StarCraft AI Competitions, Bots and Tournament Manager Software

Michal Čertický, David Churchill, Kyung-Joong Kim, Richard Kelly, Jan Malý and Michal Šustr

***Abstract*—Real-Time Strategy (RTS) games have become an increasingly popular test-bed for modern artificial intelligence 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 (SSCAIT), the Computational Intelligence in Games (CIG) competition, and the Artificial Intelligence and Interactive Digital Entertainment (AIIDE) competition. In this paper we will give an overview of the current state of these competitions, describe the bots that compete in them, and introduce an underlying open-source tournament manager software.**

I. Introduction

Real-time Strategy (RTS) games are a genre of video games in which players manage economic and strategic tasks by gathering resources, 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 a well-defined, complex adversarial systems [1] which 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].

Unlike turn-based abstract board games like 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 to that, 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 non-deterministic nature, makes 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].

Michal Čertický, Jan Malý and Michal Šustr are with the Artificial Intelligence Center at Faculty of Electrical Engineering, Czech Technical University in Prague. Emails: certicky@agents.fel.cvut.cz, jan.maly@aic.fel.cvut.cz, michal.sustr@aic.fel.cvut.cz

David Churchill and Richard Kelly are with Department of Computer Science, Memorial University of Newfoundland. Emails: dave.churchill@gmail.com, richard.kelly@mun.ca

Kyung-Joong Kim is with Department of Computer Science and Engineering, Sejong University. Email: kimkj@sejong.ac.kr

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 numerous scientific publications over last 8 years, dealing with all kinds of sub-problems inherent to RTS games. A comprehensive overview can be found in [6], [7] or [2].

In addition to AI research, StarCraft and BWAPI are often used for educational purposes as part of AI-related courses at universities, including UC Berkeley (US), Washington State University (US), University of Alberta (CA), Comenius University (SK), Czech Technical University (CZ), University of Žilina (SK) and most recently Technical University Delft (NL), where a new course entitled "Multi-agent systems in StarCraft" has been opened for over 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 existing until today. 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 other two annual competitions – Student StarCraft AI Tournament (SSCAIT), organized as a standalone long-term event at Comenius University in Bratislava and Czech Technical University in Prague, and CIG StarCraft AI competition collocated with 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 3 sections detailing the SSCAIT, AIIDE, and CIG Starcraft AI Competitions. We will also take a closer look at the state of the research and briefly describe current participants (bots) and AI methods they use. Finally, we will introduce the open-source Tournament Manager Software powering the competitions.

II. SSCAIT: Student StarCraft AI Tournament

The Student StarCraft AI Tournament (SSCAIT) is the StarCraft AI competition with the highest number of total participants. It started as a part of an AI course at Comenius University, and initial seasons included several dozen student

submissions from this course, in addition to submissions from across the globe. Since then, SSCAIT started accepting non-student participants and team submissions. 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 co-located with a scientific conference or any other real-world event.
2) There are two phases of SSCAIT each year: a competitive *tournament phase*, lasting for up to four weeks and a *ladder phase* which runs for approximately eleven months 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 24 hours a day. The AIIDE and CIG competitions instead play as many games as possible at maximum speed, with no public broadcast.

SSCAIT's *tournament phase* takes place every winter in late December and early January. At the time of writing of this paper, SSCAIT 2017/18 tournament was in progress with 78 participants. Since the 2017/18 results are still unknown, we cover the results of previous SSCAIT year.

SSCAIT 2016/17 Updates & News

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

First, the ladder phase was updated, with SSCAIT introducing so-called "weekly reports". Every weekend, there is a 1-2 hours long segment of curated AI vs. AI matches with insightful commentary on the live stream.

Second, the voting system was implemented, allowing bot programmers and viewers to select which bots will play the next ladder match on live stream (fig. 1). 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 itself was improved by perfecting the custom observer script [8] which now moves the camera fluently to 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 run in HD resolution using a "resolution hack" (fig. 2).

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

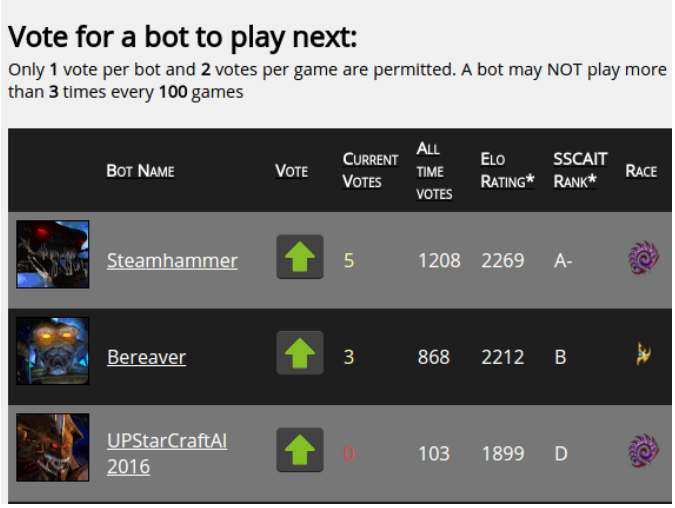


Fig. 1. A web-based interface allowing SSCAIT viewers and bot programmers to vote for the next ladder match.

Fig. 2. SSCAIT live stream running in HD resolution and controlled by the custom observer script [8].

Furthermore, two additional metrics were added to the ladder ranking system due to popular demand: ELO rating [9], which is commonly used in adversarial games like chess, and "SSCAIT rank", based on so-called "ICCUP ranking system", typical for competitive StarCraft.

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.

Tournament Phase Updates

The 2016/17 installment of SSCAIT's tournament phase took place during three weeks at the end of December 2016 and beginning of January 2017 and sported 45 participants. The tournament was divided into two divisions:

Student Division: Round Robin tournament of 1980 games, where every bot played two games against every opponent. Only the bots created by a single participant, who was a student at that time, 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 2016/17 were:

1) LetaBot (Martin Rooijackers), University of Maastricht (Netherlands) with 82 wins
2) Wulibot, University of Southern California (USA) with 63 wins
3) Zia Bot, Ulsan National Institute of Science and Technology (South Korea) with 54 wins

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

Mixed Division: After the student division ended, sixteen bots with the most wins among all the participants were selected for the additional mixed division elimination bracket. Since two bots (Steamhammer and Zia bot) were tied for 16th place, both with 54 wins, additional tie-breaker match was scheduled to decide who gets 16th position in the mixed division bracket.

The following 16 bots played in the mixed division elimination bracket: *LetaBot, Steamhammer, Overkill, WuliBot, ZZZbot, UAlbertaBot, IronBot, IceBot, Bereaver, BeeBot, XIMP, Skynet, Krasi0bot, Flash_AI, KillerBot* and *tscmoo*. First round (Ro16) matches consisted of only a single game, but Ro8 and Ro4 were played as "best of 3" matches. Finals were played as "best of 5".

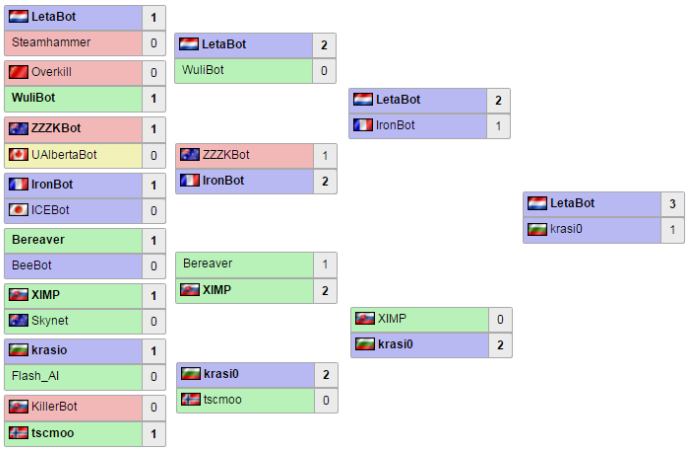


Fig. 3. SSCAIT 2016/17 mixed division elimination bracket.

Interestingly, *LetaBot* (created by Martin Rooijackers) managed to win the mixed division elimination bracket, in addition to winning the student division, after beating *Krasi0bot* (created by Krasimir Krystev) 3 to 1 in the finals (fig. 3). *LetaBot* had hard-coded special strategies against specific opponents. It successfully executed an "SCV rush" in the final games – a strategy for which *Krasi0bot* was not prepared.

All the elimination bracket games were published as videos with commentary on SSCAIT YouTube channel[2] and as replay files on SSCAIT website.[3] Tournament winner, Martin Rooijackers, has recently been invited to have a talk about this achievement at a business-oriented AI conference *World Summit AI*.[4]

III. AIIDE: 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 all bots be open source, and that their source code will be published for public download after the competition has finished. Running 24 hours a day for 2 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 2010 - The First StarCraft AI Competition

The AIIDE Starcraft AI Competition was first run in 2010 by Ben Weber at the Expressive Intelligence Studio at University of California, Santa Cruz, as part of the AIIDE (Artificial Intelligence and Interactive Digital Entertainment) conference. A total of 26 entrants competed in four different game modes which 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 was kept for bots to learn about opponents between matches. The 2010 competition had 4 different tournament categories in which to compete. Tournament 1 was a flat-terrain unit micro-management battle consisting of four separate unit composition games. Of the six competitors, FreSCBot won the competition with Sherbrooke coming in 2nd place. Tournament 2 was another micro-focused game with non-trivial terrain. Two competitors submitted for this category, with FreSCBot once again coming in 1st by beating Sherbrooke. 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. 8 bots faced off in this double-elimination tournament with MimicBot taking first place over Botnik in the final. As this was a perfect information variant of StarCraft, MimicBot adopted a strategy of "mimic its opponent's build order, gaining an economic advantage whenever possible" which worked quite well.

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 5 games. Competitors could play as any of the

[2]http://youtube.com/playlist?list=PLS6Qj916df6K9z3dOLQhCS0KatlvBqhbE

[3]http://sscaitournament.com/index.php?action=2016

[4]http://worldsummit.ai/

three races, with the only limitations in gameplay being those that were considered "cheating" in the StarCraft community. Since computer programs written with BWAPI have no limit to the number of actions they can issue to the Starcraft game engine, certain behaviours are possible which were not intended by the developers such as sliding buildings and walking ground units over walls, these sorts of actions are considered cheating and not allowed in the tournament. A map pool of 5 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 California, Berkeley, who defeated the Terran bot Krasi0 by Krasimir Krastev in the finals.

Overmind relied heavily on using the powerful and agile Zerg Mutalisk flying unit, which it controlled to great success using potential fields. Overmind's overall strategy consisted of an initial defense of Zerglings and Sunken Colonies (static defense towers) to protect its initial expansion while gathering enough resources to construct its initial Mutalisks. Once the Mutalisks had been constructed, it sent them to the enemy base and patrolled and harasses around the perimiter of the enemy base. If the bot did not win outright on the initial attack, it would slowly patrol and pick off any units which were undefended, eventually wearing down the enemy to the point where it could overwhelm it with a final attack. The 2nd place bot krasi0 played a very defensive Terran strategy, constructing Bunkers, Sige Tanks, and Missle Turrets for defense. After a certain number of units had been constructed it would then send an army of mechanical units to siege the enemy base. This bot performed quite well, only losing to Overmind during the competition. An excellent article about Overmind was written by Ars Technica in 2011.

B. 2011 - Automated Game Playing

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 a few reasons. One 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 we could no longer watch each game to detect if any cheating tactics were being employed, which would be more easily detected by inspecting the source code (which was not allowed to be heavily obfuscated). 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. The 2011 competition received 13 entrants.

The 2010 tournament was run Ben Weber on two laptops, and 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 results this elimination-style tournament was quite dependent on pairing luck, so for the 2011 competition we set out to eliminate all randomness from the tournament and play a round robin style format. Playing a round robin format requires far more games to be played, and would no longer be possible to run each game manually. In the summer of 2011, David Churchill, Jason Lorenz, and Michael Buro wrote the StarCraft AI Tournament Managing software (see section X) which could automatically schedule and play a round robin tournaments of Starcraft on an arbitrary number of locally networked computers. This software used a server/client architecture with a single server machine scheduling games and storing results, and each other machine running the client software which started the Starcraft game, monitored the game's progress and recorded the results when finished. Bot files, replays, and final results were accessed by clients via a shared Windows folder on the local network which was visible to all client machines. 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 10 total maps in the competition, chosen from expert human tournaments that were known to be balanced for each race, and 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.

At the end of the five day competition, Skynet placed 1st, UAlbertaBot placed 2nd, and Aiur placed 3rd. Skynet is a Protoss bot written by Andrew Smith, a software engineer from the United Kingdom, and used a number of solid Protoss strategies such as a Zealot rush, Dragoon / Zealot mid-game army, and a late game army consisting of Zealots, Dragoons, and Reavers. Its soloid economic play and good early defense were able to hold off the more offensive Protoss bots of UAlbertaBot and Aiur.

C. AIIDE 2016-17 Updates & News

The 2016 AIIDE competition had a total of 21 competitors, and the round robin games ran on 12 machines for nearly two weeks. A total of 90 round robin rounds were completed, with each bot playing 1800 games. No new rules or maps were used for the 2016 tournament that were not in place for the 2015 tournament. As the AIIDE competition was held shortly after the CIG competition, many of the submissions were the same, which is reflected in the results of both competitions. The top four finishers can be seen in Figure 4, with Iron placing 1st, ZZZKbot placing 2nd, and tscmoo coming in 3rd place.

Bot	Games	Win	Loss	Win %
Iron	1800	1574	226	87.44
ZZZKBot	1799	1530	269	85.05
tscmoo	1799	1488	311	82.71
LetaBot	1796	1329	467	74

Fig. 4. Results of the top 4 finishers in the 2016 AIIDE competition. Iron and LetaBot are Terran bots, while ZZZBot and tscmoo played as Zerg.

In 2016, the AIIDE competition website was updated to include an archive[5] of data and results of all of the annual AIIDE and CIG competitions, including final results, bot source code and binary download links, and information about each bot.

The 2017 AIIDE competition will have several updates:

- An updated map pool consisting of 10 new maps.
- Updated tournament managing software capable of playing more games in the same period of time.
- Support for BWAPI version 4.2.0
- Bots that achieved a 30% or higher win rate in the 2016 competition will be carried forward to 2017
- Plans to support GPU computation for bots

IV. CIG: Computational Intelligence in Games

The CIG StarCraft AI Competition has been a part of the program of the IEEE Computational Intelligence in Games (CIG) conference since August 2010. 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 met with several serious technical problems, which stemmed from the decision to use custom made Starcraft maps as opposed to traditional human-tested maps, which caused the Brood War Terrain Analysis (BWTA) software to crash for many of the bots. Because of these frequent crashes, it was decided that no winner could be announced for the competition. Mike Preuss and his team members organized the event from 2011 to 2013. Traditionally, open-source policy had not been enforced and the maps were unknown to competitors prior to the competition day. Since 2014, the Sejong University team (led by Kyung-Joong Kim) has organized the StarCraft AI competition at the IEEE CIG conference. Table I shows the change of the competition settings over years.

Since 2010, there have been multiple changes in the selection of tournament management software, open-source policy and map pool announcement policy. The tournament management (TM) software is used to distribute the matches over multiple machines on the network and to automate the competition operation (see the photo in Figure 5). Although CIG organizers developed their own JAVA-based TM sofware, the AIIDE TM has been used for the competition since 2014 (see details in Section VI).

[5]http://www.cs.mun.ca/~dchurchill/starcraftaicomp/archive.shtml

Year	Tourn. manager	Open-Source	Maps
2011	Manual play	Optional	Unknown
2012	AIIDE TM	Optional	Unknown
2013	Java-based TM	Optional	Unknown
2014	AIIDE TM	Forced	Unknown
2015	AIIDE TM	Optional	Unknown
2016	AIIDE TM	Forced	Announced
2017	AIIDE TM	Forced	Announced

TABLE I
The running settings of CIG StarCraft AI Competition.

Nowadays, the open-source policy is enforced and all the bot source codes are published after the competition. Unlike AIIDE competition, the map pool was not known to the participants before the competition to promote generalization ability of the entries. However, it was found out that participants usually did not exploit the map as a prior knowledge when they prepared the entries. Therefore, the set of competition's maps could be announced in advance.

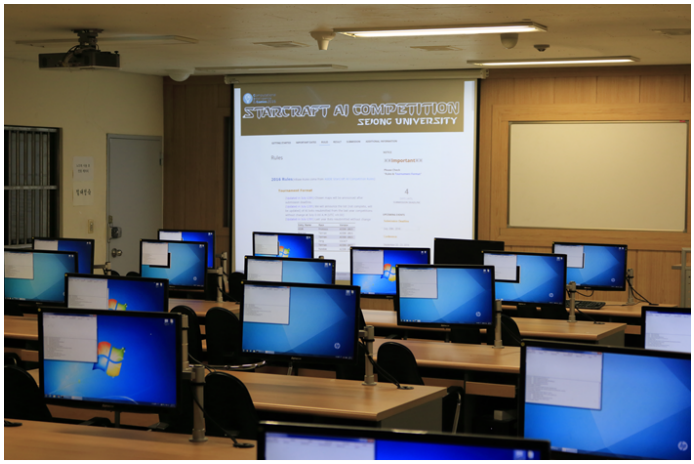


Fig. 5. In the computing lab, the Tournament Manager software distributes the game matches to multiple machines.

CIG 2016 and 2017 Updates & News

In 2016 competition, the organizers introduced the two-stage evaluation, where only half of the entries advance to the second stage based on the win ratio of the first stage. The concept of the qualification stage is not new. For example, the simulated car racing competition adopted the two-stage competitions divided into qualification stage and main race [10]. Since StarCraft AI competition is based on the full-round robin tournament, it is important to get high average win ratio against all the opponents. The organizers intended to reduce the chance that the top rankers exploit the low rankers to maximize their win ratio.

The final stage of the competition consisted only of the games among top rankers (top-8 players) without low-level weak players. Figure 6 depicts the change of rankings between the qualification and the final competition stage. We can see that Tscmoo bot replaced Iron bot as the stage winner.

The 2016 installment of CIG competition hosted a total of *16 participants*, out of which 9 were new or updated bots and

Rank	Bot	Win%
1	Iron	79.2
2	Tscmoo	76.97
3	LetaBot	74.07
4	Overkill	70.98
5	MegaBot	70.11
6	UAlbertaBot	69.25
7	ZZZKBot	69.18
8	Aiur	63.15

Rank	Bot	Win%
1	Tscmoo	65.14
2	Iron	54.43
3	LetaBot	53.71
4	ZZZKBot	53.08
5	Overkill	51.43
6	UAlbertaBot	49.07
7	MegaBot	38.00
8	Aiur	35.14

Fig. 6. The change of the rankings between the qualification stage (left) and main competition stage (right).

7 were re-entries from previous year. In total, 11988 round-robin games were played in the qualification stage and 2799 games in the final stage. All the games ran on 17 computers for 8 days.

Any persistent files accumulated by the bots in the qualifying stage were deleted before entering the finals. Best 8 bots, who competed in the final stage were: *tscmoo, IronBot, LetaBot, ZZZbot, Overkill, UAlbertaBot, MegaBot* and *Aiur*. The winner of the final stage, *tscmoo* bot created by Vegard Mella from Norway, became the overall winner of CIG 2016 tournament with 456 wins and 65.14% win rate in the final stage. Detailed results are depicted in Figure 7.

Bot	Win %	tscm	Iron	Leta	ZZZK	Over	UAlb	Mega	Aiur
tscmoo	65.14	-	52/100	44/100	79/100	71/100	77/100	83/100	50/100
Iron	54.43	48/100	-	38/100	49/100	49/100	74/100	30/100	93/100
LetaBot	53.71	56/100	62/100	-	49/100	81/100	69/100	30/100	29/100
ZZZKBot	53.08	21/100	51/100	51/100	-	42/100	35/99	93/100	78/100
Overkill	51.43	29/100	51/100	19/100	58/100	-	43/100	81/100	79/100
UAlbertaBot	49.07	23/100	26/100	31/100	64/99	57/100	-	76/100	66/100
MegaBot	38	17/100	70/100	70/100	07/100	19/100	24/100	-	59/100
Aiur	35.14	50/100	07/100	71/100	22/100	21/100	34/100	41/100	-

Fig. 7. Detailed results of the CIG 2016 competition final stage.

In 2017 competition, the two-stage format was changed back to the single round format. The reason was that the participants did not seem to change their strategy to consider the two-stage tournament.

Note: 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 system usually produces the final outcome similar to round-robin with a smaller number of rounds.

In 2017 CIG competition, organizers tried to increase the number of rounds and finally reached 125 rounds with 190 games per round. Although it required 47,500 games running on 22 machines for two weeks, it allows for a better understanding of the bots' win count dynamics over time. Currently, AI bots often use multiple pre-prepared strategies and adapt them or their selection against specific opponents. Long gameplay experience allows them to learn which strategies are good against which opponents. Figures 8 and 9 depict the win counts of specific bots over time. We can see that the win count tends to increase for some bots and decrease for others - depending on the machine learning approaches they take.

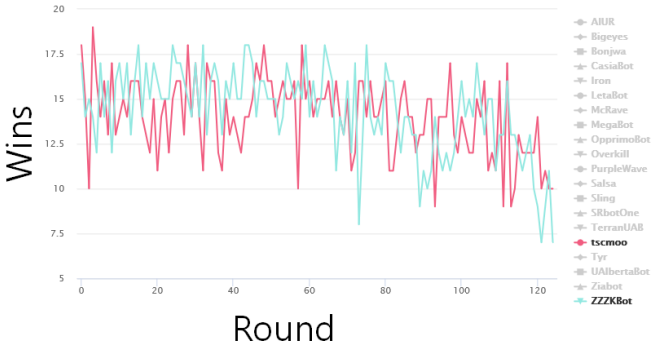


Fig. 8. The change of win count over multiple rounds for ZZZKBot (1st ranker, blue), and TSCMOO (2nd ranker, red).

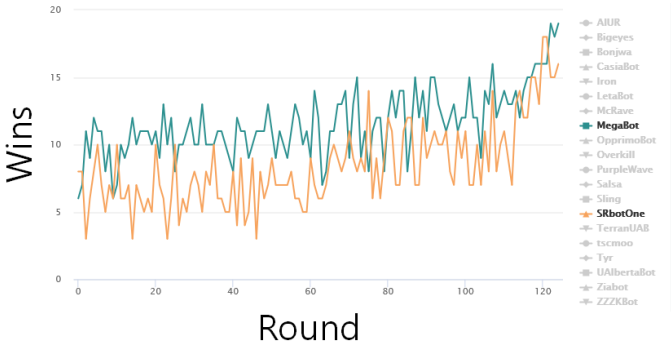


Fig. 9. The change of win count over multiple rounds for MegaBot (6th ranker, blue) and SRBotOne (14th ranker, orange).

After the 2017 competition, at 31st of October 2017, Sejong University organized a special event where human players were matched against the AI bots (Figure 10). 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, the winner of CIG 2017 competition, tscmoo bot, 2nd ranker in CIG 2017, and MJBOT, an AI bot specially designed against human players. The 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 (9 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 the 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 managed to win against all the AI bots. This suggests that the AI bots have a potential to compete against novice and middle-level players, but are not yet at the level of professional gamers. All the results and replay files can be found at http://cilab.sejong.ac.kr/.

V. Current StarCraft Bots

Over the years, StarCraft AI competitions have motivated many individuals and groups to combine and integrate various AI techniques and methods into complete bots, capable of playing complete StarCraft 1v1 games.

Fig. 10. Professional player Byung-Gu Song playing against AI bots.

In this section, we provide an overview of a selection of bots (in alphabetical order) and discuss some of the AI approaches they implement. We only mention those bots that are currently active in one of the competitions, have recently been updated, and employ some more complex AI techniques (we do not mention simple hard-coded or rule-based bots).

- *AIlien:* AIlien is a relatively new Zerg bot. Various kinds of its decisions rely on "scoring" systems. Other than that, it employs state machines – especially for higher-level strategy and macro decisions.

- *CherryPi:* CherryPi is a TorchCraft [11] Zerg bot developed by Facebook AI Research team. It uses UCB1 algorithm to learn which opening to play against each opponent and the blackboard architecture for communication between bot's modules.

- *cpac:* Zerg bot caled cpac combines hard-coded rules with a multi-layer perceptron network for unit production. The network is trained on state-action pairs extracted from datasets available at http://www.starcraftai.com/wiki/StarCraft_Brood_War_Data_Mining. The core of cpac bot is based on the Steamhammer bot (see below).

- *ForceBot:* ForceBot is a Zerg bot written in GOAL language – an agent-based programming language designed on top of BWAPI for programming cognitive agents.[6] Since the GOAL language is designed to implement multi-agent systems, all the ForceBot's units have their own corresponding agent with specific beliefs and goals. Each agent more or less follows a rule-based AI pattern.

- *GarmBot:* GarmBot is organized as a multi-agent system using the blackboard architecture. Every unit is controlled by a single agent, implemented as a state machine.

- *Ian Nicholas DaCosta:* This Protoss bot uses genetic algorithms for targeting, and detecting enemy army threat levels. Supervised learning was applied for the detection of opponent's strategies and builds.

- *Iron:* Iron bot is a decentralized multi-agent 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 the resources should be spent and what units should be trained, based on heuristics. Main motivation for this approach is robustness of the overall behavior (avoiding the blocking situations, indecisions and predictability).

- *KaonBot:* For resource and unit allocation based on prioritized needs, KaonBot applies a competing priority algorithm. According to the author, the bot will learn these priorities from experience in future releases.

- *KillAll:* The KillAll bot is based on Overkill bot by Sijia Xu and most of its functionality is hard-coded. However, its production module uses Q-learning to select unit types to produce, based on current situation.

- *Krasi0bot:* Krasi0bot has been around for many years, but it is still being actively developed. Even though it originally started as a rule-based bot, it currently makes some use of genetic algorithms, neural networks and potential fields. The author also actively experiments with various other techniques at the moment.

- *LetaBot:* The most interesting technique used by Martin Rooijackers' LetaBot is Monte Carlo Tree Search (MCTS) algorithm, which is used to plan the movement of squads (groups of units) around the map. A similar approach has previously been used by the author of Nova bot, Alberto Uriarte [12]. An implementation MCTS algorithm for squad movement has also been very recently released in form of ready-to-use library StarAlgo.[7] In addition to MCTS, LetaBot employs cooperative pathfinding for resource gathering and text mining to extract build orders directly 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.

- *MegaBot:* For every game, MegaBot [13] chooses one of three approaches, each of which is implemented as a different bot (Skynet, Xelnaga or NUSBot). Algorithm selection is modeled as a multi-armed bandit. At the beginning of the game, an algorithm is selected using epsilon-greedy strategy. After the game, the reward is perceived (+1, 0, -1 for victory, draw and loss, respectively)

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

[7]http://github.com/vnikk/StarAlgo

[8]http://wiki.teamliquid.net/starcraft

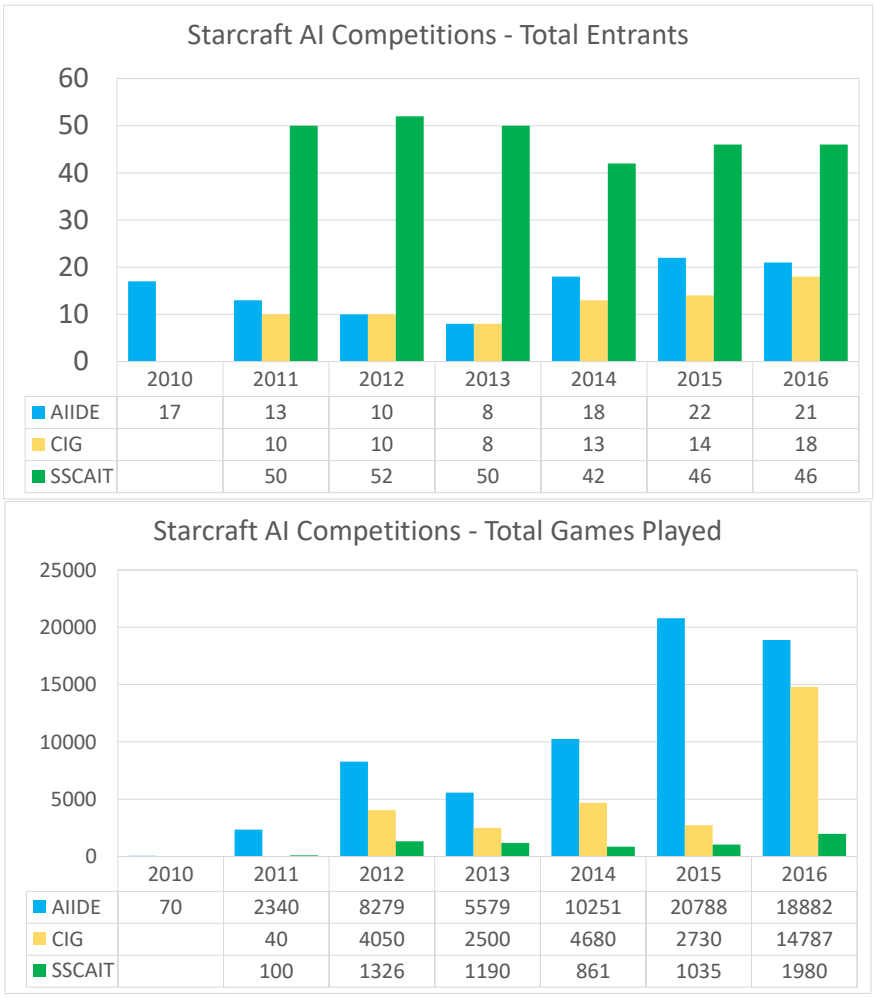


Fig. 11. Statistics for each of the 3 major annual StarCraft AI Competitions: AIIDE, CIG, and SSCAIT, since the first competition in 2010. Shown on the left is the number of total entrants for each competition, and on the right are the total number of games played in each competition.

and the value of the selected algorithm is updated via an incremental version of recency-weighted exponential average (Q-learning update rule).

- *Monica / Maria / Brenda:* Zerg, Protoss and Terran bots employing a game simulation inside the BEAM Erlang/OTP VM. It uses TorchCraft [11] – a library for machine learning research on RTS games. The unit logic is written in Lua.

- *PurpleWave:* The decision making of PurpleWave bot is mainly based on hierarchical task networks. For the micromanagement, it uses a hybrid squad/multi-agent approach and nearest neighbors clustering. The bot then simulates the outcomes of battles and suggests tactics for the 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 aproximately two dozen simple, reusable stateless behaviors. Uses heuristics including potential fields for the movement. The strategies are chosen based on results of previous games against the same opponent, race, map, and number of starting positions. It has a graph of strategy selections, like opening build orders paired with mid game transitions and late-game compositions.

- *StarcraftGP:* StarcraftGP is the first StarCraft meta-bot – a program that autonomously creates a program that autonomously plays StarCraft [14]. Currently, StarcraftGP v0.1 is using (Linear) Genetic Programming and it is able to directly write C++ code. Its first creations, namely Salsa and Tequila, have been the first bots not written by a human to participate in international competitions.

- *Steamhammer / Randomhammer:* Zerg bot Steamhammer, developed by Jay Scott, and its random-race version Randomhammer both employ sophisticated combat simulation with alpha-beta search and portfolio search 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, while Zerg production is currently rule-based.

- *tscmoo:* tscmoo uses no external libraries: it has its

own combat simulation code to predict the outcome of battles (while others typically use the SparCraft combat simulation package[9]), it does not use BWTA[10] to analyze the terrain and it even has its own threat-aware pathfinding for individual units. The bot can use many different strategies and selects among them 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.

- *Václav Bayer:* Q-learning / Reinforcement Learning and Markov Decision Processes are used by this bot – mainly to select the best build order with respect to opponent's strategy.

- *Zia Bot:* Despite the fact that most of Zia Bot's functionality is heuristic and rule-based, the bot tries to recognize and remember all the strategies used by its opponents and by itself. This memory is used to select the better performing game plans in the following games.

- *ZZZbot:* This Zerg bot, developed by Chris Coxe, remembers all the games against specific opponents and uses this information to pick the best combination of strategy parameters in a rule-based manner.

VI. Tournament Manager Software

All three StarCraft AI competitions covered in this paper use the same open-source tool, with varying number of modifications, to automate the bot games. The tool is called StarCraft AI Tournament Manager (TM) [11] and was created / maintained by David Churchill and Richard Kelly for the AIIDE competition. It allows tournaments of thousands of one-on- one games to be played automatically. 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 vs. 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, while the server is fully platform-independent. All the data sent and received is compressed and passed through Java sockets over TCP/IP, so no special network configuration is required.

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 Figure 12). It can also output the results in HTML format (see Figure 13).

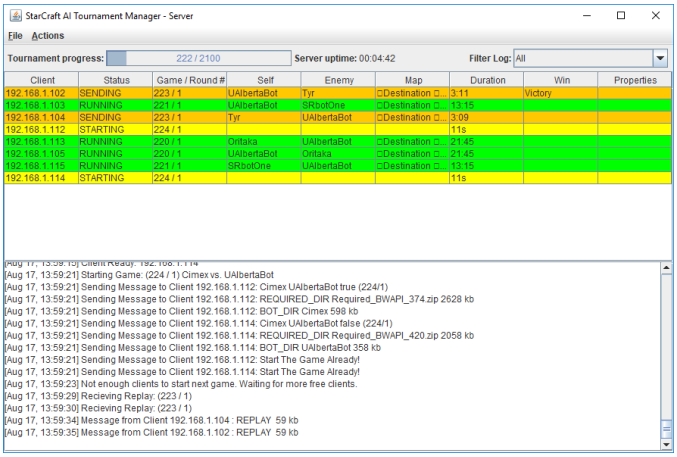


Fig. 12. Tournament Manager server GUI

The server program has a threaded component which 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 Figure 12):

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

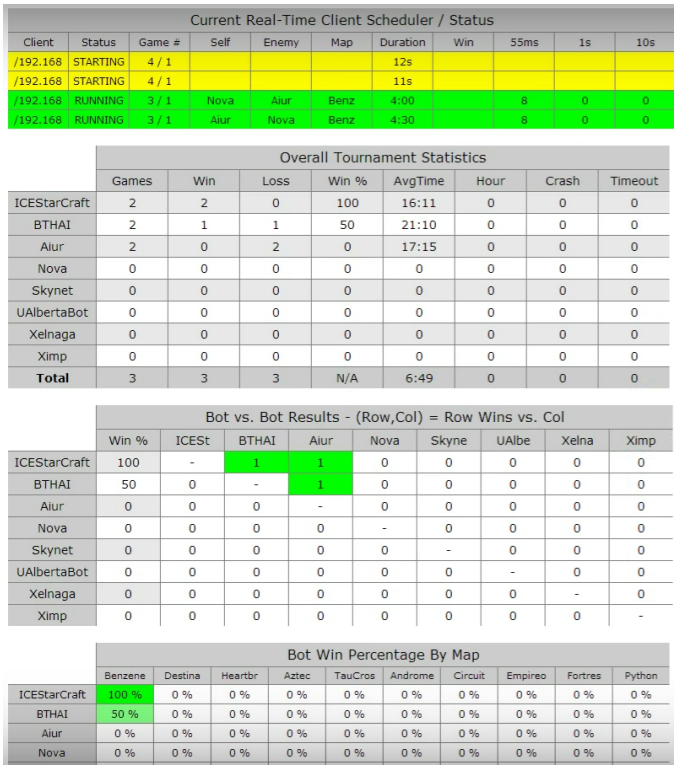
The server's main scheduling loop tries to schedule the next game from the games list every 2 seconds. A new game is started whenever both of these conditions are true:

1) two or more Clients are in READY state, and
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

[9]http://github.com/davechurchill/ualbertabot/wiki/SparCraft-Home
[10]http://bitbucket.org/auriarte/bwta2
[11]http://github.com/davechurchill/StarcraftAITournamentManager



Current Real-Time Client Scheduler / Status

Client	Status	Game #	Self	Enemy	Map	Duration	Win	55ms	1s	10s
/192.168	STARTING	4 / 1				12s				
/192.168	STARTING	4 / 1				11s				
/192.168	RUNNING	3 / 1	Nova	Aiur	Benz	4:00		8	0	0
/192.168	RUNNING	3 / 1	Aiur	Nova	Benz	4:30		8	0	0

Overall Tournament Statistics

	Games	Win	Loss	Win %	AvgTime	Hour	Crash	Timeout
ICEStarCraft	2	2	0	100	16:11	0	0	0
BTHAI	2	1	1	50	21:10	0	0	0
Aiur	2	0	2	0	17:15	0	0	0
Nova	0	0	0	0	0	0	0	0
Skynet	0	0	0	0	0	0	0	0
UAlbertaBot	0	0	0	0	0	0	0	0
Xelnaga	0	0	0	0	0	0	0	0
Ximp	0	0	0	0	0	0	0	0
Total	3	3	3	N/A	6:49	0	0	0

Bot vs. Bot Results - (Row,Col) = Row Wins vs. Col

	Win %	ICESt	BTHAI	Aiur	Nova	Skyne	UAlbe	Xelna	Ximp
ICEStarCraft	100	-	1	1	0	0	0	0	0
BTHAI	50	0	-	1	0	0	0	0	0
Aiur	0	0	0	-	0	0	0	0	0
Nova	0	0	0	0	-	0	0	0	0
Skynet	0	0	0	0	0	-	0	0	0
UAlbertaBot	0	0	0	0	0	0	-	0	0
Xelnaga	0	0	0	0	0	0	0	-	0
Ximp	0	0	0	0	0	0	0	0	-

Bot Win Percentage By Map

	Benzene	Destina	Heartbr	Aztec	TauCros	Androme	Circuit	Empireo	Fortres	Python
ICEStarCraft	100 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %
BTHAI	50 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %
Aiur	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %
Nova	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %	0 %

Fig. 13. Tournament Manager results in HTML format

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 up remote machines. The tournament can be resumed upon re-launching 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 to 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 to 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 changed back 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 tournament manager also supports both DLL based bots and bots with their own executable file which interface with BWAPI.

VII. Conclusion

In this paper we have given an overview of the 3 major annual StarCraft AI competitions, introduced the open-source software powering them, and described some of the top performing bot participants. As seen in Figure 11, 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 2-3 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. While the bots currently play at an amateur human level, we hope that advancing RTS AI techniques coupled with the recent involvement of industry research, they will be able to compete with human experts in the next few years.

References

[1] M. Buro, "Call for AI research in RTS games," in *Proceedings of the 4th Workshop on Challenges in 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 Transactions on Computational Intelligence and AI in Games*, vol. 5, pp. 293–311, 2013.

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

[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," https://github.com/bwapi/bwapi, 2013. [Online]. Available: https://github.com/bwapi/bwapi

[6] D. Churchill, M. Preuss, F. Richoux, G. Synnaeve, A. Uriarte, S. Ontanón, and M. Certickỳ, "Starcraft bots and competitions," 2016.

[7] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "RTS AI: Problems and Techniques," 2015.

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

[9] A. E. Elo, *The rating of chessplayers, past and present*. Arco Pub., 1978.

[10] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lonneker, L. Cardamone, D. Perez, Y. Sáez *et al.*, "The 2009 simulated car racing championship," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 131–147, 2010.

[11] G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier, "Torchcraft: a library for machine learning research on real-time strategy games," *arXiv preprint arXiv:1611.00625*, 2016.

[12] A. Uriarte and S. Ontanón, "High-level representations for game-tree search in rts games," in *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[13] A. Tavares, H. Azpúrua, A. Santos, and L. Chaimowicz, "Rock, paper, starcraft: Strategy selection in real-time strategy games," in *12th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2016, pp. 93–99.

[14] P. Garćıa-Sánchez, A. Tonda, A. M. Mora, G. Squillero, and J. Merelo, "Towards automatic starcraft strategy generation using genetic programming," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 284–291.