# 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 [1] 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 - Situation*

Starcraft: Brood War is a popular RTS game released in 1998 by Blizzard Entertainment. Starcraft is set in a science-fiction based world where the player must choose one of the three races: Terran, Protoss or Zerg. The good work done by the people of Blizzard makes this game one of the most extremely well-balanced RTS games ever created.

- Terrans, human exiled from Earth, provide units that are versatile and flexible giving a balanced option between Protoss and Zergs.
- Protoss units have lengthy and expensive manufacturing process, but they are strong and resistant. These conditions makes players follow a strategy of quality over quantity.
- Zergs, the insectoid race, units are cheap and weak. They can be produced fast, encouraging players to overwhelm their opponents with sheer numbers.

TODO: explain the gameplay a little more, how are ppl not familiar to SC able to understand the problems for AI that we raise below?

### B. *Challenges in RTS Game AI - Problems*

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: opponent modeling is key if we have to adapt our strategy. As different strategies are dominating each others, forming multiple Nash equilibria, the AI has to be able to infer the intend of its opponent.
- Group and Individual Control ("micro"): the task of controlling units efficiently (we can not speak of optimality here due to the huge state space, and of the enemy behavior entails numerous Nash equilibria), focusing fire to diminish enemy's firepower and keeping our units alive the longest, casting defensive and offensive spells and abilities.
- Planning and Resource Allocation ("macro")
- Spatial reasoning ("tactics")

Maybe the list above goes better in Section VI?

TODO merge above and bottom

**Planning** TODO (Santi?) just a paragraph on exposing why there is a planning problem in RTS AI.

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.

**Spatial and Temporal Reasoning** TODO just a paragraph on exposing why there are temporal and spatial reasoning in RTS AI.

**Learning**
When talking about learning in RTS games, with regard to a particular match against an opponent (where a match is a sequence of games against the same 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 parameters beforehand through simulated annealing or genetic evolution.
- In-game learning: done between the different actions (decisions) taken during a game. For instance, a bot can take into account the successes and failures of its attacks (location, type, timings).
- In-match learning: done between the different games of the match, it encompasses opponent modeling across multiple games and adapting the bots' strategy to its successes and failures.

Learning in all these situations 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.

**Uncertainty**
In RTS games, there are two main kinds of uncertainty, existing for two different reasons. First, there is 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 (to infer what is possible given what has been seen). Secondly, there is the fact that we will never be able to read the opponent's mind, the "intentional" uncertainty that we cannot 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 for extensional uncertainty can $P(Hidden|Seen)$ be *directly* linked to data. 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.

## III. Existing work on RTS Game AI - Resolutions

### A. RTS AI Decomposition

As work on RTS AI has to address several of the aforementioned problems together, in this section, we divide the existing

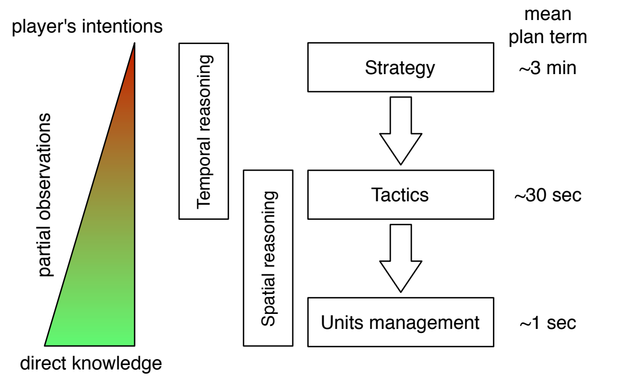



Fig. 1. RTS AI levels of abstraction and their properties: uncertainty (from intentions and from partial observation) is going higher as the abstraction levels are raising. The timings on the right correspond to an estimate of the duration of a behavior switch in StarCraft. Spatial and temporal reasoning are indicated for part for which greedy solutions are really not efficient.

work on RTS AI in three different levels of abstraction, widely known by RTS players: strategy, tactics and units control (micro-management), see Figure 1. We consider strategy to be everything related to the *technology tree*, buildings, upgrades, and army composition. It is the most deliberative level, as a player performs a strategy with future stances (aggressive, defensive, economy, technology) and tactics in mind. To change its strategy, we consider that a player need about 3 minutes: they should need to construct a few new buildings and/or research some upgrades and/or produce units changing the army composition. Tactics represent the attacks and defenses between groups of units. For the human player, decisions on tactics are made from spatial (exploiting the terrain) and temporal (army movements) reasoning, constrained on the possible types of attacks by the army composition of the player and their opponent. Unit control, also called micro-management, is how the player takes action to maximize the efficiency of its individual units (applying the tactics) in real-time (simultaneous) and adversary conditions. As shown in Figure 1, strategy constrains possible tactics which in turn constrain units control. On a prediction/adaptation point of view, the units observation level gives information about the opponent's tactics and strategy. Flaws in opponents tactics can also steer our strategy into exploiting them.
perhaps TODO examples?

### B. Strategy

In commercial RTS games, strategies are often "hard coded" with finite state machines (FSM) [2]. The idea behind FSMs is to decompose an object's behavior into easily manageable states, such as "attacking", "gathering resources" or "repairing" and establish the conditions that trigger transitions between them. Hierarchical FSM give higher (strategic) states some control over lower ones (actions). However, strategic decisions have to be made in a context, not in a vacuum. It is hard to encode dynamic, adaptive behaviors in (H)FSM due to the exponential number of transitions (between states) to account for. To this end, several approaches have been developed like hierarchical task networks, behavior trees and the comeback of STRIPS [3] planning in the industry [4]. Regardless, one cannot but notice that adaptability is not the strong point of industrial RTS AI. In the context of RTS AI strategy adaptation is related to the idea of opponent modeling. An adapting AI needs to keep track of what the enemy is doing in order to estimate the probability that they will use a specific strategy and adapt its own strategy accordingly.

In academic RTS AI research, the main methods used for strategic decision-making have been case-based reasoning (CBR) and planning or plan recognition [5]–[9]. Aha et al. [5] used CBR to perform dynamic plan retrieval extracted from domain knowledge in Wargus (Warcraft II clone). Ontañón et al. [6] base real-time case-based planning (CBP) system on a plan dependency graph which is learned from human demonstration. In [7], [10], they use CBR learned from expert demonstrations on Wargus. They update their previous CBP approach by using situation assessment for retrieving the "right" plan, and improve the speed of CBP by using a decision tree to select relevant features. Hsieh and Sun [11] based their work on [5] CBR model and used StarCraft replays to construct states and building sequences ("build orders"). Schadd et al. [12] describe opponent modeling through hierarchically structured models of the opponent behavior and they applied their work to the Spring RTS (Total Annihilation clone). Hoang et al. [8] use hierarchical task networks (HTN) to model strategies in a first person shooter with the goal to use HTN planners.

Weber and Mateas [13] proposed "a data mining approach to strategy prediction" and performed supervised learning (from buildings features) on labeled StarCraft replays. Dereszynski et al. [14] used HMM to learn the transition probabilities of sequences of constructions and kept the most probables to produce probabilistic behavior models (in StarCraft). Synnaeve and Bessière [15] extended [13] and used a Bayesian semi-supervised (labeling based on EM clustering considering appropriate features) model to learn from replays and predict openings (early game strategies) from StarCraft replays. Then [16], they presented an unsupervised learning Bayesian model for tech-tree prediction, still using replays.

One final consideration is that RTS games are typically partially observable. For example games like StarCraft implement the "fog of war" idea, which basically means that a player can only see the areas of the map close to her own units. Areas of the map away from the field of view of individual units are not observable. Therefore, in order to play an RTS game, players need to scout the opponent in order to obtain information about the opponent's strategy. Most commercial games avoid this problem by using perfect information all the time (cheating). In order to make the human player believe the built-in AI of this games does scouting, sometimes they simulate some scouting tasks as Bob Fitch described in his AIIDE 2011 keynote for the WarCraft game series and StarCraft: Broodwar.

## C. Tactics

Units have different abilities, which leads to different possible tactics. For example, very often, fast, invisible or flying units are not cost-effective in head-to-head fights against slower bulky units. Taking good tactical decisions is taking advantage of the terrain and positions of the different units (in a timely manner). Terrain analysis supplies the AI with structured information about the map in order to help making decisions. This analysis is usually performed off-line, in order to save CPU time during the game. Pottinger [17] described the *BANG* engine implemented by Ensemble Studios for the game Age of Empires II. This engine provides terrain analysis functionalities to the game using influence maps and areas with connectivity information. Forbus et al. [18] showed the importance to have qualitative spatial information for wargames, for which they used geometric and pathfinding analysis. Hale et al. [19] presented a 2D geometric navigation mesh generation method from expanding convex regions from seeds. Finally, Perkins [20] applied Voronoi decomposition (then pruning) to detect regions and relevant choke points in RTS maps. This approach is implemented for StarCraft in BWTA[1].

Hladky and Bulitko [21] benchmarked hidden semi-Markov models (HSMM) and particle filters in first person shooter games (FPS) units tracking. They showed that the accuracy of occupancy maps was improved using movement models (learned from the player behavior) in HSMM. Hagelbäck and Johansson [22] presented a multi-agent potential fields based bot able to deal with fog of war in Tankbattle. Weber et al. [23] used a particle model with a simple linear trajectory update to track opponent units under fog of war in StarCraft. They also produced tactical goals through reactive planning and goal-driven autonomy [24], [25], finding the more relevant goal(s) to spawn in unforeseen situations. Kabanza et al. [26] improve the probabilistic hostile agent task tracker (PHATT [27], a simulated HMM for plan recognition) by encoding strategies as HTN, used for plan and intent recognition to find tactical opportunities. Sharma et al. [28] combined CBR and reinforcement learning to enable reuse of tactical plan components. Cadena and Garrido [29] used fuzzy CBR (fuzzy case matching) for strategic and tactical planning. [30] adapted Monte-Carlo tree search (MCTS) to planning in RTS games and applied it to a capture-the-flag mod of Open RTS. Balla and Fern [31] applied upper confidence bounds on trees (UCT: a MCTS algorithm) to tactical assault planning in Wargus. Finally, Avery et al. [33] and Smith et al. [34] co-evolved influence map trees for spatial (tactical) reasoning in RTS games.

## D. Units Control

Units control, also called micro-management in RTS games, aims at maximizing the effectiveness of units, it starts with pathfinding and goes up to simultaneous control of units of different types in complex battles on heterogeneous terrain. The most common pathfinding algorithm is A*, but its big problem is CPU time and memory consumption, hard to satisfy in a complex, dynamic, real-time environment on multiple units. A common technique consists in reducing the search into a navigation graph map. Among others, Triangulation Reduction A* computes polygonal triangulations on a grid-based map [35]. One of the problems of A* related approaches in

[1]http://code.google.com/p/bwta/

RTS games is that moving units block/get in the way of/ each others and the path has to be recomputed often. Craig Reynolds [36] introduced algorithmic steering behaviors, which simulate the aggregate motion of swarms. His model describes how a system of multiple agents, each steering according to simple local rules, can produce the collective movement found in flocks of birds. Danielsiek et al. [37] studied intelligent squad movements to do tactical moves to flank the opponent in a RTS game this way. In recent commercial RTS games like Starcraft 2 or Supreme Commander 2, flocking-like behaviors are inspired of continuum crowds ("flow field") [39], and a comprehensive review about (grid-based) pathfinding was recently done by Sturtevant [40].

Another popular technique is potential fields (PF) and influence maps. Basically, PF work like a charged particle moving through a magnetic field. The parameters of PF charges can be learned through Q-Learning [41]. Another common issue of potential fields is the local optima problem that can stuck the unit away from the goal. Some uses of potential fields in RTS games are: avoiding obstacles (navigation), avoiding opponent fire, or staying at maximum shooting distance [42], [43]. Self-organizing-maps (SOM) learned with evolutionary algorithms provide some way to learn (realistic) parameters for RTS units behavior [32]. Bayesian modeling has been applied to inverse fusion of the sensory inputs of the units [44], which subsumes potential fields, allowing for integration of tactical goals directly in micro-management. There is also a cognitive approach to RTS AI [45], using a human-like attention model (with units grouping and vision of a unique screen location) for micro-management.

There are some interesting uses of reinforcement learning (RL) [46]: Marthi et al. [47] employ concurrent hierarchical Q-learning (units Q-functions are combined at the group level) RL to efficiently control units in a "one robot with multiple effectors" fashion. Madeira et al. [48] advocate the use of prior domain knowledge to allow faster RL learning and applied their work on a turn-based strategy game. In real game setups, the actions spaces to explore is gigantic, it requires to use the structure of the game in a partial program (or a partial Markov decision process) and a shape function (or a heuristic) [47]. There are other way to learn parameters of an underlying model, for instance Ponsen and Spronck [49] used evolutionary learning techniques, but face the same problem of dimensionality. The difficulty to work with multi-scale goals and plans is handled directly by case-based reasoning (CBR), which has been adapted for units behavior with continuous action models [50], an integrated RL/CBR algorithm using continuous models, or with hybrid CBR/RL transfer learning [28]. Reactive planning [24], a decompositional planning similar to hierarchical task networks [8], allows for plans to be changed at different granularity levels and so for multi-scale (hierarchical) goals integration of low-level control. Synnaeve and Bessière [44] achieve hierarchical goals (coming from tactical decisions) integration through the addition of another sensory input corresponding to the goal's objective.

## IV. State of the Art Bots for Starcraft - Information/Productions

Thanks to the recent organization of international game AI competitions focused around the popular StarCraft game (see Section V), several groups have been working on integrating many of the techniques described in the previous section into complete "bots", capable of playing complete StarCraft games. In this section we will overview some of the currently available top bots, and focus, first on the architecture being used to integrate all the techniques being used for the different aspects of the bot, and then on which subsets of techniques are used for each bot.

### A. RTS Bot Architectures

Playing an RTS game involves dealing with all the problems described above. A few approaches, like CAT [5], Darmok [51] or ALisp [47] 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).

Figure 2 shows some representative examples of the integration architectures used by different bots in the AIIDE and CIG 2011 Starcraft AI competitions [**?**], [52]: BroodwarQ [**?**], Nova [**?**], UAlbertaBot [**?**], Skynet [**?**], SPAR [**?**], and BTHAI [**?**] (see Section V for a comparison of their performance). 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 BroodwarBotQ (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 mostly through writing data in a shared blackboard, and in BBQ they coordinate only when

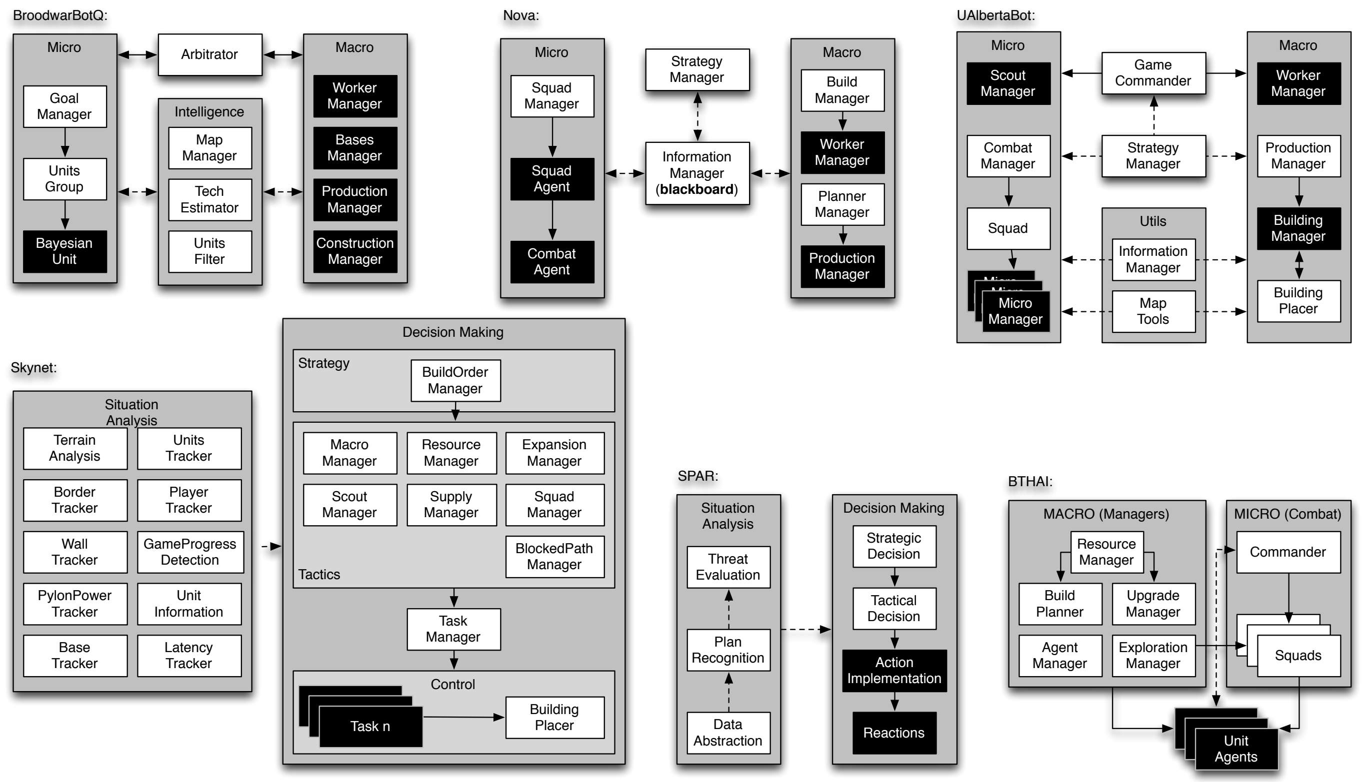



Fig. 2. Architecture of 6 Starcraft bots obtained by analyzing their source code. Modules with black background sent commands directly to Starcraft, dashed arrows represent data flow, and solid arrows represent control.

they have to use a shared resource (unit) by means of an arbitrator.

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 2, we can see the following use of abstraction and divide-in-conquer in the bots:

- BroodwarBotQ: 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 BroodwarBotQ, 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).
- UAlbertaBot: also uses abstraction in micro-management like the previous two bots. But it also uses it in macro-management: as can be seen, the production manager sends commands to the building manager, who is in charge of producing the buildings. This bot also uses divide-and-conquer, and tasks like scouting and resource gathering are managed by separate, independent 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.
- 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.
- 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.

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*.

Most bots using divide-and-conquer (except for BBQ and UAlbertaBot), assume that each of the modules can act independently and that their actions can be executed without interference. BBQ and UAlbertaBot use an arbitrator (*Game Commander'* in UAlbertaBot) 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. Nova and Skynet are exceptions, and 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 rigid and inflexible, but responds to the much higher complexity of the Starcraft game. An interesting exception to that is UAlbertaBot, which uses a search algorithm in the *Production Manager* to find near-optimal build orders.

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.

This section has focused on discussing the architecture of existing Starcraft bots. Let us now focus on what is the state of the art on the techniques used for each of the individual modules used by them.

### B. Individual AI Techniques

TODO

## V. Recent Starcraft AI Competitions

### A. CIG 2011

An initial attempt to run a StarCraft tournament at the Computational Intelligence in Games conference 2010 suffered from technical problems. These mainly stemmed from the desire to use evolved, largely untested maps which proved to look interesting but made the submitted bots crash so often that it would have been unjustifiable to announce a winner.

At the CIG 2011, the tournament was therefore run with a (secret) selection of maps used in league play. It was organized by Tobias Mahlmann and Mike Preuss and attracted 10 bots.

Bracket B

Crashes Games Bot Wins 12 40 Xelnaga 25 3 40 BroodwarBotQ 23 0 40 BTHAI 23 17 40 Protoss Beast Jelly 17 0 40 EvoBot 12 As BroodwarBotQ and BTHAI have the same number of wins, their direct encounter is evaluated which accounted 6 : 4 for the BroodwarBotQ.

Qualified for the final round: Xelnaga and BroodwarBotQ. Note that all qualified bots play Protoss. Maps used for the first round: (2)MatchPoint 1.3, (4)Fighting Spirit 1.3, iCCupdestination 1.1, iCCup gaia, iCCup great barrier reef

Final Round

The final round was played in a similar mode as each of the first round brackets, using another set of 5 maps (see below), resulting in 30 games for each bot.

Crashes Games Bot Wins 0 30 Skynet 26 0 30 UAlbertaBot 22 3 30 Xelnaga 11 2 30 BroodwarBotQ 1

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 ...

TABLE I
BRACKET A RESULTS AT CIG 2011, QUALIFIED FOR THE FINAL ROUND: UALBERTABOT AND SKYNET.

| Crashes | games | bot | wins |
|---|---|---|---|
| 0 | 40 | UAlbertaBot | 33 |
| 1 | 40 | Skynet | 31 |
| 2 | 40 | AIUR | 24 |
| 1 | 40 | Nova | 8 |
| 0 | 40 | LSAI | 4 |

## 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] R. Houlette and D. Fu, "The ultimate guide to fsms in games," *AI Game Programming Wisdom 2*, 2003.

[3] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3–4, pp. 189 – 208, 1971. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0004370271900105

[4] J. Orkin, "Three states and a plan: The a.i. of f.e.a.r." in *GDC*, 2006.

[5] 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.

[6] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Learning from demonstration and case-based planning for real-time strategy games," in *Soft Computing Applications in Industry*, ser. Studies in Fuzziness and Soft Computing, B. Prasad, Ed. Springer Berlin / Heidelberg, 2008, vol. 226, pp. 293–310.

[7] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *Proceedings of the 7th International conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, ser. ICCBR '07. Springer-Verlag, 2007, pp. 164–178.

[8] H. Hoang, S. Lee-Urban, and H. Muñoz-Avila, "Hierarchical plan representations for encoding strategic game ai," in *AIIDE*, 2005, pp. 63–68.

[9] M. Ramírez and H. Geffner, "Plan recognition as planning," in *Proceedings of the 21st international joint conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., 2009, pp. 1778–1783.

[10] K. Mishra, S. Ontañón, and A. Ram, "Situation assessment for plan retrieval in real-time strategy games," in *ECCBR*, 2008, pp. 355–369.

[11] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *IJCNN*, 2008, pp. 3106–3111.

[12] F. Schadd, S. Bakkes, and P. Spronck, "Opponent modeling in real-time strategy games," in *GAMEON*, 2007, pp. 61–70.

[13] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *IEEE Symposium on Computational Intelligence and Games (CIG)*, 2009.

[14] E. Dereszynski, J. Hostetler, A. Fern, T. D. T.-T. Hoang, and M. Udarbe, "Learning probabilistic behavior models in real-time strategy games," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, AAAI, Ed., 2011.

[15] G. Synnaeve and P. Bessiere, "A Bayesian Model for Opening Prediction in RTS Games with Application to StarCraft," in *Proceedings of 2011 IEEE CIG*, Seoul, Corée, République De, Sep. 2011, p. 000.

[16] G. Synnaeve and P. Bessière, "A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft," in *Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2011)*, ser. Proceedings of AIIDE, AAAI, Ed., Palo Alto, États-Unis, Oct. 2011, pp. 79–84.

[17] D. C. Pottinger, "Terrain analysis for real-time strategy games," in *Proceedings of Game Developers Conference 2000*, 2000.

[18] K. D. Forbus, J. V. Mahoney, and K. Dill, "How qualitative spatial reasoning can improve strategy game ais," *IEEE Intelligent Systems*, vol. 17, pp. 25–30, July 2002. [Online]. Available: http://dx.doi.org/10.1109/MIS.2002.1024748

[19] D. H. Hale, G. M. Youngblood, and P. N. Dixit, "Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds," *Artificial Intelligence and Interactive Digital Entertainment AIIDE*, pp. 173–178, 2008. [Online]. Available: http://www.aaai.org/Papers/AIIDE/2008/AIIDE08-029.pdf

[20] L. Perkins, "Terrain analysis in real-time strategy games : An integrated approach to choke point detection and region decomposition," *Artificial Intelligence*, pp. 168–173, 2010.

[21] S. Hladky and V. Bulitko, "An evaluation of models for predicting opponent positions in first-person shooter video games," in *CIG (IEEE)*, 2008.

[22] J. Hagelbäck and S. J. Johansson, "Dealing with fog of war in a real time strategy game environment," in *CIG (IEEE)*, 2008, pp. 55–62.

[23] B. G. Weber, M. Mateas, and A. Jhala, "A particle model for state estimation in real-time strategy games," in *Proceedings of AIIDE*, AAAI Press. Stanford, Palo Alto, California: AAAI Press, 2011, p. 103–108.

[24] 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.

[25] B. G. Weber, M. Mateas, and A. Jhala, "Applying goal-driven autonomy to starcraft," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2010.

[26] F. Kabanza, P. Bellefeuille, F. Bisson, A. R. Benaskeur, and H. Irandoust, "Opponent behaviour recognition for real-time strategy games," in *AAAI Workshops*, 2010.

[27] C. W. Geib and R. P. Goldman, "A probabilistic plan recognition algorithm based on plan tree grammars," *Artificial Intelligence*, vol. 173, pp. 1101–1132, July 2009.

[28] M. Sharma, M. Holmes, J. Santamaria, A. Irani, C. L. Isbell, and A. Ram, "Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL," in *International Joint Conference of Artificial Intelligence, IJCAI*, 2007.

[29] P. Cadena and L. Garrido, "Fuzzy Case-Based Reasoning for Managing Strategic and Tactical Reasoning in StarCraft," in *MICAI (1)*, ser. Lecture Notes in Computer Science, I. Z. Batyrshin and G. Sidorov, Eds., vol. 7094. Springer, 2011, pp. 113–124.

[30] M. Chung, M. Buro, and J. Schaeffer, "Monte carlo planning in rts games," in *IEEE Symposium on Computational Intelligence and Games (CIG)*, 2005.

[31] R.-K. Balla and A. Fern, "Uct for tactical assault planning in real-time strategy games," in *International Joint Conference of Artificial Intelligence, IJCAI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 40–45.

[32] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Piatkowski, R. Ster, A. Thom, and S. Wessing, "Towards intelligent team composition and maneuvering in real-time strategy games," *Transactions on Computational Intelligence and AI in Games (CIG)*, vol. 2, no. 2, pp. 82–98, June 2010.

[33] P. Avery, S. Louis, and B. Avery, "Evolving Coordinated Spatial Tactics for Autonomous Entities using Influence Maps," in *Proceedings of the 5th international conference on Computational Intelligence and Games*, ser. CIG'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 341–348. [Online]. Available: http://dl.acm.org/citation.cfm?id=1719293.1719350

[34] G. Smith, P. Avery, R. Houmanfar, and S. Louis, "Using co-evolved rts opponents to teach spatial tactics," in *CIG (IEEE)*, 2010.

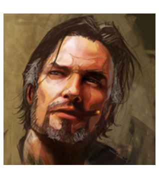
[35] D. Demyen and M. Buro, "Efficient triangulation-based pathfinding," *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, pp. 942–947, 2006.

[36] C. W. Reynolds, "Steering behaviors for autonomous characters," *Proceedings of Game Developers Conference 1999*, pp. 763–782, 1999.

[37] H. Danielsiek, R. Stuer, A. Thom, N. Beume, B. Naujoks, and M. Preuss, "Intelligent moving of groups in real-time strategy games," *2008 IEEE Symposium On Computational Intelligence and Games*, pp. 71–78, 2008.

[38] J. Van Den Berg and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," *2008 IEEE International Conference on Robotics and Automation*, pp. 1928–1935, 2008.

[39] A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 1160–1168, 2006.
[40] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, 2012. [Online]. Available: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf
[41] L. Liu and L. Li, "Regional cooperative multi-agent q-learning based on potential field," pp. 535–539, 2008.
[42] J. Hagelbäck and S. J. Johansson, "Using multi-agent potential fields in real-time strategy games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, ser. AAMAS '08. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 631–638.
[43] ——, "A multiagent potential field-based bot for real-time strategy games," *Int. J. Comput. Games Technol.*, vol. 2009, pp. 4:1–4:10, January 2009.
[44] G. Synnaeve and P. Bessiere, "A Bayesian Model for RTS Units Control applied to StarCraft," in *Proceedings of IEEE CIG 2011*, Seoul, Corée, République De, Sep. 2011, p. 000.
[45] S. Wintermute, J. Z. Joseph Xu, and J. E. Laird, "Sorts: A human-level approach to real-time strategy AI," in *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 2007, pp. 55–60.
[46] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.
[47] 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.
[48] C. Madeira, V. Corruble, and G. Ramalho, "Designing a reinforcement learning-based adaptive AI for large-scale strategy games," in *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 2006.
[49] M. Ponsen and I. P. H. M. Spronck, "Improving adaptive game AI with evolutionary learning," in *University of Wolverhampton*, 2004, pp. 389–396.
[50] M. Molineaux, D. W. Aha, and P. Moore, "Learning continuous action models in a real-time strategy strategy environment," in *FLAIRS Conference*, 2008, pp. 257–262.
[51] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "On-line case-based planning," *Computational Intelligence*, vol. 26, no. 1, pp. 84–119, 2010.
[52] "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.