Agent-Oriented Programming Framework Design for Real-Time Strategy AI

XXXXXX and **XXXXXXX** and **XXXXXXX** and **XXXXXXX**
123 Alphabet Lane
Somewheresville, NotYourState 11111-2222

Abstract

We present a Multi-Agent System Framework, KhasBot, to handle the Artificial Intelligence in a Real-Time Strategy Game StarCraft: BroodWar. This Framework will be tested in the SC AI 2011 competition as well.

Introduction

Real-Time Strategy Games are an area of interest in Artificial Intelligence Systems for many reasons. They must adapt quickly to changing environments, make decisions on non-complete information, and involve multiple complex AI issues in general (for example, reactive planning, path-finding, and resource scheduling).

Background

What we do in this project is combine two different areas together, BWAPI (BroodWar API) and JADE (Java Agent DEvelopment Framework).

BWAPI

BWAPI is an open-source api that allows injection of the game Starcraft: Broodwar. It was developed in C++, but many extensions have allowed access to other languages such as Python, C#, and Java. We used Java.

BWAPI breaks down the StarCraft game into 5 basic objects:

- *Game*: holds all information about the current game being played.
- *Player*: holds all information a player would have, for instance current resources, buildings, and controllable units.
- *Unit*: represents a piece in the game, be it a mineral, building, or probe.
- *Bullet*: represents a projectile from an attack.
- *Force*: represents a collection of players attempting a singular objective.

With these objects, what generally happens is that another object *AIModule.cpp* is made as a *dll* which is used as the injected code.

The basic flow of BWAPI then works as follows, when the actual game begins BWAPI will call up an *onStart()* method from the *AIModule* which will setup all the above objects with the begining of the game information. After this, an *onFrame()* method is called every frame of the game. This is the looping method which holds all the reasoning and compuation of the actual AI.

The *AIModule* returns orders to the StarCraft game via *UnitCommand*s. By giving various Units in control of the AI player, the game can progress. These commands are things such as build, attack, research, and upgrade.

Agent-Oriented Programming

Agent-Orient Programming is a Programming Paradigm started back in the 1990s. The basis is to on the concept of 'agents' as opposed to say, objects. By doing this, one can focus more readily on the reasoning processes and communication paths of a system, rather than getting lost in all the method calls. Aside from communication, Agents really only do one or two things.

The basic flow of Agent-Oriented Programming, which our Agents follow, is that Agents hold a set of beliefs and then based off changes (or additions) to them, the agents belief set will change, causing a possible reaction or change in the behaviour of said Agent. Behaviours are actions attempted by the Agents.

For this particular framework, we used JADE (Java Agent DEvelopment Framework) to model our agents. JADE is a Java middleware framework designed to handle and run agents developed from a package given by JADE. JADE also handles running the agents platform along with all the communication.

Design

To start with, we took our base ideas from another RTS AI framework, BWSAL. However, this is developed in C++ and is not Agent-Based. The idea behind using an Agent-Oriented design was to offload reasoning for the RTS game to individual Agents that only cared about one facet of the overall problem. We split the problem into 7 different agents as follows:

- *Commander Agent* : concerned with keeping the information up to date and controlling current reactive build orders.

- *Building Manager Agent* : concerned with placement of future buildings and builds structures on demand
- *Structure Manager Agent* : concerned with training and upgrading used by any building in control of the AI
- *Battle Manager Agent* : concerned with all combat matters including scouting
- *Resource Manager Agent* : concerned with all resource gathering, including "rate of" gathering
- *Map ManagerAgent* : concerned with maintaining an up to date map and efficient path finding for any units in AI control
- *Unit Manager Agent* : concerned with mainting unity between all the pieces used by the other Manager Agents and merging together unit commands for Commander

In addition to the above agents, we also developed one, *Proxy Bot Agent*, that was in charge of interfacing with the BWAPI and socket information from the StarCraft game and converting that information into messages the Agent System could understand. It also converted messages from the Agent System to something the BWAPI and socket information understand.

Most of our ManagerAgents follow a similar behavior to each other. Due to this, we were able to create a superclass KhasbotAgent that took care of most of the repetitive tasks.

With many agents working on seperate areas, the key here is organizing their communication with each other. Luckily, all of our protocols exhibit a request/inform type flow. All manager agents are informed of game updates by *Commander Agent* and all Manager Agents request units and inform new commands to units to *Unit Manager Agent*.

The actual Starcraft Game is mapped to a *GameObject* that the agents use to keep track of what they are doing. As the game progresses, the MAS continues to get game updates via a *GameUpdateObject*, which is essentially a reduced game object. The *GameObject* is broken down into two different things, Player Information and Map Information. Map Information includes stuff like where is it safe to build or walk and what the starting locations are for the map in the game. Player Information handles all units and ways the units can do research or upgrades. These units include structures, mineral patches, and all other pieces in the game that can change.

Implementation

The basic idea that each Manager Agent follows is that they are each an agent inside of JADE that have multiple threaded behaviours that focus on different protocols between the various Manager Agents.

Due to the multiple behaviours at once, each Agent heavily used the DataStore ability from JADE to keep its information consistent with the other behaviours.

Results

Conclusion and Future Work