The First MicroRTS Artificial Intelligence Competition

Santiago Ontañón, Nicolas A. Barriga, Cleyton R. Silva, Rubens O. Moraes, Levi H. S. Lelis

■ *This article presents the results of the first edition of the microRTS AI competition, which was hosted by the IEEE Computational Intelligence in Games (CIG) 2017 conference. The goal of the competition is to spur research on AI techniques for real-time strategy (RTS) games. In this first edition, the competition received three submissions, each addressing problems such as balancing long-term and short-term search, using machine learning to learn how to play against certain opponents, and finally, dealing with partial observability in RTS games.*

The first microRTS (μRTS) AI competition was hosted at the IEEE Computational Intelligence in Games (CIG) 2017 conference. The goal of the competition is to spur research on artificial intelligence techniques for real-time strategy (RTS) games.

RTS games are a genre of video games that are notoriously difficult for AI techniques (Buro 2003) since, compared to traditional board games such as Chess or Go, they have a very large state space and a very large branching factor. Additionally, these games are executed in real time, which leaves little time to decide a move, and they are often nondeterministic and partially observable. For these reasons, and as evidenced by other AI competitions organized around RTS games (such as the StarCraft AI competitions [Ontañón et al. 2013]), human players are still much stronger players than AI bots. Compared to the StarCraft AI competitions, the goal of the μRTS competition is to focus on the key underlying research problems that RTS games feature from an AI point of view, rather than on solving the many engineering problems that apply when dealing with a full-fledged commercial RTS game like StarCraft.

Copyright © 2018, Association for the Advancement of Artificial Intelligence. All rights reserved. ISSN 0738-4602

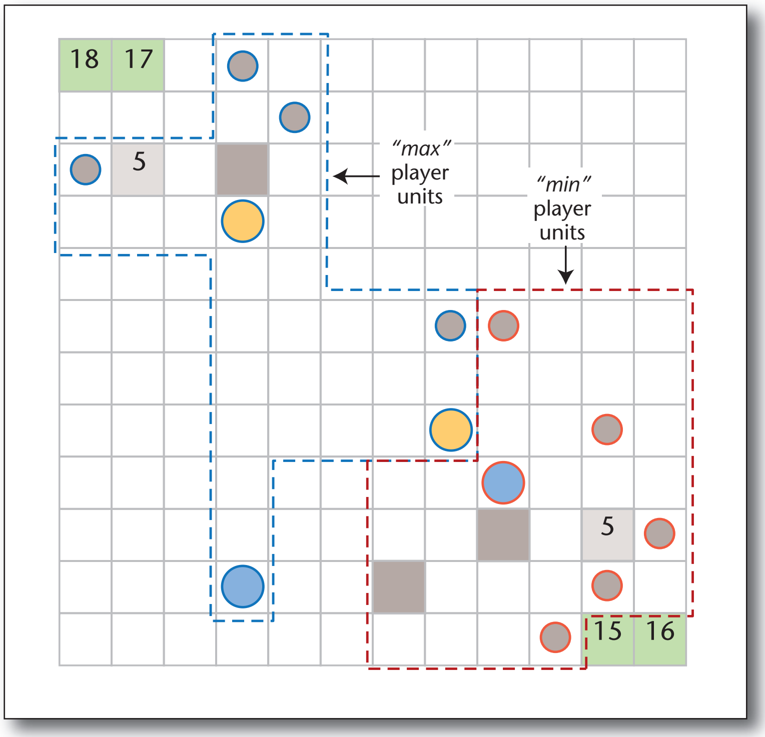

Figure 1. A Screenshot of the μRTS Simulator.

Square units correspond to bases (light grey) that can produce workers, barracks (dark grey) that can produce military units, and resource mines (green) from which workers can extract resources to produce more units. Circular units correspond to workers (small, dark grey) and military units (large, yellow or light blue).

In this article, we will describe μRTS, emphasizing the research questions it raises, the rules of the competition, the bots submitted to the competition, and an analysis of the competition results.

μRTS

μRTS[1] is a minimalist implementation of a two-player RTS game designed specifically for AI research. Figure 1 shows a screenshot of μRTS. Players in μRTS play in a rectangular grid-based map of arbitrary size and control a set of units with the goal of destroying all the units of the opponent. μRTS features a reduced set of units compared to other RTS games such as StarCraft, but this is configurable, so more unit types can be added if desired. The basic unit types include bases and barracks (shown as square units in the figure), which are buildings that cannot move but that can produce other units; workers (shown as small circular units in the figure), which can gather resources and create new buildings; and military units (shown as larger circular units in the figure). Players need to gather resources (green squares in the figure) and use them to expand either their economy (build more workers, bases, or barracks) or their military force.

From an AI point of view, μRTS raises some of the same theoretical challenges raised by other, more complex RTS games. μRTS has a huge decision space. The branching factor of games like Chess or Go has been estimated to be about 35 and 180, respectively. By contrast, in μRTS, branching factors of over 1022 have been reported (Ontañón 2017) for 16 × 16 maps. The branching factor in the game state shown in figure 1, for example, is 1,008,288 for the max player and 1,680,550 for the min player. μRTS also has real-time constraints, meaning that the game is designed to execute about 10 decision cycles per second, leaving players with just a fraction of a second to decide the next action. Moreover, players can issue actions simultaneously, and actions are durative.

μRTS can be configured to be deterministic or nondeterministic. When it is configured to be nondeterministic, if two units issue contradicting actions simultaneously, one action is executed at random and the other is canceled. The damage that units deal each other when attacking is also stochastic.

μRTS can be configured to be fully or only partially observable. When it is configured to be partially observable, each unit has a predefined sight range, and a player can see only those parts of the map that her units can see. This is often referred to as the fog of war.

Finally, μRTS comes with a built-in forward model that can be used by the bots to simulate the effect of actions. This simplifies the development of search-based techniques (as explored, for example, by Churchill and Buro [2013]; Justesen et al. [2014]; Ontañón [2017]; Barriga, Stanescu, and Buro [2017b]; Lelis [2017]). However, while μRTS does not encourage learning forward models (as shown, for example, by Uriarte and Ontañón [2017]) for domains where they are not already available or developing techniques that do away with the need for forward models altogether, we would like to point out that these lines of work are important research directions both for RTS game AI and for game AI in general. This is the case because in most real-world situations where these algorithms might find application, it is likely that those forward models will not be available.

Previous research in μRTS has focused on areas such as Monte Carlo tree search (MCTS) (Ontañón 2013; Ontañón 2017, 2016; Shleyfman, Komenda, and Domshlak 2014; Komenda, Shleyfman, and Domshlak 2014); adversarial search algorithms that perform search at some level of abstraction, such as Puppet Search (Barriga, Stanescu, and Buro 2017b) or adversarial HTN planning (Erol, Hendler, and Nau 1994); and even deep learning for RTS games (Stanescu et al. 2016).

In summary, we can see that μRTS has been used mainly to study the scalability problems that arise in RTS games. The goal of having several tracks in the μRTS competition was to also spur research into two of the other main problems in RTS games: nondeterminism and partial observability. We detail the tracks

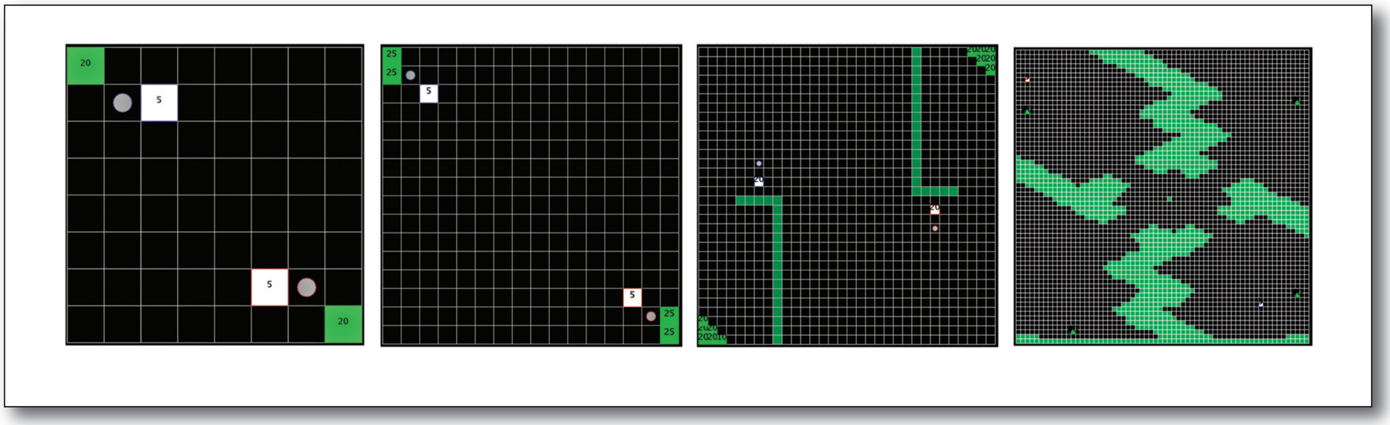

Figure 2. The Four Open Maps Used in the Competition.

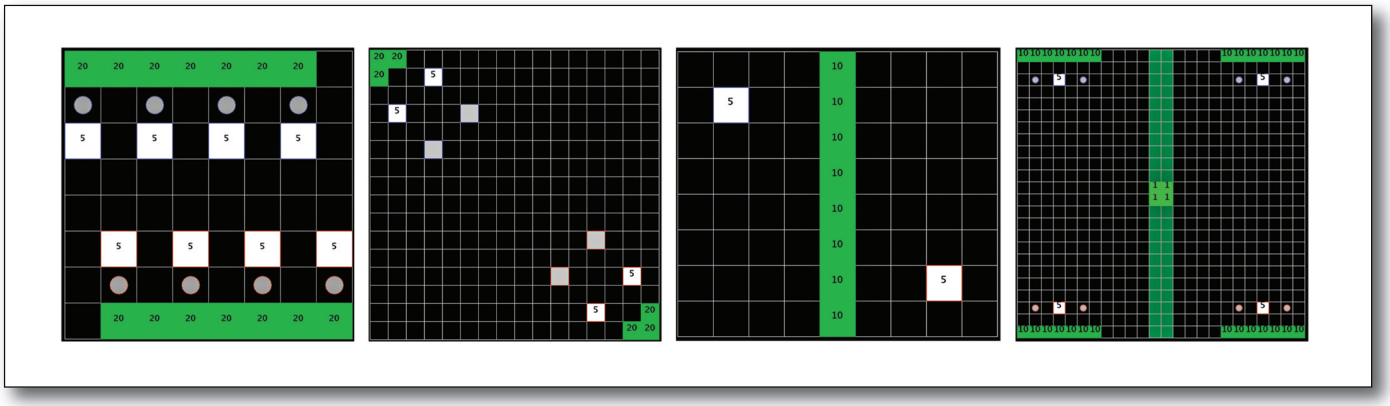

Figure 3. The Four Hidden Maps Used in the Competition.

available in the competition, as well as the rules for each, in the section that follows.

Competition Rules

The competition was organized around three separate tracks: standard, nondeterminism, and partial observability. Each track was run as a series of round-robin tournaments, where each bot played against all other bots in a collection of eight different maps. Five round-robin tournaments were run in each of the eight maps, for a total of 40 round-robin rounds per track.

Each game in the competition was a full-game, two-player match. Each bot was given a computation budget of 100 milliseconds per game cycle. Multiple threads were allowed, but once the 100 milliseconds were up, all threads were stopped so as to prevent one bot from using the CPU during the other bot's time. An upper limit was set on the length of games and if a game went beyond that limit, it was considered a draw. The three tracks were the standard track (fully observable and deterministic); the nondeterminism track (fully observable, but nondeterministic); and the partial observability track (deterministic, but partially observable using the concept of fog of war, common to RTS games).

Eight maps were used for the competition. Four of them were known beforehand to all participants (open maps), and four of them were kept secret (hidden maps). Results are reported below both on the open maps and on the set of all maps. Figures 2 and 3 show the open and hidden maps, respectively. As the figures show, the open maps represented conventional, RTS game-start situations, whereas the hidden maps represented nonconventional start positions. For example, in the last hidden map, bots had to play two parallel games and win both to win the match. The open maps are numbered 1 through 4; the hidden maps, 5 through 8.

To assess the performance of the submissions, in addition to the competition entries, we included five preexisting bots: (1) RandomBiased, a bot that picks actions randomly (but with a bias to attack or harvest); (2) POWorkerRush, a hard-coded bot that always uses one worker to harvest resources, while constantly training and sending the remaining workers to attack; (3) POLightRush, a hard-coded bot that also uses one worker to harvest resources, following which it builds a barracks and constantly trains and sends light units to attack; (4) NaïveMCTS, an MCTS bot that uses Naive Sampling (Ontañón 2017); and (5) PuppetSearch, a bot that uses nondeterministic scripts as the basis for search and then does adversarial search by considering only the choice points offered by the scripts (Barriga, Stanescu, and Buro 2017b).

Competition Entries

The 2017 competition received three submissions: StrategyTactics,[2] Strategy Creation through Voting (SCV),[3] and BS3NaïveMCTS.[4] We describe each of the submissions in the following paragraphs.

Entry 1: StrategyTactics

StrategyTactics (Barriga, Stanescu, and Buro 2017a) builds on the strategic strengths of PuppetSearch (Barriga, Stanescu, and Buro 2017b) and the tactical performance of NaïveMCTS (Ontañón 2017). Most strategic algorithms suffer from weak tactical decisions due to the coarse abstractions necessary for long-term planning. Conversely, algorithms that make good low-level decisions usually don't scale well to larger scenarios with higher branching factors or long-term action consequences.

StrategyTactics splits the search time between both algorithms. First, a reduced game state is generated, one which includes only those units from both sides that are at most one tile away from the sight range of opponent units. Then, if this game state is empty, PuppetSearch is executed as usual on the full game state. Alternatively, if the reduced state has units, PuppetSearch is run for 20 percent of the time available on the complete game state, and then NaïveMCTS is executed for the rest of the time on the reduced game state. The actions this second search produces then replace the ones produced by PuppetSearch. In summary, PuppetSearch produces actions for all units and NaïveMCTS then refines the actions of units in combat.

Entry 2: Strategy Creation Through Voting (SCV)

Strategy Creation Through Voting (SCV)[5] uses a set of base strategies P_{base} to create a larger set of novel strategies P. It does this by using a voting scheme with subsets of base strategies. For example, a subset of base strategies $\{A, B, C\}$ of P_{base} defines a novel strategy by deciding on an action for each unit u through voting (A, B, and C each vote on one action for each unit). Draws are broken deterministically assuming an ordering of the strategies from most to least preferred.

The initial set of strategies P_{base} used in the competition contains 10 strategies similar to the bots RandomBiasedAI, POWorkerRush, and POLightRush described earlier. P contains all subsets of P_{base} with size 1 to 4, which resulted in 385 novel strategies. We do not use the powerset of P_{base} as SCV's P because that would substantially increase the algorithm's training time.

SCV assumes a set of opponent types as well as a set of map types. During the match, SCV predicts the opponent and map types based on what it observes. Then, SCV selects from P the strategy that maximizes its end-game value for the predicted opponent and map types. The set of opponent types included hard-coded bots such as RandomBiasedAI, POWorkerRush, and POLightRush as well as search-based bots such as PuppetSearch and NaïveMCTS. The set of map types included the maps shown in figure 2 — map1 (8×8), map2 (16×16), map3 (32×32), map4 (64×64) — and a map that is similar to map3 but with size 24×24.

SCV has an offline training phase in which it tests all 385 strategies against all predefined opponents in all predefined maps, as well as features of the game state every 100 game cycles. At run time, SCV runs its classifier every 100 game cycles to predict the type of opponent it is playing against and which map is being used, so as to choose the strategy with the largest expected end-game value.

Entry 3: BS3NaïveMCTS

BS3NaïveMCTS (Uriarte and Ontañón 2016) is an extension of NaïveMCTS to handle partial observability in RTS games. To the best of our knowledge, BS3NaïveMCTS is the first attempt to deal with partial observability in RTS games in the context of game tree search.

BS3NaïveMCTS assumes that both players know the initial board configuration. With this assumption, BS3NaïveMCTS uses the idea of determinization (Corlett and Todd 1986; Whitehouse, Powley, and Cowling 2011): infer the whole game state (the believe state) from the parts of it that can be observed, and then perform game tree search as if we were in a fully observable scenario.

To generate the believe state, BS3NaïveMCTS records the position of all the enemy units that become visible during the game, and remembers their position even if we are not currently seeing them. Additionally, it performs a limited form of inference to infer units that must be there but we have not seen (for example, having melee units implies having barracks).

Competition Results

This section presents the results of each of the three tracks of the competition.

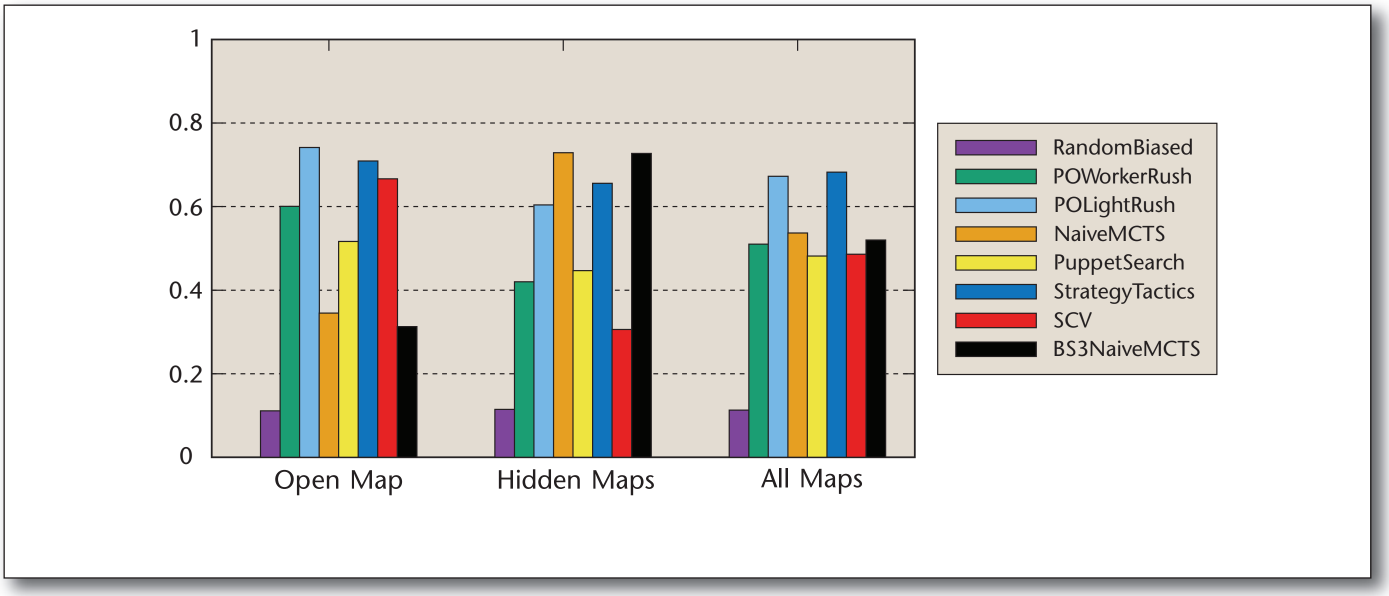

Figure 4. Win Ratios of the Bots in the Standard Track, by Map Type.

Standard Track

Eight bots were used for the standard track: the five preexisting bots (RandomBiased, POWorkerRush, POLightRush, NaïveMCTS, PuppetSearch) and all three competition entries (StrategyTactics, SCV, BS3NaïveMCTS).

Each round robin tournament consisted of 8 × 7 = 56 games (since we discarded self-play matches), and we performed five full round-robin tournaments in each of the eight maps, for a total of 5 × 8 × 56 = 2, 240 games.

Figure 4 shows the win ratios achieved by each of the bots in this track, organized by type of map. The bot that achieved the highest win ratio over all maps was StrategyTactics, with a win ratio of 0.682. The second-best bot in this scenario was POLightRush, one of the preexisting bots, with a win ratio of 0.672. Figure 4 also shows the win ratio averaged over the open and the hidden maps, and, as can be seen, the win ratios of the different bots vary widely between those two.

We can draw a few interesting observations from these results. First, hard-coded bots perform very well in maps that capture standard situations (for example, where the game starts with a base and some workers, and the strategy hard-coded into the bots applies). This can be seen by the great performance of POWorkerRush and POLightRush in the open maps. These hard-coded bots perform poorly, however, in situations where their scripts do not apply, as can be seen by their lower performance on the hidden maps. This turned out particularly to be the case with map7.

Conversely, game tree search excels precisely in nonstandard situations, where the lack of appropriateness of the hard-coded bots can be exploited. This can be seen by the performance of NaïveMCTS, BS3NaïveMCTS, and StrategyTactics on the hidden maps. However, some game tree search bots (NaïveMCTS and BS3NaïveMCTS) struggle in larger maps like map3 and map4, likely because they are unable to search deep enough for the size of the map.

StrategyTactics achieved the highest win ratio overall in all maps, since it was robust enough to perform well in most maps (except map7), whereas other bots performed less consistently. StrategyTactics's high win ratio was due to its mix of high-level and low-level search.

SCV performed extremely well on standard maps — and most especially on map3 and map4, likely due to its being trained on these map types. However, it struggled in nonstandard maps, which hurt its overall win ratio.

Interestingly, although StrategyTactics is an integration of PuppetSearch and NaïveMCTS, SCV performed better than PuppetSearch on the open maps, where high-level search is more important. This result indicates that a StrategyTactics-like bot that integrates SCV with NaïveMCTS could potentially outperform this year's competition bots. Additionally, SCV used very little computation time, and thus, it would lend itself very well to such integration.

Another factor that plays an important role in the performance of the bots is the size of the map. Figure 5 shows the win ratio of the different bots in this track plotted as a function of map size. We can see clearly that the performance of game tree search bots (NaïveMCTS and BS3NaïveMCTS) decreases as the

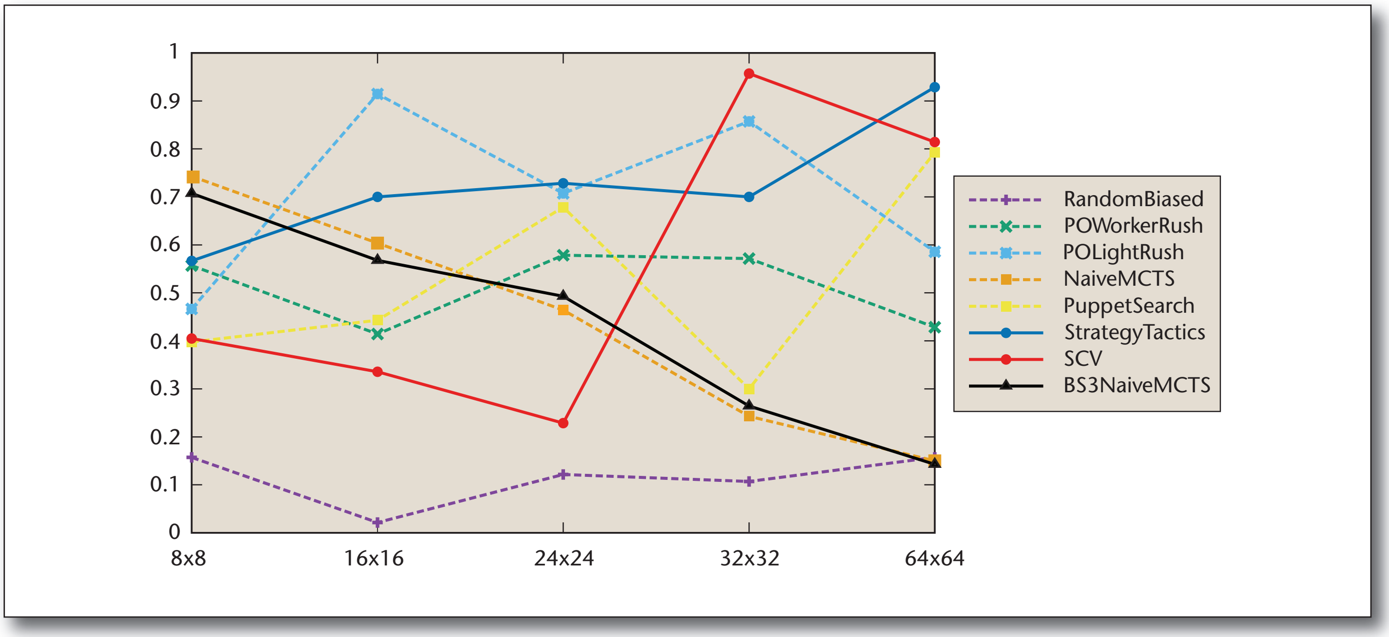

Figure 5. Win Ratios of the Bots in the Standard Track Plotted as a Function of Map Size.

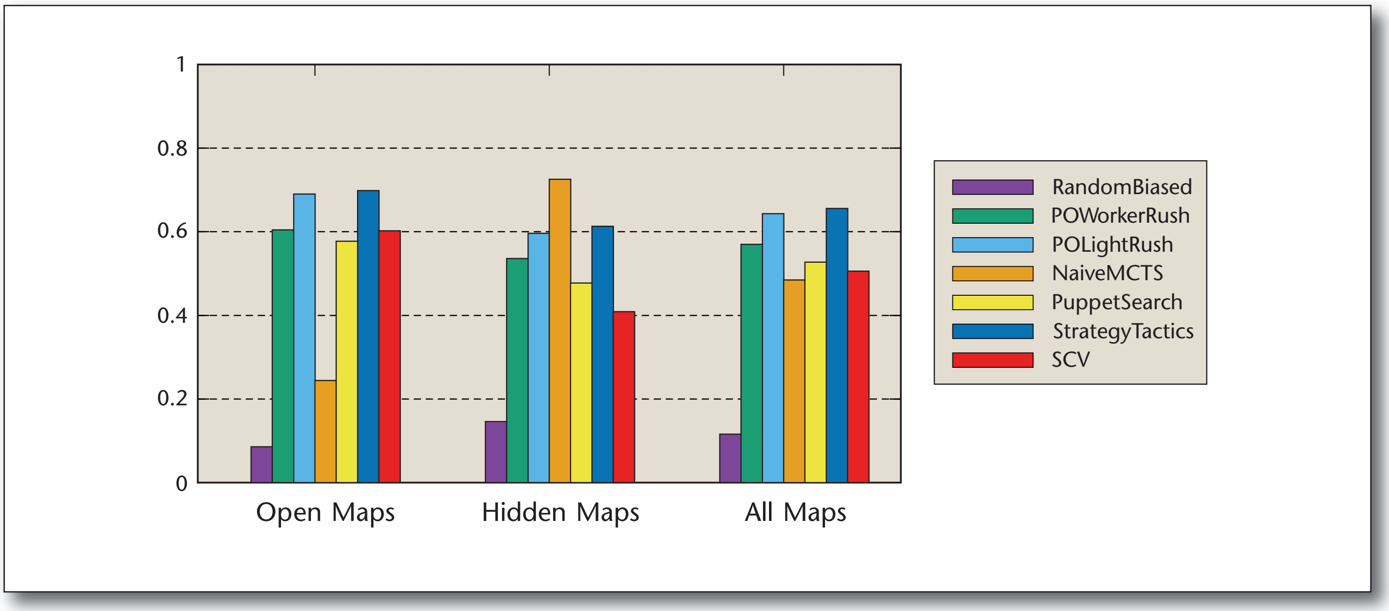

Figure 6. Win Ratios of the Bots in the Nondeterminism Track, by Map Type.

map grows in size, while the performance of bots that have more high-level reasoning capabilities (StrategyTactics and SCV) increases.

Nondeterminism Track

Seven bots were used for the nondeterminism track: the five preexisting bots (RandomBiased, POWorkerRush, POLightRush, NaïveMCTS, PuppetSearch) and two of the competition entries, StrategyTactics and SCV. Each round robin tournament consisted of 7 × 6 = 42 games (since we discarded self-play matches), and we performed five full round-robin tournaments in each of the eight maps, for a total of 5 × 8 × 42 = 1,680 games.

Figure 6 shows the win ratios achieved by each of the bots in this track, organized by type of map. The bot that achieved the highest win ratio over all maps was StrategyTactics, with a win ratio of 0.655. The second-best bot in this scenario was again POLightRush, with a win ratio of 0.643. Figure 6 also

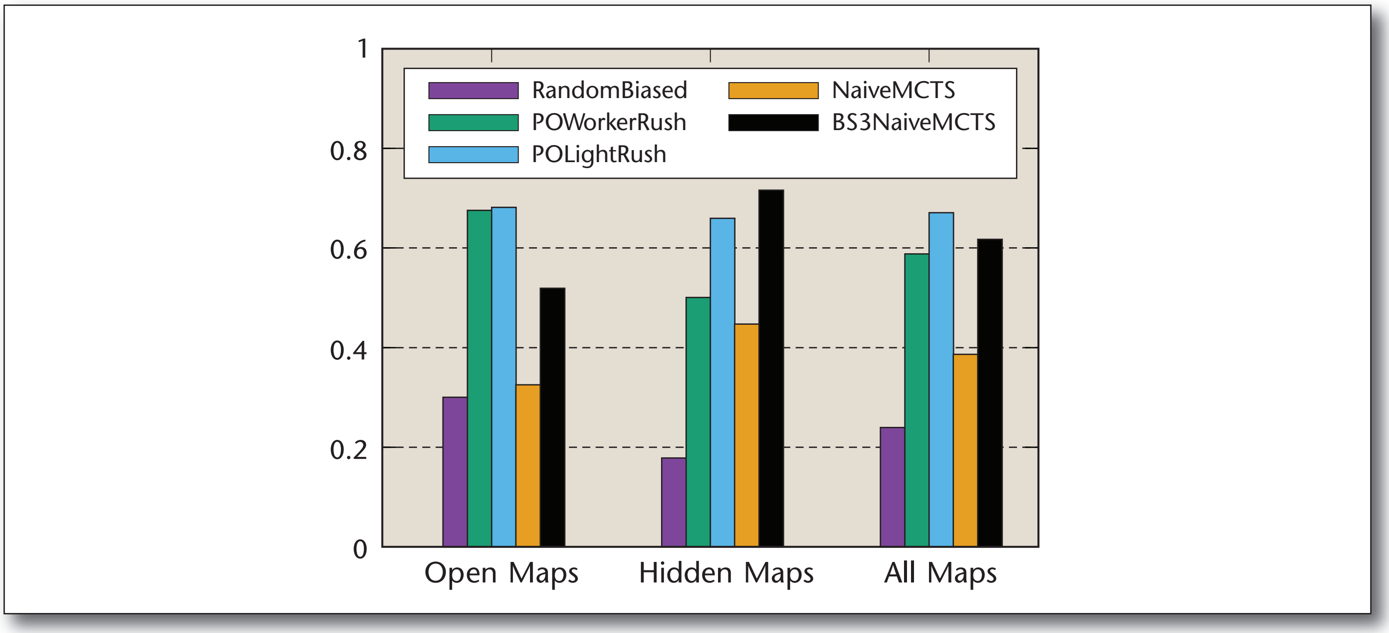

Figure 7. Win Ratios of the Bots in the Partial Observability Track, by Map Type.

The left plot shows the win ratios averaged over only the open maps; the center over the hidden maps; and the right plot the average of all maps.

shows the win ratio averaged over the open and the hidden maps. As with the standard track, there are substantial differences in the performance of the bots between the two.

The same observations apply as in the standard track: hard-coded bots perform well in standard situations, and not as well in nonstandard situations. Adding nondeterminism did not change this fact.

POWorkerRush, PuppetSearch, and SCV performed better in this track than in the standard track, whereas NaïveMCTS performed worse, being outperformed by SCV. POLightRush and StrategyTactics performed slightly worse in this track, but that result follows from the first three bots having performed better. The lower performance of NaïveMCTS, in particular, was to be expected, since its game tree search nature assumes a deterministic game. Overall, however, it seems that the low amount of nondeterminism present in this track, which is representative of commercial RTS games, did not affect the performance of most bots excessively. The exception is NaïveMCTS, which performed significantly worse in the open maps.

Partial Observability Track

Five bots were used for the partial observability track: four of the preexisting bots (RandomBiased, POWorkerRush, POLightRush, NaïveMCTS) and one competition entry, BS3NaïveMCTS. Each round robin tournament consisted of $5 \times 4 = 20$ games (since we discarded self-play matches), and we performed five full round-robin tournaments in each of the eight maps, for a total of $5 \times 8 \times 20 = 800$ games.

Figure 7 shows the win ratios achieved by each of the bots in this track, organized by type of map. The bot that achieved the highest win ratio over all maps was POLightRush, with a win ratio of 0.670. The second-best bot in this scenario was BS3NaïveMCTS, the competition entry, with a win ratio of 0.617. Figure 7 also shows the win ratio averaged over the open and the hidden maps.

In terms of the observations to be made, the first thing we see is that the random bot (RandomBiased) performed much better in this track, which might be due to the fact that, since there is partial observability, it is harder for the other bots to pinpoint where the opponent is. The random bot also tends to create a large number of workers, which can be hard to deal with later in the game.

We see that BS3NaïveMCTS performs significantly better than NaïveMCTS, which is to be expected, since there is a direct improvement over it to handle partial observability.

We can also see that BS3NaïveMCTS outperformed both hard-coded bots in the hidden maps, whereas this did not happen in the standard track. Hard-coded bots still have an advantage, though, on the open maps.

Looking at the per-map results, we see that in this partially observable setting, no bot managed to win a single game in the very large map4, so all games there ended in a tie. The main problem is that in such a large map, it was difficult for the bots to find each other, which highlights the importance of

Track	Open Maps	Hidden Maps	All Maps
Standard	POLightRush	NaïveMCTS	StrategyTactics
Nondeterminism	StrategyTactics	NaïveMCTS	StrategyTactics
Partially Observable	POLightRush	BS3NaïveMCTS	POLightRush

Table 1. Competition Winners.

scouting and exploration in partial observability settings. A similar, but not as extreme, situation can be seen in the 32 × 32 maps (map3 and map8), where there were also a large number of ties.

Conclusions

This article has presented the results of the first µRTS AI competition. The competition was organized around three tracks, each of which used four open and four hidden maps. Winners of the competition are shown in table 1. Observing the results of all three tracks, we can see that hard-coded bots (POWorkerRush and POLightRush) perform very well in standard maps, where their hard-coded rules apply, but they perform worse on nonstandard maps. These results are consistent with results from the StarCraft AI competitions (Churchill et al. 2016), where all maps are standard, and thus hard-coded bots still dominate. We have also seen that game tree search bots perform better on nonstandard maps, but struggle with larger maps. Additionally, we have seen that the concept used by StrategyTactics of integrating long-term script-based search with low-level game tree search can achieve very good performance, being the only approach that outperformed the hard-coded bots consistently. Finally, we have seen that the one machine learning–based bot (SCV) performs very well in scenarios similar to its training data, but struggles when the maps look very different. One strong point of the machine learning approaches is the low CPU requirements, which leaves a lot of free CPU time for potential integration with other techniques.

The good performance of the hard-coded bots, which would not be difficult for a human to beat, indicates that there is significant room for improvement, and plenty of work yet to be done, to achieve human-level AI bots in RTS games.

From the results, we can also see that the largest factors in determining the performance of a bot are (1) the size of the map (scalability), (2) the type of map (whether standard or not, and thus whether hard-coded strategies will work or not), and (3) partial observability. We observed that the small amount of nondeterminism in RTS games did not have as much effect as these three factors, which might provide hints as to which aspects to focus on in future work.

A final thought, for future competitions, is that although the bots presented in this edition represent a wide spectrum of AI approaches, many others that have performed well in related domains (such as reinforcement learning or evolutionary algorithms) were not represented. The results of this first edition of the competition, however, provide a good baseline for comparison, and hint at the different strengths and weaknesses of the different approaches.

Notes

1. github.com/santiontanon/microrts.
2. github.com/nbarriga/microRTSbot.
3. github.com/rubensolv/SCV.
4. Incorporated into the main branch of µRTS at github.com/santiontanon/microrts.
5. SCV was referred to as PVAIML ED in the competition website.

References

Barriga, N. A.; Stanescu, M.; and Buro, M. 2017a. Combining Strategic Learning and Tactical Search in Real-Time Strategy games. In *Proceedings of the Thirteenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (AIIDE). Palo Alto, CA: AAAI Press. doi.org/10.1109/TCIAIG.2017.2717902

Barriga, N. A.; Stanescu, M.; and Buro, M. 2017b. Game Tree Search based on Non-Deterministic Action Scripts in Real-Time Strategy Games. *IEEE Transactions on Computational Intelligence and AI in Games* (PP)99. doi.org/10.1109/TCIAIG.2017.2717902

Buro, M. 2003. Real-Time Strategy Games: A New AI Research Challenge. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence,* 1534–1535. San Francisco: Morgan Kaufmann.

Churchill, D., and Buro, M. 2013. Portfolio Greedy Search and Simulation for Large-Scale Combat in StarCraft. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games* (CIG), 2, 1–8. Piscataway, NJ: Institute for Electrical and Electronics Engineers. doi.org/10.1109/CIG.2013.6633643

Churchill, D.; Preuss, M.; Richoux, F.; Synnaeve, G.; Uriarte, A.; Ontañón, S.; and Certicky, M. 2016. StarCraft Bots and Competitions. *Springer Encyclopedia of Computer Graphics and Games*. Berlin: Springer.

Corlett, R. A., and Todd, S. J. 1986. A Monte-Carlo Approach to Uncertain Inference. In *Artificial Intelligence and Its Applications,* 127–137. New York: John Wiley & Sons.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence,* 1123–1128. Menlo Park, CA: AAAI Press.

Justesen, N.; Tillman, B.; Togelius, J.; and Risi, S. 2014. Script- and Cluster-Based UCT for StarCraft. In *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games* (CIG), 1–8. Piscataway, NJ: Institute for Electrical and Electronics Engineers. doi.org/10.1109/CIG.2014.6932900

Komenda, A.; Shleyfman, A.; and Domshlak, C. 2014. On Robustness of CMAB Algorithms: Experimental Approach. In *The Third Workshop on Computer Games,* 16–28. Berlin: Springer.

Lelis, L. H. S. 2017. Stratified Strategy Selection for Unit Control in Real-Time Strategy Games. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence,* 3735–3741. Vienna, Austria: IJCAII, Inc.

Ontañón, S. 2013. The Combinatorial Multi-Armed Bandit Problem and its Application to Real-Time Strategy Games. In *Proceedings of the Ninth AAAI Artificial Intelligence and Interactive Digital Entertainment Conference*. Palo Alto, CA: AAAI Press.

Ontañón, S. 2016. Informed Monte Carlo Tree Search for Real-Time Strategy Games. In *Proceedings of the 2016 IEEE Conference on Computational Intelligence in Games* (CIG), 1–8. Piscataway, NJ: Institute for Electrical and Electronics Engineers. doi.org/10.1109/CIG.2016.7860394

Ontañón, S. 2017. Combinatorial Multi-Armed Bandits for Real-Time Strategy Games. *Journal of Artificial Intelligence Research* 58: 665–702.

Ontañón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games* 5(4): 293-311. doi.org/10.1109/TCIAIG.2013.2286295

Shleyfman, A.; Komenda, A.; and Domshlak, C. 2014. On Combinatorial Actions and CMABs with Linear Side Information. In *Proceedings of the 21st European Conference on Artificial Intelligence,* 825–830. Amsterdam, The Netherlands: IOS Press.

Stanescu, M.; Barriga, N. A.; Hess, A.; and Buro, M. 2016. Evaluating Real-Time Strategy Game States Using Convolutional Neural Networks. In *Proceedings of the 2016 IEEE Conference on Computational Intelligence in Games* (CIG), 1–7. Piscataway, NJ: Institute for Electrical and Electronics Engineers. doi.org/10.1109/CIG.2016.7860439

Uriarte, A., and Ontañón, S. 2016. Single Believe State Generation for Partially Observable Real-Time Strategy Games. In *Proceedings of the 2017 IEEE Conference on Computational Intelligence in Games* (CIG), 296-303. Piscataway, NJ: Institute for Electrical and Electronics Engineers.

Uriarte, A., and Ontañón, S. 2017. Combat Models for RTS Games. *IEEE Transactions on Computational Intelligence and AI in Games* (PP)99. doi.org/10.1109/TCIAIG.2017.2669895

Whitehouse, D.; Powley, E. J.; and Cowling, P. I. 2011. Determinization and Information Set Monte Carlo Tree Search for the Card Game Dou Di Zhu. In *Proceedings of the 2011 IEEE Conference on Computational Intelligence in Games* (CIG), 87–94. Piscataway, NJ: Institute for Electrical and Electronics Engineers. doi.org/10.1109/CIG.2011.6031993

Santiago Ontañón is an associate professor in the Computer Science Department at Drexel University. His main research interests are game AI, case-based reasoning, and machine learning, fields in which he has published more than 150 peer-reviewed papers. He obtained his PhD from the Autonomous University of Barcelona, Spain. Before joining Drexel University, he held postdoctoral research positions at the Artificial Intelligence Research Institute in Barcelona and at the Georgia Institute of Technology in Atlanta, and he lectured at the University of Barcelona.

Nicolas A. Barriga holds a PhD in computing science from the University of Alberta as well as a BSc, in engineering and a MSc in informatics engineering from Universidad Técnica Federico Santa María. After a few years working as a software engineer for the Gemini and ALMA astronomical observatories, he turned to game AI research, in which domain he is currently working on learning, search, and abstraction mechanisms for RTS games.

Cleyton R. Silva has a bachelor's degree in computer science from the Universidade Federal de Viçosa, Brazil, and he is currently a master's student at the same institution. He is interested in AI and intelligent agents.

Rubens O. Moraes has a bachelor's degree in computer science from Universidade Cândido Mendes, Brazil, and a specialization in project management and computer information systems from Instituto Federal Fluminense. Moraes is currently a master's student at Universidade Federal de Viçosa, and he is interested in AI, machine learning, and real-time strategy games.

Levi H. S. Lelis is an assistant professor in the Departamento de Informática at Universidade Federal de Viçosa (UFV), Brazil. He is interested in AI and has focused his research on the subfields of heuristic search and planning. Lelis joined UFV after obtaining his PhD from the University of Alberta, Canada.