CS 461 - Fall 2016 - Design Document

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 how we will design and implement our project. The design choices we make will help as a road map and checklist to follow.

CONTENTS

I. Introduction

A. Date of issue and status

12/4/2016

B. Scope

This document will provide details on the design of the documentation, modularity, and the strategy implementation of the Starcraft AI project.

C. Authorship

Brandon Chatham - Modularity

Kristen Patterson - Documentation

Jacob Broderick - Decision Making

D. Design languages

The design language used will be a UML diagram.

E. Glossary

AI - Artificial intelligence

Agent - The program that will play Starcraft Brood War by itself

Client - Person or organization that has requested the project to be developed

User - Any future Oregon State University club member that wishes to interact with the system

Starcraft Brood War - A real time strategy game created by Blizzard Entertainment

API - Application Programming Interface

BWAPI - API used to interact with Brood War's code

Script - Procedure in a C program that calls a variety of generic real time strategy procedures

Micro - A term used in Starcraft to describe actions pertaining to controlling individual units, mostly related to attacking

Macro - A term used in Starcraft to describe actions pertaining to managing resources and constructing buildings, mostly related to economy management

Protoss - Alien race, very futuristic technology, builds advanced units

Unit - A controllable object in Starcraft that can move, attack, and gather resources

Race - In Starcraft there are three different playable races Terran, Zerg, and Protoss.

Resource - An item in Starcraft used to construct units and buildings

Terran - Human race, very military based

Zerg - Zerg is an alien race, focuses on building lots of units, swarm and hive mind type of race

F. Change history

II. Design Stakeholders

A major stakeholder in the design process is future students involved in an Oregon State University club. These students will be reading and using the code included in the project.

III. Design concerns

A. Documentation

The design concerns of the future students are that the code is well documented and organized.

B. Modularity

In order for the code to be transferable, it should be broken into modular pieces focused on specific functionality accessible through the BWAPI.

C. Decision Making

The best path to a successful bot that plays Starcraft is to have proper decision making. The decision making must be good enough that there is a chance for success.

IV. Design Views

A. Documentation View

The code is well documented if it has concise comments and includes functional header comments. In order for the code to be organized, it must be modularized into simple functions and further separated into header files containing functions with similar purposes. The functions also need to be listed in order of complexity with the most complex functions requiring the highest amount of dependencies is located at the bottom.

B. Modularity

The code will be modular with respect to the functionality of the BWAPI. Creating stand-alone modules that perform necessary Starcraft game actions will improve the learning process for student developers who use the code, and will enable them to produce AI agents more quickly even with limited or no experience with AI software design. Furthermore, the modules should have limited or no dependency on other modules. This allows for simple transferability into a developer's code.

C. Decision Making

The AI is able to decide what strategy to implement based on the state of the game. Multiple strategies will exist that determine the behavior of the AI. Certain buildings and units will be created depending on the strategy chosen. What the units do will be determined by the strategy, but the way they move will be executed by a movement module. The strategy will work for a specific race within the game, and would be catered to fit that specific strategy. Many different strategies can be implemented and tested, and should just require specific calls from the system.

V. Design Viewpoints

A. Documentation

1) Pattern: The pattern viewpoint covers the design for the commenting of the code. Commenting in the program will require reuse of function header comments to describe the function's purpose. Functional header comments will follow a specified template of describing the name, process, inputs, outputs, and requirements of a function. In-line comments must

be included for any variable declarations, loops, and function calls as well as any ambiguous declarations. The parts of the functional header comment must directly associate with code in the function. For example, the inputs are the parameters of a function and the outputs are the returned values. This viewpoint will be supported using the UML composite structure diagram.

2) Composition: The composition viewpoint is used for the design of modular functions and files. Composition will help to localize and simplify blocks of code into functions. Functions will then be localized into separate files based on their purpose. Most functions will be decomposed into modules with a singular purpose while their will be a few functions that control the system and call upon the simpler modules. The HIPO diagram will be used to supplement this viewpoint.

3) Dependency: The dependency viewpoint designs the order of the functions in each file. The design of each file will follow the order of least complex function with least dependencies on top while more complex functions that use other functions are on the bottom of the file. A UML component diagram will be used to show the dependencies.

B. Modularity

1) Structure: The design concerns of modularity are code transferability, and understandability. Transferability refers to a student developer's ability to move code from one project to another with little or no required code edits. Understandability refers to the level of effort a student developer would have to invest in order to understand what a module is doing. Modularity of the library will be categorized with respect to individual classes or small groups of classes relevant to a desired functionality. The BWAPI classes are already broken into categories for interacting with the AI agent and game. This library will access functions relevant to specific actions based on functionality of the module. This design convention makes learning easier by providing concise examples for student developers to use and study. An example would be pulling game state information with regards to in-game resources. This will be important for many of the logic modules created. These modules will organize the functions that pull the appropriate data and can send it to other modules for further algorithm-based actions. In this example of resource collection, modularity makes it easier for students to separate the logic of data gathering from the other categories of functionality. Additionally, it makes modules more transferable to other AI agents without disrupting other systems within the overall body of code.

C. Decision Making

1) Strategies: The strategy viewpoint covers a scripted strategy that will accomplish a single thing. For the race Terran, this could be a strategy that involves spending all available resources on the marine unit. These strategies will also determine how and when the AI will take certain actions such as attack/defend. These strategies will be based around timings that are used to determine when it is the best possible time to do certain actions.

2) Strategy Choice: The choice of strategy will depend on the state of the game. If the opponent is playing aggressively, a defensive strategy can be chosen. It also can determine when to upgrade units, or choose different units to build. If the enemy has numerous air units, switching to an anti-air option would be more successful than continuing with a non anti-air unit based strategy.

VI. Design Overlays

A. Modularity

As mentioned in the Requirements Document, categories will be broken into micro and macro game actions. Micro actions are tactical and more unit-specific while macro actions are resource gathering and base construction focused. Also, these categories are broken into decision-making modules that pull game data and algorithmically decide what should be done and communicate with action modules that act on the decision making. This is applicable to the modularity design concerns as it provides developers easier code sharing and transferability when developing competitive AI's with other developers.

1) Modularity of Strategies: These strategies should be modular, so that they can be tested and removed as needed. This will allow for optimization of the AI when playing many games with other AIs. This will also allow us to test our bot against itself, with different configurations.

VII. Design Rationale

A. Documentation

1) Commenting: Adding function header comments and using a template will help future students to quickly and easily find the information they need about the function. In-line comments will also help future students follow the process in the function.

2) Modular functions and files: Keeping functions modular and organized into separate files will help future students to easily navigate the code. It will also help them use the code as the can take separate modules or files easily.

3) Ordered functions: Ordering functions by complexity will help the future students navigate as they can quickly find simple functions to use. They will also be able to physically see the use of simple functions to support larger more complex functions.

B. Modularity

1) Categorization of Modules: Categorizing the library modules first based on the functionality they serve with respect to the game, then micro and macro game actions, and finally AI decision-making or action will make library components concise, easier to understand, and more transferable across AI agents.

C. Decision Making

1) Choosing Strategies: Multiple strategies will be required in order to ensure that the AI has the highest chance of success. It will gather data from data modules that will assist in discovering the best option on how to proceed.