Adaptive Real-Time Strategic Agent in StarCraft

Abdelrahman Elogeel

Dept of Computer Science and Engineering

University of Washington, Tacoma

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

**ABSTRACT**

Recent advances in computer hardware motivated the work on incorporating planning and learning in RTS games which has a direct impact on many industries like military, robotics, city building and lastly game industry. The current state of art lakes of having adaptive agents that reacts in real time so in this proposal, we present an adaptive real-time agent in that plays StarCraft (a popular RTS game) successfully on high strategic level. The evaluation at the end shows how the agent is capable being adaptive in real-time compared to static agents which don’t adapt to the current environment state.

# INTRODUCTION

Real-Time Strategy Games offer a wealth research environment by having an unstationary environment 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 aspects like group formations, analyzing attacks and 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.

Many research work have been done on this area included techniques that are adaptive but not real-time [2] or works in a reactive and good time manner (using for example FSM) but not adaptive on high-strategic level [3]. Most of the current developed agents are far away of being commercial or close to 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 will be specific for 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 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 from high level aspect 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 different levels which are medium level tactics and low level (sometimes called micro-management) which are not included in this proposal.

# Real-Time Strategy Games

Computer games have many genres one of them is called Real-Time Strategy (RTS) 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 aliases and attack enemies. The goal of a RTS game is to defeat all the enemies and destroy all their structures.

RTS games have some attributes that make them special genre:

* 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*

In this proposed work, StarCraft [4] a popular RTS game developed by Blizzard Entertainment will be used as test bed. 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 included three different areas as researches approached the problem from automated planning, machine learning or automated planning blended with machine learning approaches. Related work for each approach is described in the coming sections.

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

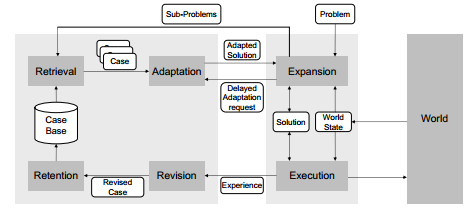


Figure 2: OLCBP cycle.

The expansion part is responsible for taking a problem divide it into sub-problems using HTN representation, then the it retrieves a suitable plan for the current world state from the casebase and send this plan to execution module which in turn takes the HTN evaluates it and find out ready plans for execution. The system also provides revision phase where a learning algorithm can be plugged in (which is done in [5]) 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 player and learn from its playing strategy. Darmok can tackled the adaptivity problem successfully, its capable of playing complete game in adaptive manner but it lacks of being real-time as the retriever performance is too slow compared to a commercial RTS game. Besides that Darmok acts under perfect information assumption which is not valid for commercial RTS games.

An extension for Darmok system called I-Strategizer [5] improved the case revision phase with a more decent and reliable one. The agent used Reinforcement Learning techniques specially Lambda Sarsa which is a subset from Temporal Difference approach. Besides that it used Eligibility Traces which adds the concept of credibility to a plan. The agent starts with exploration phase when it tries to use any available case and evaluate it, after having some history about the used cases/plans the agent starts it’s exploitation phase when 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 gets more intelligence into its playing strategy but on the other hand still face issue of being real-time and works under perfect information environments only.

In EISBot [6] the strategy selection is implemented using a combination of reactive and deliberative techniques. A collection of hand-authored behaviors is included in strategy manager for executing exact build order. Goal-driven autonomy [7] 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’s closed to the existing situation (similar to [2]) and by doing this opponent modelling and strategy selection tasks are considered to be done in adaptive manner.

Baumgarten, et al. [8] 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.

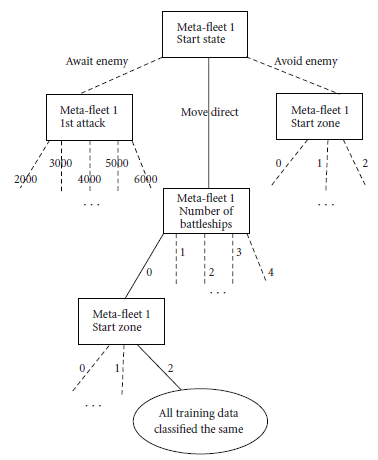


Figure 3: Example of selected path in the decision tree

# TECHNICAL PROPOSAL

This proposal is focused on improving the strategy and plan retrieval in real-time for I-Strategizer agent mentioned in [5]. The current retrieval algorithm performance is not good enough for commercial game <To Do: measure FPS for current system>. An acceptable frame per second (FPS) rate for a commercial game is between 30 and 60 FPS [9]. This degrade of performance is resulted from couple of reasons a) searching through 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 suggests learning a decision tree offline for each game goal (like BuildCity, BuildArmy, etc…) and then uses this tree in the online gameplay. The tree uses binary search besides having relative and cheap – as possible – computation features for the matching. At the end of the search, the suitable plan for the current game state should be returned. Following this will exclude big part of search space which is all plans that doesn’t belong to the goal in ask and 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

## Turing Test

A Turing test [10] 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

## Measurements

A bunch of games played by the agent will be compared with others played by static agents. The graph will show how the agent’s actions are different based on the current environment where the static agent’s actions won’t adapt and will be same.

# Schedule

|  |  |
| --- | --- |
| Schedule | Task |
| W1 | Setup the development environment using BWAPI and do initial work |
| W2 | Read about decision trees and how to learn a decision tree from a given goal |
| W3 | Change the current retriever which uses Euclidian distance to use decision tree. Will do this for just one goal and then get feedback on the results |
| W4 | Improve the previous implemented work based on the given feedback from advisors and expand it to include all goals |
| W5 | Find the results and do the experiments |
| 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, K. Mishra, N. Sugandh and a. A. Ram, "On-line case-based," *Computational Intelligence,* vol. 26, no. 1, pp. 84-119, 2010. |
| [3] | R. Houlette and D. Fu, "The ultimate guide to fsms in games," in *AI Game Programming Wisdom 2*, 2003. |
| [4] | Wikipedia, 2013. [Online]. Available: http://en.wikipedia.org/wiki/StarCraft:\_Brood\_War. |
| [5] | 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. |
| [6] | 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. |
| [7] | M. Molineaux, M. Klenk and D. W. and Aha, "Goal-Driven Autonomy in a Navy Strategy Simulation," in *AAAI*, 2010. |
| [8] | 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. |
| [9] | Wikipedia, "Frame rate," Wikimedia Foundation, 24 11 2013. [Online]. Available: http://en.wikipedia.org/wiki/Frame\_rate#Video\_games. [Accessed 28 11 2013]. |
| [10] | A. Turing, "Computing Machinery and Intelligence," *Mind,* vol. LIX, no. 59, pp. 433-460, 1950. |