modelled policy space. The same 250 maps are used for all match-ups between agent pairs; each map is played twice with agents alternating sides for a total of 500 games between each pair.

V. RESULTS

A. OM Accuracy

Figure 2 shows that using an opponent model with the correct Offence parameter against H3 leads to better performance than not using an opponent model; there seems to be no significant effect of the Defence parameter. This effect falls off rapidly and RHEA with H3 as an opponent model (hereafter abbreviated to RHEA+H3) is only better than RHEA against Heuristic agents with an Offence of either 9 or 10. For any lower values RHEA performs better with no opponent model (i.e. where RHEA assumes that the enemy will sit still and make no new moves at all). This is not specific against H3, and the same pattern is shown against H0 (with Offence of

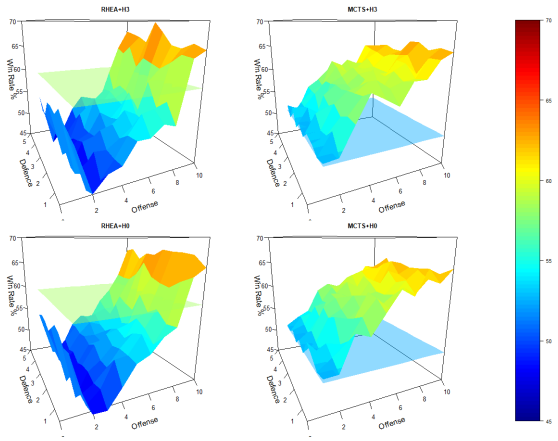

Fig. 2. The left-hand side shows the win rate of RHEA+OM against H3 and H0 fixed opponents, where 'OM' is based on H3 with Offence between 1 and 10, and Defence between 0.5 and 5.0. The right-hand side shows MCTS+OM on the same basis. 2000 games on random maps were run for each point. The plane in each shows the baseline win rate of RHEA/MCTS with no opponent model against the fixed opponent.

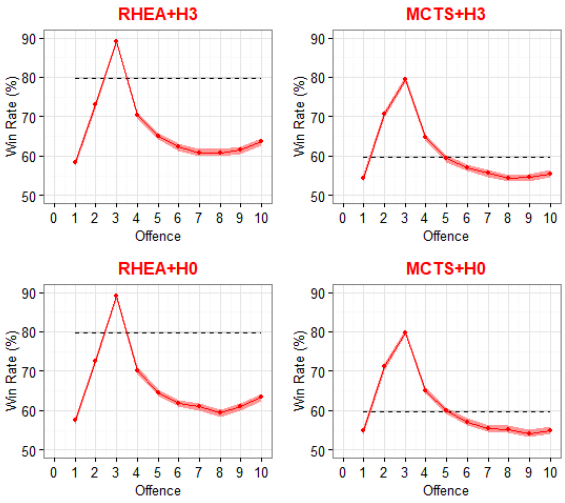

Fig. 3. Effect of varying opponent model against a fixed H0 opponent with Offence of 3. The experiments are as in Figure 2, but against H0 as the opponent and the Defence parameter is marginalised out. Shaded regions show 99% confidence intervals. Solid red line is the win rate of the RHEA/MCTS algorithm and the dotted black line is the baseline performance against H0 of RHEA/MCTS with no opponent model.

3) in Figure 3, which marginalises out the Defence parameter for clarity.

MCTS+H3 also performs better than MCTS with no opponent model when its actual opponent is close in Offence parameter. The fall-off as the opponent model becomes less accurate is less steep than with RHEA, and notably *any* opponent model helps performance. MCTS with an opponent model is always much better against H3 than vanilla MCTS in striking contrast to the RHEA results; and against H0 the deterioration with a very inaccurate model is small.

The effect of keeping the opponent model fixed and varying the actual opponent is shown in Figures 4 and 5. These

Agent		A	B	C	D	E	F	G	H	I	J	K	L	M	Avg
A	H0	50.0	44.4	51.2	52.0	51.4	52.0	34.6	47.6	44.6	45.0	45.2	51.6	37.2	47.5
B	H1	55.6	50.0	52.2	53.0	51.0	52.0	35.2	32.8	34.8	34.6	34.6	38.8	32.6	43.7
C	H2	48.8	47.8	50.0	50.2	48.8	49.6	51.8	47.4	44.8	44.2	42.0	51.4	46.4	48.1
D	H3	48.0	47.0	49.8	50.0	48.0	50.0	52.0	46.0	43.6	43.0	43.6	51.6	46.8	47.7
E	H4	48.6	49.0	51.2	52.0	50.0	51.6	50.4	47.2	36.2	34.6	35.2	50.6	38.8	46.4
F	H5	48.0	48.0	50.4	50.0	48.4	50.0	40.4	46.0	47.8	46.8	47.6	47.4	42.6	47.6
G	RHEA+H0	**65.4**	64.8	48.2	48.0	49.6	**59.6**	50.0	49.6	49.6	48.0	49.4	46.2	43.6	52.4
H	RHEA+H1	52.4	67.2	52.6	54.0	52.8	54.0	50.4	50.0	48.4	47.4	46.8	47.4	41.4	52.0
I	RHEA+H2	55.4	65.2	55.2	56.4	63.8	52.2	50.4	51.6	50.0	48.8	50.4	47.0	45.6	53.9
J	RHEA+H3	55.0	65.4	55.8	**57.0**	**65.4**	53.2	52.0	52.6	51.2	50.0	50.6	48.4	44.2	54.7
K	RHEA+H4	54.8	65.4	**58.0**	56.4	64.8	52.4	50.6	53.2	49.6	49.4	50.0	49.0	45.2	54.5
L	RHEA+RND	48.4	61.2	48.6	48.4	49.4	52.6	53.8	52.6	53.0	51.6	51.0	50.0	45.0	51.7
M	RHEA	62.8	**67.4**	53.6	53.2	61.2	57.4	**56.4**	**58.6**	**54.4**	**55.8**	**54.8**	**55.0**	**50.0**	**57.6**

TABLE II

PERCENTAGE WIN RATES OVER 500 GAMES ON RANDOM MAPS BETWEEN EACH PAIR OF RHEA OR HEURISTIC AGENTS. THE HIGHEST SCORING AGENT AGAINST EACH OPPONENT IS IN BOLD, AND A GREEN BACKGROUND HIGHLIGHTS ALL AGENTS WITHIN A ONE-TAILED 95% CONFIDENCE BOUNDARY OF THE BEST RESULT USING AN EXACT BINOMIAL TEST.

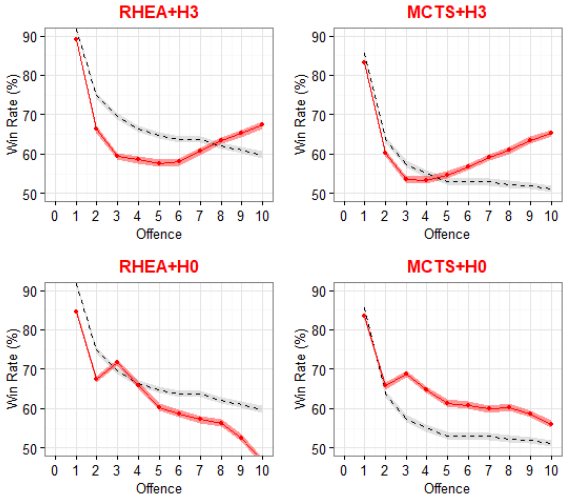

Fig. 4. Effect of varying an opponent based on H3 against a fixed opponent model. The top line uses H3 as an opponent model (Offence of 10); the bottom line uses H0 (Offence of 3 and restricted Actions). Shaded regions show 99% confidence intervals. Solid red line is the win rate of RHEA/MCTS+H3 and the dotted black line is the performance of RHEA/MCTS with no opponent model.

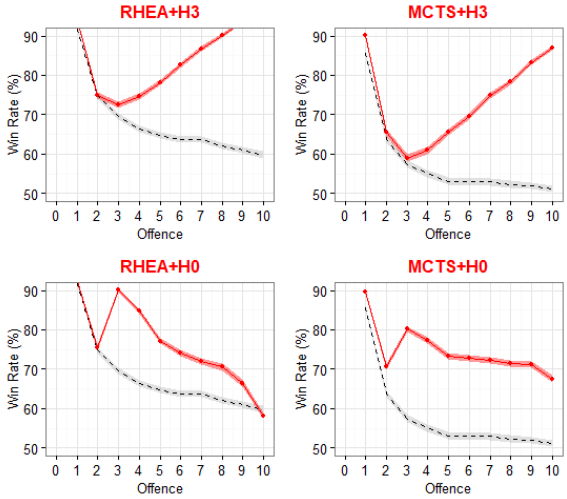

Fig. 5. Effect of varying an opponent based on H0 against a fixed opponent model. The experiments and key are as in Figure 4

show the same pattern. MCTS is almost always better with an opponent model, however inaccurate, while RHEA only benefits from an accurate opponent model. The exception to this is when the opponent is based on H0 (Figure 5), where RHEA also does better with any opponent model, but not as much as MCTS. H0 never Reinforces or Redeploys, and this reduced action set seems to give the real opponent fewer opportunities to 'surprise' the opponent model.

B. OM Efficacy

The RHEA results are summarised in Table II. RHEA+H0 does best against H0 and H5, the two Heuristic agents with the same Offence rating of 3. RHEA+H3 or RHEA+H4 do well against any Heuristic with an Offence of 10 (H2/H3/H4). RHEA with no opponent model only does especially well against H1, as do most RHEA+H? agents. However it does much better overall, and beats every RHEA+H? agent in a head-to-head. This supports the conclusions from the Accuracy experiments; having a reasonably accurate opponent model improves performance, but hinders performance against opponents for whom the model is not a good fit.

The MCTS results in Table III show a similar picture. MCTS+H0 does best against H0 and H5 as the most similar Heuristics, and MCTS+H1/3/5 are better against Heuristics than against other MCTS agents. The results of Section V-A where MCTS with *any* opponent model does much better against H3 than MCTS without one are replicated in column D of Table III. As with RHEA, MCTS with no opponent model generally performs much better against MCTS using an inaccurate opponent model, and MCTS consistently beats MCTS+H?. However best overall is MCTS+MCTS, which uses MCTS to model the opponent's actions. This is worse against specific opponents than MCTS using an accurate opponent model, but is best against every single MCTS+H? agent bar one. MCTS with no opponent model and MCTS with a random opponent model come a close second overall.

Agent		A	B	C	D	E	F	G	H	I	J	K	L	M	Avg
A	H0	50.0	46.0	52.0	51.0	52.2	51.2	36.6	46.6	47.8	46.6	49.0	46.6	46.0	47.8
B	H1	54.0	50.0	51.2	53.2	51.6	51.6	36.2	36.2	36.4	35.6	34.8	34.6	35.2	43.1
C	H2	48.0	48.8	50.0	50.4	48.6	48.6	48.4	43.8	43.0	43.6	49.8	49.8	**52.0**	48.1
D	H3	49.0	46.8	49.6	50.0	48.4	49.6	48.2	45.8	43.4	43.2	48.0	50.4	51.0	48.0
E	H4	47.8	48.4	51.4	51.6	50.0	51.0	49.6	39.8	41.6	40.6	46.2	45.2	49.8	47.2
F	H5	48.8	48.4	51.4	50.4	49.0	50.0	42.6	47.4	47.8	49.0	46.6	45.4	48.2	48.1
G	MCTS+H0	**63.4**	63.8	51.6	51.8	50.4	**57.4**	50.0	50.0	51.2	49.8	45.8	45.8	40.4	51.6
H	MCTS+H1	53.4	63.8	56.2	54.2	**60.2**	52.6	50.0	50.0	50.4	50.0	44.4	45.0	41.8	51.7
I	MCTS+H3	52.2	63.6	**57.0**	56.6	58.4	52.2	48.8	49.6	50.0	48.6	43.6	45.8	40.8	51.3
J	MCTS+H5	53.4	64.4	56.4	**56.8**	59.4	51.0	50.2	50.0	51.4	50.0	44.6	46.0	40.6	51.9
K	MCTS+RND	51.0	65.2	50.2	52.0	53.8	53.4	54.2	55.6	56.4	55.4	50.0	49.2	47.6	53.4
L	MCTS	53.4	**65.4**	50.2	49.6	54.8	54.6	54.2	55.0	54.2	54.0	50.8	50.0	47.8	53.4
M	MCTS+MCTS	54.0	64.8	48.0	49.0	50.2	51.8	**59.6**	**58.2**	**59.2**	**59.4**	**52.4**	**52.2**	50.0	**54.5**

TABLE III

PERCENTAGE WIN RATES OVER 500 GAMES ON RANDOM MAPS BETWEEN EACH PAIR OF MCTS OR HEURISTIC AGENTS. THE HIGHEST SCORING AGENT AGAINST EACH OPPONENT IS IN BOLD, AND A GREEN BACKGROUND HIGHLIGHTS ALL AGENTS WITHIN A ONE-TAILED 95% CONFIDENCE BOUNDARY OF THE BEST RESULT USING AN EXACT BINOMIAL TEST.

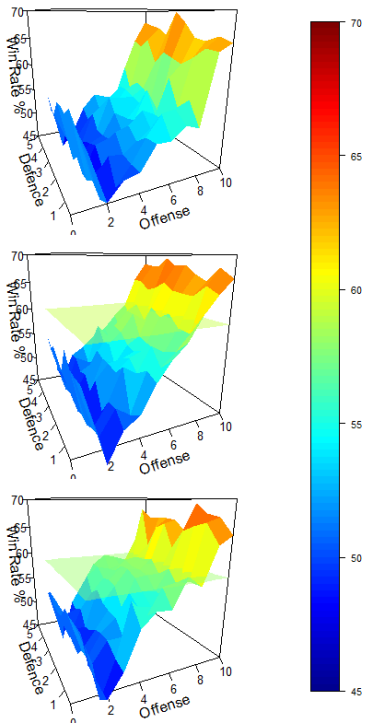

Fig. 6. Impact of changing RHEA sequence length (the number of forward actions in each plan). The top-most graph is for a sequence length for 4 actions as in Figure 2. The middle graph is for a sequence length of 2 actions, and the bottom-most for just a single action.

C. RHEA Horizon Length

Ground War is a 2-player zero sum game, and what the other player does very emphatically affects our score. In this environment we expect modelling the opponent action to be helpful, and the fact that RHEA is often more successful if it assumes the opponent does nothing is counter-intuitive and in need of explanation. The MCTS result is more congruent with our a priori expectation.

The results suggest that RHEA is particularly sensitive to a poor opponent model in comparison to MCTS. One hypothesis is that RHEA is more sensitive to a poor assumption because it always plans forward for x actions (where $x = 4$ in Figure 2), and applies equal amounts of computation to each action. MCTS focuses more of its budget on the first action taken as it builds up the game tree over iterations; every single iteration will make a choice in the tree for the first action, but only the last few iterations of the 50, if any, will make a decision for the third action. This means that RHEA may generate a plan that is dependent on anticipated moves by the opponent over longer period of time relative to MCTS. In this case searching less far forward in time can be beneficial.

To test this hypothesis we ran repeat experiments with RHEA that have a reduced horizon of 1 or 2 actions. These results are shown in Figure 6. Even when RHEA is only planning one action forward there is still a steep fall-off in performance for an incorrect opponent model to below a baseline with none. There is however a plateau for an opponent model Offence rating between 4 and 8 that is not significantly different to the baseline. This lends some support to our hypothesis but it is clearly far from sufficient to explain the full difference between the effect on an opponent model in RHEA and MCTS. A full understanding of this is a key area for future work.

VI. CONCLUSION AND FUTURE WORK

We have used a parameterised Heuristic opponent in a simplified RTS, Ground War, to investigate the effect of accuracy of an opponent model for performance of MCTS and RHEA agents with a small computational budget. For both algorithms we have varied the actual opponent while keeping the opponent model constant, and vice versa. We repeated this for two different constant opponents and opponent models (H0 and H3) picked to be very different from each other.

We have shown that in this domain having an accurate opponent model in statistical forward planning is beneficial and improves performance. With RHEA this benefit rapidly falls away as the opponent model becomes less accurate and the experimental results suggest that using no opponent model at all (assuming that the opponent never acts) can be the best approach if we are uncertain about the their actual policy.

MCTS also benefits from an accurate opponent model, and here the fall-off is much shallower. An opponent model is usually still beneficial even if quite inaccurate. However, modelling the opponent within the MCTS tree itself is much more robust and is preferable if the opponent policy is unknown.

In this work we do not adapt the opponent model based on observation of actions taken during the game so far. This is a common approach and to the extent that it improves the accuracy of the opponent model should be beneficial. The research question of how accurate an opponent model needs to be is just as valid in this adaptive method given that any learned model is still constrained to a policy-space that may not include the actual opponent.

Further work is especially needed to understand the precise origin of the difference in behaviour of RHEA and MCTS with an opponent model, and investigate this effect in other games beyond the one used here.

VII. Acknowledgments

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/S022325/1.

References

[1] Albrecht, S. V., and Stone, P. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence 258* (2018), 66–95.

[2] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games 4*, 1 (Mar 2012), 1–43.

[3] Camerer, C. F., Ho, T.-H., and Chong, J.-K. A cognitive hierarchy model of games. *The Quarterly Journal of Economics 119*, 3 (2004), 861–898.

[4] Carmel, D., and Markovitch, S. Incorporating opponent models into adversary search. In *AAAI/IAAI, Vol. 1* (1996), 120–125.

[5] Chaslot, G., De Jong, S., Saito, J.-T., and Uiterwijk, J. Monte-carlo tree search in production management problems. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence* (2006), 91–98.

[6] Chaslot, G. M. J., Winands, M. H., HERIK, H. J. V. D., Uiterwijk, J. W., and Bouzy, B. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation 4*, 03 (2008), 343–357.

[7] Childs, B. E., Brodeur, J. H., and Kocsis, L. Transpositions and move groups in monte carlo tree search. In *2008 IEEE Symposium On Computational Intelligence and Games*, IEEE (Dec 2008), 389–395.

[8] Churchill, D., and Buro, M. Portfolio greedy search and simulation for large-scale combat in starcraft. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, IEEE (Aug 2013), 1–8.

[9] Coulom, R. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, Springer (2006), 72–83.

[10] Dockhorn, A., Lucas, S. M., Volz, V., Bravi, I., Gaina, R. D., and Perez-Liebana, D. Learning local forward models on unforgiving games. In *2019 IEEE Conference on Games (CoG)*, IEEE (2019), 1–4.

[11] Engel, J. H. A verification of lanchester's law. *Journal of the Operations Research Society of America 2*, 2 (1954), 163–171.

[12] Gaina, R. D., Lucas, S. M., and Perez-Liebana, D. Rolling horizon evolution enhancements in general video game playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE (Aug 2017), 88–95.

[13] Gilpin, A., Hoda, S., Peña, J., and Sandholm, T. *Gradient-Based Algorithms for Finding Nash Equilibria in Extensive Form Games*, vol. 4858. Springer Berlin Heidelberg, 2007, 57–69.

[14] Gmytrasiewicz, P. J., and Durfee, E. H. A rigorous, operational formalization of recursive modeling. In *ICMAS* (1995), 125–132.

[15] Hartley III, D. S., and Helmbold, R. L. Validating lanchester's square law and other attrition models. *Naval Research Logistics (NRL) 42*, 4 (1995), 609–633.

[16] Johanson, M., Bard, N., Burch, N., and Bowling, M. Finding optimal abstract strategies in extensive-form games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence* (2012).

[17] Kocsis, L., and Szepesvári, C. Bandit based monte-carlo planning. In *European conference on machine learning*, Springer (2006), 282–293.

[18] Liu, J., Pérez-Liébana, D., and Lucas, S. M. Rolling horizon coevolutionary planning for two-player video games. In *2016 8th Computer Science and Electronic Engineering (CEEC)*, IEEE (2016), 174–179.

[19] Lucas, S. M. Game ai research with fast planet wars variants. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE (2018), 1–4.

[20] Lucas, S. M., Dockhorn, A., Volz, V., Bamford, C., Gaina, R. D., Bravi, I., Perez-Liebana, D., Mostaghim, S., and Kruse, R. A local approach to forward model learning: Results on the game of life game. *arXiv preprint arXiv:1903.12508* (2019).

[21] Lucas, S. M., Liu, J., Bravi, I., Gaina, R. D., Woodward, J., Volz, V., and Perez-Liebana, D. Efficient evolutionary methods for game agent optimisation: Model-based is best. *arXiv preprint arXiv:1901.00723* (2019).

[22] Lucas, S. M., Liu, J., and Perez-Liebana, D. The n-tuple bandit evolutionary algorithm for game agent optimisation. *arXiv:1802.05991 [cs]* (Feb 2018). arXiv: 1802.05991.

[23] Mizukami, N., and Tsuruoka, Y. Building a computer mahjong player based on monte carlo simulation and opponent models. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE (Aug 2015), 275–283.

[24] Moraes, R. O., Marino, J. R., Lelis, L. H., and Nascimento, M. A. Action abstractions for combinatorial multi-armed bandit tree search. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference* (2018).

[25] Ontanón, S. Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research 58* (2017), 665–702.

[26] Ontanón, S., and Buro, M. Adversarial hierarchical-task network planning for complex real-time games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence* (2015).

[27] Perez, D., Samothrakis, S., Lucas, S., and Rohlfshagen, P. Rolling horizon evolution versus tree search for navigation in single-player real-time games. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*, ACM Press (2013), 351.

[28] Perez Liebana, D., Dieskau, J., Hunermund, M., Mostaghim, S., and Lucas, S. Open loop search for general video game playing. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*, ACM Press (2015), 337–344.

[29] Rebstock, D., Solinas, C., and Buro, M. Learning policies from human data for skat. In *2019 IEEE Conference on Games (CoG)*, IEEE (2019), 1–8.

[30] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. Mastering atari, go, chess and shogi by planning with a learned model, 2019.

[31] Shum, M., Kleiman-Weiner, M., Littman, M. L., and Tenenbaum, J. B. Theory of minds: Understanding behavior in groups through inverse planning. In *33rd AAAI Conference on Artificial Intelligence* (2019).

[32] Stahl, D. O., and Wilson, P. W. On players models of other players: Theory and experimental evidence. *Games and Economic Behavior 10*, 1 (Jul 1995), 218–254.

[33] Sturtevant, N., and Bowling, M. Robust game play against unknown opponents. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems - AAMAS '06*, ACM Press (2006), 713.

[34] Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning* (1993), 330–337.

[35] van den Herik, H. J., Donkers, H., and Spronck, P. H. Opponent modelling and commercial games. *Proceedings of the IEEE* (2005), 15–25.

[36] Walton-Rivers, J., Williams, P. R., Bartle, R., Perez-Liebana, D., and Lucas, S. M. Evaluating and modelling hanabi-playing agents. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, IEEE (Jun 2017), 1382–1389.

[37] Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. Regret minimization in games with incomplete information. In *Advances in neural information processing systems* (2008), 1729–1736.