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 Recap - Kristen Patterson

A. Problem Statement

The purpose of the project statement was to clarify with our client the exact problem that he wanted fixed. The problem our client and us had decided on, in general terms, is to create a template artificial intelligence for the real-time strategy game, Starcraft Brood War. This artificial intelligence will be created for future club members at OSU to use as an example to make their own. Through the process of developing an artificial intelligence, we need to create an extensive library of functions to help future users be able to easily interact with the API used to build the artificial intelligence. The API is used to interact with the game and has many classes built of many functions to error check and perform actions. The base requirement for the artificial intelligence is one strategy implementation, but if there is time in the development process, there are stretch goals to make many different scripted strategies and have the agent switch between them based on triggers in the game. There is also a stretch goal of competing in a tournament to see how effective our created agent is, however, this was made a stretch goal as it requires a much earlier deadline that we are still not sure we will be able to make. Our problem has not changed so we have not had to revise our problem statement.

B. Requirements Document

The requirements document was a tool to specify the exact details of the problem. Specifying the details helped to categorize the project and separate the tasks needed to be completed. It also provided a timeline for the stages of the project. In the requirements document, we specified that our code needed to be well documented in order for students to be able to easily read and understand our code. The code also needed to be organized in a way that had little dependencies and ease of access so students could navigate our code effectively. In order to be able to participate in the tournament and not get disqualified, the code also had to be efficient and the agent can't slow the game down by a specified amount of frames. We also specified the different types of actions the functions and classes could be specified under. The artificial intelligence needs to be able to handle micromanagement of individual units in combat and macromanagement of the base and unit creation. Any stretch goals that will be implemented in our code will have to follow the requirements described for the base goals. This document also helped with time management as it provided an even amount of time to each part of the project. There has not been any changes in the requirement so there will be no documented revision as of now.

C. Technology Review

The purpose of the technology review was to identify and compare the tools available to help development of the project. It also split the roles of the project for each group member to be responsible for. Firstly we specified the programming paradigm we should follow when coding and then we described the environment we would be using, namely the language and integrated development environment. We also described the API we would be using to interact with the game and the documentation we would use to understand the API. The technology review also detailed which choice algorithm we would be using when we implemented our stretch goal of choosing from scripted strategies. The API that we chose had some extensions to help in our development of our agent and we decided on which one we would use. Finally, we discussed the compiling environment we would use to compile our code and how we would compile the code. Some of the technologies we chose for our project were either preferred by the client or required by the API we had chosen. There were also a couple of technologies that did not have many alternatives. This helped the development team specialize in their specific areas and

provided a starting point for development. Due to some slight problems encountered in the development of our code we have added some revisions the technology review.

D. Design Document

In the design document, we further described how we were going to develop our code. The three sections we focused on discussing for our design document were documentation, modularity, and decision making. For our documentation, we talked about how in order to increase the chance other easily understand our code we will make function headers for each function describing the purpose and the input and output of the function. We also talked about how it needs to be organized. Since this is a large project we will be further organizing our classes and their functions into different files for better navigation. Modularity of the code was also discussed and we decided that the least dependencies that our code had, the better. Modularity was a focus because if others wanted to use specific sections of our code they could easily pick and choose what they wanted. The last focus of the document was our decision making process. While decision making in our program is a stretch goal, it is a difficult topic and requires further discussion to remove ambiguities. Again, due to some complications during the coding process, there have been slight revisions to the design document.

II. Project Development and Document Revisions - Jacob Broderick

The project uses a program called chaoslauncher to inject code into Starcraft Brood War while it is running. The program takes a .dll file and follows the commands listed in it while it is running. The project only needs chaoslauncher, a proper game .dll, and an updated installation of Starcraft. Chaoslauncher will start Starcraft, and all that needs to happen is a player controlled game needs to be started. After that the game will run by itself. The project so far has involved changing the program so that the dll is able to run the game. The Starcraft AI currently is capable of doing all of the things necessary to play the game Starcraft Brood War. It can manage resources, build buildings, and build units. There currently is a dumbed down combat class. It keeps track of military units, and can force them to be moved around and attack. There are still many optimizations that are currently being worked on to make sure these things are able to compete with other artificial intelligences, even at a basic level. There is an event handler that is partially completed that will allow for the tracking and responding to events. This works like an interrupt for when the ai is under attack or when supply needs to be increased, or something similar. There is also a player class that handles all of the data that keeps track of game state. This will be used with all of the classes, and will be integral in making sure our AI can be written to do scripted strategies. The player class will also hold other helper classes in it such as an expansion class. This will keep track of all of the mineral patches next to an expansion, and order them based on how efficiently the drones can mine them. It will also keep track of gas, and any gold expansions. Optimizations like ordering minerals based on distance are what the group is trying to focus on moving forward.

A. Classes

We have added many different classes that each determine how a certain aspect of the ai is supposed to function. The first is the ResourceGathering module. The resource gathering module is in charge of the creation and execution of any workers, and the building of expansions. This focuses a lot on the economy of the game, and is important in making sure the game can be won if the game goes late. Currently it determines where the scvs farm, and when to build them. It also determines how many scvs are required for the different tasks.

The BuildingConstruction module determines how the building are created, and where they are placed. This currently will build new expansions, supply depots, and the required buildings for building the army. It uses the terrain analyzer to make sure that the placement is more optimal than if this task were to be done manually. The building construction also ensures that there is enough total supply by checking to see if supply depots are necessary, and manages building them. The building construction class also manages any other buildings that could need to be built. This will be helpful when trying to add future ai scripts.

The maptools class assists with any necessary map based actions. This can be from scouting to building. The maptools class will scan any maps it hasn't encountered, or use data scanned previously with BWTA. BWTA is the terrain analyzer written for Starcraft Brood War. The Terrain analyzer helps with expansion placement and locating distances between objects. It will also be useful for blocking off ramps into bases, and flying the buildings around.

The unitaction class is in charge of creating military units, moving them, and scouting. This will use some pieces of other classes as well. The scouting will be extra important in the early game. Some AIs do not handle being attacked early very well. This part of the project can cause people to lose before the game even starts. It also handles marine movement, and how they will attack.

B. Document Revisions

There have been some changes to the design philosophy of the AI. Originally there was going to be an emphasis on the separation of macro and micro. Now the two are integrated in order to make a more readable library. Having unit control and unit creation be a part of a unit class helps clean up our namespace, and have less data to keep track of. It also allows to trigger events in a more meaningful way. Originally the plan was to have all of the resource based actions be their own thing. Now that they are not separate we can also create objects that have the benefit of using both micro and macro actions in the same call. This could be useful when moving drones around to defend against an attack while also ensuring that no minerals are lost. It also allows certain units with the sole purpose of macro to be used more closely with the micro units. Most of the proposed goals are still possible, so no stretch goals need to be removed from the original plan. The technologies that were proposed are still completely valid.

III. 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.

A. 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.

1) 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.

2) 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.

3) 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.

4) 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.

5) 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.

6) 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.

B. Optimizations

1) 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.

2) 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.

IV. 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.