RTS AI: Problems and Techniques

Santiago Ontañón[1], Gabriel Synnaeve[2], Alberto Uriarte[1], Florian Richoux[3], David Churchill[4], and Mike Preuss[5]

[1] Computer Science Department at Drexel University, Philadelphia, PA, USA.
{santi,albertouri}@cs.drexel.edu
[2] Cognitive Science and Psycholinguistics (LSCP) of ENS Ulm, Paris, France.
gabriel.synnaeve@gmail.com
[3] Nantes Atlantic Computer Science Laboratory (LINA), Univ. Nantes, France.
florian.richoux@univ-nantes.fr
[4] Computing Science Department of the University of Alberta, Edmonton, Canada.
cdavid@cs.ualberta.ca
[5] Department of Computer Science of Technische Universität Dortmund, Germany.
mike.preuss@cs.tu-dortmund.de

Synonyms

Real-Time Strategy games; RTS games; Artificial Intelligence; AI; Game AI

Definition

Real-Time Strategy (RTS) games is a sub-genre of strategy games where players need to build an economy (gathering resources and building a base) and military power (training units and researching technologies) in order to defeat their opponents (destroying their army and base). Artificial Intelligence problems related to RTS games deal with the behavior of an artificial player. This consists among others to learn how to play, to have an understanding about the game and its environment, to predict and infer game situations from a context and sparse information.

Introduction

The field of Real-Time Strategy (RTS) game Artificial Intelligence (AI) has advanced significantly in the past few years, partially thanks to competitions as the "ORTS RTS Game AI Competition" (held from 2006 to 2009), the "AIIDE StarCraft AI Competition" (held since 2010), and the "CIG StarCraft RTS AI Competition" (held since 2011). Based on the work work presented in [39], here we first define RTS games, then list the open problems in creating AI for RTS games, and finally point to the approaches that have been proposed to address these problems.

Real-Time Strategy Games

From a theoretical point of view, the main differences between RTS games and traditional board games such as Chess are:

- RTS games are *simultaneous move* games, where more than one player can issue actions at the same time.
- Action in RTS games are *durative*, i.e. actions are not instantaneous, but take some amount of time to complete.
- RTS games are "real-time", which actually means is that each player has a very small amount of time to decide the next move, and that, in contrast to *turn-based* games, the game keeps advancing even if a player does not execute any actions. Compared to Chess, where players may have several minutes to decide the next action, in StarCraft, a popular RTS game, the game executes at 24 frames per second, which means that players can act as fast as every 42ms, before the game state changes.
- Most RTS games are partially observable: players can only see the part of the map that has been explored. This is referred to as the *fog-of-war*.
- Most RTS games are non-deterministic: some actions have a chance of success, and the amount of damage dealt by different units is sometimes stochastic.
- And finally, the complexity of these games, both in terms of state space size and in terms of number of actions available at each decision cycle is very large. For example, the state space of Chess is typically estimated to be around 10^{50}, heads up no-limit Texas holdem poker around 10^{80}, and Go around 10^{170}. In comparison, the state space of StarCraft in a typical map is estimated to be many orders of magnitude larger than any of those, as discussed in the next section.

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 [6]. For those reasons, a large spectrum of techniques have been attempted to deal with this domain, as we will describe below.

In the past few years, *StarCraft: Brood War* (an immensely popular RTS game released in 1998 by Blizzard Entertainment) has become the standard testbed for evaluating AI techniques in RTS games. StarCraft is set in a science-fiction based universe where the player must choose one of the three races: Terran, Protoss or Zerg. In order to win a StarCraft game, players must first gather resources (minerals and Vespene gas). As resources become available, players need to allocate them for creating more buildings (which reinforce the economy, and allow players to create units or unlock stronger units), research new technologies (in order to use new unit abilities or improve the units) and train attack units. Units must be distributed to accomplish different tasks such as reconnaissance, defense and attack. While performing all of those tasks, players also need

to strategically understand the geometry of the map at hand, in order to decide where to place new buildings (concentrate in a single area, or expand to different areas) or where to set defensive outposts. Finally, when offensive units of two players meet, each player must quickly maneuver each of the units in order to fight a battle, which requires quick and reactive control of each of the units.

From a theoretical point of view, the state space of a StarCraft game for a given map is enormous. For example, consider a typical 128×128 map. At any given moment there might be between 50 to 400 units in the map, each of which might have a complex internal state (remaining energy and hit-points, action being executed, etc.). This quickly leads to an immense number of possible states (way beyond the size of smaller games, such as Chess or Go). For example, just considering the location of each unit (with 128×128 possible positions per unit), and 400 units, gives us an initial number of $16384^{400} \approx 10^{1685}$. If we add the other factors playing a role in the game, such as resources, hit-points, energy, research status, cool-down timers, etc., we obtain even larger numbers (see [39] for a more in-depth description of the complexity of StarCraft).

Challenges in RTS Game AI

Early research in AI for RTS games [5] identified the following six challenges: Resource management, Decision making under uncertainty, Spatial and temporal reasoning, Collaboration (between multiple AIs), Opponent modeling and learning, and Adversarial real-time planning. While there has been a significant work in many, others have been untouched (e.g. collaboration). Moreover, recent research in this area has identified several additional research challenges, such as how to exploit the massive amounts of existing domain knowledge (strategies, build-orders, replays, and so on). Thus, the challenges in RTS game AI can be grouped in six main different areas, described below.

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

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

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

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). However, scouting also means deliberately reducing economic progress in order to obtain information. 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.

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 defenses or walls 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, regarding its own position and opponent's bases. Finally, spatial reasoning is key to tactical reasoning: players need to decide where to place units for battle, favoring, for instance, engagements when the opponent's units are lead into a bottleneck.

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 about when to perform long-term impact economic actions such as upgrades, building construction, strategy switching, etc. all taking into account that the effects of these actions are not immediate, but longer term.

Domain Knowledge Exploitation In traditional board games such as Chess, researchers have exploited the large amounts of existing domain knowledge to

create good evaluation functions to be used by alpha-beta search algorithms, extensive opening books, or end-game tables. In the case of RTS games, it is still unclear how the significantly large amount of domain knowledge (in the forms or strategy guides, replays, etc.) can be exploited. Most work in this area has focused on two main directions: (1) hard-coding existing strategies into bots (so that bots only need to decide which strategies to deploy, instead of having to solve the complete planning problem), and (2) mining large datasets of replays [63, 55] to automatically extract strategies, trends or plans.

Task Decomposition Most existing approaches to play RTS games work by decomposing the problem into a collection of smaller problems to be solved independently. Specifically, a common subdivision is:

- *Strategy*: corresponds to the high-level decision making process. This is the highest level of abstraction for the game comprehension. Finding an efficient strategy or counter-strategy against a given opponent is key in RTS games, and concerns the whole set of units a player owns.
- *Tactics*: 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*: is the implementation of tactics. This 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*: consists in the analysis of regions composing the map: choke-points, minerals and gas emplacements, low and high walkable grounds, islands, etc.
- *Intelligence gathering*: corresponds to information collected about the opponent. Because of the fog-of-war, players must regularly send scouts to localize and spy enemy bases.

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

- *Micro*: 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*: 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 RTS game AI in [65]. Although the previous task decomposition is common, other task decompositions have been explored (see [39] for an overview of the task decomposition used by several StarCraft bots).

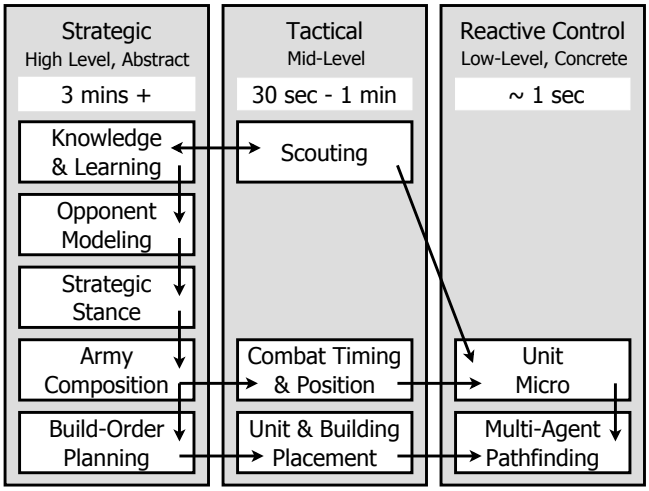

Fig. 1. RTS AI levels of abstraction and typical sub-problems associated with them: Timings correspond to an estimate of the duration of a behavior switch in StarCraft.

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 corresponds to "micro").

Figure 1 illustrates how strategy, tactics, and reactive control are three points in a continuum scale where strategy corresponds to decisions making processes that affect long spans of time (several minutes in the case of StarCraft), reactive control corresponds to low-level second-by-second decisions, and tactics sit in the middle. Also, strategic decisions reason about the whole game at once, whereas tactical or reactive control decisions are localized, and affect only specific groups of units. Typically, strategic decisions 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.

The remainder of this section presents work toward addressing the previous six open RTS game AI problems, grouped as work focused toward strategy, tactics, or reactive control, as well as a final section dedicated to holistic approaches that attempt to deal with all three levels at once.

Strategy

In the context of RTS games, high-level strategic reasoning has been addressed using many AI techniques, like hard-coded approaches, planning, or machine learning. We cover each of these approaches in turn.

Hard-coded approaches They have been extensively used in commercial RTS games. The most common ones use finite state machines (FSM) [24] 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, 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.

Planning Approaches using planning techniques have also been explored in the literature. For example Ontañón et al. [38] 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 run-time in order to form full-fledged strategies to play the game. In [34] 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 [23]. 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 case-based planning being the only ones explored so far. Moreover, none of these approaches addresses any timing or scheduling issues, which are key in RTS games. On notable exception is the work of Churchill and Buro [9], who used planning in order to construct its economic build-orders, taking into account timing constraints of the different actions.

Machine Learning Concerning machine learning-based approaches, Weber and Mateas [63] proposed a data mining approach to strategy prediction and performed supervised learning on labeled StarCraft replays. Dereszynski et al. [15] used Hidden Markov Models (HMM) to learn the transition probabilities of sequences of building construction orders and kept the most probable ones to produce probabilistic behavior models (in StarCraft). Synnaeve and Bessière [52] used the dataset of [63] and presented a Bayesian semi-supervised model to learn from replays and predict openings (early game strategies) from StarCraft replays. The openings are labeled by EM clustering considering appropriate features. Then, in [53], 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 [70], showing improvement over static priorities.

Case-Based Reasoning Also falling into the machine-learning category, a significant group of researchers has explored case-based reasoning (CBR) [1] approaches for strategic decision making. For example Aha et al. [2] used CBR to perform dynamic plan retrieval in the Wargus domain. Hsieh and Sun [25] based their work on Aha et al.'s CBR model [2] and used StarCraft replays to construct states and building sequences ("build orders"). Schadd et al. [47] 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. [27] study the use of CBR for automatic goal selection, while playing an RTS game. These goals will then determine which Q-tables to be used in a reinforcement learning framework. Finally, Čertický et al. [61] used CBR to build their army, based on the opponent's army composition, and they pointed out on the importance of proper scouting for better results.

Scouting One final consideration concerning strategy is that RTS games are typically partially observable. Games like StarCraft implement the "fog-of-war" idea, which basically means that a player can only see the areas of the map close to her own units. Areas of the map away from the field of view of individual units are not observable. Players need to scout in order to obtain information about the opponent's strategy. The size of the state space in StarCraft prevents solutions based on POMDPs from being directly applicable, and very few of the previous approaches deal with this problem. 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 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. [66], 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 [67, 64], finding the more relevant goal(s) to spawn in unforeseen situations.

Tactics

We will divide the work on mid-range tactical reasoning in RTS games in two large groups: spatial reasoning and decision making (that has been addressed both using machine learning and game tree search).

Spatial Reasoning The most common form of spatial reasoning in the literature of RTS games is terrain analysis. Terrain analysis supplies the AI with structured information about the map. This analysis is usually performed offline, in order to save CPU time during the game. For example, Pottinger [43] 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. [16] showed the importance to have qualitative spatial information for wargames, for which they used geometric and pathfinding analysis. Hale et al. [21] presented a 2D geometric navigation mesh generation method from expanding convex regions from seeds. Finally, Perkins [41] applied Voronoi decomposition (then pruning) to detect regions and relevant choke points in RTS maps. This approach is implemented for StarCraft in the BWTA[6] library, used by most state of the art StarCraft bots.

Another form of spatial reasoning that has been studied in RTS games is *walling*. 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 addressed this constraint satisfaction problem using Answer Set Programming (ASP) [62]. Richoux et al. [46] presented an alternative approach based on constraint programming and local search, designed to be run-time.

Machine Learning Concerning tactical decision making, many different approaches have been explored such as machine learning or game tree search. Hladky and Bulitko [22] benchmarked hidden semi-Markov models (HSMM) 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 HSMM. Kabanza et al. [28] improve the probabilistic hostile agent task tracker (PHATT [17], a simulated HMM for plan recognition) by encoding strategies as HTN, used for plan and intent recognition to find tactical opportunities. Sharma et al. [48] combined CBR and reinforcement learning to enable reuse of tactical plan components. Cadena and Garrido [7] used fuzzy CBR (fuzzy case matching) for strategic and tactical planning. [56] combined space abstraction into regions from [41] and tactical-decision making by assigning scores (economical, defenses, etc.) to regions and looking for their correspondences to tactical moves (attacks) in pro-gamers replays. Finally, Miles [33] 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 used evolutionary algorithms to learn IMTrees for each strategic decision in the game involving spatial reasoning by combining a set of basic influence maps.

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

Game tree search These techniques have also been explored for tactical decision making. Churchill and Buro [11] presented the ABCD algorithm (Alpha-Beta Considering Durations), a game tree search algorithm for tactical battles in RTS games. Chung et al. [8] applied Monte-Carlo planning to a capture-the-flag version of Open RTS. Balla and Fern [4] 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. Uriarte and Ontañón [60, 59] explored different game state abstractions in the context of Monte-Carlo Tree Search for high-level tactical reasoning in StarCraft. Other algorithms, such as Greedy Portfolio Search [10], perform abstraction at the level of actions, by employing a collection of predefined "scripts", and using these scripts as the possible actions that the players can execute in the context of game tree search.

Reactive Control

Reactive control has been addressed mainly via the application of potential fields or by using machine learning to learn good control policies. We also include work on path-finding as part of reactive control.

Potential fields 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 [58], or staying at maximum shooting distance [20]. Potential fields have also been combined with A* path-finding to avoid local traps [18]. Hagelbäck and Johansson [19] presented a multi-agent potential fields based bot able to deal with fog-of-war in the Tankbattle game. Avery et al. [3] and Smith et al. [49] co-evolved influence map trees for spatial reasoning in RTS games. Danielsiek et al. [13] used influence maps to achieve intelligent squad movement to flank the opponent in a RTS game. Despite their success, a drawback for potential field-based techniques is the large number of parameters that has to be tuned in order to achieve the desired behavior. Approaches for automatically learning such parameters have been explored, for example, using reinforcement learning [30], or self-organizing-maps (SOM) [44]. We would like to note that potential fields are a reactive control technique, and as such, they do not perform any form of lookahead. As a consequence, these techniques are prone to make units stuck in local optima.

Machine Learning 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 [54], which subsumes potential fields, allowing for integration of tactical goals directly in micro-management.

Additionally, there have been some interesting uses of reinforcement learning (RL) [51]: Wender and Watson [68] evaluated the different major RL algorithms for (decentralized) micro-management, which perform all equally. Marthi et al.

[32] 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. [31] advocate the use of prior domain knowledge to allow faster RL learning and applied their work on a turn-based strategy game. This is because the action space to explore is gigantic for real game setups. It requires exploiting the existing structure of the game in a partial program (or a partial Markov decision process) and a shape function (or a heuristic) [32]. Another approach has been proposed by Jaide and Muñoz-Avila [26] 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 [42] 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 micro-management control model, as shown by [40] which optimizes an AIIDE 2010 micro-management competition bot.

Finally, approaches based on game tree search are recently being explored for micro-management. Churchill et al. [12] 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. [69] have explored human-like attention models (with units grouping and vision of a unique screen location) for reactive control.

Pathfinding Finally, although pathfinding does not fall under our previous definition of reactive control, we include it in this section, since it is typically performed as a low-level service, not part of either tactical nor strategical reasoning (although there are some exceptions, like the tactical pathfinding of Danielsiek et al. [13]). 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 [29] exist, it is most common to use A* combined with a map simplification technique that generates a simpler navigation graph to be used for pathfinding. An example of such technique is Triangulation Reduction A*, that computes polygonal triangulations on a grid-based map [14]. Considering movement for groups of units, rather then individual units, techniques such as steering of flocking behaviors [45] can be used on top of a path-finding algorithm in order to make whole groups of units follow a given path. In recent commercial RTS games like StarCraft 2 or Supreme Commander 2, flocking-like behaviors are inspired of continuum crowds ("flow field") [57]. A comprehensive review about (grid-based) pathfinding was recently done by Sturtevant [50].

Holistic Approaches

Finally, 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 [36] (which uses a combination of case-based reasoning and learning from demonstration) or ALisp [32], 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 [37]. Techniques that scale up to large RTS games as StarCraft are still not available.

A related problem is that of integrating reasoning at multiple levels of abstraction. Molineaux et al. [35] showed that the difficulty of working with multi-scale goals and plans can be handled directly by case-based reasoning (CBR), via an integrated RL/CBR algorithm using continuous models. Reactive planning [67], a decompositional planning similar to hierarchical task networks [23], allows for plans to be changed at different granularity levels and so for multi-scale (hierarchical) goals integration of low-level control. Synnaeve and Bessière [54] achieve hierarchical goals (coming from tactical decisions) integration through the addition of another sensory input corresponding to the goal's objective.

Conclusions

This entry has defined real-time strategy (RTS) games from an AI point of view, and summarized the set of open problems in RTS game AI. After that, we have summarized existing work toward addressing those problems.

RTS games can be seen as a simulation of real complex dynamic environments, in a finite and smaller world, but still complex enough to study a series of key interesting problems. 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.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. Artificial Intelligence Communications 7(1), 39–59 (1994)
2. Aha, D.W., Molineaux, M., Ponsen, M.J.V.: Learning to win: Case-based plan selection in a real-time strategy game. In: ICCBR. pp. 5–20 (2005)
3. Avery, P., Louis, S., Avery, B.: Evolving Coordinated Spatial Tactics for Autonomous Entities using Influence Maps. In: Proceedings of the 5th international conference on Computational Intelligence and Games. pp. 341–348. CIG'09, IEEE Press, Piscataway, NJ, USA (2009), http://dl.acm.org/citation.cfm?id=1719293.1719350
4. Balla, R.K., Fern, A.: Uct for tactical assault planning in real-time strategy games. In: International Joint Conference of Artificial Intelligence, IJCAI. pp. 40–45. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2009)

5. Buro, M.: Real-time strategy games: A new ai research challenge. In: IJCAI 2003. pp. 1534–1535. International Joint Conferences on Artificial Intelligence (2003)
6. Buro, M., Churchill, D.: Real-time strategy game competitions. AI Magazine 33(3), 106–108 (2012)
7. Cadena, P., Garrido, L.: Fuzzy Case-Based Reasoning for Managing Strategic and Tactical Reasoning in StarCraft. In: Batyrshin, I.Z., Sidorov, G. (eds.) MICAI (1). Lecture Notes in Computer Science, vol. 7094, pp. 113–124. Springer (2011)
8. Chung, M., Buro, M., Schaeffer, J.: Monte carlo planning in rts games. In: IEEE Symposium on Computational Intelligence and Games (CIG) (2005)
9. Churchill, D., Buro, M.: Build order optimization in starcraft. Proceedings of AIIDE pp. 14–19 (2011)
10. Churchill, D., Buro, M.: Portfolio greedy search and simulation for large-scale combat in starcraft. In: Computational Intelligence in Games (CIG), 2013 IEEE Conference on. pp. 1–8. IEEE (2013)
11. Churchill, D., Saffidine, A., Buro, M.: Fast heuristic search for rts game combat scenarios. In: AIIDE (2012)
12. Churchill, D., Saffidine, A., Buro, M.: Fast heuristic search for rts game combat scenarios (2012)
13. Danielsiek, H., Stuer, R., Thom, A., Beume, N., Naujoks, B., Preuss, M.: Intelligent moving of groups in real-time strategy games. 2008 IEEE Symposium On Computational Intelligence and Games pp. 71–78 (2008)
14. Demyen, D., Buro, M.: Efficient triangulation-based pathfinding. Proceedings of the 21st national conference on Artificial intelligence - Volume 1 pp. 942–947 (2006)
15. Dereszynski, E., Hostetler, J., Fern, A., Hoang, T.D.T.T., Udarbe, M.: Learning probabilistic behavior models in real-time strategy games. In: AAAI (ed.) Artificial Intelligence and Interactive Digital Entertainment (AIIDE) (2011)
16. Forbus, K.D., Mahoney, J.V., Dill, K.: How qualitative spatial reasoning can improve strategy game ais. IEEE Intelligent Systems 17, 25–30 (July 2002), http://dx.doi.org/10.1109/MIS.2002.1024748
17. Geib, C.W., Goldman, R.P.: A probabilistic plan recognition algorithm based on plan tree grammars. Artificial Intelligence 173, 1101–1132 (July 2009)
18. Hagelbäck, J.: Potential-field based navigation in starcraft. In: CIG (IEEE) (2012)
19. Hagelbäck, J., Johansson, S.J.: Dealing with fog of war in a real time strategy game environment. In: CIG (IEEE). pp. 55–62 (2008)
20. Hagelbäck, J., Johansson, S.J.: A multiagent potential field-based bot for real-time strategy games. Int. J. Comput. Games Technol. 2009, 4:1–4:10 (January 2009)
21. Hale, D.H., Youngblood, G.M., Dixit, P.N.: Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds. Artificial Intelligence and Interactive Digital Entertainment AIIDE pp. 173–178 (2008), http://www.aaai.org/Papers/AIIDE/2008/AIIDE08-029.pdf
22. Hladky, S., Bulitko, V.: An evaluation of models for predicting opponent positions in first-person shooter video games. In: CIG (IEEE) (2008)
23. Hoang, H., Lee-Urban, S., Muñoz-Avila, H.: Hierarchical plan representations for encoding strategic game ai. In: AIIDE. pp. 63–68 (2005)
24. Houlette, R., Fu, D.: The ultimate guide to fsms in games. AI Game Programming Wisdom 2 (2003)
25. Hsieh, J.L., Sun, C.T.: Building a player strategy model by analyzing replays of real-time strategy games. In: IJCNN. pp. 3106–3111 (2008)
26. Jaidee, U., Muñoz-Avila, H.: Classq-l: A q-learning algorithm for adversarial real-time strategy games. In: Eighth Artificial Intelligence and Interactive Digital Entertainment Conference (2012)

27. Jaidee, U., Muñoz-Avila, H., Aha, D.W.: Case-based learning in goal-driven autonomy agents for real-time strategy combat tasks. In: Proceedings of the ICCBR Workshop on Computer Games. pp. 43–52 (2011)
28. Kabanza, F., Bellefeuille, P., Bisson, F., Benaskeur, A.R., Irandoust, H.: Opponent behaviour recognition for real-time strategy games. In: AAAI Workshops (2010)
29. Koenig, S., Likhachev, M.: D*lite. In: AAAI/IAAI. pp. 476–483 (2002)
30. Liu, L., Li, L.: Regional cooperative multi-agent q-learning based on potential field pp. 535–539 (2008)
31. Madeira, C., Corruble, V., Ramalho, G.: Designing a reinforcement learning-based adaptive AI for large-scale strategy games. In: AI and Interactive Digital Entertainment Conference, AIIDE (AAAI) (2006)
32. Marthi, B., Russell, S., Latham, D., Guestrin, C.: Concurrent hierarchical reinforcement learning. In: International Joint Conference of Artificial Intelligence, IJCAI. pp. 779–785 (2005)
33. Miles, C., Louis, S.J.: Co-evolving real-time strategy game playing influence map trees with genetic algorithms. In: Proceedings of the International Congress on Evolutionary Computation, Portland, Oregon (2006)
34. Mishra, K., Ontañón, S., Ram, A.: Situation assessment for plan retrieval in real-time strategy games. In: ECCBR. pp. 355–369 (2008)
35. Molineaux, M., Aha, D.W., Moore, P.: Learning continuous action models in a real-time strategy strategy environment. In: FLAIRS Conference. pp. 257–262 (2008)
36. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: On-line case-based planning. Computational Intelligence 26(1), 84–119 (2010)
37. Ontañón, S.: The combinatorial multi-armed bandit problem and its application to real-time strategy games. In: AIIDE (2013)
38. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: Learning from demonstration and case-based planning for real-time strategy games. In: Prasad, B. (ed.) Soft Computing Applications in Industry, Studies in Fuzziness and Soft Computing, vol. 226, pp. 293–310. Springer Berlin / Heidelberg (2008)
39. Ontañon, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M., et al.: A survey of real-time strategy game ai research and competition in starcraft. IEEE Transactions on Computational Intelligence and AI in games 5(4), 1–19 (2013)
40. Othman, N., Decraene, J., Cai, W., Hu, N., Gouaillard, A.: Simulation-based optimization of starcraft tactical ai through evolutionary computation. In: CIG (IEEE) (2012)
41. Perkins, L.: Terrain analysis in real-time strategy games : An integrated approach to choke point detection and region decomposition. Artificial Intelligence pp. 168–173 (2010)
42. Ponsen, M., Spronck, I.P.H.M.: Improving adaptive game AI with evolutionary learning. In: University of Wolverhampton. pp. 389–396 (2004)
43. Pottinger, D.C.: Terrain analysis for real-time strategy games. In: Proceedings of Game Developers Conference 2000 (2000)
44. Preuss, M., Beume, N., Danielsiek, H., Hein, T., Naujoks, B., Piatkowski, N., Ster, R., Thom, A., Wessing, S.: Towards intelligent team composition and maneuvering in real-time strategy games. Transactions on Computational Intelligence and AI in Games (TCIAIG) 2(2), 82–98 (June 2010)
45. Reynolds, C.W.: Steering behaviors for autonomous characters. Proceedings of Game Developers Conference 1999 pp. 763–782 (1999)

46. Richoux, F., Uriarte, A., Ontañón, S.: Walling in strategy games via constraint optimization. In: Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (2014)
47. Schadd, F., Bakkes, S., Spronck, P.: Opponent modeling in real-time strategy games. In: GAMEON. pp. 61–70 (2007)
48. Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C.L., Ram, A.: Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. In: International Joint Conference of Artificial Intelligence, IJCAI (2007)
49. Smith, G., Avery, P., Houmanfar, R., Louis, S.: Using co-evolved rts opponents to teach spatial tactics. In: CIG (IEEE) (2010)
50. Sturtevant, N.: Benchmarks for grid-based pathfinding. Transactions on Computational Intelligence and AI in Games (2012), http://web.cs.du.edu/ sturtevant/papers/benchmarks.pdf
51. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning). The MIT Press (March 1998)
52. Synnaeve, G., Bessiere, P.: A Bayesian Model for Opening Prediction in RTS Games with Application to StarCraft. In: Proceedings of 2011 IEEE CIG. p. 000. Seoul, Corée, République De (Sep 2011)
53. Synnaeve, G., Bessière, P.: A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft. In: AAAI (ed.) Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2011). pp. 79–84. Proceedings of AIIDE, Palo Alto, États-Unis (Oct 2011)
54. Synnaeve, G., Bessiere, P.: A Bayesian Model for RTS Units Control applied to StarCraft. In: Proceedings of IEEE CIG 2011. p. 000. Seoul, Corée, République De (Sep 2011)
55. Synnaeve, G., Bessiere, P.: A Dataset for StarCraft AI & an Example of Armies Clustering. In: AIIDE Workshop on AI in Adversarial Real-time games 2012 (2012)
56. Synnaeve, G., Bessière, P.: Special Tactics: a Bayesian Approach to Tactical Decision-making. In: CIG (IEEE) (2012)
57. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. ACM Transactions on Graphics 25(3), 1160–1168 (2006)
58. Uriarte, A., Ontañón, S.: Kiting in rts games using influence maps. In: Eighth Artificial Intelligence and Interactive Digital Entertainment Conference (2012)
59. Uriarte, A., Ontañón, S.: Game-tree search over high-level game states in rts games. In: Tenth Artificial Intelligence and Interactive Digital Entertainment Conference (2014)
60. Uriarte, A., Ontañón, S.: High-level representations for game-tree search in rts games. In: Second AIIDE Workshop on Artificial Intelligence in Adversarial Real-Time Games (2014)
61. Čertický, M., Čertický, M.: Case-based reasoning for army compositions in real-time strategy games. In: Proceedings of Scientific Conference of Young Researchers. pp. 70–73 (2013)
62. Čertický, M.: Implementing a wall-in building placement in starcraft with declarative programming (2013)
63. Weber, B.G., Mateas, M.: A data mining approach to strategy prediction. In: IEEE Symposium on Computational Intelligence and Games (CIG) (2009)
64. Weber, B.G., Mateas, M., Jhala, A.: Applying goal-driven autonomy to starcraft. In: Artificial Intelligence and Interactive Digital Entertainment (AIIDE) (2010)
65. Weber, B.G., Mateas, M., Jhala, A.: Building human-level ai for real-time strategy games. In: Proceedings of AIIDE Fall Symposium on Advances in Cognitive Systems. AAAI Press, AAAI Press, Stanford, Palo Alto, California (2011)

66. Weber, B.G., Mateas, M., Jhala, A.: A particle model for state estimation in real-time strategy games. In: Proceedings of AIIDE. p. 103–108. AAAI Press, AAAI Press, Stanford, Palo Alto, California (2011)
67. Weber, B.G., Mawhorter, P., Mateas, M., Jhala, A.: Reactive planning idioms for multi-scale game AI. In: IEEE Symposium on Computational Intelligence and Games (CIG) (2010)
68. Wender, S., Watson, I.: Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar. In: CIG (IEEE) (2012)
69. Wintermute, S., Joseph Xu, J.Z., Laird, J.E.: Sorts: A human-level approach to real-time strategy AI. In: AI and Interactive Digital Entertainment Conference, AIIDE (AAAI). pp. 55–60 (2007)
70. Young, J., Hawes, N.: Evolutionary learning of goal priorities in a real-time strategy game (2012)