Home Compete Results Rules Resources Contact

2013 AIIDE StarCraft AI Competition Report

Written By: David Churchill - dave (dot) churchill (at) gmail (dot) com

Report Outline

Introduction

The 4th annual AIIDE StarCraft AI Competition was held in August 2013, the 3rd year that it was hosted at the University of Alberta and organized by David Churchill and Michael Buro. From August 20th to 24th, the state of the art in StarCraft AI bots competed against each other 24 hours a day in 20-machine undergraduate computer lab in the Computing Science Daprartment at the U of A.

Custom tournament managing software was written to allow the automatic scheduling and running of games during this period, allowing 5600 games to be played in just a few days. With many bots incorporating AI techniques which learn about opponents over time, it was very important to play as many games as possible to ensure statistically significant results.

The video below shows the competition being run in 2011, which is identical to the 2013 setup:

Entrants

This year, 10 teams registered to compete in the competition, with 8 teams actually submitting bots. Two registrants were not able to finish their bots in time for the tournament.

Bot Name	Author	Affiliation	Race	Download
Aiur	Florian Richoux	University of Nantes	Protoss	bot / replays
BTHAI	Johan Hagelback	Blekinge Institute of Technology	Terran	bot / replays
ICE	Kien Quang Nguyen	Ritsumeikan University	Terran	bot / replays
Nova	Alberto Uriarte	Drexel University	Terran	bot / replays
Skynet	Andrew Smith	Independant	Protoss	bot / replays
UAlbertaBot	David Churchill	University of Alberta	Protoss	bot / replays
Xelnaga	Ho-Chul Cho	Sejong University	Protoss	bot / replays
Ximp	Tomas Vajda	Comenius University	Protoss	bot / replays

This was the lowest turn-out yet for an AIIDE tournament, however I believe that it is the strongest field we have ever had. With the addition of Ximp bot who has been dominating the SSCAI Tournament, we were definitely in for a strong tournament this year.

Why was the turn-out low for the tournament this year? I suspect it is the combination of a few different factors, but mostly the time it takes to create a bot to submit to a tournament coupled with the dominance of several bots who do well every year. I have been told by several prospective developers that it is quite intimidating to enter a tournament where the same few bots have been winning every year. This makes sense, however it should be known that top performing bots such as UAlbertaBot, Skynet, and Aiur have made their code publicly available and fully documented for modifcation and re-use in these tournaments. Curious authors should start with these resources, as there is no longer a need to create a bot from scratch.

Competition Format

The tournament format for 2013 was the same as 2012, with bots playing 1 vs. 1 in the full game of StarCraft: BroodWar. Bots play against each other in round-robin format with each bot playing against each other 20 times on each of the 10 maps, for a total of 200 full round robins. At the end of the tournament, bots are ranked based on their overall win percentage. No playoffs are played.

Bots are given access to read and write files throughout the tournament in order to implement AI techniques such as learning about opponents for strategy selection. Bots can continuously write to a specific 'write' folder, and read from a specific 'read' folder, with the contents of the 'write' folder being copied to the 'read' folder after each full round robin has been played. This means that bots have access to information from every previously completed round.

For full details on tournament format and rules, check the official competition rules section.

Why not StarCraft 2?

This is the question we always get asked when we tell people we are doing a BroodWar AI competition. This competition relies completely on BWAPI as a programming interface to BroodWar. BWAPI was created by reverse engineering BroodWar and relies on reading and writing to the program memory space of BroodWar in order to read data and issue commands to the game. Since any program that does this can essentially be seen as a map hack or cheat engine, Blizzard has told us that they don't want us to do anything similar for StarCraft 2. In fact, most of the StarCraft 2 EULA specifically deals with not modifying the program in any way. We are happy that Blizzard have allowed us to continue holding tournaments using BWAPI, and they have also helped out by providing prizes to the AIIDE tournament, however until their policy changes we will not be able to do the same for StarCraft 2.

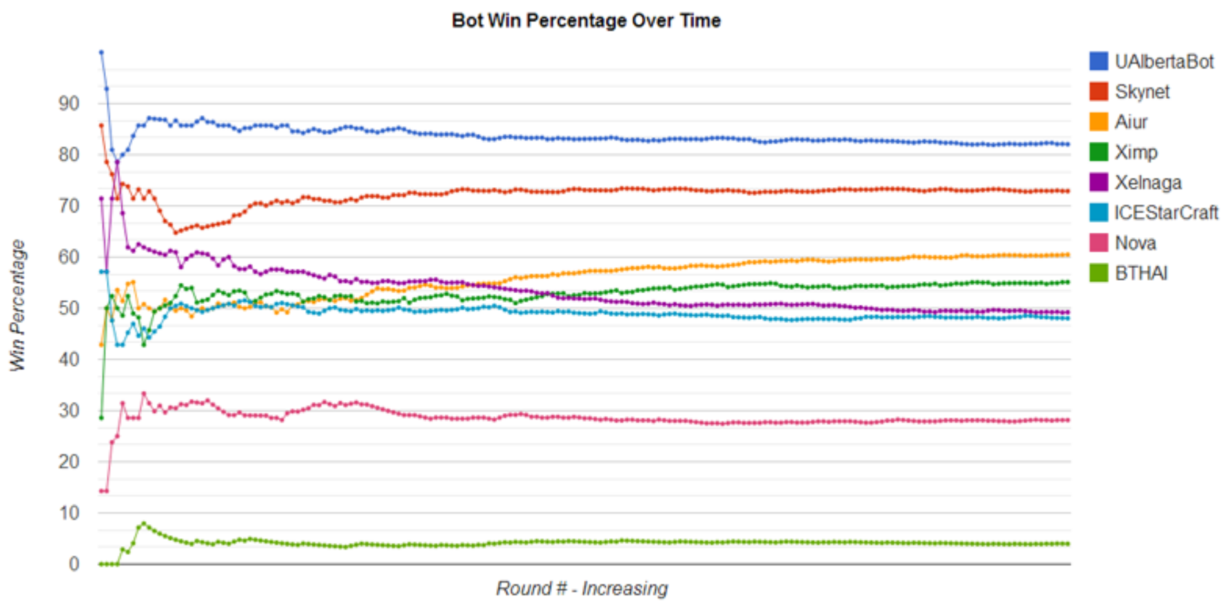
Results & Analysis

All 2013 Official Results Here: 2013 AIIDE StarCraft Competition - Final Results.

The following table shows the final win percentage of each bot and the wins and losses of each bot pairing. The value in cell (row,column) is the number of wins for the row bot vs. the column bot.

	Win %	UAB	Skynet	Aiur	Ximp	Xelnaga	ICE	Nova	BTHAI
UAlbertaBot	82.43	-	113	151	196	150	196	151	197
Skynet	72.77	87	-	169	92	183	95	192	200
Aiur	60.29	49	31	-	103	139	152	175	195
Ximp	55.29	4	108	97	-	112	114	161	178
Xelnaga	49.96	50	17	61	88	-	157	141	185
ICEStarCraft	47.82	4	105	48	86	43	-	192	191
Nova	27.47	49	7	25	39	58	8	-	198
BTHAI	3.93	3	0	5	22	15	8	2	-

Perhaps even more interesting than the final results are the results shown over time. In the following graph we can see the win percentage of each bot after each round of the tournament is completed.



Both Skynet and Aiur learn which strategy to choose against its opponents as the tournament progresses, and this is evident in the win percentage over time. Skynet quickly learns and gains about 10% win rate over 30-40 rounds, finally stabilizing at its final win %. Aiur seems to learn a little slower, but eventually moves from 6th to 3rd place through its strategy selection learning. UAlbertaBot used learning in 2012, but found that it had a single dominant strategy in that tournament, so no learning via file I/O was used this year.

An unfortunate bot was Xelnaga, who started out in the tournament quite strong but did worse as the tournament progressed, perhaps caused by the learning strategies present in other bots. Ximp bot unfortunately had some sort of bug in its programming that caused it to cease functioning entirely on the Fortress map. Below is a table that shows bot win percentage by each map played.

	Benzene	Heartbr	Destina	Aztec	TauCros	Empireo	Androme	Circuit	Fortres	Python
UAlbertaBot	80 %	73 %	82 %	75 %	86 %	85 %	87 %	87 %	79 %	87 %
Skynet	68 %	65 %	62 %	68 %	66 %	80 %	75 %	79 %	85 %	75 %
Aiur	61 %	67 %	66 %	55 %	52 %	56 %	59 %	52 %	68 %	62 %
Ximp	60 %	56 %	71 %	59 %	62 %	57 %	62 %	52 %	0 %	70 %
Xelnaga	48 %	56 %	45 %	61 %	50 %	53 %	31 %	45 %	62 %	43 %
ICEStarCraft	50 %	54 %	48 %	52 %	57 %	41 %	45 %	47 %	52 %	28 %
Nova	30 %	23 %	20 %	24 %	22 %	22 %	36 %	28 %	35 %	30 %
BTHAI	0 %	2 %	2 %	2 %	1 %	2 %	2 %	7 %	15 %	2 %

Additional statistics are also kept about each game which are shown in the next table.

	Games	Win	Loss	Win %	AvgTime	Hour	Crash	Timeout
UAlbertaBot	1400	1154	246	82.43	9:46	7	0	4
Skynet	1399	1018	381	72.77	11:29	4	0	5
Aiur	1400	844	556	60.29	13:47	31	0	25
Ximp	1400	774	626	55.29	16:06	10	0	202
Xelnaga	1399	699	700	49.96	16:08	52	0	72
ICEStarCraft	1399	669	730	47.82	15:49	30	0	30
Nova	1398	384	1014	27.47	14:12	22	0	43
BTHAI	1399	55	1344	3.93	14:26	0	0	79
Total	5597	5597	5597	N/A	13:58	78	0	460

The **AvgTime** column indicates the average game length of a bot in minutes:seconds. We can see from these results that UAlbertaBot's rush tactics definitely resulted in quicker games than any other bot, with its average game being over 4 minutes faster than the tournament average, and over 1 minute faster than UAlbertaBot's

2012 average.

The **Hour** column shows how many games a bot took to the hour time limit, for which the game was then decided by in-game score. Games typically only go to the full hour when one bot fails to find the last remaining buildings of a vanquished opponent. In 2013 bots went to the hour time limit an average of 1.39% of their games, vs. the 2012 average of 3.24% of their games. This seems to indicate that bots are getting more intelligent in the late-game and are more able to hunt down their opponents and finish them off.

The **Crash** column indicates how many times the bot crashed during the competition. A crash in this sense is a programming error that caused the bot's program to terminate, resulting in a windows error shutting down StarCraft. From the results it appears that this didn't happen at all during the tournament! This is either a huge improvement in the quality of bot programming, or an error in our tournament managing software to report crashes as timeouts. I apologize if this is the case, and I will try to figure this out as soon as possible.

The **Timeout** column indicates how many times a bot 'timed out', resulting in a game loss. The exact conditions for a time out are specified on our rules page, but the main two reasons are when a bot takes more than one minute on a single frame (usually caused by infinite loops), or when a bot has more than 320 frames which take more than 55ms to compute. Since this is a real-time strategy game, we want to enforce strict rules on bots taking too long to decide on what to do. Of note in these statistics is Ximp bot, who timed out in over 14% of its games played. This, coupled with Ximp bot's inability to perform on one of the tournament maps resulted in much worse results than expected from the SSCAI leader bot. On inspection of Ximp bot during the running of the tournament I can confirm that once it got a lot of carriers in play its play slowed quite dramatically, and the timeouts are indeed justified.

AI Techniques & Bot Details

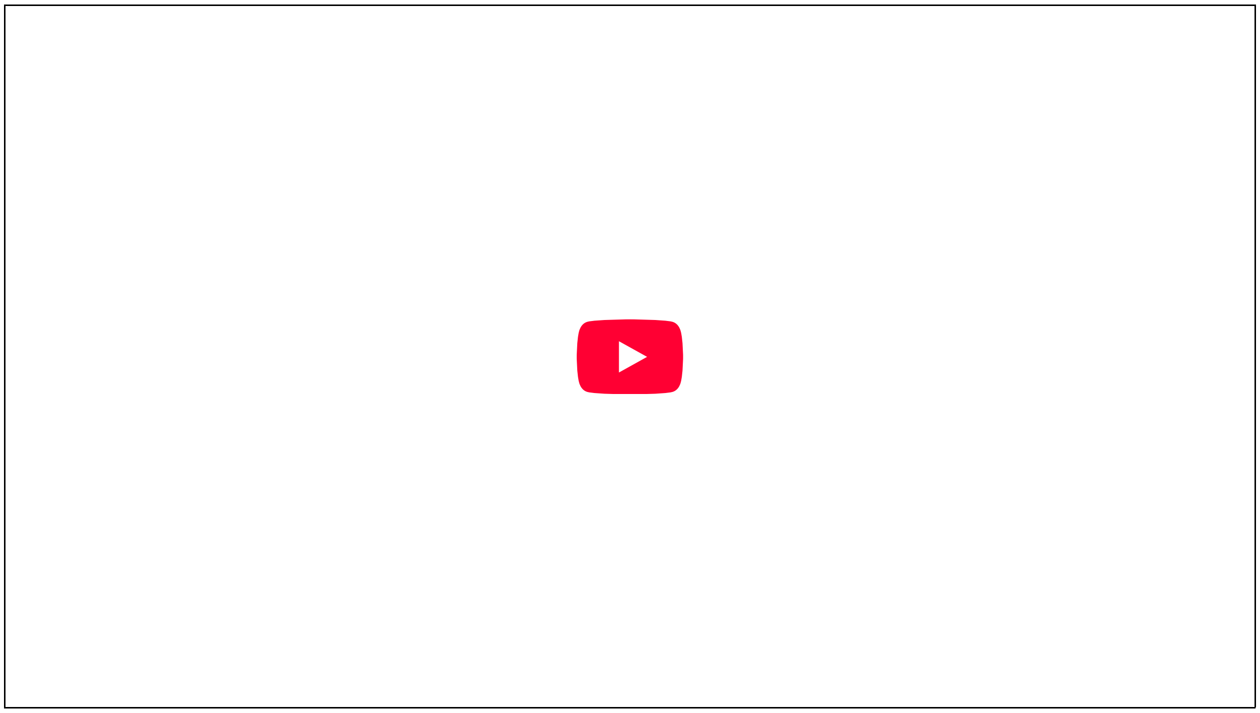
For an excellent up to date overview on the state of the art in StarCraft AI techniques and bot artchitecture descriptions, I highly recommend reading the following journal paper:

- A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft [2013]
  S. Ontanon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss
  Accepted to TCIAIG (August 2013)

Here are bot-specific details which I will post as I receive them from bot authors:

UAlbertaBot (Description by David Churchill)

UAlbertaBot's source code, documentation and AI Systems overview are available on the UAlbertaBot Google Code Project. Here is an overview video of UAlbertaBot made for CIG 2013:



As this year's winner, many people have asked how we were finally able to beat Skynet, as we had come in 2nd place to them just weeks prior at the 2013 CIG StarCraft Competition. There were several reasons:

- UAlbertaBot now uses SparCraft to do combat simulation to predict attack success / failure so it can retreat or advance on an opponent. No hard coded attack timings are used in the bot. (Was present for CIG 2013)
- After CIG UAlbertaBot's initial rush build order was changed to be a little quicker, allowing zealots to arrive at the opponents base about 15 seconds earlier.
- The worker manager of UAlbertaBot was tuned to better allocate newly created workers to mineral patches. This may not sound like much, but this leads to savings of up to 10 seconds even on a simple zealot rush.
- Several cases occured where UAlbertaBot's zealots or workers would chase enemy worker scouts inside its own base, which would almost always result in a loss since it delayed its rush. This was fixed so that a single worker would now chase enemy scouts.
- UAlbertaBot chose a Dark Templar rush against any bot named "Skynet". In 2012 Skynet was weak to DT rushes and in 2013 it seemed that hadn't changed. While this didn't outright win UAlbertaBot the tournament, it helped the results against Skynet specifically. This tactic of exploitation is often employed by human players in tournaments as they are aware of who they are playing against in advance.
- The absense of AdjutantBot (A bot we had a losing record against at CIG) and the presence of Ximp bot (A bot we beat almost all the time) helped our results.

Aiur (Description by Florian Richoux)

The starting idea of the Artificial Intelligence Using Randomness (AIUR) was to make stochastic choices in order to be unpredictable. Even if this goal is difficult to reach, we try since 2011 to stay on course. AIUR source code in under the GNU GPL v3 license and can be found on the Aiur Google Code Project

At the beginning of a game, AIUR randomly picks up one of the five available "moods":

- Cheese, where we try to locate as soon as possible the enemy base and build photon cannons near to its mineral line, in order to smash its economy during the early-game. It is a risky strategy, since if our probe or buildings are revealed and destroyed, AIUr has no units to defend itself.
- Rush, where it tries to -more or less- quickly produce a couple of Zealots and see if there is something to easily destroy in the enemy's main base or natural extension. Then, it continues on its classical Zealot - Dragoon - Archon composition (plus Corsair if air units or buildings have been scouted), with an aggressive behavior (we try to often attack).
- Aggressive, which is a modified mood for 2013 competitions. It now performed a fast drop zealot into the

enemey's mineral line early in the game to avoid ground defenses and break its economy. Fast Expand, where expanding is the very first thing we do.

- Macro, which is a regular game, with the regular AIUR's army composition.

Note that there exists a sixth mood named Defensive, which is impossible to be selected at the beginning of a game. So far, it only triggered against a Zerg opponent when we think it is rushing (for instance, when we scout its base and count an abnormally low number of drones).

One important aspect of AIUR is the ability to learn which moods are working well against a given opponent. Thus, for each map size (i.e., the number of starting spots) we try twice each mood before modifying the probability distribution of the mood selection, in order to favor efficient moods. Note that we don't necessarily select the best mood, or even an efficient mood each time: Indeed this is intentional to remains unpredictable.

The following changes are new since the 2012 competition:

- The learning mechanism was available last year, but did not take the map size as a parameter. Indeed, some moods can be efficient on 2-player maps (like Cheese), and some others on biggest maps like Fast Expand. Taking the map size as a new parameter makes sharper the learning. However, the initial learning phase was faster since it performed 20 games, with uniformly randomly selected moods, before modifying the probability distribution. Now, since we try twice each mood on each map size, for competitions like AIIDE and CIG, we need 30 games per opponent before exploiting our learning.
- The Agressive mood has been transformed into a fast drop zealot. This mood has been added to counter bots like Ximp, which makes a strong early defense at the base entrance but shows huge weaknesses if these defenses is avoided. Before, this mood was running like a regular macro game but with different attack timings.
- Micromanagement: A heuristic to focus enemy units during attacks has been added and was operational for CIG 2013, but has been disabled for AIIDE 2013 since it was not working well.
- The Photon Cannon rush (Cheese mood) has been improved to be more stealthy. Indeed, before the scouting probe was not very discreet when it penetrates the enemy base.

Further Discussion

After organizing three tournaments and authoring a well performing bot, here are my personal thoughts on where we need to improve the state of the art in StarCraft AI.

Where are StarCraft bots weak?

If the 2012 man vs. machine match (between Skynet (2012 winner) and Bakuryu, an A- human Zerg player) taught us anything, it's that bots are still very weak against human players. Humans are able to detect patterns of behaviour in bots which allow them to be exploited to extreme amounts. In this example, Bakuryu notices that Skynet's units will chase his zerglings if they are near, and proceeds to run around Skynet's base to distract Skynet long enough so that Bakuryu can make fying units to come attack Skynet's base. This type of behaviour is incredibly hard to detect in a bot, since it requires knowledge of the larger context of the game which may only have consequences 5 or more minutes in the future. If there was a single area of improvement that would make the most difference in man vs. machine play, it would be in detecting that your behaviour is being exploited.

Another area where bots seem weak is strategy switching. Human players often choose an initial strategy to play with (rush, turtle, expand, etc) and then switch this strategy based on observations of opponents. While this does occur to a small extend in bot play, it is nowhere near the level required to defeat humans. For example, UAlbertaBot has a strong early rush game, but if that strategy doesn't succeed it has a very poor transition into a late game strategy and often just tries to continue to overwhelm the opponent with early game units. We could build hand-code a large state machine to handle some strategy switching cases, such as the human rules of "if he is defending, then expand" or "if he is expanding, then attack", but does this advance the state of AI? More work is required in the field of identifying the need for a strategy switch, and the choosing of which strategy to switch to, without having to rely on expert knowledge. High-level strategy selection and long-term planning are definitely low-hanging fruit in RTS AI research.

Base construction and building placement (often called SimCity) is another area where very little work has been done with RTS AI. In high-level human Zerg play, much (if not all) of the success of the Zerg player relies on building placement. Due to the Zerg's production capabilities relying on the number of hatcheries (bases) it has, it is forced to expand early in the game to be successful. This expansion often delays the production of early game mobile combat units, which then requires expert placement of both buildings and defensive structures in order to create a maze-like structure so that enemies cannot easily attack the base. This is illustrated by the human player Bakuryu here in the 2012 man vs. machine match when he creates a wall of buildings to stop Skynet's incoming mass of zealots. Normally 3 sunken colonies would easily fall to that many zealots, but due to the maze created by Bakuryu, most of the zealots die before even being able to attack the defensive structures. Not only are bots weak in constructing these intricate structures, but as we can see they are also weak at attacking into these structures, as they require complex path finding and combat simulation to determine whether success is possible.

Where are StarCraft bots getting stronger?

Although it may seem trivial to people who have not tried to write a StarCraft bot, software architechure and solid programming hindered bot developers in the past due to the large number of crashes and time-outs in tournament settings. This has improved dramatically over the years we can see that bots are crashing less and finding solutions to complex problems in real-time that to not cause time-outs.

Bots have started learning! It is evident from the 2013 competition that bots like Skynet and Aiur have shown that learning is important, and can improve results dramatically over the course of a tournament. By having multiple strategies available in their bots and using learning techniques to select which strategies to implement, Skynet and Aiur were able to gain over 10% win rate in the competition. In 2012, competition entrant BroodwarBotQ employed learning techniques not only for strategy selection over time, but also to learn about strategies its opponents were implementing. As the tournament ran, BBQ gained more confidence about which strategy its opponent would choose and changed its strategy accordingly. If the community collaborated these learning techniques into a single agent, I believe that we would see performance dramatically improve in the future.

Complex real-time search algorithms and simulations are now being performed in real time in these competitions. UAlbertaBot incorporates a depth-first branch and bound heuristic search algorithm to search for time-optimal build orders during games. No hard coded build orders are used beyond the first 2-3 minutes of the game. UAlbertaBot also uses SparCraft, a StarCraft combat simulation package used to predict the outcome of battles. UAlbertaBot sends attacking units at its opponent's base with no hard-coded attack timings and relies on the outcome of SparCraft combat simulations to determine whether or not to keep attacking or retreat and wait for reinforcements. This video shows UAlbertaBot attacking Aiur's base, using combat simulation to retreat and advance as it predicts it will win. Techniques such as these help to lessen the reliance on expert knowledge which currently dominate the development of StarCraft AI bots.

When will we beat humans?

Ironically, I think that by the time bots get good enough to beat the best humans, there might not be any humans who are still good at BroodWar. But if someone like Flash stayed as good as he was at his prime, I would expect at least 5-10 years until bots could beat the best human players in a best of 7 match. I think that bots can beat amateur humans today in a single game by surprising them with rushes or strange tactics, but over a best of 7 match human players would be able to identify and exploit the strategies of the bots quite easily.

Highlights, Media & Replays

This section will be made available as I get time to sort through replays.