Artificial Intelligence Design for Real-time Strategy Games

Firas Safadi
University of Liège
fsafadi@ulg.ac.be

Raphael Fonteneau
University of Liège
raphael.fonteneau@ulg.ac.be

Damien Ernst
University of Liège
dernst@ulg.ac.be

Abstract

For now over a decade, real-time strategy (RTS) games have been challenging intelligence, human and artificial (AI) alike, as one of the top genre in terms of overall complexity. RTS is a prime example problem featuring multiple interacting imperfect decision makers. Elaborate dynamics, partial observability, as well as a rapidly diverging action space render rational decision making somehow elusive. Humans deal with the complexity using several abstraction layers, taking decisions on different abstract levels. Current agents, on the other hand, remain largely scripted and exhibit static behavior, leaving them extremely vulnerable to flaw abuse and no match against human players. In this paper, we propose to mimic the abstraction mechanisms used by human players for designing AI for RTS games. A non-learning agent for StarCraft showing promising performance is proposed, and several research directions towards the integration of learning mechanisms are discussed at the end of the paper.

1 Introduction

The real-time strategy (RTS) video game genre began to appear roughly two decades ago. Published in 1992, Dune II (Westwood Studios)[1] already featured the core concepts of RTS. However, the genre only started getting popular a few years later with the release of titles such as Warcraft (Blizzard Entertainment)[1] in 1994 or Command & Conquer (Westwood Studios)[1] in 1995. Other titles followed, each bringing new additions to the variety and, in 1998, StarCraft (Blizzard Entertainment)[1], one of the best-selling video games and now an acknowledged reference, cemented RTS in the industry. Part of the appeal of RTS games comes from their complexity and the control difficulty resulting from the intensive multitasking they require. In StarCraft tournaments for example, professional players routinely exceed 200 actions per minute. While multitasking poses no issues to computers, they are confronted with the intractable problem of learning in such convoluted environments. The large number of units to control as well as the diverse tasks to complete, coupled with partial observability, result in complex game dynamics.

Besides partial observability, the main cause for imperfect decision making in RTS games has to do with the way the complexity is managed, which is abstraction. Abstraction is an example mechanism used to summarize large quantities of information into a more compact and manageable, albeit abstract form. While it is an amazingly useful and powerful tool, there is at least one significant downside to using it: loss of information. By successively synthesizing numerous basic elements into abstract form, knotty details are eventually overlooked. Thus, abstraction could be seen as an ultimately lossy compression process. Because the quantity of information available at any moment and throughout the entire game is too large, part of it is inevitably discarded in the process of analyzing and rationalizing the game state.

[1]Westwood Studios and Blizzard Entertainment as well as Dune II: The Building of a Dynasty, Command & Conquer, Warcraft and StarCraft are registered trademarks.

Within an ideal framework, agents should not lose any information when analyzing the game state as this loss is only associated to the abstraction mechanisms used by human beings. In practice, this is not the case: agents also discard information because we are not capable of programming them to consider every information. Although this may seem counter-intuitive, the same flaw we suffer from is reintroduced in agents for the simple fact that we do not possess the technology to handle the game in raw form. Part of the objective of this work is thus to propose to the RTS community a simple and generic agent model based on abstract concepts to help efficiently build agents for different RTS games. In this document, we explore the RTS context and propose a modular and hierarchical agent design inspired by abstraction mechanisms used by human players with the aim of subsequently adding learning features. We then provide some promising experimental results in the particular case of StarCraft, and discuss some research directions opened by this work.

The following of the paper is structured as follows. Section 2 quickly covers some related work. Section 3 presents RTS games in detail. In Section 4, we provide an efficient modular and hierarchical agent design. Finally, we conclude and briefly discuss some future works in section 5.

2 Related Work

During the last decade, the scientific community has acknowledged that RTS games constitute rich environments for AI researchers to evaluate different AI techniques. Development frameworks for RTS agents such as the ORTS (Open RTS) project (Buro and Furtak, 2004) appeared and research work started to tackle some of the challenges offered by the RTS genre. Due to the inherent difficulty of designing good RTS agents able to address the multitude of problems they are confronted to, most work has been concerned with specific aspects of the game.

The strategy planning problem is probably the one that has received the most attention with the success of case-based planning methods to identify strategic situations and manage build orders. An approach based on case generation using behavioral knowledge extracted from existing game traces was tested on Wargus[2] (Ontañón et al., 2007). Other approaches for case retrieval and build order selection were tested on the same game (Aha et al., 2005; Weber and Mateas, 2009a). A case retrieval method based on conceptual neighborhoods was also proposed (Weber and Mateas, 2009b). The strategy planning problem has also been addressed with other techniques such as data mining (Weber and Mateas, 2009c). By analyzing a large collection of game logs, it is possible to extract building trends and timings which are then used with matching algorithms to identify a strategy or even predict strategic decisions. Evolutionary methods have been employed as well, mostly in strategy generation (Ponsen et al., 2006). Meanwhile, some work has focused on lower-level problems like micro-management. Monte Carlo planning was among other things applied to simple CTF ("Capture The Flag") scenarios on the ORTS platform (Chung et al., 2005).

Although the above-mentioned works all showed interesting results, none actually takes on all the aspects of the RTS genre. Other works have instead considered the entire problem and present complete agents. A cognitive approach was tested using the ORTS infrastructure, though it suffered from artificial human multitasking limitations (Wintermute et al., 2007). Another approach was based on an integrated agent composed of distinct managers each responsible for a domain of competence (McCoy and Mateas, 2008). Although the design was clear, it lacked hierarchical structure and unit management was largely simplified.

While interesting, these do not offer a clear and simple design to develop and improve agents for RTS games. The model we suggest in this document is simple, efficient and generic and can potentially be used to add learning capabilities in new agents.

3 Real-time Strategy

In RTS games, players typically confront each other in a map with a unique terrain configuration. They start with a set number of units and must build a force to destroy all opponents. Players do not have access to the entire game state. Only areas where they have deployed units are visible. This is commonly referred to as the fog of war. Different types of units can be built throughout

[2]Wargus is a clone of Warcraft II, a RTS title published by Blizzard Entertainment in 1995.

the game. Each has its own attributes such as hit points, speed, range, whether it can fly, whether it is biological, or any other attribute part of the game mechanics. Each unit type also costs a certain amount of resources and requires some technology. Resources can be gathered from specific locations on the battlefield while technologies can be researched by players to unlock the desired unit types. Besides attributes, units also have special abilities (i.e., activating a shield). These abilities are either innate or need be unlocked. Depending on the game, some units may evolve during their lifespan, either acquiring new abilities or improving their base attributes. Some games also feature an upgrade system, which allows players to increase the performance of certain units (i.e., increase all infantry weapon damage).

Another characteristic of RTS games adding to their diversity and complexity is the concept of race. Players do not necessarily have access to the same units and technologies. Before the game starts, each player may choose a race. Depending on the choice, entirely different, yet balanced, sets of units and technologies can be available to different players. In StarCraft, players can choose among three races: the Terrans who excel at defense and adaptation, the Zerg with their overwhelming swarms and the Protoss, a humanoid species with unmatched individual fighting prowess.

One last, and important, aspect in RTS is diplomacy. Players can decide to form alliances or break them during a game to further their own interests. Extreme complexity may arise from such contexts and, as far as it pertains to this document, we instead focus on the simpler free-for-all setting when discussing agents.

Obviously, players must constantly take a multitude of decisions on different time scales. They must decide what and when units should be built, whether the current income is sufficient or new resource sites should be controlled, when to attack or to defend, when to retreat during an attack or whether a diversion is required, whether some unit should be on the front line, whether a special ability should be used, etc. It is also clear that players need to know what their opponents are planning in order to make good decisions and therefore have to constantly go on reconnaissance. By noting that an order can be sent to each unit at any time in the game and that the number of units a player owns is often larger than one hundred, it comes with no surprise that these games are challenging.

4 Agent Design

We first describe in Subsection 4.1 the concept of abstraction upon which the modular and hierarchical design detailed in Subsection 4.2 is based. We provide experimental results showing the performance of our agent playing StarCraft and discuss some limitations in Subsection 4.3.

4.1 Overcoming Complexity

Abstraction can be seen as the ability to reduce information into sets and structures for addressing a particular purpose. In the context of RTS games, the complexity and the large number of units to control cause human players to instinctively resort to abstraction. Instead of thinking about each unit and its abilities individually, they think about groups of units and more abstract abilities such as defense or harassment. Players thus use abstraction to efficiently control the environment. Also simplified by abstraction are objectives. Typically, the objective in an RTS game is to destroy all enemy units. Since there are many units, humans do not think about defeating each enemy unit. Rather, they move along abstract objectives like defeating an enemy outpost or primary base. Using abstract elements of this kind, we can then understand how RTS can be structured into a more manageable problem.

The modular and hierarchical agent model we next present is structured around the primary tasks we were able to identify. It is composed of different abstract managers generic enough to be used in a wide array of RTS games.

4.2 A Modular and Hierarchical Design

When looking at the tasks players must solve, we were inclined to identify to identify two categories based on task decomposition: production-related tasks and combat-related ones. Production tasks include everything from economy management and expansion to build order management and technology appraisal. On the other hand, combat tasks regroup all combat management elements such

as attacking a base or defending an outpost. Furthermore, we divide tasks according to temporal factors. The first group deals with long-term objectives and is referred to as strategy. Decisions at this high level consist of abstract orders like performing a rush[3]. The second one, called tactics, consists of mid-term tasks such as winning a battle. Examples of decisions would be engaging on two fronts or retreating from combat to force the enemy to move to a designated location. The third and last group gathers the remaining short-term objectives. These involve more concrete orders like destroying a specific unit or using a special ability.

As a result, a hierarchical and modular model is proposed in Figure 1. It features a strategy manager at the top level taking all strategic decisions. Directly below come two other managers. A production manager handles construction as well as build orders and controls in turn a number of work squad managers. On the other side, a combat manager takes care of tactical decisions and controls several military squad managers. Information thus travels from the strategy manager and is successively translated into lesser abstract decisions until it reaches the managers at the lowest level which relay direct orders to units on the battlefield. This process is illustrated in Figure 2.

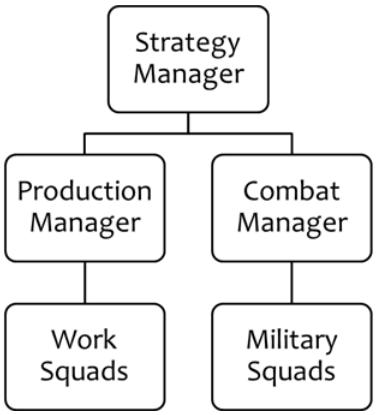

Figure 1: A hierarchical and modular design.

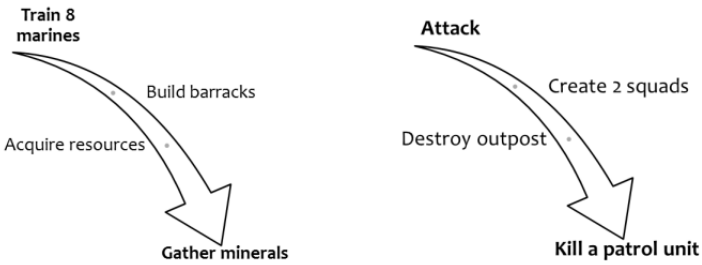

Figure 2: Order processing. In the production order example (left), the strategy manager decides to train 8 marines. This order is relayed to the production manager where it is processed and translated into a series of orders suchs as building a barracks and acquiring the necessary resources. The latter is in turn passed to a mining squad manager which sends workers out to a mineral field. The combat order example (right) can be analyzed is a similar fashion. When the strategy manager decides to attack, it raises an attack flag in the combat manager which responds by preparing some squads using the available units and locating a nearby target like an outpost. It then sends the squads there which run into a patrol unit on the way and receive the order to destroy it from their managers. We can thus see how orders spawn in the strategy manager as abstract decisions and are processed into more and more concrete decisions until they are eventually translated into specific unit commands.

[3]Quick attack against a supposedly unprepared opponent

4.3 Experimental Results and Limitations

In this section, we illustrate the performances of our agent in the particular case of StarCraft. The resulting agent is capable of playing full games as a Terran player facing a Zerg opponent. As can be seen in Tables 1 and 2, the agent controls units very efficiently compared to the default Zerg AI. It managed to score an average of 3 units lost in 10 games versus the opponent's 84.5 average. It even succeeded in winning without losing a single unit in a couple of games.

While this design allows for efficient agent implementations, it does not address the primary issue agents face in RTS games. Indeed, without learning, even an efficient agent will eventually lose against an adaptive opponent as the latter detects its inevitable flaws and starts exploiting them. Flaws can originate from multiple sources. First, it may be that our understanding of the way humans process game information is erroneous and omits important functions, resulting in an incomplete design. Furthermore, even a perfectly similar processing design would still fail to account for all possible game scenarios, leaving some situations uncovered and potentially exploitable.

Game	A	B	C	D	E
Units produced	73	62	72	69	68
Units killed	81	79	78	75	72
Units lost	10	2	0	0	1

Game	F	G	H	I	J
Units produced	77	76	63	85	78
Units killed	100	101	83	74	102
Units lost	2	1	0	13	1

Table 1: Unit statistics.

AUP	AUK	AUL	Games won	Total games
72.3	84.5	3	10	10

Table 2: Average units produced (AUP), killed (AUK) and lost (AUL).

5 Conclusions and Future Works

In this document, we have discussed some thoughts about RTS and the difficulties around it. We proposed a modular and hierarchical agent design inspired by human abstraction mechanisms for which promising experimental results were reported.

While the model described above can be used to efficiently implement effective agents, we still need to embed learning capabilities in order to address the main weakness of current agents, that is the lack of adaptation. Hence, with this model, we can imagine adding learning for carefully selected tasks in the different managers. For example, the strategy manager could learn new strategies by examining the opponents build orders and mimicking them. New tactics, such as squad formations, could also be learned from the opponents own unit grouping. At yet lower levels, military squad managers could learn to prioritize select targets based on the units the opponent takes out first. Although this has not been tested yet, the possibility of adding such features opens avenues to an exciting future for AI in RTS games. Yet further, another interesting step would be the automatic learning of control structures such as the one we proposed.

Acknowledgements

Raphael Fonteneau is a postdoctoral researcher of the FRS-FNRS from which he acknowledges the financial support. This paper presents research results of the Belgian Network DYSCO funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office.

References

Michael Buro and Timothy M. Furtak. RTS games and real-time AI research. In *Proceedings of the 13th Behavior Representation in Modeling and Simulation Conference (BRIMS-04)*, pages 51–58, 2004.

Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-based planning and execution for real-time strategy games. In *Proceedings of the 7th International Conference on Case-Based Reasoning (ICCBR-07)*, pages 164–178, 2007.

David W. Aha, Matthew Molineaux, and Marc J. V. Ponsen. Learning to win: case-based plan selection in a real-time strategy game. In *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR-05)*, pages 5–20, 2005.

Ben Weber and Michael Mateas. Case-based reasoning for build order in real-time strategy games. In *Proceedings of the 5th Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE-09)*, 2009a.

Ben G. Weber and Michael Mateas. Conceptual neighborhoods for retrieval in case-based reasoning. In *Proceedings of the 8th International Conference on Case-Based Reasoning (ICCBR-09)*, pages 343–357, 2009b.

Ben G. Weber and Michael Mateas. A data mining approach to strategy prediction. In *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG-09)*, pages 140–147. IEEE Press, 2009c.

Marc Ponsen, Héctor Muñoz-Avila, Pieter Spronck, and David Aha. Automatically generating game tactics via evolutionary learning. *AI Magazine*, 27(3):75–84, 2006.

Michael Chung, Michael Buro, and Jonathan Schaeffer. Monte-Carlo planning in RTS games. In *Proceedings of the 1st IEEE Symposium on Computational Intelligence and Games (CIG-05)*, 2005.

Samuel Wintermute, Joseph Xu, and John E. Laird. SORTS: A human-level approach to real-time strategy AI. In *Proceedings of the 3rd Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE-07)*, pages 55–60, 2007.

Josh McCoy and Michael Mateas. An integrated agent for playing real-time Strategy games. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1313–1318, 2008.

Joseph Bates, A. Bryan Loyall, and W. Scott Reilly. Integrating reactivity, goals, and emotion in a broad agent. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, 1992.

Michael Buro. Real-time strategy games: a new AI research challenge. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1534–1535, 2003.

Danny C. Cheng and Ruck Thawonmas. Case-based plan recognition for real-time strategy games. In *Proceedings of the 5th International Conference on Computer Games: Artificial Intelligence, Design and Education (CGAIDE-04)*, pages 36–40, 2004.

Paul R. Cohen. Empirical methods for artificial intelligence. *IEEE Expert*, 11(6):88, 1996.

Alex Kovarsky and Michael Buro. A first look at build-order optimization in real-time strategy games. In *Proceedings of the 2006 GameOn Conference*, pages 18–22, 2006.