Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus

Mark J. Nelson
The MetaMakers Institute
Falmouth University
Penryn, Cornwall, UK
Email: mjn@anadrome.org

***Abstract*—The General Video Game AI Competition (GVG-AI) invites submissions of controllers to play games specified in the Video Game Description Language (VGDL), testing them against each other and several baselines. One of the baselines that has done surprisingly well in some of the competitions is `sampleMCTS`, a straightforward implementation of Monte Carlo tree search (MCTS). Although it has done worse in other iterations of the competition, this has produced a nagging worry to us that perhaps the GVG-AI competition might be too easy, especially since performance profiling suggests that significant increases in number of MCTS iterations that can be completed in a given time limit will be possible through optimizations to the GVG-AI competition framework. To better understand the potential performance of the baseline vanilla MCTS controller, I perform scaling experiments, running it against the 62 games in the public GVG-AI corpus as the time budget is varied from about 1/30 of that in the current competition, through around 30x the current competition's budget. I find that it does not in fact master the games even given 30x the current time budget, so the challenge of the GVG-AI competition is safe (at least against this baseline). However, I do find that given enough computational budget, it manages to avoid explicitly *losing* on most games, despite failing to win them and ultimately losing as time expires, suggesting an asymmetry in the current GVG-AI competition's challenge: not losing is significantly easier than winning.**

I. Introduction

The General Video Game AI Competition (GVG-AI) [1], [2] is a recurring videogame-playing competition intended to stimulate progress in general videogame AI (as distinguished from AI written for a specific game) by testing submitted AI agents against previously unseen videogames. Games used in the competition are written in the Video Game Description Language (VGDL) [3], a domain-specific language designed to capture the variety of arcade-style games in which the rules take the form of sprite movement and interaction on a 2d grid. In the current competition, games are all single-player, and may be deterministic or stochastic. The set of unpublished test games on which agents are scored is periodically replaced with a fresh set of games, to keep agents from having the opportunity to even inadvertently become specialized to a specific reused test set. When a new test set is added, the old one is released publicly. Therefore a corpus of VGDL games has slowly grown, currently at 62 public games (as of April 2016).

The 62 VGDL games distributed with the GVG-AI competition framework do represent a specific subset of games—there is nothing here like chess, Starcraft, or even Tetris. But within the style of arcade games that VGDL targets, they cover a fairly wide range of challenges and characteristics, from twitch-type action games to puzzle games, games with NPCs and without, games with counter-based, spatial, or time-based win conditions, and so on.[1] This makes the corpus useful as a testbed for investigating differences between algorithms and games.

The purpose of this paper is to take an extended look at the performance scaling of one specific algorithm across this GVG-AI corpus: how the play of vanilla Monte Carlo tree search (MCTS) improves, or doesn't, on these 62 games as its computation budget is increased. The goal of doing so is to better understand both sides of the pairing: to use the GVG-AI corpus to look at how MCTS scales with performance across a range of videogames, and also to use the MCTS performance curves as a way of characterizing the nature of the challenges found in the current public set of GVG-AI competition games.

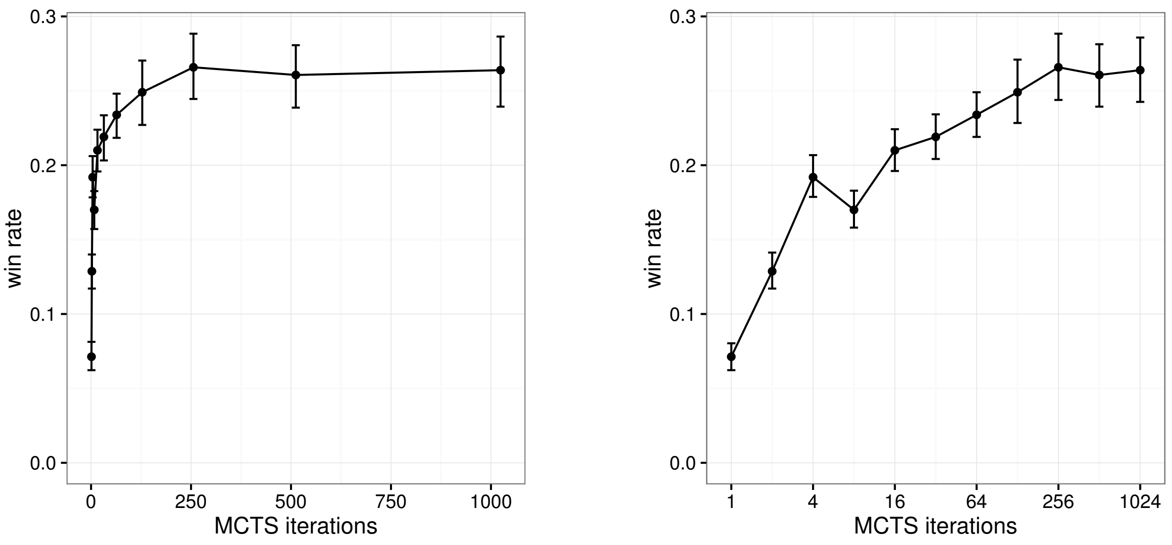
One specific question motivating this scaling experiment was whether the GVG-AI competition might be too easy. In the first GVG-AI competition at CIG 2014, the `sampleMCTS` controller implementing a vanilla MCTS search, included with the SDK and intended as baseline, somewhat surprisingly came in 3rd place, achieving a win rate of about 32%. In the competitions since then it has not done as well, as both its competitors and the test games have gotten more challenging; at CEEC 2015 it did particularly badly, placing 31st with a win rate of only 12%.[2]

The poorer recent results do not entirely dispel the worry about difficulty. The competition tests agents with only one specific time limit, 40 milliseconds per move. That leaves open the possibility that the challenge might be mostly posed by the short time budget. But as hardware gets faster, and the GVG-AI framework becomes more optimized,[3] many more MCTS iterations will be possible to complete within the same 40 milliseconds. Would the competition *then* become too easy, with the `sampleMCTS` agent making quick work of

[1]A feature matrix comparing the first 30 games is given in Table 1 of [1].

[2]Past competition results are taken from the online rankings at http://www.gvgai.net/.

[3]Profiling suggests that a large portion of the computation time in the GVG-AI framework's forward model is taken up by Java collections bookkeeping; significant speedups are likely possible by reworking the code here.



(a) Linear scale

(b) Logarithmic scale

Fig. 1. Number of MCTS iterations versus overall win rate across all trials of the 62 games, shown with both (a) linear and (b) logarithmic x axes. Error bars, here and in subsequent figures, represent 95% confidence intervals, estimated via a nonparametric bootstrap.

the games?

The experiments here find that the answer is no. Performance on the current corpus of 62 games for the `sampleMCTS` agent does improve as computation time is increased from the current limit. But it plateaus at a win rate of 26% when given about 10x the current competition's computation budget, and doesn't further improve beyond that (tested out to around 30x). Therefore we can conclude that even after an order of magnitude increase in speed of the forward model (whether through hardware improvements or optimizations), the GVG-AI competition will still pose challenges requiring something more than vanilla MCTS to tackle them.

I do however find that as MCTS's computation budget is increased, the way it plays in the games it loses changes considerably. In many of the GVG-AI games, failing to achieve a win within the maximum number of timesteps is a loss. This is in addition to more explicit ways of losing, such as dying due to collision with an enemy. As its computation budget increases, vanilla MCTS manages to avoid most of its explicit losses—only to end up losing via timeout instead. This makes sense if we think of many classic arcade games as consisting of two layers of challenge: avoid dying, and while doing so, achieve a goal. Vanilla MCTS becomes much better at the first challenge as it is given time to perform more iterations, but only slightly better at the second one. This suggests that much of the future challenge in the GVG-AI competition lies in the goal-achievement part; vanilla MCTS can mostly avoid dying, but it often nonetheless can't win.

II. Background and Related Work

MCTS [4] is an anytime search algorithm: it has a core iteration loop that can be stopped when it runs out of computation budget, returning the best estimate it has come up with so far. Given more time, it will generally perform better, but existing research has found mixed results regarding what that scaling curve looks like.

Since search trees grow exponentially with depth, there is some heuristic reason to believe that performance will scale with the logarithm of computation time, meaning each doubling of computation time will produce a linear increase in playing strength. This is indeed what some researchers have reported, for example on the game Hex [5]. But other experiments, on Go, have reported diminishing returns with repeated doubling of the computational budget, as performance plateaus rather than continuing to increase linearly [6]. This paper supplements the existing knowledge on MCTS performance scaling by adding results on a fairly large set of single-player games to the existing studies that have focused on specific two-player games.

Besides explicit tests of scaling, which are clearly the most relevant related work, investigating algorithm performance curves has been used for several other purposes in games. Togelius and Schmidhuber [7] propose using the performance curve of a genetic algorithm repeatedly playing a game as a fitness function to measure game quality: good games, they hypothesize, are learnable at a moderate pace, leading to mastery neither too quickly nor too slowly. This is a reinforcement learning formulation rather than a planning formulation, so the x-axis of their performance curve represents numbers of games played, rather than time budget within a single game as here, but nonetheless the ideas are closely related.

Several other researchers have recently used ratios between different algorithms' performance on a game, called a relative algorithm performance profile (RAPP), as part of a game-quality fitness function [8], [9], [10]. This is based on a hypothesis that the strength difference between good and bad

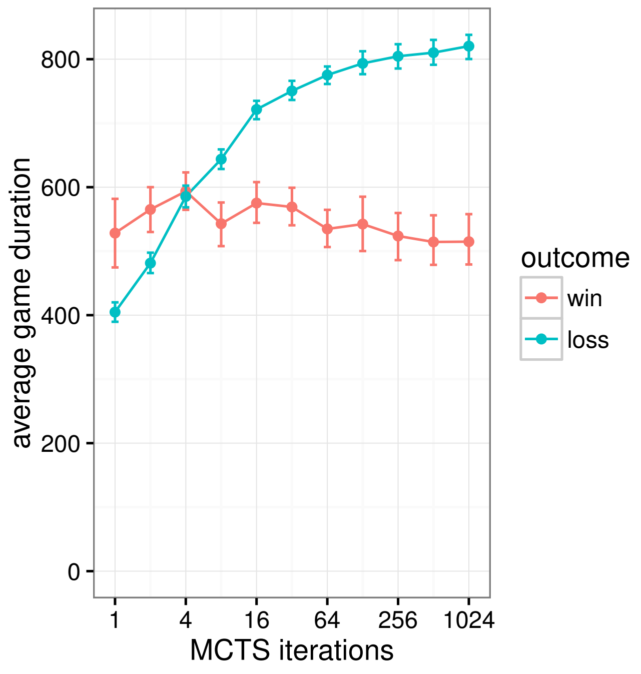


Fig. 2. Average game duration by number of MCTS iterations and win–loss outcome.

agents playing a game can be used as a proxy for game quality, or at least as a filter of bad-quality games: a game on which a random agent performs as well as an agent employing a smarter strategy likely lacks interesting depth. The single-algorithm performance curves I investigate here can be seen as a generalization of RAPPs to look at not only ratios between algorithms with specific settings (such as random play vs. a specific MCTS configuration), but also curves with parameter variation of a single algorithm. Although generating games is not the purpose of the present paper, properties of an MCTS performance curve may be interesting to use as part of a game-generation fitness function. For example, games where MCTS performance plateaus early, versus late, versus increases linearly, may pose different types of challenges (at least, as such challenge is viewed by the MCTS agent).

III. Methodology

Since I'm investigating the scaling curve of MCTS as its computation budget is increased, the basic methodology is simple: have the agent repeatedly play each game in the GVG-AI corpus starting from a small computation budget, then double the computational budget and re-run the same trials on all the games.

Tests were run using a minor variant of the GVG-AI framework, starting from the April 4, 2016 revision in its GitHub repository.[4] I modified the framework to use a limit on MCTS iterations, rather than a time limit, as the method of time budgeting. This change was made in order to make the tests more reproducible. A time limit of, for example, 40 milliseconds or 80 milliseconds per move only produces comparable data if run on exactly the same hardware, with system load and other factors held constant. Since I'm running a large number of trials, it was convenient to use cloud-computing resources to carry out the experiments, an environment where it's not possible to assume that every trial will be run on identical hardware with identical system load. A time budget of, for example, 32 or 256 MCTS iterations, on the other hand, is completely reproducible on any hardware.

As a point of reference to anchor the MCTS iteration counts in this paper with the wall-clock time limit of 40 milliseconds in the GVG-AI competition, in my tests I found that 40 ms is enough time to run around 30 MCTS iterations on average, although this varies significantly depending on the specific game and hardware. Therefore, since the tests in this paper run from 1 through 1024 iterations, they represent a time budget starting at about 1/30 of the current competition's time budget, and going up to about 30x.

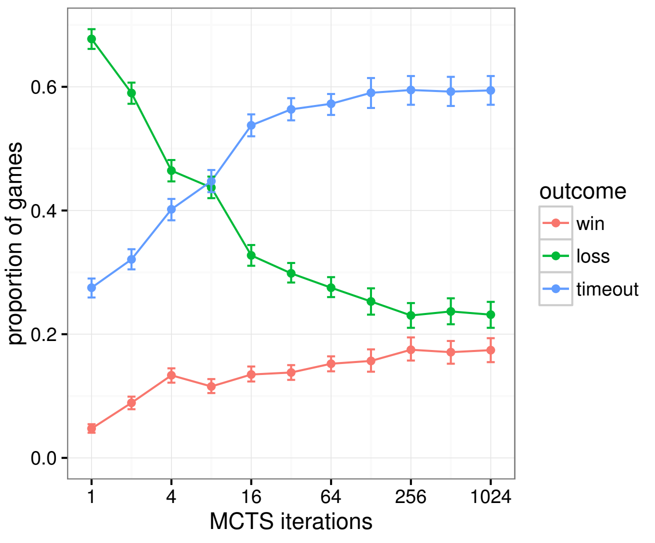
The specific implementation of vanilla MCTS I test here is the one included with the GVG-AI SDK as the `sampleMCTS` controller, and described in [1]. It uses an exploration-exploitation constant of $\sqrt{2}$ and a play-out depth of 10 moves, with cutoff states evaluated by giving them a large positive score if a win, large negative score if a loss, and otherwise the current point value. I chose this controller because it is already used as a baseline in the GVG-AI competition, widely available to every GVG-AI competition participant, and has reported competition results going back several years; therefore what happens to its performance if the current competition's time budget were significantly increased serves as a useful baseline. There are of course many other things about this baseline that could be varied; here I vary only the number of iterations, not any other parameter choices, though doing so would be interesting future work.

The GVG-AI game corpus includes 62 games, each of which comes with 5 levels. I run 10 trials of each level, i.e. 50 of each game, for the tests with MCTS iteration limits of 1 through 64, and (to reduce computational resources needed) 5 trials of each level, or 25 per game, for the 128-iteration through 1024-iteration experiments. The difference is visible in the slightly larger error bars on the higher-iteration-count data points in Figures 1–3; the data is otherwise completely comparable. For each of these trials, I record whether the outcome was a win or loss, and the number of timesteps to end of game. Games are allowed to run for a maximum of 1000 timesteps. The GVG-AI framework also defines a point score, but we don't use it here, since the MCTS controller is mainly trying to maximize its wins vs. losses rather than playing for points, and comparing points across games is often not very illuminating in any case.

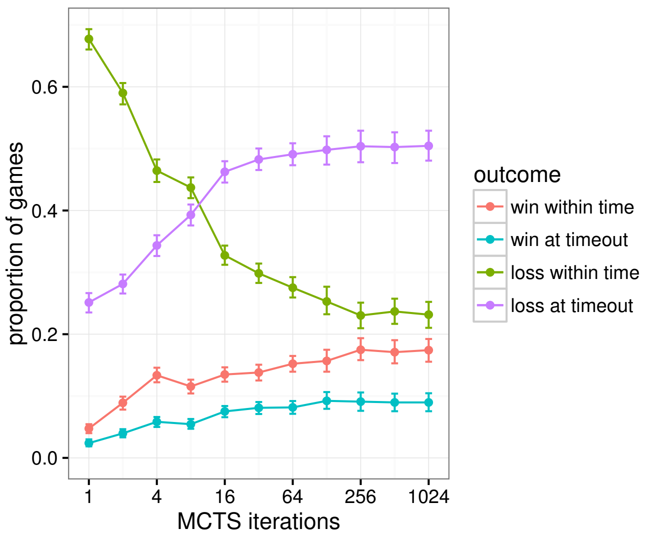
IV. Results

The top-level result is shown in Figure 1, plotting win rate (across all trials of all games) versus MCTS iteration limit. This shows a rapid increase in win rate initially, as the iteration limit is increased from its very low starting point of 1, followed

[4]https://github.com/EssexUniversityMCTS/gvgai



(a)



(b)

Fig. 3. Same data as shown in Figure 1, but reinterpreted as either 3- or 4-valued outcome rather than the 2-valued win/loss outcome. In (a), the three outcomes are win, loss, or timeout. In (b), whether a game is a win or loss is crossed with whether it was a timeout or not, producing 4 possible outcomes.

by diminishing returns up towards a plateau of about 26% past the 256-iteration trials.

A. *Overall scaling*

The version of the win-rate curve with a logarithmic x axis (b) gives a more detailed look at the scaling properties. The general expectation that MCTS should scale logarithmically with increased iterations is partly confirmed: the scaling does look logarithmic, i.e., linear on the log-axis graph, up until the plateau past 256, although with some anomalies. There is a higher slope from 1-4, followed by a lower slope from 16 to 256, interrupted by a strange but significant dip in performance at 8. The reason for this dip is not entirely clear, but is visible across a number of specific games as well, when looking at the results broken down per-game in Figure 4.

The main takeaway from the top-level results is therefore that overall performance of the `sampleMCTS` agent would improve only modestly if the GVG-AI competition's time budget allowed for more than the current 30 or so MCTS iterations, either through increasing time budget or (more likely) optimizing the code for rollouts: from around 22% to 26% overall win rate. Therefore, my initial hypothesis that the GVG-AI competition's challenge might be somewhat illusory, due mainly to the small time budget and unoptimized code, is not confirmed: the baseline `sampleMCTS` agent would not start dominating the competition even if its time budget were increased by 10x+. And based on the clear plateau in performance, it is unlikely to do so even if it were given still more time than that.

B. *Types of wins and losses*

While running these experiments, I noticed that experiments with higher iteration limits were taking much longer to complete, by more than would be expected just from the increased number of MCTS iterations. Investigating further, the reason appears to be that when given more computation time, the MCTS agent plays much longer games. And specifically, its losses drag out for longer, while its wins stay at about the same length, as shown in Figure 2.

In fact not only is the MCTS agent playing longer games in its losses as its time budget increases, but an increasing proportion of losses come right at the maximum length limit of 1000 timesteps. This suggests that significant qualitative changes of play are taking place that are somewhat masked what looking only at the win-rate results (since long losses and short losses are still losses). One way of making these changes more visible in the overall results is to change from scoring games on a 2-outcome scale of win–loss, to a 3- or 4-outcome scale that treats timeouts when the maximum game length is reached differently.

Figure 3 reinterprets the overall win-rate data using two alternate ways of treating timeouts. On the left (a), a 3-outcome scoring is used, with games resulting in either a *win* (within time), a *loss* (within time), or a *timeout*, treated as different from either a win or loss. From this it can be seen that while wins slowly increase, the biggest swing in overall performance is that, given more computation time, the MCTS agent manages to convert losses into timeouts. On the right (b), the timeouts are further broken down into wins at timeout and losses at timeout, which shows that the biggest swing is specifically from losses within time, to losses at timeout.

From this view of the data, the MCTS agent can actually be said to come close to mastering the GVG-AI games if the goal were *not to lose*, with timeouts treated as non-losses: while its win rate plateaus at a not-that-impressive 26%, it brings its overall loss-within-time rate down to a mere 22%, i.e. it manages to avoid explicitly losing (except by timeout) in 78% of games. This quite large gap in two ways of looking at what constitutes good performance leads to a hypothesis that the current GVG-AI competition has two distinct types of challenges, and a vanilla MCTS agent can master one but not the other: first, avoid losing, and then, figure out how to win. As one of this paper's anonymous reviewers aptly pointed out, however, what a timeout means varies by type of game, sometimes indicating partial mastery and other times, especially in puzzle games, not indicating much success at all: "It's an accomplishment to stay alive in Pac-Man even if you don't eat all the dots, but it's not an accomplishment in Sokoban if you put the blocks in an unwinnable state and then run out the clock".

Further study would be needed to clarify the nature of the challenges posed by the different games. The fact that controllers do get better at avoiding explicit losses suggests that there is challenge involved in doing so, but it may be that in most of the GVG-AI games, winning requires a more complex policy than avoiding explicit losses does, which the controller is unable to find. For example, avoiding explicit losses in some of the games requires only short-term reactive behavior such as avoiding an enemy, while winning requires putting together a sequence of steps to achieve a goal. The sampleMCTS controller's 10-depth search cutoff would further make it entirely unable to find winning plans in games requiring longer sequences of action. This finding also suggests that the choice of whether a GVG-AI game should result in a win or loss at timeout has a significant effect on the challenge posed, if judged by the headline win–loss rate; most of the current GVG-AI games result in a loss at timeout, but a few result in a win.

The different insight into performance given by looking at only wins and losses, as in Figure 1, and by treating timeouts separately, as in Figure 3, suggests that future analyses of algorithm performance on the GVG-AI corpus may want to report both measures, in order to provide a fuller view of the algorithm's playing strengths and weaknesses.

C. Per-game results

Figure 4 breaks the results down for each of the 62 games, showing win rate, represented by brightness of the table entry, as MCTS iterations increase. The table is ordered from top to bottom by the sum of win rates across all trials for that game, i.e. games at the top are overall won by the MCTS agent more often than those at the bottom.

A few noticeable aspects are worth pointing out. First, a relatively small number of games, about a dozen, are the only ones that really differentiate performance of the lower-iteration and higher-iteration MCTS agents. Most games are insensitive to iterations: in more than half, the MCTS agent rarely wins, regardless of time budget, and in a few, it mostly wins even with small time budgets. Only a few games, such as `seaquest` and `racebet2`, seem to present the MCTS agent with a difficulty curve, where it performs better as it computes more. And a few games are completely mastered with increasing iterations: the most stark cliff is for `intersection`, which reaches an 100% win rate at mere 4 iterations, but is not so easy that it can be mastered with only 1 or 2 iterations. The odd decrease in performance at 8 iterations observed in the overall results can also be seen in a number of individual games here, such as `plaqueattack` and `sheriff`.

The rather few number of games that show performance differentiation has implications for uses of algorithm performance profiles as a fitness function in game generation. Algorithm performance profiles are only a meaningful stand-in for difficulty curves or challenge depth if we can blame a lack of performance differentiation in a specific game on the *game* rather than the *algorithm*. The many GVG-AI games on which vanilla MCTS fails to achieve any win rate at all implies it may not be a great candidate for use in such fitness functions, since it will reject many games which do have challenge depth but which it is simply not able to play.

V. Conclusions and Future Work

The experiments in this paper investigate scaling the baseline vanilla MCTS controller used in the GVG-AI competition from about 1/30 of the current competition's time budget through about 30x of the current competition's time budget. The two purposes of doing so were to better understand MCTS scaling properties, using the 62 games in the GVG-AI corpus as the testbed, and to better understand the challenge posed by the GVG-AI competition.

Regarding scaling, I found performance increased roughly with the logarithm of increased computation time, although with some anomalies in the slope, up to a plateau at around 256 MCTS iterations, after which performance did not increase further. The plateau implies that even massive amounts of computational power would be insufficient to solve the challenges in the current competition using the baseline MCTS agent. This may be due to the 10-depth cutoff in the Monte Carlo rollouts, or it may be due, as Perez *et al.* [1] hypothesize, to suboptimal estimates produced by closed-loop MCTS on stochastic games.

The initial skeptical hypothesis regarding the GVG-AI competition's medium-term challenge is refuted by these results. I had worried that the competition might really be too easy; that if rollouts were optimized so that agents were given, say, 10x the effective time budget they have now, the competition might become trivial, with the baseline controller mastering the games. However, I find that is not the case. Instead, performance plateaus past 256 MCTS iterations, which is roughly 10x the time budget of the current GVG-AI competition, and furthermore plateaus at only a 26% win rate. Therefore the GVG-AI competition would remain interesting, at least relative to the current baseline, even if orders of magnitude more computation time were given to the agents entering the competition, through either an increase in time budget or optimizations to the framework.

I do however find that quite a lot of change in gameplay is happening beneath the headline win-rate data. As time budget is increased, MCTS takes longer to lose the games that it's going to lose. And, it often loses them in a qualitatively different way, by hitting the maximum game length (which

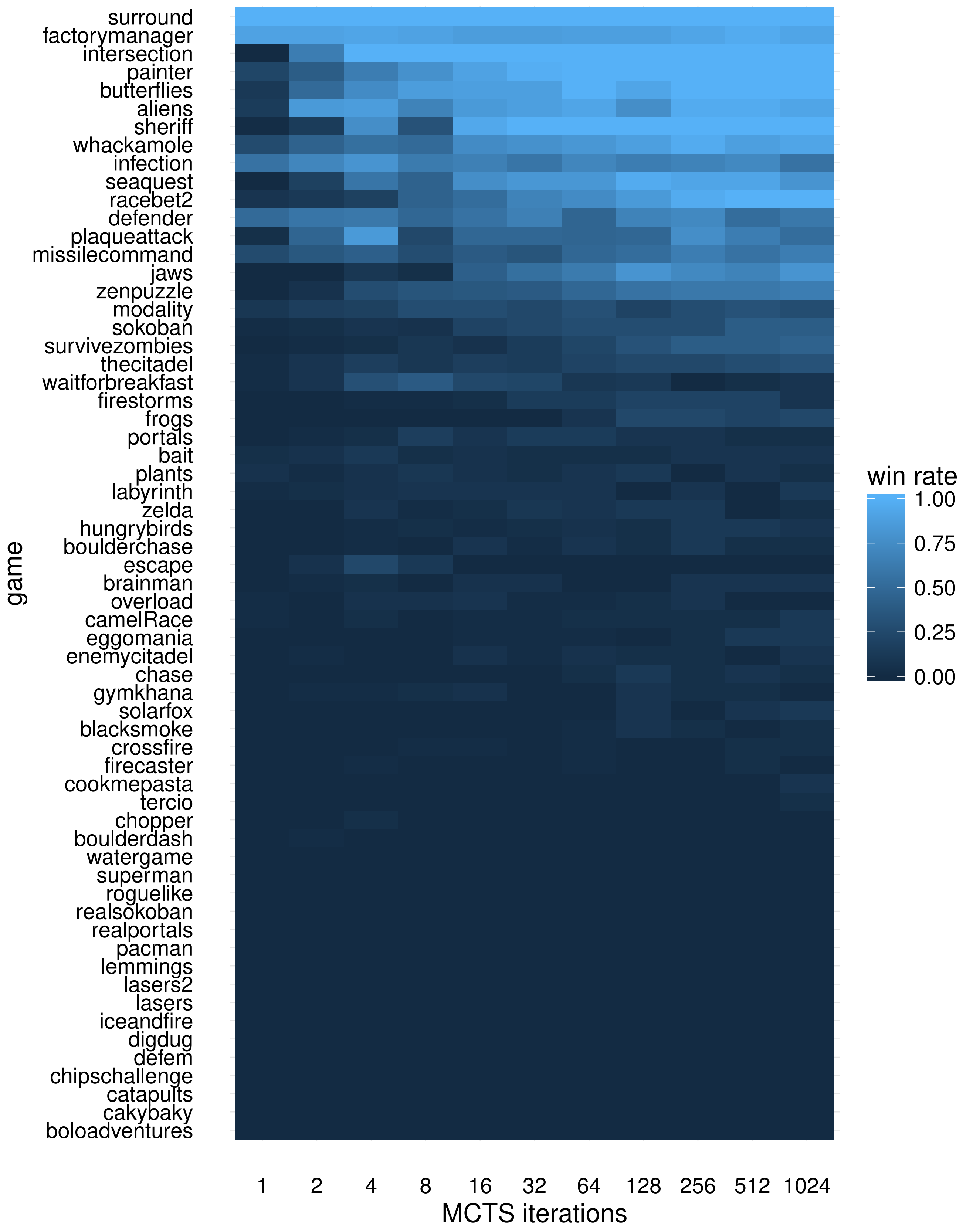


Fig. 4. Win rates for each of the 62 individual GVG-AI games, as number of MCTS iterations is increased. Brighter table entries represent higher win rate. Games are sorted by sum of win rates across MCTS iterations.

in many GVG-AI games is a loss through timeout), rather than succumbing to one of the explicit loss conditions, such as colliding with an enemy. When the overall results are reinterpreted using a 3-valued outcome of win, loss, or timeout, the vanilla MCTS agent does manage to avoid losses in the majority of games, at the larger time budgets, bringing its "non-loss" rate up to 78%—much more impressive than its 26% win rate—with the majority of games ending in a timeout.

This suggests to us two conclusions. First, it may be helpful to look beyond the headline win rate in the GVG-AI competition when comparing agents, and treat games that timeout differently from those where a win or loss is recorded within time. This provides a fuller view of what the agent is doing, and which kinds of challenges it is succeeding or failing at. Secondly, it suggests that in the current corpus of GVG-AI games, the challenge is structured so that not-losing is easier than winning. Further work might help clarify why this is; for example, it may be that in the majority of the games a relatively simple controller can avoid losing, perhaps with even a very simple reactive policy like "move away from enemies", but more complex planning may be needed to achieve win conditions. One approach to confirming whether this is the case could be to solve for optimal policies for several of the games and investigate their structure.

Future work should look at more algorithms beyond the one baseline whose scaling properties I test here. In this paper, only the time budget in the baseline MCTS controller is varied, but there are many variations of MCTS and quite a few other parameters that can be varied. MCTS can also be compared on this corpus with the scaling of other algorithms, such as traditional full-width search.

Acknowledgment

This work was supported in part by the European Union FP7 grant 621403 (ERA Chair: Games Research Opportunities). Thanks to the reviewers for helpful comments and suggestions.

References

[1] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couëtoux, 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.

[2] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. Lucas, "General video game AI: Competition, challenges and opportunities," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016, pp. 4335–4337.

[3] T. Schaul, "An extensible description language for video games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 325–331, 2014.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[5] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo tree search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[6] A. Bourki, G. Chaslot, M. Coulm, V. Danjean, H. Doghmen, J.-B. Hoock, T. Hérault, A. Rimmel, F. Teytaud, O. Teytaud, P. Vayssière, and Z. Yu, "Scalability and parallelization of Monte-Carlo tree search," in *Proceedings of the 7th International Conference on Computers and Games*, 2010, pp. 48–58.

[7] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 111–118.

[8] T. S. Nielsen, G. A. B. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *Proceedings of the 18th Conference on Applications of Evolutionary Computation*, 2015, pp. 369–380.

[9] ——, "Towards generating arcade game rules with VGDL," in *Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games*, 2015, pp. 185–192.

[10] J. Kowalski and M. Szykuła, "Evolving chess-like games using relative algorithm performance profiles," in *Proceedings of the 19th Conference on Applications of Evolutionary Computation*, 2016, pp. 574–589.