CS 461 - Fall 2016 - Progress Report

Project DevAI

Jacob Broderick, Kristen Patterson, Brandon Chatham

Abstract

The goal of this project is to create an agent to play the game Starcraft Brood War, a real time strategy game created by Blizzard Entertainment. The solution to the project will be a template that future students can use to develop better solutions than those provided in this project. In this document, we will discuss what we have completed in our project as well as discuss our process and problems we had. The goal of this document is to reflect on our progress as well as make sure we are on time and on task. Furthermore, this document discusses revisions made to project documentation that reflects any deviations from the original plans during development.

I. Project Development and Document Revisions

A. Current Progress - Jacob Broderick

B. Where the Project is Headed - Brandon Chatham

Seeing as our project has nearly if not completely met our first project goal, we have started considering what stretch goals we want to implement. Because there are a plethora of Starcraft strategies, we do not want to implement functionality scoped tightly to any of them at this point in development. The library is still extremely basic, and we want to give as broad a range of tools to users as we can before our release. Furthermore, we hope to make our tools competitive and use powerful algorithms to optimize them. Optimization is key for competition and important for us to really show off our project.

1) Functionality to Consider: Some of these ideas are much harder than others to implement. These are the additions we are considering with a brief explanation of the importance it might play in an AI module using our library, and a brief summary of implementation details.

a) Scouting: In order to make strategic decisions, the AI is going to need information about the map and information about the enemy. A standard strategy is to use one of the first worker units you train to explore the map and identify the opponent's starting position. It is safe to assume functionality for early scouting would be used by almost all AI modules and therefore worthwhile implementing sooner rather than later. As strategy decision trees get more complex, scouting will allow the AI to respond accordingly to the opponent's strategy. There are two parts to scouting implementation: scout maneuvering and event handling.

b) Scout Maneuvering: This operation will use BWTA information to direct any scouting unit(s) to the potential starting locations received from the function call BWTA::getStartingLocations(). If it is later in the game, this could be adjusted to first check potential expansion locations before going to the main base as the main base will likely be harder to infiltrate.

c) Event Handling: As the map is explored and the opponent's units have been seen, events will trigger. These events are added to a list of "unhandled" events. The AI can access these events and keep track of the information they provide. We have a decent understanding the life-cycle of events as they go from unhandled, to handled, but until we start testing, we will not fully understand how they work beyond that. Event handling is also crucial for machine learning techniques. Keeping track of these events allows the AI to assess the situation and make some calculation based on data from previous instances to make the most advantageous decision.

d) Advantageous Positioning: This would be a large effort for the project, but BWTA identifies choke-points on the map. If used properly, this information can allow the AI to make the most of each unit in combat and make defending its base easier by building structures in locations that impede the entry of enemy units.

e) Player Information: This would be a Player class that encapsulate the player information of the AI. This encapsulation helps the AI keep all of the information it needs in one object rather than instantiating several global variables as we have seen in some more basic AI modules.

f) Enemy Information: Much like the Player class, this would encapsulate meaningful information about the opponent that would be updated using event handling. If an AI is able to explore enough of an opponent's forces, it could begin to make projections calculations about them. Essentially, the AI would begin to predict the opponent's strategies similar to the way a human would.

2) Optimizations:

a) Mineral Mining: In Starcraft, workers collect resources by being stationary next to resource deposits. Only so many workers can gather from any one resource deposit at the same time. We are working on an algorithm that would keep track of how many workers are mining from the mineral fields and to which cluster of minerals to send a worker to minimize the amount of time spent in queue. This will be a queue-based system that pays attention to the total number of workers assigned to each mineral deposit and the total number of workers gathering at that base.

b) Data Mining: While this is likely an overly ambitious addition to make to the project at this point, we are considering it. First, we need replays of games. There are databases of Starcraft Broodwar replays available but we could also use a crawler to download a specific subset of replays we may be looking for. This would be useful for future developers when trying to gather very specific datasets. After that, we need a replay analyzer. This can be done in a few different ways but in simple terms, the tools all produce a dataset of the replays which we will apply machine learning techniques to.

C. Problems Faced - Brandon Chatham

This term has had fewer unforeseen problems than the last. While setting up our development tools, we did run into a crippling case of configuration issues that caused our development and testing to stop for roughly a week as we integrated the Broodwar Terrain Analyzer (BWTA) API into our project. It was a headache for our team, but it would be a mistake to not use such a powerful tool just because it was giving us unusual configuration errors. Despite referencing several Github pages and the starter guide for BWTA, our AI module consistently crashed when trying to analyze the game map. Eventually, we contacted BWTA developers who helped us debug and resolve our issues. After further research we found that BWTA and BWAPI were incompatible for a short time because BWTA was not consistently being developed upon. We unknowingly downloaded the source code and libraries of BWTA from this time of incompatibility. Even though we had done everything else correctly on our end, the code under the hood was not correct. As for design problems, we did not encounter any. We did however adjust our plans to abstract our AI module functionality less in order to make our tools more intuitive for users. Future developers using this library will likely not be considering macro or micro concepts in development decisions. However, that abstraction may be necessary when machine learning is implemented in the future in order to help the AI prioritize decision making. But for the sake of users, we decided not to worry about that.

II. Retrospective