A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft

Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, Dave Churchill, Mike Preuss

***Abstract*—This paper presents an overview of the existing work on AI for real-time strategy (RTS) games. Specifically, we focus on the work around the game *StarCraft*, which has emerged in the past few years as the unified test-bed for this research. We describe the specific AI challenges posed by RTS games, and overview the solutions that have been explored to address them. Additionally, we also present a summary of the results of the recent StarCraft AI competitions, describing the architectures used by the participants. Finally, we conclude with a discussion emphasizing which problems in the context of RTS game AI have been solved, and which remain open.**

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

I. Introduction

THE field of real-time strategy (RTS) game AI has advanced significantly since Michael Buto's call for research in this area [1]. Specially, competitions like the "AIIDE StarCraft AI Competition" have motivated the exploration of many AI approaches in the context of RTS AI. We will list and classify these approaches, explain their strengths and weaknesses and conclude on what is left to achieve human-level RTS AI.

Complex dynamic environments, where it is not possible to have perfect nor complete information about the current state or about the dynamics of the environment, pose significant challenges for artificial intelligence. Road traffic, finance, or weather forecasts are examples of such large, complex, real-life dynamic environments. RTS games can be seen as a simplification of one such real-life environment, with simpler dynamics in a finite and smaller world, although still complex enough to study some of the key interesting problems like decision making under uncertainty or real-time adversarial planning. Finding efficient techniques for tackling these problems on RTS games can thus benefit other AI disciplines and application domains, ans also have concrete and direct applications in the ever growing industry of video games.

This paper aims to provide a one-stop guide on what is the state of the art in RTS AI, with a particular emphasis on the work done in StarCraft. It is organized as follows: Section II introduces RTS games, in particular the game StarCraft, and their main AI challenges. Section III reviews the existing work on tackling these challenges in RTS games. Section IV analyzes several current state of the art RTS game playing agents (called *bots*), selected form the participants to annual StarCraft AI competitions. Section V presents results of the recent annual competitions held at the AIIDE and CIG conferences and a StarCraft bot game ladder[1]. Section VI compiles open questions in RTS game AI. Finally, the paper concludes on discussions and perspectives.

Santiago Ontañón is with the Computer Science Department at Drexel University, Philadelphia, PA, USA.

Gabriel Synnaeve is with the Laboratory of Cognitive Science and Psycholinguistics (LSCP) of ENS Ulm in Paris, France.

Alberto Uriarte is with the Computer Science Department at Drexel University, Philadelphia, PA, USA.

Florian Richoux is with the Laboratory of Computer Science of the University of Nantes, France.

Dave Churchill is with the Computing Science Department of the University of Alberta, Edmonton, Canada.

Mike Preuss is with the Department of Computer Science of Technische Universität Dortmund, Germany.

II. Real-Time Strategy Games

Real-time Strategy (RTS) is a sub-genre of strategy games where players need to build an economy (gathering resources and building a base) and military power (training units and researching technologies) in order to defeat their opponents (destroying their army and base). From a theoretical point of view, the main differences between RTS games and traditional board games such as Chess are:

- They are *simultaneous move* games, where more than one player can issue actions at the same time. Additionally, these actions are *durative*, i.e. actions are not instantaneous, but take some amount of time time to complete.
- RTS games are "real-time", which actually means is that each player has a very small amount of time to decide the next move. Compared to Chess, where players have several minutes to decide the next action, in StarCraft, the game executes at 24 iterations per second, which means that players only have 55ms to decide a move, before the game state changes.
- Most RTS games are partially observable: players can only see the part of the map that has been explored. This is referred to as the *fog-of-war*.
- Most RTS games are non-deterministic. Some actions have a chance of success.
- And finally, the complexity of these games, both in terms of state space size and in terms of number of actions available at each decision cycle is very large. For example, the state space of Chess is typically estimated to be around 10^{50}, heads up no-limit texas holdem poker around 10^{80}, and Go around 10^{170}. In comparison, the state space of StarCraft in a typical map is estimated to be many orders of magnitude larger than any of those, as discussed in the next section.

[1]An extended tournament, which can potentially go on indefinitely.

For those reasons, standard techniques used for playing classic board games, such as game tree search, cannot be directly applied to solve RTS games without the definition of some level of abstraction, or some other simplification. Interestingly enough, humans seem to be able to deal with the complexity of RTS games, and are still vastly superior to computers in these type of games [2]. For those reasons, a large spectrum of techniques have been attempted to deal with this domain, as we will describe below. The remainder of this section is devoted to describe StarCraft as a research testbed, and on detailing the open challenges in RTS game AI.

A. StarCraft

StarCraft: Brood War is an immensely 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. One of the most remarkable aspects of StarCraft is that the three races are extremely well balanced:

- Terrans, 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 make 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.

Figure 1 shows a screenshot of StarCraft showing a player playing Terrans. In order to win a StarCraft game, players must first gather resources (minerals and Vespene gas). As resources become available, players need to allocate them for creating more buildings (which reinforce the economy, and allow players to create units or unlock stronger units), research new technologies (in order to use new unit abilities or improve the units) and train attack units. Units must be distributed to accomplish different tasks such as reconnaissance, defense and attack. While performing all of those tasks, players also need to strategically understand the geometry of the map at hand, in order to decide where to place new buildings (concentrate in a single area, or expand to different areas) or where to set defensive outposts. Finally, when offensive units of two players meet, each player must quickly maneuver each of the units in order to fight a battle, which requires quick and reactive control of each of the units.

A typical StarCraft map is defined as a rectangular grid, where the $width \times height$ of the map is measured in the number of 32×32 square of pixels, a.k.a build tiles. Although the resolution at which units move is in squares of 8×8 pixels, a.k.a walk tiles. The typical dimensions for maps range from 64×64 to 256×256 build tiles. Each player can control up to 200 units (not including buildings). Moreover, each different race contains between 30 to 35 different types of units and buildings, each one with a significant number of special abilities. All these factors together make StarCraft a significant challenge, in which humans are still much better than com-

Fig. 1. A screenshot of *StarCraft: Brood War*.

puters. For instance, in the game ladder iCCup[2] where users are ranked by their current point totals (E being the lowest possible rank, and A^+ and $Olympic$ being the second highest and highest ranks, respectively), the best StarCraft AI bots are ranked between D and D^+, where average amateur players are ranked between C^+ and B. For comparison, StarCraft professional players are usually ranked between A^- and A^+.

From a theoretical point of view, the state space of a StarCraft for a given map is enormous. For example, consider a 128x128 map. At any given moment there might be between 50 to 400 units in the map, each of which might have a complex internal state (remaining energy and hit-points, action being executed, etc.). This quickly leads to an immense number of possible states (way beyond the size of smaller games, such as Chess or Go). For example, just considering the location of each unit (with 128x128 possible positions per unit), and 400 units, gives us an initial number of $16384^{400} \approx 10^{1685}$. If we add the other factors playing a role in the game, we obtain even larger numbers.

Another way to measure the complexity of the game is by looking at the branching factor, b, and the depth of the game, d, as proposed in [3]. In Chess, $b \approx 35$ and $d \approx 80$. In more complex games, like Go, $b \approx 30-300$, and $d \approx 150-200$. In StarCraft, humans can issue only one action at a time (this is because of the GUI used by the game, computer players can issue several actions, to different units, at any given time), thus, assuming a player controls between 50 to 200 units, and that each unit can execute between 4 to 20 actions (moving in each of the 4 cardinal directions, attacking units in range, use special abilities, etc.), we estimate the branching factor for human players to be around $b \approx 200-4000$. Good players can execute about 300 actions per minute, and the typical length of a game is about 25 minutes. Thus, $d \approx 7500$.

B. Challenges in RTS Game AI

Early research in AI for RTS games [1] identified the following six challenges:

- Resource management
- Decision making under uncertainty

[2]http://www.iccup.com/StarCraft/

- Spatial and temporal reasoning
- Collaboration (between multiple AIs)
- Opponent modeling and learning
- Adversarial real-time planning

While there has been a significant work in many, others have been untouched (e.g. collaboration). Moreover, recent research in this area has identified several additional research challenges, such as how to exploit the massive amounts of existing domain knowledge (strategies, build-orders, replays, and so on). Below, we describe current challenges in RTS Game AI, grouped in six main different areas.

1) Planning: As mentioned above, the size of the state space in RTS games is much larger than that of traditional board-games such as Chess or Go. Additionally, the number of actions that can be executed at a given instant of time is also much larger. Thus, standard adversarial planning approaches, such as game tree search are not directly applicable. As we elaborate later, planning in RTS games can be seen as having multiple levels of abstraction: at a higher level, players need long-term planning capabilities, in order to develop a strong economy in the game; at a low level, individual units need to be moved in coordination to fight battles taking into account the terrain and the opponent. Techniques that can address these large planning problems by either sampling, or hierarchical decomposition do not yet exist.

2) Learning: Given the difficulties in playing RTS games by directly using adversarial planning techniques, many research groups have turned attention to learning techniques. We can distinguish three types of learning problems in RTS games:

- *Prior learning*: How can we exploit available data, such as existing replays, or information about specific maps for learning appropriate strategies before hand? A significant amount of work has gone in this direction.
- *In-game learning*: How can bots deploy online learning techniques that allow them to improve their game play while playing a game? These techniques might include reinforcement learning techniques, but also opponent modeling. The main problem again is the fact that the state space is too large and the fact that RTS games are partially observable.
- *Inter-game learning*: What can be learned from one game that can be used to increase the chances of victory in the next game? Some work has used simple game-theoretical solutions to select amongst a pool of predefined strategies, but the general problem remains unsolved.

3) Uncertainty: Adversarial planning under uncertainty in domains of the size of RTS games is still an unsolved challenge. In RTS games, there are two main kinds of uncertainty. First, the game is partially observable, and players cannot observe the whole game map (like in Chess), but need to scout in order to see what the opponent is doing. This type of uncertainty can be lowered by good scouting, and knowledge representation (to infer what is possible given what has been seen). Second, there is also uncertainty arising from the fact that the games are adversarial, and a player cannot predict the actions that the opponent(s) will execute. For this type of uncertainty, the AI, as the human player, can only build a sensible model of what the opponent is likely to do.

4) Spatial and Temporal Reasoning: Spatial reasoning is related to each aspect of terrain exploitation. Spatial reasoning is involved in tasks such as building placing, or base expansion. In the former, the player needs to carefully consider building position into its own bases to both protect them by creating a wall against invasions and to avoid bad configurations where large units could be stuck. In base expansion, the player has to choose good available emplacements to build a new base, regarding its own position and opponent's bases. Finally, spatial reasoning is key to tactical reasoning: players need to decide where to place units for battle, favoring, for instance, engagements when the opponent's units are into a bottleneck.

For instance, in StarCraft, it is always an advantage to have its units on high ground whereas enemy is on low ground, since units on low ground have no vision on high ground, and firing from low ground have a chance to hit units on high ground.

Analogously, temporal reasoning is key in tactical or strategic reasoning. For example, timing attacks and retreats to gain advantage. At a higher strategic level, players need to reason about when to perform long-term impact economic actions such as upgrades, building construction, strategy switching, etc. all taking into account that the effects of these actions are not immediate, but longer term.

5) Domain Knowledge Exploitation: In traditional board-games such as Chess, researchers have exploited the large amounts of existing domain knowledge to create good evaluation functions to be used by alpha-beta search algorithms, extensive opening books, or end-game tables. In the case of RTS games, it is still unclear the ways in which the significantly large amount of domain knowledge (in the forms or strategy guides, replays, etc.) can be exploited by bots. Most work in this area has focused on two main directions: on the one hand, researchers are finding ways in which to hard-code existing strategies into bots, so that bots only need to decide which strategies to deploy, instead of having to solve the complete problem of deciding which actions to execute by each individual unit at each time step. One the other hand, large datasets of replays have been created [4], [5], from where strategies, trends or plans have been tried to learn. However, StarCraft games are quite complex, and how to automatically learn from such datasets is still an open problem.

6) Task Decomposition: For all the previous reasons, most existing approaches to play games like StarCraft work by decomposing the problem of playing a RTS game into a collection of smaller problems, to be solved independently. Specifically, a common subdivision is:

- *Strategy*: corresponds to the high-level decision making process. This is the highest level of abstraction for the game comprehension. Finding an efficient strategy or counter-strategy against a given opponent is key in RTS games. It concerns the whole set of units and buildings a player owns.
- *Tactics*: are the implementation of the current strategy. It implies army and building positioning, movements,

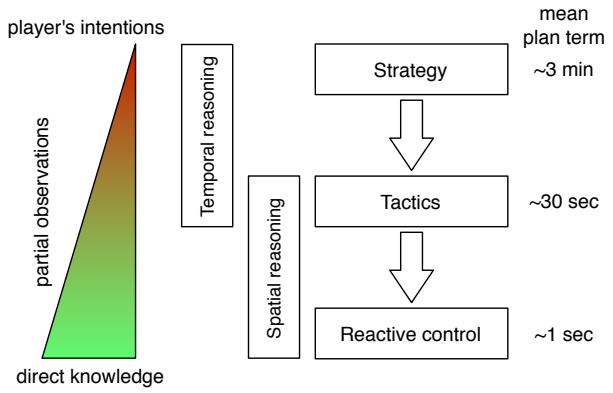

Fig. 2. RTS AI levels of abstraction and their properties: uncertainty (from intentions and from partial observation) is going higher as the abstraction levels are raising. The timings on the right correspond to an estimate of the duration of a behavior switch in StarCraft. Spatial and temporal reasoning are indicated for part for which greedy solutions are really not efficient.

timing, and so on. Tactics concerns a group of units.

- *Reactive control*: is the implementation of tactics. This consists in moving, targeting, firing, fleeing, hit-and-run techniques (a.k.a. "kiting") during battle. Reactive control focuses on a specific unit.
- *Terrain analysis*: consists in the analysis of regions composing the map: choke-points, minerals and gas emplacements, low and high walkable grounds, islands, etc.
- *Intelligence gathering*: corresponds to information collected about the opponent. Because of the fog-of-war, players must regularly send scouts to localize and spy enemy bases.

In comparison, when humans play StarCraft, they typically divide their decision making in a very different way. The StarCraft community typically talk about two tasks:

- *Micro*: is the ability to control your units individually. A good *micro* player usually keeps their units alive during more time.
- *Macro*: is the ability to produce units and to expand at the appropriate times to keep your production of units flowing. A good *macro* player usually has the larger army.

The reader can find a good presentation of task decomposition for AIs playing RTS in [6]. Although the previous task decomposition is common, a significant challenge is on designing architectures so that the individual AI techniques that address each of those tasks can communicate and effectively work together, resolving conflicts, prioritizing resources between them, etc. Section IV provides an overview of the task decompositions that state-of-the-art bots use.

III. Existing work on RTS Game AI

Systems that play RTS games need to address most, if not all, the aforementioned problems together. Therefore, it is hard to classify existing work on RTS AI as addressing the different problems above. For that reason, in order to classify existing work on RTS AI, we will divide it according to three levels of abstraction: strategy (which loosely corresponds to "macro"), tactics and reactive control (which loosely corresponds to "micro").

Figure 2 graphically illustrates how strategy, tactics and reactive control are three points in a continuum scale where strategy corresponds to decisions making processes that affect long spans of time (several minutes in the case of StarCraft), reactive control corresponds to low-level second-by-second decisions, and tactics sit in the middle. Also, strategic decisions reason about the whole game at once, whereas tactical or reactive control decisions are localized, and affect only specific groups of units. Typically, strategic decisions constraint future tactical decisions, which in turn condition reactive control. Moreover, information gathered while performing reactive control, can cause reconsideration of the tactics being employed; which could trigger further strategical reasoning.

Following this idea, we consider strategy to be everything related to the technology trees, build-order[3], upgrades, and army composition. It is the most deliberative level, as a player selects and performs a strategy with future stances (aggressive, defensive, economy, technology) and tactics in mind. We consider tactics to be everything related to confrontations between groups of units. Tactical reasoning involves both 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. Finally, we will call reactive control to how the player controls individual units to maximize their efficiency (when executing tactics) in real-time. The main difference between tactics and reactive control is that tactical reasoning typically involves some sort of planning ahead for some short spans of time, whereas reactive control involves no planning ahead whatsoever.

For example, after starting a game, a player might decide to use a *rushing* strategy (which involves quickly building an army and sending it to attack as early as possible in the game); then, when performing the attack use a *surrounding* tactic, where the player tries to surround the enemy cutting potential escape routes; finally, while executing the surrounding tactic, the player might decide to use reactive control techniques that command individual units to perform repeated *attack and flee* movements, to maximize the efficiency of each of the units being used in the attack.

A. Strategy

Strategic decision making in real-time domains is still an open problem. In the context of RTS games is has been addressed using many AI techniques, like hard-coded approaches, planning-based approaches, or machine learning-based approaches. We cover each of these approaches in turn.

Hard-coded approaches have been extensively used in commercial RTS games. The most common approaches use finite state machines (FSM) [7] in order to let the AI author hard-code the strategy that the AI will employ. The idea behind FSMs is to decompose the AI behavior into easily manageable states, such as "attacking", "gathering resources" or "repairing" and establish the conditions that trigger transitions between them. Commercial approaches also include Hierarchical

[3]The *build-order* is the specific sequence in which buildings of different types will be constructed at the beginning of a game, and completely determines the long-term strategy of a player.

FSMs, in which FSMs are composed hierarchically. These hard-coded approaches have achieved a significant amount of success, and, as we will discuss later, have also been used in many academic RTS AI research systems, as discussed in Section IV. However, these hard-coded approaches struggle to encode dynamic, adaptive behaviors, and are easily exploitable by adaptive opponents.

Approaches using planning techniques have also been explored in the literature. For example Ontañón et al. [8] explored the use of real-time case-based planning (CBP) in the domain of Wargus (a Warcraft II clone). In their work, they used human demonstration to learn plans, that are then composed at run-time in order to form full-fledges strategies to play the game. In [9] they improve over their previous CBP approach by using situation assessment for improving the quality and speed of plan retrieval. Hierarchical Task-Network (HTN) planning has also been explored with some success in the context of simpler first-person shooter games [10]. Planning approaches offer more adaptivity of the AI strategy compared to hard-coded approaches. However, the real-time constraints of RTS games limit the planning approaches that can be applied, being HTN and case-based planning the only ones explored so far. Moreover, none of these approaches addresses any timing or scheduling issues, which are key in RTS games. On notable exception is the work of Churchill and Buro [11], who used planning in order to construct its economic build-orders taking into account timing constraints of the different actions.

Concerning machine learning-based approaches, Weber and Mateas [4] proposed a data mining approach to strategy prediction and performed supervised learning on labeled StarCraft replays. Dereszynski et al. [12] used Hidden Markov Models (HMM) to learn the transition probabilities of sequences of building construction orders and kept the most probable ones to produce probabilistic behavior models (in StarCraft). Synnaeve and Bessière [13] used the dataset of [4] and presented a Bayesian semi-supervised model to learn from replays and predict openings (early game strategies) from StarCraft replays. The openings are labeled by EM clustering considering appropriate features. Then, in [14], they presented an unsupervised learning Bayesian model for tech-tree prediction, still using replays. Finally, evolutionary approaches to determine priorities of high level tasks was explored by Young and Hawes in their QUORUM system [15], showing improvement over static priorities.

Also falling into the machine-learning category, a significant group of researchers has explored case-based reasoning (CBR) [16] approaches for strategic decision making. For example Aha et al. [17] used CBR to perform dynamic plan retrieval in the Wargus domain. Hsieh and Sun [18] based their work on Aha et al.'s CBR model [17] and used StarCraft replays to construct states and building sequences ("build orders"). Finally, Schadd et al. [19] applied a CBR approach to opponent modeling through hierarchically structured models of the opponent behavior and they applied their work to the Spring RTS game (a "Total Annihilation" clone).

One final consideration concerning strategy is that RTS games are typically partially observable. 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. Players need to scout in order to obtain information about the opponent's strategy. The size of the state space in StarCraft prevents solutions based on POMDPs from being directly applicable, and very few of the previous approaches deal with this problem. Most research in RTS game AI assumes perfect information all the time. In the case of commercial games, most AI implementations cheat, since they have perfect information, while the human player does not. In order to make the human player believe the AI of this games does not cheat, sometimes they simulate some scouting tasks as Bob Fitch described in his AIIDE 2011 keynote for the WarCraft and StarCraft game series. A notable exception is the work of Weber et al. [20], who used a particle model with a 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 [21], [22], finding the more relevant goal(s) to spawn in unforeseen situations.

B. Tactics

Tactical reasoning involves reasoning about the different abilities of the units in a group and about the environment (terrain) and positions of the different groups of units in order to gain military advantage in battles. For example, it would be a very bad tactical decision to send fast, invisible or flying units (typically expensive) in the first line of fire against slower heavier units, since they will be wiped out fast. We will divide the work on tactical reasoning in two parts: terrain analysis and decision making.

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. For example, Pottinger [23] 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. [24] showed the importance to have qualitative spatial information for wargames, for which they used geometric and pathfinding analysis. Hale et al. [25] presented a 2D geometric navigation mesh generation method from expanding convex regions from seeds. Finally, Perkins [26] applied Voronoi decomposition (then pruning) to detect regions and relevant choke points in RTS maps. This approach is implemented for StarCraft in the BWTA[4] library, used by most state of the art StarCraft bots.

Concerning tactical decision making, many different approaches have been explored such as machine learning or game tree search. Hladky and Bulitko [27] 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. Kabanza et al. [28] improve the probabilistic hostile agent task

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

tracker (PHATT [29], a simulated HMM for plan recognition) by encoding strategies as HTN, used for plan and intent recognition to find tactical opportunities. Sharma et al. [30] combined CBR and reinforcement learning to enable reuse of tactical plan components. Cadena and Garrido [31] used fuzzy CBR (fuzzy case matching) for strategic and tactical planning. Finally, [32] combined space abstraction into regions from [26] and tactical-decision making by assigning scores (economical, defenses, etc.) to regions and looking for their correspondences to tactical moves (attacks) in pro-gamers replays. Game tree search techniques have also been explored for tactical decision making. [33] applied Monte-Carlo planning to a capture-the-flag version of Open RTS. Balla and Fern [34] applied the UCT algorithm (a Monte Carlo Tree Search algorithm) to tactical assault planning in Wargus. To make game tree search is applicable at this level, abstract game state representations are used in order to reduce the complexity. Also, abstractions, or simplifications about the set of possible actions to execute in a given game state need to be used.

Additionally, scouting is equally important in tactical decision making as in strategic decision making. However, as mentioned earlier, very little work has been done in this respect, being that of Weber et al. [20] the only exception. All previous approaches, including all game tree search ones, assume complete information.

C. Reactive Control

Reactive control aims at maximizing the effectiveness of units, including simultaneous control of units of different types in complex battles on heterogeneous terrain.

Potential fields (or influence maps) have been found to be a useful technique for reactive decision making. Some uses of potential fields in RTS games are: avoiding obstacles (navigation), avoiding opponent fire [35], or staying at maximum shooting distance [36]. Potential fields have also been combined with A* path-finding to avoid local traps [37]. Hagelbäck and Johansson [38] presented a multi-agent potential fields based bot able to deal with fog-of-war in the Tankbattle game. Avery et al. [39] and Smith et al. [40] co-evolved influence map trees for spatial reasoning in RTS games. Danielsiek et al. [41] used influence maps to achieve intelligent squad movement to flank the opponent in a RTS game. Despite their success, a drawback for potential field-based techniques is the large number of parameters that has to be tuned in order to achieve the desired behavior. Approaches for automatically learning such parameters have been explored, for example, using reinforcement learning [42], or self-organizing-maps (SOM) [43]. We would like to note that potential fields are a reactive control technique, and as such, they do not perform any form of lookahead. As a consequence, these techniques are prone to make units stuck in local maxima.

There has been a significant amount of work on using machine learning techniques for the problem of reactive control. 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.

Additionally, there has been some interesting uses of reinforcement learning (RL) [45]: Wender and Watson [46] evaluated the different major RL algorithms for (decentralized) micro-management, which perform all equally. 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. This is because the action space to explore is gigantic for real game setups. It requires exploiting the existing structure of the game in a partial program (or a partial Markov decision process) and a shape function (or a heuristic) [47]. Another approach is that proposed by Jaide and Muñoz-Avila [49] through learning just one Q-function for each unit type, in order to cut down the search space. Other approaches that aim at learning the parameters of an underlying model have also been explored. For example Ponsen and Spronck [50] used evolutionary learning techniques, but face the same problem of dimensionality. For example, evolutionary optimization by simulating fights can easily adapted to any parameter-dependent micro-management control model, as shown by [51] which optimizes an AIIDE 2010 micro-management competition bot.

Finally, approaches based on game tree search are recently being explored for micro management. Churchill et al. [52] presented a variant of alpha-beta search capable of dealing with simultaneous moves and durative actions, which could handle reactive control for situations with up to eight versus eight units.

Other research falling into reactive control has been performed in the field of cognitive science, where Wintermute et al. [53] have explored human-like attention models (with units grouping and vision of a unique screen location) for reactive control.

Finally, although pathfinding does not fall under our previous definition of reactive control, we include it in this section, since it is typically performed as a low-level service, not part of either tactical nor strategical reasoning (although there are some exceptions, like the tactical pathfinding of Danielsiek et al. [41]). 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 with large numbers of units. Even if specialized algorithms, such as D*-Lite [54] exist, it is most common to use A* combined with a map simplification technique that generates a simpler navigation graph to be used for pathfinding. An example of such technique is Triangulation Reduction A*, that computes polygonal triangulations on a grid-based map [55]. Considering movement for groups of units, rather then individual units, techniques such as steering of flocking behaviors [56] can be used on top of a path-finding algorithm in order to make whole groups of units follow a given path. In recent commercial RTS games like StarCraft 2 or Supreme Commander 2, flocking-like behaviors are inspired of continuum crowds ("flow field") [57]. A comprehensive review about (grid-based) pathfinding was recently done by Sturtevant [58].

D. Holistic Approaches

Holistic approaches to address RTS AI attempt to address the whole problem using a single unified method. To the best of our knowledge, with a few exceptions, such as the Darmok system [59] (which uses a combination of case-based reasoning and learning from demonstration) or ALisp [47], there hasn't been much work in this direction. The main reason is that the complexity of RTS games is too large, and approaches that decompose the problem into smaller, separate, problems, achieve better results in practice. However, holistic approaches, based, for example, on Monte Carlo Tree Search, have not been explored so far, and would be a very interesting venue of future work.

A related problem is that of integrating reasoning at multiple levels of abstraction. Molineaux et al. [60] showed that the difficulty of working with multi-scale goals and plans can be handled directly by case-based reasoning (CBR), via an integrated RL/CBR algorithm using continuous models. Reactive planning [21], a decompositional planning similar to hierarchical task networks [10], 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

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.

Playing an RTS game involves dealing with all the problems described above. A few approaches, like CAT [17], Darmok [59] or ALisp [47] try to deal with the problem in a monolithic manner, by using a single AI technique. 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.

Figure 3 shows some representative examples of the architectures used by different bots in the AIIDE and CIG StarCraft AI competitions (see Section V): BroodwarBotQ [44], Nova [35], UAlbertaBot [11], Skynet, SPAR, AIUR, and BTHAI [37]. 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*. Situation analysis in SPAR has three modules dedicated to analyze the current situation of the game. Decision making in SPAR is done through four hierarchically organized modules, 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 two 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 mostly 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.

By analyzing the structure of these bots, we can see that there are two main tools being used in these integration architectures:

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

- BroodwarBotQ[5]: 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[6]: is similar in design as BroodwarBotQ, and

[5]http://github.com/SnippyHolloW/BroodwarBotQ
[6]http://nova.wolfwork.com/

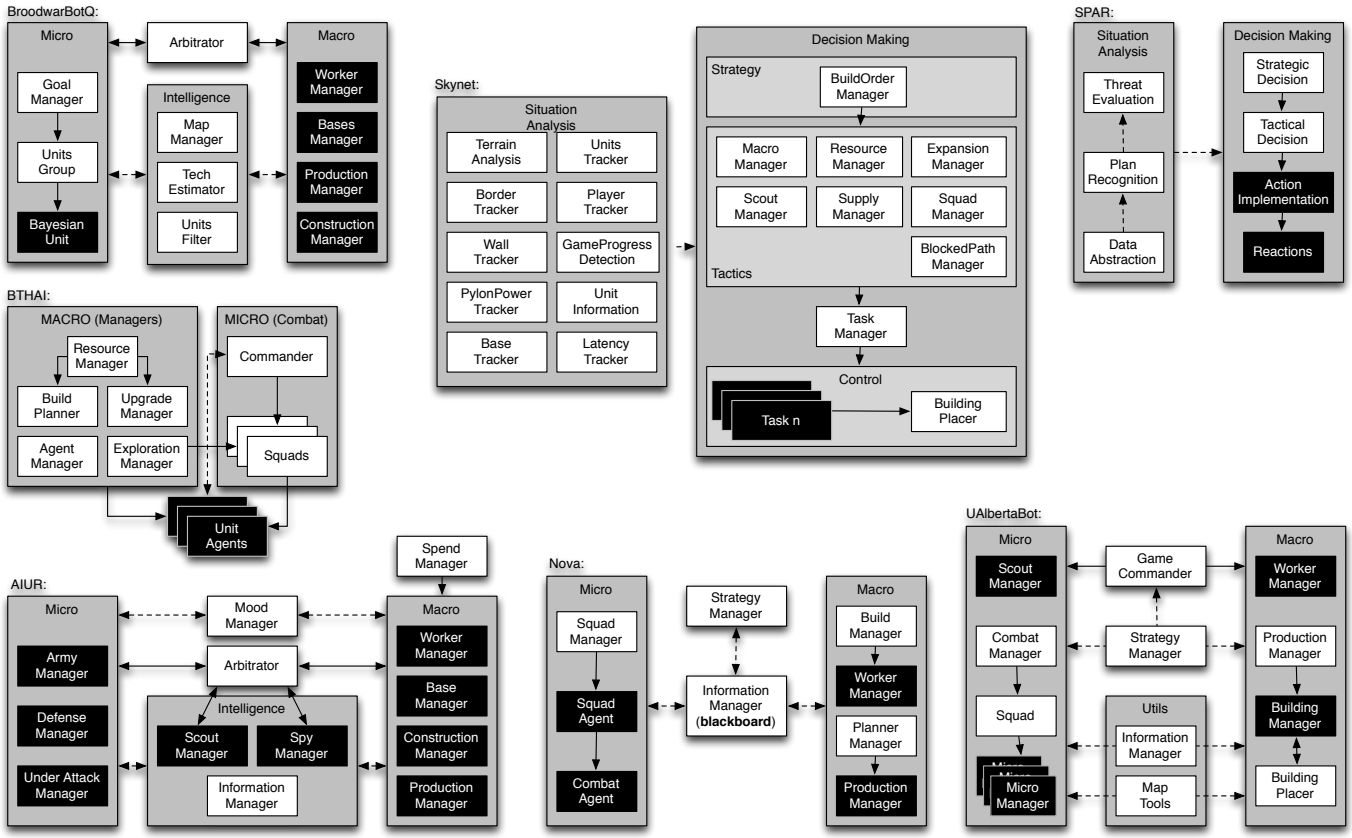

Fig. 3. Architecture of 7 StarCraft bots obtained by analyzing their source code. Modules with black background sent commands directly to StarCraft, dashed arrows represent data flow, and solid arrows represent control.

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[7]: 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[8]: 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[9]: 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.
- AIUR[10]: is mainly divide-and-conquer oriented, for macro as well as for micro, with a slight abstraction on macro due to a SpendManager deciding how to spend and share resources among Base, Production and Construction Managers. At the beginning of a game, the MoodManager initializes a "mood" which will influence both tactics and strategy. Tactics are divided into three independent managers: the *Defense Manager*, controlling military units when there is nothing special, the *Under Attack Manager*, activated when the opponent is attacking our bases, and the *Army Manager*, taking control of units when it is time to attack, following a timing given by the current mood. This bot does not manage however any kind of reactive controls so far.
- BTHAI[11]: 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.

[7]http://code.google.com/p/ualbertabot/
[8]http://code.google.com/p/skynetbot/
[9]http://www.planiart.usherbrooke.ca/projects/spar/
[10]http://code.google.com/p/aiurproject/
[11]http://code.google.com/p/bthai/

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*.

Some bots using divide-and-conquer, assume that each of the modules can act independently and that their actions can be executed without interference. BBQ, UAlbertaBot and AIUR, however 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. Following available resources and timing, AIUR's *Spend Manager* orders Base, Production and Construction Managers what they have to build/produce. It also orders to start tech research and upgrades. The idea here is not to let the different managers allocate the resources they want, but to do the opposite, that is: finding how the AI can spend the available money.

One interesting aspect of the seven bots described above is that, while all of them (except AIUR) are reactive at the lower level (reactive 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. Another interesting case is AIUR, that uses a *Mood Manager* to randomly pick a mood among six (cheese, rush, aggressive, defensive, macro, fast expand), which will influence the build order, strategy and tactics.

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. For space reasons, we do not include an exhaustive comparison of the architectures of all the participating bots. Some other bots have been documented by their authors, such as SCAIL [61] or QUORUM [15]. Let us now focus on their performance.

V. Recent StarCraft AI Competitions

This section reviews the results of the recent international competitions on AI for StarCraft. These competitions, typically co-located with scientific conferences, have been possible thanks to the existence of the the Brood War Application Programming Interface (BWAPI)[12], which enables replacing the human player interface with C++ code. The following subsections summarize the results of all the StarCraft AI competitions held at the AIIDE (Artificial Intelligence for Interactive Digital Entertainment) and CIG (Computational Intelligence in Games) conferences during the past years. Additionally we analyze the statistics from the StarCraft Bot Ladder, where the best bots play against each other continuously over time.

TABLE I
Results of the AIIDE 2010 competition.

Position	bot
1	Overmind
2	Krasi0
3	Chronos

A. AIIDE

Started in 2010, the AIIDE StarCraft AI Competition[13] is the most well known and longest running StarCraft AI Competition in the world. Each year, AI bots are submitted by competitors to do battle within the retail version of StarCraft: BroodWar, with prizes supplied by Blizzard Entertainment.

The first competition in 2010 was organized and run by Ben Weber in the Expressive Intelligence Studio at University of California, Santa Cruz[14]. 26 total submissions were received from around the world. 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 micromanagement 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 the complete game of *StarCraft: BroodWar* with fog-of-war enforced. The tournament was run with a random pairing double-elimination format with each match

[12]http://code.google.com/p/bwapi/
[13]http://www.StarCraftAICompetition.com
[14]http://eis.ucsc.edu/StarCraftAICompetition

being best of 5 games. Competitors could play as any of the three races, with the only limitations in gameplay being those which were considered "cheating" in the StarCraft community. A map pool of 5 well-known professional maps was announced to competitors in advance, with a random map being chosen for each game.

Results are shown in Table I. The team that won was Overmind[15], from University of California, Berkeley. Using the Zerg race, their strategy was to defend early aggression with zergling units while amassing mutalisk units which they used to contain and eventually defeat their opponents. The mutalisk is a very fast and agile flying unit which is able to attack while moving with no drawback, which makes them quite a powerful unit when controlled by a computer. Overmind used a potential-field based micromanagement system to guide their mutalisks, which led them to victory. Krasi0 came in 2nd place with a standard defensive Terran opening strategy which transitioned into mech play in the late game.

In 2011 the competition was hosted by the University of Alberta, with organization by Michael Buro and David Churchill[16]. Due to a lack of entrants in tournament categories 1-3 in the 2010 competition, it was decided that only the full game category would be played in the 2011 competition. Another important change in the 2011 competition was the introduction of automated tournament-managing software running StarCraft games simultaneously on 20 computers, allowing a total of 1170 games to be played in far less time than the 108 games of the 2010 competition. This increase in games played also allowed the tournament to switch to a round-robin format, eliminating the "luck" factor of the pairings inherent in bracket style tournaments. The bot which achieved the highest win percentage over the course of the competition would be determined the winner. Also, the competition became open-source, in an effort not only to prevent possible cheating, but to promote healthy competition in future tournaments by giving newcomers and easier entry point by basing their design off of previous bots.

In the end, Skynet won the competition with its solid Protoss play (results are summarized in Table II). The bot executed one of a small set of strategies randomly at the start of the match based on the map and the race of the opponent. Skynet would then amass a medium to large sized army and expand before moving out to attack. Good use of Dragoon range and kiting micromanagement allowed it to hold off the early aggression of other bots such as UAlbertaBot, which came in 2nd.

UAlbertaBot used an early zealot-rush strategy to take advantage of the power of early game Protoss units. It would send out the first zealots which were made and immediately attack the enemy base, using a unit counting heuristic to determine whether or retreat or keep pushing. Of note is that UAlbertaBot used an online planning algorithm to construct all of its economic build-orders [11], as no hard-coded build orders were used.

AIUR also chose Protoss, with a strategy which was in between Skynet and UAlbertaBot in terms of attack timings.

[15]http://overmind.cs.berkeley.edu

[16]https://skatgame.net/mburo/sc2011/

TABLE II
RESULTS OF THE AIIDE 2011 COMPETITION.

Position	bot	win %
1	Skynet	88.9%
2	UAlbertaBot	79.4%
3	AIUR	70.3%
4	ItayUndermind	65.8%
5	EISBot	60.6%

At that time, AIUR chose one mood among five (leading to slightly different strategies and tactics) at the beginning of a game and kept it until the end. These five moods were:

- Rush: where the bot tries early attacks, and have good probabilities to send the two or three first Zealots to harass the opponent.
- Aggressive: where we have less chance to perform harasses with the first Zealots, and the first attack is usually a bit delayed with regard to the Rush mood.
- Macro: where the AI do not try any early attacks and focus a bit more on its economy before attacking.
- Defense: where the AI "turtles" and wait to have a consequent army before running an attack.
- Fast expand: where the first building constructed it a base expansion, for a very economical-oriented game.

Notice that build orders are not fully hard-coded since they can be altered by AIUR's Spend Manager.

Of note in these results was that a rock-paper-scissors effect happened among the top 3 finishers. Of the 30 rounds, Skynet beat UAlbertaBot 26 times, UAlbertaBot beat AIUR 29 times, and AIUR beat Skynet 19 times. Another notable result is that Overmind did not choose to compete despite winning the 2010 competition. After the competition, many bot programmers (including the Overmind team) realized that their 2010 strategy was quite easily defeated by early game rushing strategies, and so they submitted a Terran bot instead, called Undermind, which finished in 7th.

After the competition was over, a man vs. machine match was held between the winner (Skynet) and an ex-professional StarCraft player named Oriol Vinyals. Oriol was a competitor in the 2001 World Cyber Games StarCraft competition, and though he had been out of practice for a few years was still quite a good player. The match was arranged to see how well StarCraft AI bots had progressed and to see if they could actually beat a decent human opponent.

For the best-of-three match, Oriol chose his races randomly and ended up beating Skynet in a 2-0. In the first match, Oriol played Zerg vs. Skynet's Protoss on Python, a four player map. Oriol chose to start with a fast expansion strategy and transition into two base mutalisk production. Skynet chose to rush with a few early zealots, which was luckily the best possible choice given Oriol's strategy. Skynet's initial attack destroyed Oriol's early defenses, and nearly won the game in the first few minutes, however it then proceeded to send zealots to attack one at a time rather than group up its units before moving in, which allowed Oriol to catch up. Once Oriol produced his mutalisks, Skynet did not produce sufficient air defenses and Oriol quickly destroyed Skynet's base. In

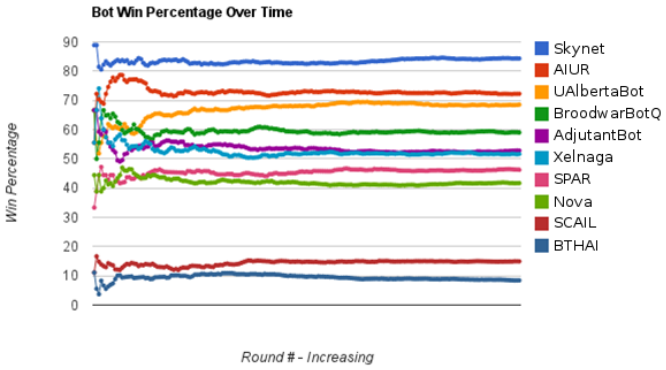

Fig. 4. Evolution of the win percentage of each bot participating in the AIIDE 2012 competition

the second game, Oril played Terran, again on Python. After holding off early Dragoon pressure from Skynet, Oriol moved out with a few marines, medics and tanks. Skynet tried to defend with its army of Dragoons, however due to poor unit targeting decisions it started to attack useless medics after the marines had died, rather than the tanks. Oriol overcame the Dragoon army and was victorious. Later analysis of the match concluded that Skynet, while dominant over the other bots, was unable to properly adapt and transition into a mid-game strategy in game one once its early pressure failed, and in game two made a key blunder in unit targeting which cost it the game. Humans were still in command.

The 2012 competition was also hosted by the University of Alberta, with the major difference from the 2011 competition being the addition of persistent storage. Bots could now write information to disk during a match, and then read the information during other matches, allowing them to adjust strategies based on previous results. 6 of the 10 entrants used this feature to aid in strategy selection, including the top 4 finishers. More improvements to the tournament environment also meant that a total of 4240 games could now be played in the same time period. Results are shown in Table III.

Skynet once again won the competition with its solid Protoss build orders and good Dragoon kiting. AIUR and UAlbertaBot switched positions from the previous year to come 2nd and 3rd respectively. Both AIUR and UAlbertaBot used data stored from the results of previous games to select a strategy for future matches. UAlbertaBot did this using the UCB [62] algorithm, while AIUR used a uniform distribution to choose its mood before altering this distribution after some games against the same opponent to favor efficient strategies, achieving similar results than UAlbertaBot. Notice that, compared to AIIDE 2011, AIUR proposes a new mood, *Cheese*, implementing a Photon Cannon rush strategy in order to surprise the opponent and to finish the game as soon as possible. The effect of this strategy selection process can be seen Figure 4 which shows bot win percentages over time. While the earlier rounds of the tournament fluctuated wildly in results, eventually the results converged to their final values. One of the main reasons for this is due to the bots learning which strategies to use as the tournament progressed.

The 2012 man vs. machine match again used the winner

TABLE III
RESULTS OF THE AIIDE 2012 COMPETITION.

Position	bot	win %
1	Skynet	84.4%
2	AIUR	72.2%
3	UAlbertaBot	68.6%
4	BroodwarBotQ	59.1%
5	AdjutantBot	52.8%

of the competition (Skynet), who played against Mike Lange, also known as Bakuryu. At the time of the match, Bakuryu was an A- ranked Zerg player on ICCup, and known as one of the best non-Korean Zerg players in the world. Bakuryu was considered much stronger than Oriol at the time that the match was played, and the results showed that this was true. In the first game of the best-of-three, Bakuryu made Skynet look quite silly by running around inside Skynet's base with a small number of zerglings while Skynet's zealots and half of its worked chased then in vain. After killing off several probes and buying enough time to set up his expansion, he cleaned up Skynet's army with a flying army of mutalisks. In the second game, Bakuryu contained Skynet inside its base with a group of zerglings positioned within Skynet's expansion. Skynet then constructed several Dark Templar and along with some Dragoons and Zealots attacked into Bakuryu's expansion which was heavily defended, and was crushed almost instantly, allowing Bakuryu's zergling force to finish off the Protoss base.

In this match it was shown that the true weakness of state of the art StarCraft AI systems was that humans are very adept at recognizing scripted behaviors and exploiting them to the fullest. A human player in Skynet's position in the first game would have realized he was being taken advantage of and adapted his strategy accordingly, however the inability to put the local context (Bakuryu kiting his units around his base) into the larger context of the game (that this would delay Skynet until reinforcements arrived) and then the lack of strategy change to fix the situation led to an easy victory for the human. These problems remain as some of the the main challenges in RTS AI today: to both recognize the strategy and intent of an opponent's actions, and how to effectively adapt your own strategy to overcome them.

All results, videos, and replays from the AIIDE StarCraft AI Competition can be found in http://www.StarCraftAICompetition.com.

B. CIG

An initial attempt to run a StarCraft tournament at the Computational Intelligence in Games conference (CIG 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 and the Broodwar Terrain Analyzer (BWTA) provided with the BWAPI interface crash so frequently that it would have been unjustifiable to announce a winner.

At CIG 2011, the tournament was therefore run with a (secret) selection of maps used in league play, which can be

TABLE IV
RESULTS OF THE FIRST ROUND AT CIG 2011, HELD IN TWO BRACKETS. QUALIFIED FOR THE FINAL ROUND: UALBERTABOT AND SKYNET (FROM A), XELNAGA AND BROODWARBOTQ (FROM B, THE LATTER BY COMPARING DIRECT ENCOUNTERS WITH BTHAI OF WHICH 6:4 WERE WON).

Bracket A					Bracket B				
Position	crashes	games	bot	win%	Position	crashes	games	bot	win%
A1	0	40	UAlbertaBot	82.5%	B1	12	40	Xelnaga	62.5%
A2	1	40	Skynet	77.5%	B2	3	40	BroodwarBotQ	57.5%
A3	2	40	AIUR	60.0%	B3	0	40	BTHAI	57.5%
A4	1	40	Nova	20.0%	B4	17	40	Protoss Beast Jelly	42.5%
A5	0	40	LSAI	10.0%	B5	0	40	EvoBot	30.0%

regarded as the most important difference to the AIIDE tournament that employed a known list of maps. The competition was organized by Tobias Mahlmann and Mike Preuss and attracted 10 bots. In addition to the ones discussed in previous sections (UAlbertaBot, Skynet, AIUR, Nova, BroodwarBotQ, BTHAI), the set also contained LSAI, Xelnaga, Protoss Beast Jelly, and EvoBot, these are shortly described in the following:

LSAI (Zerg): utilizes a heavily modified BWSAL[17] to divide management of the units to different modules that communicate via a centralized information module. It works using a simple reactive strategy to try and survive early game attacks and macro up to a larger attack force and maintain map control.

Xelnaga (Protoss): is a modification of the AIUR bot that chooses the Dark Templar Opening in order to destroy the enemy base before defenses against invisible units are available.

Protoss Beast Jelly (Protoss): always goes for a 5-gate Zealot rush, supported by an effective harvesting strategy named power-mining (2 probes are assigned to every mineral patch, thereby needing 18 probes for 100% saturation in a normal map, prior to expanding). Gas is not mined as it is not needed for constructing Zealots.

EvoBot (Terran): employs an evolutionary algorithm for obtaining rational unit combinations and influence map techniques for deciding the strategic locations. Note that this bot was submitted in a very early version, with many of its designed features not yet fully ready.

1) First Round: As the CIG competition games were executed manually due to a lack of available software (the AIIDE program was not yet available at that time), the organizers separated the ten entries into two brackets. In each bracket of 5 bots, a round-robin tournament was held with 10 repetitions per pairing, resulting in 40 games per bot. The 5 maps chosen for the first round were selected from the pool of well-known

[17]https://code.google.com/p/bwsal/

TABLE V
RESULTS OF THE CIG 2011 FINAL ROUND.

Position	crashes	games	bot	win%
1	0	30	Skynet	86.7%
2	0	30	UAlbertaBot	73.3%
3	3	30	Xelnaga	36.7%
4	2	30	BroodwarBotQ	3.3%

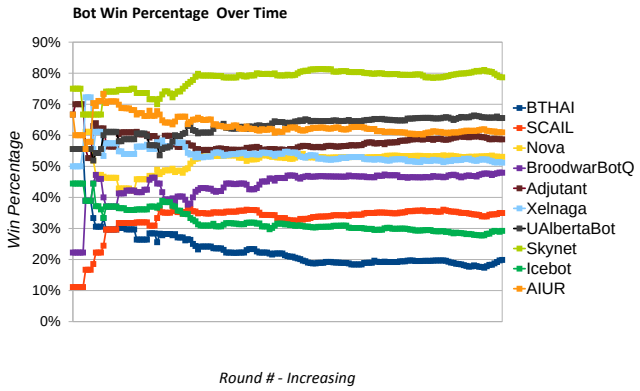

Fig. 5. Evolution of the win percentage of each bot participating in the CIG 2012 competition

league play maps found on the internet: *(2)MatchPoint 1.3*, *(4)Fighting Spirit 1.3*, *iCCupdestination 1.1*, *iCCup gaia*, and *iCCup great barrier reef*. Each bot pairing played on every map twice, with switched starting positions.

The two top bots of every bracket qualified for the final round. Table IV summarizes the results. Note that as BroodwarBotQ and BTHAI have the same number of wins, their direct encounter was evaluated which accounted 6:4 for the BroodwarBotQ. The bots going into the final were thus UAlbertaBot, Skynet (from bracket A) and Xelnaga and

TABLE VI
RESULTS OF THE CIG 2012 COMPETITION.

Position	bot	win %
1	Skynet	78.3%
2	UAlbertaBot	65.2%
3	AIUR	60.4%
4	Adjutant	58.6%
5	Nova	52.4%

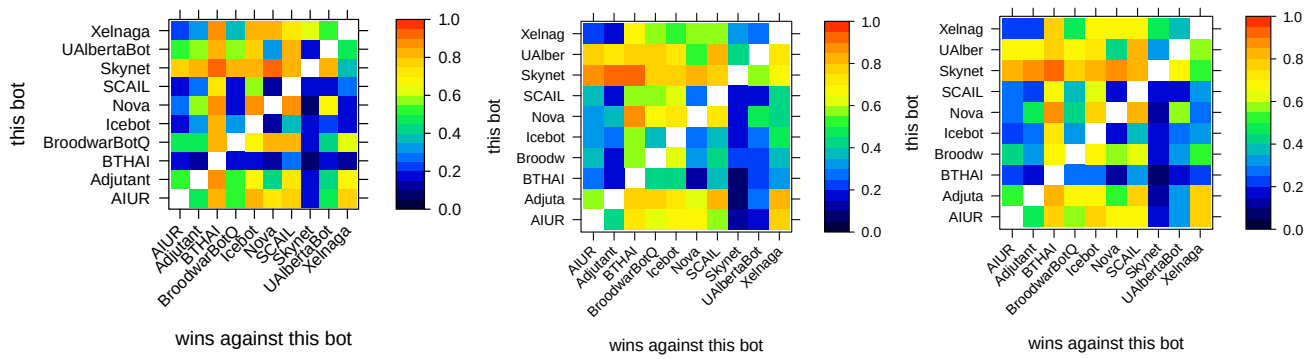

Fig. 6. Win percentages of CIG 2012 competition, from left to right: 3-player maps only, 6-player maps only, all maps. Read from line to column, bot in row wins given fraction of games against bot in column. For some bots, we find interesting differences, e.g. Xelnaga gets worse on 6-player maps, UAlbertaBot gets better. Only Xelnaga can reliably beat Skynet, but only on 3-player maps

BroodwarBotQ (from bracket B). All qualified bots play the Protoss faction. Most bots proved pretty stable, only Xelnaga and Protoss Beast Jelly crashed relatively often (each in more than a quarter of the games). Crashing of course resulted in an instant win for the other bot. In some cases, neither bot was able to finish the other off completely, so that they went into a passive state. We manually ended such games after around 15 minutes and assigned victory to the bot that had obtained more points as indicated on the end game screen.

2) Final Round: The final round was played in a similar mode as each of the first round brackets, using another set of 5 previously unknown maps: *iCCup lost temple 2.4*, *iCCup rush hour 3.1*, *iCCup swordinthemoon 2.1*, *iCCup yellow 1.1*, and *La_Mancha 1.1*. Letting each pairing play on each map twice again with switching starting positions resulted in 30 games per bot. The final results are displayed in table V, indicating Skynet as winner and UAlbertaBot as runner-up, being almost equally strong, and the two other bots as clearly inferior. The competition setup, documentation and results can be found in[18].

For CIG 2012, the AIIDE tournament software was employed, leading to a total of 4050 games played in 90 rounds of round robin. As 6 different maps were used, this means that each bot played every other on every map 15 times. As in the AIIDE competition, writing to and reading from a bot specific directory was enabled, however, due to technical reasons, this feature was constrained to the computer (of 6) the game was actually run on. We can therefore assume that this feature was of minor use for the CIG competition. The only other difference to the AIIDE competition was that the used maps were not made available to the competitors in advance.

These maps came in two flavors, namely three 3-player maps: *Athena-II*, *Neo Moon Glaive*, *Tears of the Moon*, and three 6-player maps: *Legacy*, *River of Light*, and *The Huntress 1.1*. We shall note that some bots consistently crashed on one of the originally considered maps which has thus been replaced. This is surprising as all maps are well known league play maps or have been provided with the StarCraft Broodwar distribution itself. Setup, replays and results for the CIG 2012 competition can be found here[19].

The overall results are displayed in table VI, and the win rate evolution over time in figure 5. These are quite consistent with the results of the AIIDE 2012 competition, so that we can conclude that the best bots are not very dependent on knowing the maps beforehand. However, the bot vs. bot win rates as displayed in figure 6 show some interesting trends. On the maps with more possible start points, some bots do better than others, namely SCAIL, Adjutant, Nova, and UAlbertaBot, the latter probably due to its very efficient scouting routine. Some bots however suffer from the increased uncertainty about the enemies position, namely Xelnaga and BroodwarBotQ.

As already observed before in the previously described competitions, there are also bots who consistently beat top ranked bots but have severe problems against lower ranked bots. E.g., Xelnaga is especially strong against Skynet on the 3-player maps (about 70% wins). Reviewing the replays led to the assumption that Xelnaga usually tries to attack Skynet's probes with a dark templar strategy, and often succeeds. Nova does very well against the UAlbertaBot, and the replays show that it sometimes succeeds to lure the probes into its own base, where they get killed, leading to severe resource problems. However, we cannot tell how often this happens as this would require to review every single replay between the 2 bots. Summarizing, most bots seem to have improved, which becomes clear if the nearly unchanged BTHAI bot is taken as a baseline. In 2011, it won more than half of its qualifying games, in 2012 it came out last with around 20% wins. However, designing a bot in order to beat a top bot (as for Xelnaga with Skynet) leads to a very restricted strategy that often leads to failure if playing against different bots. Note that in the direct encounter between Xelnaga and AIUR, its ancestor, Xelnaga looses consistently.

Nevertheless, from the observations we made during the tournament, we can draw the conclusion that the available bots are still very constrained. No bot in the competition played the Zerg race, which is surprising as the AIIDE 2010 winner (Overmind) did so. Presumably, implementing a good Zerg strategy is more demanding than for the Protoss or Terran races. Many bots consistently crashed when playing against a

[18]http://ls11-www.cs.tu-dortmund.de/rts-competition/StarCraft-cig2011

[19]http://ls11-www.cs.tu-dortmund.de/rts-competition/StarCraft-cig2012

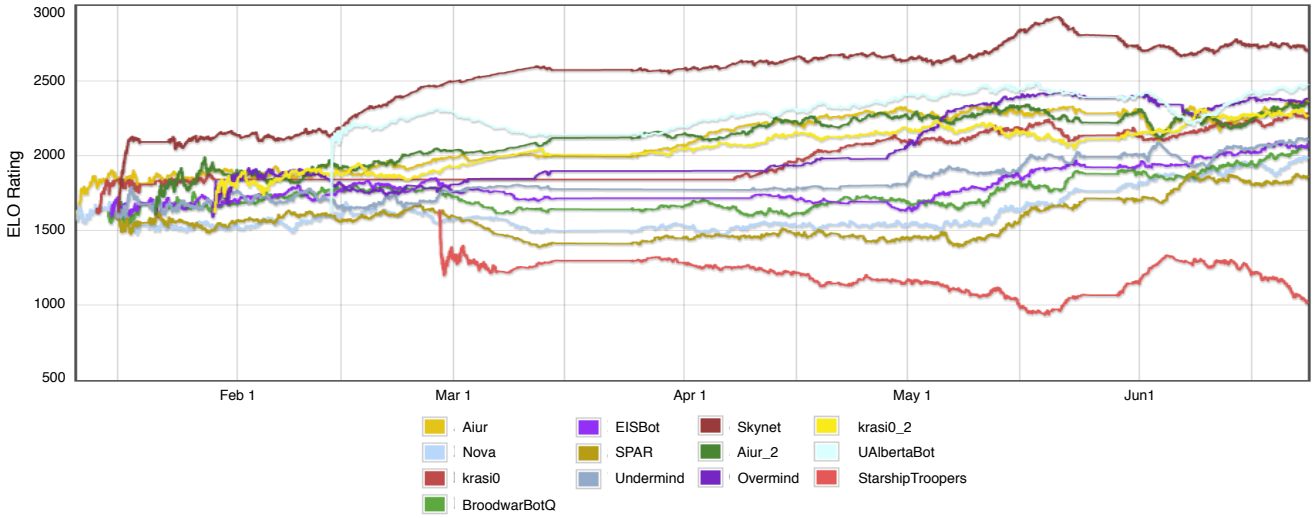

Fig. 7. StarCraft Bot Ladder: bot's Elo Rating from February 1, 2012 to June 20, 2012

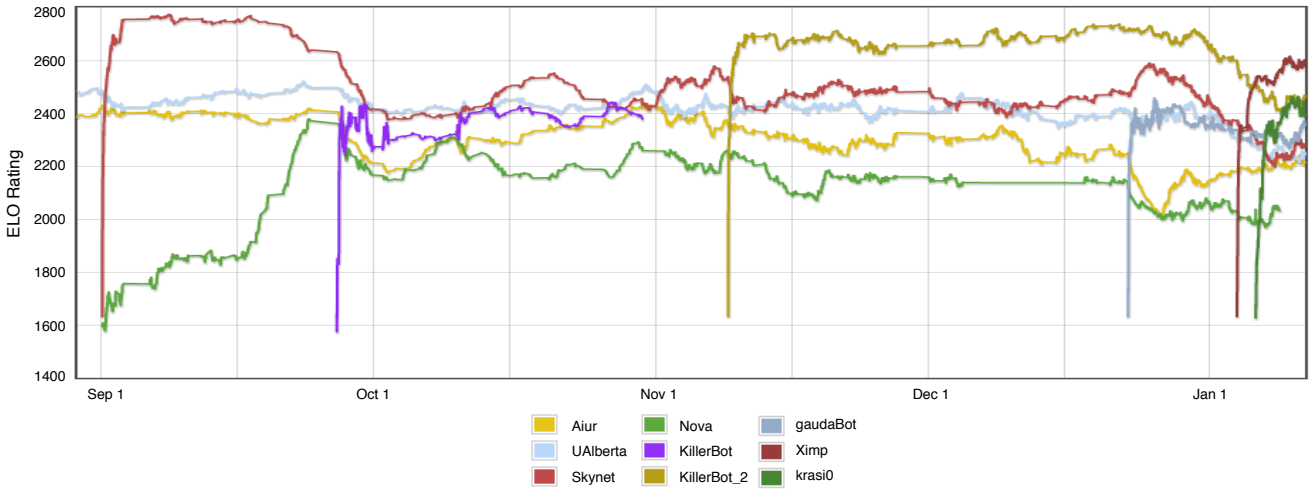

Fig. 8. StarCraft Bot Ladder: bot's Elo Rating from September 1, 2012 to January 10, 2013

random race built-in bot for testing, and also did so when the map size was changed from 128×128 to any other. Furthermore, every single bot sometimes failed to finish off an already beaten opponent, such that the game had to be stopped after a previously determined maximum time. It also seems that most of the current bots are not very good at adapting their strategy to the one of their opponent during a game, or at least (via the read/write procedure of game information) within a series of games.

C. StarCraft Bot Ladder

The StarCraft Bot Ladder is a website[20] where bot versus bot matches are automatized, and are running all the time. This ladder is a great resource for creating data sets (all the game replays are available) and statistics. For bot ranking, the ladder uses an ELO rating system suitable for calculating the *relative skill level* of a bot in two-player games. In the ELO system each player has a numerical rating that gets incremented or decremented some points after each game. The amount of points depends on the difference in the ratings of the players. A player will gain more points by beating a higher-rated player than by beating a lower-rated player. These kind of rating system is widely used in games like chess. This bot ladder compiles different versions of bots from the main worldwide competitions (like AIIDE, CIG or SSCAI[21]), even some independent or "under construction" bots. Therefore, it is a very good resource to test the performance of new bots against the current state of the art in StarCraft bots before participating in the official competitions.

Figure 7 shows the ELO rating of the bots in the ladder during the first half year of 2012. The ranking is practically equal than the AIIDE 2011 competition, showing the lack of adaptability of the current bots. We can notice these more

[20] http://bots-stats.krasi0.com

[21] http://sscaitournament.com

extremely in Figure 8 for the second half year of 2012. During this period new bots were introduced in the ladder. We can observe how the first version of KillerBot made a huge impact on Skynet ranking and finally, the second version of KillerBot quickly became the best bot for more than one month (again we can see how the rest of the bots aren't able to adapt and the ranking doesn't change so much). And finally, in January, the Ximp bot appears with a new strategy that overcomes the rest of the bots. Both KillerBot and Ximp use hard-coded strategies without any kind of adaptation capabilities. However, they implement strategies that no other bot has a counter for, and thus manage to win a very large percentage of games. This points out, once again, that one of the major open challenges in RTS game AI is how achieving adaptive strategies, that can recognize the opponent's intentions, and select an adequate response.

D. Adaptation Analysis

One of the major conclusions from the results of the StarCraft competitions, is the lack of adaptation of bots. Some switch between different build-orders, but do not fully adapt their strategy. No bot is capable of observing the opponent and autonomously synthesize a good plan from scratch to counter the opponent strategy. In this section we analyzed this claim quantitatively. We analyzed the replays from the 2011 and 2012 AIIDE competitions (shown in Figures 9 and 10 respectively). Specifically, we analyzed the way bots choose their openings depending on which other bot they are playing against. Given that there is no dominant strategy in StarCraft, and it is necessary to see what the opponent is doing in order to determine the best opening, we would expect bots to change their openings depending on which other bot they are playing (since each bot uses a different strategy, or set of strategies). Using clustering, we identified the most common openings in all the replays from the 2011 and 2012 competitions (each one shown with a different color in the figures).

Specifically, we identified the following openings (Most of the following names are specific building or units of StarCraft, for a better comprehension we refer the reader to [22]):

- Protoss openings:
  - *two_gates*: Build two *Gateways* and keep training *Zealots*.
  - *fast_dt*: Produce *Dark Templars* as soon as possible.
  - *templar*: Train *High Templars* as fast as possible.
  - *speedzeal*: Train *Zealots* and research attack and speed upgrades as soon as possible.
  - *corsair*: Produce *Corsairs* as soon as possible and then make transition to train *Dark Templars*, *Reavers* or *Dragoons*.
  - *nony*: Build three *Gateways* and massive training of *Dragoons*.
  - *reaver_drop*: Train *Reavers* as soon as possible to be able to do drops.
- Terran openings:
  - *bio*: Produce a large army of *Marines* and *Medics*.
  - *rax_fe*: Take the closest "natural expansion" as soon as possible. This provides a big economic boost in the early to mid game.
  - *two_facto*: Build two *Factories* and keep producing *Tanks*.
  - *vultures*: Produce mainly *Vultures* and research mines.
  - *air*: Produce *Wraiths* as soon as possible for an air attack.
  - *drop*: Train *Dropships* as soon as possible to be able to do drops.
- Zerg openings:
  - *speedlings*: Train *Zerlings* and research speed upgrade as soon as possible.
  - *fast_mutas*: Produce mainly *Mutalisks*.
  - *mutas*: Expand two times for a stronger economy before massive training of *Mutalisks*.
  - *lurkers*: Train *Lurkers* as soon as possible to take benefit of its cloak ability.
  - *hydras*: Massive production of *Hydralisks*.

As the figures show, the top three ranked bots in the competition (Skynet, Aiur and UalbertaBot) do not change their strategy at all depending on their opponent. For example, the Skynet bot (both in 2011 and 2012), always uses the same opening (*two_gates*), except when playing a Terran opponent, when it uses *nony*. This reflects the trend that the performance of bots is still more dependent on carefully hand-crafted and non-adaptive behaviors, than on on-line decision making procedures. This is so, since most of the problems that need to be solved in order to implement such procedures are still open.

VI. Open Questions in RTS Game AI

As illustrated in this paper, there is a set of problems in RTS game AI that could be considered mostly solved, of for which we have very good solutions. One example of such problems is pathfinding (mostly solved) or low-scale micro-management (for which we have good solutions). However, there are many other problems for which this is not the case. For example, there is no current StarCraft bot that can come up with its own tactical moves, such as "unit drops" in response to an observed opponent strategy. Some bots do drops, but only if this is hard-coded; no bot has the capability of reasoning about the current situation, synthesize a tactical move that involves a "unit drop", and determine that this move is the best one in the current situation. This is related to the lack of real-time adversarial planning techniques that scale up to the size required for RTS games.

We present here a list of problems that are currently unsolved, grouped in various categories.

- **Learning and adaptation:**
  - Adaptation to opponent strategy: observing the opponent strategy, and synthesizing an adequate counter strategy. Current bots switch between predefined strategies based on hard-coded preconditions, or based on the performance of each predefined strategy against an opponent in previous games, but no

[22] http://wiki.teamliquid.net/starcraft/Category:Strategies

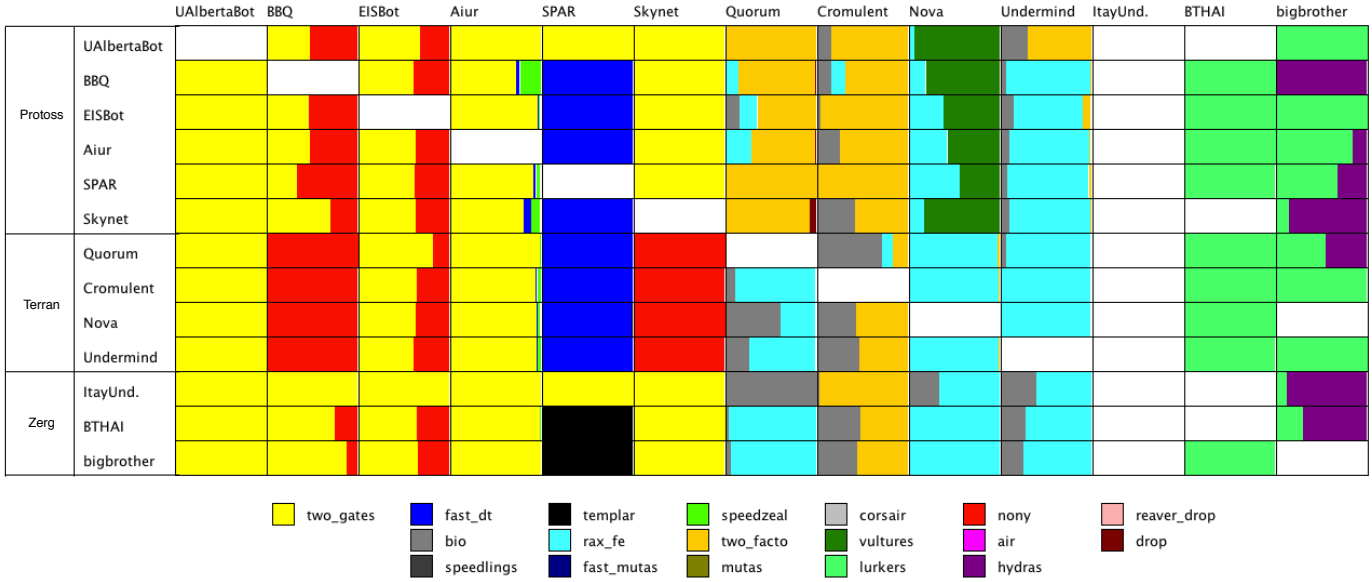

Fig. 9. Distribution of different openings performed by the different bots participating in the AIIDE 2011 competition. For each bot match-up, the colors show the proportion of times that the column bot used a particular opening against the row bot.

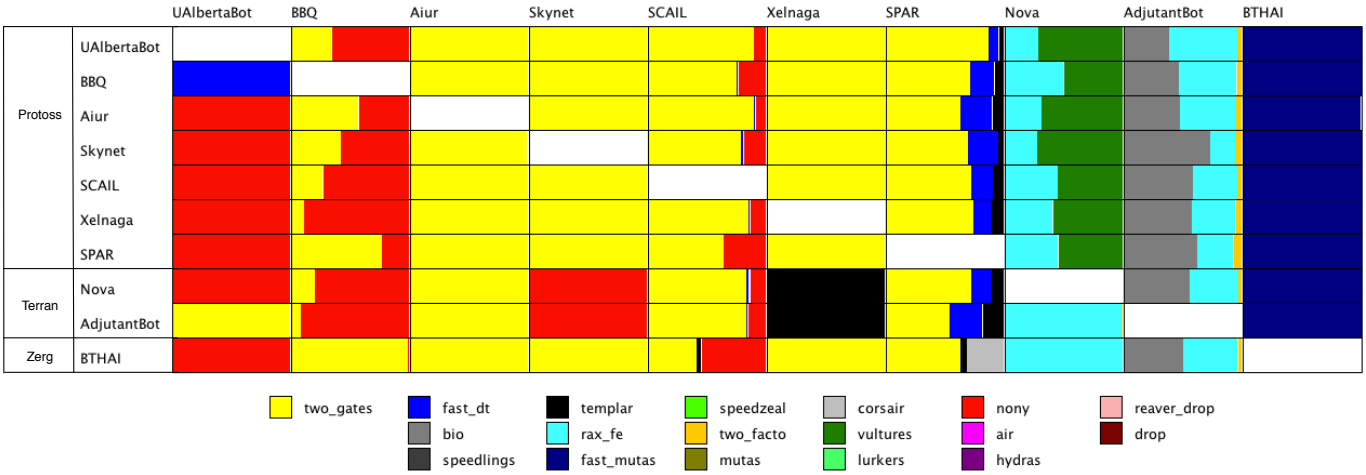

Fig. 10. Distribution of different openings performed by the different bots participating in the AIIDE 2012 competition. For each bot match-up, the colors show the proportion of times that the column bot used a particular opening against the row bot.

current bot creates new strategies (like Chess or Go playing programs do).

- Learning from experience in RTS games: how can we make a bot that improves performance over time? some current bots learn which strategy (out of a predefined set of strategies) is best against a given opponent, but how can we devise learning strategies that can perform more general learning? This has been achieved in classical board games, such as Chess [63], in the context of game-tree search (by learning the evaluation function). But it's unclear how to do it in RTS games.
- Learning from observation (from demonstration, or from observing the opponent) in RTS games: how can we learn by observing the game play of other players? can we devise algorithms that can automatically extract strategies from observation, and later apply them? There has been some work in this direction [59], but it is very far from being mature.

- **Planning:**
  - Adversarial planning under real-time constraints: although some solutions for small-scale real-time planning have been recently proposed (such as [52], based on alpha-beta game-tree search), the problem of large-scale adversarial planning under real-time constraints is still open.
  - Adversarial planning under uncertainty of partially-observable domains: how can we adapt adversarial planning techniques for dealing with uncertainty? This problem has been widely studied in the context of simple games such as back-gammon [64], or Poker [65]. However, the techniques developed for those domains do not scale to RTS-game scenarios.
  - Adversarial planning with resources: similarly, even

if there exist planning algorithms that handle resources (like GRT-R [66]), they cannot scale up to the size of problems needed for RTS games like StarCraft.

- **Integration:**
  - Multi-scale planning/reasoning: as described in this paper, all the bots developed for the StarCraft AI competitions decompose the problem of playing an RTS game into smaller subproblems, and then solutions for each of those subproblems are integrated in to a common architecture to play the game. However, the integration of each of the modules in a unified architecture is still an open problem. For example, how can decisions made at high-level modules be integrated with decisions made at lower-level modules?
- **Domain Knowledge:**
  - We know how to incorporate some aspects of domain knowledge (e.g. build orders) into RTS game playing agents. But, in general, how to incorporate some forms of domain knowledge into algorithms for RTS games is still an open problem. For example, standard techniques to encode strategies for other forms of games, like Behavior Trees, are hard to deploy in RTS games. Is it possible to devise techniques that can automatically mine the existing collections of domain knowledge for an RTS game like StarCraft, and incorporate it into the bot? An initial exploration of this idea was carried out by Branavan et al. [67].

VII. Conclusions

As the list in the previous section indicates, Real-Time Strategy games are an excellent testbed for AI techniques, which pose a very large list of open problems. As Section V has shown, the current top performing programs to play RTS games such as StarCraft still rely mainly on hard-coded strategies. It is still possible to perform strongly, or even win, one of these competitions simply by finding a hard-coded strategy that no other bot has a predefined counter-strategy for. Additionally, good human players are still clearly superior to the best computer programs.

One of the main goals of this paper is to provide a centralized and unified overview to the research being done in the area of RTS game AI. To that end, in this paper we have highlighted the existing challenges in RTS games, from an AI point of view, and surveyed the recent advances towards addressing these challenges with a focus on StarCraft (which has emerged as a unified test-bed). Given that playing an RTS game is a very challenging task, researchers tend to divide such task into smaller tasks, that can be individually addressed by AI techniques. We have also surveyed the different task subdivisions used in some of the top StarCraft-playing programs, highlighting advantages and disadvantages. Additionally, we have presented an analysis of the results of the different StarCraft AI competitions, highlighting strengths and weaknesses of each of the bots. Finally, we have closed the paper with a list of specific open research questions for future research.

As argued before, addressing these open research questions would not just lead to better RTS game playing bots, but would bring significant advances to many other areas of artificial intelligence. This is so because RTS games embody many characteristics of many other problems appearing in many areas of AI (such as robotics, multiagent systems, etc.).

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] M. Buro and D. Churchill, "Real-time strategy game competitions," *AI Magazine*, vol. 33, no. 3, pp. 106–108, 2012.

[3] G. Synnaeve, "Bayesian programming and learning for multi-player video games," *Ph.D. Thesis*, 2012.

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

[5] G. Synnaeve and P. Bessiere, "A dataset for starcraft ai & an example of armies clustering," in *AIIDE Workshop on AI in Adversarial Real-time games 2012*, 2012.

[6] B. G. Weber, M. Mateas, and A. Jhala, "Building human-level ai for real-time strategy games," in *Proceedings of AIIDE Fall Symposium on Advances in Cognitive Systems*, AAAI Press. Stanford, Palo Alto, California: AAAI Press, 2011.

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

[8] 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.

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

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

[11] D. Churchill and M. Buro, "Build order optimization in starcraft," *Proceedings of AIIDE*, pp. 14–19, 2011.

[12] 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.

[13] 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.

[14] 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.

[15] J. Young and N. Hawes, "Evolutionary learning of goal priorities in a real-time strategy game," 2012.

[16] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *Artificial Intelligence Communications*, vol. 7, no. 1, pp. 39–59, 1994.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

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

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

[24] 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

[25] 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

[26] 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.

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

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

[29] 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.

[30] 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.

[31] 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.

[32] G. Synnaeve and P. Bessière, "Special tactics: a bayesian approach to tactical decision-making," in *CIG (IEEE)*, 2012.

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

[34] 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.

[35] A. Uriarte and S. Ontañón, "Kiting in rts games using influence maps," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[36] J. Hagelbäck and S. J. Johansson, "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.

[37] J. Hagelbäck, "Potential-field based navigation in starcraft," in *CIG (IEEE)*, 2012.

[38] 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.

[39] 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

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

[41] 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.

[42] L. Liu and L. Li, "Regional cooperative multi-agent q-learning based on potential field," pp. 535–539, 2008.

[43] 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 (TCIAIG)*, vol. 2, no. 2, pp. 82–98, June 2010.

[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] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[46] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar," in *CIG (IEEE)*, 2012.

[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] U. Jaidee and H. Muñoz-Avila, "Classq-l: A q-learning algorithm for adversarial real-time strategy games," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[50] M. Ponsen and I. P. H. M. Spronck, "Improving adaptive game AI with evolutionary learning," in *University of Wolverhampton*, 2004, pp. 389–396.

[51] N. Othman, J. Decraene, W. Cai, N. Hu, and A. Gouaillard, "Simulation-based optimization of starcraft tactical ai through evolutionary computation," in *CIG (IEEE)*, 2012.

[52] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for rts game combat scenarios," 2012.

[53] 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.

[54] S. Koenig and M. Likhachev, "D*lite," in *AAAI/IAAI*, 2002, pp. 476–483.

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

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

[57] A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 1160–1168, 2006.

[58] 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

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

[60] 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.

[61] J. Young, F. Smith, C. Atkinson, K. Poyner, and T. Chothia, "Scail: An integrated starcraft ai system," in *CIG (IEEE)*, 2012.

[62] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[63] G. Tesauro, "Comparison training of chess evaluation functions," in *Machines that learn to play games*. Nova Science Publishers, Inc., 2001, pp. 117–130.

[64] ——, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.

[65] J. Rubin and I. Watson, "Computer poker: A review," *Artificial Intelligence*, vol. 175, no. 5, pp. 958–987, 2011.

[66] I. Refanidis and I. Vlahavas, "Heuristic planning with resources," in *ECAI*, 2000, pp. 521–525.

[67] S. Branavan, D. Silver, and R. Barzilay, "Learning to win by reading manuals in a monte-carlo framework," in *Proceedings of ACL*, 2011, pp. 268–277.