StarCraft AI Competitions, Bots, and Tournament Manager Software

Michal Čertický, David Churchill, Kyung-Joong Kim, Martin Čertický, and Richard Kelly

***Abstract*—Real-time strategy games have become an increasingly popular test bed for modern artificial intelligence (AI) techniques. With this rise in popularity has come the creation of several annual competitions, in which AI agents (bots) play the full game of *StarCraft: Broodwar* by Blizzard Entertainment. The three major annual StarCraft AI Competitions are the Student StarCraft AI Tournament, the Computational Intelligence in Games competition, and the Artificial Intelligence and Interactive Digital Entertainment competition. In this paper, we will give an overview of the current state of these competitions, describe the bots that compete in them, and describe the underlying open-source Tournament Manager software that runs them.**

***Index Terms*—Artificial intelligence, computational intelligence, education, computer science education, educational programs, learning, machine learning.**

I. Introduction

REAL-TIME strategy (RTS) games are a genre of video games in which players manage economic and strategic tasks by gathering resources and building bases, increase their military power by researching new technologies and training units, and lead them into battle against their opponent(s). They serve as an interesting domain for artificial intelligence (AI) research and education, since they represent well-defined, complex adversarial systems [1] that pose a number of interesting AI challenges in the areas of planning, dealing with uncertainty, domain knowledge exploitation, task decomposition, spatial reasoning, and machine learning [2].

Manuscript received January 8, 2018; revised April 11, 2018, July 11, 2018, August 27, 2018, and November 9, 2018; accepted November 13, 2018. Date of publication November 26, 2018; date of current version September 13, 2019. This work was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT and Future Planning (2017R1A2B4002164). *(Corresponding author: Michal Čertický.)*

M. Čertický is with the Artificial Intelligence Center, Czech Technical University in Prague, Prague 16000, Czech Republic (e-mail: certicky@agents.fel.cvut.cz).

D. Churchill and R. Kelly are with the Department of Computer Science, Memorial University of Newfoundland, St. John's, NF A1C 5S7, Canada (e-mail: dave.churchill@gmail.com; richard.kelly@mun.ca).

K.-J. Kim was with the Department of Computer Science and Engineering, Sejong University, Seoul 143-747, South Korea. He is now with the School of Integrated Technology, Gwangju Institute of Science and Technology (GIST), Gwangju 61005, South Korea (e-mail: kjkim@gist.ac.kr).

M. Čertický is with the Department of Cybernetics and Artificial Intelligence, Technical University in Košice, Košice 04001, Slovakia (e-mail: martin.certicky@tuke.sk).

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

Digital Object Identifier 10.1109/TG.2018.2883499

Unlike turn-based abstract board games, such as chess and go, which can already be played by AI at super-human skill levels, RTS games are played in *real time*, meaning the state of the game will continue to progress even if the player takes no action, and so actions must be decided in fractions of a second. In addition, individual turns in RTS games (game frames) can consist of issuing simultaneous actions to hundreds of units at any given time [3]. This, together with their partially observable and nondeterministic nature, makes the RTS game genre one of the hardest game AI challenges today, attracting the attention of the academic research community, as well as commercial companies. For example, Facebook AI Research, Microsoft, and Google DeepMind have all recently expressed interest in using the most popular RTS game of all time: *StarCraft* as a test environment for their AI research [4].

Meanwhile, the academic community has been using StarCraft as a domain for AI research since the advent of the Brood War Application Programming Interface (BWAPI) in 2009 [5]. BWAPI allows programs to interact with the game engine directly to play autonomously against human players or against other programs (bots). The introduction of BWAPI gave rise to many scientific publications over the last eight years, addressing many subproblems inherent to RTS games. A comprehensive overview can be found in [2], [6], and [7].

In addition to AI research, *StarCraft* and BWAPI are often used for educational purposes as part of AI-related courses at universities, including the University of California at Berkeley (USA), Washington State University (USA), University of Alberta (Canada), Comenius University (Slovakia), Czech Technical University (Czech Republic), and most recently Technical University Delft (The Netherlands), where a new course titled "Multi-agent systems in StarCraft" has been opened for more than 200 students. The educational potential of *StarCraft* has recently been extended even further, when Blizzard Entertainment released the game entirely for free in April 2017.

Widespread use of *StarCraft* in research and education has lead to a creation of three annual StarCraft AI competitions. The first competition was organized at the University of California, Santa Cruz in 2010 as part of the AAAI Artificial Intelligence and Interactive Digital Entertainment (AIIDE) conference program. The following year gave rise to two other annual competitions—the Student StarCraft AI Tournament (SSCAIT), organized as a standalone long-term event at Comenius University and Czech Technical University, and the CIG StarCraft AI competition colocated with the IEEE Computational Intelligence in Games (CIG) conference.

In this paper, we will talk about these three major StarCraft AI competitions and provide the latest updates on each of them, with the following three sections detailing the SSCAIT, AIIDE, and CIG StarCraft AI Competitions. We will then describe the state-of-the-art bots under active development that compete in these competitions, and the AI methods they use. Finally, we will introduce the open-source Tournament Manager software powering the competitions.

II. Student StarCraft AI Tournament

The SSCAIT is the StarCraft AI competition with the highest number of total participants. There are three fundamental differences between SSCAIT and the remaining two competitions.

1) SSCAIT is an online-only event. Unlike AIIDE or CIG, it is not colocated with a scientific conference/event.
2) There are two phases of SSCAIT each year: a competitive *tournament phase*, lasting for up to four weeks and a *ladder phase* that runs for the rest each year. In other words, SSCAIT is live at all times with only a few short interruptions for maintenance.
3) Games are played one at a time and are publicly streamed live on Twitch.tv[1] and SmashCast.tv. The AIIDE and CIG competitions instead play as many games as possible at maximum speed, with no broadcast.

SSCAIT's *tournament phase* takes place every winter in late December and early January.

A. SSCAIT History

The first SSCAIT was organized in 2011 by Michal Čertický, as a part of the "Fundamentals of Artificial Intelligence" course at Comenius University, Bratislava, Slovakia. It started as a closed event, with 50 local students competing for extra points for their course evaluation. Since the event received a lot of positive feedback from the participants, the organizers decided to open it for the international public and for nonstudents next year (although the word "Student" remained in the competition name for historic reasons).

SSCAIT changed significantly over the course of 2012—both in terms of the format and technology behind it. The organizers implemented a collection of simple Python and AHK scripts that were able to run and evaluate the bot games automatically. This allowed for the creation of 24/7 bot *ladder* with online live stream, similar to the one available today.[2] The live-streamed ladder simplifies the bot debugging process (since bot authors can watch their creations play all kinds of AI opponents), encourages continuous development over the whole year and accelerates the growth of StarCraft AI research community.

The registration of new bots on the ladder was simplified in 2013 with the introduction of web-based user interface for bot creators. They can now upload new versions of their bots to the ladder at any time. In 2014, the custom automation scripts were replaced by Tournament Manager software, developed originally for AIIDE competition (details in Section VI), which needed to be heavily modified in order to work with SSCAIT user and game databases and to support the ladder format. Further modifications and the introduction of the Dockerized multiplatform version of *StarCraft* [8] are planned soon.

Fig. 1. SSCAIT live stream running in HD resolution and controlled by the custom observer script [9].

Currently, SSCAIT is organized by the *Games and Simulations* Research Group[3]—part of Artificial Intelligence Center, Czech Technical University, Prague.

B. SSCAIT 2017/2018 Tournament & Ladder

The activity of bot programmers and the general public surrounding SSCAIT has grown considerably over the course of the past year, with new members to the organizing team making a number of improvements to the live stream, and better community engagement during the Ladder phase.

First, the ladder phase was updated, with SSCAIT introducing so-called "weekly reports." Every weekend, there is a 1–2-h long segment of curated AI versus AI matches with insightful commentary on the live stream. Second, a voting system was implemented, allowing bot programmers and viewers to select which bots will play the next ladder match on live stream. This not only supports viewer engagement, but also greatly simplifies bot debugging process. Bot programmers can now quickly test their newest updates against specific opponents. This change might have contributed to the significant increase in bot update frequency. Approximately 5–6 bots are updated every day, in contrast to 0–2 updates per week in 2015.

Another update was the introduction of "minitournaments" to SSCAIT. These are easily configurable, irregular, and unofficial short competitions, taking up to one day. The format of these minitournaments and the selection of participants is usually up to the stream viewers and moderators. Visual quality of the stream was improved by updating the custom observer script [9], which now moves the camera fluently to the most interesting parts of the game in real time and displays SSCAIT-related information on top of the game. The stream was also upgraded to HD using a "resolution hack" (see Fig. 1). The overall number of stream views has increased to 376 920 views on Twitch.tv and additional 434 216 views on SmashCast.tv over the past 12 months. Two additional metrics were added to the ladder ranking system due to popular demand: ELO rating [10], which is used in adversarial games, such as chess, and "SSCAIT rank," based on

[1] http://www.twitch.tv/sscait

[2] The SSCAIT bot ladder was inspired by an older automated StarCraft bot ladder, available at that time at http://bots-stats.krasi0.com/

[3] http://gas.fel.cvut.cz/

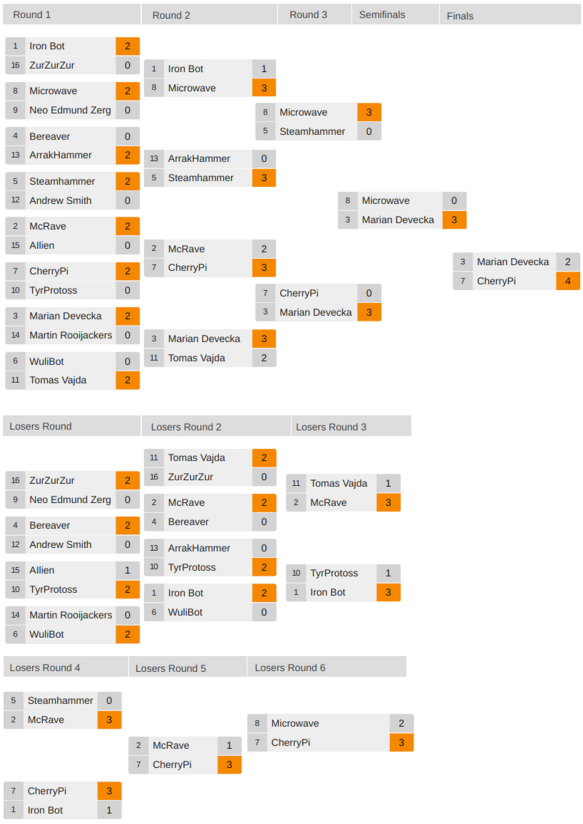

Fig. 2. SSCAIT 2017/18 mixed division double-elimination bracket.

the so-called "ICCUP ranking system," typical for competitive *StarCraft*.

1) SSCAIT Tournament Phase Updates: The 2017/18 installment of SSCAIT's tournament phase took place during four weeks at the end of December 2017 and beginning of January 2018 and sported 78 participants. The tournament was divided into the following two divisions.

a) Student Division: Round Robin tournament of 6006 games, where every bot played two games against every opponent. Only the bots created by individual students were considered "student" bots and were eligible for victory in this division. Other bots were tagged as "mixed-division" bots (they played the games, but could not win the student division title). Winners of the student division in 2017/2018 were as follows.

1) Wulibot, University of Southern California (USA) with 124 wins.
2) LetaBot (Martin Rooijackers), University of Maastricht (The Netherlands) with 109 wins.
3) Carsten Nielsen, Technical University of Denmark (Denmark) with 101 wins.

The student division of SSCAIT exists so that the students stand a chance of winning in the presence of more experienced, nonstudent participants, and team-created bots.

b) Mixed Division: After the student division ended, 16 bots with the most wins among all the participants were selected for the additional mixed division double elimination bracket, consisting of 30 matches (best of three, five, or seven games), which is shown in Fig. 2.

CherryPi, created by the Facebook AI Research team won the mixed division by beating KillerBot by Marian Devecka 4-2 in the finals. Interestingly, CherryPi encountered KillerBot earlier in the tournament (in winner's round 3) and lost that match 0-3, dropping down to the losers bracket. The bot then managed to win the whole losers bracket, meet Killerbot again in the finals, and win by exploiting its experience from their previous games. More information about CherryPi can be found in Section V.

All the elimination bracket games were published as videos with commentary on SSCAIT YouTube channel[4] and as replay files on SSCAIT website.[5]

III. Artificial Intelligence and Interactive Digital Entertainment

The AIIDE StarCraft AI Competition is the longest running annual StarCraft competition, and has been held every year since 2010 along with the AAAI Artificial Intelligence and Interactive Digital Entertainment conference. Unlike the CIG and SSCAIT competitions, the AIIDE competition requires (since 2011) that bot source code be submitted, and that the code will be published for download after the competition has finished. Running 24 hours a day for two weeks with games played at super-human speed, the competition is a single round-robin format with the winner being the bot with the highest win percentage when the time limit has been reached.

A. AIIDE StarCraft AI Competition History

The AIIDE StarCraft AI Competition was first run in 2010 by Ben Weber at the Expressive Intelligence Studio at the University of California, Santa Cruz, as a part of the AIIDE conference. In total, 26 entrants competed in four different game modes that varied from simple combat battles to the full game of *StarCraft*. As this was the first year of the competition, and little infrastructure had been created, each game of the tournament was run manually on two laptop computers and monitored by hand to record the results. Also, no persistent data were kept for bots to learn about opponents between matches. The 2010 competition had four different tournament categories in which to compete. Tournament 1 was a flat-terrain unit micromanagement battle consisting of four separate unit composition games. Tournament 2 was another microfocused game with nontrivial terrain. Tournament 3 was a tech-limited *StarCraft* game on a single known map with no fog-of-war enforced. Players were only allowed to choose the Protoss race, with no late game units allowed.

Tournament 4 was considered the main event, which involved playing the complete game of *StarCraft: Brood War* with fog-of-war enforced. The tournament was run with a random pairing double-elimination format with each match being best of five games. A map pool of five well-known professional maps were announced to competitors in advance, with a random map being chosen for each game. Tournament 4 was won by Overmind—a Zerg bot created by a large team from the University of

[4]https://goo.gl/icbYdK
[5]http://sscaitournament.com/index.php?action=2017

TABLE I
TOURNAMENT SETTINGS AND TOP 3 FINISHERS IN THE AIIDE, CIG, AND SSCAIT (STUDENT DIVISION) COMPETITIONS

Competition	Year	TM Software	Open-Source	Maps	1st Place	2nd Place	3rd Place
AIIDE	2010	Manual	Optional	Unannounced	Overmind	Krasi0	Chronos
AIIDE	2011	AIIDE TM	Forced	Announced	Skynet	UAlbertaBot	Aiur
AIIDE	2012	AIIDE TM	Forced	Announced	Skynet	Aiur	UAlbertaBot
AIIDE	2013	AIIDE TM	Forced	Announced	UAlbertaBot	Skynet	Aiur
AIIDE	2014	AIIDE TM	Forced	Announced	IceBot	Ximp	LetaBot
AIIDE	2015	AIIDE TM	Forced	Announced	tscmoo	ZZZKBot	Overkill
AIIDE	2016	AIIDE TM	Forced	Announced	Iron	ZZZKBot	tscmoo
AIIDE	2017	AIIDE TM	Forced	Announced	ZZZKBot	PurpleWave	Iron
CIG	2011	Manual	Optional	Unannounced	Skynet	UAlbertaBot	Xelnaga
CIG	2012	AIIDE TM	Optional	Unannounced	Skynet	UAlbertaBot	Xelnaga
CIG	2013	Java-based TM	Optional	Unannounced	Skynet	UAlbertaBot	Aiur
CIG	2014	AIIDE TM	Forced	Unannounced	IceBot	Ximp	LetaBot
CIG	2015	AIIDE TM	Optional	Unannounced	ZZZKBot	tscmoo	Overkill
CIG	2016	AIIDE TM	Forced	Announced	tscmoo	Iron	LetaBot
CIG	2017	AIIDE TM	Forced	Announced	ZZZKBot	tscmoo	PurpleWave
SSCAIT	2011	Custom	Optional	Announced	Roman Danielis	N/A	N/A
SSCAIT	2012/13	Custom	Optional	Announced	DementorBot	Marcin Bartnicki	UAlbertaBot
SSCAIT	2013/14	Custom	Optional	Announced	Ximp	WOPR	UAlbertaBot
SSCAIT	2014/15	AIIDE TM + Mod	Optional	Announced	LetaBot	WOPR	UAlbertaBot
SSCAIT	2015/16	AIIDE TM + Mod	Optional	Announced	LetaBot	Carsten Nielsen	UAlbertaBot
SSCAIT	2016/17	AIIDE TM + Mod	Optional	Announced	LetaBot	Wulibot	Zia Bot
SSCAIT	2017/18	AIIDE TM + Mod	Optional	Announced	Wulibot	LetaBot	Carsten Nielsen

California, Berkeley, who defeated the Terran bot Krasi0 by Krasimir Krastev in the finals.

From 2011 to 2016, the AIIDE competition was hosted by the University of Alberta, and was organized and run each year by David Churchill and Michael Buro. Due to the low number of entries to Tournaments 1, 2, and 3 from the 2010 AIIDE competition, it was decided that the AIIDE competition for 2011 would only consist of the full game of *StarCraft* (with the same rules as the 2010 Tournament 4), with no smaller micromanagement tournaments. The 2011 tournament rules were also updated so that all entrants must submit the source code of their bot and allow it to be published after the competition is over, which was done for several reasons. The first reason was to lower the barrier to entry for future competitions—since programming a StarCraft AI bot was very time consuming, future entrants could download and modify the source code of previous bots to save considerable effort. Another reason was to more easily prevent cheating—with thousands of games being played in the tournament, no longer could each game be manually inspected to detect if any cheating tactics were being employed, which would be more easily detected by inspecting the source code. The final reason was to help advance the state of the art in StarCraft AI by allowing future bots to borrow strategies and techniques of previous bots by inspecting their source code—ideally, all bots in future competitions should be at least as strong as the bots from the previous year.

Since the first competition was run by a single person on two laptops, games were played by manually starting the *StarCraft* game and creating and joining games by hand. As the physical demand was quite high, a simple random-pairing double-elimination tournament was played with approximately 60 games in total. This caused some negative feedback that this elimination-style tournament was quite dependent on pairing luck, so for the 2011 competition all chance was eliminated from the tournament by playing a round-robin style format. Playing a round-robin format requires far more games to be played, and it would no longer be possible to run each game manually. In the summer of 2011, the StarCraft AI Tournament Manager software was written (see Section VI) that could automatically schedule and play round-robin tournaments of StarCraft on an arbitrary number of locally networked computers. The initial version of this software allowed for a total of 2340 games to be played in the same time period as the 2010 competition's 60 games, with each bot playing each other bot a total of 30 times. There were ten total maps in the competition, chosen from expert human tournaments that were known to be balanced for each race, which were available for download several months in advance on the competition website. The AIIDE competition was modeled on human tournaments where the map pool and opponents are known in advance in order to allow for some expert knowledge and opponent modeling.

The 2012 AIIDE competition brought a major change to the functionality of the StarCraft AI Competitions: persistent file storage, which allowed the bots to learn throughout the course of the competition. The tournament managing software was updated so that each bot had access to a read folder and a write folder contained on a shared folder that was accessible to all the client machines. During each round bots could read from their "read" folder and write to their "write" folder, and at the end of each round robin (one game between each bot pairing on a single map) the contents of the write folder were copied to the read folder, giving access to all information written about previous rounds. This new functionality was used by several bots to implement strategy selection, in which their bot selected which of several strategies to use based on the results of previous rounds versus the same opponent, which typically increased their win rates over time during the competitions.

The AIIDE competitions between 2013 and 2016 did not have any major rule changes, and continued to use the same pool of ten maps for each competition. Competition appeared to stagnate between 2011 and 2013, with a relatively low number of entrants, and saw the same three bots (Aiur, Skynet, and UAlbertaBot) trading first, second, and third place during these years. The 2014 to 2016 competitions, however, saw many new entries to the competition, with new bots taking the top three positions each year. Top three finishers of each year's

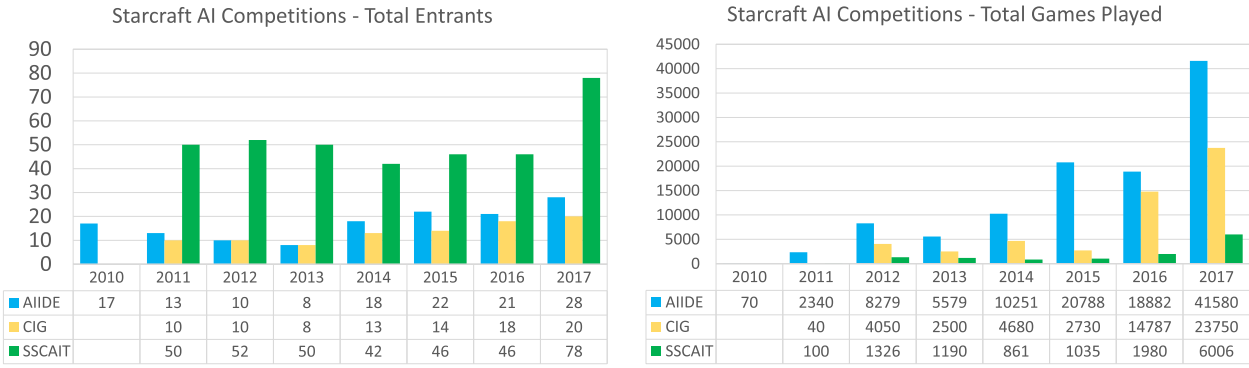

Fig. 3. Statistics for each of the three major annual StarCraft AI Competitions: AIIDE, CIG, and SSCAIT, since the first competition in 2010. Shown on the left is the number of total entrants for each competition, and on the right are the total number of games played in each competition.

TABLE II
RESULTS OF THE TOP EIGHT FINISHERS IN THE 2017 AIIDE COMPETITION

Bot	Race	Games	Win	Loss	Win %
ZZZKBot	Zerg	2966	2465	501	83.11
PurpleWave	Protoss	2963	2440	523	82.53
Iron	Terran	2965	2417	548	81.52
cpac	Zerg	2963	2104	859	71.01
Microwave	Zerg	2962	2099	863	70.86
CherryPi	Zerg	2966	2049	917	69.08
McRave	Protoss	2964	1988	976	67.07
Arrakhammer	Zerg	2963	1954	1009	65.95

competition are shown in Table I. Improvements to the tournament software and hardware infrastructure allowed for more games to be played each year are shown in Fig. 3.

B. 2017 AIIDE Competition

The 2017 AIIDE competition[6] had a total of 28 competitors, and the round-robin games ran on 14 virtual machines for two weeks. In total, 110 rounds of round-robin play were completed, with each bot playing 2970 games for a total of 41 580 games. Any bot that achieved a win rate of 30% or higher in the 2016 competition that did not receive a new submission was automatically entered into the 2017 competition. No new rules or maps were used for the 2017 tournament that were not in place for the 2016 tournament. The AIIDE Tournament Manager software had been updated with new features, such as support for BWAPI version 4.2.0, and the ability for client machines to be listed with special properties, such as GPU computation ability. In combination with this update, a hardware upgrade for the tournament allowed for GPU computation support for any bots that required it, however, no 2017 bots used the feature. The 2017 competition had the closest top three finish of any competition yet, with the top three bots separated by less than 2% win rate, and 3rd–6th place bots also separated by less than 2% win rate. Statistics for the top eight finishers are shown in Table II.

The win percentage over time of the top three bots of the competition is shown in Fig. 4, and demonstrates the importance of implementing some form of opening modeling/learning over time. Although Iron (shown in green) led for the vast majority of the competition, it did not implement any form of learning over the course of the competition, and its win rate slowly dropped over time. ZZZKBot (blue) and PurpleWave (black) implemented strategy selection learning, and their win rates slowly climbed to the point where they overtook Iron near round 85 of 110.

[6]http://www.cs.mun.ca/~dchurchill/starcraftaicomp/2017/

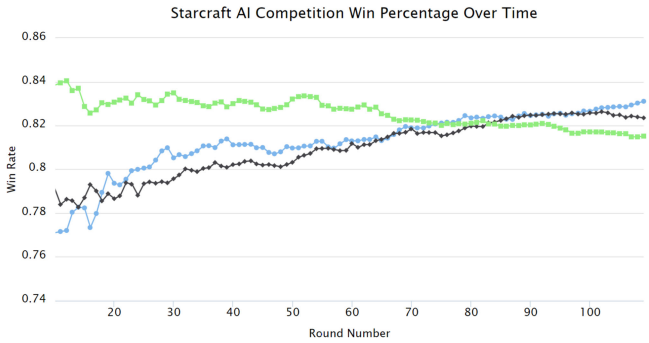

Fig. 4. Win percentage over time for the top three bots of the 2017 AIIDE StarCraft AI Competition. First place ZZZKBot shown in blue, second place PurpleWave in black, and third place Iron in green.

IV. COMPUTATIONAL INTELLIGENCE IN GAMES

The CIG StarCraft AI Competition has been a part of the program of the IEEE Computational Intelligence in Games conference since August 2011. Since the date of the CIG competition was usually just before the AIIDE competition, many of the bots submitted to both competitions ended up being nearly identical, therefore, the CIG competition has several rule differences with AIIDE in order to keep the results interesting. The biggest rule difference was that the CIG competition did not disclose which maps would be used for the competition, meaning that the bots could not use any hard-coded map information like they could for AIIDE.

A. CIG Competition History

The CIG conference is well known for hosting and organizing many AI-related competitions, such as the Mario AI Competition and the General Video Game Playing Competition, and in 2010, the first CIG StarCraft AI Competition was held. Organized by Johan Hagelback, Mike Preuss, and Ben Weber, the CIG 2010 competition was to have a single game mode similar to the tech-limited Tournament 3 from the AIIDE 2010

competition, but using the Terran race instead of the Protoss race. Unfortunately, the first year of the CIG competition had several technical problems, and no winner could be announced for the competition. Mike Preuss and his team members then successfully organized the CIG competition each year from 2011 to 2013. Since 2014, the Sejong University team (led by Kyung-Joong Kim) has been organizing the CIG competition at the IEEE CIG conference. In order to provide a more diverse competition, CIG rules and settings have changed each year, as shown in Table I.

Throughout its history, CIG has had multiple changes in the selection of tournament management software, open-source policy, and map pool announcement policy. The tournament management software (see Section VI) is used to distribute the matches over multiple machines on the network and to automate the competition operation. Although CIG organizers developed their own JAVA-based TM software, the AIIDE TM has been used for the competition since 2014 (see details in Section VI). Since 2016, CIG has enforced an open-source code policy, and all of the bots' source code are published after the competition. Unlike the AIIDE competition, the CIG map pool was not known to the participants before the competition to promote generalization ability of the entries. However, it was found that participants usually did not exploit map knowledge, and so since 2016, maps in the CIG competition have been announced in advance.

In the 2016 competition, the organizers introduced a second stage to the competition such that half of the entries advance to the second stage based on the win ratio of the first stage. This was inspired by the Simulated Car Racing Competition [11] that adopted a two-stage competition divided into a qualification stage and a main race. Since the single-pool round-robin format is based on win percentage only, it is important to get high average win ratio against all the opponents. The CIG organizers introduced two pools with the intent to reduce the chance that the top ranking bots win just by exploiting lower ranked bots to boost their win ratio. Bots were randomly split into two groups, and then the top half of bots from each group were brought together for a final stage group, to be played as round robin, with all learned information being deleted before the beginning of the final stage.

B. 2017 CIG Competition

In the 2017 CIG competition, the two-stage format was changed back to the single group format. The reason was that the participants did not seem to change their strategy to consider the two-stage tournament, and it did not seem to have much of an effect on the final results. In the future, CIG organizers consider adopting the SWISS-system widely used in the board game community. In this setting, a player does not play against all the other opponents. Instead, the participants are paired with the opponents with similar scores. Such systems usually produce final outcomes similar to round-robin while playing fewer total games.

In 2017, CIG organizers tried to play as many games as possible, and reached 125 rounds with 190 games per round, which resulted in 23 750 games in the two-round format. Currently, AI bots often use multiple preprepared strategies and adapt them or their selection against specific opponents. The more games played during a tournament, the more experience allows them to learn which strategies are good against which opponents. As in the 2017 AIIDE competition, many bots implemented learning strategies that dramatically increased their win rates over time. Detailed results of the top 6 bots in the competition can be seen in Table III.

TABLE III
RESULTS OF THE 2017 CIG COMPETITION FINAL STAGE

Bot	Race	Games	Win	Loss	Win %
ZZZKBot	Zerg	1984	1628	356	82.06
tscmoo	Random	1992	1541	451	77.36
PurpleWave	Protoss	2021	1360	661	67.29
LetaBot	Terran	2026	1363	663	67.28
UAlbertaBot	Random	2005	1315	690	65.59
Overkill	Zerg	2024	1270	754	62.75

After the 2017 CIG competition, Sejong University organized a special event where human players were matched against the AI bots. The human players included one novice player (ladder rating around 1100), one middle-level player (around 1500), and a professional gamer: Byung-Gu Song. AI bots in the event were ZZZKBot (winner of CIG 2017), tscmoo bot (2nd place in CIG 2017), and MJBOT, an AI bot specially designed against human players. MJBOT has been developed since June 2017 by Cognition Intelligence Laboratory to beat novice/middle-level human players. Each human player played a single game against each AI bot (nine games). The novice human player lost two games against ZZZKBOT and TSCMOO, but won the game against MJBOT, which was not able to finish the game due to a programming bug. In the next session, the middle-level human player lost all three games against the AI bots. Finally, the professional human player Byung-Gu Song won against all the AI bots.[7] This suggests that the AI bots have a potential to compete against novice and middle-level players, but not professionals.

V. CURRENT STARCRAFT BOTS

Over the years, StarCraft AI competitions have motivated many individuals and groups to implement a variety of bots capable of playing complete *StarCraft* 1v1 games. The structure of most current bots emerges from the attempts to decompose the game into a hierarchy of smaller subproblems, such as higher level strategy, tactics, combat unit control, terrain analysis, and intelligence gathering. For more information about the individual challenges, refer to [2] and [6].

Bots vary in complexity, and while many are rule based, top-performing bots are now employing more sophisticated AI techniques, such as real-time search/planning, pretrained neural network controllers, and online learning during the competition games. In this section, we provide an overview of a selection of bots and discuss some of the AI approaches they implement. We only mention those bots that were active in one of the 2017

[7] http://cilab.sejong.ac.kr/

TABLE IV
OVERVIEW OF THE TECHNIQUES USED IN 2017 BOTS DESCRIBED IN SECTION V

Bot	Created	Rules	ML	HS	IO	SIM
CherryPi	2017	Yes	No	No	Yes	Yes
cpac	2017	Yes	No	Yes	No	Yes
ForceBot	2017	Yes	No	No	No	No
Iron	2016	Yes	No	No	No	No
KillAll	2017	Yes	Yes	Yes	No	Yes
Krasi0bot	2010	Yes	Yes	Yes	Yes	Yes
LetaBot	2014	Yes	No	Yes	Yes	Yes
McRave	2017	Yes	No	No	No	Yes
MegaBot	2016	Yes	No	No	Yes	Yes
PurpleWave	2017	Yes	No	Yes	Yes	Yes
StarcraftGP	2015	Yes	Yes	No	No	No
Steamhammer	2016	Yes	No	Yes	Yes	Yes
tscmoo	2015	Yes	Yes	Yes	Yes	Yes
UAlbertaBot	2010	Yes	No	Yes	Yes	Yes
ZZZKbot	2015	Yes	Yes	No	Yes	No

Shown are bots' creation year, and whether the bot uses any: "Rules" = rule-based systems, "ML" = machine learning, "HS" = heuristic search, "IO" = file I/O for learning during competitions, and "SIM" = simulations.

competitions, have recently been updated, and employ some more complex AI techniques (see Table IV).

1) *CherryPi:* CherryPi is a TorchCraft [12] Zerg bot developed by Facebook AI Research team. It is implemented as a collection of heterogeneous modules that can be added, removed, or configured via the command line. This design allows individual modules to be easily replaced with learning-powered modules, or to do narrow experiments using only a subset of them. The modules communicate by exchanging two kinds of data elements via the blackboard architecture: Key-value pairs and so-called UPC objects (Unit, Position, Command), which have a generic enough meaning to be loosely interpreted by other modules. In general, a UPC represents a probability distribution of units, a probability distribution of positions, and a single command. Build orders are represented as a set of prioritized requests pushed into a queue that fills in requirements and applies optimizations (like allocating resources for just-in-time construction). Fight-or-flight decisions are made by clustering units and running a combat simulation. CherryPi's combat simulation works similarly to the SparCraft simulation package (see UAlbertaBot) with a naive "attack-the-closest-target" policy for enemy behavior. A threat-aware path-finding is used to find the least dangerous routes to a safety. The selection of high-level strategy across multiple games is based on UCB1 algorithm, selecting from a fixed list of strategies against each race. The bot uses TorchCraft to communicate with BWAPI over TCP, which allows it to run on a different host than *StarCraft*.
2) *cpac:* A Zerg bot created by a 13-person team from China, cpac combines hard-coded rules with a multilayer perceptron network for unit production. The network is trained on state-action pairs extracted from a large data set of BroodWar games.[8] The core of cpac bot is based on the bots UAlbertaBot and Steamhammer (see ahead).
3) *ForceBot:* ForceBot is a Zerg bot written in GOAL—an agent-based programming language designed on top of BWAPI for programming cognitive agents.[9] Since the GOAL language is designed to implement multiagent systems, all of ForceBot's units have their own corresponding agent with specific beliefs and goals. Each agent more or less follows a rule-based AI pattern.
4) *Iron*[10]: Iron bot won the 2016 AIIDE competition, and is a decentralized multiagent system, with each unit controlled by a highly autonomous individual agent, able to switch between 25 behaviors. All its units share one simple aim: go to the main enemy base and destroy it. It often seems like a harasser bot, which is due to its units having mainly individual behavior. There are also so-called "expert" agents who autonomously recommend how resources should be spent and what units should be trained, based on heuristics.
5) *KillAll:* KillAll is a Zerg bot based on the Overkill bot by Sijia Xu, and most of its functionality is rule based. However, its production module uses Q-learning to select unit types to produce based on the current situation.
6) *Krasi0bot:* Krasi0bot has competed every year since 2010, and is still being actively developed. According to the author, it originally started as a rule-based bot, and currently makes some use of genetic algorithms, neural networks, and potential fields. As the bot is not open source, these details cannot be verified. Krasi0bot plays the Terran race, and is known for its strong defensive capabilities, and wide variety of strategies implemented.
7) *LetaBot*[11]: LetaBot won the 2014, 2015, and 2016 SSCAIT tournaments. It uses the Monte Carlo Tree Search to plan the movement of groups of units around the map. A similar approach has previously been used by the author of Nova bot, Alberto Uriarte [13]. It employs cooperative pathfinding for resource gathering and text mining to extract build orders from Liquipedia articles.
8) *McRave:* All the decisions of McRave bot are based on current enemy unit composition—there are no hard-coded tech choices. The bot also builds an opponent model and uses it to select build orders.
9) *MegaBot*[12]: For every game, MegaBot [14] chooses one of three approaches, each of which is implemented as a different bot (Skynet, Xelnaga, or NUSBot). Algorithm selection is modeled as a multiarmed bandit. At the beginning of the game, an algorithm is selected using epsilon-greedy strategy. After the game, the reward is perceived (+1, 0, and − 1 for victory, draw, and loss, respectively) and the value of the selected algorithm is updated via an incremental version of recency-weighted exponential average (Q-learning update rule).

[8]http://www.starcraftai.com/wiki/StarCraft_Brood_War_Data_Mining

[9]http://goalapl.atlassian.net/wiki/spaces/GOAL/

[10]http://bwem.sourceforge.net/Iron.html

[11]https://github.com/MartinRooijackers/LetaBot

[12]https://github.com/andertavares/MegaBot

10) *PurpleWave*[13]: The decision making of the PurpleWave bot is mainly based on hierarchical task networks. For micromanagement, it uses a hybrid squad/multiagent approach and nearest neighbors clustering. The bot then simulates the outcomes of battles and suggests tactics for squads by min–maxing tactical approaches by each side (e.g., "charge in," "run away," or "fight with workers"). In the end, each unit takes the tactical suggestion under advisement, but behaves independently. The units choose between approximately two dozen simple, reusable stateless behaviors. The bot heuristics include using potential fields for unit movement. Strategies are chosen based on results of previous games, race, map, and number of starting positions. It has a graph of strategy selections, like opening build orders paired with midgame transitions and late-game compositions.
11) *StarCraftGP:* StarcraftGP is the first StarCraft metabot—a program that autonomously creates a program that autonomously plays *StarCraft* [15]. Currently, StarcraftGP v0.1 is using (Linear) Genetic Programming and it is able to directly write C++ code. Its first creations: Salsa and Tequila, have been the first bots not directly written by a human to participate in international competitions.
12) *Steamhammer*[14]: The Zerg bot Steamhammer, developed by Jay Scott, and its random-race version Randomhammer are based on UAlbertaBot (see ahead), employing sophisticated combat simulation to predict the outcome of battles. The bots also use hierarchical reactive control for the units. For Protoss and Terran production, Randomhammer uses branch-and-bound search, whereas Zerg is currently rule based.
13) *tscmoo*[15]: tscmoo won the 2015 AIIDE and 2016 CIG competitions. The bot uses no external libraries: it has its own combat simulation code to predict the outcome of battles, it does not use BWTA[16] to analyze the terrain and it even has its own threat-aware path-finding for individual units. The bot is one of the most strategically diverse, and selects among its many strategies based on their success in previous games. Recent versions of the bot experimented with recurrent neural networks for high-level strategy and build-order decisions.
14) *UAlbertaBot*[17]: UAlbertaBot has competed in every major StarCraft AI Competition since 2010, and won the 2013 AIIDE competition. UAlbertaBot uses a dynamic heuristic-search-based Build-Order Search System to plan all its build orders in real time, as well as a *StarCraft* combat simulation system called SparCraft for estimating the outcome of in-game battles. The bot uses the results of previous games against specific opponents to choose a strategy to implement at the beginning of each game, with each strategy being defined in an external JSON configuration file. Its development has focused on its ease of use and modification, and as such has become the basis of more than ten other bots in current competitions, including LetaBot, Overkill, and Steamhammer. In 2017, UAlbertaBot became CommandCenter,[18] the first bot capable of playing both *BroodWar* and *StarCraft 2*.
15) *ZZZKbot*[19]: ZZZKBot, a Zerg bot developed by Chris Coxe, was the winner of the 2017 AIIDE and CIG competitions. Its overall strategy implements four simple one-base rush strategies: four-pool, Speedlings, Hydralisks, and Mutalisks. If the initial rush does not end the game, the bot switches to either Mutalisks or Guardians for the late game, while researching upgrades for all its units. The bot records win/loss information for each opponent, and uses this information to pick the best combination of strategy parameters for future games in a rule-based manner. The majority of the bots rules for unit control and micromanagement are simple rule-based behaviors based on expert knowledge prioritization.

We can observe over the past few years that StarCraft AI bots are indeed getting stronger overall. In the AIIDE and CIG competitions, several bots from previous years are intentionally left in the next year to serve as a benchmark for progress, and we see each time that these benchmark bots do worse over time. Also, expert players and enthusiasts observe replays and note how they feel bots have gotten better or worse over time. Most notably, many of these expert players feel that the bots have been gradually adapting a more "standard" playing style than earlier bots, who traditionally did one strategy such as a rush, but not much else. More modern bots have developed mid and even late game tactics that were not seen in earlier rushing bots. Overall, bots seem to be getting better at army composition, build-order selection, building placement, and overall game strategy.

While the strongest bots currently play at an amateur human level, expert players have noted that they still appear to be weak in a few key areas. Most importantly, bots still seem quite weak at adapting their strategies dynamically during a match in response to information gained about their opponent. The majority of bots employ a playbook of several strategies that they choose from at the start of a match and follow through to the end of the game, with only a few bots attempting to dramatically change things if the opponent does something unexpected. This means that bots are still quite vulnerable to human players who are more easily able to change strategies and tactics as a game goes on. Current bots also seem quite vulnerable to the human ability to quickly identify bot patterns and behavior, and exploit this quickly during a match. For example, one human player during a human versus machine match noted that one bot unit would chase his Zergling when it got close to the bots' units, and proceeded to run the Zergling next to the bot army, and then lead the bot on a wild goose chase throughout the entire map. The entire time, the bot may have been reasoning that its army could win the fight against the single Zergling unit while not realizing that the human was just buying time until its army

[13]https://github.com/dgant/PurpleWave
[14]http://satirist.org/ai/starcraft/steamhammer/
[15]https://github.com/tscmoo
[16]https://bitbucket.org/auriarte/bwta2
[17]https://github.com/davechurchill/ualbertabot
[18]https://github.com/davechurchill/commandcenter/
[19]https://github.com/chriscoxe/ZZZKBot

Fig. 5. Tournament Manager software running in a computer laboratory for the CIG competition, which distributes bot games to multiple machines.

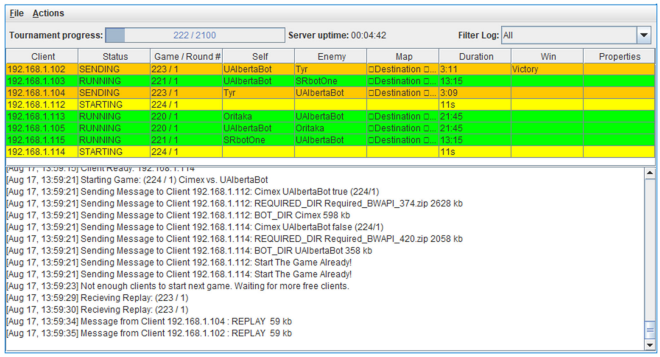

Fig. 6. Tournament Manager server GUI.

was ready for the final attack. This also illustrates one of the biggest challenges in all of artificial intelligence: understanding the long-term effects of actions that have delayed rewards. An expert human is able to quickly understand that they are being exploited in such a way, and that it will have negative effects down the road, and is able to stop the behavior. This long-term vision that is so intuitive to humans remains a problem for current RTS AI.

VI. Tournament Manager Software

All three StarCraft AI competitions covered in this paper use the same open-source tool (with different parameters) to automate the bot games. The tool is called StarCraft AI Tournament Manager (TM)[20] and was created/maintained by David Churchill and Richard Kelly for the AIIDE competition. It allows tournaments of thousands of bot versus bot games to be played automatically on any number of physical or virtual machines, as shown in Fig. 5. The original version of the software was created in 2011 for the AIIDE StarCraft AI Competition. The CIG StarCraft AI Competition has used the TM software since 2012, and SCCAIT has used a modified version of the Tournament Manager since 2014.

The TM software supports both round robin and one versus all tournaments for testing one bot against others. The server stores all bot and map files, as well as results and replay files generated by the *BroodWar* clients. Files are sent over Java sockets between the server and client machines. The Tournament Manager supports bots using different versions of BWAPI, and support for new versions can easily be added, allowing bots written in any version of BWAPI to play in the same tournament. Each client machine currently requires an installation of *StarCraft: BroodWar* version 1.16.1.

The TM software uses a server–client architecture distributed over multiple physical or virtual machines connected via LAN, with one machine acting as a server (coordinating the matchups and processing results) and any number of other machines acting as clients (running the bots and *StarCraft*). The tournament manager is written entirely in Java. The clients should run on Windows machines due to system requirements of *StarCraft*, whereas the server is fully platform independent. All the data sent and received are compressed and passed through Java sockets over TCP/IP.

A. Server

When running the software, one machine acts as a server for the tournament. The server machine holds central repository of all the bot files, their custom I/O data files, cumulative results, and replay files. The server monitors and controls each client remotely and displays the tournament's progress in real time via the server GUI (see Fig. 6), also writing the current results to an HTML file every few seconds.

The server program has a threaded component that monitors client connections and detects disconnections, maintaining a current list of clients. Each client is in one of the following states at all times (column "Status" in Fig. 6).

1) READY: Client is ready to start a game of *StarCraft*.
2) STARTING: Client has started the *StarCraft* LAN lobby but the match has not yet begun.
3) RUNNING: A game of *StarCraft* is currently in progress on the client machine.
4) SENDING: Client has finished the game and is sending results and data back to the server.

The server's main scheduling loop tries to schedule the next game from the games list every 2 s. A new game is started whenever both of these conditions are true:

1) two or more Clients are in READY state;
2) no clients are in STARTING state.

Once these two conditions are met, the server sends the required bot files, BWAPI version used by the bots, map file, and DLL injector to the client machines. The state of those clients is then set to STARTING.

Each client is handled by a separate thread in the server, and if the client is STARTING, RUNNING, or SENDING, it sends periodic status updates back to the server once per second for remote monitoring. Data updates include current game time, time-out information, map name, game ID, etc. When a client finishes a game, it also sends the results, I/O data files created by the bots and replay files, which are all stored on the server. This process is repeated until the tournament has finished.

Shutting down the server via the GUI will cause all the client games to stop and all clients to shut down and properly clean

[20] http://github.com/davechurchill/StarcraftAITournamentManager

up remote machines. The tournament can be resumed upon relaunching the server as long as the results file, games list, and settings files do not change. If the server is shut down with games in progress (results not yet received by the server), those games will be rescheduled and played again. The server GUI can send commands to the client machines, take screenshots of the client machine desktops, and remove clients from the tournament. Individual client machines can be added and removed without stopping the current tournament.

B. Client

The client software can be run on as many machines as needed. After an initial setup of the client machine (installing *StarCraft*, required libraries, etc.) the client software connects to the server machine via TCP/IP and awaits instructions.

The client machine will stay idle until it receives instructions from the server that a game should be run. Once the client receives the instructions and required files from the server, it ensures that no current *StarCraft* processes are running, ensures a clean working *StarCraft* directory, records a current list of the running processes on the client machine, writes the BWAPI settings file, and starts the game. When the game starts, a custom BWAPI Tournament Module DLL is injected into the *StarCraft* process. It updates a special GameState file on the hard drive every few frames—this is used to monitor the current state of the running *StarCraft* game. The client software watches this file to check for various conditions, such as bot time outs, crashes, game frame progression, and game termination. While the game is running, the client also sends the contents of the GameState file to the server once per second for centralized monitoring.

Once the game has ended or was terminated for any reason, the results of the game, replay files, and bot's I/O data files are sent to the server. After this, the client shuts down any processes on the machine, which were not running when the game began, to prevent crashed proxy bots or stray threads from hogging system resources from future games. *StarCraft* is shut down, the machine is cleaned of any files written during the previous game, and the client state is reverted to READY.

Since 2017, client machines can be labeled with custom properties such as extra ram or GPU, and bots can be labeled with matching custom requirements. Only clients that have all the requirements of a bot will be used for hosting that bot, and clients with special properties will be reserved to be used last to increase their availability for bots requiring them. The TM software also supports both DLL-based bots and bots with their own executable file that interface with BWAPI. This server-client architecture has allowed tens of thousands of games to be played in AI competitions each year.

VII. Conclusion

In this paper, we have given an overview of the three major annual StarCraft AI competitions, introduced the open-source software powering them, and described some of the top performing bot participants. As shown in Fig. 3, each year, participation in these competitions has continued to rise, as well as the number of games played between bots in the competitions. In the past two to three years, the bots in these competitions have become more strategically complex and functionally robust, employing a range of state-of-the-art AI techniques from the fields of heuristic search, machine learning, neural networks, and reinforcement learning. For many researchers, StarCraft AI competitions continue to be the environment of choice to showcase state-of-the art techniques for real-time strategic AI systems. With the increase in participation in StarCraft AI competitions, combined with the surge in activity in the field of RTS AI from hobbyist programmers, academic institutions, and industry researchers, we are hopeful that we may see a StarCraft AI capable of challenging human experts within the next few years.

References

[1] M. Buro, "Call for AI research in RTS games," in *Proc. 4th Workshop Challenges Game AI*, 2004, pp. 139–142.

[2] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 4, pp. 293–311, Dec. 2013.

[3] M. Buro and D. Churchill, "Real-time strategy game competitions," *AI Mag.*, vol. 33, no. 3, 2012, Art. no. 106.

[4] E. Gibney, "What Google's winning go algorithm will do next," *Nature*, vol. 531, no. 7594, pp. 284–285, 2016.

[5] A. Heinermann, "Broodwar API," 2013. [Online]. Available: https://github.com/bwapi/bwapi

[6] D. Churchill *et al.*, "StarCraft bots and competitions," *Encyclopedia of Computer Graphics and Games*. New York, NY, USA: Springer, 2016.

[7] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "RTS AI: Problems and techniques," in *Encyclopedia of Computer Graphics and Games*. New York, NY, USA: Springer, 2015.

[8] J. Malý, M. Šustr, and M. Čertický, "Multi-platform version of StarCraft: Brood war in a Docker container: Technical report," 2018, arXiv:1801.02193.

[9] B. P. Mattsson, T. Vajda, and M. Čertický, "Automatic observer script for StarCraft: Brood War bot games (technical report)," 2015, arXiv:1505.00278.

[10] A. E. Elo, *The Rating of Chessplayers, Past and Present*. New York, NY, USA: Arco, 1978.

[11] D. Loiacono *et al.*, "The 2009 simulated car racing championship," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 2, pp. 131–147, Jun. 2010.

[12] G. Synnaeve *et al.*, "Torchcraft: A library for machine learning research on real-time strategy games," 2016, arXiv:1611.00625.

[13] A. Uriarte and S. Ontanón, "High-level representations for game-tree search in RTS games," in *Proc. 10th Artif. Intell. Interactive Digit. Entertainment Conf.*, 2014.

[14] A. Tavares, H. Azpúrua, A. Santos, and L. Chaimowicz, "Rock, paper, starCraft: Strategy selection in real-time strategy games," in *Proc. 12th Artif. Intell. Interactive Digit. Entertainment Conf.*, 2016, pp. 93–99.

[15] P. Garćıa-Sánchez, A. Tonda, A. M. Mora, G. Squillero, and J. Merelo, "Towards automatic starCraft strategy generation using genetic programming," in *Proc. IEEE Conf. Comput. Intell. Games*, 2015, pp. 284–291.

Michal Čertický received the M.Sc. degree in cognitive science and Ph.D. degree in computer science.

He is currently Senior Researcher with the Artificial Intelligence Center, Czech Technical University, Prague, Czech Republic.

His research interests include machine learning and agent-based modeling. Since 2011, he has also been involved in computer game AI research. He is the Founder of the Student StarCraft AI tournament.

David Churchill received the M.Sc. degree in computer science and the Ph.D. degree in computing science.

He is currently an Assistant Professor in computer science with the Memorial University of Newfoundland, St. John's, NF, Canada.

Since 2011, he has been organizing and running the AIIDE StarCraft AI Competition. He is also the author of UAlbertaBot and CommandCenter StarCraft AI agents.

Kyung-Joong Kim received the B.S., M.S., and Ph.D. degrees in computer science from Yonsei University, Seoul, South Korea, in 2000, 2002, and 2007, respectively.

He is currently an Associate Professor with the Department of Computer Science and Engineering, Sejong University, Seoul, South Korea. His research interests include artificial intelligence, game, and robotics.

Martin Čertický is currently working toward the Ph.D. degree in the field of AI at the Department of Cybernetics and Artificial Intelligence, Technical University of Košice, Košice, Slovakia.

His research is focused on optimizing user experience in electronic entertainment and the use of artificial intelligence in video games.

Richard Kelly is currently working toward the Graduate degree at the Memorial University of Newfoundland, St. John's, NF, Canada.

Since 2016, he has been a co-organizer of the AIIDE StarCraft AI Competition.