CS 463 - Spring 2017 - Midterm Progress Report

Project DevAI

Jacob Broderick, Kristen Patterson, Brandon Chatham

I. Recap

The project's primary focus is to create an environment where students can learn about artificial intelligence in a club setting. Professor Fern, our sponsor, wants to have a well documented example of an AI for a game. The chosen game was Starcraft Brood War. The AI was to be built in BWAPI (Brood War API). The API is well documented and had many other working examples to use as reference to get our project working. The API provides function calls that get injected directly into the code.

A. Starcraft Brood War

Starcraft Brood War is a real time strategy game created by Blizzard entertainment. A real time strategy game allows players to build and control an army in real time. The game requires strong micromanagement skills because the player has to control all of the units simultaneously. Starcraft allows players to play as three different races: Terran, Protoss, and Zerg. These races all have unique playstyles and require different strategies to ensure victory. This makes Starcraft a good game to choose for creating an AI. A computer is not physically limited by mouse and keyboard input, so there is an increase in the amount of actions performed per minute. It also allows different strategies to be implemented for each race. The chosen strategy of this project is to use the Terran strategy to build marines. This strategy is one of the more simple ones because it only requires one building type to build units, and the units only require one of the two resource types. This allows us to focus more on the AI and less on the strategy for victory.

B. BWAPI

Brood War API is the chosen development platform because it offers all of the functionality needed to create a program that plays Starcraft. It contains all of the functions for building, controlling, and monitoring units that would be necessary to create a bot that plays Starcraft. It works by pairing with another program, chaoslauncher, to inject itself directly into the game while it is running. It receives information from the process and responds by sending inputs to the game.

C. Documentation

The documentation for this project needs to be incredibly thorough. There needs to be enough information that a student can open the project and start working on it. This will help future students that would like to join a club based on this project. The documentation needs to show what is required to get the project up and running. This includes how to download the required files in order to run chaos launcher and the game.

D. Artificial Intelligence

The requirements of the project is making a working AI that can use the functionality of Starcraft Brood War. This includes things like managing resources, building units, and moving them. The goal is to implement a Terran marines strategy that is as straight forward as possible. There are many stretch goals for the AI such as tournament competing, learning between matches, and learning off of battles. These are all possible, but there may not be enough time for them.

II. What we have

The main components required for our project are simple artificial intelligence to play StarCraft Brood War, an extensive library to interact with the API for the game, and a thorough documentation detailing how to download, install and create an artificial intelligence using our code. In order for our artificial intelligence portion of the project to be considered complete, our artificial intelligence needs to be able to gather resources, build buildings, train units, and attack enemies. Our library is considered complete if our artificial intelligence can do all of the tasks it needs through our library rather than directly with the API. It is also considered complete if it is organized and functions are simple. The documentation is considered complete if it accurately describes how to install and run the project as well as describes the library in enough detail to understand each individual function and their purpose.

A. *Artificial Intelligence*

Our simple artificial intelligence has done almost all of the previously described purposes. First, it has fully utilized the functions created in our library in order to get all idle workers to gather resources. It also, when given a flag, selects a worker to build a specific building depending on the progress of the game. The buildings can also be selected and can train either workers or marines. The only portion we have not completed is getting our military units to attack the enemy. We have added in a function to explore and scout out the enemy, which is an important and difficult part of attacking the enemy.

1) Process: The process of the artificial intelligence is a an initial startup period followed by nested for loops to iterate through individual units in the game. In the startup period, the artificial intelligence initializes some of the flags used to initiate actions in the game, and the artificial intelligence uses the terrain analyzer extension to the API, called BWTA, in order to read the game map and convert the game map into data that then can be used. It also runs the initial startup procedures that the API needs to run in order to start up the artificial intelligence. After the startup period, the artificial intelligence moves into the main nested for loop of the program. This nested for loop loops every frame and then loops through each individual unit in order to give each one a command. There are 4 main units to control in our program, and they are all checked for the current status to see whether they are busy or if they can be used in other actions. They are workers, command center, barracks, and marines. The workers have a default function to collect minerals when idle. The workers can also be called and used in functions to construct a building or to scout the enemy. The command center only has one real function which is to create workers. The barracks also has a simple function of mainly just training marines. The marines purpose is to move towards the enemy and attack them, but this function hasn?t been implemented yet. Within each of these units functions there are conditional checks to see if there is enough resources and supply count in-game to execute the functions. If there is not enough resources then the function just has to wait. If there is not enough supply count then the artificial intelligence grabs a worker and has it build a supply depot in order to expand the maximum supply allowed. We have also implemented an expansion function which is used to solidify the in-game economy.

2) Strategy: The main strategy that our artificial intelligence follows is a basic marine-ball. This a simple strategy for the game which entails building up a sturdy economy with enough workers and then transitioning into an aggressive approach. This aggressive approach is just continually training marines and sending them to attack the enemy until one of the players wins. The winning condition is when all of a player?s buildings and workers are destroyed. We also had a stretch goal that we did not reach, where the artificial intelligence would have multiple different strategies to choose from and based on the performance of the strategy the artificial intelligence would try and switch it up. The artificial intelligence could be expanded upon further by adding in a way for the program to analyze what type of military unit the enemy is using and then go through the steps to build up a counter to it.

B. Library

Our library is the portion of the code that we focused the most on because the main point was a more simplified interaction with the API. The library has 6 main classes to it each organized under a type of function related to the game. These 6 classes are called BuildingConstruction, ResourceGathering, UnitAction, ScoutManager, PlayerInfo, and MapTools. The first class, BuildingConstruction, is where most of the building units? creation in the game is error checked and initiated. Our class can build any of the core buildings for any race including command center, barracks, supply depots, and gas refineries. Most of the functions in this class just require a nearby unit to be passed as a parameter and the conditional checking and placement are done within each function. It is possible to send the tile position for the placement of a building, which is done when building an expansion using information gathered from our MapTools class. The ResourceGathering class is where we placed most of the code to handle the economy of the game. The purpose of this class is to expand the economy to get resources quickly and keep the supply of resources stable. The functions within the class build workers, check if the worker is busy, and then sends the workers to gather if they are not busy. The UnitAction class is supposed to handle any actions that a unit needs to take that is not categorized in any other class. This class handles checking the unit?s state, training marines, and storing the marines in a data type for quick access. The ScoutManager class handles finding the enemy using the terrain analyzer extension for the API. The PlayerInfo class is where most of the variables for the happening of events and offsets are stored. This class was created in order to avoid global variables. The final class MapTools is mainly for the use of the terrain analyzer and is where the current playing map is converted into data that the artificial intelligence can understand. This class focuses on providing a tile position for an expansion.

C. Documentation

The last portion of the project was documentation. Documentation is vital for our project since we want others to interact and read directly from our code rather than just run the program. The documentation is split up into 4 different parts. The first part is the procedure on how to install the project and all other necessary files and it also includes a link to an in-depth tutorial on setting up the environment. The documentation then goes on to describe each class and each individual function talking about the purpose, input, output, and parameters. The next part that the documentation describes is how to import the finished artificial intelligence into the game. The last portion talks about a side feature added in where the user can run a download scraper to download videos of past games being played.

III. What is left

A. Stretch Goals

1) Base Expansion: The only primary objective we have left in development that has not been completed and passed testing is the expansion but which will be touched out more late.

2) Other Stretch Goals: Fortunately, the other stretch goals we started have been finished. There may be one small addition made to add the ability for scouts to attack the opponent when it finds them, but other than that, stretch goals are being wrapped up and in some cases, being put through regression testing.

B. AI Module

In order to show all of our development in action, or at least the interesting parts, we need to design our AI module to have a scripted strategy it takes. This will not be difficult because most of the code is already in the AI module only it is commented out for unit testing purposes. Once we build the script the way we like, we will film it and create our pitch for the product.

C. The Pitch

Our pitch will largely be focused on how effective a resource this could be for students to learn about AI and machine learning algorithms. AI and machine learning overlap a bit conceptually but they are often thrown around as buzzwords in the industry. This project will give students the ability to see the steps behind scraping data, analyzing the data, and running it through data analysis algorithms for dynamic AI decisions and that is what our pitch will focus on.

IV. Problems

A. Expansion Bug

Our largest issue to-date is the expansion function bug. While we have the function written similarly to other functions that build structures, the primary difference is the location it chooses to build. The location of an expansion is very important because you want it as close to the resource deposits as possible. As we have mentioned in previous reports, our code finds the correct base location and is able to send units to that position, however, when we try to construct a base there, the function behaves not as expected. At first, we did not think this would be a persisting bug. It is very unfortunate that it has been stubborn as it was one of our bigger talking points for our library and pretty fun to watch visually. We will continue to try to fix this until expo and we have reached out to developers of the BWTA API for assistance on what we might be doing wrong. Hopefully they will be able to find a small hiccup in our code.

B. Deep Mind

While this is not a problem in the general sense of the word, our project has been focused on abstracting the difficulties of using BWAPI to develop rudimentary AI modules. Deep Mind has announced it will be developing a Starcraft API that will likely be vastly better than BWAPI. Additionally, Deep Mind is partnering with Blizzard, the creators of Starcraft to make this the industry standard of AI libraries for Starcraft and potentially all real time strategy games. So while this is not directly a problem for our project, it does significantly devalue it as Deep Mind resources will dwarf anything we have put together. Although, this project is not exactly cutting edge anyways.

C. New Starcraft Version

Because of the preparation for Deep Mind development, Blizzard has released a new patch. This may cause direct problems for our code because we are not sure the new version will be backwards compatible with BWAPI. As of now, we have not tried it out because we have found resources online say it is difficult to return to the older version once you have updated. For now, we plan to stay on our current version until expo is completed and our project is done being evaluated. Then we can take the chance to see if BWAPI still works.