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

Santiago Ontañón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and 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, review1, *StarCraft*.**

I. Introduction

THE field of real-time strategy (RTS) game AI has advanced significantly since Michael Buro's call for research in this area [1]. Specifically, competitions like the Open Real-Time Strategy (ORTS) Game AI Competition (held from 2006 to 2009), the Artificial Intelligence and Interactive Digital Entertainment (AIIDE) StarCraft AI Competition (held since 2010), and the Computational Intelligence and Games (CIG) StarCraft RTS AI Competition (held since 2011) 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 neither perfect nor complete information about the current state or about the dynamics of the environment are available, pose significant challenges for AI. 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, such as 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, and 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 from 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.

II. Real-Time Strategy Games

RTS is a subgenre 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 as follows.

- 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 to complete.
- RTS games are "real time," which actually means that each player has a very small amount of time to decide the next move. Compared to *Chess*, where players may have several minutes to decide the next action, in *StarCraft*, the game executes at 24 frames/s, which means that players can act as fast as every 42 ms, 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 nondeterministic. Some actions have a chance of success.

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

Manuscript received March 20, 2013; revised August 09, 2013; accepted October 03, 2013. Date of publication October 18, 2013; date of current version December 11, 2013.

S. Ontañón and A. Uriarte are with the Computer Science Department, Drexel University, Philadelphia, PA 30309 USA (e-mail: santi@cs.drexel.edu; albertouri@cs.drexel.edu).

G. Synnaeve is with the Laboratory of Cognitive Science and Psycholinguistics (LSCP), Ecole Normale Supérieure (ENS), Paris 75005, France (e-mail: gabriel.synnaeve@gmail.com).

F. Richoux is with the Nantes Atlantic Computer Science Laboratory (LINA), Université de Nantes, Nantes 44322, France (e-mail: florian.richoux@univ-nantes.fr).

D. Churchill is with the Computing Science Department, University of Alberta, Edmonton, AB T6G 2R3 Canada (e-mail: dave.churchill@gmail.com).

M. Preuss is with the European Research Center for Information Systems (ERCIS), Münster University, Münster 48149, Germany (e-mail: mike.preuss@uni-muenster.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2013.2286295

- Finally, the complexity of these games, both in terms of state-space size and in terms of a 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 Section II-A.

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 types 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 describing *StarCraft* as a research testbed, and to 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 universe 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 processes, but they are strong and resistant. These conditions make players follow a strategy of quality over quantity.
- Zerg, the insectoid race, units are cheap and weak. They can be produced fast, encouraging players to overwhelm their opponents with sheer numbers.

Fig. 1 shows a screenshot of *StarCraft* showing a player playing the Terran race. 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 to 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 squares of pixels, also known as build tiles. However, the resolution of walkable areas is in squares of 8×8 pixels, also known as walk tiles. The typical dimensions for maps range from 64×64 to 256×256 build tiles. Each player can control up to 200 units (plus an unlimited number of buildings). Moreover, each different race contains between 30 and 35 different types of units and buildings, most of them with a significant number of special abilities. All these factors together make *StarCraft* a significant challenge, in which humans are still much better than computers. 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^+.

Fig. 1. Screenshot of *StarCraft: Brood War*.

From a theoretical point of view, the state space of a *StarCraft* game for a given map is enormous. For example, consider a 128×128 map. At any given moment, there might be between 50 and 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 128×128 possible positions per unit), and 400 units, gives us an initial number of $16\,384^{400} \approx 10^{1685}$. 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], with a total game complexity of b^d. In *Chess*, $b \approx 35$ and $d \approx 80$. In more complex games, like *Go*, $b \approx 30$ to 300, and $d \approx 150$ to 200. In order to determine the branching factor in *StarCraft* when an AI plays it, we must keep in mind that the AI can issue actions simultaneously to as many units in the game as desired. Thus, considering that, in a typical game, a player controls between 50 and 200 units, the branching factor would be between u^{50} and u^{200}, where u is the average number

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

of actions each unit can execute. Estimating the value of u is not easy, since the number of actions a unit can execute is highly dependent on the context. Let us make the following assumptions: 1) at most, 16 enemy units will be in range of a friendly unit (larger values are possible, but unlikely); 2) when an AI plays *StarCraft*, it only makes sense to consider movement in the eight cardinal directions per unit (instead of assuming that the player can issue a "move" command to anywhere in the map at any point in time); 3) for "build" actions, we consider that SCVs (Terran worker units) only build in their current location (otherwise, if they need to move, we consider that as first issuing a "move" action, and then a "build"); and 4) let us consider only the *Terran* race. With those assumptions, units in *StarCraft* can execute between 1 (units like "Supply Depots," whose only action is to be "idle") to 43 actions (Terran "Ghosts"), with typical values around 20–30. Now, if we have in mind that actions have cooldown times, and thus not all units can execute all of the actions at every frame, we can take a conservative estimation of about ten possible actions per unit per game frame. This results in a conservative estimate for the branching factor between $b \in [10^{50}, 10^{200}]$, only considering units (ignoring the actions buildings can execute). Now, to compute d, we simply consider the fact that typical games last for about 25 min, which results in $d \approx 36\,000$ (25 min $\times$ 60 s $\times$ 24 frames/s).

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;
- spatial and temporal reasoning;
- collaboration (between multiple AIs);
- opponent modeling and learning;
- adversarial real-time planning.

While there has been 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 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 (RL) techniques, but also opponent modeling. The main problem again is the fact that the state space is too large as well as the fact that RTS games are partially observable.
- Intergame 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 among 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. It is involved in tasks such as building placement or base expansion. In the former, the player needs to carefully consider building positioning 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 locations to build a new base, with regard to both 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 led into a bottleneck.

Another example of spatial reasoning in *StarCraft* is that it is always an advantage to have one's own units on high ground while the enemy is on low ground, since units on low ground have no vision onto the high ground.

Analogously, temporal reasoning is key in tactical or strategic reasoning, for example, timing attacks and retreats to gain an advantage. At a higher strategic level, players need to reason 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 endgame tables. In the case of RTS games, it is still unclear how the significantly large amount of

domain knowledge (in the form of 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. On the other hand, large data sets of replays have been created [4], [5], from where strategies, trends, or plans have been tried. However, *StarCraft* games are quite complex, and how to automatically learn from such data sets is still an open problem.

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

- Strategy: It 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, and concerns the whole set of units a player owns.
- Tactics: These are the implementation of the current strategy. It implies army and building positioning, movements, timing, and so on. Tactics concerns a group of units.
- Reactive control: This is the implementation of tactics. It consists in moving, targeting, firing, fleeing, hit-and-run techniques (also knows as "kiting") during battle. Reactive control focuses on a specific unit.
- Terrain analysis: It consists in the analysis of regions composing the map: chokepoints, minerals and gas emplacements, low and high walkable grounds, islands, etc.
- Intelligence gathering: It 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 talks about two tasks.

- Micro: It is the ability to control units individually (roughly corresponding to reactive control above, and part of tactics). A good micro player usually keeps their units alive over a longer period of time.
- Macro: It is the ability to produce units and to expand at the appropriate times to keep your production of units flowing (roughly corresponding to everything but reactive control and part of tactics above). 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. Moreover, we would like to point out that the task decomposition above is not the only possible approach. Some systems, such as the influence-map-based artificial intelligence (IMAI) [7], divide gameplay into much smaller tasks, which are then assigned resources depending on the expected benefits of achieving each task.

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, we will divide it according to three levels of abstraction: strategy (which loosely corresponds to "macro"), tactics, and reactive control (which loosely correspond to "micro").

Fig. 2 graphically illustrates how strategy, tactics, and reactive control are three points in a continuum scale where strategy corresponds to decision-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 constrain 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 strategic 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, reactive control describes how the player controls individual units to maximize their efficiency 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, in order 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, it

[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.

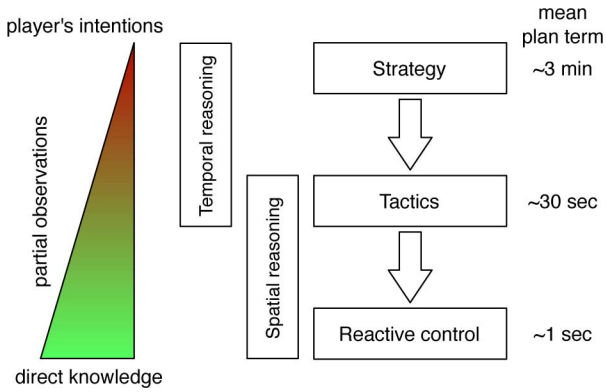

Fig. 2. RTS AI levels of abstraction and their properties: uncertainty (coming from partial observation and from not knowing the intentions of the opponent) is higher for higher abstraction levels. Timings on the right correspond to an estimate of the duration of a behavior switch in *StarCraft*. Spatial and temporal reasoning are indicated for the levels at which greedy solutions are not enough.

has been addressed using many AI techniques, such as 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 ones use finite-state machines (FSM) [8] 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 FSMs, in which FSMs are composed hierarchically. These hard-coded approaches have achieved a significant amount of success, and, as we will discuss in Section IV, they have also been used in many academic RTS AI research systems. 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.* [9] 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, which are then composed at runtime in order to form full-fledge strategies to play the game. In [10], 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 [11]. 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, HTN and CBP being the only ones explored so far. Moreover, none of these approaches addresses any timing or scheduling issues, which are key in RTS games. One notable exception is the work of Churchill and Buro [12], 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.* [13] used hidden Markov models (HMMs) 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 [14] used the data set of [4] and presented a Bayesian semisupervised model to learn from replays and predict openings (early game strategies) from *StarCraft* replays. The openings are labeled by expectation–maximization (EM) clustering considering appropriate features. Then, in [15], they presented an unsupervised learning Bayesian model for tech-tree prediction, still using replays. Finally, evolutionary approaches to determine priorities of high level tasks were explored by Young and Hawes in their QUORUM system [16], showing improvement over static priorities.

Also falling into the machine learning category, a significant group of researchers has explored case-based reasoning (CBR) [17] approaches for strategic decision making. For example, Aha *et al.* [18] used CBR to perform dynamic plan retrieval in the Wargus domain. Hsieh and Sun [19] based their work on Aha *et al.*'s CBR model [18] and used *StarCraft* replays to construct states and building sequences (build orders). Schadd *et al.* [20] 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). Jaidee *et al.* [21] study the use of CBR for automatic goal selection, while playing an RTS game. These goals will then determine which Q-tables are to be used in an RL framework. Finally, Čertický and Čertický [22] used CBR to build their army, based on the opponent's army composition, and they pointed out the importance of proper scouting for better results.

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 partially observable Markov decision processes (POMDPs) from being directly applicable, and very few of the previous approaches deal with this problem. Much work in RTS game AI assumes perfect information all the time. For example, in the case of commercial games, most AI implementations cheat, since the AI can see the complete game map at all times, while the human player does not. In order to make the human player believe that the AI of these 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. Even if the *StarCraft* AI competition enforces fog of war, which means that bots are forced to work under partial information, little published research exists on this topic. A notable exception is the work of Weber *et al.* [23], 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 [24], [25], 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 into two parts: terrain analysis and decision making. Terrain analysis supplies the AI with structured information about the map in order to help make decisions. This analysis is usually performed offline, in order to save central processing unit (CPU) time during the game. For example, Pottinger [26] 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.* [27] showed the importance of having qualitative spatial information for wargames, for which they used geometric and pathfinding analysis. Hale *et al.* [28] presented a 2-D geometric navigation mesh generation method from expanding convex regions from seeds. Finally, Perkins [29] applied Voronoi decomposition (then pruning) to detect regions and relevant choke points in RTS maps. This approach is implemented for *StarCraft* in the *Brood War* terrain analyzer (BWTA)[4] library, used by most state-of-the-art *StarCraft* bots.

Walling is the act of intentionally placing buildings at the entrance of your base to block the path and to prevent the opponent's units from getting inside. This technique is used by human *StarCraft* players to survive early aggression and earn time to train more units. Čertický solved this constraint satisfaction problem using answer set programming (ASP) [30].

Concerning tactical decision making, many different approaches have been explored, such as machine learning or game-tree search. Hladky and Bulitko [31] benchmarked hidden semi-Markov models (HSMMs) and particle filters for unit tracking. Although they used first-person shooter (FPS) games for their experimentation, the results apply to RTS games as well. They showed that the accuracy of occupancy maps was improved using movement models (learned from the player behavior) in HSMMs. Kabanza *et al.* [32] improve the probabilistic hostile agent task tracker (PHATT [33], a simulated HMM for plan recognition) by encoding strategies as HTN, used for plan and intent recognition to find tactical opportunities. Sharma *et al.* [34] combined CBR and RL to enable reuse of tactical plan components. Cadena and Garrido [35] used fuzzy CBR (fuzzy case matching) for strategic and tactical planning. Synnaeve and Bessière [36] combined space abstraction into regions from [29] and tactical decision making by assigning scores (economical, defenses, etc.) to regions and looking for their correspondents in tactical moves (attacks) in pro-gamer replays. Finally, Miles and Louis [37] created the idea of IMTrees, a tree where each leaf node is an influence map, and each intermediate node is a combination operation (sum, multiplication). Miles and Louis used evolutionary algorithms to learn IMTrees for each strategic decision in the game, involving spatial reasoning by combining a set of basic influence maps.

Game-tree search techniques have also been explored for tactical decision making. Churchill *et al.* [38] presented the alpha–beta considering duration (ABCD) algorithm, a game-tree search algorithm for tactical battles in RTS games. Chung *et al.* [39] applied Monte Carlo planning to a capture-the-flag version of ORTS. Balla and Fern [40] applied the UCT algorithm (a Monte Carlo tree search algorithm) to tactical assault planning in Wargus. To make game-tree search 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, with that of Weber *et al.* [23] being 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 and influence maps have been found to be useful techniques for reactive decision making. Some uses of potential fields in RTS games are: avoiding obstacles (navigation), avoiding opponent fire [41], or staying at maximum shooting distance [42]. Potential fields have also been combined with A* pathfinding to avoid local traps [43]. Hagelbäck and Johansson [44] presented a multiagent potential field-based bot able to deal with fog of war in the *Tankbattle* game. Avery *et al.* [45] and Smith *et al.* [46] coevolved influence map trees for spatial reasoning in RTS games. Danielsiek *et al.* [47] used influence maps to achieve intelligent squad movement to flank the opponent in an RTS game. Despite their success, a drawback for potential field-based techniques is the large number of parameters that have to be tuned in order to achieve the desired behavior. Approaches for automatically learning such parameters have been explored, for example, using RL [48], or self-organizing maps (SOMs) [49]. 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 getting 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 [50], which subsumes potential fields, allowing for integration of tactical goals directly in micromanagement.

Additionally, there have been some interesting uses of RL [51]: Wender and Watson [52] evaluated the different major RL algorithms for (decentralized) micromanagement, which all

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

perform equally. Marthi *et al.* [53] 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.* [54] advocate the use of prior domain knowledge to allow faster RL 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) [53]. Another approach has been proposed by Jaide and Muñoz-Avila [55] 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 [56] used evolutionary learning techniques, but face the same problem of dimensionality. For example, evolutionary optimization by simulating fights can easily be adapted to any parameter-dependent micromanagement control model, as shown by [57], which optimizes an AIIDE 2010 micromanagement competition bot.

Finally, approaches based on game-tree search have recently been explored for micromanagement. Churchill *et al.* 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.* [58] have explored humanlike 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 or strategical reasoning (although there are some exceptions, like the tactical pathfinding of [47]). 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 [59] 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 a technique is Triangulation Reduction A*, which computes polygonal triangulations on a grid-based map [60]. Considering movement for groups of units, rather than individual units, techniques such as steering of flocking behaviors [61] can be used on top of a pathfinding 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 have been inspired by continuum crowds (flow field) [62]. A comprehensive review about (grid-based) pathfinding was recently done by Sturtevant [63].

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 [64] (which uses a combination of CBR and learning from demonstration) or ALisp [53], there has not 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 only been explored in the context of smaller scale RTS games [65]. Techniques that scale up to large RTS games such as *StarCraft* are still not available.

A related problem is that of integrating reasoning at multiple levels of abstraction. Molineaux *et al.* [66] showed that the difficulty of working with multiscale goals and plans can be handled directly by CBR, via an integrated RL/CBR algorithm using continuous models. Reactive planning [24], a decompositional planning similar to hierarchical task networks [11], allows for plans to be changed at different granularity levels and so for multiscale (hierarchical) goals integration of low-level control. Synnaeve and Bessière [50] 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 Section III 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 [18], Darmok [64], or ALisp [53], 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-level 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 subproblems, which can be dealt with using individual AI techniques.

Fig. 3 shows some representative examples of the architectures used by different bots in the AIIDE and CIG *StarCraft* AI competitions (see Section V): BroodwarBotQ [50], Nova [41], UAlbertaBot [12], Skynet, SPAR, AIUR, and BTHAI [43]. 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 into two sets of modules: intelligence and decision making. Intelligence in SPAR has three modules dedicated to analyzing 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 BroodwarBotQ (BBQ) only use a hierarchical organization for combat (controlling the attack units), but use a decentralized organization

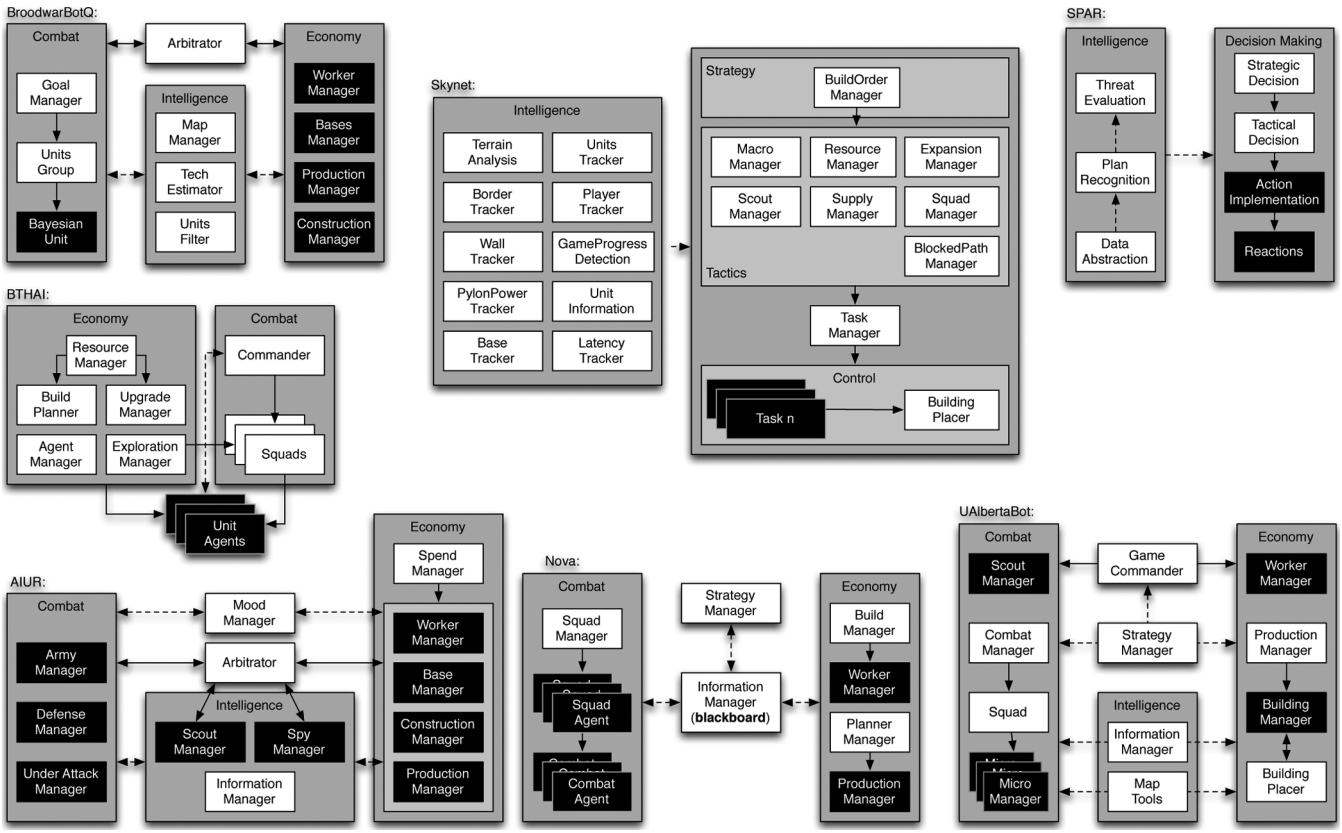

Fig. 3. Architecture of seven *StarCraft* bots obtained by analyzing their source code. Modules with black background send commands directly to *StarCraft*, dashed arrows represent data flow, and solid arrows represent control.

for the rest of the bots. 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: a bidding market and broker for settling units control, military and civilian groups/task forces bid for units proportionally to their usefulness, and the task importance.

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 2-D plane, where the vertical axis represents abstraction, and the horizontal axis represents the different aspects of the game (combat, 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 Fig. 3, we can see the following use of abstraction and divide and conquer in the bots.

- BroodwarBotQ[5]: It uses abstraction for combat, and divide and conquer for economy 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]: It is similar in design to BroodwarBotQ, and uses abstraction for combat, and divide and conquer for economy. 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 the modules follow (thus, making use of abstraction).

[5]http://github.com/SnippyHolloW/BroodwarBotQ

[6]http://nova.wolfwork.com/

- UAlbertaBot[7]: It also uses abstraction in combat like the previous two bots. But it also uses it in economy: 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]: It makes extensive use of both abstraction and divide and conquer. We can see a high level module that issues commands to a series of tactic modules: the collection of tactic module 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 three-layered abstraction hierarchy, and uses divide and conquer in all levels except the highest.
- SPAR[9]: It 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]: It is mainly divide-and-conquer oriented, with a slight abstraction on economy 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. Combat is 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. However, this bot does not manage any kind of reactive controls so far.
- BTHAI[11]: It uses a two-tier abstraction hierarchy, where a collection of high-level modules command a collection of lower level agents in charge of each of the units. At the high level, BTHAI uses divide and conquer, having multiple high-level modules issuing commands to the lower level units.

Additionally, except for BTHAI, all other agents use divide and conquer at a higher level bot design and divide all the modules into two or three categories: intelligence gathering and decision making (sometimes divided into combat and economy).

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 few bots handle the problem of how to coordinate resource usage among 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 dictates base, production, and construction managers as to what they have to build/produce. It also orders the start of 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 build-order 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 prescripting 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 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, which 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 [67] or QUORUM [16]. 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 colocated with scientific conferences, have been possible thanks to the existence of the *Brood War* application programming interface (BWAPI),[12] which enables replacing the human player interface with the C++ code. Sections V-A and V-B summarize the results of all the *StarCraft* AI competitions held at the AIIDE and CIG conferences during past years. Additionally we analyze the statistics from the *StarCraft* bot ladder, where the best bots play against each other continuously over time.

A. AIIDE

Started in 2010, the AIIDE *StarCraft* AI Competition[13] is the most well-known and longest running *StarCraft* AI Competition

[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/

[12]http://code.google.com/p/bwapi/

[13]http://www.StarCraftAICompetition.com

in the world. Each year, AI bots are submitted by competitors to do battle within the retail version of *StarCraft: Brood War*, with prizes supplied by Blizzard Entertainment.

The first competition in 2010 was organized and run by Ben Weber in the Expressive Intelligence Studio at the University of California, Santa Cruz, CA, USA.[14] Twenty six 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 were kept for bots to learn about opponents between matches.

The 2010 competition had four 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 second place. Tournament 2 was another microfocused game with nontrivial terrain. Two competitors submitted for this category, with FreSCBot once again coming in first 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. Eight 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: Brood War* with fog of war enforced. The tournament was run with a random pairing double-elimination format with each match being best of five games. Competitors could play as any of the three races, with the only limitations in game play being those that were considered "cheating" in the *StarCraft* community. A map pool of five well-known professional maps were 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 the University of California at Berkeley, Berkeley, CA, USA. 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 it quite a powerful unit when controlled by a computer. Overmind used a potential field-based micromanagement system to guide its mutalisks, which led it to victory. Krasi0 came in second place with a standard defensive Terran opening strategy that transitioned into "mech" play in the late game.

In 2011, the University of Alberta, Edmonton, AB, USA, hosted the competition, 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 that achieved the highest win percentage over the course of the competition would be the determined winner. Also, the competition became open source, in an effort not only to prevent possible cheating, but also to promote healthy competition in future tournaments by giving newcomers an easier entry point by basing their design on previous bots.

TABLE I
RANKING OF THE THREE BEST BOTS OF THE AIIDE 2010 COMPETITION

Position	Bot
1	Overmind
2	Krasi0
3	Chronos

TABLE II
RESULTS OF THE FIVE BEST BOTS 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%

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 (powerful ranged ground unit with clumsy movement) range and kiting micromanagement allowed it to hold off the early aggression of other bots such as UAlbertaBot, which came in second.

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 that were made and immediately attack the enemy base, using a unit counting heuristic to determine whether to retreat or keep pushing. Of note is that UAlbertaBot used an online planning algorithm to construct all of its economic build orders [12], as no hard-coded build orders were used.

AIUR also chose Protoss, with a strategy that was in between Skynet and UAlbertaBot in terms of attack timings. 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 has good probabilities to send the two or three first Zealots (basic contact attack ground unit) to harass the opponent;
- aggressive, where we have less chance to perform harassment with the first Zealots, and the first attack is usually a bit delayed with regard to the rush mood;

[14]http://eis.ucsc.edu/StarCraftAICompetition

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

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

- macro, where the AI does not try any early attacks and focuses a bit more on its economy before attacking;
- defense, where the AI "turtles" and waits to have a suitable army before running an attack;
- fast expand, where the first building constructs a base expansion, for a very economically 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 three finishers. Of the 30 rounds, Skynet beat UAlbertaBot 26 times, UAlbertaBot beat AIUR 29 times, and AIUR beat Skynet 19 times. Another notable point 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 seventh.

After the competition was over, a man versus 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, he 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-the-three match, Oriol chose his races randomly and ended up beating Skynet 2–0. In the first match, Oriol played Zerg versus 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 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 the second game, Oriol 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 midgame strategy, in game one, once its early pressure failed, and in game two, it made a key blunder in unit targeting which cost it the game. Humans were still in command.

The University of Alberta also hosted the 2012 competition, 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. Six of the ten entrants used this feature to aid in strategy selection, including the top four 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 second and third, 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 upper confidence bound (UCB) algorithm [68], 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 as UAlbertaBot. Note 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 Fig. 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.

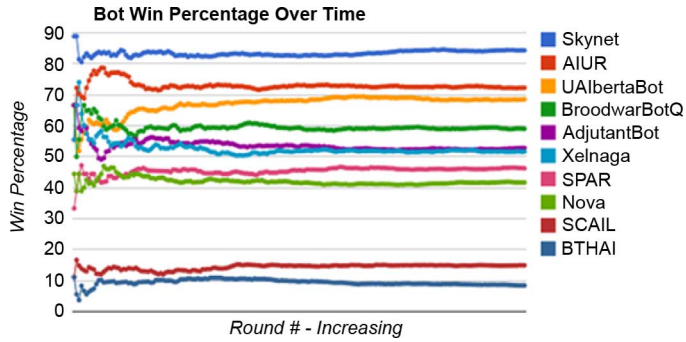

Fig. 4. Evolution of the win percentage of each bot participating in the AIIDE 2012 competition.

TABLE III
RESULTS OF THE FIVE BEST BOTS 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%

The 2012 man versus machine match again used the winner 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 workers chased them in vain. After killing off several probes and buying enough time to set up his expansion, he cleaned up Skynet's army with

TABLE IV
RESULTS OF THE FIRST ROUND AT THE 2011 CIG, HELD IN TWO BRACKETS. QUALIFIED FOR THE FINAL ROUND: UALBERTABOT AND SKYNET (FROM BRACKET A) AND XELNAGA AND BROODWARBOTQ (FROM BRACKET 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%

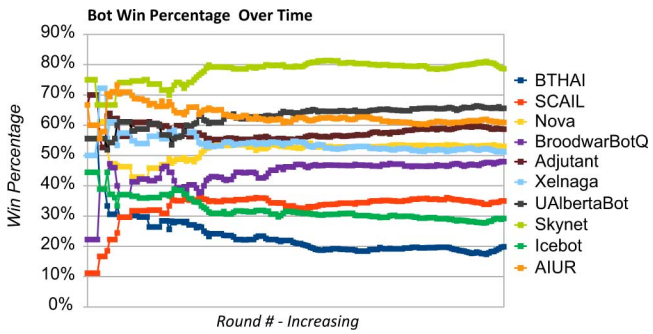

Fig. 5. Evolution of the win percentage of each bot participating in the 2012 CIG competition.

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. In the first game, a human player in Skynet's position 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 to be some of 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 at http://www.StarCraftAICompetition.com.

B. CIG

An initial attempt to run a *StarCraft* tournament at the 2010 CIG Conference 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 BWTA provided with the BWAPI interface crash so frequently that it would have been unjustifiable to announce a winner.

At the 2011 CIG Conference, the tournament was therefore run with a (secret) selection of maps used in a league play, which can be 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 ten bots. In addition to the ones discussed in previous sections (UAlbertaBot, Skynet, AIUR, Nova, BroodwarBotQ, and BTHAI), the set also contained LSAI, Xelnaga, Protoss Beast Jelly, and EvoBot; these are briefly described in the following.

1) Lsai (Zerg): It 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 then macro up to a larger attack force and maintain map control.

2) Xelnaga (Protoss): It 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.

3) Protoss Beast Jelly (Protoss): It always goes for a five-gate zealot rush, supported by an effective harvesting strategy named power-mining (two 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.

4) Evobot (Terran): It 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.

5) 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 five bots, a round-robin tournament was held with ten repetitions per pairing, resulting in 40 games per bot. The five maps chosen for the first round were selected from the pool of well-known league play maps found on the Internet: (2) MatchPoint 1.3, (4) Fighting Spirit 1.3, iCCup Destination 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 and Skynet from bracket A and Xelnaga and BroodwarBotQ from bracket B. All qualified bots play the Protoss faction.

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

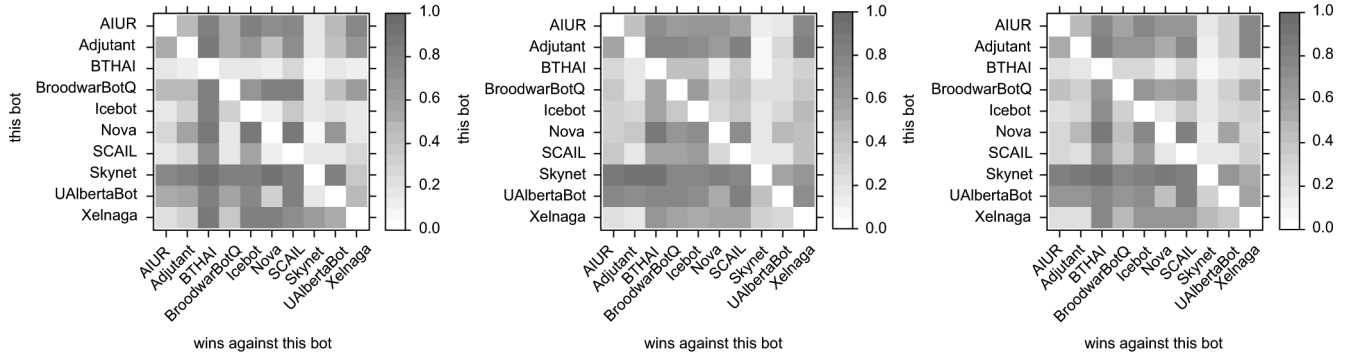

Fig. 6. Win percentages of the 2012 CIG competition, from left to right: three-player maps only, six-player maps only, and all maps. Read from line to column, bot in row wins, given fraction of games against bot in column. For some bots, we find interesting differences, e.g., Xelnaga gets worse on six-player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

TABLE V
RESULTS OF THE 2011 CIG COMPETITION

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%

TABLE VI
RESULTS OF THE 2012 CIG COMPETITION

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

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 min and assigned victory to the bot that had obtained more points as indicated on the end game screen.

6) Final Round: The final round was played in a similar mode to each of the first round brackets, using another set of five 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 the winner and UAlbertaBot as the runner-up, being almost equally strong, and the two other bots as clearly inferior.[18]

For the 2012 CIG, the AIIDE tournament software was employed, leading to a total of 4050 games played in 90 rounds of round robin. As six 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 from 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 three-player maps: *Athena-II*, *Neo Moon Glaive*, *Tears of the Moon*, and three six-player maps: *Legacy*, *River of Light*, and *The Huntress 1.1*. We will note that some bots consistently crashed on one of the originally considered maps, which has since been replaced. This is surprising as all maps are well-known league play maps or have been provided with the *StarCraft: Brood War* distribution itself.[19]

The overall results are displayed in Table VI, and the win rate evolution over time is shown in Fig. 5. These are quite consistent with the results of the 2012 AIIDE competition, so that we can conclude that the best bots are not very dependent on knowing the maps beforehand. However, the bot versus bot win rates, as displayed in Fig. 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 in the previously described competitions, there are also bots who consistently beat top ranked bots but have severe problems against lower ranked bots. For example, Xelnaga is especially strong against Skynet on the three-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 in luring the probes into its own base, where they get killed, leading to severe resource problems for UAlbertaBot. However, we cannot tell how often this happens as this would require reviewing every single replay between the two 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 (such as Xelnaga with Skynet) leads to a very restricted strategy that often leads to failure if playing against different bots. Note that in the direct

[18]The competition setup, documentation, and results can be found at http://ls11-www.cs.tu-dortmund.de/rts-competition/StarCraft-cig2011.

[19]Setup, replays, and results for the 2012 CIG competition can be found at http://ls11-www.cs.tu-dortmund.de/rts-competition/StarCraft-cig2012.

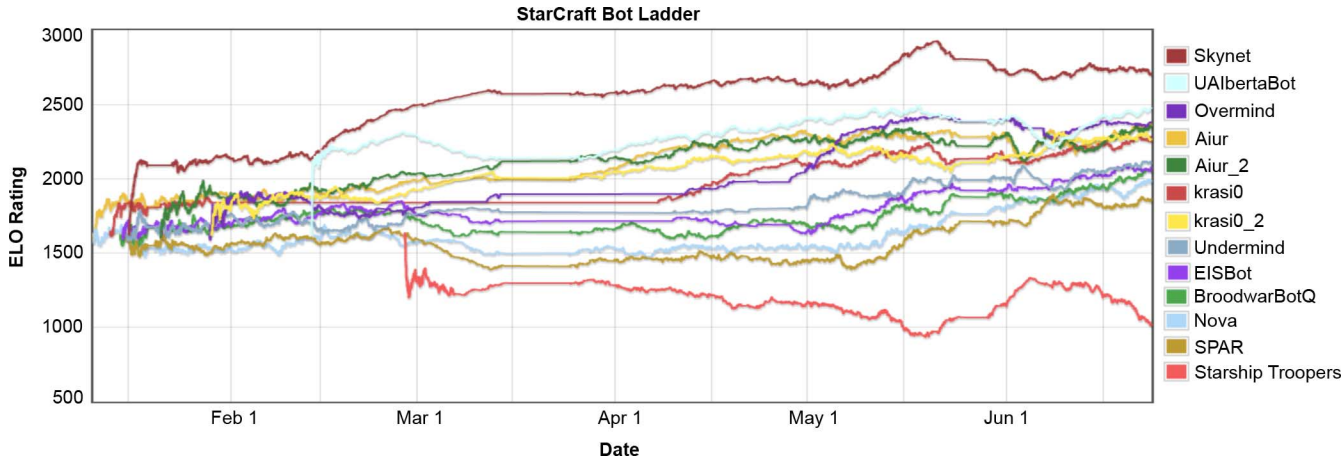

Fig. 7. Bot's Elo rating from February 1, 2012 to June 20, 2012.

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 2010 AIIDE winner (Overmind) did so. Presumably, implementing a good Zerg strategy is more demanding than implementing one for the Protoss or Terran races. Many bots consistently crashed when playing against a 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 bot entered at times 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 with 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. This kind of rating system is widely used in games like *Chess*. This bot ladder compiles different versions of bots from the main worldwide competitions (such as 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.

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

[21]http://sscaitournament.com

Fig. 7 shows the Elo rating of the bots in the ladder during the first half of 2012. The ranking is practically equal to the one at the 2011 AIIDE competition, showing the lack of adaptability of the current bots. We can see these better in Fig. 8 for the second half of 2012. During this period, new bots were introduced into the ladder. We can observe how the first version of KillerBot made a huge impact on Skynet's 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 are unable to adapt and the ranking does not change so much). 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 the RTS game AI is how to achieve 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 synthesizing a good plan from scratch to counter the opponent's strategy. In this section, we analyze this claim quantitatively using the tools of [14]. We analyze the replays from the 2011 and 2012 AIIDE competitions (shown in Figs. 9 and 10, respectively). We analyze 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 that it is necessary to see what the opponent is doing in order to determine the best opening, we expect the 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 identify the most common openings in all the replays from the 2011 and 2012 competitions (each one shown with a different color in the figures). No data are shown for the ItayUnvermind bot, since its opening does not match significantly with any of the ones used in our study (extracted from humans pro-gamers).

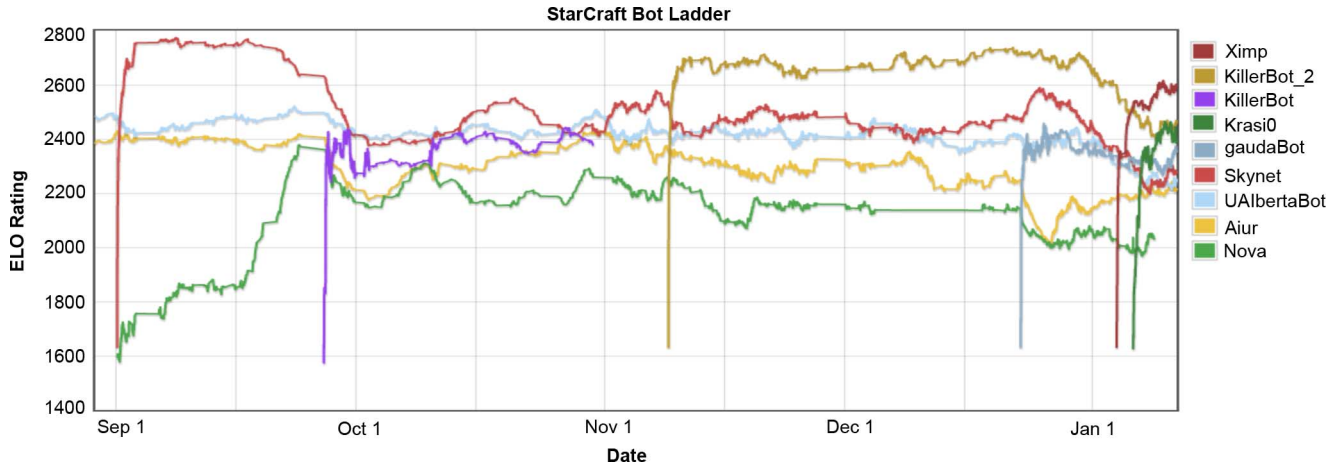

Fig. 8. Bot's Elo rating from September 1, 2012 to January 10, 2013.

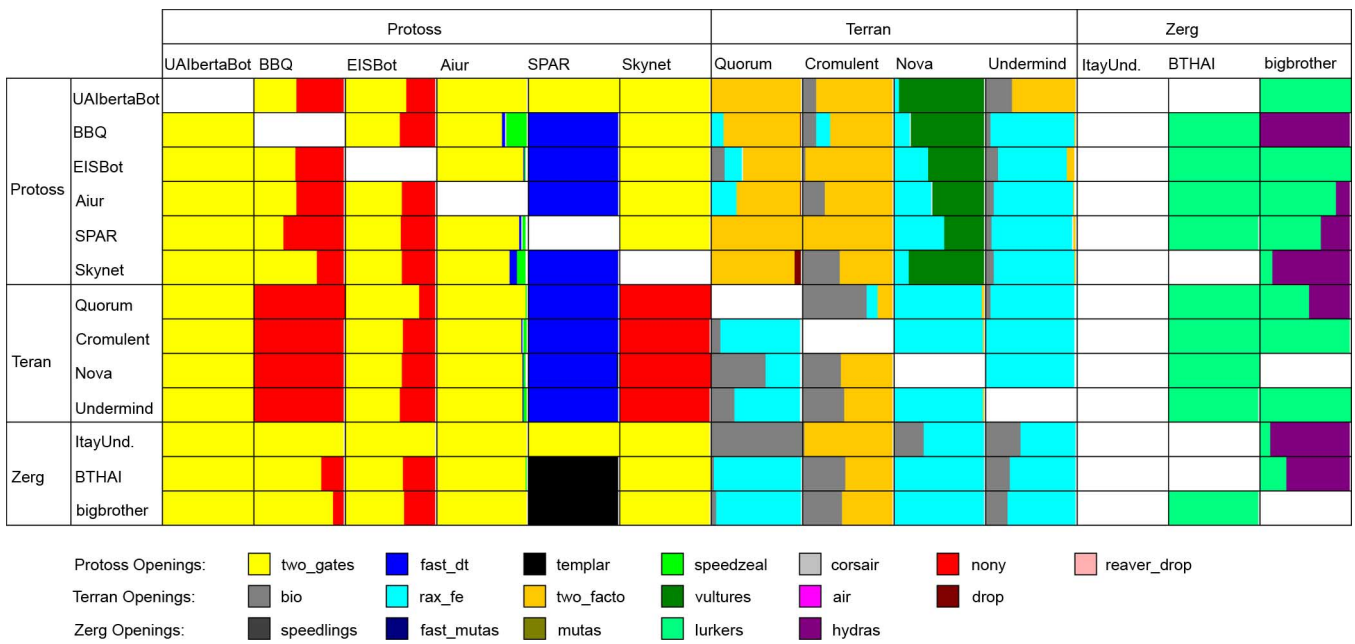

Fig. 9. Distribution of different openings performed by the different bots participating in the 2011 AIIDE competition. For each bot matchup, the colors show the proportion of times that the column bot used a particular opening against the row bot.

Specifically, we identify the following openings.[22]

- Protoss openings.
  - — Two_gates: Build two gateways and keep training zealots (basic contact attack ground unit); this is the quickest way to apply pressure.
  - — Fast_dt: Produce dark templars (technologically advanced stealth ground unit) as soon as possible, sacrificing early game power for a technological advance, hard-countered by detectors technology.
  - — Templar: Train high templars (technologically advanced zone attack unit) as fast as possible, the same as above; it is less deadly, but less easily countered.
  - — Speedzeal: Train zealots and research attack and speed upgrades as soon as possible; some early game power transitioning into late game tech.
  - — Corsair: Produce corsairs (air–air flying unit) as soon as possible and then transition into training dark templars (safe from Zerg's flying detectors thanks to corsairs), reavers (ground artillery unit), or dragoons. Weak early game.
  - — Nony: Build three gateways and massive training of dragoons. Slower than two_gates but still some early game (ranged) power.
  - — Reaver_drop: Train reavers as soon as possible to be able to do drops (air transport of artillery units).
- Terran openings.
  - — Bio: Produce a large army of marines (basic ranged ground unit) and medics (can heal biological units). Quickest way to apply pressure.
  - — Rax_fe: Take the closest "natural expansion" as soon as possible. This provides a big economic boost in the midgame by sacrificing some early game power.

[22]For a better comprehension of strategies, buildings, or units of *StarCraft*, we refer the reader to Teamliquid's wiki: http://wiki.teamliquid.net/starcraft/Category:Strategies.

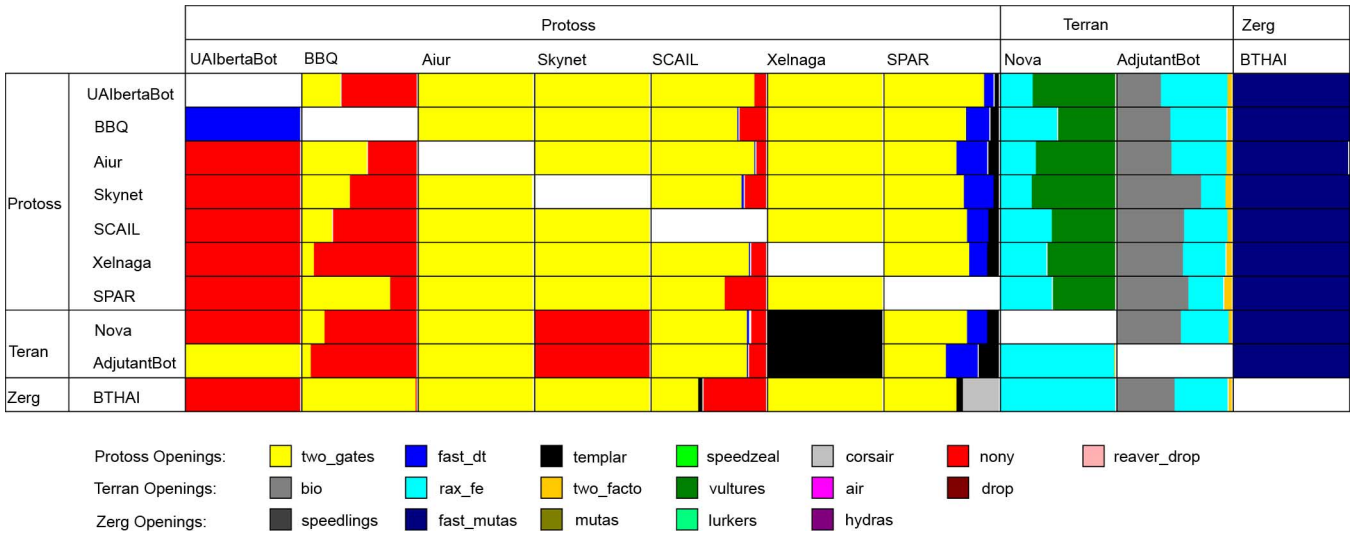

Fig. 10. Distribution of different openings performed by the different bots participating in the 2012 AIIDE competition. For each bot matchup, the colors show the proportion of times that the column bot used a particular opening against the row bot.

  - Two_facto: Build two factories and keep producing tanks (ground artillery unit). Vulnerable while building up to it and then very powerful on ground.
  - Vultures: Produce mainly vultures (fast ground ranged unit, excels against small units) and research mines. Quicker to reach (technologically) and build than tanks, can transition into tanks.
  - Air: Produce wraiths (ranged flying units) as soon as possible for an air attack. Vulnerable to anti-air openings or quick rushes.
  - Drop: Train dropships (flying transports) as soon as possible to be able to perform (mostly tanks or marines) drops, leveraging efficient tactics.
- Zerg openings.
  - Speedlings: Train zerlings (basic cheap, fast, ground contact attack unit) and research speed upgrade as soon as possible. Quickest way to apply pressure.
  - Fast_mutas: Produce mainly mutalisks (ranged flying units). Vulnerable in the early game while gathering gas and researching the technology.
  - Mutas: Expand two times for a stronger economy before massive training of mutalisks. Slower but more powerful buildup than above.
  - Lurkers: Train lurkers (ground, stealth artillery unit) as soon as possible to benefit from their (advanced technology) zone attack and cloak ability.
  - Hydras: Massive production of hydralisks (ground ranged unit). Much quicker to reach technologically than lurkers and can transition into them.

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 handcrafted and nonadaptive behaviors, than on online decision-making procedures. This is so, because 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 the RTS game AI that could be considered mostly solved, or for which we have very good solutions. One example of such problems is pathfinding (mostly solved) or low-scale micromanagement (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 to reason 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 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* [69], in the context of game-tree search (by learning the evaluation function). But it is 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 [64], 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 , 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* [70], or *Poker* [71]. 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 [72]), they cannot scale up to the size of problems needed for RTS games like *StarCraft*.
- Integration (multiscale 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 into 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.* [73].

VII. Conclusion

As the list in the previous section indicates, RTS games are an excellent testbed for AI techniques, which create a very long 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 counterstrategy for. Additionally, good human players are still clearly superior to the best computer programs. From an industry point of view, one additional challenge is to make bots more believable to play against, and thus, more fun for human players (this includes, for example, doing scouting, instead of cheating and having full information of the game state).

One of the main goals of this paper is to provide a centralized and unified overview of the research being done in the area of the 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 toward addressing these challenges with a focus on *StarCraft* (which has emerged as a unified testbed). Given that playing an RTS game is a very challenging task, researchers tend to divide such a task into smaller tasks, which 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.

RTS games encompass many interesting and complex subproblems that are closely related not only to other fields of AI research, but to real-world problems as well. For example, optimizing assembly line operations in factories is akin to performing build-order optimizations. Troop positioning in military conflicts involves the same spatial and tactical reasoning used in RTS games. Robot navigation in unknown environments requires real-time pathfinding and decision making to avoid hitting obstacles. All of these issues mentioned in this paper must be tackled by the RTS game AI community, and in doing so, we will not only be improving techniques for writing tournament-winning bots, but also advancing the state of the art for many other fields as well.

References

[1] M. Buro, "Real-time strategy games: A new AI research challenge," in *Proc. Int. Joint Conf. Artif. Intell.*, 2003, pp. 1534–1535.

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

[3] G. Synnaeve, "Bayesian programming and learning for multi-player video games," Ph.D. dissertation, Dept. Comput. Sci. Math., Université de Grenoble, Grenoble, France, 2012.

[4] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *Proc. IEEE Symp. Comput. Intell. Games*, 2009, pp. 140–147.

[5] G. Synnaeve and P. Bessière, "A dataset for StarCraft AI & an example of armies clustering," in *Proc. AIIDE Workshop AI Adversarial Real-Time Games*, 2012, pp. 25–30.

[6] B. G. Weber, M. Mateas, and A. Jhala, "Building human-level AI for real-time strategy games," in *Proc. AIIDE Fall Symp. Adv. Cogn. Syst.*, Palo Alto, CA, USA, 2011, pp. 329–336.

[7] C. E. Miles, *Co-Evolving Real-Time Strategy Game Players*. Ann Arbor, MI, USA: ProQuest, 2007.

[8] R. Houlette and D. Fu, "The ultimate guide to FSMs in games," *AI Game Programming Wisdom*, vol. 2, pp. 283–302, 2003.

[9] 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. Berlin, Germany: Springer-Verlag, 2008, vol. 226, pp. 293–310.

[10] K. Mishra, S. Ontañón, and A. Ram, "Situation assessment for plan retrieval in real-time strategy games," in *Advances in Case-Based Reasoning*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2008, vol. 5239, pp. 355–369.

[11] H. Hoang, S. Lee-Urban, and H. Muñoz-Avila, "Hierarchical plan representations for encoding strategic game AI," in *Proc. Artif. Intell. Interactive Digit. Entertain. Conf.*, 2005, pp. 63–68.

[12] D. Churchill and M. Buro, "Build order optimization in StarCraft," in *Proc. Artif. Intell. Interactive Digit. Entertain. Conf.*, 2011, pp. 14–19.

[13] E. Dereszynski, J. Hostetler, A. Fern, T. D. T.-T Hoang, and M. Udarbe, "Learning probabilistic behavior models in real-time strategy games," in *Proc. Artif. Intell. Digit. Entertain. Conf.*, 2011, pp. 20–25.

[14] G. Synnaeve and P. Bessière, "A Bayesian model for opening prediction in RTS games with application to StarCraft," in *Proc. IEEE Conf. Comput. Intell. Games*, Sep. 2011, pp. 281–288.

[15] G. Synnaeve and P. Bessière, "A Bayesian model for plan recognition in RTS games applied to StarCraft," in *Proc. 7th Artif. Intell. Interactive Digit. Entertain. Conf.*, Oct. 2011, pp. 79–84.

[16] J. Young and N. Hawes, "Evolutionary learning of goal priorities in a real-time strategy game," in *Proc. 8th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2012, pp. 87–92.

[17] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *Artif. Intell. Commun.*, vol. 7, no. 1, pp. 39–59, 1994.

[18] D. W. Aha, M. Molineaux, and M. J. V. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2005, vol. 3620, pp. 5–20.

[19] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2008, pp. 3106–3111.

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

[21] U. Jaidee, H. Muñoz-Avila, and D. W. Aha, "Case-based learning in goal-driven autonomy agents for real-time strategy combat tasks," in *Proc. ICCBR Workshop Comput. Games*, 2011, pp. 43–52.

[22] M. Čertický and M. Čertický, "Case-based reasoning for army compositions in real-time strategy games," in *Proc. Sci. Conf. Young Res.*, 2013, pp. 70–73.

[23] B. G. Weber, M. Mateas, and A. Jhala, "A particle model for state estimation in real-time strategy games," in *Proc. 7th AAAI Conf. Artif. Intell. Interactive Digit. Entertain.*, 2011, pp. 103–108.

[24] B. G. Weber, P. Mawhorter, M. Mateas, and A. Jhala, "Reactive planning idioms for multi-scale game AI," in *Proc. IEEE Symp. Comput. Intell. Games*, 2010, pp. 115–122.

[25] B. G. Weber, M. Mateas, and A. Jhala, "Applying goal-driven autonomy to StarCraft," in *Proc. 6th Artif. Intell. Interactive Digit. Entertain.*, 2010, pp. 101–106.

[26] D. C. Pottinger, "Terrain analysis for real-time strategy games," in *Proc. Game Develop. Conf.*, 2000.

[27] K. D. Forbus, J. V. Mahoney, and K. Dill, "How qualitative spatial reasoning can improve strategy game AIs," *IEEE Intell. Syst.*, vol. 17, no. 4, pp. 25–30, Jul. .

[28] D. H. Hale, G. M. Youngblood, and P. N. Dixit, "Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds," in *Proc. 4th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2008, pp. 173–178.

[29] L. Perkins, "Terrain analysis in real-time strategy games: An integrated approach to choke point detection and region decomposition," in *Proc. 6th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2010, pp. 168–173.

[30] M. Čertický, "Implementing a wall-in building placement in StarCraft with declarative programming," 2013.

[31] S. Hladky and V. Bulitko, "An evaluation of models for predicting opponent positions in first-person shooter video games," in *Proc. IEEE Symp. Comput. Intell. Games*, 2008, pp. 39–46.

[32] F. Kabanza, P. Bellefeuille, F. Bisson, A. R. Benaskeur, and H. Irandoust, "Opponent behaviour recognition for real-time strategy games," in *Proc. Workshops 24th AAAI Conf. Artif. Intell.*, 2010, pp. 29–36.

[33] C. W. Geib and R. P. Goldman, "A probabilistic plan recognition algorithm based on plan tree grammars," *Artif. Intell.*, vol. 173, pp. 1101–1132, Jul. 2009.

[34] 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 *Proc. Int. Joint Conf. Artif. Intell.*, 2007, pp. 1041–1046.

[35] P. Cadena and L. Garrido, "Fuzzy case-based reasoning for managing strategic and tactical reasoning in StarCraft," in *Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, I. Z. Batyrshin and G. Sidorov, Eds. Berlin, Germany: Springer-Verlag, 2011, vol. 7094, pp. 113–124.

[36] G. Synnaeve and P. Bessière, "Special tactics: A Bayesian approach to tactical decision-making," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 409–416.

[37] C. Miles and S. J. Louis, "Co-evolving real-time strategy game playing influence map trees with genetic algorithms," in *Proc. Int. Congr. Evol. Comput.*, 2006, pp. 88–95.

[38] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for RTS game combat scenarios," in *Proc. 8th AAAI Conf. Artif. Intell. Interactive Digit. Entertain.*, 2012, pp. 112–117.

[39] M. Chung, M. Buro, and J. Schaeffer, "Monte Carlo planning in RTS games," in *Proc. IEEE Symp. Comput. Intell. Games*, 2005, pp. 117–124.

[40] R.-K. Balla and A. Fern, "UCT for tactical assault planning in real-time strategy games," in *Proc. Int. Joint Conf. Artif. Intell.*, 2009, 2012, pp. 40–45.

[41] A. Uriarte and S. Ontañón, "Kiting in RTS games using influence maps," in *Proc. AI Adversarial Real-Time Games Workshop at AIIDE 2012*, 2012, pp. 31–36.

[42] 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, Jan. 2009.

[43] J. Hagelbäck, "Potential-field based navigation in StarCraft," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 388–393.

[44] J. Hagelbäck and S. J. Johansson, "Dealing with fog of war in a real time strategy game environment," in *Proc. IEEE Symp. Comput. Intel. Games*, 2008, pp. 55–62.

[45] P. Avery, S. Louis, and B. Avery, "Evolving coordinated spatial tactics for autonomous entities using influence maps," in *Proc. 5th Int. Conf. Comput. Intell. Games*, 2009, pp. 341–348.

[46] G. Smith, P. Avery, R. Houmanfar, and S. Louis, "Using co-evolved RTS opponents to teach spatial tactics," in *Proc. IEEE Symp. Comput. Intel. Games*, 2010, pp. 146–153.

[47] H. Danielsiek, R. Stuer, A. Thom, N. Beume, B. Naujoks, and M. Preuss, "Intelligent moving of groups in real-time strategy games," in *Proc. IEEE Symp. Comput. Intell. Games*, 2008, 2010, pp. 71–78.

[48] L. Liu and L. Li, "Regional cooperative multi-agent q-learning based on potential field," in *Proc. 4th Int. Conf. Neural Comput.*, 2008, pp. 535–539.

[49] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Piatkowski, R. Stüer, A. Thom, and S. Wessing, "Towards intelligent team composition and maneuvering in real-time strategy games," *Trans. Comput. Intell. AI Games*, vol. 2, no. 2, pp. 82–98, Jun. 2010.

[50] G. Synnaeve and P. Bessière, "A Bayesian model for RTS units control applied to StarCraft," in *Proc. IEEE Comput. Intell. Games*, Sep. 2011, pp. 190–196.

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

[52] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:broodwar," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 402–408.

[53] B. Marthi, S. Russell, D. Latham, and C. Guestrin, "Concurrent hierarchical reinforcement learning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2005, pp. 779–785.

[54] C. Madeira, V. Corruble, and G. Ramalho, "Designing a reinforcement learning-based adaptive AI for large-scale strategy games," in *Proc. AI Interactive Digit. Entertain. Conf.*, 2006, pp. 121–123.

[55] U. Jaidee and H. Muñoz-Avila, "CLASSQ-L: A q-learning algorithm for adversarial real-time strategy games," in *Proc. AI Adversarial Real-Time Games Workshop at AIIDE 2012*, 2012, pp. 8–13.

[56] M. Ponsen and I. P. H. M. Spronck, "Improving adaptive game AI with evolutionary learning," Univ. Wolverhampton, West Midlands, U.K., 2004, pp. 389–396.

[57] N. Othman, J. Decraene, W. Cai, N. Hu, and A. Gouaillard, "Simulation-based optimization of StarCraft tactical AI through evolutionary computation," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 394–401.

[58] S. Wintermute, J. Z. X. Joseph, and J. E. Laird, "Sorts: A human-level approach to real-time strategy AI," in *Proc. AI Interactive Digit. Entertain. Conf.*, 2007, 2012, pp. 55–60.

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

[60] D. Demyen and M. Buro, "Efficient triangulation-based pathfinding," in *Proc. 21st Nat. Conf. Artif. Intell.*, 2006, vol. 1, pp. 942–947.

[61] C. W. Reynolds, "Steering behaviors for autonomous characters," in *Proc. Game Develop. Conf.*, 1999, pp. 763–782.

[62] A. Treuille, S. Cooper, and Z. Popovic, "Continuum crowds," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1160–1168, 2006.

[63] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 144–148, Jun. 2012.

[64] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "On-line case-based planning," *Comput. Intell.*, vol. 26, no. 1, pp. 84–119, 2010.
[65] S. Ontañón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Proc. 9th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2013, pp. 58–64.
[66] M. Molineaux, D. W. Aha, and P. Moore, "Learning continuous action models in a real-time strategy environment," in *Proc. FLAIRS Conf.*, 2008, 2013, pp. 257–262.
[67] J. Young, F. Smith, C. Atkinson, K. Poyner, and T. Chothia, "SCAIL: An integrated StarCraft AI system," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 438–445.
[68] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.
[69] G. Tesauro, "Comparison training of chess evaluation functions," in *Machines That Learn to Play Games*. Hauppauge, NY, USA: Nova Science, 2001, pp. 117–130.
[70] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Comput.*, vol. 6, no. 2, pp. 215–219, 1994.
[71] J. Rubin and I. Watson, "Computer poker: A review," *Artif. Intell.*, vol. 175, no. 5, pp. 958–987, 2011.
[72] I. Refanidis and I. Vlahavas, "Heuristic planning with resources," in *Proc. Eur. Conf. Artif. Intell.*, 2000, pp. 521–525.
[73] S. Branavan, D. Silver, and R. Barzilay, "Learning to win by reading manuals in a Monte-Carlo framework," in *Proc. Annu. Meeting Assoc. Comput. Linguist.*, 2011, pp. 268–277.

Santiago Ontañón received the Ph.D. degree in artificial intelligence from the Autonomous University of Barcelona (UAB), Barcelona, Spain, in 2005.

He is an Assistant Professor in the Computer Science Department, Drexel University, Philadelphia, PA, USA. His main research interests are game AI, case-based reasoning, and machine learning, fields in which he has published more than 100 peer-reviewed papers. Before joining Drexel University, he held postdoctoral research positions at the Artificial Intelligence Research Institute (IIIA), Barcelona, Spain, at the Georgia Institute of Technology (GeorgiaTech), Atlanta, GA, USA, and at the University of Barcelona, Barcelona, Spain.

Gabriel Synnaeve received the Ph.D. degree in computer science and mathematics from Grenoble University, Grenoble, France, in 2012.

He is a Postdoctoral Researcher at the Laboratory of Cognitive Science and Psycholinguistics (LSCP), Ecole Normale Supérieure (ENS), Paris, France. His current research interests are deep learning and Bayesian nonparametrics to model language acquisition from unsupervised speech and textual data. Previously, he worked on Bayesian modeling in game AI research at INRIA, Grenoble, France and Collège de France, Paris, France.

Alberto Uriarte received the B.S. degree in computer science and the M.S. degree in computer vision and artificial intelligence from the Autonomous University of Barcelona (UAB), Barcelona, Spain, in 2006 and 2011, respectively. He is currently working toward the Ph.D. degree in computer science at Drexel University, Philadelphia, PA, USA.

His research interest includes game AI, real-time strategy (RTS) games, multiagents systems, procedural content generation, computational geometry, machine learning, and drama management.

Florian Richoux received the Ph.D. degree in theoretical computer science from the École Polytechnique, Palaiseau, France, in 2009

He is an Associate Professor at the Laboratory of Computer Science, University of Nantes, Nantes, France. His main research field concerns parallel algorithms for solving constraint-based problems, and he has strong interests in AI, in particular, game AI and machine learning. He was a CNRS Research Fellow at the Japanese–French Laboratory for Informatics, University of Tokyo, Tokyo, Japan, for three years.

David Churchill received the M.Sc. degree in computer science in the field of autonomous robotics from the Memorial University of Newfoundland, St John's, NL, Canada. He is currently working toward the Ph.D. degree in the Computing Science Department, University of Alberta, Edmonton, AB, Canada.

His main research interests are in real-time strategy (RTS) video game AI, specifically in real-time heuristic search techniques for *StarCraft*. He is the author of *UAlbertaBot*

Mr. Churchill has been the co-organizer and facilitator of the AIIDE *StarCraft* AI Competition since 2011.

Mike Preuss received the Diploma degree from the Technical University of Dortmund, Dortmund, Germany, in 1998 and the Ph.D. degree in computer science from the Technische Universität Dortmund, Dortmund, Germany, in 2913.

He is a Research Associate at the European Research Center for Information Systems (ERCIS), Münster University, Münster, Germany. He is interested in multimodal and multiobjective niching for evolutionary algorithms on real-valued problems, and in applying this methods to various engineering domains and computer games. In the computational intelligence in games (CIG) area, he is especially active in real-time strategy (RTS) games and procedural content generation (PCG).

Mr. Preuss was involved in founding the EvoGames track at Evo* and the Digital Entertainment Technologies and Arts (DETA) track at GECCO and organized the *StarCraft* competitions at the Computational Intelligence and Games (CIG) Conference from 2011 to 2013.