# TODO: A Survey of StarCraft AI Techniques

FirstName LastName, *Member, IEEE,* Jim Raynor, *Fellow, RR,* and Sarah Kerrigan, *Life Fellow, ZS*

***Abstract*—TODO: Idea of the paper is: "one-stop guide on what is the state of the art in Starcraft AI". It should help people participating in the competition focus their efforts, and also should help people implementing AI for RTS games in general (e.g. industry). In Gabriel's words "RTS AI problems, Solutions, State-of-the-art, conclude on what's "solved" since (Buro 2004) and what's not."**

**For example, if someone wants to implement a bot, and wonders "how should I do scouting", our paper should provide a summary of the existing techniques, and pointers to know more.**

***Index Terms*—Game AI, Real-Time Strategy, Starcraft, Review1**

## I. Introduction

SINCE Michael Buro's call for research in RTS games [1], many researchers have answered the call. Specially, AI competitions like the Starcraft one have caused many AI techniques to be applied to RTS AI. We will list and classify these approaches, explain their power and their downsides and conclude on what is left to achieve human-level RTS AI.

Motivate the paper, and provide an outline.

Some arguments to use in the motivation could be that games are a good application to motivate novel AI research (as has been happening throughout the history of AI), and that techniques and algorithms developed for RTS games, in addition to be useful and relevant to the game industry, have broader application to other areas.

Reiterate that the goal of this paper is to provide a one-stop guide on what is the state of the art in Starcraft AI

## II. Real-Time Strategy Games

### A. Starcraft

Introduce the game plus screenshot. Not too lengthy.

### B. Challenges in RTS Game AI

This section should be brief, and just give us some basic ideas of which are the challenges in RTS Game AI, so that when we refer to related work, we can use this as a reference point. The most important thing of this section is to demonstrate that RTS games are complex.

Many years have passed since Buro's call for research (8!!). he identified 6 challenges:

- Resource management
- Decision making under uncertainty
- Spatial and temporal reasoning
- Collaboration
- Opponent modeling, Learning
- Adversarial real-time planning

There has been a lot of work in many, but others have been untouched (e.g. Collaboration). Additionally, other challenges that are not in the list appeared, and have been worked on, like: how to exploit the existing domain knowledge (strategies, build-orders, replays, etc.), or how to design an architecture that integrates all the modules required for an agent to play an RTS games.

- Task Decomposition (or "Architecture")
- Integration of Domain Knowledge
- Reasoning with Uncertainty (including information gathering)
- Opponent Modeling and Adaptation
- Group and Individual Control ("micro")
- Planning and Resource Allocation ("macro")
- Spatial reasoning

Maybe the list above goes better in Section VI?

## III. Existing work on RTS Game AI

Include here the literature review of existing work on RTS Game AI. We could also compare with what is being used in industry, and use the real AI of Starcraft as an example (Bob Fitch's talk in AIIDE provides enough material for that). I have included here as subsections the two blurbs of text we had for integration of domain knowledge and reasoning with uncertainty, since I guess they correspond here.

All bots attempt at integrating human domain knowledge in a way or another, be it in the form of build-orders, heuristics, strategies, or by automatically learning from human replays. This has been shown to be key to good performance in bots (for now, as Dave was saying: "Starcraft AI is like Chess was in the 60s").

Talk about learning, scripting, and other techniques to incorporate such knowledge.

When talking about learning in RTS games, with regard to a particular match against an opponent, one has to differentiate between:

- prior learning: done before the match, like when a bot learns what is likely to happen on a given match-up (and map) from replays, or when a bot optimizes its micro-management parameters through simulated annealing or genetic evolution.
- in-match learning: done between the different games of match, "the opponent rushed me last game so I will play this one safe".
- in-game learning: done between the different actions of the game, "I tried to attack here and it was a failure, let us try somewhere else".

FirstName LastName is with the Department of Names GA, 30332 USA e-mail: (see http://www.michaelshell.org/contact.html).

J. Raynor and S. Kerrigane are with the Romeo&Juliet Inc.

Manuscript received April 19, 2499; revised January 11, 2500.

All these learning can be supervised (labeled replays, scripted situations) or unsupervised (unlabeled data, exploratory situations). The last two kinds of situations of learning are more inclined towards reinforcement learning because the AI has to deal with the exploration-exploitation duality.

I think there are two important aspects of uncertainty in RTS games:

- First is trying to model uncertainty and try to reduce it (information gathering, and representation of knowledge), this includes scouting, and maintaining good estimates of locations and number of units of the enemy, etc.
- Second is actually using the uncertain information. Taking decisions like army strength estimation (like to attack or retreat).

There are two main kinds of uncertainty, existing for two different reasons, in RTS games. First, there is the incompleteness of information, due to the fog of war, which leads to some kind of "extensional" uncertainty, that can be lowered by good scouting, and knowledge representation and reasoning (to infer what is possible given what has been seen). Also, there is the fact that we will never be able to read the opponent's mind, the "intentional" uncertainty that we can't really be sure to scratch off, even with the best scouting possible. For intentional uncertainty, the AI, as the human player, can only build a sensible model of what the opponent is likely to do, to think, or to think knowing what we think he thinks. Both these kinds of uncertainty can be studied through probabilities, but only $P(Hidden|Seen)$ can be *directly* linked to data only for extensional uncertainty. One could argue that the extensional data is the only thing that matters because it is the only thing that has an influence on the game, but in reality, for very highly skilled players, discovering the intentions of the enemy is the key to winning. It allows for a compact understanding of the game state and development, while also giving additional information about the future. Whereas a player is doing action A to follow it up by action B or C depends on its intention, while the information about action A is visible and information about B or C is hidden, only intention allows for a choice.

## IV. State of the Art Bots for Starcraft

Here, describe the existing top bots for Starcraft. The difference between this section and the previous, is that the previous focuses on existing ideas, techniques, etc. whereas this section focuses on how some of those have been put together to create specific bots that play Starcraft.

## V. The Starcraft AI Competition

Here describe the results obtained in the competition comparing the bots.

## VI. Open Questions in RTS Game AI

Here, describe what is left to be done.

## VII. Empirical Evidence of What is Missing in Existing Bots

Here, whatever experiments we want to include to support which aspects of RTS Game AI require more work.

## VIII. Discussion and Conclusions

## Acknowledgments

This research is partially funded by projects ... and ...

## Appendix

### This is the old task-division section

I've included here the section we had for not losing it :)

Here we should say that complex RTS games require multi-scale reasoning [2], and thus complex architectures that integrate many AI modules have been devised. Discuss how different researchers have divided up the task of playing Starcraft into a collection of subtasks and integrated them (see the end of this paper for a preliminary write-up on this)

Playing an RTS game involves dealing with all the problems described above. A few approaches, like CAT [3], Darmok [4] or ALisp [5] try to deal with the problem in a monolithic manner, by using a single AI technique. This resembles approaches to solve other games, such as Chess or Go, where a single game-tree search approach is enough to play the game at human level. However, none of those systems aims at achieving near human performance. In order to achieve human-level performance, RTS AI designers use a lot of domain knowledge in order to divide the task of playing the game into a collection of sub-problems, which can be dealt-with using individual AI techniques (as discussed in the previous section). Thus, an integration architecture is required to put together all of those techniques into a single coherent system, which is the focus of this section.

Figure 1 shows some representative examples of the integration architectures used by different bots in the AIIDE 2011 Starcraft AI competition [6]. Each box represents an individual module with a clearly defined task (only modules with a black background can send actions directly to Starcraft). Dashed arrows represent data flow, and solid arrows represent control (when a module can command another module to perform some task). For example, we can see how SPAR is divided in two sets of modules: *situation analysis* and *decision making*, the first with three modules dedicated to analyze the current situation of the game, and the later with 4 modules dedicated to exploit that information to decide what to do. We can see how the decision making aspect of SPAR is organized hierarchically, with the higher-level module (*strategic decision*) issuing commands to the next module (*tactical decision*), which sends commands to the next module (*action implementation*), and so on. Only the lower-level modules can send actions directly to Starcraft.

On the other hand, bots such as NOVA or Broodwar-BotQ (BBQ) only use a hierarchical organization for *micro-management* (controlling the attack units), but use a decentralized organization for the rest of the bot. In Nova and BBQ, there is a collection of modules that control different aspects of the game (workers, production, construction, etc.). These modules can all send actions directly to Starcraft. In Nova those modules coordinate only through writing data in a shared blackboard, and in BBQ they coordinate only when they have to use a shared resource (unit) by means of an arbitrator.

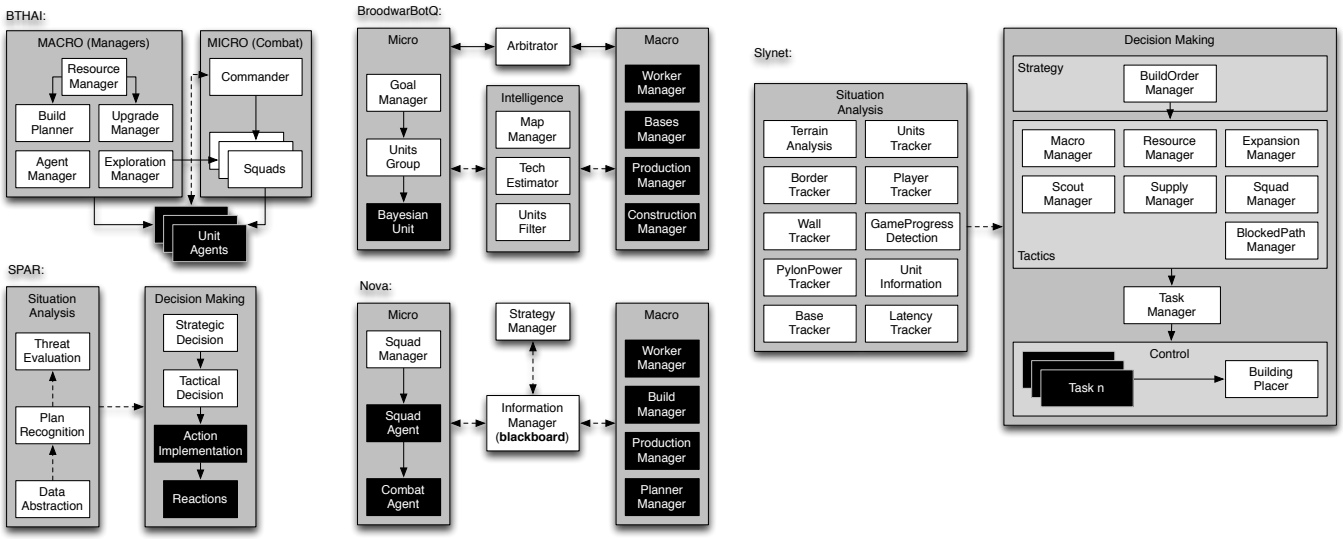


Fig. 1. General architecture of 5 Starcraft AI bots.

By analyzing the structure of these bots, we can see that there are two main tools that can be used when designing an integration architecture:

- *Abstraction*: complex tasks can be formulated at different levels of abstraction. For example, playing an RTS game can be seen as issuing individual low-level actions to each of the units in the game, or at a higher level, it can be seen as deploying a specific strategy (e.g. a "BBS strategy", or a "Reaver Drop" strategy). Some bots, reason at multiple levels of abstraction at the same time, making the task of playing Starcraft simpler. Assuming that each module in the architecture of a bot has a goal and determines some actions to achieve that goal, the actions determined by higher-level modules are considered as the goals of the lower level modules. In this way, each module can focus on reasoning at only one level of abstraction, thus, making the problem easier.
- *Divide-and-conquer*: playing a complex RTS, such as Starcraft, requires performing many conceptually different tasks, such as gathering resources, attacking, placing buildings, etc. Assuming each of these tasks can be performed relatively independently and without interference, we can have one module focusing on each of the tasks independently, thus making the problem easier.

If we imagine the different tasks to perform in a complex RTS game in a two-dimensional plane, where the vertical axis represents abstraction, and the horizontal axis represents the different aspects of the game (micro-management, resource gathering, etc.), abstraction can be seen as dividing the space with horizontal lines, whereas divide-and-conquer divides the space using vertical lines.

Different bots, use different combinations of these two tools. Looking back at Figure 1, we can see the following use of abstraction and divide-in-conquer in the bots:

- BTHAI: uses a two-tier abstraction hierarchy, where a collection of high-level modules command a collection of lower-level agents in charge of each of the units. At the high-level, BTHAI uses divide-and-conquer, having multiple high-level modules issuing commands to the lower-level units.
- BBQ: uses abstraction for micro-management, and divide-and-conquer for macro-management and intelligence gathering. To avoid conflicts between modules (since the individual tasks of each of the modules are not completely independent), BBQ uses an arbitrator.
- Nova: is similar in design as BBQ, and uses abstraction for micro-management, and divide-and-conquer for macro-management. The differences are that Nova does not have an arbitrator to resolve conflicts, but has a higher-level module (*strategy manager*), which posts information to the blackboard that the rest of modules follow (thus, making use of abstraction).
- SPAR: only uses abstraction. Its high-level module determines the strategy to use, and the tactical decision module divides it into a collection of *abstract actions*, that are executed by the lower-level modules.
- Skynet: makes extensive use of both abstraction and divide-and-conquer. Considering the decision making component of Skynet, we can see a high level module that issues commands to a series of tactics modules. The collection of tactic modules queue *tasks* (that are analogous to the abstract actions used in SPAR). Each different task has a specific low level module that knows how to execute it. Thus, Skynet uses a 3 layered abstraction hierarchy, and uses divide-and-conquer in all levels except the highest.

Additionally, except for BTHAI, all other agents use divide-and-conquer at a higher-level bot design and divide all the modules into two or three categories: *information gathering* and *decision*, or *information gathering*, *micro-management* and *macro-management*.

Notice that most bots using divide-and-conquer (except for BBQ), assume that each of the modules can act independently

and that their actions can be executed without interference. BBQ is the exception, including an arbitrator that makes sure that modules do not send contradictory orders to the same unit. However, very little bots handle the problem of how to coordinate resource usage amongst modules, for instance BTHAI uses a first-come-first-serve policy for spending resources, the first module that requests resources is the one that gets them. The exceptions are Nova and Skynet implement some rudimentary prioritization based on the high level strategy.

One interesting aspect of the five bots described above is that, while all of them are reactive at the lower level (unit control), most if not all of them, are scripted at the highest level of abstraction. BTHAI reads build and squad formations from a predefined script, Nova's *Strategy Manager* is a predefined finite-state machine, BBQ's construction manager reads the build order from a predefined script, and Skynet's *BuildOrder Manager* is basically a predefined script. Such scripts describe the strategy that the bots will use, however, such strategy is always fixed. One could see this pre-scripting as if each bot defined a "high-level programming language" to describe Starcraft strategies, and the bots themselves are just interpreters of such strategy. Compared to current approaches for Chess or Go, this scripting seems a bit rigid and inflexible, but responds to the much higher complexity of the Starcraft game.

In conclusion, we can see that there are two basic tools that can be used in an integration architecture: abstraction and divide-and-conquer, which are widely used by the existing Starcraft bots. However, several open questions remain:

- Strengths and weaknesses?
- Resource coordination in a divide-and-conquer approaches?
- High-level strategies that do not depend on scripting? (not here, this should be discussed in Section)

## REFERENCES


[1] M. Buro, "Real-time strategy games: A new ai research challenge," in *IJCAI 2003*. International Joint Conferences on Artificial Intelligence, 2003, pp. 1534–1535.

[2] B. G. Weber, P. Mawhorter, M. Mateas, and A. Jhala, "Reactive planning idioms for multi-scale game AI," in *IEEE Symposium on Computational Intelligence and Games (CIG)*, 2010.

[3] D. W. Aha, M. Molineaux, and M. J. V. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *ICCBR*, 2005, pp. 5–20.

[4] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "On-line case-based planning," *Computational Intelligence*, vol. 26, no. 1, pp. 84–119, 2010.

[5] B. Marthi, S. Russell, D. Latham, and C. Guestrin, "Concurrent hierarchical reinforcement learning," in *International Joint Conference of Artificial Intelligence, IJCAI*, 2005, pp. 779–785.

[6] "AIIDE starcraft AI competition," http://skatgame.net/mburo/sc2011/, [Online; accessed 27-Febrary-2012].


**Jim Raynor** Jim Raynor was a Confederate marshal on Mar Sara at the time of the first zerg incursions on that world. He is now with Raynor's Raiders Inc.