General Video Game for 2 Players: Framework and Competition

Raluca D. Gaina
University of Essex
Colchester CO4 3SQ, UK
Email: rdgain@essex.ac.uk

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

Simon M. Lucas
University of Essex
Colchester CO4 3SQ, UK
Email: sml@essex.ac.uk

***Abstract*—This paper presents a new track of the General Video Game AI competition for generic Artificial Intelligence agents, which features both competitive and cooperative real time stochastic two player games. The aim of the competition is to directly test agents against each other in more complex and dynamic environments, where there is an extra uncertainty in a game, consisting of the behaviour of the other player. The framework, server functionality and general competition setup are analysed and the results of the experiments with several sample controllers are presented. The results indicate that currently Open Loop Monte Carlo Tree Search is the overall leading algorithm on this set of games.**

I. Introduction

General Video Game Playing (GVGP) is a sub-domain of Game Artificial Intelligence which aims to create generic enough agents, capable of playing any given game, without pre-computed game-specific heuristics, in possibly unknown environments. It is meant to put complex algorithms to the test, challenging their adaptibility to new situations. There are general approaches such as deep reinforcement learning combined with Monte Carlo Tree Search which can be engineered to do brilliantly in some specific games (e.g. AlphaGo in Go [1]). However, there is still a significant challenge in designing general game plaing agents which can play well on new games without requiring any extensive training time, and without any human-designed game features, which is where the need for GVGP agents arises.

In order to test these agents, there is the need to define a set of constraints and specific challenges in order to make comparisons between algorithms and identify valuable advances in the field. The General Video Game AI competition (GVGAI) [2] provides just that, acting as an excellent benchmark that offers multiple sets of real time stochastic games.

This paper describes a new track of this competition, the Two Player GVGAI Competition (GVGAI2P), which is meant to test agents directly against each other in more complex problems, featuring both competitive and cooperative games and therefore forcing agents to adopt various play styles and techniques.

The rest of this paper is structured as follows: Section II gives an overview of previous work carried out in this area, Section III presents the framework underlying GVGAI2P, Section IV describes the competition set up and Section V introduces the sample controllers available with the framework and the results obtained from initial experiments. The paper concludes with a summary of this research, an account of its importance and future work.

II. Relevant Research

An internationally successful competition which had multi-player games as its focus (e.g. Planet Wars in 2010, Ants in 2011) was the Google AI Challenge[1] which was organised between 2009 and 2011 by the Computer Science Club of the University of Waterloo. The framework used in this competition and the way the agents were able to interact with the environment were very similar to the GVGAI competition. However, agents would still use game specific heuristics to aid their decision making process. The users were supplied with sample controllers and tools to test their controllers locally or record replays. The way rankings were calculated for the Google AI Challenge was using the Elo scoring system, which provided rather slow feedback to users due to the large number of participants.

Samothrakis et al. [3] considered this problem, as well as that of selecting entries in a multi-player competition to play against each other, together with ranking of entries according to the skills demonstrated in matches played. To this extent, the paper looks at the Ms Pac-Man vs Ghosts competition [4], in which AI controllers for either Pac-Man or the group of ghosts (or both) are submitted. The system pits against each other opponents of the same type, taking into account the wins and losses of each agent and analyses two different ranking algorithms: Glicko and Bayes Elo. Based on the number of games needed to determine ranks, the paper suggests that Glicko, even as the less popular method, outperforms Elo scoring. This is the reason behind using Glicko-2 ratings for the GVGAI2P competition, which is an improved system based on the original Glicko.

The General Game Playing (GGP) competition [5] is a worthy mention for two player GVGP competitions, which runs as part of the AAAI conference. It is made up of various stages, including qualifiers, at the end of which the four best entries are chosen to move on, followed by semifinals and finals, where an overall winner is decided. This winning agent plays against a human expert for a final test of its intelligence.

[1] http://aichallenge.org/

Another unique aspect of this competition is that it aims to find a controller generic enough to be able to play both single and multi-player games. However, the games these agents are tested on are turn-based strategic deterministic environments (mostly board and puzzle games). This is one area where GVGAI raises the challenge to real time stochastic games, giving the agents a very limited time budget for decision making.

Another notable contribution is the Geometry Friends Game AI (GFGAI) [6] competition, running as part of the GECCO conference. It has three separate tracks (single-player Circle, single-player Rectangle and two-player Cooperation), but the most relevant for the research described in this paper is the last one mentioned. It is focused on a specific real-time game, in which two characters have different abilities and aim to solve puzzles, while being affected by simulated physics, including attrition, gravity and dynamic elements, an additional challenge for the agents. GFGAI provides the users with only one sample controller which employs a Reinforcement Learning algorithm and 5 levels of the game to locally test their controllers. Competition entries are tested on 5 different levels and ranked according to the time they took to complete a level and the number of diamonds collected. This is similar to the GVGAI usage of different training and test sets and the elements taken into account when calculating an agents score.

There are certain aspects which improve a game AI competition's chances at being successful and Togelius offers an overview on how to achieve this result [7]. He highlights the benefits of this type of competitions, as not only an efficient way to benchmark algorithms, but also useful in attracting the interest of new researchers in the field. The factors discussed in this paper that might influence the results of competitions include transparency and reliability, persistence and availability, as well as making it easy for entrants to get started, or running a discussion group to encourage peer technical support and active involvement. Togelius suggests that an important aspect is making sure that the competition evolves in the future in order not to die out.

III. Framework

A. Video Game Description Language

The GVGAI framework uses a Java port of the original version of the Video Game Description Language (VGDL) [8] written in python for general purpose game definition. VGDL is a logic programming language inspired by the Game Description Language (GDL) [9], which was used for GGP competitions. However, where GDL relies on logical rules, VGDL instead describes games by defining entities and the interactions between them. Although this game description is done through text files, the dynamics associated with an entity (e.g. movement, abilities, behaviour) are programmed separately within the framework.

The latest version of VGDL expanded for the purpose of this research supports the definition of both single and multi-player two dimensional games, with possibly multiple levels associated with one game. The various sprites can be defined and parameterized, as well as placed in the game world through the level definition. The `InteractionSet` (see Algorithm 1) section specifies rules for collision events and finally, the `TerminationSet` identifies end of game and winning conditions.

Algorithm 1 VGDL Definition 2 Player Sokoban

```
1:  BasicGame key_handler=Pulse no_players=2
2:    SpriteSet
3:      hole >Immovable color=DARKBLUE img=hole
4:      ground >Immovable img=water hidden=True
5:      avatar >MovingAvatar
6:        avatar1 >img=avatar
7:        avatar2 >img=alien
8:      bbox >Passive
9:        box >img=box
10:       boxin >img=city
11:   LevelMapping
12:     0 >hole
13:     1 >box ground
14:     . >ground
15:     A >avatar1 ground
16:     B >avatar2 ground
17:   InteractionSet
18:     avatar wall avatar >stepBack
19:     bbox avatar >bounceForward
20:     bbox wall bbox >undoAll
21:     box hole >transformTo stype=boxin
22:                 scoreChange=1,1
23:     boxin ground >transformTo stype=box
24:                   scoreChange=-1,-1
25:   TerminationSet
26:     SpriteCounter stype=box limit=0 win=True,True
```

Only game description files are different for the 2 player version, with the need to define two avatar sprites, which may be of different types, as well as their interaction with other game objects and with each other. The text file must specify the number of players in the first line, which is 1 by default, to keep backcompatibility with the single player games. Furthermore, the effects of each interaction (i.e. score change, see Algorithm 1, line 21) and termination conditions (winning or losing the game) should be specified for each player. A game can end with one player winning and the other losing, both players winning or both players losing.

B. State Observation and Forward Model

The game and level information written in VGDL can be communicated to both AI and human players in visual or contextual forms, encapsulated in game objects.

AI agents are provided with knowledge about the current game state through a StateObservationMulti object, which contains the current game step counter, a list of observations (category, type and position of other objects in the game, such as portals, resources, NPCs etc.), an observation grid and a history of the events that have taken place in the game up until

the current game step. In addition, information is given on all avatars in the game, including their game score and victory state (win, loss or still ongoing) and their list of available actions, as well as their avatar's position, speed, orientation, health points, resources and avatar type. All queries about a specific avatar must pass the player ID in the method call.

A forward model (accessed via the advance method of the StateObservation object) allows for possible future game states to be simulated. An array of actions for all players must be supplied to the *advance* method. The actual definition of the game is not available to the controllers, therefore creating a challenge in finding out aspects such as what the goal of the game is, how it can be won, which NPCs are friendly and which are enemies, etc.

C. AI Agents and Game Cycle

A game cycle can be divided into two main parts: the system providing the current game state to the players and the players returning one of the actions available (defined in the class ontology.Types): No action (*ACTION_NIL*), move left (*ACTION_LEFT*), right (*ACTION_RIGHT*), up (*ACTION_UP*), down (*ACTION_DOWN*) or a special action, that produces different effects in distinct games (*ACTION_USE*).

Any controller designed for this competition has to extend the AbstractMultiPlayer class and implement two methods: a *constructor*, which is called once at the start of the game, allowing the agents time for any initialisation necessary, with an execution time limit of 1 second, and an *act* method, which is to return a discrete action to be performed by the agent, in under 40 milliseconds, called at every game step.

The controller gets disqualified if it exceeds any of the time limits (with an exception to the act method, which, if it completes in between 40ms and 50ms, then the agent is not disqualified, but the default action *ACTION_NIL* is returned instead). An instance of the StateObservationMulti object is passed as an attribute to both of these methods, together with a timer. Moreover, the player ID is passed to the agent in the constructor.

D. Software

The structure of the GVGAI framework is described next:

- Package *controllers*. Contains all of the sample controllers; more information about them can be found in Section V.
- Package *core*. Contains all the game code.
  - Package *competition*. Code necessary for running the competition, including competition parameters.
  - Package *content*. VGDL content classes.
  - Package *game*. Core classes, including game execution, forward model and state observation.
  - Package *player*. Contains abstract classes controllers must extend (for both single and multi player agents).
  - Package *termination*. Contains classes for the VGDL termination conditions.
  - VGDL classes and ArcadeMachine.java (used for different framework execution modes).
- Package *ontology*. Contains all other Java code for VGDL definitions of avatars, effects, sprites, physics and types.
- Package *tools*. Various classes and pathfinding code.
- Several classes for the various framework execution modes, as detailed below.

Similar to the single-player version, a game may be played in the GVGAI 2 Player framework in different modes, as available in the class TestMultiPlayer.java. All of the options offer the possibility of recording the moves played in a text file for a later replay. The order of the controllers passed in these methods indicate which one is the first and second player. The different execution modes include playing as two humans, a human against an AI, two AI players against each other in one game, two AI players run in multiple games and levels or a round robin tournament between two AI players.

IV. Competition

All that is needed to get started with an entry for the competition is a Java class called "Agent", in a package using the username registered on the website (submitted as a ZIP file and including any other classes the controller might use, not already included in the framework). No files can be written from the controller's directory and any files to be read should use a relative path and be included in the submission.

All submissions are saved in a database. Execution, which can be monitored by the users through their profile page, is then split between two different servers: the GVGAI server and a separate GVGAI 2 Player server.

The GVGAI server periodically checks if there are any new controllers in the database, unzips and compiles them; users are notified of any errors that might occur during this process. If more than 2 controllers are available (successfully compiled), the Glicko-2 system is used to select two of them to play a round of games: in the training phase, the agents will play 1 randomly selected level from each one of the 10 games in either the training set (public games - see Table I) or the validation set (games hidden from the contestants), once as the first player and again as the second player; for the final rankings, the agents will play all levels of all games in the test set (formed by a different set of 10 secret games), 5 times each, with positions swapped for each match as well.

The GVGAI 2 Player server regularly checks if new runs are queued and executes them, if it is capable of handling more processes at that moment. It records the results of the run and communicates the information back to the GVGAI server, which calculates and updates the database with the new Glicko-2 ratings, rating deviation and volatility of the controllers that were involved in that particular run.

Glicko-2 is a rating system created by Mark E. Glickman and based on the original Glicko system, in which all players have a 3-tuple associated, which consists of a rating (r), a rating deviation (RD) and a volatility (σ). The default values for an unranked player are $(1500, 350, 0.06)$. The volatility represents how likely is a player's score to fluctuate, therefore being lower if the player's performance is kept at a constant rate. The RD represents the confidence in a player's rating,

Game	Description	Type	Action Set
Akka Arrh	The players have to defend a locked spaceship from aliens attacking it (2 points earned for killing an alien). In order to win the game, they have to retrieve the key that unlocks the ship (earning 10 bonus points) and get inside.	Cooperative	A0, A0
Asteroids	The players control spaceships in an area full of asteroids. Shooting an asteroid splits the target into smaller asteroids, which are destroyed after 3 iterations (earning 3 points). Shooting the other player's ship kills the opponent and grants 10 points. Protective blocks can also be shot for 1 point.	Competitive	A0, A0
Capture Flag	The level is divided into two areas, one for each player, containing the opponent's flag. The goal of each player is to retrieve their flag from the enemy territory and bring it back into their own, causing all flags and avatars to respawn at their starting positions and earning 3 points. The players can also catch the opponent when they are in their own territory to gain 2 points if the enemy has their flag or 1 point otherwise, in addition forcing positional reset.	Competitive	A1, A1
Cops N Robbers	One player takes the role of the robber, who has to collect all the diamonds in the level (earning 1 point per diamond collected). The other player becomes the cop, who has to catch the robber before all the diamonds are gone (gaining 4 points if successful). In case of timeout, both players lose the game.	Competitive	A1, A0
Gotcha	One player has to chase the other, the game resulting in a win for the first player if they catch the opponent, or the other way around if the second player manages to avoid capture until time runs out. There are NPCs blocking the players' movement and safe areas where the pursued agent can hide for a limited amount of time, after which the safe spots are destroyed.	Competitive	A1, A1
Klax	The players compete to get the highest score by collecting differently coloured blocks that fall from the sky. The first time a player collects a block, they are assigned the block's colour and gain 2 points from thereon by catching blocks of the same colour. 1 point is awarded for catching blocks of the opponent's colour. All of one player's points are lost if they catch a block of a colour unclaimed by either avatar.	Competitive	A2, A2
Samaritan	One player has the goal of crossing a portal to another world, while the other has to prevent that from happening until timeout, by either blocking the path or using their special action on the opponent, which results in the first player's position to be reset. If the second player falls into a portal, both players lose.	Competitive	A1, A0
Sokoban	The players push boxes into predetermined special locations. 1 point is awarded for both players for each box successfully placed in the right place, or removed if the boxes are moved out of a correct location. Both players lose on timeout and both players win if all of the boxes have been correctly placed.	Cooperative	A1, A1
Steeplechase	A racing game in which the players compete to reach the finish line. There are obstacles in the level, which can be destroyed with a melee weapon, and one gem hidden in one of the boxes which grants 1000 points to the player that collects it.	Competitive	A0, A0
Tron	The players compete in an arena, constantly moving (ACTION_NIL still makes the avatar move forward), only able to turn left or right and leaving solid trails behind them. If they hit either a wall or a trail left behind by any of the avatars, the player loses and the opponent wins. Both players lose on timeout.	Competitive	A3, A3

TABLE I
TRAINING SET GAMES. ACTIONS LEGEND: A0: ALL MOVES; A1: ONLY DIRECTIONAL; A2: LEFT AND RIGHT; A3: ALL (CONSTANTLY MOVING)

higher if the player has not played in a while (a greater change in rating could be expected) and it can be used to report a player's rating as an interval for more accurate analysis. These values are calculated at the end of a rating period, which consists of several games played between different users.

This system has been adopted for the GVGAI2P competition with several modifications. As the agents are tested on several different games, each of them are appointed a game specific 3-tuple, which would be averaged for an overall approximation of performance. With the runs being relatively self contained, all of the values are updated for each pair of controllers after playing all matches on one game. As the Glicko-2 algorithm would result in the rating deviations slowly decreasing, the rating period was simulated by increasing the RD values of all the controllers which were not part of those matches.

In addition, after observing the sample controllers test the system, it became clear that the default values were not appropriate for all of the games, as the average performance of the controllers was better in some games and worse in others. This situation raised the need of using game specific default ratings, which are periodically recalculated as the average ratings in each particular game. Moreover, a minimum rating value was introduced as 0, in order to prevent scores from becoming very small negative values in games where controllers hardly win (e.g. Sokoban).

The Formula-1 system is used to award points in each game, depending on the ranking according to the Glicko rating (the number of points awarded are, in order, from first to last: 25, 18, 15, 12, 10, 8, 6, 4, 2, 1 and 0 for all the other entries). The points in all games are summed up in order to determine a controller's overall performance and rank.

V. SAMPLE CONTROLLERS

This section describes the sample controllers currently available within the framework.

A. DoNothing

This controller is the most basic one. It always returns *ACTION_NIL*, therefore it does not perform any actions during the game.

B. SampleRandom

This is another very basic controller. It does not require any information processing, instead selecting a random action from those available at every game step.

C. SampleOneStepLookAhead (OneStep)

In single player games, the One Step Look-Ahead controller analyses all of the actions available at every game step and picks for execution that which leads to the next best state.

The simple heuristic used for game state evaluation aims to minimise the distance to NPCs and portals, minimise the number of NPCs still in the game and maximise score, as well as giving a great bonus for winning the game, or a great penalty for losing.

In multi player games, the behaviour of the controller is quite similar, using the same heuristic for approximating state values. However, it now needs to pass an array of actions, for all players in the game, to advance the forward model and reach these new possible states which would allow it to evaluate actions. While the agent still iterates through all the actions available to itself, the action picked for the opponent in each case is calculated by another method. This method returns a random action out of those available to the opponent, which they would make assuming the current player performs *ACTION_NIL* without resulting in them losing the game.

After all of the agent's actions are evaluated, the one with the highest reward is chosen for execution.

D. SampleGA

This controller uses a Rolling Horizon Evolutionary Algorithm (RHEA) [10], which takes advantage of the forward model to simulate possible future states of the game and creates plans of a pre-defined number of actions. These plans are considered the individuals in the population evolved by the algorithm. Due to the real-time restriction of the games, RHEA only generates one new individual every iteration, instead of an entire population. When the algorithm reaches the maximum number of iterations or has used up all the time in its budget, it returns the first action of the individual resultant after the evolutionary process.

The fitness of each individual is calculated by the value of the state reached after executing all the actions in one individual. The heuristic used to evaluate a game state aims to maximise the score of the current player. In addition, a large bonus is awarded if the player wins or the opponent loses and a large penalty is applied if the contrary is true. As in multi player games, an array of actions for all players is required to advance the forward model, this sample controller assumes the opponent will always do a random move out of those available.

E. SampleMCTS

Monte Carlo Tree Search (MCTS) [11] is a Tree Search algorithm, made up of four main steps. The first step is *Selection*, during which a non-terminal node is chosen using the tree policy, out of all the nodes which have not yet been fully expanded. The sample controller uses UCB1 (see Equation 1) as the Tree Policy, with a constant C of $\sqrt{2}$, to balance between exploration and exploitation.

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

The second step is *Expansion*, which is focused on adding a new child of this node to the tree. A Monte Carlo (MC) *Simulation* follows, in which actions are picked uniformly at random (Default Policy) from the newly-added child until the end of the game or a pre-defined depth has been reached.

Finally, during *Back-propagation*, the value of the state obtained at the end of the MC simulation is used to update the values of N(s), N(s,a) and Q(s,a) of all visited nodes, up to the root of the search tree. The heuristic used to evaluate a game state maximises the current player's score, adding a large bonus for winning the game, or giving a penalty for losing.

In the multi player version of MCTS, in the array of actions for all players required to advance the forward model, the opponent's move is assumed to be random. It is worth highlighting the simultaneous nature of the games, as opposed to turn-based: each tree level contains states derived from both players executing a move at the same time, instead of alternating between the first and second player's actions.

F. SampleOLMCTS

Open Loop Monte Carlo Tree Search (OLMCTS) [12] is an MCTS variation, which does not store the game state in the nodes of the trees, instead using the forward model to reevaluate future states at each iteration. This method is particularly efficient in GVGAI due to the stochastic nature of the games, which means that storing game states in the nodes of the tree would result in possibly inaccurate results and wrong assumptions to be made based on previously different scenarios. However, this approach does face the problem of having to reach future states using the forward model more often. This controller uses the same heuristic, policies and opponent action selection as the sample MCTS agent.

G. Performance of sample controllers

The Glicko-2 system was used to compute the performance of the sample controllers. They played two matches on a randomly selected level of all 10 games of the training set (with the order of the controllers reversed in the second match). The results obtained were processed using the Formula-1 system to rank the controllers in each game according to their Glicko-2 ratings (as described in Section IV). The points are summed up across all games to obtain the rankings presented in Table II. The rankings reached a stable order after 17 runs (340 games) in less than 1 hour.

The results indicate that OLMCTS is clearly the best out of the sample controllers. A possible line of further development could be focused on improving upon this algorithm, stating with a better than random assumption on the opponents actions. Similarly, there is clear room for improvement in the GA controller as well.

These rankings were verified with a round robin tournament between all sample controllers on a similar set up to the Glicko-2 system (10 training set games, 1 random level, each match played twice). However, assuming more entries will be added to the competition, recomputing the rankings using the Glicko-2 system would be faster than the round-robin alternative, for which $m \times \frac{n+1}{n-1}$ more games would be necessary, where m is the last number of games played and n is the current total number of participants in the competition.

Controller	Akka Arrh	Asteroids	Capture Flag	Cops N Robbers	Gotcha	Klax	Samaritan	Sokoban	Steeple chase	Tron	Total
Sample OLMCTS	**7.45% (25, 0)**	**91.49% (25, 2169)**	**63.85% (25, 830)**	**50.00% (25, 67)**	**77.66% (25, 1903)**	89.36% (18, 1582)	63.82% (15, 910)	0.00% (25, 0)	8.51% (18, 0)	**98.94% (25, 1794)**	**55.10% (226)**
Sample MCTS	5.43% (18, 0)	69.57% (18, 1977)	58.70% (18, 644)	41.30% (18, 0)	64.13% (18, 1710)	**80.43% (25, 1605)**	**59.78% (25, 1000)**	0.00% (25, 0)	7.60% (15, 0)	38.04% (18, 1424)	42.50% (198)
OneStep	**7.45% (25, 0)**	31.37% (15, 1672)	34.31% (15, 620)	17.65% (8, 0)	32.10% (12, 1371)	43.33% (12, 1185)	52.94% (10, 817)	0.00% (25, 0)	**100% (25, 858)**	45.09% (15, 1414)	36.47% (162)
Sample GA	0.00% (12, 0)	40.42% (12, 1636)	56.38% (12, 587)	32.98% (15, 0)	52.12% (15, 1405)	48.93% (15, 1549)	41.48% (18, 1000)	0.00% (25, 0)	6.38% (12, 0)	47.87% (10, 1270)	32.67% (146)
Do Nothing	0.00% (12, 0)	23.96% (10, 1494)	26.67% (10, 472)	16.67% (12, 0)	42.09% (10, 1338)	18.89% (10, 728)	12.22% (8, 293)	0.00% (25, 0)	0.00% (8, 0)	51.11% (12, 1296)	21.55% (117)
Sample Random	0.01% (15, 0)	45.56% (8, 1352)	25.00% (8, 437)	17.71% (10, 0)	28.73% (8, 1287)	10.42% (8, 563)	52.08% (12, 845)	0.00% (25, 0)	2.08% (10, 0)	19.79% (8, 1104)	18.23% (112)

TABLE II

SAMPLE CONTROLLERS RANKED ON THE TRAINING SET USING THE GLICKO-2 SYSTEM. PERCENTAGE OF GAMES WON ARE SHOWN FIRST, FOLLOWED BY F1 POINTS AND GLICKO-2 RATING PER GAME IN BRACKETS.

VI. CONCLUSION

This paper presented the expansion of the General Video Game AI Competition to a Two Player Track, which will challenge general Artificial Intelligence agents in new and interesting ways. As the games in the framework vary in nature, the agents would have to adapt to both competitive and cooperative environments, as well as interacting and coordinating effectively with other agents in order to succeed. Although it may seem like a difficult task, the benefits of agents succeeding in this competition would outweigh this aspect, as they could make use of shared experiences to accelerate and improve learning.

To conclude, the rankings of the sample controllers on the training set suggest OLMCTS to be the best. However, One Step Look Ahead exhibits particularly good behaviour in games such as Steeplechase, where its heuristic focused on portals allows it to outperform tree search techniques, for which the search space is too large to produce good results. Game specific features will be further explored to continue the work presented in this paper, together with the possibility of automatically evolving such features. [13] [14]

Future work could also be focused on the competition itself, not only by increasing the number of games for testing agents, but also by exploring different competition configurations. A first option would be having more than two players taking part in a single game, each controlled by an individual agent. Optimisations for execution times should be considered in this case, but this scenario could lead to controllers learning to adapt to a community, work together to solve larger scale problems, elect leaders, eliminate enemies, form alliances, etc. This could potentially create a significant breakthrough in general group intelligence, while individuals still give more weight to their own different goals and strategies.

Furthermore, in keeping with the larger number of players or characters in a game, one agent could take control of a group of them, instead of only one (e.g. tactic games). This would pose a vastly different challenge, as multi objective optimisation and task decomposition methods might come into play, with the agents having to manage the individuals part of their group and any resources available, as well as focusing on the major goals and still acknowledging other player presences in the game, all in the same multi-game general setting.

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] 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. PP, no. 99, 2015, p. 1.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] C. Browne, D. W. E. Powley, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, 2012, pp. 1–43.

[12] 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.

[13] S. M. Lucas, S. Samothrakis, and D. Perez, "Fast Evolutionary Adaptation for Monte Carlo Tree Search," in *EvoGames*, 2014.

[14] D. Perez, S. Samothrakis, and S. M. Lucas, " Knowledge-based Fast Evolutionary MCTS for General Video Game Playing," in *IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1–8.