Analyzing the Robustness of General Video Game Playing Agents

Diego Pérez-Liébana
University of Essex
Colchester CO4 3SQ
United Kingdom
dperez@essex.ac.uk

Spyridon Samothrakis
University of Essex
Colchester CO4 3SQ
United Kingdom
ssamot@essex.ac.uk

Julian Togelius
New York University
715 Broadway, 10003
New York
julian@togelius.com

Tom Schaul
Google DeepMind
5 New Street Square
London EC4A 3TW
United Kingdom
schaul@google.com

Simon M. Lucas
University of Essex
Colchester CO4 3SQ
United Kingdom
sml@essex.ac.uk

***Abstract*—This paper presents a study on the robustness and variability of performance of general video game-playing agents. Agents analyzed includes those that won the different legs of the 2014 and 2015 General Video Game AI Competitions, and two sample agents distributed with its framework. Initially, these agents are run in four games and ranked according to the rules of the competition. Then, different modifications to the reward signal of the games are proposed and noise is introduced in either the actions executed by the controller, their forward model, or both. Results show that it is possible to produce a significant change in the rankings by introducing the modifications proposed here. This is an important result because it enables the set of human-authored games to be automatically expanded by adding parameter-varied versions that add information and insight into the relative strengths of the agents under test. Results also show that some controllers perform well under almost all conditions, a testament to the robustness of the GVGAI benchmark.**

I. Introduction: Games and competitions

Evaluation of algorithms using games and competitions is a common practice in the Game AI community, and to a certain extent in the wider AI community. Games provide parameterizable benchmarks that allow for fast experimentation with multiple approaches, while competitions establish a common framework and set of rules to guarantee that these algorithms are compared in a fair manner [1].

Recently, a new general framework for creating and playing video games was introduced [2], [3], [4], accompanied by a competition [5], [6]. This framework is called the General Video Game AI Framework, and the competition the General Video Game AI Competition; both are abbreviated "GVGAI". A main feature of the framework is to allow for the creation of arbitrary games in a high-level game-specific language, which can then be used as benchmarks for artificial (and maybe real?) agents. A distinct advantage of GVGAI over other benchmarks is the possibility to generate/create new games in addition to using a pre-existing set of older, established games (as, for example, is done in the very popular Arcade Learning Environment [7]). Additionally, one can systematically vary certain qualities of the games involved and examine how different controllers react. One could even go a step further and design games that embody specific qualities that would advantage or disadvantage certain agent creation methods. Until now, this ability of the GVGAI framework has not been explored; we have not seen either carefully tuned games aiming to portray different agent qualities, or any exploitation of the ability to modify any of the properties of the games.

It is well-known that some game-playing methods are more robust to imperfections in the sensors or forward model, noise or hidden information than others. For example, A* can play Super Mario Bros near-optimally given linear levels, but tends to create "brittle" plans that rely on planned actions executing perfectly. Monte Carlo Tree Search, with its stochastic estimates of action values, struggles to keep up with A*-based planning under normal conditions. However, when noise is introduced to the model the performance of A* drops drastically whereas MCTS performs almost as well as before [8].

An important part of the justification for GVGAI in particular and general game playing in general is that the agents' *general* intelligence is tested, as agent developers cannot tailor their performance to a particular game. That's why agents are tested on unseen games, which are developed for each round of the competition. However, the developers of agents could still rely on certain assumptions about the GVGAI game engine, for example about the determinism of games and reliability of the forward model. Arguably, agents that are less dependent on such assumptions—less brittle—are more generally capable or "intelligent". The obvious way to find out how brittle agents are is to vary all aspects of the game engine and see what happens to the performance of said agents.

This paper is an initial exploration of the effects of large-scale modification of game characteristics. The goal is to identify how robust game-playing algorithms are to particular changes in the reward structure and the existence of uncertainty in the form of noise.

While, to the best of our knowledge, this is the first time that such a systematic exploration is conducted in such a large number of games, with the explicit aim of testing robustness, there has been some work generating games using a parameter space and then using controllers that portrayed certain human-like qualities in order to better understand the resulting design parameter space [9]; once game-space is understood, it can be searched for game variants that differ from existing games while still being playable [10].

The rest of the paper is organised as follows; section II describes the framework used, while Section III introduces

a selection of controllers that we are going to use in our evaluation. Section IV discusses the original rankings obtained by the controllers presented previously in a subset of GVGAI games. Section V describes the modifications done to the games and how each controller fared. We conclude with a short discussion in Section VI.

II. General Video Game AI

A. The Framework

The GVGAI framework provides information about the game state via Java game objects. Its interface provides means to create agent controllers that can play in any game defined in the Video Game Description Language (VGDL [2], [4]). An agent implemented in this environment must be able to select moves in real-time, providing a valid action in no more than 40ms at each time step.

This controller receives information about the game state, including factors like the game status (winner of the game, score, time step), the player's state (position, resources gathered), and position of the different sprites (identified only by an integer id for its type) in the level. The dynamics of these sprites and the victory conditions are never given to the player. It is the agents responsibility to discover the game mechanics while playing. However, the agent is provided with a forward model to reason about the environment, a tool that allows the agent to simulate actions and roll the game forward to one of the possible next states of the game. The forward model is very fast and almost all successful agents simulate hundreds or thousands of game states for each decision taken. For more information about the interface and constituents of the framework, the reader is referred to [5].

B. The Games

Four games (out of the 60 distributed with the framework) have been used in this study: *Aliens*, *Butterflies*, *Sheriff* and *Seaquest*. These games have been chosen according to the following characteristics:

- High percentage of victories: Not even the best controllers submitted to the competition (by rankings, the ones used in this study) are able to achieve victories in all games distributed with the GVGAI framework. Three of the games selected average a percentage of victories above 90%, with only *Seaquest* averaging around 45%.
- Smooth scoring: All games provide small increments of score through their play (rather than having no score change but a point given or taken when the game is won or lost, respectively). Games that provide a different score landscape are left for future work.
- Different set of actions: Not all games in GVGAI provide the same set of available actions. By choosing games with different sets, the experiments will permit an analysis on how this factor affects results after applying the different game modifications.
- They are all stochastic in nature.

These four games are described next:

- **Aliens:** Similar to the classic *Space Invaders*, this game features the player (avatar) moving along the bottom of the screen, shooting upwards at aliens, who fire back at the avatar. The avatar can use the actions *Left*, *Right* and *Use* (to shoot). The player loses if touched by an alien or its bullet, and wins if all aliens are destroyed. 1 point is awarded for each alien or protective structure destroyed by the avatar and 1 point is subtracted if the player is hit.
- **Butterflies:** The avatar must capture butterflies that move randomly. If a butterfly touches a cocoon, more butterflies are spawned. The player wins if it collects all butterflies, but loses if all cocoons are opened. 2 points are awarded for each butterfly captured. The avatar can use the actions *Left*, *Right*, *Up* and *Down*.
- **Sheriff:** The avatar is at the center of the screen and the objective is to kill all the bandits that move in circles along the level, shooting at the player. There are also some structures in the level that can be used as cover. 1 point is awarded for each bandit killed, and 1 point is subtracted if the avatar dies. The avatar can move in the four directions and shoot.
- **Seaquest:** Remake of the Atari game by the same name. The player controls a submarine that must avoid animals whilst rescuing divers by taking them to the surface. Also, the submarine must return to the surface regularly to collect more oxygen, or the player loses. The submarine's capacity is 4 divers, and it can shoot torpedoes at the animals. 1 is point awarded for killing an animal with a torpedo, and 1000 points for saving 4 divers in a single trip to the surface. As in *Sheriff*, the avatar can move in the four directions and shoot.

C. The Rankings

The GVGAI Competition rankings system, which is also used in this paper, aims to reward those controllers that perform well across different games, rather than relying on differences of performance in particular games.

For each one of the games used, all controllers are sorted according to three criteria, in the following order of importance: percentage of victories, average of score achieved and time spent on the victories (the lower, the better). Then, controllers are awarded with points according to this game ranking, following the Formula 1 scoring system: $\{25, 18, 15, 12, 10, 8, 6, 4, 2, 1\}$, where 25 points are awarded to the best controller, 1 to the tenth, and no points beyond that rank. In order to determine the overall best, all points per game are added up and the controller with the highest sum is declared the winner. In case of a draw in points, the number of first positions in a game unties the ranking, proceeding to the highest number of second, third, etc. positions until the tie is broken.

III. Controllers

This section describes the different controllers that have been used in this study. The first two, Sample Open

Loop Monte Carlo Tree Search (Sample OLMCTS, Section III-A) and Rolling Horizon Genetic Algorithm (RHGA, Section III-B), are sample controllers distributed with the framework. The third controller, Open Loop Expectimax Tree Search (OLETS, Section III-C), was the winner of the 2014 GVGAI competition. Finally, the last three controllers[1] were the winners of the three legs of the 2015 GVGAI Competition: YOLOBOT (GECCO 2015, Section III-D), Return42 (CIG 2015, Section III-E) and YBCRIBER (CEEC 2015, Section III-F).

A. Sample OLMCTS

Monte Carlo Tree Search (MCTS) [11] is a very popular tree search technique that iteratively builds an asymmetric tree in memory to estimate the value of the different actions available from a given state. Starting from the current state, the algorithm repeats the following steps in iteration until the time budget is over:

First, a *Tree Selection* process selects actions until reaching a state from which not all possible moves have been taken. These actions are selected according to a `Tree Policy`, like for instance the Upper Confidence Bounds (UCB1; see Equation 1 [12]), which balances between exploitation of the best actions found so far and exploration of the ones employed less often.

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

where $N(s)$ represents the number of times the state s is visited, $N(s,a)$ is the number of times an action a is taken from s, and $Q(s,a)$ indicates the empirical average of the rewards obtained when picking an action a from s. The exploration-exploitation balance can be tempered by the value of C: setting high values gives priority to exploration, while values closer to 0 reward those actions $a \in A(s)$ with a higher expected reward.

The second step, *Expansion*, adds a new child to the node reached at the end of the previous one. Next, a *Monte Carlo Simulation* is performed from the new node until reaching the end game or a predetermined depth. This simulation picks actions on each state according to some `Default Policy`, which could select moves uniformly at random or biased by an heuristic based on the features of the state. Lastly, the *Back-propagation* phase uses the reward observed in the state reached at the end of the *Monte Carlo Simulation*, to update the $Q(s,a)$ values of all nodes visited during the *Tree Selection* step.

The distinction between Open Loop and Closed Loop MCTS resides in using the forward model during the *Tree Selection* phase or not. In closed loop MCTS, the algorithm assumes that is stable to store game states on the nodes of the tree when *Expansion* is performed, and therefore the *Tree Selection* step can simply navigate the tree without the need of calculating the new states. If randomness is encountered, instead of acting according to the tree policy, a random guess is made as to what state one might land after an action. This is a valid approach for all games, and indeed the only really "correct" one but it may lead to sub-optimal performance on stochastic scenarios (as the games used in this research work), where one might focus too much on exploring all future possible states, never having enough time to collect enough information to perform well. Another approach is to behave in an open-loop manner - Open Loop MCTS (OLMCTS) only stores the statistics on the tree nodes, and generates the next state using the forward model to average over the distribution of possible next states. Note that for deterministic settings open-loop and close-loop are the same. For more details about this distinction, the reader is encouraged to read [13].

For the experiments performed in this experiment, the number of moves performed on each iteration is set to 10, and $C = \sqrt{2}$.

B. RHGA

Rolling Horizon Genetic Algorithm (RHGA) employs a fast evolutionary algorithm to evolve a sequence of actions to be executed from the current game state. It is an open loop implementation of a minimalistic steady state genetic algorithm, known as a microbial GA [14].

Each individual receives a fitness equal to the reward observed in the state reached at the end of the action sequence, which has a length of 7. Two different individuals are selected and evaluated from a population, and the one that obtains the worse fitness is mutated randomly, with probability $1/7$, whereas certain parts of its genome are recombined with parts from the other's genome with probability 0.1.

Both OLMCTS and RHGA use the same function to evaluate a state. The procedure works as follows: the reward is the score of the game at that state plus a high positive (respectively, negative) number if the game is finished with a victory (resp. loss).

C. OLETS

Open Loop Expectimax Tree Search (OLETS), created by Adrien Couëtoux, is an algorithm inspired by Hierarchical Open-Loop Optimistic Planning (HOLOP, [15]). As OLMCTS, OLETS does not store the states in memory, but uses the sampled sequences to build a tree.

A first difference with OLMCTS is that OLETS does not use any roll-out and relies on the game scoring function to give a value to the leaves of the tree. Additionally, another important difference is that the empirical average of rewards obtained by performing simulations is not used in the UCB1 policy (see Equation 1). Instead, OLETS replaces $Q(s,a)$ with the *Open Loop Expectimax* (OLE) value ($r_M(n)$), as calculated in Equation 3).

$$r_M(n) = \frac{R_e(n)}{n_s(n)} + \frac{(1 - n_e(n))}{n_s(n)} \max_{c \in C(n)} r_M(c) \qquad (2)$$

[1]To the knowledge of the authors of this paper, the descriptions of these controllers have not been published to date. All these controllers are accessible for download at the competition website, www.gvgai.net

where $n_s(n)$ the number of simulations that visited the node n, $n_e(n)$ the amount of them that end in n, and $R_e(n)$ the accumulated reward from this last subset. $C(n)$ the set of children of n, and $P(n)$ the parent of n. For more details about this algorithm, please consult [5].

D. YOLOBOT

This controller, created by Tobias Joppen, Nils Schroeder and Miriam Moneke, was declared winner of the 2015 GVGAI championship, as they obtained the highest sum of scores across the three legs run that year. Their approach uses pathfinding to first identify those sprites that can be reached from the avatar's position, creating a list with the nearest reachable sprite of each type. At the same time, it also tries to identify if the game is deterministic or not, using the forward model to spot differences on states reached after applying the same action from a given state. This is done to choose which algorithm to use to try to discover how valuable these sprites are within the game. If the game is deterministic, YOLOBOT uses Best First Search (BFS) to navigate to the target sprite. In case the game is deemed as stochastic, the algorithm of choice is an open loop version of MCTS, in order to get closer to the aimed sprite without losing the game due to stochasticity.

E. Return42

This controller, created by Tobias Welther, Oliver Welther, Frederik Buss-Joraschek and Stefan Hbecker is a hyper-heuristic that combines different algorithms which are used depending on the type and state of the game. Initially, the games are differentiated by being deterministic or not, a feature checked using the forward model to determine if multiple states derived from the same original state are the same. If the game is deterministic, an A-Star algorithm is used to determine future states with high scores and possibly winning conditions. In case the game is stochastic, random walks are used to determine the best action based on a hand-crafted heuristic that considers score and changes on resources and NPCs in the game.

F. YBCRIBER

This controller was submitted by Ivan Geffner, Tomas Geffner and Felix Mirave. The algorithm is based on Iterative Width (IW [16]) with a dynamic look-ahead scheme. A previous version of this work can be found at [17]. `YBCRIBER` employs some basic statistical learning to save information about each sprite at each look ahead, which it then uses to select actions in stochastic games and to prune actions over the IW search. Additionally, a danger prevention mechanism minimizes the chances of the avatar being killed in close proximity of hazards.

IV. Default Rankings

All controllers described in Section III have been executed 100 times in each one of the 5 levels of the 4 games detailed in Section II-B. Therefore, each controller plays 500 times on each game. The percentage of victories, average of scores and time spent are recorded, and non-parametric Wilcoxon Signed Rank tests are computed to determine statistical significance (p-value < 0.05). All experiments performed in this research have been carried out in this manner, for the default settings and for each one of the different environment configurations described in Section V.

Table I shows the results for the tested controllers in the games selected. The percentage of victories, average of scores and time steps used to complete the game are shown here with their respective standard error measures. First of all, it is worthwhile mentioning that there is no superior algorithm that achieves the best results in all games tested. Both in *Aliens* and *Butterflies*, three controllers achieve 100% of victories, the first metric in order of importance. Note that these controllers are not the same in both games. *Sheriff* is revealed to be a slightly more complicated game, as no controller achieves the maximum amount of victories. It seems, however, to be easier than *Seaquest*, where the best controller obtained less than 70% of victories.

The variability of these games can also be observed in two factors: First, winners of some games can perform badly in others (i.e., `YOLOBOT` is the leader in *Aliens*, while achieving 0.20% of victories in *Seaquest*; or like `Return42`, which is the best controller in *Seaquest* but the worst one in *Sheriff*). Secondly, there is a high variance in the scores typically achieved on each game, as Table I shows.

Table II shows the rankings derived from these results. The controller that ranks first in this set of games is `OLETS`, closely followed by `YBCRIBER`. It is interesting to see how `YBCRIBER` ranks high albeit it does not perform the best in any game. This is due to its high general performance (ranking 2^{nd} or 3^{rd} in all games), a consequence derived from this ranking system, which rewards controllers that perform well across different games.

V. Experiments

This section describes the experiments performed for this paper. Each section details the changes and results obtained for each one of the different configurations tested.

A. Reward Penalization

In this setting, the GVGAI framework is modified so that every time an agent performs any action, the score in the game is reduced by 1 point. In principle, one could assume that controllers that are able to perform well using the minimum possible amount of moves would be rewarded in the rankings. These rankings are shown in Table III[2].

All controllers seem to resist quite well the penalizations set to the actions performed, with the exception of `Return42`. This controller is specially affected by this change, as it is the one with the highest drop in percentage of victories (from 81.55% to 71.10%). The first and the second controller alternate positions compared to the original rankings (where `OLETS` was 1^{st} and `YBCRIBER` was 2^{nd}).

[2]To save space, no tables are reported for individual games and scores achieved, albeit some of those results are discussed.

TABLE I

PERCENTAGE OF VICTORIES AND AVERAGE OF SCORE ACHIEVED (PLUS STANDARD ERROR) IN 4 DIFFERENT GAMES. FOURTH, SIXTH AND EIGHTH COLUMNS INDICATE THE APPROACHES THAT ARE SIGNIFICANTLY WORSE THAN THAT OF THE ROW, USING THE NON-PARAMETRIC WILCOXON SIGNED-RANK TEST WITH P-VALUE < 0.05. BOLD FONT FOR THE ALGORITHM THAT IS SIGNIFICANTLY BETTER THAN ALL THE OTHER 5 IN EITHER VICTORIES OR SCORE.

Game	Algorithm	Victories (%)	Significantly better than ...	Scores	Significantly better than ...	Timesteps	Significantly better than ...
Aliens	**A: YOLOBOT**	100.00 (0.00)	B, D, E	70.56 (0.59)	B, C, D, E, F	434.55 (1.52)	B, C, D, E, F
	B: OLETS	98.80 (0.49)	Ø	66.91 (0.64)	E, F	790.96 (3.65)	Ø
	C: YBCRIBER	100.00 (0.00)	B, D, E	69.56 (0.60)	B, E, F	470.41 (1.82)	B, D, E, F
	D: Return42	97.40 (0.71)	Ø	68.43 (0.68)	B, E, F	513.13 (8.50)	B, E, F
	E: OLMCTS	99.20 (0.40)	D	61.16 (0.52)	Ø	613.02 (4.12)	B
	F: RHGA	100.00 (0.00)	B, D, E	64.99 (0.60)	E	599.38 (3.02)	B, E
Butterflies	A: YOLOBOT	95.60 (0.92)	E	27.80 (0.63)	C, D	528.03 (24.26)	E
	B: OLETS	100.00 (0.00)	A, E, F	26.13 (0.53)	C, D	170.53 (5.74)	A, E, F
	C: YBCRIBER	100.00 (0.00)	A, E, F	23.19 (0.43)	Ø	59.45 (0.88)	A, B, D, E, F
	D: Return42	100.00 (0.00)	A, E, F	24.59 (0.48)	C	73.68 (1.38)	A, B, E, F
	E: OLMCTS	86.60 (1.52)	Ø	31.44 (0.69)	A, B, C, D	728.91 (21.82)	Ø
	F: RHGA	94.40 (1.03)	E	32.89 (0.75)	A, B, C, D	447.54 (17.48)	A, E
Sheriff	A: YOLOBOT	95.00 (0.97)	D	8.52 (0.09)	C, D, E, F	826.31 (13.63)	C, E
	B: OLETS	97.20 (0.74)	A, D, F	9.18 (0.07)	A, C, D, E, F	679.67 (11.08)	A, C, D, E, F
	C: YBCRIBER	96.80 (0.79)	D, F	6.09 (0.08)	Ø	1006.00 (4.97)	D
	D: Return42	59.00 (2.20)	Ø	6.49 (0.17)	C	1018.96 (28.82)	Ø
	E: OLMCTS	97.40 (0.71)	A, D, F	6.56 (0.08)	C, D	1001.62 (4.69)	D
	F: RHGA	93.40 (1.11)	D	8.04 (0.08)	C, D, E	808.00 (13.47)	A, C, E
Seaquest	A: YOLOBOT	0.20 (0.20)	Ø	97.54 (12.69)	Ø	1572.78 (2.29)	Ø
	B: OLETS	60.00 (2.19)	A, E, F	1309.77 (74.33)	A, C, E, F	1266.04 (16.90)	A, E, F
	C: YBCRIBER	60.60 (2.19)	A, E, F	452.48 (28.84)	A	1180.91 (11.55)	A, B, E, F
	D: Return42	69.80 (2.05)	A, B, C, E, F	2858.29 (123.42)	A, B, C, E, F	1170.54 (13.14)	A, B, C, E, F
	E: OLMCTS	46.60 (2.23)	A, F	508.48 (38.61)	A	1266.75 (12.99)	A, F
	F: RHGA	24.60 (1.93)	A	301.21 (27.40)	A	1384.76 (12.76)	A

TABLE II

RANKINGS TABLE FOR THE COMPARED ALGORITHMS ACROSS ALL GAMES. IN THIS ORDER, THE TABLE SHOWS THE RANK OF THE ALGORITHMS, THEIR NAME, TOTAL POINTS, AVERAGE OF VICTORIES AND POINTS ACHIEVED PER GAME, FOLLOWING THE F1 SCORING SYSTEM.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	**OLETS**	68	89.00	10	**25**	18	15
2	**YBCRIBER**	66	89.35	18	15	15	18
3	**Return42**	59	81.55	8	18	8	**25**
4	**YOLOBOT**	57	72.70	**25**	12	12	8
4	**OLMCTS**	57	82.45	12	8	**25**	12
5	**RHGA**	45	78.10	15	10	10	10

TABLE III

RANKINGS TABLE IN THE *Reward Penalization* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	**YBCRIBER**	80	94.55	18	**25**	12	**25**
2	**OLETS**	61	89.45	10	18	18	15
3	**YOLOBOT**	58	72.95	**25**	10	15	8
4	**OLMCTS**	57	82.95	12	8	**25**	12
5	**Return42**	49	71.10	8	15	8	18
6	**RHGA**	47	80.15	15	12	10	10

Penalizations affect controllers differently, in different degrees, but the changes in performance are not extremely large in this setting. Regarding scores obtained, all controllers obtain now negative scores, but the cross comparison among them shows stability in the results, without major changes in performance in this metric.

TABLE IV

RANKINGS TABLE IN THE *Discounted Reward* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	**OLETS**	78	80.70	10	**25**	18	**25**
2	**OLMCTS**	73	87.15	15	15	**25**	18
3	**RHGA**	58	81.15	18	18	12	10
4	**YOLOBOT**	56	63.90	**25**	8	15	8
5	**Return42**	44	56.45	12	12	8	12
6	**YBCRIBER**	43	55.95	8	10	10	15

B. Discounted Reward

In this setting, the score returned by the forward model for a given state s is discounted depending on the depth of search (d), according to the following scheme:

$$r_{disc}(s) = r_{raw}(s) \times D^d \tag{3}$$

where D is the discount factor, set to 0.9 to produce a significant (but not too damaging) effect on the controllers. The question that this modification poses is to verify how robust the controllers are to delayed rewards that are discounted in the future. The rankings for this configuration are shown in Table IV

This setting affects the controllers more than the previous one, although the first ranked controller is still the same (`OLETS`). In this configuration, `YBCRIBER` is the agent that suffers the most significant drop on the averages of victories, going from 89.35% to 55.95%.

TABLE V
RANKINGS TABLE IN THE *Noisy World* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	**OLMCTS**	83	74.75	**25**	8	**25**	**25**
2	**Return42**	63	50.70	10	**25**	10	18
3	**YBCRIBER**	53	49.00	15	18	8	12
4	**RHGA**	52	53.70	12	12	18	10
5	**YOLOBOT**	51	53.70	18	10	15	8
6	**OLETS**	50	49.75	8	15	12	15

It is interesting to note that reward discounting has such a surprisingly disruptive effect on the rankings. This modification affects the distinct agents tested in this study in different ways, and suggests as an open question whether it would be possible to identify concrete changes that would benefit particular controllers. In this scenario, the biggest impact happens in the game *Seaquest*, where the performances of YBCRIBER and Return42 plummet (the latter in percentage of victories, the former in both victories and score), OLMCTS increases slightly, and OLETS remains the same, enough to keep the first position in this game (and consequently, in the overall ranking).

Another interesting observation is that both sample controllers (OLMCTS and RHGA) are resilient to this setting, which allows them to climb to the 2^{nd} and 3^{rd} positions of the rankings, respectively.

C. Noisy World

In this modification, noise is added to the actions executed by the controller. Concretely, with a probability p, a different random action is chosen to be performed instead of the one intended by the controller. p was set to a high value, 0.25, in order to achieve a big impact in the controllers employed in this study. This noise is introduced both in the real game and in the forward model. The rankings obtained with this setting are shown in Table V.

This modification in the game engine and forward model produces a very important change in the rankings. The most significant is that a new controller gains the first position in the rankings: OLMCTS, with a relevant difference of points and percentage of victories with the second (Return42, 19 and 24.05%). Actually, it becomes the best controller in three out of the four games tested. On the other hand, OLETS, the best controller in the default setting, drops to the last position.

In general, all agents observe an important drop on the average of victories achieved (between 20% and 40%), with the exception of OLMCTS that, resilient to this modification, only suffers a drop of 7.7%. The differences on scores achieved are not large, with the exception of *Seaquest*, where all controllers achieved significantly lower scores. The loss in *Sheriff* and *Aliens* is smaller, and *Butterflies* experiences a slight increase. This change in *Butterflies* could be explained by the nature of the game (see Section II-B): higher scores are achievable only when less cocoons remain closed, but the game is lost when all cocoons open.

TABLE VI
RANKINGS TABLE IN THE *Broken World* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	**OLMCTS**	85	68.50	**25**	10	**25**	**25**
2	**YBCRIBER**	58	46.50	15	15	10	18
3	**YOLOBOT**	56	49.75	18	8	18	12
4	**OLETS**	55	40.30	10	18	12	15
5	**Return42**	49	34.00	8	**25**	8	8
6	**RHGA**	49	42.15	12	12	15	10

TABLE VII
RANKINGS TABLE IN THE *Broken Forward Model* SETTING.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3
1	**OLMCTS**	71	83.05	18	10	**25**	18
2	**OLETS**	63	63.80	10	18	10	**25**
3	**YBCRIBER**	60	77.15	15	12	18	15
4	**YOLOBOT**	58	64.05	**25**	8	15	10
5	**RHGA**	51	57.00	12	15	12	12
6	**Return42**	49	38.45	8	**25**	8	8

D. Broken World

In this setting, the same configuration as in the previous case was used, but in this case only the real game can introduce noise in the actions supplied, while the forward model is always accurate. Again, $p = 0.25$ and the rankings are detailed in Table VI. The idea of this modification is to test how the agents can cope with a forward model that does not reproduce noise in the real game.

The new results obtained with this modification are similar to those achieved in the previous case. OLMCTS becomes the highest ranked entry achieving the best result in the same three games as shown in Section V-C, and drop in victory percentage happens across all controllers.

Note that the drop in percentage of victories is higher than in the previous scenario, where even OLMCTS loses 20.5 percentage points. This could be explained by the fact that inaccuracies are now introduced due to the noise included in the actions executed in the real game, but not in the forward model. However, it is interesting to note that again one of the sample (hence, simplest with regards to the value function) controllers suffers this effect the least.

Finally, regarding the games in particular, Butterflies still remains as the game where percentage of victories change the less (hence also being the game where OLMCTS does not rank the first).

E. Broken Forward Model

Finally, this setting proposes the complementary scenario to the one shown in the previous section. Noise with $p = 0.25$ is introduced only in the forward model, while the actions supplied to the game are never altered. The rankings for this configuration are shown in Table VII.

In this final setting, OLMCTS achieves again the highest position in the rankings. It is worth noting, however, that in this case the difference with the second ranked entry (OLETS) is only of 8 ranking points. Additionally, it only achieves the

first position in one of the four games, and all controllers suffer a smaller loss in the percentage of victories than in the previous case.

An interesting observation that can be drawn from this results is that, when noise in the actions is only present in the real game instead of in the forward model, the agents have more difficulties to deal with this hazard. In other words, the algorithms tested are more robust to noise present in the forward model (when no noise is present in the real game) than vice-versa.

F. Overall Comparison

Figure 1 depicts the average of victories of all controllers in the four games tested, for the different six configurations experimented with in this research. This graphic summarizes well the findings of this study. It is clear that the latter modifications (adding noise in different parts of the framework) affect the controllers more than the first two changes in most of the games (*Butterflies* remains as an exception to this statement, where the loss in average of victories is smaller).

Concretely, it can be observed how the `Broken World` configuration produces the higher variance in the results: a forward model that is not able to simulate the noise on the actions that is present in the real game is not good enough for most controllers. However, this change does not equally affect all agents. The ones that use simpler value functions (with less domain knowledge, like `OLMCTS`) respond better to a noisy world without a noisy forward model.

It is also worth mentioning that a forward model that simulates noise, even at the high rate of executing a random action with $p = 0.25$, can cope with both a noisy and a non-noisy real game environment. In fact, in some occasions, results obtained in the `Broken Model` configuration are better than the ones from a `Noisy World`, which suggests that these techniques (especially `OLMCTS`) are robust to a noisy forward model even if the game itself is not noisy.

VI. Conclusions and Future Work

This paper described a study on the robustness of several good general video game AI controllers (concretely, the winners of the four legs of the previous competitions and some sample controllers from the GVGAI framework) when the conditions of the rewards and/or actions are changed in the environment. In this research, alterations in the rewards (introducing penalties for using certain actions, or discounting the game score) and in the action performed (either by including noise in the real game, or in the forward model, or both) are introduced to analyze how the rankings change. A key finding of the research is that some of these changes can dramatically alter the rankings of the agents, which provides a simple way to effectively expand the set of GVGAI games.

An interesting outcome of this study is that simpler controllers, those that utilize a state value function that only focuses on score and winning conditions, achieve better results when noise is introduced on the actions. The effect on the ranking differs significantly depending on the game and the agent and the nature of the modifications. For instance, controllers that included an element of best-first search (Return42 and YOLOBOT) seem to handle unexpected noise badly. This is consistent with earlier results where MCTS is able to handle the introduction of noise much better than A* [8].

Furthermore, not all simple controllers perform well under noisy circumstances: RHGA is not able to climb in the rankings as much as OLMCTS, which becomes the 1^{st} ranked entry in some scenarios or OLETS, which is able to keep the second position in these settings. Furthermore, results show that, in the noisy settings, a noisy forward model with a non-noisy real game makes the controllers behave better than introducing noise in the real game (either alone, or together with noise in the forward model). The latter condition (noisy model, deterministic world) is likely to most closely model non-game situations such as robot control.

The results shown in this paper leave us with multiple open questions for future investigation. A straightforward one could be to explore the parameter space (like the values of the noise probability p or the discount factor D) to find out at which point they actually trigger the modifications observed in this paper. In other words, it is possible to analyze the continuum of values of p to identify at which point the amount of noise introduces a change in the rankings. It would also be possible to introduce other types or noise (like variations in the states observed) to analyze how does that modify the rankings, and study the effect of this in more games (especially in those omitted by the decisions explained in Section II-B).

For instance, given that the performance of the agents does also depend on the game used, a possible question to ask is if it is possible to identify or classify games with respect to what changes can make controllers go up or down in the rankings. For instance, what features make certain games be more indifferent to penalizations in the moves made? Could we infer some game design lessons from these categorizations?

As different controllers react differently to the changes made, it is worth investigating if it is possible to automatically find the parameters that will make some controllers behave better than others. In other words, could we find, maybe by evolution, the values of certain parameters that would permit us to have any ranking desired using a specific set of games? This would parallel previous work on evolving game maps to induce differential rankings between agents [18].

This research also proposes a new way of evaluating controllers: the same agents in a set of games can perform differently depending on the setting used. Therefore, it is at least thought provoking to consider if the best controller in a competition should be the one that resists such changes in the environment best.

Finally, it could be argued that we are not only testing the robustness of the controllers, but also the robustness of the competition itself, and thus its value as a benchmark. If the rankings of controllers only depended on the amount and type of noise, this would mean the benchmark would be rather brittle. However, as observed above, some controllers do better than others under all or almost all conditions. For

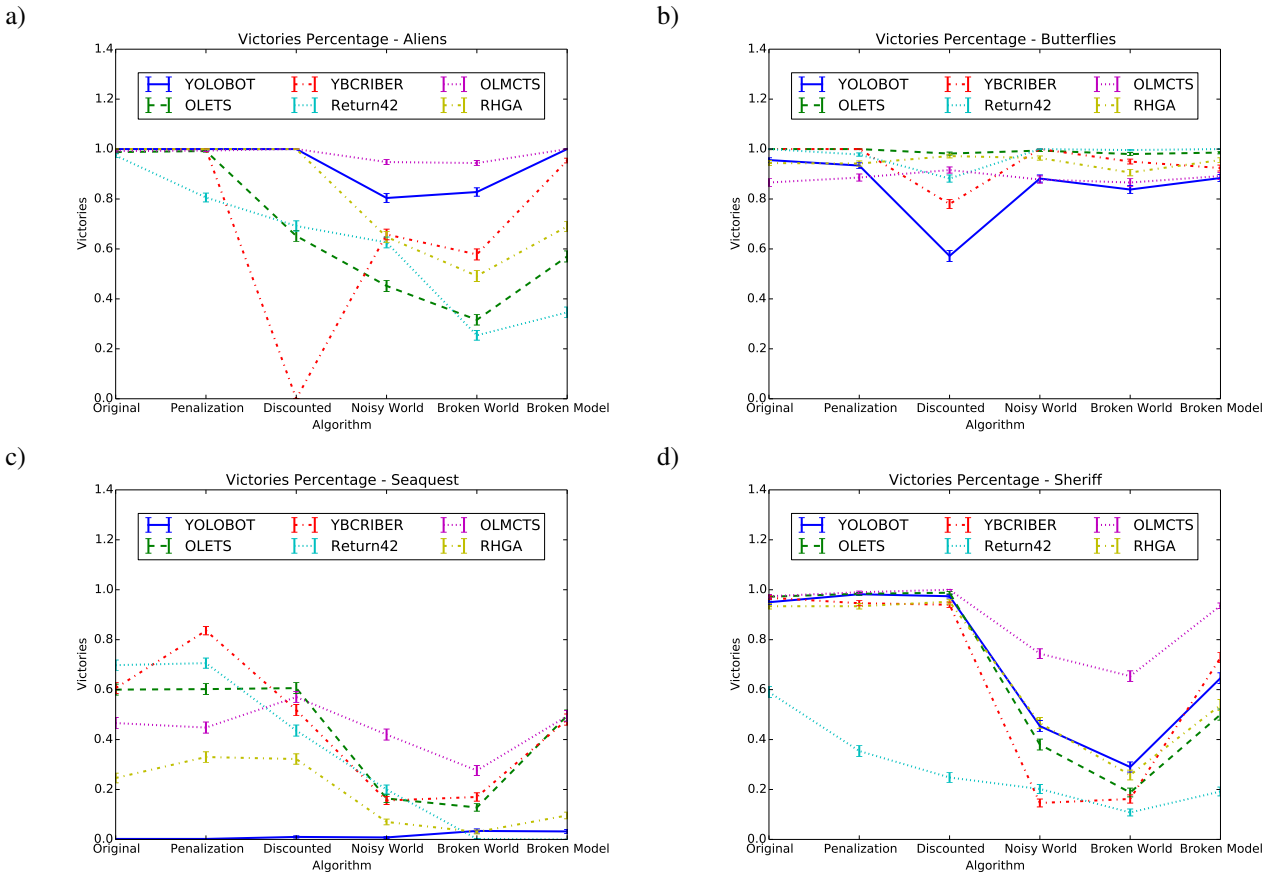

Fig. 1. Victory percentages per configuration and game.

example, OLMCTS always performs better than RHGA. It therefore seems that the underlying challenge of the GVGAI competition is fairly robust to perturbations.

Acknowledgments

The authors would like to thank the participants of the several GVGAI competition rounds for their submissions.

References

[1] J. Togelius, "How to Run a Successful Game-Based AI Competition," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 8, no. 1, pp. 95–100, 2016.

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

[3] 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, 2013.

[4] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General Video Game Playing," *Dagstuhl Follow-Ups*, vol. 6, 2013.

[5] D. Perez-Liebana, J. Togelius, S. Samothrakis, T. Schaul, S. M. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.

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

[7] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research*, 2012.

[8] E. J. Jacobsen, R. Greve, and J. Togelius, "Monte Mario: Platforming with MCTS," in *Proceedings of the Conference on Genetic and Evolutionary Computation*, New York, NY, USA, 2014, pp. 293–300.

[9] A. Isaksen, D. Gopstein, and A. Nealen, "Exploring Game Space Using Survival Analysis," in *Foundations of Digital Games*, 2015.

[10] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, "Discovering Unique Game Variants," in *ICCC Games Workshop*, 2015.

[11] C. Browne, E. J. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 4:1, pp. 1–43, 2012.

[12] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *In: ECML-06. Number 4212 in LNCS*. Springer, 2006, pp. 282–293.

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

[14] I. Harvey, "The Microbial Genetic Algorithm," in *Advances in artificial life. Darwin Meets von Neumann*. Springer, 2011, pp. 126–133.

[15] A. Weinstein and M. L. Littman, "Bandit-Based Planning and Learning in Continuous-Action Markov Decision Processes," in *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS, Brazil*, 2012.

[16] N. Lipovetzky and H. Geffner, "Width and Serialization of Classical Planning Problems," in *Proceedings of the European Conference on Artificial Intelligence*, 2012, pp. 281–285.

[17] T. Geffner and H. Geffner, "Width-based Planning for General Video-Game Playing," in *Proceedings of the IJCAI Workshop on General Intelligence in Game Playing Agents (GIGA)*, 2015.

[18] D. Perez, J. Togelius, S. Samothrakis, P. Rohlfshagen, and S. M. Lucas, "Automated Map Generation for the Physical Traveling Salesman Problem," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 5, pp. 708–720, 2014.