General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms

Diego Perez-Liebana, *Member, IEEE*, Jialin Liu, *Member, IEEE*, Ahmed Khalifa, Raluca D. Gaina, *Student Member, IEEE*, Julian Togelius, *Member, IEEE*, and Simon M. Lucas, *Senior Member, IEEE*

***Abstract*—General video game playing aims at designing an agent that is capable of playing multiple video games with no human intervention. In 2014, the General Video Game Artificial Intelligence (GVGAI) competition framework was created and released with the purpose of providing researchers a common open-source and easy-to-use platform for testing their artificial intelligence (AI) methods with potentially infinity of games created using the video game description language (VGDL). The framework has been expanded into several tracks during the last few years to meet the demands of different research directions. The agents are required either to play multiple unknown games with or without access to game simulations, or to design new game levels or rules. This survey paper presents the VGDL, the GVGAI framework, existing tracks, and reviews the wide use of GVGAI framework in research, education, and competitions five years after its birth. A future plan of framework improvements is also described.**

***Index Terms*—Artificial intelligence, computational intelligence, games, General Video Game AI (GVGAI), general video game playing (GVGP), video game description language (VGDL).**

I. Introduction

GAME-BASED benchmarks and competitions have been used for testing artificial intelligence capabilities since the inception of the research field. Since the early 2000s, a number of competitions and benchmarks based on video games have sprung up. So far, most competitions and game benchmarks challenge the agents to play a single game, which leads to overspecialization, or overfitting, of agents to individual games. This is reflected in the outcome of individual competitions—for example, over more than five years, the Simulated Car Racing Competition [1][1] ran, submitted car controllers got better at completing races fast, but incorporated more and more game-specific engineering and arguably less of general artificial intelligence (AI) and machine learning algorithms. Therefore, this trend threatens to negate the usefulness of game-based AI competitions for spurring and testing the development of stronger and more general AI.

The General Video Game Artificial Intelligence (GVGAI) competition [3] was founded on the belief that the best way to stop AI researchers from relying on game-specific engineering in their agents is to make it impossible. Researchers would develop their agents without knowing what games they will be playing, and after submitting their agents to the competition all agents are evaluated using an unseen set of games. Every competition event requires the design of a new set of games, as reusing previous games would make this task impossible.

While the GVGAI competition was initially focused on benchmarking AI algorithms for playing the game, the competition and its associated software has multiple uses. In addition to the competition tracks dedicated to game-playing agents, there are now tracks focused on generating game levels or rules. There is also the potential to use GVGAI for game prototyping, with a rapidly growing body of research using this framework for everything from building mixed-initiative design tools to demonstrating new concepts in game design.

The objective of this paper is to provide an overview of the different efforts from the community on the use of the GVGAI framework (and, by extension, of its competition) for general game artificial intelligence. This overview aims at identifying the main approaches that have been used so far for agent AI and procedural content generation (PCG), in order to compare them and recognize possible lines of future research within this field. This paper starts with a brief overview of the framework

Manuscript received February 26, 2018; revised July 27, 2018 and December 22, 2018; accepted February 17, 2019. Date of publication March 11, 2019; date of current version September 13, 2019. This work was supported in part by the EPSRC CDT in Intelligent Games and Game Intelligence under Grant EP/L015846/1, in part by the Shenzhen Peacock Plan under Grant KQTD2016112514355531, in part by the Science and Technology Innovation Committee Foundation of Shenzhen under Grant ZDSYS201703031748284, and in part by the Program for University Key Laboratory of Guangdong Province under Grant 2017KSYS008. *(Corresponding author: Jialin Liu.)*

D. Perez-Liebana, R. D. Gaina, and S. M. Lucas are with the Department of Electrical Engineering and Computer Engineering (EECS), Queen Mary University of London, London E1 4NS, U.K. (e-mail: diego.perez@qmul.ac.uk; r.d.gaina@qmul.ac.uk; simon.lucas@qmul.ac.uk).

J. Liu is with the Shenzhen Key Laboratory of Computational Intelligence, University Key Laboratory of Evolving Intelligent Systems of Guangdong Province, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China (e-mail: liujl@sustc.edu.cn).

A. Khalifa and J. Togelius are with the Department of Computer Science and Engineering, New York University, New York, NY 11201 USA (e-mail: aak538@nyu.edu; julian@togelius.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TG.2019.2901021

[1]We cite Yannakakis [1] and Russell [2] as standard references for Games and AI (respectively) to reduce the number of non-GVGP references.

and the different competition tracks, for context and completeness, which summarizes work published in other papers by the same authors. The bulk of this paper is centered in the next few sections, which are devoted to discussing the various kinds of AI methods that have been used in the submissions to each track. Special consideration is given to the single-player planning track, as it has existed for longest and received the most submissions up to date. This is followed by a section cataloguing some of the noncompetition research uses of the GVGAI software. The final few sections provide a view on the future use and development of the framework and competition: how it can be used in teaching, open research problems (specifically related to the planning tracks), and the future evolution of the competition and framework itself.

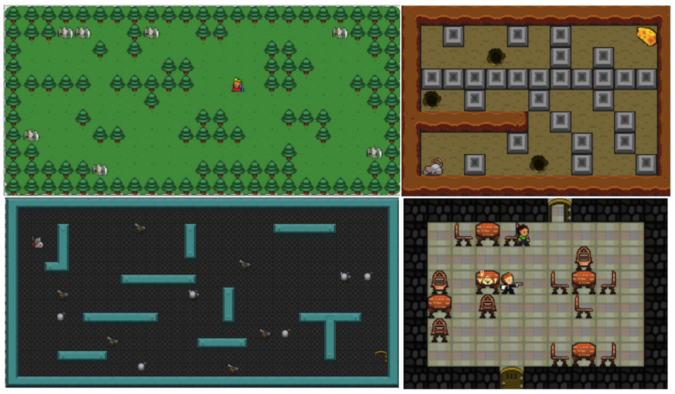
Fig. 1. Examples of VGDL games. From top to bottom, left to right: Butterflies, Escape, Crossfire, and Wait for Breakfast.

II. GVGAI Framework

Ebner *et al.* [4] and Levine *et al.* [5] first described the need and interest for such a framework that could accommodate a competition for researchers to tackle the challenge of general video game playing (GVGP). The authors proposed the idea of the video game description language (VGDL), which was later developed by Schaul [6], [7] in a Python framework for model-based learning and released the first game engine in 2013. Years later, Perez-Liebana *et al.* [3] implemented a version of Schaul's initial framework in Java and organized the first GVGAI competition, in 2014 [8], which employed games developed in VGDL. In the following years, this framework was extended to accommodate two-player games [9], [10], level [11], rule [12] generation, and real-world physics games [13]. These competition tracks accumulate hundreds of submissions. Furthermore, the GVGAI framework and competition have been used as tools for research and education around the globe, including their usage in taught modules, M.Sc. and Ph.D. dissertation projects (see Section XI).

VGDL is a text description language that allows for the definition of 2-D, arcade, grid-based physics and (generally) stochastic games and levels. Originally designed for single-player games, the language now admits two-player challenges. VGDL permits the definition of sprites (objects within the game) and their properties (from speed and behavior to images or animations) in the sprite set. Thus, this set defines the type of sprites that can take part in the game. Their interactions are regulated in the interaction set, which defines the rules that govern the effects of two sprites colliding with each other. This includes the specification of score for the games. The termination set defines how the game ends, which could happen due to the presence or absence of certain sprites or due to timers running out. Levels in which the games can be played are defined also in text files. Each character corresponds to one or more sprites defined in the sprite set, and the correspondence between sprites and characters is established in the mapping set. At the moment of writing, the framework counts on 120 single-player and 60 two-player games. Examples of VGDL games are shown in Fig. 1.

VGDL game and level files are parsed by the GVGAI framework, which defines the ontology of sprite types and interactions that are allowed. The benchmark creates the game that can be played either by a human or a bot. For the latter, the framework provides an API that bots (or *agents*, or *controllers*) can implement to interact with the game—hence, GVGAI bots can play any VGDL game provided. All controllers must inherit from an abstract class within the framework and implement a constructor and three different methods: INIT, called at the beginning of every game; ACT, called at every game tick and must return the next action of the controller; and RESULT, called at the end of the game with the final state.

The agents do not have access to the rules of the game (i.e., the VGDL description) but can receive information about the game state at each tick. This information includes the game status; i.e., winner, time step and score; state of the player (also referred to in this paper as *avatar*), i.e., position, orientation, resources, health points; and history of collisions and positions of the different sprites in the game identified with a unique *type id*. Additionally, sprites are grouped in *categories* attending to their general behavior: nonplaying characters (NPCs), static, movable, portals (which spawn other sprites in the game, or behave as entry or exit point in the levels), and resources (that can be collected by the player). Finally, each game has a different set of actions available (a subset of *left*, *right*, *up*, *down*, *use,* and *nil*), which can also be queried by the agent.

In the *planning* settings of the framework (single-player [8] and two-player [10]), the bots can also use a forward model. This allows the agent to copy the game state and roll it forward, given an action, to reach a potential next game state. In these settings, controllers have 1 s for initialization and 40 ms at each game tick as decision time. If the action to execute in the game is returned between 40 and 50 ms, the game will play the move *nil* as a penalty. If the agent takes more than 50 ms to return an action, the bot will be disqualified. This is done in order to keep the real-time aspect of the game. In the two-player case, games are played by two agents in a simultaneous move fashion. Therefore, the forward model requires the agents to also supply an action for the other player, thus facilitating research in general opponent modeling. Two-player games can also be competitive or cooperative, a fact that is not disclosed to the bots at any time.

The *learning* setting of the competition changes the information that is given to the agents. The main difference with the planning case is that no forward model is provided, in order to foster research by learning to play in an episodic manner [14].

This is the only setting in which agents can be written not only in Java, but also in Python, in order to accommodate for popular machine learning libraries written in this language. Game state information (same as in the planning case) is provided in a Json format, and the game screen can be observed by the agent at every game tick. Since 2018, Torrado *et al.* [15] interfaced the GVGAI framework to the OpenAI Gym environment.

The GVGAI framework can also be used for PCG. In the *level generation* setting [11], the objective is to program a generator that can create playable levels for any game received. In the *rule generation* case [12], the goal is to create rules that allow agents to play in any level received. The framework provides, in both cases, access to the forward model, so agents can be used to test and evaluate the content generated.

When generating levels, the framework provides the generator with all the information needed about the game such as game sprites, interaction set, termination conditions, and level mapping. Levels are generated in the form of 2-D matrix of characters, with each character representing the game sprites at the specific location determined by the matrix. The challenge also allows the generator to replace the level mapping with a new one. When generating rules, the framework provides the game sprites and a certain level. The generated games are represented as two arrays of strings. The first array contains the interaction set, while the second array contains the termination conditions.

As can be seen, the GVGAI framework offers an AI challenge at multiple levels. Each one of the settings (or competition *tracks*) is designed to serve as benchmark for a particular type of problems and approaches. The planning tracks provide a forward model, which favors the use of statistical forward planning and model-based reinforcement learning methods. In particular, this is enhanced in the two-player planning track with the challenge of player modeling and interaction with another agent in the game. The learning track promotes research in model-free reinforcement learning techniques and similar approaches, such as evolution and neuroevolution. Finally, the level and rule generation tracks focus on content creation problems and the algorithms that are traditionally used for this: search-based [evolutionary algorithms (EAs) and forward planning methods], solver (SAT, answer set programming), cellular automata, grammar-based approaches, noise, and fractals.

III. GVGAI Competition

For each one of the settings described in the previous section, one or more competitions have been run. All GVGAI competition tracks follow a similar structure: games are grouped in different sets (ten games on each set, with five different levels each). *Public* sets of games are included in the framework and allow participants to train their agents on them. For each year, there is one *validation* and one *test* set. Both the sets are private and stored in the competition server.[2] Participants can submit their entries any time before the submission deadline to all training and validation sets, and preliminary rankings are displayed in the competition website (the names of the validation set games are anonymous).

[2]www.gvgai.net; Intel Core i5 machine, 2.90 GHz, and 4 GB of memory.

TABLE I
Winners of all Editions of the GVGAI Planning Competition

Contest Leg	**Winner**	**Type**	**Section**
CIG-14	OLETS	Tree Search Method	IV-B [8]
GECCO-15	YOLOBOT	Hyper-heuristic	IV-E [17]
CIG-15	Return42	Hyper-heuristic	IV-E [16]
CEEC-15	YBCriber	Hybrid	IV-D [18]
GECCO-16	YOLOBOT	Hyper-heuristic	IV-E [17]
CIG-16	MaastCTS2	Tree Search Method	IV-B [19]
WCCI-16 (2P)	ToVo2	Hybrid	V-A [10]
CIG-16 (2P)	Number27	Hybrid	V-B [10]
GECCO-17	YOLOBOT	Hyper-heuristic	IV-E [17]
CEC-17 (2P)	ToVo2	Hybrid	V-A [10]
WCCI-18 (1P)	YOLOBOT	Hyper-heuristic	IV-E [17]
FDG-18 (2P)	OLETS	Tree Search Method	IV-B [10]

A. Game Playing Tracks

In the game playing tracks (planning and learning settings), the competition rankings are computed by first sorting all entries per game according to victory rates, scores, and game lengths, in this order. These per-game rankings award points to the first 10 entries, from the first position to the tenth position: $25, 18, 15, 12, 10, 8, 6, 4, 2$, and 1. The winner of the competition is the submission that sums more points across all games in the test set. For a more detailed description of the competition and its rules, the reader is referred to [8]. All controllers are run on the test set after the submission deadline to determine the final rankings of the competition, executing each agent multiple times on each level.

1) Planning Tracks: The first GVGAI competition ever held featured the single-player planning track, in 2014. A full description of this competition can be found in [8]. The year 2015 featured three legs in a year-long championship, each one of them with different validation and test sets. The two-player planning track [9] was added in 2016, with the aim of testing general AI agents in environments that are more complex and present more direct player interaction [10]. Since then, the single- and two-player tracks have run in parallel until 2018.

Table I shows the winners of all editions up to date, along with the section of this survey in which the method is included and the paper that describes the approach more in depth.

2) Learning Track: The GVGAI single-player learning track has run for two years: 2017 and 2018, both at the IEEE Conference on Computational Intelligence and Games (CIG).

In the 2017 edition, the execution of controllers was divided into two phases: learning and validation. In the learning phase, each controller has a limited amount of time, 5 min, for learning the first three levels of each game. The agent could play as many times as desired, choosing among these three levels, as long as the 5-min time limit is respected. In the validation phase, the controller plays ten times the levels 4 and 5 sequentially. The results obtained in these validation levels are the ones used in the competition to rank the entries. Besides the two sample random agents written in Java and Python and one sample

TABLE II
SCORE AND RANKING OF THE SUBMITTED AGENTS IN THE 2018'S GVGAI LEARNING COMPETITION

Game	Game 1			Game 2			Game 3			Ranking
Level	**3**	**4**	**5**	**3**	**4**	**5**	**3**	**4**	**5**	
fraBot-RL-Sarsa	-2	**1**	-1	0	0	0	**2**	**3**	**2**	1
fraBot-RL-QLearning	-2	-1	-2	0	0	0	1	0	**2**	2
Random††	-0.5	**0.2**	**-0.1**	0	0	0	**3.5**	**0.7**	**2.7**	3
DQN†	**61.5**	-1	**0.3**	0	0	0	-	-	-	-
Prioritized Dueling DQN†	**36.8**	-1	-2	0	0	0	-	-	-	-
A2C†	8.1	-1	-2	0	0	0	-	-	-	-
OLETS Planning Agent	41.7	48.6	3.1	0	0	2.2	4.2	8.1	14	-

agent using Sarsa written in Java, the first GVGAI single-player learning track received three submissions written in Java and one in Python [20]. The winner of this track is a naive implementation of Q-Learning algorithm (see Section VI-A4).

The 2018 edition featured, for the first time, the integration of the framework with the OpenAI Gym API [15], which results as GVGAI Gym.[3] This edition also ran with some relaxed constraints. First, only three games are used for the competition, and they are made public. Only two levels for each are provided to the participants for training purposes, while the other three are kept secret and used for computing the final results. Second, each agent has an increased decision time of 100 ms. Third, the participants were free to train their agent by themselves using as much time and computational resources as they wanted before the submission deadline.

This edition of the competition received only two entries, *fraBot-RL-QLearning* and *fraBot-RL-Sarsa*, submitted by the same group of contributors from the Frankfurt University of Applied Science. The results of the entries and sample agents (*random*, *DQN*, *Prioritized Dueling DQN,* and *A2C* [15]) are summarized in Table II. For comparison, the planning agent *open loop expectimax tree search (OLETS)* (with access to the forward model) is included. *DQN* and *Prioritized Dueling DQN* are outstanding on level 3 (test level) of game 1 because level 3 is very similar to level 2 (training level). Interestingly, the sample learning agent *DQN* outperformed *OLETS* on the third level of game 1. *DQN*, *Prioritized Dueling DQN* and *A2C* are not applied to game 3 because of the different game screen dimensions of different levels. We would like to refer the readers to [15] for more information about the GVGAI Gym.

B. PCG Tracks

In the PCG tracks, participants develop generators for levels or rules that are adequate for any game or level (respectively) given. Due to the inherent subjective nature of content generation, the evaluation of the entries is done by human judges who attend the conference where the competition takes place. For both tracks, during the competition day, judges are encouraged to try pairs of generated content and select the one they liked (one, both, or neither). Finally, the winner was selected based on the generator with more votes.

1) Level Generation Track: The first level generation competition was held at the International Joint Conference on Artificial Intelligence (IJCAI), in 2016. This competition received four participants. Each one of them was provided a month to submit a new level generator. Three different level generators were provided in order to help the users get started with the system (see Section VII for a description of these). Three out of the four participants were simulation-based level generators, while the remaining one was based on cellular automata. The winner of the contest was the Easablade generator, a cellular automata described in Section VII-A4. The competition was run again on the following year at IEEE CIG 2017. Unfortunately, only one submission was received; hence, the competition was canceled. This submission used an n-gram model to generate new constrained levels using a recorded player keystrokes.

2) Rule Generation Track: The rule generation track [12] was introduced and held during CIG 2017. Three different sample generators were provided (see Section VIII), and the contest ran over a month's period. Unfortunately, no submissions were received for this track.

IV. METHODS FOR SINGLE-PLAYER PLANNING

This section describes the different methods that have been implemented for single player planning in GVGAI. All the controllers that face this challenge have in common the possibility of using the forward model to sample future states from the current game state, plus the fact that they have a limited action-decision time. While most attempts abide by the 40-ms decision time imposed by the competition, other efforts in the literature compel their agents to obey a maximum number of calls of the forward model.

Section IV-A briefly introduces the most basic methods that can be found within the framework. Then, Section IV-B describes the different tree search methods that have been implemented for this setting by the community, followed by evolutionary methods in Section IV-C. Often, more than one method is combined into the algorithm, which gives place to hybrid methods (see Section IV-D) or hyperheuristic algorithms (see Section IV-E). Further discussion on these methods and their common takeaways have been included in Section X.

A. Basic Methods

The GVGAI framework contains several agents aimed at demonstrating how a controller can be created for the single-player planning track of the competition [8]. Therefore, these methods are not particularly strong.

The simplest of all methods is, without much doubt, *doNothing*. This agent returns the action *nil* at every game tick without exception. The next agent in complexity is *sampleRandom*, which returns a random action at each game tick. Finally, *onesteplookahead* is another sample controller that rolls the model forward for each one of the available actions in order to select the one with the highest action value, determined by a function that tries to maximize score while minimizing distances to NPCs and portals.

[3]https://github.com/rubenrtorrado/GVGAI_GYM

B. Tree Search Methods

One of the strongest and influential sample controllers is *sampleMCTS*, which implements the Monte Carlo tree search (MCTS) algorithm for real-time games. Initially implemented in a *closed loop* version (the states visited are stored in the tree node, without calling forward model during the tree policy phase of MCTS), it achieved the third position (out of 18 participants) in the first edition of the competition.

The winner of that edition, Couëtoux, implemented OLETS, which is an *open loop* (states visited are never stored in the associated tree node) version of MCTS, which does not include roll-outs and uses open loop expectimax (OLE) for the tree policy. OLE substitutes the empirical average reward by r_M, a weighted sum of the empirical average of rewards, and the maximum of its children r_M values [8].

Schuster, in his M.Sc. thesis [21], analyzes several enhancements and variations for MCTS in different sets of the GVGAI framework. These modifications included different tree selection, expansion, and play-out policies. Results show that combinations of move-average sampling technique (MAST) and n-gram selection technique (NST) with progressive history provided an overall higher rate of victories than their counterparts without these enhancements; although this result was not consistent across all games (with some simpler algorithms achieving similar results).

In a different study, Soemers [19], [22] explored multiple enhancements for MCTS: progressive history and NST for the tree selection and play-out steps, tree reuse (by starting at each game tick with the subtree grown in the previous frame that corresponds to the action taken, rather than a new root node), breadfirst tree initialization (direct successors of the root note are explored before MCTS starts), safety prepruning (prune those nodes with high number of game loses found), loss avoidance (MCTS ignores game lose states when found for the first time by choosing a better alternative), novelty-based pruning (in which states with features rarely seen are less likely to be pruned), knowledge-based evaluation [23], and deterministic game detection. The authors experimented with all these enhancements in 60 games of the framework, showing that most of them improved the performance of MCTS significantly, and their all-in-one combination increased the average win rate of the sample agent in 17 percentage points. The best configuration was the winner of one of the editions of the 2016 competitions (see Table I).

F. Frydenberg [24] *et al.* studied yet another set of enhancements for MCTS. The authors showed that using MixMax backups (weighing average and maximum rewards on each node) improved the performance in only some games, but its combination with reversal penalty (to penalize visiting the same location twice in a play-out) offers better results than those of vanilla MCTS. Other enhancements, such as macroactions (by repeating an action several times in a sequence) and partial expansion (a child node is considered expanded only if its children have also been expanded) did not improve the results obtained.

Perez-Liebana *et al.* [23] implemented KB-MCTS, a version of MCTS with two main enhancements. First, distances to different sprites were considered features for a linear combination, where the weights were evolved to bias the MCTS roll-outs. Second, a knowledge base (KB) is kept about how *interesting* for the player the different sprites are, where *interesting* is a measure of curiosity (roll-outs are biased toward unknown sprites) and experience (a positive/negative bias for getting closer/farther to beneficial/harmful entities). The results of applying this algorithm to the first set of games of the framework showed that the combination of these two components gave a boost in performance in most games of the first training set.

The work in [23] has been extended by other researchers in the field, which also puts a special effort on biasing the Monte Carlo (MC) simulations. Chu *et al.* [25] modified the random action selection in MCTS roll-outs by using potential fields, which bias the roll-outs by making the agent move in a direction akin to the field. The authors showed that KB-MCTS provides a better performance if this potential field is used instead of the Euclidean distance between sprites implemented in [23]. Additionally, in a similar study [26], the authors substituted the Euclidean distance for a measure calculated by a path-finding algorithm. This addition achieved some improvements over the original KB-MCTS, although the authors noted in their study that using path-finding does not provide a competitive advantage in all games.

Another work by Park and Kim [27] tackles this challenge by: 1) determining the goodness of the other sprites in the game; 2) computing an influence map (IM) based on this; and 3) using the IM to bias the simulations, in this occasion by adding a third term to the upper confidence bound (UCB) equation [1] for the tree policy of MCTS. Although not compared with KB-MCTS, the resultant algorithm improves the performance of the sample controllers in several games of the framework, albeit performing worse than these in some of the games used in the study.

Biasing roll-outs is also attempted by Santos *et al.* [28], who introduced redundant action avoidance (RAA) and nondefeat policy (NDP); RAA analyzes changes in the state to avoid selecting sequences of actions that do not produce any alteration on position, orientation, properties, or new sprites in the avatar. NDP makes the recommendation policy ignore all children of the root node who found at least one game loss in a simulation from that state. If all children are marked with a defeat, normal (higher number of visits) recommendation is followed. Again, both modifications are able to improve the performance of MCTS in some of the games, but not in all.

Waard *et al.* [29] introduced the concept of options of macroactions in GVGAI and designed option MCTS (O-MCTS). Each option is associated with a goal, a policy, and a termination condition. The selection and expansion steps in MCTS are modified so the search tree branches only if an option is finished, allowing for a deeper search in the same amount of time. Their results show that O-MCTS outperforms MCTS in games with small levels or a few number of sprites, but loses in comparison to MCTS when the games are bigger due to these options becoming too large.

In a similar line, Perez-Liebana *et al.* [13] employed macroactions for GVGAI games that used continuous (rather than grid-based) physics. These games have a larger state space, which, in turn, delays the effects of the player's actions and modifies the way agents navigate through the level. Macroactions are defined as a sequence or repetition of the same action during M steps, which is arguably the simplest kind of macroactions that can be devised. MCTS performed better *without* macroactions on average across games, but there are particular games where MCTS needs macroactions to avoid losing at every attempt. The authors also concluded that the length M of the macroactions impacts different games distinctly, although shorter ones seem to provide better results than larger ones, probably due to a more fine control in the movement of the agents.

Some studies have brought multiobjective optimization to this challenge. For instance, Perez-Liebana *et al.* [30] implemented a multiobjective version of MCTS, concretely maximizing score and level exploration simultaneously. In the games tested, the rate of victories grew from 32.24% (normal MCTS) to 42.38% in the multiobjective version, showing great promise for this approach. In a different study, Khalifa *et al.* [31] applied multiobjective concepts to evolving parameters for a tree selection confidence bounds equation. A previous work by Bravi [32] (also discussed in Section IV-D) provided multiple UCB equations for different games. The work in [31] evolved, using S-metric selection evolutionary multiobjective optimization algorithm, the linear weights of a UCB equation that results from combining all from [32] in a single one. All these components respond to different and conflicting objectives, and their results show that it is possible to find good solutions for the games tested.

A significant exception to MCTS with regard to tree search methods for GVGAI is that of Geffner and Geffner [18] (winner of one of the editions of the 2015 competition, *YBCriber*, as indicated in Table I), who implemented iterated width [IW; concretely IW(1)]. IW(1) is a breadth-first search with a crucial alteration: a new state found during search is pruned if it does not make true a new tuple of at most 1 atom, where *atoms* are Boolean variables that refer to position (and orientations in the case of avatars) changes of certain sprites at specific locations. The authors found that IW(1) performed better than MCTS in many games, with the exception of puzzles, where IW(2) (pruning according to pairs of atoms) showed better performance. This agent was declared winner in the CEEC 2015 edition of the single-player planning track [3].

Babadi [33] implemented several versions of enforced hill climbing (EHC), a breadth-first search method that looks for a successor of the current state with a better heuristic value. EHC obtained similar results to KB-MCTS in the first set of games of the framework, with a few disparities in specific games of the set.

Nelson [34] ran a study on MCTS in order to investigate if, giving a higher time budget to the algorithm (i.e., increasing the number of iterations), MCTS was able to master most of the games. In other words, if the real-time nature of the GVGAI framework and competition is the reason why different approaches fail to achieve a high victory rate. This study provided up to 30 times more budget to the agent, but the performance of MCTS only increased marginally even at that level. In fact, this improvement was achieved by means of losing less often rather than by winning more games. This paper concludes that the real-time aspect is not only a factor in the challenge, but also the diversity in the games. In other words, increasing the computational budget is not the answer to the problem that GVGAI poses, at least for MCTS.

Finally, another study on the uses of MCTS for single-player planning is carried out by Bravi *et al.* [35]. In this paper, the focus is set on understanding why and under which circumstances different MCTS agents make different decisions, allowing for a more in-depth description and behavioral logging. This study proposes the analysis of different metrics (recommended action and their probabilities, action values, consumed budget before converging on a decision, etc.) recorded via a *shadow* proxy agent, used to compare algorithms in pairs. The analysis described in the paper shows that traditional win-rate performance can be enhanced with these metrics in order to compare two or more approaches.

C. *Evolutionary Methods*

The second big group of algorithms used for single-player planning is that of evolutionary algorithms (EAs). Concretely, the use of EAs for this real-time problem is mostly implemented in the form of rolling horizon EAs (RHEAs). This family of algorithms evolves sequences of actions with the use of the forward model. Each sequence is an individual of an EA whose fitness is the value of the state found at the end of the sequence. Once the time budget is up, the first action of the sequence with the highest fitness is chosen to be applied in that time step.

The GVGAI competition includes *SampleRHEA* as a sample controller. *SampleRHEA* has a population size of 10, individual length of 10, and implements uniform crossover and mutation, where one action in the sequence is changed for another one (position and new action chosen uniformly at random) [8].

Gaina *et al.* [36] analyzed the effects of the RHEA parameters on the performance of the algorithm in 20 games, chosen among the existent ones in order to have a representative set of all the games in the framework. The parameters analyzed were population size and individual length, and results showed that higher values for both parameters provided higher victory rates. This study motivated the inclusion of random search (*SampleRS*) as a sample in the framework, which is equivalent to RHEA but with an infinite population size (i.e., only one generation is evaluated until budget is consumed) and achieves better results than those achieved by RHEA in some games. Gaina *et al.* [36] also compared RHEA with MCTS, showing better performance for an individual length of 10 and high population sizes.

Santos *et al.* [37] implemented three variants for RHEA with shifted buffer (RHEA-SB) by: 1) applying the one-step-look-ahead algorithm after the buffer shifting phase; 2) applying a spatial RAA policy [28]; and 3) applying both the techniques. The experimental tests on 20 GVGAI single-player games showed that the third variant of RHEA-SB achieved promising results.

Santos and Bernardino [38] applied the avatar-related information, spacial exploration and knowledge obtained during game playing to the game state evaluation of RHEA. These game state evaluation enhancements have also been tested on an MCTS agent. The enhancements significantly increased the win rate and game score obtained by RHEA and MCTS on 20 tested games.

A different type of information was used by Gaina *et al.* [39] to dynamically adjust the length of the individuals in RHEA: The flatness of the fitness landscape is used to shorten or lengthen the individuals in order for the algorithm to better deal with sparse reward environments (using longer roll-outs for identification of further away rewards), while not harming performance in dense reward games (using shorter roll-outs for focus on immediate rewards). However, this had a detrimental effect in RHEA, while boosting MCTS results. Simply increasing the roll-out length proved to be more effective than this initial attempt at using the internal agent state to affect the search itself.

A different evolutionary computation agent was proposed by Jia *et al.* [40], [41], which consists of a genetic programming (GP) approach. The authors extract features from a screen capture of the game, such as avatar location and the positions and distances to the nearest object of each type. These features are inputs to a GP system that, using arithmetic operands as nodes, determines the action to execute as a result of three trees (horizontal, vertical, and action use). The authors report that all the different variations of the inputs provided to the GP algorithm give similar results to those of MCTS, on the three games tested in their study.

D. Hybrids

The previous studies feature techniques in which one technique is predominant in the agent created; albeit they may include enhancements that can place them in the boundary of hybrids. This section describes those approaches that, in the opinion of the authors, would in their own right be considered as techniques that mix more than one approach in the same, single algorithm.

An example of these approaches is presented by Gaina *et al.* [42], which analyzed the effects of seeding the initial population of a RHEA using different methods. Part of the decision time budget is dedicated to initialize a population with sequences that are promising, as determined by *onesteplookahead* and *MCTS* agents. Results show that both seeding options provide a boost in victory rate when population size and individual length are small, but the benefits vanish when these parameters are large.

Other enhancements for RHEA proposed in [43] are incorporating a bandit-based mutation, a statistical tree, a shifted buffer, and roll-outs at the end of the sequences. The bandit-based mutation breaks the uniformity of the random mutations in order to choose new values according to suggestions given by a univariate armed bandit. However, the authors reported that no improvement on the performance was noticed. A statistical tree, previously introduced in [44], keeps the visit count and accumulated rewards in the root node, which are subsequently used for recommending the action to take at that time step. This enhancement produced better results with smaller individual length and smaller population sizes. The shifted buffer enhancement provided the best improvement in the performance, which consist of shifting the sequences of the individuals of the population one action to the left, removing the action from the previous time step. This variation, similar to keeping the tree between frames in MCTS, combined with the addition of roll-outs at the end of the sequences provided an improvement in victory rate (20 percentile points over vanilla RHEA) and scores.

A similar (and previous) study was conducted by Horn *et al.* [45]. In particular, this study features RHEA with roll-outs (as in [43]), RHEA with MCTS for alternative actions (where MCTS can determine any action with the exception of the one recommended by RHEA), RHEA with roll-outs and sequence planning (same approach as the shifted buffer in [43]), RHEA with roll-outs and occlusion detection (which removes unneeded actions in a sequence that reaches a reward), and RHEA with roll-outs and NPC attitude check (which rewards sequences in terms of proximity to sprites that provide a positive or negative reward). Results show that RHEA with roll-outs improved performance in many games, although all the other variants and additions performed worse than the sample agents. It is interesting to see that in this case the shifted buffer did not provide an improvement in the victory rate, although this may be due to the use of different games.

Schuster [21] proposed two methods that combine MCTS with evolution. One of them, (1+1)-EA as proposed by [23], evolves a vector of weights for a set of game features in order to bias the roll-outs toward more interesting parts of the search space. Each roll-out becomes an evaluation for an individual (weight vector), using the value of the final state as fitness. The second algorithm is based on strongly typed GP and uses game features to evolve state evaluation functions that are embedded within MCTS. These two approaches join MAST and NST (see Section IV-B) in a larger comparison, and the study concludes that different algorithms outperform others in distinct games, without an overall winner in terms of superior victory rate, although superior to vanilla MCTS in most cases.

The idea of evolving weight vectors for game features during the MCTS roll-outs introduced in [23] (KB-MCTS)[4] was explored further by Eeden in his M.Sc. thesis [46]. In particular, the author added A* as a path-finding algorithm to replace the Euclidean distance used in KB-MCTS for a more accurate measure and changing the evolutionary approach. While KB-MCTS used a weight for each pair feature-action, being the action chosen at each step by the Softmax equation, this paper combines all move actions on a single weight and picks the action using Gibbs sampling. The author concludes that the improvements achieved by these modifications are marginal, and likely due to the inclusion of path-finding.

Additional improvements on KB-MCTS are proposed by Chu *et al.* [47]. The authors replace the Euclidean distance feature to sprites with a grid view of the agent's surroundings, and also the (1+1)-EA with a Q-Learning approach to bias the MCTS roll-outs, making the algorithm update the weights at each step in the roll-out. The proposed modifications improved the victory rate in several sets of games of the framework and also achieved

[4]This approach could also be considered an hybrid. Given its influence in other tree approaches, it has also been partially described in Section IV-B

the highest average victory rate among the algorithms it was compared with.

İlhan and Etaner-Uyar [48] implemented a combination of MCTS and *true online* Sarsa (λ). The authors use MCTS roll-outs as episodes of past experience, executing true online Sarsa at each iteration with a ϵ-greedy selection policy. Weights are learnt for features taken as the smallest Euclidean distance to sprites of each type. Results showed that the proposed approaches improved the performance on vanilla MCTS in the majority of the 10 games used in the study.

Evolution and MCTS have also been combined in different ways. In one of them, Bravi *et al.* [49] used a GP system to evolve different tree policies for MCTS. Concretely, the authors evolve a different policy for each one of the five games employed in the study, aiming to exploit the characteristics of each game in particular. The results showed that the tree policy plays a very important role on the performance of the MCTS agent, although in most cases the performance is poor—none of the evolved heuristics performed better than the default UCB in MCTS.

Finally, Sironi *et al.* [50] designed three self-adaptive MCTS that tuned the parameters of MCTS (play-out depth and exploration factor) on-line, using naive MC, an (λ, μ)-EA, and the N-tuple bandit EA (NTBEA) [51]. Results show that all tuning algorithms improve the performance of MCTS where vanilla MCTS performs poorly, while keeping a similar rate of victories in those where MCTS performs well. In a follow-up study, however, Sironi and Winands [52] extend the experimental study to show that online parameter tuning impacts the performance in only a few GVGP games, with NTBEA improving performance significantly in only one of them. The authors conclude that online tuning is more suitable for games with longer budget times, as it struggles to improve performance in most GVGAI real-time games.

E. Hyperheuristics/Algorithm Selection

Several authors have also proposed agents that use several algorithms, but rather than combining them into a single one, there is a higher level decision process that determines which one of them should be used at each time.

Ross, in his M.Sc. thesis [53] proposes an agent that is a combination of two methods. This approach uses A* with EHC to navigate through the game at a high level and switches to MCTS when in close proximity to the goal. The work highlights the problems of computing paths in the short time budget allowed, but indicate that goal targeting with path-finding combined with local maneuvering using MCTS does provide good performance in some of the games tested.

Joppen *et al.* [17] implemented *YOLOBOT*, arguably the most successful agent for GVGAI up to date, as it has won several editions of the competition. Their approach consists of a combination of two methods: a heuristic best first search (BFS) for deterministic environments and MCTS for stochastic games. Initially, the algorithm employs BFS until the game is deemed stochastic, an optimal solution is found, or a certain game tick threshold is reached, extending through several consecutive frames if needed for the search. Unless the optimal sequence of actions is found, the agent will execute an enhanced MCTS consistent of informed priors and roll-out policies, backtracking, early cutoffs, and pruning. The resultant agent has shown consistently a good level of play in multiple game sets of the framework.

Another hyperheuristic approach, also the winner of one of the 2015 editions of the competition (*Return42*, see Table I), determines first if the game is deterministic or stochastic. In case of the former, A* is used to direct the agent to sprites of interest. Otherwise, random walks are employed to navigate through the level [16].

Azaria *et al.* [54] applied GP to evolve hyperheuristic-based agents. The authors evolved three step-lookahead agents, which were tested on the three game sets from the first 2014 GVGAI competition. The resultant agent was able to outperform the agent ranked at the third place in the competition (sample MCTS).

The fact that this type of portfolio agents has shown very promising results has triggered more research into hyperheuristics and game classification. The work by Bontrager *et al.* [55] used K-means to cluster games and algorithms attending to game features derived from the type of sprites declared in the VGDL files. The resulting classification seemed to follow a difficulty pattern, with four clusters that grouped games that were won by the agents at different rates.

Mendes *et al.* [56] built a hyperagent that selected automatically an agent from a portfolio of agents for playing individual game and tested it on the GVGAI framework. This approached employed game-based features to train different classifiers (support vector machines—SVM, multilayer perceptrons, decision trees—J48, among others) in order to select which agent should be used for playing each game. Results show that the SVM and J48 hyperheuristics obtained a higher victory rate than that of the single agents separately.

Horn *et al.* [45] (described before in Section IV-D) also includes an analysis on game features and difficulty estimation. The authors suggest that the multiple enhancements that are constantly attempted in many algorithms could potentially be switched ON and OFFdepending on the game that is being played, with the objective of dynamically adapting to the present circumstances.

Ashlock *et al.* [16] suggest the possibility of creating a classification of games, based on the performance of multiple agents (and their variations: different enhancements, heuristics, objectives) on them. Furthermore, this classification needs to be stable, in order to accommodate the ever-increasing collection of games within the GVGAI framework, but also flexible enough to allow a hyperheuristic algorithm to choose the version that better adapts to unseen games.

Finally, Gaina *et al.* [57] gave a first step toward algorithm selection from a different angle. The authors trained several classifiers on agent log data across 80 games of the GVGAI framework, in particular obtained only from player experience (i.e., features extracted from the way search was conducted, rather than potentially human-biased game features), to determine if the game will be won or not at the end. Three models are trained, for the early, mid, and late game, respectively, and tested in previously not seen games. Results show that these predictors are able to foresee, with high reliability, if the agent is going to lose or win the game. These models would, therefore, allow to indicate when and if the algorithm used to play the game should be changed. A visualization of these agent features, including win prediction, displayed live while playing games, is available using the VertigØ tool [58], which means to offer better agent analysis for deeper understanding of the agents' decision making process, debugging, and game testing.

V. Methods for Two-Player Planning

This section approaches agents developed by researchers within the two-player planning setting. Most of these entries have been submitted to the two-player planning track of the competition [9]. Two methods stood out as the base of most entries received so far, MCTS and EA [10]. On the one hand, MCTS performed better in cooperative games, showing the ability to adapt better to asymmetric games, which involved a role switch between matches in the same environment. EAs, on the other hand, excelled in games with long lookaheads, such as puzzle games, which rely on a specific sequence of moves being identified.

Counterparts of the basic methods described in Section IV-A are available in the framework as well, the only difference being in the one step lookahead agent, which requires an action to be supplied for the opponent when simulating game states. The opponent model used by the sample agent assumes they will perform a random move (with the exception of those actions that would cause a loss of the game).

A. Tree Search Methods

Most of the competition entries in the first three seasons (2016–2018) were based on MCTS (see Section IV-B). It is interesting to note that the 2016 winner won again in 2018—highlighting the difficulty of the challenge and showing the need for more research focus on multiplayer games for better and faster progress.

Some entries employed an open loop version of MCTS, which would only store statistics in the nodes of the trees and not game states, thereby needing to simulate through the actions at each iteration for a potentially more accurate evaluation of the possible game states. Due to this being unnecessarily costly in deterministic games, some entries such as *MaasCTS2* and *YOLOBOT* switched to breadth-first search in such games after an initial analysis of the game type, a method that has shown ability to find the optimal solution if the game lasts long enough.

Enhancements brought to MCTS include generating value maps, either regarding physical positions in the level, or higher level concepts (such as higher values being assigned to states where the agent is closer to objects it has not interacted with before; or interesting targets as determined by controller-specific heuristics). The winner of the 2016 WCCI and 2017 CEC legs, *ToVo2*, also employed dynamic MC roll-out length adjustments (increased with the number of iterations to encourage further lookahead if budget allows) and weighted roll-outs (the weights per action generated randomly at the beginning of each roll-out).

All agents use online learning in one way or another (the simplest form being the base MCTS backups, used to gather statistics about each action through multiple simulations), but only the overall 2016 and 2018 championship winner, *adrienctx*, uses offline learning on the training set supplied to tune the parameters in the stochastic gradient descent function employed, learning rate and mini batch size.

B. Evolutionary Methods

Two of the 2016 competition entries used an EA technique as a base as an alternative to MCTS: Number27 and CatLinux [10].

Number27 was the winner of the CIG 2016 leg, the controller placing the fourth position overall in the 2016 Championship. Number27 uses a genetic algorithm (GA), with one population containing individuals that represent fixed-length action sequences. The main improvement it features on top of the base method is the generation of a value heat-map, used to encourage the agent's exploration toward interesting parts of the level. The heat-map is initialized based on the inverse frequency of each object type (therefore a lower value the higher the object number) and including a range of influence on nearby tiles. The event history is used to evaluate game objects during simulations and to update the value map.

CatLinux was not a top controller on either of the individual legs run in 2016, but secured the fifth position overall in the championship. This agent uses a RHEA. A shift buffer enhancement is used to boost the performance, specifically keeping the population evolved during one game tick in the next, instead of discarding it; each action sequence is shifted one action to the left (therefore removing the previous game step), and a new random action is added at the end to complete the individual to its fixed length.

No offline learning was used by any of the EA agents, although there could be scope for improvement through parameter tuning (offline or online).

C. Opponent Model

Most agents submitted to the two-player competition use completely random opponent models. Some entries have adopted the method integrated within the sample *One Step Lookahead* controller, choosing a random but nonlosing action. In the 2016 competition, *webpigeon* assumed the opponent would always cooperate and would, therefore, play a move beneficial to the agent. *MaasCTS2* used the only advanced model at the time: It remembered Q-values for the opponent actions during simulations and added them to the statistics stored in the MCTS tree nodes; an ϵ-greedy policy was used to select opponent actions based on the Q-values recorded. This provided a boost in the performance on the games in the WCCI 2016 leg, but it did not improve the controller's position in the rankings for the following CIG 2016 leg. Most entries in the years 2017 and 2018 seasons employed simple random opponent models.

Opponent models were found to be an area to explore further in [10] and Gonzalez and Perez-Liebana looked at nine different models integrated within the sample MCTS agent provided with the framework [59]. *Alphabeta* builds a tree incrementally, returning the best possible action in each time tick, while *Minimum* returns the worst possible action. *Average* uses a similar tree structure, but it computes the average reward over all the actions, and it returns the action closest to the average. *Fallible* returns the best possible action with a probability $p = 0.8$ and the action with the minimum reward otherwise. *Probabilistic* involved offline learning over 20 games in the GVGAI framework in order to determine the probability of an MCTS agent to select each action and then using these to determine the opponent action while playing online. *Same Action* returns the same action the agent plays, while *Mirror* returns its opposite. Finally, *LimitedBuffer* records the last $n = 20$ actions performed by the player and builds probabilities of selecting the next action based on this data, while *UnlimitedBuffer* records the entire history of actions during the game. When all nine opponent models were tested in a round-robin tournament against each other, the probabilistic models achieve the highest win rates and two models, *Probabilistic* and *UnlimitedBuffer,* outperforming a random opponent model.

Finally, the work done on two-player GVGAI has inspired other research on Mathematical General Game Playing. Ashlock *et al.* [60] implemented general agents for three different mathematical coordination games, including the Prisoner's Dilemma. Games were presented at once, but switching between them at certain points and experiments show that agents can learn to play these games and recognize when the game has changed.

VI. Methods for Single-Player Learning

The GVGAI framework has also been used from an agent learning perspective. In this setting, the agents do not use the forward model to plan ahead actions to execute in the real game. Instead, the algorithms learn the games by repeatedly playing them multiple times (as *episodes* in reinforcement learning), ideally improving their performance progressively. This section describes first the approaches that tackled the challenge set in the single-player learning track of the 2017 and 2018 competitions and then moves to other approaches.

A. Competition Entries

1) Random Agent: A sample random agent, which selects an action uniformly at random at every game tick, is included in the framework (in both Java and Python) for the purposes of testing. This agent is also meant to be taken as a baseline: A learner is expected to perform better than an agent that acts randomly and does not undertake any learning.

2) Multiarmed Bandit (MAB) Algorithm: DontUnderestimateUchiha by K. Kunanusont is based on two popular MAB algorithms, ϵ-decreasing greedy algorithm and UCBs. At any game tick T, the current *best* action with probability $1 - \epsilon_T$ is picked; otherwise, an action is uniformly randomly selected. The *best* action at time T is determined using UCB with increment of score as reward. This is a very interesting combination, as the UCB-style selection and the ϵ-decreasing greedy algorithm both aim at balancing the tradeoff between exploiting more the best-so-far action and exploring others. Additionally, ϵ_0 is set to 0.5, and it decreases slowly along time, formalized as $\epsilon_T = \epsilon_0 - 0.0001T$. According to the competition setting, all games will last longer than 2,000 game ticks, so $\forall T \in \{1, \ldots, 2000\}$, $0.5 \geq \epsilon_T \geq 0.3$. As a result, random decisions are made for approximately 40% time.

3) State-Action-Reward-State-Action (Sarsa) Algorithm: sampleLearner, *ercumentilhan,* and *fraBot-RL-Sarsa* are based on Sarsa [2]. The *sampleLearner* and *ercumentilhan* use a subset of the whole game state information to build a new state to reduce the amount of information to be saved and to take into account similar situations. The main difference is that the former uses a square region with fixed size centered at the avatar's position, while the latter uses a first-person view with a fixed distance. *fraBot-RL-Sarsa* uses Sarsa, and it uses the entire screenshot of the game screen as input provided by GVGAI Gym. The agent has been trained using 1000 episodes for each level of each game, and the total training time was 48 h.

4) Q-Learning: kkunan, by K. Kunanusont, is a simple Q-learning [2] agent using most of the avatar's current information as features, which a few exceptions (such as avatar's health and screen size, as these elements that vary greatly from game to game). The reward at game tick $t + 1$ is defined as the difference between the score at $t + 1$ and the one at t. The learning rate α and discounted factor γ are manually set to 0.05 and 0.8. During the *learning phase*, a random action is performed with probability $\epsilon = 0.1$; otherwise, the best action is selected. During the *validation phase*, the best action is always selected. Despite its simplicity, it won the first track in 2017. *fraBot-RL-QLearning* uses the Q-Learning algorithm. It has been trained using 1000 episodes for each level of each game, and the total training time was 48 h.

5) Tree Search Methods: YOLOBOT is an adaption of the *YOLOBOT* planning agent (as described previously in Section IV-E). As the forward model is no more accessible in the learning track, the MCTS is substituted by a greedy algorithm to pick the action that minimizes the distance to the chosen object at most. According to the authors, the poor performance of *YOLOBOT* in the learning track, contrary to its success in the planning tracks, was due to the collision model created by themselves that did not work well.

B. Other Learning Agents

One of the first works that used this framework as a learning environment was carried out by Samothrakis *et al.* [61], who employed neuroevolution in 10 games of the benchmark. Concretely, the authors experimented with separable natural evolution strategies using two different policies (ϵ-greedy versus softmax) and a linear function approximator versus a neural network as a state evaluation function. Features like score, game status, avatar, and other sprites information were used to evolve learners during 1000 episodes. Results show that ϵ-greedy with a linear function approximator was a better combination to learn how to maximize scores on each game.

Braylan and Miikkulainen [62] performed a study in which the objective was to learn a forward model on 30 games. The objective was to learn the next state from the current one plus an action, where the state is defined as a collection of attribute values of the sprites (spawns, directions, movements, etc.), by means of logistic regression. Additionally, the authors transfer the learnt object models from game to game, under the assumption that many mechanics and behaviors are transferable between them. Experiments showed the effective value of object model transfer in the accuracy of learning forward models, resulting in these agents being stronger at exploration.

Also in a learning setting, Kunanusont *et al.* [63], [64] developed agents that were able to play several games via screen capture. In particular, the authors employed a deep Q-Network (DQN) in seven games of the framework of increasing complexity and included several enhancements to GVGAI to deal with different screen sizes and a nonvisualization game mode. Results showed that the approach allowed the agent to learn how to play in both deterministic and stochastic games, achieving a higher winning rate and game score as the number of episodes increased.

Apeldoorn and Kern-Isberner [65] proposed a learning agent that rapidly determines and exploits heuristics in an unknown environment by using a hybrid symbolic/subsymbolic agent model. The proposed agent-based model learned the weighted state-action pairs using a subsymbolic learning approach. The proposed agent has been tested on a single-player stochastic game, *Camel Race*, from the GVGAI framework, and won more than half of the games in different levels within the first 100 game ticks, while the standard Q-Learning agent never won given the same game length. Based on [65], Dockhorn and Apeldoorn [66] used exception-tolerant hierarchical KBs (HKBs) to learn the approximated forward model and tested the approach on the 2017 GVGAI learning track framework, respecting the competition rules. The proposed agent beats the best entry in the learning competition organized at CIG-17 [66], but still performed far worse than the best planning agents, which have access to the real forward models.

Using the new GVGAI Gym, Torrado *et al.* [15] compared three implemented deep reinforcement learning (DRL) algorithms of the OpenAI Gym, DQN, prioritized dueling DQN, and advance actor-critic (A2C), on eight GVGAI games with various difficulties and game rules. All the three RL agents perform well on most of the games; however, DQNs and A2C perform badly when no game score is given during a game playing (only win or loss is given when a game terminates). These three agents have been used as sample agents in the learning competition organized at CIG-18.

Finally, Justesen *et al.* [67] implemented A2C within the GVGAI-Gym interface in a training environment that allows learning by procedurally generating new levels. By varying the levels in which the agent plays, the resulting learning is more general and does not overfit to specific levels. The level generator creates levels at each episode, producing them in a slowly increasing level of difficulty in response to the observed agent performance.

C. Discussion

The presented agents differ from each other in the input game state (Json string or screen capture), the amount of learning time, and the algorithm used. Additionally, some of the agents have been tested on a different set of games and sometimes using different game length (i.e., maximal number of game ticks allowed). None of the agents, which were submitted to the 2017 learning competition, using the classic GVGAI framework, have used screen capture.

The Sarsa-based agents performed surprisingly bad in the competition, probably due to the arbitrarily chosen parameters and very short learning time. Also, learning three levels and testing on three more difficult levels given only 5 min learning time is a difficult task. An agent should take care of the learning budget distribution and decide when to stop learning a level and to proceed the next one.

The learning agent using exception-tolerant HKBs [66] learns fast. However, when longer learning time is allowed, it is dominated by DRL agents. Out of the eight games tested by Torrado *et al.* [15], none of the tested three DRL algorithms outperformed the planning agents on six games. However, on the heavily stochastic game Seaquest, A2C achieved almost double score than that of the best planning agent, MCTS.

VII. Methods for Level Generation

Different researchers used different approaches to generate levels for the GVGAI framework. The following section describes all known generators either included in the framework or developed during the competition.

A. Constructive Methods

Constructive generators are designed to generate levels based on general knowledge. For example, enemies should be away from the avatar, walls should not divide the world into islands, etc. Based on the game, the generator adjusts a couple of parameters and rules to fit the game as, for example, the number of NPCs in the generated level. Constructive generators do not need any simulations after generating the level. The following are the known constructive generators.

1) Sample Random Generator: This is the most naive method to generate a level. The generator first identifies solid sprites. Solid sprites block the avatar and all NPCs from moving and do not react to anything. The generator adds a selected solid sprite as a border for the generated level to prevent sprites from wandering outside the game screen, followed by adding one of each character in the level mapping section to a random location in the level. This step ensures the game is playable. Finally, it adds a random amount of random sprites from the level mapping to random locations in the level.

2) Sample Constructive Generator: This generator uses some general game knowledge to generate the level. First, the generator calculates the level dimensions and the number of sprites in the level, then labels game sprites based on their interactions and sprite types. After that, it constructs a level layout using the solid sprites, to later add the avatar to a random empty location. After knowing the avatar position, the generator adds harmful sprites (those that can kill the avatar) in a far location from the avatar and adds other sprites at any random free locations. Finally, the generator makes sure that the number of goal sprites is sufficient to prevent winning or losing automatically when the game starts.

3) Easablade Constructive Generator: This is the winner generator for the first level generator competition. The generator is similar to the sample constructive generator, but it uses cellular automata to generate the level instead of layering the objects randomly. The cellular automata is run on multiple layers. The first layer is to design the map obstacles, followed by the exit and the avatar, then the goal sprites, harmful sprites, and others.

4) N-Gram Constructive Generator: This generator uses an n-gram model to generate the level. The generator records the player actions from a previous play-through. This action sequence is used to generate the levels using predefined rules and constraints. For example, if the player uses the USE action quite often, the generator will include more enemies in the level. The n-gram is used to specify the rules. Instead of reacting to each separate action, the model reacts to an n-sequence of actions. During the generation process, the algorithm keeps track of the number and position of every generated object to ensure the generated sprites do not overpopulate the level. A single avatar sprite is placed in the lower half of the level.

5) Beaupre's Constructive Pattern Generator: Beaupre *et al.* [68] automatically analyzed 97 different games from the GVG-AI framework using a 3×3 sliding window over all the provided GVG-AI levels. They constructed a dictionary of all the different patterns (they discovered 12 941 unique patterns) with labels about the type of objects in them. The constructive generator starts by checking if the game contain solid sprites (sprites that does not allow player to pass through them). If that was the case, the generator fills the edges using border patterns (patterns that contain solid sprites and exist on the edge of the maps). The rest of the game area is filled by random selecting of patterns that maintain the following two heuristics: 1) only one avatar sprite should be found in the level; and 2) all nonsolid game areas are connected.

B. Search-Based Methods

Search-based generators use simulations to make sure the generated level is playable and better than just placing random objects. The following are the known search-based generators.

1) Sample Genetic Generator: This is a search-based level generator based on the feasible infeasible 2 population GA (FI2Pop). FI2Pop is a GA which uses two populations, one for feasible chromosomes and the other for infeasible chromosomes. The feasible population tries to increase the difference between the OLETS agent (see Section IV-B) and one-step look ahead, while the infeasible population tries to decrease the number of chromosomes that violate the problem constraints (i.e., at least one avatar in the game; the avatar must not die in the first 40 steps, etc.). Each population evolves on its own, where the children can transfer between the two populations. This generator initializes the population using sample constructive generator.

2) Amy12 Genetic Generator: This generator is built on top of the sample genetic generator. The main idea is to generate a level that fits a certain suspense curve. Suspense is calculated at each point in time by calculating the number of actions that leads to death or tie using the OLETS agent. The algorithm modifies the levels to make sure the suspense curve is not constant during the life time of the game. Good generators are aimed at producing three suspense peeks with values of 50% (where half of the actions, on average, lead to losing the game). One of the advantages of using this technique is that it makes sure that the generated level is winnable. Games that are hard to win will have a higher peak in the suspense curve, which is not valued highly by the generator.

3) Jnicho Genetic Generator: This generator [69] uses a standard GA with similar crossover and mutation operators to the sample GA. The fitness function used is a combination between the score difference and the constraints specified in the sample genetic generator. The score difference is calculated between an MCTS agent and one step look ahead agent. The score difference is normalized between 0 and 1 to make sure it will not overshadow the constraint values.

4) Number13 Genetic Generator: This is a modified version of the sample genetic generator. These modifications include using adaptive crossover mechanism, adaptive mutation rate, a better agent than OLETS, and allowing crossover between feasible and infeasible population, which is not allowed in the sample genetic generator.

5) Sharif's Pattern Generator: This generator is still work in progress. Sharif *et al.* [70] identified 23 different patterns by analyzing the grouping of different game sprites from several GVG-AI games. They are working now on using these design patterns as a fitness function for a search-based generator.

6) Beaupre's Evolutionary Pattern Generator: Similar to Beaupre's constructive pattern generator in Section VII-A5, they used the constructed dictionary for designing a search-based generator. They modified the sample genetic generator provided with the framework to work using patterns instead of using game sprites. They also initialized the generator using the constructive version to speed up the generation process.

C. Constraint-Based Methods

1) ASP Generator: This generator [71] uses answer set programming (ASP) to generate levels. The main idea is to generate ASP rules that generate suitable levels for the current game. The generated rules consists of three different types. The first type are basic rules, which are based on specific decisions to keep the levels simple (for instance, levels can only have one sprite per tile). The second type are game specific rules, which are extracted from the game description file. An example is the identification of singleton sprites that should only have one sprite in the level. The last type are additional rules to minimize the search space. These rules limit the minimum and maximum numbers of each sprite type. All the rules are evolved using evolutionary strategy with the algorithm performance difference between *sampleMCTS* and a random agent as the fitness function.

D. Discussion

The presented generators differ in the amount of time needed to generate a level and the features of the generated content. The constructive generators take the least amount of time to generate a single level without a guarantee that the generated level is beatable. However, both search-based and constraint-based generators take longer time, but generate challenging beatable levels as they use automated playing agents as a fitness function. The constraint-based generator only takes long time to find an ASP generator that could be used to generate many different levels as fast as the constructive generators, while search-based generators take a long time to find a group of similar looking levels.

For the generators that participated in the GVG-AI level generation competition (Easablade, Amy12, Jnicho, and Number13), they have been evaluated during IJCAI 2016 by asking the conference delegates to play two randomly selected levels and choose a preferred one. Each generator was used to generate three levels for four different games (The Snowman, Freeway, Run, and Butterflies). Easablade was chosen most often (78.4%), followed by Number13, amyP2, and jnicho (40.3%, 39.13%, and 34.54%, respectively). The winner, Easablade, generated fewer objects than the opponents and nice looking layouts produced by the cellular automata, which likely is the main reason behind its victory. Most of the generated levels by Easablade, however, were either unbeatable or easy compared to the other generators.

VIII. Methods for Rule Generation

This section describes the different algorithms that are included in the framework or have been found in the literature [72] toward generating rules for the GVG-AI framework.

A. Constructive Methods

Constructive methods are algorithms that generate the rules in one pass without the need to play the game. The constructive methods often incorporate knowledge about game design to generate more interesting games.

1) Sample Random Generator: This is the simplest generator provided with the framework. The main idea is to generate a game that compiles with no errors. For example, the game should not contain interactions such as killing the end of screen (EOS) sprite. The algorithm starts by generating a random number of interactions by selecting two random sprites (including EOS) and a random interaction rule one by one. The algorithm checks that every interaction is valid before adding it to the generated game. After generating the random interactions, the algorithm generates two termination conditions, one for winning and the other for losing. The losing condition is fixed to the avatar being killed, while the winning is either winning the game after a random amount of frames or winning the game when certain sprite count reaches zero.

2) Sample Constructive Generator: This is a more complex generator that utilizes knowledge about VGDL language and level design to generate more interesting games. The algorithm starts by classifying the game sprites into different categories, such as wall sprites (those that surround the level), collectible/harmful sprites (immovable sprites that cover around 10% of the level), spawner sprites (sprites that spawn another), etc. For each type of sprite, the algorithm has rules to generate interactions based on them. For example, harmful sprites kill the avatar on collision, wall sprites either prevent any movable object from passing through or kill the movable object upon collision, etc. For more details about the rules, the reader is referred to [12]. After the game interactions are generated, two termination conditions are generated, one for winning and the other for losing. The losing condition is fixed to the avatar's death, while the winning condition depends on the current sprites. For example: If collectible sprites exist in the current definition, the winning condition is set to collect them all.

B. Search-Based Methods

Search-based methods use a search-based algorithm to find a game based on certain criteria that ensure the generated game have better rules than just randomly choosing them.

1) Sample Genetic Generator: Similar to the level generation track, the search-based algorithm uses FI2Pop to evolve new games. As discussed before, FI2Pop keeps two populations, one for feasible games and the other for infeasible games. The infeasible games try to become feasible by satisfying multiple constraints such as minimizing the number of bad frames (frames contains sprites outside the level boundaries) under certain threshold, the avatar does not die in the first 40 frames, etc. However, the feasible chromosomes try to maximize its fitness. The fitness consists of two parts; the first part is to maximize the difference in performance between the OLETS and MCTS agents, and the difference between MCTS and random agent. The second part is to maximize the number of interaction rules that fires during the simulation of the generated game.

2) Thorbjrn Generator: This generator [72] is similar to the sample genetic generator. It tries to maximize the difference between the performance of different algorithms. This generator uses evolutionary strategies with mutation and crossover operators to generate an entire game instead of an interaction set and termination conditions.

C. Discussion

Similar to the level generators, the difference between the different generators is the time used in creation and the features in the output game. The constructive methods take less time but do not guarantee different games or playability, while the search-based generators take long time to generate one game, attempting to satisfy the playability constraints using automated playing agents. *Thorbjorn* is the only generator that creates the whole game, not only the interaction rules and termination conditions, which makes it harder to compare to the rest of the generators.

The remaining ones are the three sample generators that come with the framework, which are compared to one another by doing a user study on the generated games [12]. The generators are used to generate three new games for three different levels (Aliens, Boulderdash, and Solarfox). The participants in the study were subjected to two generated games by two randomly selected generators and asked to pick the one they prefer. The constructive generator was the preferred one (chosen 76.38% of the time), followed by the genetic (44.73%), and random

(24.07%) generators. An explanation for the low preference shown for the genetic generator could be its fitness function: It incorporates a constraint that tries to make sure that the game sprites are always in the playing area. This constraint caused the GA in the current allocated time to favor games that limit considerably the movement of the sprites.

IX. Research That Builds on GVGAI

A. Learn the Domain Knowledge

Besides the work relevant to the learning competition, there are some other research works around reinforcement learning using the GVGAI framework. Narasimhan *et al.* [73] combined a differentiable planning module and a model-free component to a two-part representation, obtained by mapping the collected annotations for game playings to the transitions and rewards, to speed up the learning. The proposed approach has been tested on four GVGAI single-player games and shown its effectiveness on both transfer and multitask scenarios on the tested games. The GVGAI learning competition proposes to use a screenshot of the game screen (at pixel level) at every game tick to represent the current game state. Instead of directly using the screenshot, Woof and Chen [74] used an object embedding network (OEN), which extracted the objects in the game state and compressed object feature vectors (e.g., position, distance to the nearest sprite, etc.) into one single fixed-length feature vector. The DRL agent based on OEN has been evaluated on five of the GVGAI single-player games and showed various performance levels on the tested games [74].

B. AI-Assisted Game Design

Machado *et al.* [75] implemented a recommender system based on the VGDL to recommend game elements, such as sprites and mechanics. Then, the recommender system was expanded to *Cicero* [76], [77], an AI-assisted game design, and debugging tool built on top of the GVGAI. *Cicero* has a statistics tool of the interactions to help figure out the unused game rules, a visualization system to illustrate the information about game objects and events, a mechanics recommender, a query system [78] for in-game data, a playtrace aggregator, a heatmap-based game analysis system, and a retrospective analysis application *SeekWhence* [79]. The gameplay sessions by human players or AI agents can be recorded, and every single frame at every game tick can be easily extracted for further study and analysis.

Recently, Liu *et al.* [80] applied a simple random mutation hill climber (RMHC) and a multiarmed bandit RMHC together with resampling methods to tune game parameters automatically. Games instances with significant skill-depth have been evolved using GVGAI agents. Furthermore, Kunanusont *et al.* [51] evolved simultaneously the GVGAI agents as part of the game (opponent models).

Guerrero *et al.* [81] explored 5 GVGAI agents using 4 different heuristics separately on playing 20 GVGAI games, allowing different behaviors according to the diverse scenarios presented in the games. In particular, the aforementioned work explored heuristics that were not focused on winning the game, but to explore the level or interact with the different sprites of the games. These agents can be used to evaluate generated games, thus help evolve them with preferences to particular behaviors.

Khalifa *et al.* [82] modified MCTS agents by editing the UCT formula used in the agent. Human playing data have been used for modeling to make the modified agents playing in a human-like way. Primary results showed that one of the studied agents achieved a similar distribution of repeated actions to the one by human players. The work was then extended by Bravi *et al.* [49], in which game-play data have been used to evolve effective UCT alternatives for a specific game. The MCTS agents using new formulas, with none or limited domain information, are compared to a standard implementation of MCTS (the *sampleMCTS* agent of GVGAI) on the game *Missile Command*. Applying the UCT alternatives evolved using game-playing data to a standard MCTS significantly improved its performance.

Besides designing games and the agents used in them, the automatic generation of video game tutorials (aimed at helping players understand how to play a game) is also an interesting subfield of study. Green *et al.* [83] pointed out that the GVGAI Framework provides an easy testbed for tutorial generation. The game rules in GVGAI are defined in VGDL; therefore, the tutorial generation can be easily achieved by reading and translating VGDL files. Furthermore, Green *et al.* [84] build a system (AtDelfi) that generates tutorials using the VGDL file and automated AI agents. AtDelfi reads the VGDL file and builds a graph of interactions between the game sprites. AtDelfi analyzes the graph to identify the winning path (sequence of nodes starting from player sprite that leads to the winning condition in the graph), losing paths (sequence of nodes starting from the losing condition till there is no dependence), and score path (sequence of nodes starting from player sprite that leads to score change in the graph). These paths are represented as text and videos that explain to the user how to play the game. The text is generated using a string replacement method to generate a human readable instructions, while the videos are recorded using a group of automated agents that won the GVGP Competition [8] and record every group of frames that cause one of the interactions on the path to trigger.

A more recent work by Anderson *et al.* [85] focused on designing deceptive games to deceive AI agents and lead the agents away from a globally optimal policy. Designing such games helps understand the capabilities and weaknesses of existing AI agents and can serve as a preparation step for designing a meta-agent for GVGP, which combines the advantages of different agents. The authors categorized the deceptions and imported various types of deception to the existing GVGAI games by editing the corresponding VGDL files. The agents submitted to the GVGAI single-player planing competition have been tested on the new games. Interestingly, the final ranking of the agents on each of the games differed significantly from the rankings in the GVGAI competition. The new designed deceptive games successfully explored the weaknesses of agents that have performed well on the test set of the official competition.

Finally, C. Guerrero-Romero *et al.* in a vision paper [86], proposed a methodology that consists of the use of a team of general AI agents with differentiated skill levels and goals (winning, exploring, eliminating sprites, collecting items, etc.). The methodology is aimed at aiding game design by analyzing the performance of this team of agents as a whole and the provision of logged and visual information that shows the agent experience through the game.

C. Game Generation With RAPP

Nielsen *et al.* [87] proposed relative algorithm performance profile (RAPP) as a measure of relative performance of agents and tested their approach on different general game-playing AI agents using GVGAI framework. The authors showed that well-designed games have clear skill-depth, thus being able to distinct good or bad players. In other words, a strong agent or human player should perform significantly better than a weak agent or human player over multiple playings on well-designed games. For instance, a skillful agent is expected to perform better than a random agent, or one that does not move.

Then, Nielsen *et al.* [72] integrated the differences of average game scores and win rate between any agent and a random agent to the evaluation of new games, either randomly generated or generated by editing existing GVGAI games. Although most of the resulted games are interesting to play, yet there are some exceptions, in which the core challenge of the game has been removed. For instance, the enemy cannot heart the player, which makes it no more an enemy. But, it still provides useful starting points for human designers.

Kunanusont *et al.* [51] extended the idea of RAPP. Five GVGAI agents and a deterministic agent designed for the tested Space Battle Game are used as the candidate opponent, which is considered as part of the game to be evolved. Two GVGAI agents, one step look ahead (weak), MCTS (strong), and the deterministic agent (mediocre), are used to play multiple times the evolved game for evaluation. The evaluation function is defined as the minimum of the difference of game scores between the strong and mediocre agents, and the difference of game scores between the mediocre and weak agents, aiming at generating games that can clearly distinguish stronger agents and weak agents.

Recently, Kunanusont *et al.* [88] used the NTBEA to evolve game parameters in order to model player experience within the game. The authors were able to find parameterizations of three games that, when played by MCTS and RHEA agents, produce predefined and different score trends.

D. Robustness Testing

Perez-Liebana *et al.* [89] ran a study on the winners of the 2014 and 2015 editions of the single-player planning competition in order to analyze how robust they were to changes in the environment with regard to actions and rewards. The aim of the aforementioned work was to analyze a different type of generality: Controllers for this framework are developed to play in multiple games under certain conditions, but the authors investigated that that could be the effect of breaking those compromises: an inaccurate forward model, an agent that does not execute the move decided by the algorithm or score penalties incurred by performing certain actions.

An interesting conclusion on this study is that, once the conditions have been altered, sample agents climb up to the top of the rankings and that the good controllers behave worse. Agents that rely on BFS or A* (such as *YOLOBOT* or *Return42*, already described in this paper) handled noise very badly. MCTS also showed to be quite robust in this regard, above other rolling horizon agents that could not cope so well with these changes. This paper also reinforced the idea that the GVGAI framework and competition are also robust. Despite the changes in the performance of the agents, some controllers do better than others under practically all conditions. The opposite (rankings depending only on noise factors, for instance) would mean that the framework is fragile.

More recently, Stephenson *et al.* [90] have pointed out that the selection of a proper subset of games for comparing a new algorithm with others is critical, as using a nonsuitable representative subset may have a bias to some algorithms. More general, the questions is, given a set of sample problems, how to sample a subset as fair as possible for the algorithms to be tested, and to avoid the bias to any of the algorithms. The authors use an information-theoretic method in conjunction with game playing data to assist in the selection of GVGAI games. Games with higher information gains are used for testing a new agent.

X. Discussion and Open Research Problems on Single- and Two-Player Planning

The single- and two-player planning versions of GVGAI are the ones that have received most attention and research. Despite their popularity and efforts, the best approaches rarely surpass an approximately 50% victory rate in competition game sets, with very low victory rate in a great number of games. Similarly, different MCTS and RHEA variants (including many of the enhancements studied in the literature) struggle to achieve a higher than 25% victory rate in all (more than a hundred) single-player games of the framework. Therefore, increasing performance in a great proportion of games is probably the most challenging problem at the moment.

Literature shows multiple enhancements on algorithms and methods aiming to improve this performance; but, in the vast majority of cases, the improvements only affect a subset of games or certain configurations of the algorithms. While this is understandable due to the nature of GVGP, it also shows that the current approaches do not work in order to reach truly general approaches that work across board.

The work described in this survey has shown, however, interesting insights that can point us in the right direction. For instance, several studies show that using more sophisticated (i.e., with A* or other methods such as potential fields) distances to sprites as features works better than Euclidean distances. The downside is that computing these measurements take an important part of the decision time budget, which cannot be used in case it is needed for some games or states where the best action to take is not straightforward.

In general, one could say that one of the main points to address is how to use the decision time more wisely. Some approaches tried to make every use of the forward model count, like those agents that attempt to learn facts about the game during the roll-outs of MCTS. Again, some attempts in this direction have provided marginal improvements; but, the problem may be trying to design a general feature *extractor*. In other words, what we try to learn is influenced by what we know about existing games (i.e., some sprites are good, others are bad, some spawn other entities and there are sprites—resources—that the avatar can collect). Some games may require features that have not been thought of, especially because the challenge itself presents games that have not been seen before.

Another improvement that has been tried in several studies is the use of macroactions (in most cases, a repetition of an action during several consecutive steps) to: 1) make the action space coarser; and 2) make a better use of the time budget. Again, these modifications have improved performance in certain games (including some that had not been won by any algorithm previously), but they either did not have an impact on others, or they made the performance worse. It is likely that different games can benefit from different macroaction lengths (so, work could be done on trying to automatically and dynamically adapt the number of times the action is repeated) but also of more complex structures that allow for high-level planning. In fact, games that require high-level planning are still an open problem to be solved in this setting.

Games classification and the use of hyperheuristics are also an interesting area for research. Some of the best approaches up to date, as YOLOBOT, do make a differentiation between stochastic and deterministic games to later use one or another algorithm. An open challenge is how to make this classification more accurate and detailed, so an approach could count on a portfolio of (more than two) algorithms that adapt to every game. Attempts have been made to classify with game features, but results suggest that these classifications and the algorithms used are not strong enough. Devising more general features for this, maybe focused on the agent game-play experience rather than game features, is a line of future research.

All these unsolved issues apply to both single- and two-player settings; although the latter case adds the difficulty of having an opponent to compete or collaborate with. There are two open problems that arise from this: First, no study has been made that tries to identify the game and behavior of the opponent as collaborative or competitive. Analysis of the other player's intentions can be seen as a subfield on its own; only that in this case we add the general game playing component to it. Second, some advancements have been done in using opponent models that go beyond random, but investigation in more complicated opponent models that better capture and learn the behavior of the other player could potentially yield better results.

Besides the development of agents for game playing, AI-assisted game design, automatic game testing, and game debugging using GVGAI agents have attracted researchers' attention. Some work around evolving game skill-depth using relative performance between GVGAI agents have been done recently, and most of this work has been focused on RAPP, where *performance* is measured in terms of how well the agents play the given games. However, it is sensible to explore other aspects of agent game-play to influence game design. Factors like the amount of level explored by different agents (so a generator favors those levels or games that allow for a wider exploration, or maybe a progressive one), their decisiveness [91] on selecting the best action to take, or the entropy of their moves, can also be used to this end.

XI. Educational use of GVGAI

The GVGAI framework has been used to provide engaging assignments for taught modules and as the basis for many M.Sc. dissertation projects. The descriptions below give an idea of the educational uses of GVGAI but are not intended to be an exhaustive list.

A. Taught Modules

GVGAI has been used in at least two distinct ways within taught modules. The most typical way is to use design specific aspects of the course around the framework, and teaching the students about the main concepts of GVGAI with examples of how to write agents for the selected tracks. This is, then, followed up with an assignment, where a significant weight is given to how well each student or group's entry performs in the league. Several institutions have run private leagues for this, including Otto von Guericke Universität Magdeburg, University of Essex, University of Muenster, Universidad Carlos III de Madrid, Universidad de Malaga, and New York University. Running a private league means the course supervisor has full control over the setup of the league, including when students can enter and how thoroughly the entries are evaluated, and the set of games to evaluate them on. For the two-player track, this also allows control over the opponents chosen. The Southern University of Science and Technology and the Nanjing University have also used GVGAI framework in their AI modules, without running a private league, as assignments when teaching search or reinforcement learning methods.

Another use-case in taught modules is to teach the VGDL part of framework, then set the development of novel and interesting games as the assignment. This was done to good effect at IT University of Copenhagen, where the students produced a number of challenging puzzle games that were later used in the training and validation sets of the planning track. A similar approach was taken in a module on AI-assisted game design at the University of Essex, where planning track games were also produced.

B. M.Sc. Dissertation Projects

GVGAI offers an extensive range of interesting research challenges, some of which have been addressed in M.Sc. dissertation projects. The majority of the ones we are aware of have focused on the single-player planning track, but this is not surprising as it was the first track to be developed. The single-player planning track also has the benefit of providing some good sample agents as starting points for further work, either in the sense of extending the sample agents to achieve higher performance, or using the sample agents as a useful source of comparison. A good example is the work on MCTS with options, in which options refer to action sequences designed for specific subgoals. The version with options significantly outperformed the sample MCTS agent on most of the games studied: As with many cases what began as an M.Sc. thesis was later published as a conference paper [29]. In our experience, this usually provides an excellent educational experience for the student. Other planning track thesis include [21], the real-time enhancements of [22], knowledge-based variants [46], and goal-oriented approaches [53].

Beyond the planning tracks, other examples (already described in this survey) include applying ASP [92] or GAs [69] to the level generation track and learning from game screen capture [63]. Moreover, [63] was essentially a learning track approach before the learning track was running. Finally, another approach is to extend the framework in some way, such as developing the two-player learning track [93].

XII. Future Directions

The GVGAI framework and competition are in constant development. The opportunities that this benchmark provides for different lines of research and education are varied, and this section outlines the future directions planned ahead for the following years.

A. New Tracks

As new challenges are proposed, the possibility of organizing them as competition tracks arise. Below are listed some possible new tracks that can attract interesting research areas.

1) Automatic Game Design: The game design involves, but is not limited to, game generation, level generation, rule generation, and play-testing (playing experience, game feeling, fun, etc.), study of game market, user interface design, and audio design. The automatic game design becomes an active research topic since the late 2000s. A review of the state of the art in automatic game design can be found in [80].

A *Game Generation track* would aim at providing AI controllers, which automatically generate totally new games or game instances by varying the game parameters, i.e., parameter tuning. How to achieve the former is an open question. The straightforward way would be providing a particular theme, a database of game objects, or searching spaces of game rules, with which the participants can generate new games. The ideal case would be that the controllers automatically create totally new games from nothing. Although there is a yawning gulf between aspiration and reality, yet an interdisciplinary field combining automatic game design and domain-specific automatic programming is expected. The latter, automatic game tuning, is relatively easier. Some search-based and population-based methods have been applied to game parameter optimization aiming at maximizing the depth of game variants [80] or finding more playable games.

2) Multiplayer GVGAI: Multiagent games have drawn people's attention, for instance, real time strategy games (e.g., StarCraft) and board games (e.g., Mahjong). The study of multiagent GVGAI is a fruitful research topic. Atari games can also be extended to multiagent games. In particular, the Pac-Man can be seen as a multiagent game, and related competitions have been held since 2011. The most recent Ms Pac-Man versus Ghost Team Competition [1], which included partial observability, was held at the CIG, in 2016. Nevertheless, a more general multiagent track is favorable.

The interface of the two-player planning track was initially developed for two or more players, so it has the potential to be expanded to a multiplayer planning track, in which an agent is allowed to control more than one player, or each of the players is controlled by a separate agent. This future track can be expanded again as a multiagent learning framework, providing a two-or-more-player learning track.

3) Turing Test GVGAI: Determining if an agent that is playing a game is a human or a bot is a challenge that has been subject of study for many years [1], and the idea of applying it to a general video game setting is not new [94]. This offers an interesting opportunity to extend the framework to having a Turing test track where participants create AI agents that play like humans for any game that is given. Albeit the understandable difficulty of this problem, the interest for research in this area is significant: What are the features that can make an agent play like a human in any game?

B. General Directions

There are several improvements and additions to the framework that can be done and would potentially affect all existent and future competition tracks. One of these continuous modifications is the constant enlargement of the games library. Not only new games are added for each new edition of the competition, but also the work done on automatic game design using the GVGAI framework has the potential to create infinite number of games that can be integrated into the framework.

Adding more games can also be complemented with compatibility with other systems. Other general frameworks like OpenAI Gym [95], arcade learning environment (ALE) [96] or Microsoft Malmö [97] count on a great number of single- or multi-player, model-free or model-based tasks. Interfacing with these systems would greatly increase the number of available games, which all GVGAI agents could play via a common API. This would also open the framework to 3-D games, an important section of the environments the current benchmark does not cover.

With regard to the agents, another possibility is to provide them with a wider range of available actions. For instance, the player could be able to apply more than one action simultaneously, or these actions could form a continuous action space (i.e., pressing a throttle in a range between 0 and 1). This would enhance the number of legal combinations for the agent to choose from at each decision step.

Besides the framework itself, the website for GVGAI could also be improved to provide better and faster feedback to the competition participants. More data analysis features can be added, such as visualization of the score changes during the game playing, the action entropy, and the exploration of the game world (heat-map of visited positions). A related work is to provide better and more flexible support for game play metric logging, better support for data mining of results together with visualization, and better data saving, which will help enabling to upload replays (i.e., action logs) from AI agents and human play-throughs.

Another envisaged feature is being able to play the game in a web browser (without any download or installation) by an AI agent or human, and visualize the analyzed features during the game playing in real time. A bonus will be the easy parameterization options for games; thus, a player or an AI agent can easily set up the parameters and rules to define the desired game by inserting values directly or generate pseudorandomly a level to play using some preimplemented automatic game tuning techniques, given some particular goals or features.

XIII. Conclusion

The GVGAI framework offers the most comprehensive system to date for evaluating the performance of GVGP agents, and for testing general purpose algorithms for creating new games and creating new content for novel games. The framework has been used in multiple international competitions and has been used to evaluate the performance of hundreds of general video game agents.

The agent tracks cater for planning agents able to exploit a fast forward model and for learning agents that must learn to react sensibly without the benefits of a forward model. The planning track already comes in single- and two-player versions; the learning track is currently single-player only, but with a two-player version envisaged. Although long-term learning may also be used within the planning track, yet the best-performing agents have, as far as we know, not yet done this. Recent successes in Go indicate what can be achieved by combining learning and planning, so applying a similar system within GVGAI is an interesting prospect. In fact, the combination of different approaches into one is an interesting avenue of future research. An example is the work described in this survey, which mixes learning and procedural level generation [67], but one could imagine further synergies such as content generation and learning for two-player games.

The main alternative to GVGAI is the ALE [96]. At the time of writing, ALE offers higher quality games than those by GVGAI, as they were home-console commercial games of a few decades ago. In GVGAI terms, ALE offers just two tracks: single-player learning and planning, with the learning track being the more widely used. For future work on machine learning in video games, we predict that the two-player tracks will become the most important, as they offer open-ended challenges based on an arms race of intelligence as new players are developed, and are also outside of the current scope of ALE. Although ALE has had so far a greater uptake within some sectors of the machine learning community, yet GVGAI benefits from being much more easily extensible than ALE: It is easy to create new VGDL games, easy to create new levels for these games, and easy to create level generators for them as well. It is also easy to automatically generate variations on existing VGDL games and their levels. This allows for training on arbitrarily large sets of game variations and level variations. In contrast, agents trained on ALE games run a serious risk of overfitting to the game and level they are trained on. An immediate priority is to test the rich set of ALE agents on the equivalent GVGAI-tracks to gain a sense of the relative difficulty of each environment and to learn more of the relative challenges offered by each.

The content creation tracks offer an extremely hard challenge: creating rules or levels for unseen games. Promising directions include the further development and exploitation of a range of general game evaluation measures [91], and greater use of the best GVGAI agents to perform the play-testing of the novel rules and levels.

The VGDL has been an important part of GVGAI to date, since it makes it possible to rapidly and concisely specify new games. However, it is also a source of limitation, as its limited expressiveness makes it hard to make games that are fun for humans to play. VGDL also limits the ease with which complex game mechanics can be embedded in games, which, in turn, limits the depth of challenge that can be posed for the GVGAI agents. Hence, an important future direction is the authoring of GVGAI-compatible games in any suitable language that conforms to the necessary GVGAI API in order to ensure compatibility with the desired GVGAI track.

Finally, while the above discussion provides a compelling case for the future of GVGAI as a tool for academic study, we also believe that when it reaches a higher level of maturity it will provide an important tool for game designers. The vision is to provide an army of intelligent agents with a range of playtesting abilities, and a diverse set of metrics with which one can analyze a range of important functional aspects of a game.

Acknowledgment

The authors would like to thank the participants of all tracks of the competition for their work and submitted controllers and generators.

References

[1] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. New York, NY, USA: Springer-Verlag, 2018.
[2] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Kuala Lumpur, Malaysia: Pearson Education Limited, 2016.
[3] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General video game AI: Competition, challenges and opportunities," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 4335–4337.
[4] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius, "Towards a video game description language," *Dagstuhl Follow-Ups*, vol. 6, pp. 85–100, 2013.
[5] J. Levine *et al.*, "General video game playing," *Dagstuhl Follow-Ups*, vol. 6, pp. 77–84, 2013.
[6] T. Schaul, "A video game description language for model-based or interactive learning," in *Proc. IEEE Conf. Comput. Intell. Games*, 2013, pp. 1–8.
[7] T. Schaul, "An extensible description language for video games," *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 4, pp. 325–331, Dec. 2014.
[8] D. Perez *et al.*, "The 2014 general video game playing competition," *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 3, pp. 229–243, Sep. 2016.
[9] R. D. Gaina, D. Perez-Liebana, and S. M. Lucas, "General video game for 2 players: Framework and competition," in *Proc. 8th Comput. Sci. Electron. Eng.*, Sep. 2016, pp. 186–191.
[10] R. D. Gaina *et al.*, "The 2016 two-player GVGAI competition," *IEEE Trans. Games*, vol. 10, no. 2, pp. 209–220, Jun. 2018.
[11] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius, "General video game level generation," in *Proc. Annu. Conf. Genetic Evol. Comput. Conf.*, ACM, 2016, pp. 253–259.

[12] A. Khalifa, M. C. Green, D. Pérez-Liébana, and J. Togelius, "General video game rule generation," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 170–177.
[13] D. Pérez-Liébana, M. Stephenson, R. D. Gaina, J. Renz, and S. M. Lucas, "Introducing real world physics and macro-actions to general video game AI," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 248–255.
[14] J. Liu, D. Perez-Liebana, and S. M. Lucas, "The single-player GVGAI learning framework—technical manual," 2017. [Online]. Available: http://www.liujialin.tech/publications/GVGAISingleLearning_manual.pdf
[15] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep reinforcement learning in the general video game AI framework," in *Proc. Conf. Computational Intell. Games*, 2018, pp. 1–8.
[16] D. Ashlock, D. Pérez-Liébana, and A. Saunders, "General video game playing escapes the no free lunch theorem," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 17–24.
[17] T. Joppen, M. U. Moneke, N. Schröder, C. Wirth, and J. Fürnkranz, "Informed hybrid game tree search for general video game playing," *IEEE Trans. Games*, vol. 10, no. 1, pp. 78–90, Mar. 2018.
[18] T. Geffner and H. Geffner, "Width-based planning for general video-game playing," in *Proc. 11th Artif. Intell. Interactive Digital Entertainment Conf.*, 2015, pp. 23–29.
[19] D. Soemers, C. Sironi, T. Schuster, and M. Winands, "Enhancements for real-time monte-carlo tree search in general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games.*, 2016, pp. 1–8.
[20] J. Liu. "GVGAI single-player learning competition at IEEE CIG17." [Online]. Available: https://www.slideshare.net/ljialin126/gvgai-singleplayer-learning-compe tition-at-ieee-cig17
[21] T. Schuster, "MCTS based agent for general video games," Master's thesis, Dept. Data Sci. Knowl. Eng., Maastricht Univ., Maastricht, The Netherlands, 2015.
[22] D. Soemers, "Enhancements for real-time Monte-Carlo tree search in general video game playing," Master's thesis, Dept. Data Sci. Knowl. Eng., Maastricht Univ., Maastricht, The Netherlands, 2016.
[23] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary MCTS for general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2014, pp. 1–8.
[24] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius, "Investigating MCTS modifications in general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 107–113.
[25] C. Y. Chu, T. Harada, and R. Thawonmas, "Biasing Monte-Carlo rollouts with potential field in general video game playing," in *Proc. IPSJ Kansai-Branch Conv.*, 2015, pp. 1–6.
[26] C. Y. Chu, H. Hashizume, Z. Guo, T. Harada, and R. Thawonmas, "Combining pathfmding algorithm with knowledge-based Monte-Carlo tree search in general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 523–529.
[27] H. Park and K.-J. Kim, "MCTS with influence map for general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 534–535.
[28] E. H. dos Santos and H. S. Bernardino, "Redundant action avoidance and non-defeat policy in the Monte Carlo tree search algorithm for general video game playing," in *Proc. SBC—SBGames*, 2017, doi: 10.1109/SBGAMES.2018.00013.
[29] M. de Waard, D. M. Roijers, and S. C. J. Bakkes, "Monte Carlo tree search with options for general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 47–54.
[30] D. Perez-Liebana, S. Mostaghim, and S. M. Lucas, "Multi-objective tree search approaches for general video game playing," in *Proc. IEEE Congr. Evol. Comput.*, 2016, pp. 624–631.
[31] A. Khalifa, M. Preuss, and J. Togelius, "Multi-objective adaptation of a parameterized GVGAI agent towards several games," in *International Conference on Evolutionary Multi-Criterion Optimization*. New York, NY, USA: Springer-Verlag, 2017, pp. 359–374.
[32] I. Bravi, A. Khalifa, C. Holmgård, and J. Togelius, "Evolving UCT alternatives for general video game playing," in *Proc. IJCAI-16 Workshop Gen. Game Playing*, 2016, pp. 63–70.
[33] A. Babadi, B. Omoomi, and G. Kendall, "EnHiC: An enforced hill climbing based system for general game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 193–199.
[34] M. J. Nelson, "Investigating vanilla MCTS scaling on the GVG-AI game corpus," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 402–408.
[35] I. Bravi, D. Perez-Liebana, S. M. Lucas, and J. Liu, "Shallow decision-making analysis in general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games.*, 2018, pp. 1–8.
[36] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, "Analysis of vanilla rolling horizon evolution parameters in general video game playing," in *European Conference on the Applications of Evolutionary Computation*. New York, NY, USA: Springer-Verlag, 2017, pp. 418–434.
[37] B. S. Santos, H. S. Bernardino, and E. Hauck, "An improved rolling horizon evolution algorithm with shift buffer for general game playing," in *Proc. Brazilian Symp. Comput. Games Digital Entertainment*, 2018, pp. 31–37.
[38] B. S. Santos and H. S. Bernardino, "Game state evaluation heuristics in general video game playing," in *Proc. Brazilian Symp. Comput. Games Digital Entertainment*, 2018, pp. 147–156.
[39] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Tackling sparse rewards in real-time games with statistical forward planning methods," in *Proc. AAAI Conf. Artif. Intell.*, 2019, to be published.
[40] B. Jia, M. Ebner, and C. Schack, "A GP-based video game player," in *Proc. Annu. Conf. Genetic Evol. Comput.*, ACM, 2015, pp. 1047–1053.
[41] B. Jia and M. Ebner, "A strongly typed GP-based video game player," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 299–305.
[42] R. D. Gaina, S. M. Lucas, and D. Pérez-Liébana, "Population seeding techniques for rolling horizon evolution in general video game playing," in *Proc. IEEE Congr. Evol. Comput.*, 2017, pp. 1956–1963.
[43] R. D. Gaina, S. M. Lucas, and D. Pérez-Liébana, "Rolling horizon evolution enhancements in general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 88–95.
[44] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open loop search for general video game playing," in *Proc. Annu. Conf. Genetic Evol. Comput.*, ACM, 2015, pp. 337–344.
[45] H. Horn, V. Volz, D. Pérez-Liébana, and M. Preuss, "MCTS/EA hybrid GVGAI players and game difficulty estimation," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8.
[46] J. van Eeden, "Analysing and improving the knowledge-based fast evolutionary MCTS algorithm," Master's thesis, Faculty Sci., Dept. Inf. Comput. Sci., Utrecht University, Utrecht, The Netherlands, 2015.
[47] C.-Y. Chu, S. Ito, T. Harada, and R. Thawonmas, "Position-based reinforcement learning biased MCTS for general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 444–451.
[48] E. Ilhan and A. Ş. Uyar, "Monte Carlo tree search with temporal-difference learning for general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 317–324.
[49] I. Bravi, A. Khalifa, C. Holmgård, and J. Togelius, "Evolving game-specific UCB alternatives for general video game playing," in *European Conference on the Applications of Evolutionary Computation*. New York, NY, USA: Springer-Verlag, 2017, pp. 393–406.
[50] C. F. Sironi *et al.*, "Self-adaptive MCTS for general video game playing," in *European Conference on the Applications of Evolutionary Computation*. New York, NY, USA: Springer-Verlag, 2018.
[51] K. Kunanusont, R. D. Gaina, J. Liu, D. Perez-Liebana, and S. M. Lucas, "The n-tuple bandit evolutionary algorithm for automatic game improvement," in *Proc. IEEE Congr. Evol. Comput.*, 2017, pp. 2201–2208.
[52] C. F. Sironi and M. H. M. Winands, "Analysis of self-adaptive Monte Carlo tree search in general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–4.
[53] B. Ross, "General Video Game Playing with Goal Orientation," Master's thesis, Dept. Comput. Inf. Sci., Univ. Strathclyde, Glasgow, Scotland, 2014.
[54] I. Azaria, A. Elyasaf, and M. Sipper, *Evolving Artificial General Intelligence for Video Game Controllers*. New York, NY, USA: Springer-Verlag, 2018, pp. 53–63. [Online]. Available: https://doi.org/10.1007/978-3-319-97088-2_4
[55] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching games and algorithms for general video game playing," in *Proc. Artif. Intell. Interactive Digital Entertainment Conf.*, 2016, pp. 122–128.
[56] A. Mendes, J. Togelius, and A. Nealen, "Hyperheuristic general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8.
[57] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "General win prediction from agent experience," in *Proc. IEEE Conf. Comput. Intell. Games.*, 2018, pp. 1–8.
[58] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "VERTIGO: Visualisation of rolling horizon evolutionary algorithms in GVGAI," in *Proc. 14th AAAI Conf. Artif. Intell. Interactive Digital Entertainment*, 2018, pp. 265–267.
[59] J. M. Gonzalez-Castro and D. Perez-Liebana, "Opponent models comparison for 2 players in GVGAI competitions," in *Proc. 9th Comput. Sci. Electron. Eng.*, 2017, pp. 151–156.

[60] D. Ashlock, E.-Y. M. Kim, and D. Perez-Liebana, "Toward general mathematical game playing agents," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–7.

[61] S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, "Neuroevolution for general video game playing," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 200–207.

[62] A. Braylan and R. Miikkulainen, "Object-model transfer in the general video game domain," in *Proc. 12th AAAI Conf. Artif. Intell. Interactive Digital Entertainment.*, 2016, pp. 136–142.

[63] K. Kunanusont, "General video game artificial intelligence: Learning from screen capture," Master's thesis, Dept. Sch. Comput. Sci. Electron. Eng., University of Essex, Colchester, U.K., 2016.

[64] K. Kunanusont, S. M. Lucas, and D. Pérez-Liébana, "General video game AI: Learning from screen capture," in *Proc. IEEE Congr. Evol. Comput.*, 2017, pp. 2078–2085.

[65] D. Apeldoorn and G. Kern-Isberner, "An agent-based learning approach for finding and exploiting heuristics in unknown environments," in *Proc. COMMONSENSE*, 2017. [Online]. Available: http://ceur-ws.org/Vol-2052/

[66] A. Dockhorn and D. Apeldoorn, "Forward model approximation for general video game learning," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–8.

[67] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, "Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation," 2018, arXiv:1806.10729.

[68] S. Beaupre, T. Wiles, S. Briggs, and G. Smith, "A design pattern approach for multi-game level generation," in *Proc. 14th AAAI Conf. Artif. Intell. Interactive Digital Entertainment*, 2018, pp. 145–151.

[69] J. Nichols, "The use of genetic algorithms in automatic level generation," Master's thesis, Dept. Sch. Comput. Sci. Electron. Eng., Univ. Essex, Colchester, U.K., 2016.

[70] M. Sharif, A. Zafar, and U. Muhammad, "Design patterns and general video game level generation," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 9, pp. 393–398, 2017.

[71] X. Neufeld, S. Mostaghim, and D. Perez-Liebana, "Procedural level generation with answer set programming for general video game playing," in *Proc. 7th Comput. Sci. Electron. Eng.Conf.*, 2015, pp. 207–212.

[72] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "Towards generating arcade game rules with VGDL," in *Proc. IEEE Conf. Comput. Intell. Games.*, 2015, pp. 185–192.

[73] K. Narasimhan, R. Barzilay, and T. S. Jaakkola, "Deep transfer in reinforcement learning by language grounding," *CoRR*, vol. abs/1708.00133, pp. 849–874, 2017.

[74] W. Woof and K. Chen, "Learning to play general video-games via an object embedding network," in *Proc. IEEE Conf. Comput. Intell. Games.*, 2018, pp. 1–8.

[75] T. Machado, I. Bravi, Z. Wang, A. Nealen, and J. Togelius, "Shopping for game mechanics," in *Proc. FDG Workshop Procedural Content Gener.*, 2016, pp. 1–9.

[76] T. Machado, A. Nealen, and J. Togelius, "Cicero: A mixed-initiative AI-assisted game design tool," in *Proc. 12th Int. Conf. Found. Digital Games*, ser. FDG'17., New York, NY, USA: ACM, 2017, pp. 1–6.

[77] T. Machado, D. Gopstein, A. Nealen, O. Nov, and J. Togelius, "AI-assisted game debugging with cicero," in *Proc. IEEE Congr. Evol. Comput.*, 2018, pp. 1–8.

[78] T. Machado, D. Gopstein, A. Nealen, and J. Togelius, "Kwiri—what, when, where and Who: Everything you ever wanted to know about your game but didn't know how to ask," in *Proc. Knowl. Extraction From Games Workshop.*, AAAI, 2019, to be published.

[79] T. Machado, A. Nealen, and J. Togelius, "SeekWhence a retrospective analysis tool for general game design," in *Proc. 12th Int. Conf. Found. Digital Games*, ser. FDG '17., New York, NY, USA: ACM, 2017, pp. 4:1–4:6. [Online]. Available: http://doi.acm.org/10.1145/3102071.3102090

[80] J. Liu, J. Togelius, D. Pérez-Liébana, and S. M. Lucas, "Evolving game skill-depth using general video game AI agents," in *Proc. IEEE Congr. Evol. Comput.*, 2017, pp. 2299–2307.

[81] C. Guerrero-Romero, A. Louis, and D. Pérez-Liébana, "Beyond playing to win: Diversifying heuristics for GVGAI," in *Proc. IEEE Conf. Comput. Intell. Games*, 2017, pp. 118–125.

[82] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying MCTS for human-like general video game playing," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 2514–2520.

[83] M. C. Green, A. Khalifa, G. A. Barros, and J. Togelius, ""Press space to fire": Automatic video game tutorial generation," in *Proc. 13th Artif. Intell. Interactive Digital Entertainment Conf.*, 2017, pp. 75–80.

[84] M. C. Green, A. Khalifa, G. A. Barros, T. Machado, A. Nealen, and J. Togelius, "AtDelfi: Automatically designing legible, full instructions for games," in *Proc. Found. Digital Games*, 2018, pp. 17:1–17:10, Art. no. 17.

[85] D. Anderson, M. Stephenson, J. Togelius, C. Salge, J. Levine, and J. Renz, "Deceptive games," in *Proc. Int. Conf. Appl. Evol. Comput.*, New York, NY, USA: Springer, 2018, pp. 376–391.

[86] C. Guerrero-Romero, S. M. Lucas, and D. Perez-Liebana, "Using a team of general AI algorithms to assist game design and testing," in *Proc. IEEE Conf. Comput. Intell. Games.*, 2018, pp. 1–8.

[87] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *European Conference on the Applications of Evolutionary Computation*. New York, NY, USA: Springer-Verlag, 2015, pp. 369–380.

[88] K. Kunanusont, S. M. Lucas, and D. Perez-Liebana, "Modelling player experience with the N-Tuple bandit evolutionary algorithm," in *Proc. Artif. Intell. Interactive Digital Entertainment*, 2018, to be published.

[89] D. Pérez-Liébana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "Analyzing the robustness of general video game playing agents," in *Proc. IEEE Conf. Comput. Intell. Games.*, 2016, pp. 1–8.

[90] M. Stephenson *et al.*, "A continuous information gain measure to find the most discriminatory problems for AI benchmarking," 2018, arXiv: 1809.02904.

[91] V. Volz *et al.*, "Gameplay Evaluation Measures," *Dagstuhl Follow-Ups*, vol. 7, no. 11, pp. 86–129, 2017.

[92] X. Neufeld, "Procedural level generation with answer set programming for general video game playing," Master's thesis, Faculty Comput. Sci, Inst. Knowl. Lang. Eng., Univ. Magdeburg, Magdeburg, Germany, 2016.

[93] R. D. Gaina, "The 2 Player General Video Game Playing Competition," Master's thesis, Dept. Sch. Comput. Sci. Electron. Eng., Univ. Essex, Colchester, U.K., 2016.

[94] J. Lehman and R. Miikkulainen, "General video game playing as a benchmark for human-competitive AI," in *Proc. AAAI-15 Workshop Beyond Turing Test*, 2015, pp. 1–2.

[95] G. Brockman *et al.*, "Openai Gym," 2016, arXiv:1606.01540.

[96] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013.

[97] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The Malmo platform for artificial intelligence experimentation," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 4246–4247.