MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation

Hendrik Horn*, Vanessa Volz*, Diego Pérez-Liébana†, Mike Preuss‡
*Computational Intelligence Group
TU Dortmund University, Germany
Email: firstname.lastname@tu-dortmund.de
† School of Computer Science and Electronic Engineering
University of Essex, Colchester, UK
dperez@essex.ac.uk
‡Department of Information Systems
Westfälische Wilhelms-Universität Münster, Germany
Email: mike.preuss@uni-muenster.de

***Abstract*—In the General Video Game Playing competitions of the last years, Monte-Carlo tree search as well as Evolutionary Algorithm based controllers have been successful. However, both approaches have certain weaknesses, suggesting that certain hybrids could outperform both. We envision and experimentally compare several types of hybrids of two basic approaches, as well as some possible extensions. In order to achieve a better understanding of the games in the competition and the strength and weaknesses of different controllers, we also propose and apply a novel game difficulty estimation scheme based on several observable game characteristics.**

I. Introduction

The General Video Game AI (GVGAI) competition is an attempt to create artificial intelligence that is not tailored towards a specific game (akin to General Game Playing, GGP). In contrast to GGP games, GVGAI games are modeled after real, well known (albeit simple) video games and incorporate non-deterministic behavior of NPCs. Thus, learning to play these games is not trivial, despite the forward model offered by the GVGAI setup that can be used to explore possible futures of the current game state. The inherent non-determinism discourages plain game-tree search methods and renders this environment suitable to non-deterministic learning algorithms such as Monte-Carlo Tree Search (MCTS) and Evolutionary Algorithms (EA). Both approaches have been shown to work well, but MCTS based controllers tend to exhibit the best overall performance [14]. Still, as of yet, no submitted controller has been able to consistently be successful on all games, showing that all controllers have their weaknesses and strengths. A hybrid controller that combines the strengths of both methods therefore seems promising. But, to our knowledge, few combinations of the aforementioned algorithms into a single GVGAI controller have been suggested (cf. section III).

In this work, we therefore explore different hybridizations, namely (1) integrating parts of the MCTS method into a rolling horizon EA [7, 12], and (2) splitting the computation budget between both methods. Both combinations are experimentally shown to perform well, with the first hybrid possessing a small advantage. Next to these two naïve hybrids, we also try out further modifications addressing weaknesses discovered during the experiments. However, while improvements are visible in one area, the variations introduced new weaknesses to the controllers. For a more robust controller, further research is needed on how to balance the different components.

As a first step in this direction, we analyze the characteristics of different games and their correlation with the overall winrates of the different controllers in an attempt to uncover strengths and weaknesses. The analysis is based on a difficulty estimation scheme that uses different models to predict controller winrates as a proxy for difficulty from several observable game characteristics.

In the following, we first introduce the GVGAI framework (sect. II) and discuss related work in sect. III. The proposed hybrid controllers are explained in sect. IV and experimentally analyzed in sect. V. Section VI contains the analysis of the difficulty of the games. The paper concludes with a brief summary and outlook in sect. VII.

II. The GVGAI Framework and Competition

The General Video Game AI framework is an extension of *py-vgdl*, a benchmark for planning and learning problems implemented by Tom Schaul [15]. This environment proposes a Video Game Description Language (VGDL) to describe two-dimensional real-time games games in a very concise and object oriented manner. GVGAI utilizes this implementation, providing a responsive forward model to simulate actions within the game and an interface for controllers to participate in an open competition. The results and rules of this contest, which was initiated in 2014, can be found in [14].

The GVGAI framework communicates information about the game state to the controller via Java objects, although information about the nature of the game, its rules, the type of sprites present and the victory conditions are not provided. Information received contains the game status (score, winner -

if any -, and current time step), the available set of actions on the game, the player's state (position, resources collected) and the position of the different sprites, identified by an integer id, in the level.

Controllers can use 1s of CPU time for initialization, and 40ms at every game tick to return a valid action to play the game. If these limits are not respected, the controller loses the game automatically, with the exception of actions returned between 40 and 50ms, in which case the action executed in the game is *NIL* (no movement applied). During the allocated time, the controller can employ a forward model to explore the effects of actions, by rolling the current game state forward and reaching potential future states. It is important to highlight that most games have non-deterministic elements, so it is a responsibility of the controller to deal with the distribution of next states that the forward model provides from the same pair of state and action.

At the time of writing, the framework contains 80 single-player (some of them used in this research) and 10 two-player games. The single-player planning track was run as a competition in 2014 [14] and 2015, attracting more than 70 entries in total.

III. Background

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [2] is a very popular technique that iteratively builds an asymmetric tree by sampling the search space. It builds estimates of the action-values for the different states found during the search, by repeating a sequence of 4 consecutive steps: Tree Selection, Expansion, Monte Carlo Simulation and Back-propagation.

During the *Tree Selection* phase, the tree is navigated from the root according to a `Tree Policy`, until a node with actions not yet expanded is reached. A very common policy is UCB1, described in equation 1 [8],

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

where $N(s)$ indicates the number of visits to state s, $N(s,a)$ the number of times an action a is taken from s, and $Q(s,a)$ the empirical average of the rewards obtained from s through a. This policy balances between exploitation (first term of equation 1) and exploration (second term), tempered with the value of C.

During the *Expansion* phase, a new node is added to the tree as a new child. Then, a *Monte Carlo Simulation* starts from that point until reaching the end of the game or a predetermined depth, selecting actions according to a `Default Policy`, which typically selects moves uniformly at random. Finally, the *Back-propagation* step updates the $Q(s,a)$ values of all nodes visited during the *Tree Selection* step using the reward observed in the state reached at the end of the *Monte Carlo Simulation*.

MCTS has been successfully used in General Game Playing (GGP) [1], winning the 2007 and 2008 AAAI GGP competitions, and it has been extensively used in the GVGAI competitions up to date. The winner of the 2014 GVGAI competition, Adrien Couëtoux, implemented *OLETS* (Open Loop Expectimax Tree Search) [14], a variant of MCTS without Monte Carlo simulations. The tree is navigated from the root, using a variant of UCB1, until a new node is added, which state is evaluated and the result Back-propagated to the nodes of the tree. Also in the 2014 competition, the third ranked entry was the provided sample MCTS controller. It's worth highlighting that this controller employed a very simple state evaluation function, which used the game score plus a high positive (negative) value if the game was won (resp. lost), but it still performed well across several games.

B. Rolling Horizon Evolutionary Algorithms

Traditionally, in planning problems, evolutionary algorithms (EA) are used offline to train a controller or solver that then tackles the real problem [5]. A rolling (or receding) horizon EA (RHEA) operates by evolving a sequence of actions online, out of which only the first one of the best sequence or individual is performed in the real game. A fast EA, executed at every time step, tries to determine the best action sequence from the current state, evaluating each individual with an evaluation of the state reached at the end of such sequence.

This algorithm was first implemented for the Physical Travelling Salesman Problem (PTSP) by Perez et al. [12], showing a better performance than MCTS in the PTSP. More recently, the work by Justesen et al. [7] shows that RHEA can achieve high performance in Hero Academy, a game with a very large branching factor.

The GVGAI framework includes a sample controller that implements a steady state RHEA, known as microbial GA [6]. In this controller, individuals are compared in pairs, chosen at random, and the one with the worst fitness is mutated randomly, also taking parts from the other's genome with a small probability. Although this controller ranks worse than the MCTS Sample controller in all game sets, many participants have worked with different versions of this algorithm, and about 50% of the top 10 entries in the final rankings of the 2015 competitions were RHEA based algorithms[1].

C. Hybrids and Hyper-heuristics

The GVGAI Competition took place in three different legs during 2015, with a different winner for each one of them. Although the algorithms were different, all three had something in common: they were a combination of techniques that were selected depending on certain characteristics observed in the games. The winner of the first leg (*YOLOBOT*; and overall winner of the championship), employs Best First Search or MCTS to reach a targeted sprite in the game, depending on the game being identified as deterministic or not, respectively. The winner of the second leg, *Return42*, differentiates the game

[1]Results available at www.gvgai.net

according to the same concept, using A-Star for pathfinding in deterministic games, and random walks for stochastic scenarios. Finally, the winner of the last leg, *YBCRIBER*, combines a danger prevention mechanism with iterative width (IW [9]), using pruning and a dynamic look-ahead scheme to perform statistical learning on the different sprites of the game [4].

A different approach, which the research expands on, is the combination of several techniques into a hybrid algorithm. Specifically, combinations of MCTS and EA in GVGAI have been tried in previous works. For instance, Perez et al. [11] combined Fast Evolution with MCTS for General Video Game Playing. The objective was to evolve a set of weights $W = \{w_0, w_1, \dots w_n\}$ to bias action selection during the Monte Carlo simulations of the algorithm. In this work, every MC simulation evaluates a single individual, providing as fitness the reward calculated at the state reached at the end. For each state, a set of features $F = \{f_0, f_1, \dots f_n\}$ is extracted, and the relative strength of each action (a_i) is calculated as a linear combination of features and weights. On each move during the simulation, actions are picked at random with probabilities derived from applying a Softmax function to the relative strengths of each action. For more information about this algorithm, the reader is referred to [10].

In a different approach [13], a RHEA builds a tree search while evaluating the different individuals. Every time a sequence of actions is executed from the current state, new nodes will be added to this tree, one per action performed. As some initial sequences are repeated during the evaluation of the different individuals, most children of the root node will be visited repeatedly, allowing it to calculate an average of the rewards or fitness obtained by the different individuals. Finally, the recommendation policy (that chooses which action to take in the real game), can select the move based on this value, rather than depending only on the best individual. This helps reduce the effect of noise in the state evaluation of stochastic games without the need of evaluating the best individual multiple times.

IV. MCTS/EA Hybridizations for GVGAI

Of course, there are many ways to combine MCTS and EAs within one controller, and some of these possibilities have already been explored, as described in sect. III-C. When envisioning more direct hybrids, it appears to be simpler to augment a RHEA (see sect. III-B) with components taken from MCTS than the other way around. Something we have to keep in mind is that we are situated in a realtime environment. As there is a constant limit (40 ms) of time that can be employed for computing the next move, inserting a new mechanism into an existing controller at the same time means to reduce the available resources for the original algorithm.

Note that the parameter values employed for the different controllers stem from manual testing limited by the runtime of the experiments and could still be improved. Running a controller on 10 games with 5 levels each and several repeats can take several hours.

A. RHEA with rollouts: EAroll

This simple scheme takes over the RHEA algorithm and extends it with rollouts: when the simulation of the moves contained in the currently simulated individual/genome is finished, a predefined number of rollouts of parametrized length is performed from that point on, and the fitness of a move combination is computed as the average of the EA part and the MCTS part.

In order to characterize the type of hybridization, one could say that a minimal MCTS is performed during the assessment of move combinations within an EA. We therefore call this an integrated hybrid. The expected benefit of adding rollouts is that the algorithm can look into the future a bit further and thus the chance of detecting a critical path (that would with high probability lead to a loss) increases. It can therefore avoid such paths much earlier. Just extending the length of the genome (the number of consecutive moves evolved) would not have the same effect as that would mean that only one specific move combination (albeit a longer one than before) is tried. Taking into account that GVGAI games are usually non-deterministic, the chances of finding exactly that move combination that leads into a critical path can be expected to be much higher if we sparsely sample a game tree from a specific starting point than if we try only one move combination.

However, this advantage comes at a cost: adding rollouts of course uses up precious computation time, so that the number of moves that can be evaluated within the time constraints decreases. The results reported in this work have been obtained with the following parameters: genome length: 8 steps, simulation depth 9 steps, population size 7, and number of rollouts 300.

B. RHEA, then MCTS for alternative actions: EAaltActions

This alternative hybrid approach resembles an ensemble approach: after running the RHEA for a predefined time, we use MCTS for checking alternative solutions. That is, the MCTS is allowed to compute a suitable move first, excluding the one chosen by the RHEA. After finishing the MCTS run, we compare the results of the best moves detected by both algorithm and use the better one.

It is an open question how the available time (40 ms) should be distributed between the two approaches. For reasons of simplicity, we spend approximately the same amount of time on both. Our results have been obtained with the following parameters: genome length = simulation depth = 9 steps, population size 5, and number of rollouts 20.

C. EAroll plus sequence planning: EAroll-seqPlan

The basic idea of this controller is to reuse a computed sequence of moves (a plan). In keeping an existing plan, one could just start the computation from the next planned move. After every move, the first move is removed from the plan and the whole plan is shifted upwards. In a deterministic environment, this would make a lot of sense because in GVGAI one-player games, we have no opponent, and the NPCs are usually not very aggressive. Thus, continuing the

plan at least a few steps could save computation time for further exploration of future states. However, it is known that the GVGAI games are often heavily non-deterministic, so that it is not easy to predict how well our approach works. In contrast all other controllers suggested here, we cannot react quickly if something unforeseen happens.

We run this controller with the following parameters: genome (plan) length 8 steps, simulation depth 9 steps, population size 5, and number of rollouts 20.

D. EAroll + occlusion detection: EAroll-occ

Looking at the behavior of the EAroll controller, from time to time we observe that it stands still for some iterations and then performs a sequence of moves it could have started earlier. The reason for this is that an action sequence which leads to a reward (as, e.g., moving onto a diamond in the game boulder dash) has the same fitness if some inconsequential actions are added to the beginning of the list. In the literature, this problem is sometimes addressed with decaying rewards, but this approach needs to be parameterized. Instead, we want to completely remove any unnecessary actions from the sequence to improve the overall performance of the controller. In order to be able to detect the right position within the action sequence to start the execution, we need to reserve some time (5 ms) so that we can do a binary search for the starting position from the middle of the sequence to the beginning, watching out for the first occasion for which the reward stays the same. We then remove all earlier moves from the sequence.

This controller is run with the following parameter values: simulation depth 9 steps, population size 6, and number of rollouts 100.

E. EAroll + NPC attitude check: Earoll-att

The last controller variant also employs EAroll as base controller and tries to determine the attitude of the different NPC characters in order to allow or forbid moves into their neighborhood. As there is no information available at the start of the game that allows us to infer the consequences of collisions between the avatar and specific NPCs, this has to be learned during runtime. Fortunately, while setting up the next move, we simulate a lot of steps we do not actually perform, but we obtain information about when the controller would be rewarded or would lose a game. We assume that the behavior of a specific NPC type does not change completely over time (which is not always true, cf. Pac Man) and memorize if the collision had a good, negative or no effect. During the following game phases, we utilize this knowledge. Whenever the avatar moves into the vicinity of an NPC (this is parametrizable, we use a distance of 2 here), the corresponding move gets an extra reward or penalty.

The parameter values employed for this controller are: genome length 7 steps, simulation depth 8 steps, population size 10, and number of rollouts 300.

V. Experimental Analysis

The most promising hybridizations and corresponding parametrizations as described in IV, as well as the original MCTS and RHEA examples were tested and compared extensively using the GVGAI framework (see II). Each of the 7 controllers was run 20 times on each of the 5 levels of every game in game sets 1 and 2. This results in 100 playthroughs per controller per game and thus in 14 000 runs total.

In this section, the generality of the controllers will be examined based on their performances on all 20 games. In contrast, in section VI, the results are inspected more closely in terms of how certain aspects of a game affect the performance of different controllers.

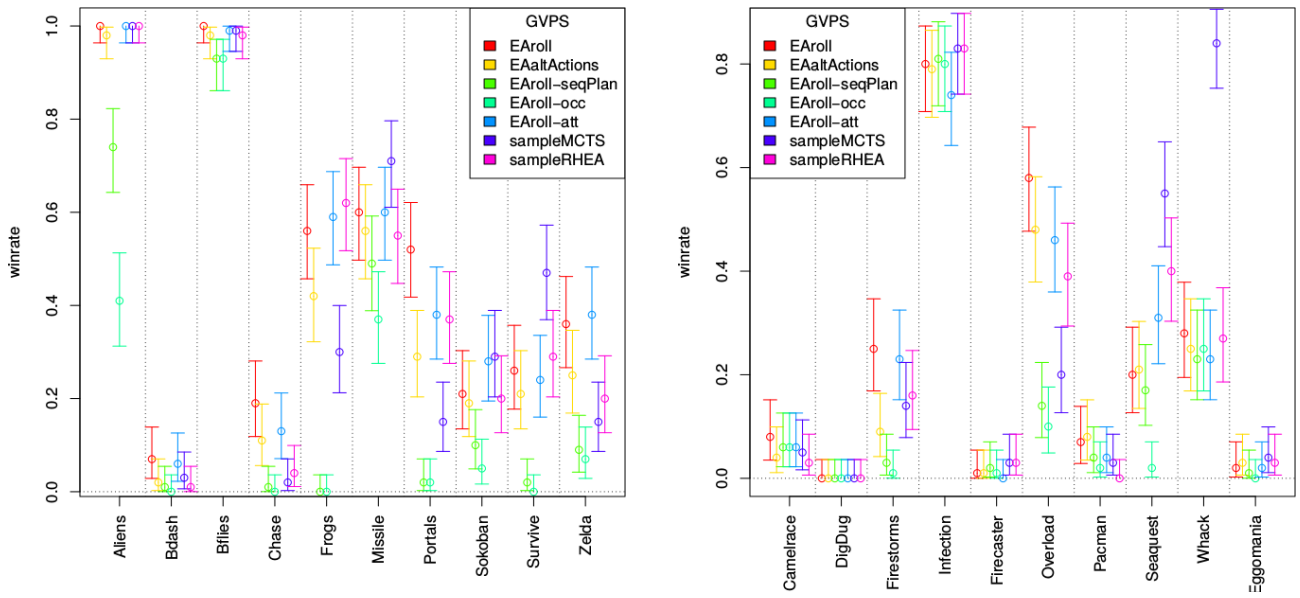
In order to obtain interpretable results that are comparable across games, we use winrates as a measure of performance since the scoring systems differ between games. However, it has to be noted that the scores potentially contain more information that could help distinguish different controllers in a statistically significant manner. In terms of measuring performance, we first compute the confidence intervals for the true winrates π of a controller on a game based on its measured winrate $\hat{\pi}$ in the experiment with $n = 100$ samples and $\alpha = 0.05$ (see Figure 1), assuming a binomial distribution and using the Pearson-Klopper method[2]. The experimental winrates are each marked with a circle within the respective interval.

From Figure 1 it is apparent that there are some games where all controllers either perform similarly badly (Camelrace, Dig Dug, Firecaster, Eggomania, Boulderdash) or well (Aliens, Butterflies). In most other games, the controllers EAaltActions, EAroll-seqPlan, EAroll-occ perform a little worse than the rest. However, there does not seem to be a clear pattern which of the other controllers is ahead of the rest in all games. Even if just taking the experimental winrates into account, there is no clear winner across all or most of the games.

However, there are some obvious differences when analyzing the controllers' overall performance, for example using the GVGAI rating system (see [14]). The GVGAI framework determines the ranking of all competing controllers on each game and rewards them according to the Formula 1 scoring system with between 25 and 0 points. The rankings on each game are determined using the winrate as a first criterion and the obtained scores as a tiebreaker. Usually, the completion time is used as a secondary tie breaker, which was dropped for this paper as we were not looking at computational speed in this context. The resulting ratings for all controllers on game sets 1 and 2 are listed in table V.

According to the ratings, EAroll performs best on both game sets, although its lead is bigger on game set 1. Overall, the ratings are much more consistent in game set 1 with EAroll-seqPlan and EAroll-occ constantly on the last two ranks while sometimes placing 2nd and 3rd on game set 2. In contrast, the rating of the MCTS controller is very robust and steady across all games. This is reflected in the total rating: EAroll-seqPlan and EAroll-occ are on the last two ranks regarding overall performance and MCTS is on 2nd place, benefiting from its

[2]R package binom



(a) (b)

Fig. 1. Confidence Intervals ($\alpha = 0.05$) for the winrates of all tested controllers on game set 1 (fig. 1a) and 2 (fig. 1b)

consistent performance. The RHEA controller and EAroll-att score similarly, with EAaltActions following behind.

However, as is apparent in figure 1, many confidence intervals for the winrates overlap and are therefore not as clear an indicator of controller performance as the resulting difference in GVGAI scores might suggest. Therefore, in order to obtain a better idea of significant differences between the overall performances of the controllers, we compute the Elo-ratings based on a pairwise comparison. The idea of the Elo-rating is to estimate the skill of a player based on the outcomes of past matches, taking into consideration the skill-level of the opponents. For more details, refer to [3].

The pairwise-comparison is conducted based on the confidence intervals for the winrates depicted in figure 1. The performances of two controllers are incomparable if the corresponding confidence intervals overlap. If they do not, one controller plays significantly better ($\alpha = 0.05$) than the other. In order to translate the comparisons to a format suitable for the Elo-rating, a controller performing significantly better than another *wins* a comparison, the other one *loses*. If the controllers are incomparable, the result is a *draw*.

It is important to note that in the GVGAI context, there is no performance development to be expected across games, since no data is transferred between games (or even runs). Therefore, for the purpose of computing the Elo-rating, all comparisons are considered to have occurred within the same time period, to avoid a bias towards the last games played. The resulting ratings and ranks are listed in table V. Both the Elo as employed by FIDE and the Glicko rating systems result in the same ranks for the controllers.

The GVGAI- and Elo-Ratings agree on placing EAroll-seqPlan and EAroll-occ on the last two ranks, which is unsurprising since they frequently seem to be performing worse than the other controllers. The Elo-Rating indicates that the first place for EAroll is due to statistically significant performance differences as well. While EAaltActions is on rank 5 in both rankings, the rest of the controllers EAroll-att, MCTS and RHEA have different rankings which seem to indicate that between those controllers, there is no clear difference in terms of overall performance.

VI. Game Difficulty Estimation

Even though some of the developed controllers are clearly not performing as well as others across all games, it is apparent from figure 1 that some games seem to be easier for all controllers than others. Additionally, despite performing

TABLE I
GVGAI-Ratings for all tested controllers for game sets 1 & 2

Rank	Player	Rating Set 1	Rating Set 2	Total
1	EAroll	206	178	384
2	sampleMCTS	163	163	326
3	sampleRHEA	138	152	290
4	EAroll-att	175	110	285
5	EAaltActions	118	141	259
6	EAroll-seqPlan	80	113	193
7	EAroll-occ	62	86	148

TABLE II
Elo-Ratings for all tested controllers based on pairwise performance comparisons on all games in game sets 1 & 2

Rank	Player	Rating	Win	Draw.	Loss
1	EAroll	2510	27	89	4
2	EAroll-att	2497	25	92	3
3	sampleRHEA	2443	20	98	2
4	sampleMCTS	2376	27	79	14
5	EAaltActions	2348	17	97	6
6	EAroll-seqPlan	1700	2	79	39
7	EAroll-occ	1525	0	70	50

at a similar level for most games, in some games, certain controllers perform significantly better or worse than the others. The best example for this is Whack-A-Mole where the standard MCTS performs significantly better than all other controllers. In this section, we take a closer look at the games in question to explain the discovered patterns.

As a preparation for a more detailed analysis we identified 10 characteristics of games that might impact the performance of controllers, extending the characterization in [14]. In most cases, the values assigned to the games per characteristic correspond to the fragment of time that the game exhibits the specific characteristic:

- enemies: Do opposing NPCs exist?
- puzzle: Does the game contain a puzzle element?
- indestructible: Are the NPCs indestructible?
- random: Is the NPCs' behavior stochastic?
- stages: Are there multiple stages to a winning condition?
- pathfinding: Is finding a path through the level necessary?
- traps: Does the game contain traps or missiles?
- chase: Do NPCs chase the player with negative effects?

There are a few exceptions, however:

- actions: Number of actions allowed in the game divided by number of actions a controller within the GVGAI framework can access (5)
- NPCs: Normalized average number of NPCs

The evaluation of the respective characteristics is done manually and may therefore contain a bias, but the characteristics were chosen so that a minimal amount of personal judgment is needed. The resulting difficulty estimation for all games in game sets 1 and 2 is shown in figure 2a with table III as legend. Considering the plot, the games seem to vary considerably in terms of difficulty and the type and combination of challenges a controller faces are diverse as well. Since the purpose of the GVGAI competition is to determine *general* video game players, this diversity between the games is expected and advantageous.

However, only in some cases does the sum of the various difficulty characteristic seem to correspond to the actual performance of the controllers, even if the controllers all perform on a similar level. For example, while Boulderdash is very difficult according to figure 2a and seems to be problematic for all controllers (cf. figure 1a), Camelrace and Firecaster result in similarly low winrates (cf. 1b) despite being considered to be much easier (cf. figure 2a). It is thus obvious, that even if the identified characteristics can describe the difficulty of a game appropriately, some factors are more important than others, some even have a positive effect on controller winrates.

To analyze the importance of the characteristics, we estimate the variable importance based on the R^2 statistic of a non-parametric regression model using only one predictor against the intercept only null model as described in the minerva R package documentation[3]. According to the Maximal Information Coefficient (MIC) that estimates the relationship strength between a difficulty characteristic and the winrate of each controller, none of the identified characteristics seem to be irrelevant. However, with average MIC values across controllers of between 0.18 and 0.41, it is clear that the relationship is more complex and can not be expressed with only one predictor. Nevertheless, the characteristics pathfinding and NPCs seem to have the highest linear relationship strength, followed by indestructible and traps. The Total Information Coefficient (TIC) reports high statistical dependence between the aforementioned characteristics and the controller winrates as well. With an average Maximum Asymmetry Score (MAS) of 0.08, all relationships appear to be relatively monotonous. Additionally, the Minimum Cell Number (MCN) is 2 for almost all relationships, indicating simple functions that can be covered with very few cells.

The various metrics mentioned indicate that it should be possible to create relatively simple models to predict the winrates of the controllers based on the difficulty characteristics. We will first learn a model that predicts controller performance based on the performance data of all controllers on both game sets. Naturally, the model in this case will only be able to pick up on the general trend, not on individual strength and weaknesses of single controllers. We used

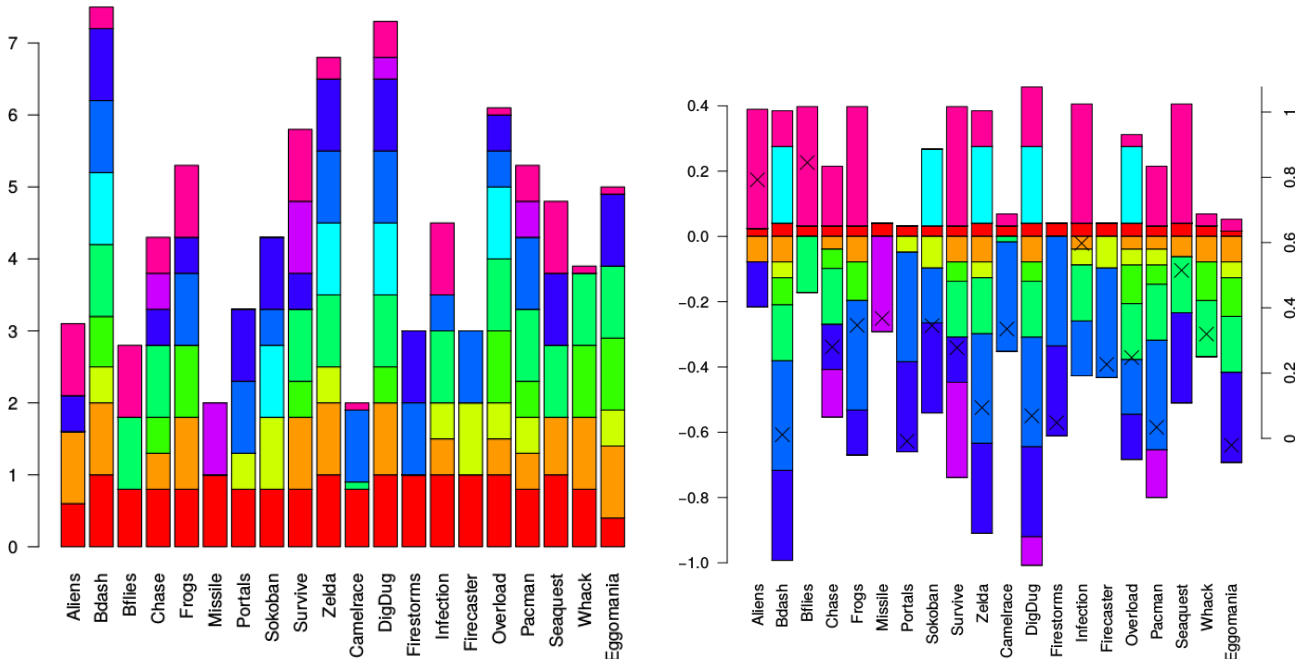
- a regression (linear, logit and logistic),
- an artificial neural network (1 hidden layer, 10 nodes),
- a random tree and forest

model with 10-fold cross-validation and a randomly drawn 90%/10% split to predict the winrates.

The neural network had the lowest mean squared error consistently (average of ≈ 0.02), but linear regression and both the random tree and random forest have very acceptable error rates as well (average MSE of $0.05, 0.03, 0.03$, respectively). Therefore, it can be seen that the identified difficulty characteristics have an effect on the winrates throughout the controllers and explain them decently. In order to analyze the influence of the different characteristic in greater detail, regression models are used for further analysis since they are easily interpretable and comparable, while still making accurate predictions for this problem. The result of a linear regression model trained on all available data is shown in figure 2b along with the predicted winrates.

The plot shows that, while most of the characteristics have an adverse effect on the predicted winrates, higher NPC, stages and action values actually seem to benefit the controllers. For NPC and action, this can be explained by the fact that all controllers are based on using the forward model in the GVGAI framework to try out different actions. This strategy works better, the earlier a sequence of actions can be evaluated in terms of the expected outcome. Having more NPCs (i.e. a high NPC value) and actions bound to every possible option available to the controller (i.e. a high actions value) results in more frequent events in the games and thus facilitates the evaluation of an action sequence. It is not clear why more complex winning conditions (as expressed by stages) improve the winrates of a controller or if this behavior is the result of having only 5 of the game with stacked winning conditions.

[3]https://cran.r-project.org/web/packages/minerva/minerva.pdf



(a) (b)

Fig. 2. Difficulty characteristics (see table III) of all games in game sets 1 and 2. Fig. 2a as estimated and fig. 2b as weighted by linear regression model. Crosses in fig. 2b represent predicted winrates to be read with the right y-axis.

TABLE III
COLORS ASSIGNED TO DIFFERENT DIFFICULTY CHARACTERISTICS

actions	enemies	puzzle	indestructible
random	stages	pathfinding	traps
chase	NPCs		

The most important characteristics in terms of the collective model are random, NPCs, chase, traps, pathfinding and stages. This can also be explained by common traits of the controllers. For example, non-deterministic games (with a high random value) decrease the reliability of the forward model. If the game involves the need to find paths and avoid traps, a general video game player that is forced to rely on exploration is at a disadvantage to strategically searching players.

However, while the collective model presented in figure 2b explains the general difficulty of games well, it is not possible to ascertain the strengths and weaknesses of individual controllers or explain the differences between controller performances on a single game. For this reason, we also learned linear regression models on all available data separately for each controller, but across all games. The resulting model coefficients are visualized in figure 3.

There are several clear differences between the controllers. For example, for the MCTS controller, pathfinding seems to be a much bigger problem than for the others, while a high number of enemies has a positive influence on its winrate. The number of actions also appears to be a much larger positive influence when compared to the other controllers, whereas the number of NPCs is less important. All these factors influence the branching factor of the game-tree and/or the number of viable options for the next action, thus indicating games where

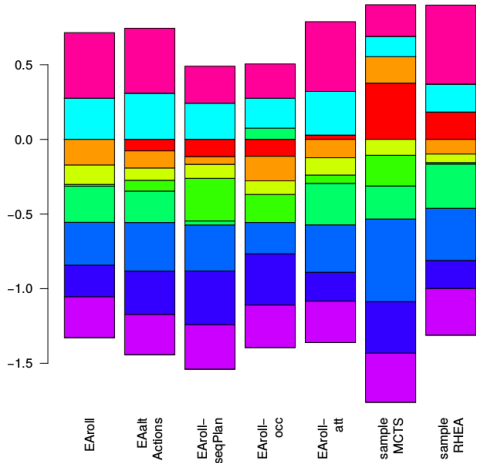


Fig. 3. Linear regression model coefficients for difficulty characteristics (see table III) visualized per controller

a Monte-Carlo approach is less likely to succeed. The MCTS controller deals well with games where the NPCs exhibit randomized behavior, probably for as long as it can execute enough rollouts. The observations also explain why MCTS is doing so well on the game Whack-a-Mole as it exhibits none of the problematic characteristics.

For the RHEA controller, being able to distinguish action sequences quickly is very important, as is reflected by the stress on the number of NPCs and events in the game. Interestingly, EAroll seems to not be affected by this difficulty as much. It seems to deal with almost all of the difficulty characteristics equally well, which explains its robust performance across all games. Its winrate seems to be almost independent from the existence of indestructible NPCs, while the modifications of

this controller have more trouble dealing with this. This is also true for the MCTS controller, while the RHEA controller is also not affected by this difficulty.

Generally, it does seem that the hybrid controllers have inherited characteristics of both controllers, resulting in more robust controllers (especially with controllers EAroll and EAroll-att), thus leading to a better overall performance. The modifications of the EAroll controller seem to fix some of its weaknesses as intended, but at the same time opening up new problems. EAroll-seqPlan, for example, is much less affected by the unpredictability of some games, possibly because it is able to save and propagate a lot more information. On the flipside, the controller is much more susceptible to indestructible enemies. However, it could very well be that these modifications and the hybridzation in general could achieve better spread of strengths and weaknesses, even eliminating some, if tuned more thoroughly.

VII. Conclusion and Outlook

Although not all of the presented hybrid RHEA/MCTS controller variants play better than the original sample controllers, we can state that there is obviously some potential in putting these two base algorithms together in order to obtain better GVGAI controllers. Judging from the difficulty analysis, the hybridization made the resulting controllers more robust. We intend to continue this line of research in (at least) two directions: a) the parametrization of our controllers has not been analyzed systematically, performance may deviate largely from our results with different parameter values, and b) it may be good to dynamically switch on/off the single modules we suggested (sequence planning, occlusion detection and NPC attitude check) as they fit for a given game. The same could also be envisioned on a larger scale for the base algorithms.

However, this requires a clear understanding of and the reasons for the effects of different modifications as well as a way to detect the difficulty characteristics of a game in real-time. A feature based difficulty rating as utilized here can be a step into that direction. Feature-based surrogate models could be employed for predicting which controller should be used for an unknown game after testing a number of actions and events. An interesting further use of the difficulty rating could be to support the selection of a set of games with balanced and distinct challenges for the GVGAI competition.

References

[1] Y. Björnsson and H. Finnsson. "Cadiaplayer: A Simulation-Based General Game Player". In: *IEEE Trans. on Computational Intelligence and AI in Games* 1.1 (2009), pp. 4–15.

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

[3] A. E. Elo. *The Rating of Chessplayers, Past and Present*. B.T. Batsford, London, UK, 1978.

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

[5] F. J. Gomez and R. Miikkulainen. "Solving Non-Markovian Control Tasks with Neuroevolution". In: *Proc. of the International Joint Conference on Artificial Intelligence*. Kaufmann, San Francisco, CA, 1999, pp. 1356–1361.

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

[7] N. Justesen, T. Mahlmann, and J. Togelius. "Online Evolution for Multi-Action Adversarial Games". In: *Applications of Evolutionary Computation*. Springer, Berlin, Germany, 2016, pp. 590–603.

[8] L. Kocsis and C. Szepesvári. "Bandit based Monte-Carlo Planning". In: *Proc. of the European Conference on Machine Learning*. Springer, 2006, pp. 282–293.

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

[10] S. Lucas, S. Samothrakis, and D. Perez. "Fast Evolutionary Adaptation for Monte Carlo Tree Search". In: *Applications of Evolutionary Computation*. Springer, Berlin, Germany, 2014, pp. 349–360.

[11] D. Perez, S. Samothrakis, and S. Lucas. "Knowledge-Based Fast Evolutionary MCTS for General Video Game Playing". In: *Proc. of the IEEE Conference on Computational Intelligence and Games*. IEEE Press, Piscataway, NJ, 2014, pp. 1–8.

[12] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen. "Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games". In: *Proc. of the Conference on Genetic and Evolutionary Computation*. ACM Press, New York, NY, 2013, pp. 351–358.

[13] D. Perez-Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas. "Open Loop Search for General Video Game Playing". In: *Proc. of the Conference on Genetic and Evolutionary Computation*. ACM Press, New York, NY, 2015, pp. 337–344.

[14] 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". In: *IEEE Trans. on Computational Intelligence and AI in Games* (2015).

[15] T. Schaul. "A Video Game Description Language for Model-based or Interactive Learning". In: *Proc. of the IEEE Conference on Computational Intelligence in Games*. IEEE Press, Piscataway, NJ, 2013, pp. 193–200.