Opponent Models Comparison for 2 Players in GVGAI Competitions

Jose Manuel Gonzalez-Castro
University of Essex
Colchester CO4 3SQ, UK
Email: jmanuelgc94@gmail.com

Diego Perez-Liebana
University of Essex
Colchester CO4 3SQ, UK
Email: dperez@essex.ac.uk

***Abstract*—The GVGAI-2P competition has shown that opponent models are an area to explore as most of the submissions assume random actions from the opponent. This paper presents 9 different opponent models implemented for and within the sample MCTS controller from the framework. All opponent models were put up against one another in a round robin style tournament. The experiment took place in an offline environment where competition parameters are not in effect. Results gathered from the experiment show that models that have a buffer with history of the actions from the player either outperform or are close to beating the random approach used by participants in the competition, and also suggest that probabilistic models (*LimitedBuffer, UnlimitedBuffer, Probabilistic*) are the ones with the highest win rates.**

I. Introduction

General Game Playing (GGP) is a field in Artificial Intelligence (AI) where players learn to play different kinds of turn-taking board games [1]. General Video Game Playing (GVGP) is an extension of GGP applied to video games [2]. Both areas count on competitions where people submit agents to be matched in a series of games, developing progress in the field with more proficient agents as time passes.

General Video Game AI (GVGAI) [3][4] is a GVGP competition with multiple tracks where participants submit agents that play real time video games created with the Video Game Description Language (VGDL) [5].

GVGAI recently introduced the 2 Player track [6] where agents are tested in both cooperative and competitive video games, in the former agents need to work together towards a common goal while the latter has them competing against one another. In both the first and second editions of the competition the results showed two important points; the first being that Monte Carlo Tree Search (MCTS) typically outperforms all the other agents submitted, and the second being that most of the agents submitted were not using an enhanced opponent model as they were assuming random actions from the opponents [6].

The results from the 2016 edition of the competition [6] stress the fact that opponent modeling in this context remains an open question. Therefore this paper addresses the matter by introducing and testing 9 different opponent models apart from the one used by controllers submitted to the competition. To get accurate results all agents were tested using MCTS in a closed environment where competition parameters such as the time limit to make an action don't affect the performance. This is meant to show the potential in using alternative opponent models in the 2 Player track of GVGAI.

The remainder of the paper is structured as follows: Section II describes previous work carried out in the area of opponent modeling, Section III details the GVGAI 2 player (GVGAI-2P) framework, Section IV explains the Monte Carlo Tree Search method, Section V introduces the opponent models used for the experiment, Section VI explains the experiment setup, Section VII shows and explains the results, Section VIII concludes the paper with the results of the research, the discussion of them and future work.

II. Relevant Research

Opponent modeling has been a subject of research for turn-table games such as Texas Hold 'Em, Poker and Chess [7][8][9]. Research carried out for these games has focused on figuring out which move the opponent is most likely to do given the current state of the game in order to maximize its score, this way the agent can react by using a move to counter his opponent. These types of games have a fixed number of possible combinations of moves, this is the reason turn-table games involve probabilistic models [10].

Real-time strategy (RTS) video games have also used opponent models in recent years given their levels of abstraction by modeling economy and warfare. More often opponent models have been created for specific games such as Civilization IV or Starcraft: Brood War but there have been approaches to create generic models for RTS games [11][12]. The main problem when dealing with RTS games is the lack of information the player has in the current time step. The previous issue has led researchers to develop two phases when creating RTS opponent models, an offline and an online one. The first phase involves looking at previous gameplay to build descriptive figures through feature selection and create a model of the opponent. The online phase then needs to determine which model the enemy behavior is closest to [11]. Another approach that has been explored has been a hierarchical model with two levels where the top level has a general play style for the opponent while the bottom one classifies the opponent's choice based on the units it has built [13].

Since there are various types of games that opponent models can deal with and given the amount of decisions and combinations of moves for even a single game, the most common approach is to create a tree with all the possible moves you can explore in the next time step. There can be hundreds of thousands of actions depending on the game, therefore researchers have also been interested in creating pruning techniques to reduce the time taken to get the next move or even to maximize the agents reward. One of the most used techniques is Max^{n} with Alphabeta pruning. Max^{n} is an extension of minimax that holds an n-tuple for n players and searches for the best move in a given state of the tree for a particular player. Alphabeta pruning has been used in the Hearts and Sergeant Major card games to reduce the size of the tree by only using moves that will maximize the agents score at a given time step, this way there is no need to explore the whole tree [14]. Another common technique that uses probability is the fallible opponent where you assume that the opponent will not select a rational move, this has been tested on the games Roshambo and Poker [15]. More work has been carried out on pruning techniques and probability with opponent models. Donkers [9] shows a comparison between variations of these opponent models in four different game domains including random game trees, Lines of Action, the King-Queen-King-Rock endgame of chess, and Chinese Bao.

To the knowledge of the authors, opponent models have not been applied to GGP [16] and GVGP but rather to specific video games or genres like RTS or turn-table. Even in the GVGAI-2P competition most of the submissions have explicitly stated that the opponent model used is the default one where an action is chosen at random. The previous model was used by the participants that got the first, third, fourth and fifth places from the 2016 edition of the competition. It was only the participant that obtained second place that used a different opponent model, this model involved keeping track of the opponents score $Q_{opp}(a)$. This value is the average of all the opponents scores back-propagated through that same node where action a was selected. Taking the previous the model chooses a random action at each node with a probability $(\varepsilon) = 0$, while the action that maximizes $Q_{opp}(a)$ is chosen with probability $1 - (\varepsilon)$ [6].

III. GVGAI-2P Framework

GVGAI uses a framework structured in an object-oriented way and it uses the Video Game Description Language (VGDL) which was built up using the Game Descriptor Language (GDL) as a baseline. The main difference between them is that VGDL uses the entities in the game and their interactions as a game descriptor. This in return offers a wider array of games, including arcade and puzzle games such as Pacman, Space Invaders, among others. Given that it is object-oriented, the framework is built in Java and it uses objects to provide information about the game state. In addition, the framework provides agents with a time limit to perform each step in the game as they only have 40ms. Using this, the agent needs to learn how to play the game, without knowing the rules, dynamics, and even the requirements to beat the game [4].

In the GVGAI-2P framework agents know whether the game involves 1 or 2 players. The game description provided as a text file indicates this in the first line, subsequent lines indicate the entities in the game, interactions among them, the level mapping and conditions to end the game. Another file is also provided, this is called the level file and it has a matrix of ASCII characters representing the state of the sprites in the game [6]. Additionally, each agent is provided with 40 milliseconds at each game step to perform an action and 1 second for initialization, the only exception is the *act* method which can return the NIL action if the time is between 40 and 50 milliseconds. Any controller that goes beyond the established time limit is disqualified [6].

Players in all games from the framework play simultaneously, this differs from turn-table games where MCTS grows the tree by alternating moves between the player and the opponent. Since the players don't know the opponents action at each turn the decision tree is constructed differently. Each tree level in MCTS has states derived from both players executing their respective moves in the same time step [6].

IV. Monte Carlo Tree Search

Monte Carlo Tree Search methods have been widely used in the different GVGAI entries and many variations have been awarded the top spots in the competition. In the 2016 edition of the GVGAI-2P competition 3 from the first 5 places were using variations of MCTS.

The underlying ideas behind MCTS are the capacity of approximating an action's value through random simulations and further using these values to get the best policy. These algorithms iteratively build a tree that can estimate the value of each action, until a given budget (i.e. time, iterations, or others) is exhausted. The four main steps of each iteration are selection, expansion, simulation, and back-propagation [17].

Selection starts at the root of the tree, where based on a policy a child is selected recursively and the tree is navigated until a non-terminal state is reached, or in other words, a child that has not been expanded. This is followed by expansion, where based on a set of actions the tree is expanded by adding one or more child nodes to the tree. From that node, a simulation of game moves is performed using a default policy until the game end or a determined simulation depth is reached. The state reached is evaluated, and the result is back propagated through the visited nodes.

Upper Confidence Bound (UCT) is the name of the MCTS algorithm when it uses UCB1 as a tree policy. In this algorithm the value of a node is the expected reward of a child node in the tree as seen through the result of the Monte Carlo simulations. Additionally UCT is considered a good approach to avoid the exploration-exploitation dilemma in MCTS as it is simple and efficient [17]. Given its characteristics it has been used in the GVGAI-2P framework as part of the default MCTS controller.

V. Opponent Models

There were 9 different opponent models created to test with the GVGAI-2P framework. Some of these models follow the probabilistic approach that has been mentioned while some are just variations and simplifications of other models described. While more opponent models have been researched as seen in [9], this section focuses on simpler models that avoid putting too much overhead on the controllers submitted to the competitions so they don't get disqualified by exceeding the time restrictions for taking an action (40 milliseconds). The opponent models covered are *Alphabeta*, *Average*, *Fallible*, *LimitedBuffer*, *Minimum*, *Mirror*, *Probabilistic*, *Same Action*, and *UnlimitedBuffer*.

A. Alphabeta

Unlike the original pruning technique with the same name as mentioned in [14], this implementation of *Alphabeta* has been simplified to avoid longer execution times and possible disqualifications from the competition. The main idea behind the algorithm is to get the current state of the game and expand the tree one step at a time. At each expansion the model gets all the actions and for each one of them simulates and gets its respective reward *Q(s,a)*. Instead of back-propagating after the simulation has ended, the model returns the action with the highest reward (see Equation 1). By doing this there is no need to create the whole tree and prune afterwards, instead at each time step the tree expands with the best action possible thus having a model that predicts the opponent will always select the best action at each time step.

$$a^* = \underset{a \varepsilon A(s)}{\arg\max}\{Q(s,a)\} \tag{1}$$

B. Average

This opponent model follows similar steps as *Alphabeta*. The underlying idea is to return an action with a moderate reward. To do so the opponent's tree is expanded, all actions are then retrieved and simulated once to get their associated rewards *Q(s,a)*. All actions and their respective rewards are used afterwards to get an average reward *avg(Q(s,a))*. Before determining which action needs to be returned, all available actions are looped over until the action closest to *avg(Q(s,a))* is found, it is then returned as the opponents action. This model assumes the opponent will not select the best or worst actions at each time step but rather an average or moderate one.

C. Fallible

As explained in [15], a fallible opponent is that which has an erratic behavior by selecting irrational moves from time to time. In order to emulate such behavior this model uses a combination of *Alphabeta* by getting the action with the highest reward as the default action, and the action with the minimum reward as the irrational one. A threshold was established to limit when the agent follows its normal behavior and when it doesn't. This was done with probabilities where the opponent has a probability $p = 0.2$ of selecting the worst possible action available after expanding the tree and simulating once for every available action. If the opponent doesn't select the worst move it will return the best action with probability $p = 0.8$.

D. Minimum

Similar to the *Alphabeta* model, *Minimum* also focuses on expanding the opponents tree one step at a time and simulating the results for every action. The only difference is the policy that determines which action is to be returned as it is the one with the minimum reward *Q(s,a)*. This model will essentially build the worst possible tree of actions as it follows the least favorable moves at every time step (see Equation 2). As with its counterpart, the model provides less overhead to the controller by just expanding the tree one step at a time instead of creating it fully.

$$a^* = \underset{a \varepsilon A(s)}{\arg\min}\{Q(s,a)\} \tag{2}$$

E. Probabilistic

Perhaps one of the most complex opponent models created as it involves offline learning to get the probabilities of the player using each action. The offline phase involved recording the moves used by the sample MCTS controller agent provided with the GVGAI-2P framework. This was done with 20 games from the framework (see Table I). After doing so the probabilities of using each move were calculated. An array of size 10000 stores all the actions based on their probability of being used. A random number between 0 and the size of the array (10000) was drawn, this served as the index to get from the array and represents the opponents move. Its disadvantage is that an action that is not used in a game can be picked. One possible improvement outside the scope of this paper and related to Machine Learning is to take features as inputs so the model learns the actions in terms of game state features.

F. Limited Buffer

As part of the models that use probabilities, *LimitedBuffer* stores the last $n = 20$ actions performed by the player. The model starts gathering the first 20 actions by following the same approach as the *Probabilistic* model by using the offline data to get the opponents action. Once it has filled in the buffer a random seed between 1 and n is created. This number is used as an index to get a value from the buffer, thus simulating what would be a probability distribution since each action stored has a given probability of occurring. The random seed is used to reduce the models complexity if it is to be used in the competition. Additionally, the buffer could be shuffled before drawing the opponents action. This model has the advantage that it only stores the most relevant actions in the buffer as some games use a limited number of them.

G. Unlimited Buffer

The last opponent model is an extension of the *LimitedBuffer* approach. It still uses the probabilities from the *Probabilistic* model for the first 20 actions. Once the buffer is filled with at least 20 actions it starts to add all the following moves from the player, this creates a probability distribution over time with the moves used per game. With this, the probabilities of each move variate less over time as more actions are performed. At each time step after adding

TABLE I
OFFLINE SET OF GAMES USED TO GET THE FREQUENCY OF ALL ACTIONS

Games				
Accelerator	Akka Arrh	Asteroids	Beekeeper	Bombergirl
Breeding Dragons	Capture Flag	Compete Sokoban	Cops N' Robbers	Donkey Kong
Dragon Attack	Drowning	Egg Hunt	Fatty	Fire Truck
Football	Ghostbusters	Gotcha	I Saw Santa	Klax

the players action to the buffer a random number between 0 and the buffer's size is rolled, this is the index from the buffer where the opponents action will be taken from. Its main disadvantage is that it takes the history of the game to select the action, the problem is that at a given point in the game some actions are less relevant than others.

H. Mirror

One of the simplest actions the opponent can choose is to select an opposite action from the one chosen by the player. This is the main idea behind this opponent model and the main reason why it is so simple and provides little to no overhead on the controller. The only actions that can be mirrored from the player are *ACTION_UP, ACTION_DOWN, ACTION_LEFT, ACTION_RIGHT* which return their respective counterparts as the opponents move. If the player uses any of the remaining actions the opponent will copy that as its action.

I. Same Action

Just as the *Mirror* opponent model, this provides less overhead to the controller as the opponents action is the same as the player. This is the simplest out of all the opponent models and it assumes the opponent will always try to follow your moves. There is no need to expand the opponents tree and simulate to get the best action as this is only done by the player and copied by the opponent.

VI. Experiment Setup

Several variations of the sample MCTS controller with the different opponent models provided in this paper were tested on 10 games (see Table II) from the GVGAI-2P framework. Each algorithm played in all 5 levels of the 10 games 5 times (therefore 25 times per game per algorithm). All agents played with the previous settings two times to swap positions and play as both player 1 and player 2. This was done to eliminate any possible bias caused by playing as any of the two players.

All controllers were tested in an offline environment, this means they were not uploaded to the competition website. Instead all controllers played in a round robin style tournament where each algorithm played against all the others swapping places as first and second player. Competition parameters were excluded, this includes the limit of 40 milliseconds per action so no controller could be disqualified. The previous was done to ensure that all controllers can build their decision trees, thus providing more time to select better actions for both the agent and the opponent when needed.

In order to avoid bias from the machine being used, all controllers were given a budget with which to work at every time step as they were given 900 forward model calls which

TABLE II
GAMES COMPRISING THE TEST SET

Games	Type
Asteroids, Capture Flag, Cops N Robbers, Gotcha, Klax, Samaritan, Sokoban, Steeplechase, Tron	Competitive
Akka Arrh, Sokoban	Cooperative

TABLE III
RESULTS FROM THE ROUND ROBIN TOURNAMENT WITH THE WINS, DRAWS, AND LOSSES OF EACH CONTROLLER, ROWS REPRESENT PLAYER 1 WHILE COLUMNS REPRESENT PLAYER 2. LEGEND: 1) ALPHABETA, 2) AVERAGE, 3) FALLIBLE, 4) LIMITEDBUFFER 5) MINIMUM, 6) MIRROR, 7) PROBABILISTIC, 8) SAMEACTION, 9) UNLIMITEDBUFFER, 10) RANDOM

	1	2	3	4	5	6	7	8	9	10
1		(4,3,3)	(5,3,2)	(2,2,6)	(4,4,2)	(3,2,5)	(2,2,6)	(4,2,4)	(1,3,6)	(2,2,6)
2	(4,2,4)		(5,3,2)	(3,2,5)	(5,2,3)	(4,3,3)	(2,2,6)	(3,3,4)	(3,3,4)	(2,2,6)
3	(3,2,5)	(5,4,1)		(2,2,6)	(5,2,3)	(2,3,5)	(3,2,5)	(4,3,3)	(1,3,6)	(3,2,5)
4	(5,3,2)	(5,3,2)	(7,2,1)		(5,4,1)	(4,3,3)	(5,3,2)	(4,3,3)	(4,3,3)	(4,3,3)
5	(4,3,3)	(3,3,4)	(6,2,2)	(1,2,7)		(1,3,6)	(1,3,6)	(4,3,3)	(3,3,4)	(2,2,6)
6	(5,3,2)	(4,4,2)	(6,3,1)	(4,2,4)	(5,3,2)		(4,2,4)	(5,3,2)	(3,5,2)	(2,3,5)
7	(6,3,1)	(5,3,2)	(5,2,3)	(3,2,5)	(5,3,2)	(5,3,2)		(4,3,3)	(4,2,4)	(5,3,2)
8	(5,3,2)	(6,2,2)	(6,2,2)	(3,3,4)	(6,3,1)	(4,2,4)	(3,2,5)		(3,4,3)	(3,3,4)
9	(6,2,2)	(7,2,1)	(5,3,2)	(3,2,5)	(4,4,2)	(5,2,3)	(4,3,3)	(5,3,2)		(5,3,2)
10	(5,3,2)	(5,3,2)	(5,2,3)	(4,4,2)	(5,3,2)	(5,2,3)	(5,3,2)	(5,2,3)	(3,4,3)	

is the average number of calls achieved by the vanilla Rolling Horizon Evolutionary Algorithm (RHEA) in the current GVGAI corpus that includes 100 games [18][19][20]. This means that at any point where the controller uses the *advance* function a counter is increased until it reaches the established limit of 900. The controllers typically use that function when they need to act and choose an action for both themselves and the opponent. Additionally, the controllers are computationally efficient so ideally they perform the same number of iterations in the same time period.

VII. Results

The results presented in this section are the outcome of the round robin tournament between all controllers as mentioned in the previous section. All results have been divided into 3 main categories, the first one representing the statistics of each controller against the others as being both the first and second player, as seen in Table III where the rows represent the model playing as player 1 and the columns show the model playing as player 2. The second being the combined statistic after swapping the opponent models as player 1 and 2, (see Table IV) and the third represents the percentage of total game wins, draws and losses (see Table VI).

Table III has a three number tuple representing the number of wins, draws and losses from each of the 10 games played from the test set. Getting the final results for a single game takes into consideration all the results from playing that game in all 5 levels 5 times. In the results from Table III a green background means that player 1 won more games than player 2, a yellow background means both players won the same number of games, and a red background implies that player 2 won more games than player 1. Therefore, an opponent model that is mostly green on its row and red on its column provides a very good performance. Based on the results from this table and the previous statement, *LimitedBuffer*, *UnlimitedBuffer* and *Random* are the top performing models. Given this it can be assumed that opponent models that expand their tree can't outperform models that avoid this step. The 900 FM calls are not enough given the extra overhead provided by expanding and simulating every action in both the *expand* and *uct* methods; from the sample MCTS controller, for both the player and the opponent.

Table IV showcases the combined results of each opponent model after they were swapped by playing as players 1 and 2. Contrary to what is shown in Table III, a green background indicates that row outperforms column as an opponent model, yellow indicates a draw among both models, and red implies that the model in column outperforms the one in row. Basically, an opponent model is considered a top performer when its row is mostly green and its column is mostly red. Table V condenses the results from Table IV indicating all the opponents beaten by each model. Ranking the opponent models by the number of controllers beaten, *LimitedBuffer* and *UnlimitedBuffer* tie as the top performing models after beating 8 of the 9 models they were matched against, followed closely by the *Random* model with 7 victories and *Probabilistic* with 6. Interestingly enough, only *UnlimitedBuffer* was able to beat the *Random* model even though it lost against *LimitedBuffer*. Next to this *Probabilistic* was the second one closest to beating the *Random* model as these two controllers were the only ones with a tie. This indicates that models with more historical data about the actions taken by each opponent; *Probabilistic* with the data gathered in the offline training and *UnlimitedBuffer* gradually adding each action to the buffer, are the only ones that can outperform or tie with the *Random* model. Thus, the fact that the player has a better understanding of how the opponent behaves over time can give it an advantage over any random action.

When taking a closer look into the percentage of number of games won per opponent model (as seen in Table VI), it is clear that the *LimitedBuffer* is the best in terms of win percentage and it is followed by both the *Random* and *Probabilistic* models in second place, leaving only the *UnlimitedBuffer* model closely behind in third. The *Random* model is better than the *Probabilistic* one when taking into account the number of defeats as the tie breaker. These results are taken from the total number of games played, 10 against each controller per player side, thus giving 90 games when being player 1 and 90 as player 2, for a total of 180 games played. Even without taking into account the individual match-ups between the opponent models, the top performers mentioned previously coincide with the results from Tables III and V. This strengthens the fact that probabilistic models (*LimitedBuffer*, *UnlimitedBuffer*, *Probabilistic*) and Random are the leading models in the current state of the framework.

Based on the results gathered it would seem like opponents that expand their tree and simulate once for every action can't compete with opponents that don't do this, the main probable reason being the fact that 900 FM calls are not enough to get both the actions from the player and the opponent. It is clear that the *Minimum* model be the worst even in percentage of victories with only 24.44% as an opponent will rarely choose the worst action all the time, the same applies to *Alphabeta*, *Fallible* and *Average*, all with less than 30% of victories as they focus too much on weighting actions based on their Q(s,a) reward values thus providing more overhead to the controllers. It is safe to assume that models with high variability in the action selected are the optimum ones, which is the case for the probabilistic models and *Random*, where the actions are selected without taking into consideration the action's rewards. As for the *Mirror* and *SameAction* models, the reason for them to perform the way they did is based mostly on the level design of the games. It also seems that all opponent models tend to struggle with certain games, most likely the cooperative ones. Table III shows that all games had at least 2 draws, corresponding to the cooperative games from the test set.

TABLE IV
RESULTS FROM THE ROUND ROBIN TOURNAMENT COMBINING THE WINS, DRAWS, AND LOSSES OF EACH CONTROLLER WITH PLAYER SWAPPING. LEGEND: 1) ALPHABETA, 2) AVERAGE, 3) FALLIBLE, 4) LIMITEDBUFFER 5) MINIMUM, 6) MIRROR, 7) PROBABILISTIC, 8) SAMEACTION, 9) UNLIMITEDBUFFER, 10) RANDOM

	1	2	3	4	5	6	7	8	9	10
1		🟩 (8,5,7)	🟩 (10,5,5)	🟥 (4,5,11)	🟥 (6,7,7)	🟥 (5,5,10)	🟥 (3,5,12)	🟥 (6,5,9)	🟥 (3,5,12)	🟥 (4,5,11)
2			🟥 (6,7,7)	🟥 (5,5,10)	🟩 (9,5,6)	🟥 (6,7,7)	🟥 (4,5,11)	🟥 (5,5,10)	🟥 (4,5,11)	🟥 (4,5,11)
3				🟥 (3,4,13)	🟥 (7,4,9)	🟥 (3,6,11)	🟥 (6,4,10)	🟥 (6,5,9)	🟥 (3,6,11)	🟥 (6,4,10)
4					🟩 (12,6,2)	🟩 (8,5,7)	🟩 (10,5,5)	🟩 (8,7,7)	🟩 (9,5,6)	🟥 (6,7,7)
5						🟥 (3,6,11)	🟥 (3,6,11)	🟥 (5,6,9)	🟥 (5,7,8)	🟥 (4,5,11)
6							🟥 (6,5,9)	🟩 (9,5,6)	🟥 (6,7,7)	🟥 (5,5,10)
7								🟩 (9,5,6)	🟥 (7,5,8)	🟨 (7,6,7)
8									🟥 (5,7,8)	🟥 (6,5,9)
9										🟩 (8,7,5)
10										

TABLE V
LIST OF OPPONENTS BEATEN BY EACH OPPONENT MODEL

Opponent Model	Controller beaten
Alphabeta	Average, Fallible
Average	Minimum
Fallible	Average
LimitedBuffer	Alphabeta, Average, Fallible, Minimum, Mirror, Probabilistic, SameAction, UnlimitedBuffer
Minimum	Alphabeta, Fallible
Mirror	Alphabeta, Average, Fallible, Minimum, SameAction
Probabilistic	Alphabeta, Average, Fallible, Minimum, Mirror, SameAction
SameAction	Alphabeta, Average, Fallible, Minimum
UnlimitedBuffer	Alphabeta, Average, Fallible, Minimum, Mirror, Probabilistic, SameAction, Random
Random	Alphabeta, Average, Fallible, LimitedBuffer, Minimum, Mirror, SameAction

TABLE VI
PERCENTAGE OF MATCHES WON, DRAWN AND LOST (WITH STANDARD ERROR) FOR EACH OPPONENT MODEL OUT OF THE 180 GAMES PLAYED AFTER SWAPPING PLAYER POSITIONS

	Win rate %	Draw rate %	Loss rate %
Alphabeta	27.22 (2.23)	26.11 (2.26)	46.66 (0.96)
Average	27.77 (2.22)	27.22 (2.23)	45 (1.17)
Fallible	25.55 (2.27)	25 (2.28)	49.44 (0.39)
LimitedBuffer	**48.33 (0.68)**	**27.22 (2.23)**	**24.44 (2.29)**
Minimum	24.44 (2.29)	28.88 (2.19)	46.66 (0.96)
Mirror	40 (1.63)	28.33 (2.21)	31.66 (2.09)
Probabilistic	45 (1.17)	25.55 (2.27)	29.44 (2.17)
SameAction	37.22 (1.82)	27.77 (2.22)	35 (1.94)
UnlimitedBuffer	43.88 (1.29)	28.88 (2.19)	27.22 (2.23)
Random	45 (1.17)	27.22 (2.23)	27.77 (2.22)

VIII. CONCLUSIONS AND FUTURE WORK

This paper presented the results of creating and comparing different opponent models with the GVGAI-2P framework. The main goal is to make a comparison between different opponent models in an offline environment, where competition parameters didn't affect the performance of the models. This was done by using the sample MCTS controller from the framework and implementing all the opponent models in it, and then playing a round robin tournament between all of them, swapping the models so they played as both players.

Based on the results from the experiment it is evident that probabilistic models (i.e. *LimitedBuffer*, *UnlimitedBuffer*, and *Probabilistic*) give the best results on the GVGAI-2P framework. The *LimitedBuffer* being the best of all the opponent models in terms of win percentage by storing the last 20 actions performed by the agent and having fresh memory of the most important moves, and the *UnlimitedBuffer* with its gradually increasing buffer with all the actions used, by being the only model capable of beating *Random*. Models that limit the number of actions performed to those available exclusively for that game (*LimitedBuffer* and *UnlimitedBuffer*) provided better results, as they don't waste actions at every time step.

Future work will focus on testing all the opponent models with the competition parameters by uploading them to the competitions website. By looking at the results it will also be worth implementing the opponent models with different algorithms besides MCTS as the performance might be different. Additionally, some of the opponent models can have slight adjustments, specially the *LimitedBuffer* and *Probabilistic* as the buffer size can be either increased or decreased. In the case of the *Probabilistic* model, data from all games available in the framework can be used for the offline phase instead of a fraction of them. Besides the previous, *Probabilistic* could record the actions of the main player from different controllers so that it doesn't overfit to a particular opponent. The previous wouldn't be difficult to implement since the website of the competition provides all previous submitted controllers for download. Additionally, the model could take features as inputs so the model learns the actions in terms of game state features. By sacrificing time, opponent models that expand and simulate the decision tree can look more than one depth further by creating the whole tree and pruning afterwards as is the case of the *Alphabeta*, *Average*, *Minimum*, and *Fallible* models. Long term work can focus on studying the impact of these opponent models on GVGP outside the framework, specifically for 2-player games.

Furthermore, detailed statistics can be gathered for each game. This can help differentiate between the performance of the models on cooperative and competitive games. If the case exists where the opponent models favor one type of game the focus should be on implementing a more universal opponent model. This would hopefully reduce the number of matches in which there's a draw between the opponents.

References

[1] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the AAAI competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005.

[2] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General video game playing," 2013.

[3] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," 2015.

[4] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[5] D. J. Soemers, C. F. Sironi, T. Schuster, and M. H. Winands, "Enhancements for real-time Monte-Carlo tree search in general video game playing," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[6] R. D. Gaina, D. Perez-Liebana, and S. M. Lucas, "General Video Game for 2 Players: Framework and Competition," in *Proceedings of the IEEE Computer Science and Electronic Engineering Conference (CEEC)*, 2016.

[7] A. J. Lockett and R. Miikkulainen, "Evolving opponent models for Texas Hold'em," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 31–38.

[8] D. Billings, D. Papp, J. Schaeffer, and D. Szafron, "Opponent modeling in poker," in *AAAI/IAAI*, 1998, pp. 493–499.

[9] J. Donkers, "Nosce hostem: searching with opponent models," 2003.

[10] R. J. Baker, P. I. Cowling, T. W. Randall, and P. Jiang, "Can opponent models aid poker player evolution?" in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. Ieee, 2008, pp. 23–30.

[11] G. M. Farouk, I. F. Moawad, and M. Aref, "Generic opponent modelling approach for real time strategy games," in *Computer Engineering & Systems (ICCES), 2013 8th International Conference on*. IEEE, 2013, pp. 21–27.

[12] M. Leece and A. Jhala, "Opponent state modeling in RTS games with limited information using Markov random fields," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–7.

[13] F. Schadd, S. Bakkes, and P. Spronck, "Opponent Modeling in Real-Time Strategy Games." in *GAMEON*, 2007, pp. 61–70.

[14] N. R. Sturtevant and R. E. Korf, "On pruning techniques for multi-player games," *AAAI/IAAI*, vol. 49, pp. 201–207, 2000.

[15] H. J. van den Herik, H. Donkers, and P. H. Spronck, "Opponent modelling and commercial games," in *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG05)*, 2005, pp. 15–25.

[16] K. Waldzik and J. Mańdziuk, "CI in General Game Playing-to date achievements and perspectives," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2010, pp. 667–674.

[17] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[18] D. P.-L. Raluca D. Gaina and S. M. Lucas, "Rolling Horizon Evolution Enhancements in General Video Game Playing," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 1–8.

[19] R. D. Gaina, S. M. Lucas, and D. Pérez-Liébana, "Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing," *arXiv preprint arXiv:1704.06942*, 2017.

[20] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2017, pp. 418–434.