Adaptive Real-Time Strategic Agent in StarCraft

Abdelrahman Elogeel

Institute of Technology

University of Washington, Tacoma

[elogeel@u.washington.edu](mailto:elogeel@u.washington.edu)

Matthew Alden

Institute of Technology

University of Washington, Tacoma

[mealden@u.washington.edu](mailto:mealden@u.washington.edu)

Andrey Kolobov

Microsoft Research

Redmond, Washington

[akolobov@microsoft.com](mailto:akolobov@microsoft.com)

Ankur Teredesai

Institute of Technology

University of Washington, Tacoma

[ankurt@u.washington.edu](mailto:ankurt@u.washington.edu)

**ABSTRACT**

Algorithms for automated planning and learning are of interest in many industries such as the military, robotics, city planning and lastly game development. Recent advances in computer hardware have made possible the application of such algorithms to Real-Time Strategy games. However, the current state of the art lacks adaptive agents that react in real time. In this proposal we present an adaptive real-time agent that plays StarCraft (a popular Real-Time Strategy game) successfully at a high strategic level. The agent is evaluated to show its capability to adapt in real-time compared to static agents that don’t adapt to the current environment state.

# INTRODUCTION

Real-Time Strategy (RTS) Games are rich research environments having non-stationary worlds with thousands of objects interacting with each other in real time with imperfect information and uncertainty challenges [1]. The research in RTS games impacts military practices like group formations, analyzing attacks, retreat timing, and composing the army. On the peaceful side, RTS games help in proposing a real-time simulation of building a comfortable city given the population growth and environment changes. Most importantly, the current commercial RTS games use static behavior in the built-in agents which result in static, predictable and boring behavior for players. Moreover, some players want to teach the AI agent their playing strategy so the agent can mimic and learn their behavior so players can focus on other tasks.

Much research has been done on this area [2] included techniques that are adaptive but not real-time [3] or works in a reactive and good time manner (using for example FSM) but not adaptive on high-strategic level [4]. Most of the current agents are far away from the level of a commercial bot. The real-time aspect is one of the challenges plus the engineering aspect as well. A commercial agent needs to be well engineered and designed to be generic (not only for one RTS game), extendible (can be extended and changed by game developers), and maintainable.

This proposal focuses on the high level strategic aspect of agents playing Starcraft (a popular RTS game). The high level strategy is responsible for determining the high level overview of how the agent will play the game from different aspects like what’s the agent opening strategy, for example, is it rushing strategy (make a quick army and attack) or defensive (build a strong economy and defend the city) or something else? The high level strategy also determines how the agent will build its economy, for example, the engine can rely on heavily farming only or getting a hiring a constant refineries that get money. What’s the high level structure of the army? There are two additional levels, medium level tactics and low level (sometimes called micro-management), which are not included in this proposal.

# Real-Time Strategy Games

Real-Time Strategy Games are a genre of computer games in which the player is supposed to build a complete city, manage its resources, builds a strong army for this city, gather resources, make alliances, and attack enemies. The goal of a RTS game is to defeat all the enemies and destroy all their structures.

RTS games have some distinctive attributes:

* Real-Time: the game is played in real time with no delay not like chess where user does one move and wait for the other player to do a move. Moreover, the actions in RTS games are *durative* that means an action takes certain amount of time to finish not gets finish instantly
* The state in RTS games is huge where a number of plays can have different number of buildings or units at a time
* RTS games are partially observable so a player can’t see what the enemy is doing right now. This is called *fog-of-war*

StarCraft [5] is a popular RTS game developed by Blizzard Entertainment will be used as a test bed in this proposal. A screenshot from the game is shown in Figure 1.



Figure 1 A screenshot of StarCraft: Brood War.

# RELARED WORK

Approaches used to improve agents in RTS game include three different areas as researchers approach the problem from automated planning, machine learning, and automated planning blended with machine learning approaches. Related work for each approach is described in this section.

Darmok system [3] used Online Case-based Planning approach to make it adaptive in the game. The approach cycle is described in **Figure 2** below

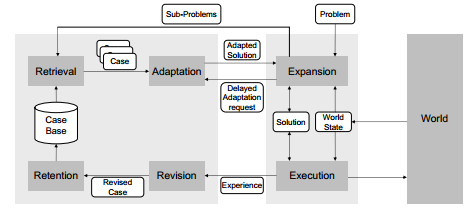


Figure 2: OLCBP cycle

The expansion part is responsible for taking a problem and dividing it into sub-problems using HTN (Hierarchical Task Network) representation. A suitable plan for the current world state is retrieved from the casebase and sent to the execution module, which in turn takes the HTN, evaluates it, and finds plans ready for execution. The system also provides a revision phase where a learning algorithm can be plugged in and later the revised case can be retained back into the casebase. Another interesting part is having prior-game learning where Darmok can observe human players and learn from their playing strategy. Darmok can tackle the adaptively problem successfully, it’s capable of playing complete game in adaptive manner but it lacks real-time performance as the retriever is too slow compared to a commercial RTS game. Besides that, Darmok acts under a perfect information assumption which is not valid for commercial RTS games.

An extension for Darmok system called I-Strategizer [6] improves the case revision phase with a more decent and reliable one. The agent used Reinforcement Learning techniques, specifically Lambda Sarsa, which is a subset from Temporal Difference approach. It also uses Eligibility Traces which add the concept of credibility to a plan. The agent starts in an exploration phase where it tries to use any available case and evaluate it. After having some history about the used cases/plans the agent starts its exploitation phase where it looks for credible and successful plans. The Lambda Sarsa helps in deciding the case success and Eligibility Traces helps in deciding the case credibility. In this approach the agent uses a more intelligent playing strategy, but on the other hand still faces the issue of being real-time and working under perfect information environments only.

In EISBot [7] the strategy selection is implemented using a combination of reactive and deliberative techniques. A collection of hand-authored behaviors is included in a strategy manager for executing exact build order. Goal-driven autonomy [8] is used to decouple strategy selection from execution. EISBot uses case-based planner to transition from the initial strategy into future strategies. The case-based planner retrieves plan that is closest to the existing situation (similar to Darmok [3]) and by doing this opponent modelling and strategy selection tasks are considered to be done in adaptive manner.

Baumgarten, et al. [9] developed an agent that combines case-based reasoning, simulated annealing and decision tree learning. The agent has prior-game learning which forms the casebase that’s used to pick plans while playing the game. The system used ID3 algorithm to learn the decision tree which is used to dictates what specific path to follow when the agent faces a new situation. An example of the generated decision tree is found in Figure 3.

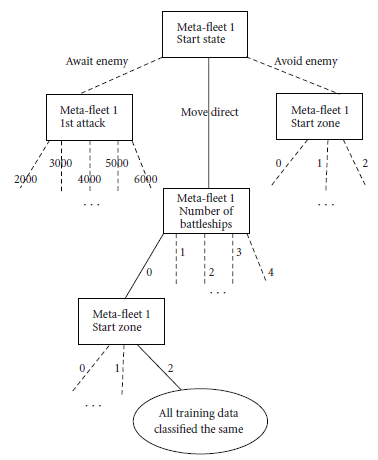


Figure 3: Example of selected path in the decision tree

# TECHNICAL PROPOSAL

This proposal is focused on improving the strategy and plan retrieval in real-time for I-Strategizer agent [6]. The current retrieval algorithm’s performance, as measured by the game’s overall frame rate, is not good enough for commercial game. An acceptable frame per second (FPS) rate for a commercial game is between 30 and 60 FPS [10]. This degradation of performance results from a couple of reasons a) searching through a huge search space with a linear search approach, b) matching the game state features by Euclidian distance for every feature, c) using non-relative and high computation features in the evaluation excessively.

The proposed approach involves learning a decision tree offline for each game goal (like BuildCity, BuildArmy, etc…) and then using this tree in the online gameplay. The tree uses binary search besides having relative and cheap – as possible – computation features for the matching. Consequently, a big part of search space will be excluded, specifically all plans that don’t belong to the goal. At the end of the search, a suitable plan for the current game state should be returned. Each decision tree will use its own features that are relative to the goal context (for example a BuildEconomy goal should not care about enemy’s number of units).

# Experiment Design

## Frame Rate

As pointed out in section 4 the range of frame rate for a commercial game is between 30 – 60 FPS, a measurement for the agent’s FPS will be recorded over couple of games.

## Adaptation Rate

A human player will play against the agent with alternate strategies approach and a measurement for how many times the agent adapted its playing strategy will be recorded from these games.

## Win Rate

The agent will play a bunch of games with human players, static AI bots and deliberative AI agents and then measure how many games the agent won from these games.

## Turing Test

A Turing test [11] will be applied to the game where a human player will play against the adaptive agent and he/she should feel that this is not a machine rather it acts like a human

A comparison with static AI bots and other deliberative agents will be recorded to show the results difference in the new followed approach.

# Schedule

|  |  |
| --- | --- |
| Schedule | Task |
| W1 | Setup the development environment using BWAPI and port the engine to use StarCraft instead of Wargus |
| W2 | Read about decision trees and how to learn a decision tree from a given goal and then decide which algorithm to use. |
| W3 | Design the solution using decision trees on one goal and experiment the design and measure the initial outcome |
| W4 | Improve the previous prototype and implement the full-fledged work with all available goals |
| W5 | Do the experiments and find out results |
| W6 | Document the final work and prepare the demo |

# Deliverables

The deliverables will include the following:

* Documentation and report for the work.
* Source code for the improved I-Strategizer agent playing StarCraft
* Presentation to describe and share the work with colleagues

# References

|  |  |
| --- | --- |
| [1] | M. Buro, "Call for AI Research in RTS Games," in *American Association for Artificial Intelligence*, Palo Alto, California, 2004. |
| [2] | S. Ontanon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill and M. Preuss, "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft," *Computational Intelligence and AI in Games, IEEE Transactions,* vol. PP, no. 99, p. 1, 2013. |
| [3] | S. Ontanon, K. Mishra, N. Sugandh and a. A. Ram, "On-line case-based," *Computational Intelligence,* vol. 26, no. 1, pp. 84-119, 2010. |
| [4] | R. Houlette and D. Fu, "The ultimate guide to fsms in games," in *AI Game Programming Wisdom 2*, 2003. |
| [5] | Wikipedia, 2013. [Online]. Available: http://en.wikipedia.org/wiki/StarCraft:\_Brood\_War. |
| [6] | M. Aref, O. Enayet and A. Al-Ogail, "Intelligent online case-based planning agent model for real-time strategy games," in *Intelligent Systems Design and Applications*, Egypt, Cairo, 2010. |
| [7] | M. M. a. A. J. B. G. Weber, "Building human-level ai for real-time strategy games," in *AIIDE Fall Symposium on Advances in Cognitive Systems*, Stanford, Palo Alto, California, 2011. |
| [8] | M. Molineaux, M. Klenk and D. W. and Aha, "Goal-Driven Autonomy in a Navy Strategy Simulation," in *AAAI*, 2010. |
| [9] | R. Baumgarten, S. Colton and M. and Morris, "Combining AI Methods for Learning Bots in a Real-Time Strategy Game," *International Journal on Computer Game Technologies,* vol. 2009, no. 129075, 2009. |
| [10] | Wikipedia, "Frame rate," Wikimedia Foundation, 24 11 2013. [Online]. Available: http://en.wikipedia.org/wiki/Frame\_rate#Video\_games. [Accessed 28 11 2013]. |
| [11] | A. Turing, "Computing Machinery and Intelligence," *Mind,* vol. LIX, no. 59, pp. 433-460, 1950. |