The 2016 Two-Player GVGAI Competition

Raluca D. Gaina, Adrien Couëtoux, Dennis J.N.J. Soemers, Mark H.M. Winands, Tom Vodopivec, Florian Kirchgeßner, Jialin Liu, Simon M. Lucas, Diego Perez-Liebana

***Abstract*—This paper showcases the setting and results of the first Two-Player General Video Game AI competition, which ran in 2016 at the IEEE World Congress on Computational Intelligence and the IEEE Conference on Computational Intelligence and Games. The challenges for the general game AI agents are expanded in this track from the single-player version, looking at direct player interaction in both competitive and cooperative environments of various types and degrees of difficulty. The focus is on the agents not only handling multiple problems, but also having to account for another intelligent entity in the game, who is expected to work towards their own goals (winning the game). This other player will possibly interact with first agent in a more engaging way than the environment or any non-playing character may do. The top competition entries are analyzed in detail and the performance of all agents is compared across the four sets of games. The results validate the competition system in assessing generality, as well as showing Monte Carlo Tree Search continuing to dominate by winning the overall Championship. However, this approach is closely followed by Rolling Horizon Evolutionary Algorithms, employed by the winner of the second leg of the contest.**

***Index Terms*—General Video Game Playing, Multi-Player Games, Real-Time Games, Monte Carlo Tree Search, Rolling Horizon Evolutionary Algorithms, Competitions**

I. Introduction

Artificial Intelligence agents can excel at playing specific games (e.g. AlphaGo in Go [1], which uses a generic Monte Carlo Tree Search technique combined with deep reinforcement learning). Although a similar study to the one proposed in this paper was carried out by Google DeepMind on Atari games [2], there still hasn't been a considerable improvement in general agents meant to achieve a high level of play in any given unknown real-time arcade game, without prior training. That is where General Video Game Playing (GVGP) comes in, offering the challenge of an advancement towards general Artificial Intelligence through video game playing agents that are able to adapt to new scenarios, without game-specific heuristics designed by humans, and rely solely on their power of correctly judging various situations and acting appropriately.

The General Video Game AI competition (GVGAI) [3] [4] was created in order to test these general agents on a multitude of real-time games (currently 100, both stochastic and deterministic) under the same conditions and constraints. It has received significant international attention in the three years it has been running and has allowed for many interesting algorithms to be tested on the large number of problems.

A new track of this competition was added in 2016: the Two-Player GVGAI Competition (GVGAI2P) [5], which proposes an extended challenge on the original setting. The aim of this competition track is not only to analyze how agents perform in a-priori unknown real-time games (as in the original version of the contest), but also to explore how they adapt to the presence of other intelligent agents, either in a competitive or a cooperative environment. Particularly, one of the most interesting aspects of this competition is that it encourages agents to consider a model of the other player's behavior. This not only addresses the most likely actions the other player will perform next, but also its intentions - is it competing or cooperating? - and its knowledge about the world - what has the other agent discovered about the game?

This paper is an extension of the competition setup [5], offering insight on the best agents submitted and an analysis of their performance on four game sets. Therefore, the authors of this paper are the top 5 participants in the final rankings, together with Raluca D. Gaina, Simon M. Lucas and Diego Pérez-Liébana as the competition organizers.

The remainder of this paper is structured as follows: Section II looks at other work already published in the area of GVGP and similar competitions, Section III presents the details of the GVGAI2P framework, Section IV shows the competition setting and ranking methods, Section V gives an overview of the agents submitted, Section VI introduces the results of the controllers on all game sets used in the competition, and Section VII concludes the paper with notes on the importance of this research and further developments.

Raluca D. Gaina, Jialin Liu, Simon M. Lucas (School of Electronic Engineering and Computer Science, 10 Godward Square, Queen Mary University of London, Mile End Rd, London E1 4FZ, UK; work done while at University of Essex; email: {r.d.gaina,jialin.liu,simon.lucas}@qmul.ac.uk),
Adrien Couëtoux (Institute of Information Science (IIS), Academia Sinica, 128 Academia Road, Section 2, Nankang, Taipei 115, Taiwan; email: adrienc@iis.sinica.edu.tw),
Dennis J.N.J. Soemers (Artificial Intelligence Lab, Vrije Universiteit Brussel, Boulevard de la Plaine 2, 1050 Ixelles, Belgium; work done while at Maastricht University; email: dsoemers@ai.vub.ac.be),
Mark H.M. Winands (Games and AI Group, Department of Data Science and Knowledge Engineering, Maastricht University, P.O. Box 616, 6200 MD, Maastricht, The Netherlands; email: m.winands@maastrichtuniversity.nl),
Tom Vodopivec (Faculty of Computer and Information Science, University of Ljubljana, Večna pot 113, Ljubljana, Slovenia; email: tom.vodopivec@fri.uni-lj.si),
Florian Kirchgeßner (Ludwig-Maximilians-Universität München, Geschwister-Scholl-Platz 1, 80539 Munich, Germany; email: Florian.Kirchgessner@gmx.de),
Diego Pérez-Liébana (School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK; email: dperez@essex.ac.uk)

II. Related Research

The Google AI Challenge (2009-2011)[1] is one example of a competition that was successful on an international scale. It was organized by the University of Waterloo's Computer Science Club and it looked at different specific games each year, such as Ants (multi-player) in 2011 and the Planet Wars and Tron (two-player) in 2010. Although this framework was similar to the GVGAI Competition, the winning entries were mainly driven by domain-specific heuristics. The ranking system used in this competition was based on Elo scoring.

Online rankings used in the GVGAI2P Competition need to be fast in order to provide feedback to competitors and return results on time. Samothrakis et al. [6] considered the issue of slower convergence achieved by the Elo system, along with competitor pairing and accurate performance ranking. In their paper, they look at the Ms Pac-Man vs Ghosts Competition, in which agents can be submitted either for Pac-Man or the group of ghosts. Two ranking systems tested in the paper, Glicko [7] and Bayes Elo [8], which score same-type entries depending on their win rate. The results suggest Glicko to be the better method, due to the low number of games needed for rankings to converge. Glicko-2 [9] is an improved version over Glicko which adds a volatility ranking to better measure variation of the player's rating (σ; see Section IV-A). These findings and improvements motivated the decision of using the Glicko-2 system in GVGAI2P.

Another multi-player competition is the General Game Playing Competition (GGP) [10], running at the Association for the Advancement of Artificial Intelligence (AAAI) and presenting numerous board and puzzle deterministic turn-based game environments, both single and multi-player. The structure of this competition is different to GVGAI, using several eliminatory stages to declare a winner. The best performing algorithm is tested against an expert human player as its final biggest challenge. This competition provides the users with the rules of the games played, while GVGAI offers no knowledge beyond the current game state, a video-game style reward structure and the action space. The difficulty of the GVGAI games is increased to real-time as well, by reducing the agents' thinking time to milliseconds for every move (seconds for GGP).

The Geometry Friends Game AI (GFGAI) Competition [11] comes closer to the structure of GVGAI, in the sense that it has multiple tracks (two single player and one cooperative two-player), it offers a sample Reinforcement Learning controller for an easy start, and it uses different level sets for training and testing (5 levels in each). However, the competition does focus on a single real-time game, in which the two characters have different abilities and need to cooperate in order to win, whilst also being affected by physics. The final rankings are calculated based on the time taken to complete levels and the number of diamonds collected.

Several other competition feature multi-player real-time decision making. The Ms Pac-Man versus Ghost Team competition [12], [13] offers participants the possibility of submitting entries to control either Ms Pac-Man or the Ghost Team, as well as introducing partial observability in the latest editions. The Starcraft AI competition [14] look at Starcraft: Brood War (Blizzard Entertainment, 1998), a complex Real-Time Strategy game also feature partial observability and thousands of actions available at every game step.

The Fighting Game AI Competition [15] focuses on the fighting game genre. Each skill used by the players has three sub-components that affect their decision process depending on the stage they reach in each move. This idea relates to the dynamic nature of the availability of actions in GVGAI, which can vary during a game. Finally, a similar, but turn-based, adversarial game is featured in the Showdown AI Competition [16], which tests the skill of AI in Pokemon (Nintendo, 1996).

When trying to create a successful competition, it is important to be aware of the various features which could help in such an endeavor. Togelius discusses this [17], suggesting that transparency, reliability, persistence and availability are most important. In addition, he recommends an easy start with competition entries for participants, as well as allowing user feedback and active peer support discussions and focusing on evolving the competition in the future. The paper also outlines the importance of game AI competitions in benchmarking algorithms and attracting new research interest in the field.

III. Framework

A. Video Game Description Language

The games in the GVGAI framework[2] are written using a Java port of the Video Game Description Language (VGDL) [18], originally written in Python and inspired by the Game Description Language (GDL) [19] used in the GGP Competition and in particular by the LUDI GDL used by Browne and Marie [20]. The GVGAI2P games are restricted to 2D real-time grid games, where avatars are limited to horizontal and/or vertical movement, together with one special action having various game-specific effects.

Two text files are needed to define a game in VGDL: one for the game description and the other for the level definition. The first is separated into four sections, describing the entities present in a game, their interactions, their representations in the level file, and the end game conditions. The level file contains an ASCII character matrix of the sprites, depicting their positions in the level. Separate Java classes are used to implement the functionality of all elements defined.

Although the level file structure was not modified at all for this track of the competition, the number of players must be specified in each game description (1 by default), and terminations and interactions must also now identify their effects for both players (e.g. $scoreChange = 2, -5$ would mean the first player receives 2 points, while the second loses 5 as a result of this interaction).

The end of the game can be reached with several different outcomes: one player winning and the other losing, both players losing or both players winning. This is not necessarily an indication of the game type (cooperative or competitive), as two winners in a competitive game signifies a draw.

[1] http://aichallenge.org/

[2] www.gvgai.net

B. State Observation and Forward Model

In GVGAI2P, the only information received by AI agents is a current state description, encapsulated in game objects.

These specify the current scores of all players, the game tick, a grid of observations (which could contain more than one at a specific position in the level if objects overlap; an observation contains the category, type and position of a game object), and a history of interactions that had occurred in the game up until the current step. Additionally, agents can query the objects for specific lists of observations, grouped by their type, such as resources, NPCs, portals, etc. They can query all avatars in the game on their legal actions, position, speed and orientation.

The agents do not only have access to the current game state, but possible future ones as well through the forward model provided. It is important to note this forward model is imperfect in stochastic games, offering just one of many possible next states. In order to advance the state in multi-player games, the agents must supply an array of actions for all players, therefore needing to guess what the other players would do in the current situation. This proposes an interesting opponent modeling problem.

No information about the game rules is received by the agents, although these may be estimated (unreliably) from the interaction history (what effect certain interactions had on the game score, what is the goal of the game, etc.).

C. AI Agents and Game Cycle

A game in GVGAI follows a regular cycle: the system offers the game state to the agents and they reply with an action. The actions available may vary and attempting to return an illegal one results in no action being performed. Other actions available are for movement (up, down, left, right) and a special game-dependent action. This range of actions was considered to be enough to cover most game types included in the competition, although they may be extended in the future.

Agents submitted to the competition must comply to the competition time budgets of 1 second of CPU time in the constructor, used for game preparation, and 40 milliseconds of CPU time at every game tick, to determine which action the agent chooses to play. Exceeding this budget results in the controller being disqualified. The agents are aware of their player ID in order to query for the correct information.

Not all information about the current game state may be available to a player, as some of the games in the framework are partially observable.

IV. Competition

A. Ranking System

The ranking system employed in the GVGAI2P Competition is Glicko-2, an improved version of the original Glicko system created by Mark E. Glickman. The players have their performance measured using three different values: a rating (r), a rating deviation (RD) and a volatility (σ), taking on the default values of (1500, 350, 0.06) for an unranked player, as devised in the initial study of this system by Glickman. While the rating is the only value the players are aware of, similar to Elo scoring, the system makes use of the other two in order to adjust the r value as necessary and carry out a more accurate analysis of a player's skill level.

The rating deviation portrays the confidence in a player's r value, varying depending on how often the player is involved in games (as little information is known about a player who has not played in a while and their skill level is expected to change). RD can be used in conjunction with the rating to display a more accurate interval instead of a single value for the player's performance. Finally, σ is the expected fluctuation of a player's rating, being lowest if the performance is kept constant. All of the values are updated at the end of a pre-defined rating period of several games, for all players, with RD and σ influencing r.

In the GVGAI2P Competition, this system was adjusted slightly in order to better suit the application. Each player receives a 3-tuple (r, RD, σ) for each individual game it plays in, as well as overall values, averaged over all games in a set. The rating period is set to one run, which leads to the values being updated for only 2 controllers, after they finish playing all of the matches in one run.

Lastly, the default values for each game need to be adjusted regularly depending on the performances during certain games, as well as kept on a limited scale ($0 - 5000$). This is due to the variety in games and performance: a general default would result in a larger number of games needed for rankings to converge. Therefore, the game specific defaults are regularly recalculated as the average of all scores in that particular game.

For example, in the game "Capture Flag", the average score is 490. Assuming a new controller is of average quality, if it were to start with the default rating of 1500, it would require more games to be played for their rating to be accurate, than if they instead started from the average rating. The most obvious benefit of this default values adjustment can be seen in difficult games, where ratings remain close to 0 for all controllers.

B. Games

The games in the competition range from puzzles to shooters, both competitive and cooperative, although this information is not offered to the players. They differ in many ways, including the environment interactions available, scoring systems and end conditions. For example, in the cooperative game "Akka Arrh", the players are required to defend a locked spaceship from aliens, while finding the key in order for both players to enter the ship and win the game. 2 points are awarded for each enemy killed and 10 upon entering the spaceship. For more details on the games, the reader is referred to the GVGAI Competition website and [5].

Each game has 5 levels, which differ in term of sprite layout or sometimes sprite properties (e.g. sprite speed or spawn point probability), without changing their behaviour.

Three different types of game sets, containing 10 games each (Figure 1), are used in the competition to avoid overfitting, as mentioned previously: training, validation and test. The training games are public and available with the framework. The validation games are secret, but available for submission

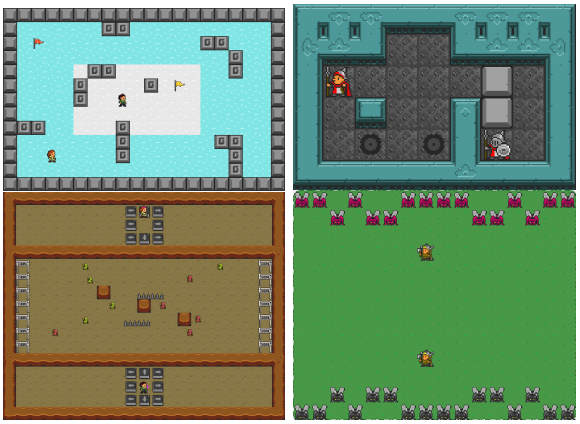

Fig. 1. Games in GVGAI2P Framework: Capture the Flag, Sokoban, Bee Keeper and Minions (from left corner, clockwise).

on the website for feedback on agent performance. Finally, the test games are secret and unavailable until the end of the competition, when the results are announced. All game sets contain the same game feature distribution (scoring systems, termination condition types, etc.); for more details, the reader is referred to [5].

C. Competition rules

The competition is hosted on the GVGAI website and all registered users may participate. The competitors may use sample agents as a starting point for their entries, an attempt to facilitate the initial process.

Controllers which meet competition requirements are paired for runs on each of the game sets available. In the training and validation sets, the pairing is done based on the Glicko-2 RD values (see Section IV-A) for quick feedback in online rankings and ease of readjustment with new entries submitted. In the test set, a round robin tournament style is employed instead for offline ranking, due to the relatively small number of entries, so controllers with the fewest games played are prioritized.

In the training and validation sets, the agents play 1 randomly selected level of each of the 10 games in one set per run, with positions swapped in the second game for each match, therefore 20 games in total. In the test set, the agents play all 5 levels of all 10 games, swapping positions for each match, therefore 100 games in total per run. All levels are played for final rankings with the motivation that general agents should be able to generalize across levels of a single game as well.

As the GVGAI2P Competition ran two legs in 2016, at the IEEE World Congress on Computational Intelligence (WCCI) and the IEEE Conference on Computational Intelligence and Games (CIG), the agents were tested on four game sets in total. The participants had the chance to update their entries between legs. The validation and test sets from the first leg were used for training and validation in the second leg, respectively. Full details of all games are available on the website.

The controllers receive points on each game in a set based on their performance, following a Formula-1 scoring system: the first ranked is awarded 25 points, the second ranked receives 18, then 15, 12, 10, 8, 6, 4, 2, 1 and 0 for the rest of the participants. The performance is measured by Glicko rating in training and validation sets (using win percentage, in-game scores and timesteps as tie-breakers, in this order), and by win percentage in test sets (using in-game scores and timesteps as tie-breakers, in this order). The points in all games are summed up to report an overall performance in one game set. The overall points in the test sets are then summed up at the end of a competition year for the final Championship rankings, a structure that has been already used in the literature [21]. Note that, albeit a difference in F1 scores is not necessarily significant, this ranking system provides a valid procedure to determine a competition winner for GVGAI, as it both rewards generality across games and higher positions in the rankings per scenario. The use of an F1 scoring system is not new in the competitions in the field [21], [22].

V. Controllers

This section presents the sample controllers included in the framework, followed by the competition entries in the order of their final ranking. All of the subsections follow a similar structure for ease of comparison: an overview of the algorithm implemented and its opponent model, its use of online and offline learning, strengths and weaknesses of the agents and any domain knowledge included in their heuristics.

A. Sample Controllers

Two of the sample controllers are very simple methods: *DoNothing* (does not perform any action) and *SampleRandom* (returns a random action from those available). The other four are more advanced and use two different types of heuristics to approximate the values of game states.

The *SimpleState* heuristic looks to maximize the game score, while also minimizing the number of NPCs and the distance to NPCs and portal. In addition, a large penalty is applied for losing the game and a large bonus is awarded for winning.

The *WinScore* heuristic is a stripped down version of *SimpleState*, taking into account only the game score and the end state cases (bonus for winning and penalty for losing).

1) SampleOneStepLookAhead (OneStep): This controller is the most basic of the four advanced agents. It uses the *SimpleState* heuristic to analyze all of the actions available to it in one game step. The action chosen for execution is the one which leads to the highest value returned by the heuristic. This agent implements a simple opponent model which assumes the opponent will do a random move, as long as it will not result in them losing the game. Therefore, the next step is checked for the opponent as well, using *ACTION_NIL* as the action performed by the agent in question.

2) SampleGA: This controller uses one of the two most advanced techniques, a Rolling Horizon Evolutionary Algorithm (RHEA) [23], taking sequences of actions as individuals and applying various evolutionary methods, including mutation, uniform crossover, and tournaments. The action returned is the first of the best individual found at the end of the process. Each individual's fitness is calculated by advancing the forward model through the sequence of actions and evaluating the

game state reached at the end with a *WinScore* heuristic. This algorithm uses a random opponent model (i.e. assumes the opponent will do a random move out of those available).

3) SampleMCTS: This controller uses a Monte Carlo Tree Search (MCTS) [24] technique, which keeps game states and statistics gathered in the nodes of the tree, while advancing through the game with the available actions, employing a random opponent model throughout. It follows four steps: selection, expansion, simulation and back-propagation.

The algorithm first searches the tree for a node not yet fully expanded, using the tree policy (UCB1 in the sample controller, $C = \sqrt{2}$, meant to reach an appropriate balance between exploration and exploitation; see Equation 1). It then expands this node and performs a Monte Carlo simulation from the new node added to the tree, during which actions are randomly selected to advance the state, until a set limit has been reached. The resultant state is evaluated using a *WinScore* heuristic and the value is backed up the tree.

$$a^* = \underset{a \in A(s)}{\arg\max} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \quad (1)$$

4) SampleOLMCTS: This controller is an Open Loop [25] version of the sample MCTS, the difference lying in the fact that this algorithm does not store the game states in the tree nodes. Therefore the forward model is used repeatedly to re-evaluate states, giving a more accurate representation in stochastic games in particular. The heuristics, policies and opponent model used are the same as in the MCTS agent.

B. 1^{st} - Championship Winner - adrienctx (VA-OLETS) - Adrien Couëtoux

1) Background: Value Approximation Open Loop Expectimax Tree Search (VA-OLETS) is an upgraded version of a previous controller, OLETS [3], which is itself inspired by [26]. This controller attempts to fix one of the main weaknesses of tree search based agents, namely the absence of generalization between observations. It does so by continuously learning an approximate value function depending on observations and then injecting it into the inner workings of the tree search.

2) Main Algorithm: OLETS (see Algorithm 1; we note P(n) the parent node of n, which is empty if and only if n is the root node, and s(n) the first state observation seen in node n) is a tree-based open loop controller, similar to the sampleOLMCTS agent. It stores sequences of actions in a tree, as well as their associated statistics (empirical average reward, standard deviation, number of simulations, etc). It then uses this data to bias further simulations, virtually pruning actions that have shown poor empirical performance in past observations. To avoid building intractably tall trees, OLETS adds only one node per simulation. It evaluates the newly added leaves by using the game score, which serves as an approximation for the real value of these nodes.

The main innovation of VA-OLETS is to learn an approximation of the value function from past observations, instead of relying solely on the game score. In this specific implementation, the approximation relies on a linear model. $\bar{V_\theta}(\varphi(s)) = \theta\varphi(s)$ represents the value of being in state s, given a parameter vector θ, and input features $\varphi(s)$. The resulting approximation $\bar{V_\theta}(\varphi(.))$ is used within VA-OLETS to give leaves a value called r_M, which is then back-propagated through the tree and thus used to balance simulations between branches.

There is one input feature for each category and type of visible object: the distance between the avatar and the closest object from that category and type. All distances are normalized, so that each $\varphi(s)$ element is between 0 and 1.

These features are also used to improve the controller's exploration. While OLETS gives a bonus to unexplored places in the observation grid, VA-OLETS gives a bonus to unexplored points in the feature space. For example, if the avatar has never been near an object of a certain category and type, the controller will assign a positive bias to state observations where the avatar is close to any object of that category and type, regardless of its specific location in the grid. This is designed to increase the exploration of the feature space, in order to improve the quality of the value approximation.

The opponent model is random, to make simulations as fast as possible and build a large tree at each time step. The same configuration of the algorithm was submitted to both competition legs.

3) Online and Offline Learning: VA-OLETS learns a good approximation of the value function by minimizing a loss function, based on the current model and observed data. An observation is defined as a 4-tuple (s, a, s', r), with s being the initial state, a the chosen action, s' the next state, and r the immediate reward measured as a game score change. The loss function is then defined as the $L2$ norm of $\bar{V_\theta}(\varphi(s)) - r$. During the online phase, after each new observation, θ is updated by mini batch Stochastic Gradient Descent (SGD) [27]. Mini batch SGD allows for fast updates that respect the limited online time budget in this challenge.

During development, the training set games were used to choose the learning rate and mini batch size for the SGD.

4) Strengths and Weaknesses: VA-OLETS obtained particularly good results on games that are about collecting multiple items or hunting another moving object. This is probably due to an efficient exploration of the feature space during the first few seconds of a match and the learned approximate value function afterwards. This controller's performances were the poorest on games with long term objectives, such as racing games. Since the search tree is kept short, in favor of its width, it is not surprising that VA-OLETS fails to learn about events that happen many time steps down the line.

5) Domain Knowledge: The only domain knowledge that VA-OLETS uses is the definition of the input features. These rely on the human observation that, for most games, an important information is the distance between the avatar and the closest object of each category and each type. As a result, if there are multiple objects of a certain category and type, VA-OLETS implicitly neglects all but the closest one.

C. 2^{nd} - MaastCTS2 - Dennis Soemers

1) Background: MaastCTS2 uses Open Loop Monte-Carlo Tree Search (MCTS), like the *SampleOLMCTS* sample controller, with a number of enhancements. It is computationally

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCIAIG.2017.2771241, IEEE Transactions on Computational Intelligence and AI in Games

Algorithm 1 VA-OLETS

procedure VA-OLETS(s, T)
  $\mathcal{T} \leftarrow$root ▷ initialize the tree
  $Data \leftarrow \emptyset$ ▷ initialize observed transitions
  Initialize $\bar{V}_\theta(\varphi(.))$
  while elapsed time $< T$ **do**
    RUNSIMULATION($\mathcal{T}$, s, $\bar{V}_\theta(\varphi(.))$, $Data$)
    UPDATEMODEL($Data$)
  return action $= \arg\max_{a \in C(root)} n_s(a)$

procedure RUNSIMULATION($\mathcal{T}$, s, $\bar{V}_\theta(\varphi(.))$, $Data$)
  $n \leftarrow$root($\mathcal{T}$) ▷ start by pointing at the root
  $Exit \leftarrow$False
  while $\neg Final(s) \wedge \neg Exit$ **do** ▷ Navigating the tree
    if n has unexplored actions **then**
      $a \leftarrow$Random unexplored action
      $s \leftarrow$Forward Model(s, a, $randomOpponent$)
      $Data \leftarrow$ add this transition
      $n \leftarrow$NewNode(a, Score(s), $\varphi(s)$)
      $Exit \leftarrow$True
    else ▷ use node scoring to select a branch
      $a \leftarrow \arg\max_{a \in C(n)} r_M(a)$ w.p. ϵ, random otherwise
      $n \leftarrow a$
      $s \leftarrow$Forward Model(s, a, $randomOpponent$)
      $Data \leftarrow$ add this transition
  $n_e(n) \leftarrow n_e(n) + 1$
  $R_e(n) \leftarrow R_e(n) + Score(s)$
  while $\neg P(n) = \emptyset$ **do** ▷ update the tree
    $n_s(n) \leftarrow n_s(n) + 1$
    $r_M(n) \leftarrow \frac{R_e(n)}{n_s(n)} + \frac{(1-n_e(n))}{n_s(n)}(\bar{V}_\theta(\varphi(s(n))) + \max_{c \in C(n)} r_M(c))$
    $n \leftarrow P(n)$

procedure UPDATEMODEL($Data$)
  Run mini batch SGD on the $Data$

expensive to advance game states in the GVGAI framework. This means that MCTS-based controllers can only base decisions on relatively few simulations. Therefore, the main goal of the search enhancements in MaastCTS2 is to make every simulation as informative as possible.

2) Main Algorithm: Before running MCTS, MaastCTS2 starts with a one-ply Breadth-First Search in every game tick to prune unsafe actions. This is referred to as *safety prepruning* and was previously used in Iterated Width in GVGAI [28]. *Progressive History* [29] and *N-Gram Selection Technique* [30] are used as the *selection* and *playout* policies of MCTS, respectively. These policies introduce a bias towards playing actions that performed well in previous simulations. The search tree generated by MCTS in one frame is stored in memory, and parts of the tree that may still be relevant are reused in the next frame. These enhancements have previously been described in more detail [31] for the single-player variant.

The version submitted to WCCI 2016 uniformly chooses random actions for the opponent in the selection and playout steps of MCTS. The opponent model for the version submitted to CIG 2016 is more advanced. For every action a that an opponent can play, every node keeps track of an opponent score $Q_{opp}(a)$. The value of $Q_{opp}(a)$ in a node is the average of all the scores for the opponent, back-propagated through that node in simulations where a was selected as the opponent action in that node. Opponent actions are selected using an ϵ-greedy strategy. In every node, a random action is uniformly chosen with probability $\epsilon = 0.5$, and the action a that maximizes $Q_{opp}(a)$ is selected with probability $1 - \epsilon$.

3) Online and Offline Learning: MaastCTS2 does not use any offline learning. It has various parameters that could likely be better tuned through offline learning, but, for the competitions of 2016, they have only been tuned manually.

For every object type i observed during a game, MaastCTS2 keeps track of a weight w_i. These weights are adjusted online based on experience gathered through MCTS simulations. Intuitively, w_i is intended to have a high value if interactions with objects of type i have been observed to lead to score gains or game wins, and it is intended to have a low value if interactions with such objects have been observed to lead to score or game losses. These weights are used in a heuristic evaluation function, which is described below. More details on the implementation and references to previous research that inspired this idea can be found in [31].

The selection and playout policies used (Progressive History and N-Gram Selection Technique) can also be viewed as a form of online learning. These policies are biased towards actions which had a positive outcome in previous simulations.

4) Strengths and Weaknesses: MaastCTS2 does not perform well in situations where the heuristic evaluation function (described in more detail below) does not correlate with the game-theoretic value, and changes in game score are sparsely observed. In such situations, MCTS cannot perform like a Best-First Search algorithm and instead explores the search space uniformly. Examples of such games are *Tron*, *Samaritan* (only as player two), and *Team Escape*.

The agent appears to perform particularly well in games where changes in game score can be observed frequently, or where its heuristic evaluation function provides useful information. In such situations, the MCTS algorithm naturally dedicates more search effort to more promising parts of the search tree and the online learning techniques also provide more useful information.

5) Domain Knowledge: MaastCTS2 uses domain knowledge in a heuristic evaluation function for evaluating non-terminal states at the end of MCTS simulations. This is done because many simulations are cut off early, before resulting in a terminal game state. This evaluation function computes a linear combination of five features for a game state s:

- A feature that rewards proximity to objects of type i with positive learned weight w_i, and punishes proximity to objects of type i with negative weight w_i. This models the importance of interacting with objects.
- A feature that rewards gathering resources. In many games, it is beneficial to collect resources and they should only be used for a specific purpose (which typically provides a score gain able to offset the punishment).

- A feature that rewards the agent if s has a lower number of objects of type i with a negative weight. Therefore, it considers it good to get rid of harmful objects.
- A feature that punishes the agent if there are many movable objects in s adjacent to walls or other obstacles. This heuristic was implemented specifically for puzzle games, like *Sokoban*, where the goal is to push boxes towards certain goals; if boxes are against walls, the puzzle becomes impossible to solve.
- A feature that punishes the agent for objects of type i with a weight $w_i > 0.5$ that are completely surrounded by obstacles, including the avatar. It is considered that objects with large weights should remain reachable.

The output of this heuristic function is finally added to a regular evaluation function, which takes into account the game score and adds or deducts a large constant value for wins or losses. These features were mainly identified as being useful heuristics in the single-player variant of the agent, so they were used in the two-player variant as well. None of the features are useful in *all* games, but they are also rarely detrimental.

D. 3^{rd} - WCCI Leg Winner - ToVo2 - Tom Vodopivec

1) Background: The ToVo2 controller was inspired by two AI fields: reinforcement learning (RL) [32] and Monte Carlo Tree Search (MCTS) [24]. The former offers strong techniques for updating the gained knowledge, whereas the latter offers strong selection and exploration policies. Considering this, for the backbone of this controller, the *Sarsa-UCT(λ)* algorithm [33] was chosen, which combines the advantages of both fields – it is a generalization of the UCT algorithm [34] with the Sarsa(λ) algorithm [35]. UCT is a very general and widely-adopted MCTS algorithm, whereas Sarsa(λ) is one of the most established and well-understood temporal-difference (TD) learning [32] algorithms. Sarsa-UCT(λ) is an instance of a *temporal-difference tree search* algorithm.

2) Main Algorithm: ToVo2 is implemented on top of the sampleOLMCTS controller from the GVGAI Framework. The value-function is represented by a direct-lookup table that is incremented online. Each entry maps one state-action pair, therefore there is no generalization across the state space – there is no value-function approximation and no use of features. Transpositions are not considered (hence a tree is built and not a directed graph). The representation does not memorize state observations, so the forward model is invoked at each step of both the MCTS tree and playout phases.

Combining the UCB1 policy [36] with TD-learning requires local normalization of value estimates [33]: for each tree node, the algorithm remembers the all-time minimum and maximum values of its children nodes and uses these as bounds for normalization to $[0, 1]$, before computing the UCB value.

A configuration of Sarsa-UCT(λ) similar to the standard UCT [24] algorithm (used in the sampleOLMCTS controller) has been submitted to several single-player GVGAI competitions (the ToVo1 controller). ToVo1 is regarded as a proof of concept and does not achieve top positions, since its configuration is very basic.

In contrast, for the two-player competitions, Sarsa-UCT(λ) was configured to exploit more of its potential. The algorithm observes not only the final reward, but also intermediate rewards (the same as the original UCT algorithm) by computing the difference in score after every game tick. It expands the tree with all the visited nodes in an iteration and retains it between searches. It gradually forgets old knowledge (through the updated step-size parameter of RL methods) and it searches for the shortest path to the solution through the reward discount factor (a parameter also in the original UCT). The opponent is modeled as a completely random player and is assumed as part of the environment – its value-estimates are not memorized. The scoring considers the win state of the opponent with a weight of 33%, compared to the weight of the own win state.

The ToVo2 controller is augmented with two specifically-designed enhancements related to the MCTS playout phase:

- *Weighted-random playouts*. The algorithm performs a weighted random selection of actions in the playout. The weight of each action is set uniformly randomly at the beginning of each playout. This causes the avatar to be more explorative and revisit the same states less often.
- *Dynamic playout length*. The ToVo2 controller starts each search with short playouts and then it prolongs them as the number of iterations increases. This emphasizes the search in the immediate vicinity of the avatar, but, when there is enough computation time, also the search for more distant goals that might be otherwise out of reach.

The same configuration of the algorithm was submitted to both competition legs.

3) Online and Offline Learning: The algorithm uses no prior knowledge and no offline training. During development, the learning parameters were tuned manually through experimentation. The following are the parameter values: exploration rate $C_p = \sqrt{2}$, reward discount rate $\gamma = 0.99$, eligibility trace decay rate $\lambda = 0.6$, 50% of knowledge forgotten after each search. The dynamic playout length starts at 5, then increases by 5 every 5 iterations, up to a maximum length of 50.

The value-normalization method adapts the bounds for each node, which is similar to tuning the exploration rate C_p online.

Online learning is implicit to the MCTS backup process: as described above, the algorithm uses TD-learning backups instead of Monte Carlo backups for state-value estimation.

4) Strengths and Weaknesses: The main strength of the ToVo2 agent is its generality. It can be applied as a backbone to other, more complex, methods in the same way as basic MCTS algorithms, while performing better. Furthermore, it has plenty of unexplored potential, given that it currently ignores state observations, does not generalize across states, does not use transpositions and uses primitive opponent-modeling.

It has similar strengths and weaknesses as classic MCTS algorithms: its performance is high in arcade games with short-term rewards, average in arcade games with long-term rewards and poor in puzzle games. Its weighted-random playouts are beneficial in traditional arcade games where exploration of the two-dimensional space is desirable, but can be detrimental in puzzle games, where the exact sequence of moves is critical.

ToVo2 achieved top positions in most games from the first three sets, but performed poorly on the fourth set, where in half of the games it scored even less than the sampleOLMCTS controller. Rough experiments confirm this is due to the

configuration of the dynamic playout length enhancement. We analyzed a number of qualitative features of these games, but we identified no significant correlation.

5) Domain Knowledge: The algorithm uses the two heuristic-based enhancements described previously: dynamic playout length and non-uniform random playouts. Otherwise, apart from the manual optimization of parameters, it uses no other expert or domain knowledge and no handcrafted rules.

E. 4^{th} - CIG Leg Winner - Number27 - Florian Kirchgeßner

1) Background: Like the sampleGA controller, Number27 consists of a genetic algorithm. But, instead of using separate populations with multiple action sequences, a single one is used, as the time distribution to improve multiple populations results in overall weaker results. MixMax [37] was first used as a risk-seeking behavior for the MCTS algorithm and was herein adapted to achieve the opposite result, namely to make the algorithm more defensive in the vicinity of opponents.

2) Main Algorithm: While the genetic algorithm ensures proper local movement, a heuristic permits the player to look beyond the horizon of simulated sequences. Its purpose is to encourage the urge to explore and getting a sense of which objects are valuable. With the knowledge of each object type's worth, a value map is created by combining each object's corresponding range of influence (see Figure 2). The result is similar to a heat map, with areas of varying attractiveness formed by an especially beneficial object, clusters of many low-valued ones or the presence of threats.

Fig. 2. Value map of the game *Zelda* after 10 game ticks.

As the value map does not guarantee proper movement across the level and results in the player getting stuck in corners, dead ends or enclosed sections, a penalty for remaining in the same area was introduced. Additionally, this technique encourages exploration when the player only exploits a single source of points and resembles a pheromone-based approach [25]. When eventually choosing the optimal action, a distinction is made between deterministic and non-deterministic conditions. If the same arrangement results in a different score, then the likelihood of certain outcomes has to be considered. Whenever the following states are deterministic, the action with the highest simulated score is chosen. However, for differing results, a variation of the MixMax algorithm is used. This approach combines the average scores of sequences with the same initial action and the maximal score out of the same sequences using Equation 2.

$$(1-maxFactor)\cdot avgScore+maxFactor\cdot maxScore \quad (2)$$

The proportions are reflected in $maxFactor$ and are determined by the number of divergent game states: With more differences, the maximal score is taken into account less, which results in a more cautious play style.

During simulations, the opponent's action is chosen completely at random, while in previous implementations losses were being avoided. However, making assumptions about which action will be taken made the controller vulnerable to unexpected outcomes and the loss avoidance took valuable computation time. With a random approach, all movement possibilities are considered and, when the opponent gets close, the player is more wary of them.

The same algorithm was submitted to both competition legs, but with different configurations.

3) Online and Offline Learning: While no offline learning was conducted, the algorithm is capable of evaluating the level during the game. The frequency of an object type is used as an initial value estimate, with a higher score for unique objects and a lower score for common ones. As an encouragement to inspect all existing types, an exploration bonus increases the basic value. This bonus is not only applied at the beginning of the game, but also when meaningful development occurs. Eventually, the change in game score after interacting with an object is determined, by monitoring the consequential events created by the framework and by assigning the game score difference to the object type stated in the event.

4) Strengths and Weaknesses: Because the genetic algorithm has a fixed simulation depth, the controller is incapable of solving puzzle games. It also does not understand the need for teamwork, since the second player is considered a randomly acting object. On the other hand, Number27 is able to quickly evaluate the game by inspecting existing objects and getting an overview of the level. The result is the ability to progress through the majority of the games at a respectable rate. Furthermore, adjusting risk and caution rates allows the player to survive levels with multiple aggressive enemies.

5) Domain Knowledge: The controller does not differentiate between the various game genres and tackles all in the same way. Only for axis-restricted games, the range of object influences is adjusted to allow the player to notice them on the value map. Therefore, the algorithm is able to score across most games in the test and validation sets and fits the intention of creating a general video game playing controller.

F. 5^{th} - CatLinux - Jialin Liu

1) Background: Rolling Horizon Evolutionary Algorithms (RHEAs) are inspired by Rolling Horizon planning (sometimes called Receding Horizon Control (RHC) or Model Predictive Control (MPC), which is commonly used in industry), together with simulation-based algorithms, for online decision making in a dynamic stochastic environment. In a state, RHEAs evolve a population of action sequences (individuals), where the length of each sequence equals the fixed depth of simulation. An optimal action sequence is recommended based on a forecast of the finite time-horizon. Only the first action of the optimal sequence is applied to the problem, then the prediction horizon is shifted forward and the optimization is repeated with updated observations and forecasts.

Perez et al. [23] applied a Rolling Horizon Genetic Algorithm (RHGA) to the Physical Traveling Salesman Problem (PTSP). RHGA is shown to be the most efficient evolutionary algorithm on the GVGAI Competition framework [25]. A Rolling Horizon version of a Coevolution Algorithm (RHCA) was introduced and applied to a two-player battle game by Liu et al. in [38].

2) Main Algorithm: Different to previous work, where the population is reset at each time step, CatLinux uses a so-called *shift buffer* technique in RHGA. As a fixed-length moving optimization horizon is used at each time step, the actions in each sequence are shifted along with the movement of the prediction horizon, then the action at the end of each sequence is randomly initialized. The parameters, population size λ, number of elites μ, mutation probability p and simulation depth D are arbitrarily set to 10, 4, 0.1 and 10, respectively.

The same configuration of the algorithm was submitted to both competition legs.

3) Online and Offline Learning: The controller does not involve any offline learning. Further work would consider the use of offline learning to recommend action sequences to start with, instead of using a randomly initialized population. CatLinux uses implicit learning from the RHEA method, similar to the sampleGA controller.

4) Strengths and Weaknesses: CatLinux is simple and efficient. The *shift buffer* makes use of the information obtained during the previous generations and accelerates the convergence to the optimal sequence. The main weaknesses are (i) the use of a random opponent model and (ii) at each game tick t, the $legalActions$ is the set of actions available from state s_t, which possibly differs to the set from state s_{t+1}. The latter cannot be solved by simply updating the $legalActions$ during the simulations, as applying an identical action twice to an identical state can result in two distinct states if the game is stochastic. The controller performed strongly on puzzle games, but it scored below average in cooperative games.

5) Domain Knowledge: The only domain knowledge CatLinux uses is the legal actions of both players, the score, and the winning state of the current player. The parameters are arbitrarily chosen: neither the parameters nor the heuristic were optimized based on the training or validation set. Nevertheless, CatLinux outperformed all of the sample controllers.

G. Other controllers

1) 7^{th} - YOLOBOT: This agent is a two-player version of its single-player counterpart submitted to the Single Player planning track of GVGAI. It first uses a system to select an appropriate technique for the game played, making a difference between deterministic (Breadth-first Search) and stochastic (Monte Carlo Tree Search) games. It also employs a target selection system, using MCTS to move towards its current target. Its opponent model is random.

2) 8^{th} - NextLevel: This agent implements an enhanced version of MCTS with a limited rollout depth (4) and attempts to learn information about the game using the event history provided. It assumes the opponent will perform a random action that will not lose them the game, therefore utilizes a similar opponent model as implemented in the sample One Step Look Ahead controller.

3) 13^{th} - webpigeon: This agent uses OLMCTS with a simple agent prediction system which assumes the opponent will always help, compared to the random opponent model from the sample agent, therefore aimed at cooperative games. This was the main reason this controller finished last in the rankings, as most games in the framework are competitive.

VI. Results

The results for the test sets of both legs in 2016, WCCI and CIG, as well as the final Championship rankings, are presented in Tables II, III and IV. The complete rankings and details of performance on individual games, as well as timing disqualifications, can be accessed on the competition website. The average Glicko ratings on all games are reported only as an indication, they do not reflect the generality aspect presented by the Formula-1 points. There were 13 valid unique entries, 5 of which were sample controllers. A 14th entry was submitted, but it was disqualified due to crashing in all games.

ToVo2 was the winner of the first leg of the competition (Table III), obtaining a larger number of points than adrienctx, who came second. However, although it ranked at the top of the table in most games, it was second to last in the cooperative game "The Bridge". The other cooperative game in the set, "Fire Truck", was dominated by the random controller, as the concept of the game was not grasped by any controller, which meant that performing random moves lead to better results. The competitor coming in last on this set only managed to gather 9 points in total over 3 different games.

The second leg, presented at CIG (Table IV), was won by Number27, and the gap between participants (the first three in particular) was closer than in the previous test set. adrienctx came in second again, after achieving top scores in most games but ending up bottom of the table in 3 games: "Mimic", "Reflection" and "Upgrade-X", in which more simple and general algorithms took the lead. Webpigeon ranked last again, with only 4 points collected in one game, "Mimic", where it achieved an average win rate of 48%.

It is interesting to note that the winners of the individual legs did not come first nor second in the final Championship ranking (Table II), where they were overtaken by adrienctx and MaastCTS2 (who came second and third, respectively, in both previous sets). This suggests that the competition system does indeed award more general agents, who may not necessarily be the best at a particular task or subset of tasks, but achieve a high level of performance across a range of problems.

An analysis into the game types shows that most agents struggle considerably more in cooperative games, the average scores and win rates being very close to 0 (with the exception of the games "Team Escape" and "Wheel Me"). This is thought to be due to the weak opponent models which do not consider the other player as an entity interacting with the environment in similar ways and working towards the same goals.

In addition, there are several other games in which average performance, as indicated by Glicko ratings, is fairly low, such as "Minions" from the WCCI test set, "Football" from the

Controller	Main Techniques	Other Enhancements	Offline Learning	Online Learning	Opponent Model
adrienctx	Monte Carlo Tree Search	Value map	✔	✔	Random
MaasCTS2	Monte Carlo Tree Search, Breadth-First Search	Progressive history, N-Gram selection	✗	✔	ϵ-greedy
ToVo2	Monte Carlo Tree Search, Sarsa-UCT(λ)	Weighted-random playouts, Dynamic playout length	✗	✔	Random
Number27	Genetic Algorithm, MixMax	Value map, Pheromone diffusion	✗	✔	Random
CatLinux	Rolling Horizon Evolutionary Algorithm	Shift buffer	✗	✔	Random
SampleOLMCTS †	Monte Carlo Tree Search	✗	✗	✔	Random
YOLOBOT	Monte Carlo Tree Search, Breadth-First Search	Game type identification, Target selection	✗	✔	Random
NextLevel	Monte Carlo Tree Search	Value Map	✗	✔	Simple predictor loss prevention
SampleMCTS †	Closed Loop Monte Carlo Tree Search	✗	✗	✔	Random
SampleGA †	Genetic Algorithm	✗	✗	✔	Random
Random †	Random Action Selection	✗	✗	✗	None
OneStep †	Best Immediate Action	✗	✗	✗	Simple predictor, loss prevention
webpigeon	Monte Carlo Tree Search	✗	✗	✔	Assumes cooperative

TABLE I
CONTROLLER TECHNIQUES SUMMARIZED. †DENOTES A SAMPLE CONTROLLER.

WCCI validation set, and "Cops N Robbers" and "Sokoban" from the training set. Except for "Cops N Robbers", which is a special case, these are games with a large search space and little to no intermediate rewards offered after long sequences of actions. Therefore, the agents are unable to look far enough into the future to figure out optimal solutions. Puzzle games have traditionally proven to be more of a challenge to general agents, portrayed in these results by both versions of the classic game of "Sokoban" (competitive and cooperative) reporting only 1 win during the entire competition.

In the case of "Cops N Robbers", although the average Glicko score is fairly low (260), the win rates range from 7% to 79%. This is explained by the Glicko volatility, which has the lowest values in this game, indicating inconsistency in performance, most likely caused by the clear asymmetry of the game and a slight bias towards the Cop entity. Although this is accounted for by mirroring player order in all games, the average win rate remains at only 43%.

One notable aspect is the fact that the average Glicko ratings on the test sets generally do go down with the ranks computed using the Formula-1 system. This validates the usage of the system for highlighting generality, as the average scores may be biased towards controllers that perform very well in some games but very poorly in others, instead of controllers able to maintain a good skill level across multiple games.

The games which saw the best performances were "Bee Keeper" and "Minesweeper" from the WCCI validation set, and "Egg Hunt" and "War Zone" from the CIG test set. All of these games are competitive and highly interactive, benefiting from multiple rewards and short term feedback for actions. Except for "Egg Hunt", they are also symmetrical, which is an advantage for agents with weak opponent models (as is the case for all competition entries). Although the win rate is highest in these games, there is still a difference of at least 800 Glicko points and 60% win rate between the first and last ranked players. This pinpoints the large skill depth and the numerous types of different problems the agents face, as well as the difficulty in achieving a generally good performance. Various strategies emerge in some games, such as "Up High", in which players might either attempt to increase their own scores or hinder the opponent instead to gain the lead.

Rank	Username	WCCI 2016	CIG 2016	Total
1	**adrienctx**	141	130	**271**
2	**MaasCTS2**	124	125	**249**
3	ToVo2	**171**	75	246
4	Number27	108	**135**	243
5	CatLinux	106	79	185
6	SampleOLMCTS †	94	85	179
7	YOLOBOT	66	106	172
8	NextLevel	46	89	135
9	SampleMCTS †	61	63	124
10	SampleGA †	36	43	79
11	Random †	27	35	62
12	OneStep †	21	41	62
13	webpigeon	9	4	13

TABLE II
FINAL CHAMPIONSHIP RESULTS. †DENOTES A SAMPLE CONTROLLER. THE WINNERS ARE HIGHLIGHTED IN GRAY (OVERALL AND INDIVIDUAL LEGS).

Table I summarizes the competition entries. The most preferred techniques were Monte Carlo Tree Search (MCTS) and Evolutionary Algorithms (EA). MCTS dominated most of the competition and its learning through back-propagation of Monte Carlo simulations led to these algorithms understanding cooperative games better, as well as allowing them to adapt to asymmetric games, where they would have to play different roles in the same environment. However, EAs proved to be better puzzle solvers, in which the sequence of moves matters the most, as well as excelling in games with longer lookaheads.

VII. CONCLUSIONS

This paper gives a detailed account of a new track of the General Video Game AI Competition that focuses on two-player games and introduces new interesting and more

Rank	Username	G-1	G-2	G-3	G-4	G-5	G-6	G-7	G-8	G-9	G-10	Total Points	Avg. Glicko-2 Rating
1	ToVo2	**25**	15	12	18	15	18	18	0	**25**	**25**	171	1077.73
2	adrienctx	2	10	10	12	**25**	8	**25**	**25**	6	18	141	979.99
3	MaastCTS2	4	18	18	0	10	**25**	15	15	18	1	124	1049.97
4	Number27	8	**25**	**25**	6	0	2	12	18	12	0	108	1006.99
5	CatLinux	15	8	15	0	8	15	10	12	15	8	106	959.85
6	SampleOLMCTS †	18	2	8	2	18	12	6	8	8	12	94	950.83
7	YOLOBOT	0	12	0	4	4	10	1	10	10	15	66	826.73
8	SampleMCTS †	12	6	6	1	6	6	8	2	4	10	61	845.96
9	NextLevel	10	1	2	8	12	4	0	6	1	2	46	825.56
10	SampleGA †	0	4	4	15	2	0	4	1	2	4	36	822.00
11	Random †	1	0	0	**25**	0	1	0	0	0	0	27	622.76
12	OneStep †	6	0	1	10	0	0	0	4	0	0	21	714.36
13	webpigeon	0	0	0	0	1	0	2	0	0	6	9	447.48

TABLE III

FINAL RESULTS OF THE WCCI LEG. †DENOTES A SAMPLE CONTROLLER. THE TOP SCORES ARE HIGHLIGHTED, THE DARKER THE COLOUR, THE BETTER THE SCORE. G-1: ACCELERATOR, G-2: BREEDING DRAGONS, G-3: DROWNING, G-4: FIRE TRUCK, G-5: GHOSTBUSTERS, G-6: MINIONS, G-7: OOPS! BROKE IT, G-8: THE BRIDGE, G-9: UP HIGH, G-10: WATCH OUT!

Rank	Username	G-1	G-2	G-3	G-4	G-5	G-6	G-7	G-8	G-9	G-10	Total Points	Avg. Glicko-2 Rating
1	Number27	18	18	10	**25**	15	12	2	6	**25**	4	135	961.74
2	adrienctx	**25**	15	18	2	2	18	15	2	18	15	130	946.11
3	MaastCTS2	12	**25**	15	0	18	**25**	0	10	12	8	125	959.68
4	YOLOBOT	15	10	**25**	0	12	0	4	0	15	**25**	106	925.32
5	NextLevel	2	1	12	1	**25**	6	10	12	8	12	89	890.42
6	SampleOLMCTS †	4	6	4	10	8	15	6	18	4	10	85	931.53
7	CatLinux	9	12	6	18	10	10	0	4	10	1	79	875.38
8	ToVo2	10	8	8	8	4	4	1	8	6	18	75	903.51
9	SampleMCTS †	6	4	2	6	6	8	8	15	2	6	63	918.51
10	SampleGA †	1	2	1	0	0	2	12	**25**	0	0	43	897.30
11	OneStep †	0	0	0	12	1	1	**25**	1	1	0	41	716.26
12	Random †	0	0	0	15	0	0	18	0	0	2	35	617.01
13	webpigeon	0	0	0	4	0	0	0	0	0	0	4	259.26

TABLE IV

FINAL RESULTS OF THE CIG LEG. †DENOTES A SAMPLE CONTROLLER. THE TOP SCORES ARE HIGHLIGHTED, THE DARKER THE COLOUR, THE BETTER THE SCORE. G-1: EGG HUNT, G-2: FATTY, G-3: I SAW SANTA, G-4: MIMIC, G-5: REFLECTION, G-6: TRAIN RIDE, G-7: TREASURE MAP, G-8: UPGRADE-X, G-9: WAR ZONE, G-10: WHEEL ME

complex problems, encouraging closer player interaction. This widens the search space for a general AI agent, as it not only has to be able to handle various types of problems, but also cope with another intelligent entity in the environment.

Games have always provided an excellent test bed for AI research. Traditionally much of this effort was focused on classic games, but recent years have seen heightened levels of focus given to video games, which require rapid reactions and arguably more human-like intelligence. The GVGAI two-player track offers a greater and more open-ended challenge, with other players to compete or collaborate with.

The final results obtained at the end of the competition show that, while agents are able to solve some of these new tasks, there is still plenty of room for improvement. Most of the entries do not seek to model the other player's behavior beyond random moves (with few exceptions which avoid self-destructive actions or ϵ-greedy) and there are no specific attempts to try to identify if the opponent is competing or cooperating. How to predict, react and adapt to the opponent's actions in a general video game setting still remains an unresolved challenge. Recent published work [39] analyzes simple opponent models that are able to outperform random, such as building a probabilistic model of the opponent's actions.

The best algorithm submitted won second place in both individual legs before winning the 2016 Championship, implementing a version of Open Loop Expectimax Tree Search. Statistical Tree Search techniques have shown to be a dominating technique in this framework and competition. However, the fourth and fifth ranked players in the final Championship rankings suggest that there is promise for Evolutionary Algorithms to reach MCTS performance, as Number27's Genetic Algorithm implementation won the CIG leg of the competition.

Future work regarding the competition will take two different multi-agent paths: games in which there are more than two players, possibly grouped in teams, and games where one player controls more than one avatar. These are vastly different problems, one looking at social intelligence and agent interaction, while the other has resource management and multi-objective optimisation tasks underlying the gameplay. However, both are very interesting challenges to be explored in the General Video Game Playing field.

Furthermore, the work carried out in GVGAI can be applied to a wider range of problems, as the games consist of practical and varied challenges such as navigation, puzzle-solving, quick decision making, adversarial tests or collaboration. There are plans of expanding to problems of broader interest, such as zero-sum games or iterated prisoner's dilemma, which should be solved with the same general algorithms.

Finally, although the emphasis of the current paper is to find the most intelligent GVGAI agents, we see enormous potential for using a diverse set of GVGAI agents to assist with playtesting novel games and novel game content. Since the agents

are so general, they can be used to directly measure important aspects of a game, such as skill depth, and less directly to estimate aspects of expected human player experience.

ACKNOWLEDGMENTS

The authors would like to thank all participants of the competition for their work and submitted controllers. This work was partially funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1

REFERENCES

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, 01 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, 2016, pp. 229–243.

[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, pp. 4335–4337.

[5] 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.

[6] S. Samothrakis, D. Perez-Liebana, P. Rohlfshagen, and S. M. Lucas, "Predicting Dominance Rankings for Score-based Games," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 1, 2014, pp. 1–12.

[7] M. E. Glickman, "Parameter Estimation in Large Dynamic Paired Comparison Experiments," *Applied Statistics*, vol. 48, pp. 377–394, 1999.

[8] A. E. Elo, *The Rating of Chess Players, Past and Present*. New York: Arco Pub., 1978.

[9] M. E. Glickman, "Example of the Glicko-2 system," *Boston University*, 2012.

[10] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," in *AI Magazine*, vol. 26, no. 2, 2005, p. 62.

[11] R. Prada, P. Lopes, J. Catarino, J. Quitrio, and F. S. Melo, "The Geometry Friends Game AI Competition," in *IEEE Conference on Computational Intelligence and Games*, 2015, pp. 431–438.

[12] P. Rohlfshagen and S. M. Lucas, "Ms Pac-Man versus ghost team CEC 2011 competition," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, 2011, pp. 70–77.

[13] P. R. Williams, D. P. Liebana, and S. M. Lucas, "Ms. Pac-Man Versus Ghost Team CIG 2016 competition," in *IEEE Conference on Computational Intelligence and Games, CIG*, 2016, pp. 1–8.

[14] S. Ontan, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft," 2013.

[15] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting Game Artificial Intelligence Competition Platform," in *Proceedings of the 2nd IEEE Global Conference on Consumer Electronics*, 2013, pp. 320–323.

[16] S. Lee and J. Togelius, "Showdown AI Competition," in *IEEE Conference on Computational Intelligence and Games, CIG*, 2017.

[17] J. Togelius, "How to Run a Successful Game-based AI Competition," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 1, 2013, pp. 95–100.

[18] T. Schaul, "A Video Game Description Language for Model-based or Interactive Learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2013, pp. 193–200.

[19] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General Game Playing: Game Description Language Specification," in *Technical Report LG-2006-01, Stanford University, Stanford, CA*, 2006.

[20] C. Browne and F. Maire, "Evolutionary Game Design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, March 2010.

[21] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lonneker, L. Cardamone, D. Perez, Y. Sáez *et al.*, "The 2009 Simulated Car Racing Championship," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 131–147, 2010.

[22] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, "Solving the Physical Traveling Salesman Problem: Tree Search and Macro Actions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 31–45, 2014.

[23] D. Perez-Liebana, S. Samothrakis, S. M. Lucas, and P. Rolfshagen, "Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2013, pp. 351–358.

[24] 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, mar 2012.

[25] D. Perez-Liebana, J. Dieskau, M. Hnermund, S. Mostaghim, and S. M. Lucas, "Open Loop Search for General Video Game Playing," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2015, pp. 337–344.

[26] A. Weinstein and M. Littman, "Bandit-Based Planning and Learning in Continuous-Action Markov Decision Processes," in *International Conference on Automated Planning and Scheduling*, 2012, pp. 335–338.

[27] W. A. Gardner, "Learning Characteristics of Stochastic-gradient-descent Algorithms: A General Study, Analysis, and Critique," *Signal Processing*, vol. 6, no. 2, pp. 113–133, 1984.

[28] T. Geffner and H. Geffner, "Width-Based Planning for General Video-Game Playing," in *Proceedings of the Eleventh Artificial Intelligence and Interactive Digital Entertainment International Conference*, A. Jhala and N. Sturtevant, Eds. AAAI Press, 2015, pp. 23–29.

[29] J. A. M. Nijssen and M. H. M. Winands, "Enhancements for Multi-Player Monte-Carlo Tree Search," in *Computers and Games (CG 2010)*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6515, pp. 238–249.

[30] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "N-Grams and the Last-Good-Reply Policy Applied in General Game Playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 73–83, 2012.

[31] D. J. N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands, "Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*. IEEE, 2016, pp. 436–443.

[32] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[33] T. Vodopivec, S. Samothrakis, and Šter Branko, "From Monte Carlo Tree Search to Reinforcement Learning and Back," *manuscript submitted for publication*, 2016.

[34] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Proceedings of the Seventeenth European Conference on Machine Learning*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Berlin/Heidelberg, Germany: Springer, 2006, pp. 282–293.

[35] G. A. Rummery and M. Niranjan, "On-line Q-learning Using Connectionist Systems," Cambridge University Engineering Department, Tech. Rep. 166, September 1994.

[36] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, May-June 2002.

[37] F. Frydenberg, K. Andersen, S. Risi, and J. Togelius, "Investigating MCTS Modifications in General Video Game Playing," in *Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games*, 2015, pp. 107–113.

[38] J. Liu, D. Pérez-Liébana, and S. M. Lucas, "Rolling Horizon Coevolutionary Planning for Two-Player Video Games," in *Proceedings of the IEEE Computer Science and Electronic Engineering Conference (CEEC)*, 2016.

[39] J. M. Gonzalez-Castro and D. Perez-Liebana, "Opponent Models Comparison for 2 Players in GVGAI Competitions," in *Computer Science and Electronic Engineering Conference (CEEC), 2017 9th*. IEEE, 2017, p. to appear.