Artificial Intelligence:
Crowd Dynamics using Particle Swarm Optimization

Keith Ainsworth
Computer Systems Research Lab
2007-2008

Abstract

Artificial Intelligence has for long been an important aspect of computer science, but unfortunately artificial intelligence is usually computed from a single agent perspective or with multiple, but highly omniscient agents. I plan on creating an artificial intelligence engine, which works by having multiple agents, each with highly limited perspective. In order to solve tasks, they need to communicate their portions with each other through a network. Using that scheme, it will much more accurately simulate crowd dynamics, using particle swarm optimization to optimize the calculations.

Methods

To program this engine, along with the accompanying game (for graphical output reasons) I'll need C++ (and therefore the g++ compiler) along with the SDL (software digital layer) libraries, for keyboard input and graphical output, and I'll be using OOP programming (therefore the gcc compiler won't be sufficient) and PSO for optimization.

Eventually I will apply this AI engine to a game I've already programmed in Java. Using this game with enemy AI, there will be a very clear visual display as to whether or not the engine is effective and fast. If the game runs smoothly and is challenging, I'd consider it a success.

I've programmed this project so far with several debugging features and text based outputs for constant error checking. While for the final project these would be commented out for the final compilation, I plan to continue programming with those features to allow for ease of code writing and testing.

In the final project, the ultimate judgment will come in the form of randomized testings. For humans controlling the AI objective will throw constantly changing conditions at my AI engine. If the engine can dynamically adapt to any situation given to it by a human player, and maintain an effective run speed then it can be considered effective. However, until the engine has been implemented in the game, testing will continue to come from tester programs, which feature extensive data outputting and try the trick cases and perform bulk testing to make sure the classes and methods being tested are functional.

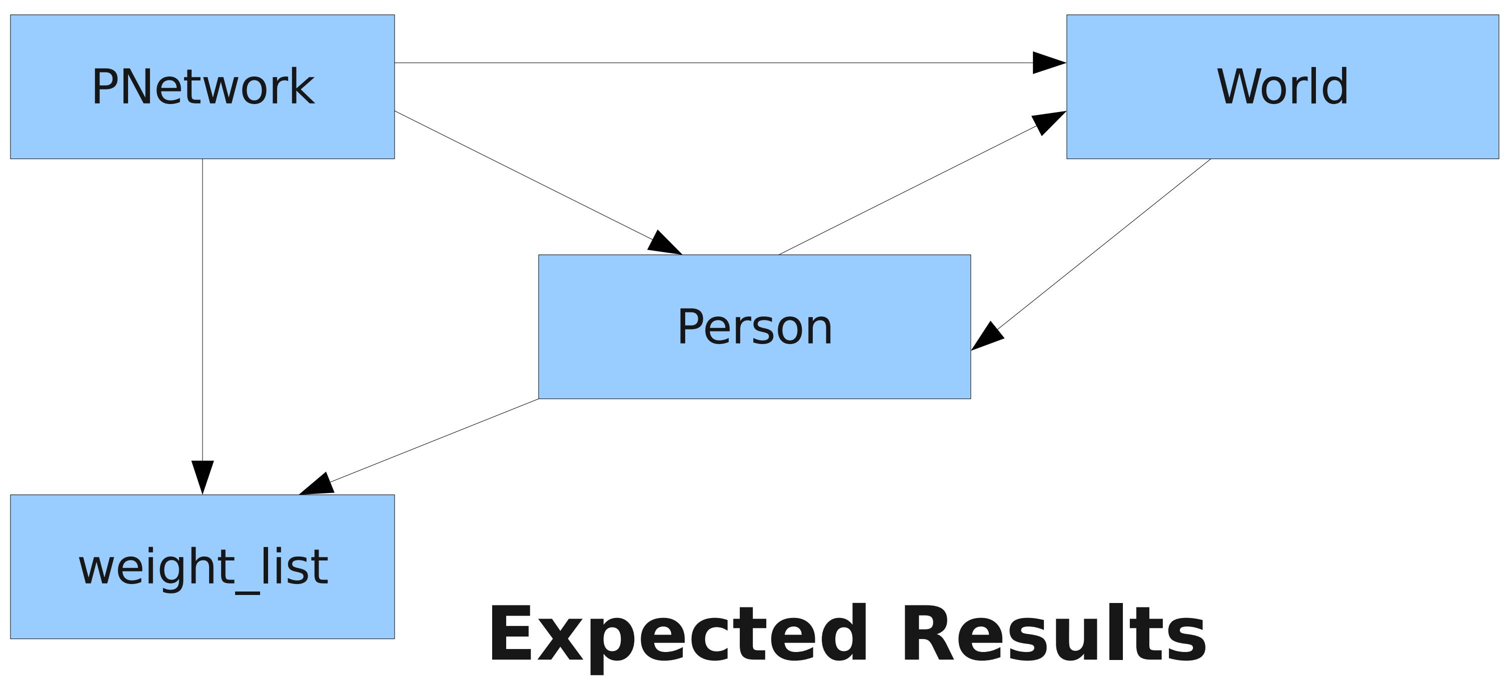
This program relies heavily on object oriented programming and function pointers, two fairly involved programming tasks. I've done many demo programs focusing on these very issues. Early on I ran into a roadblock attempting to create two simultaneously co-referencing classes. I solved that issue through template classes. I will expect to have to solve many similar issues in the future in the same manner.

Introduction

The AI engine I'm programming is implemented through C++'s object orientation. I have programmed several classes which interact in order to make a completed end project. As my engine is an agent based networking engine, naturally the first two classes are the agent, Person, and network, PNetwork classes. The main program must include an array of Persons, which is passed to the PNetwork class on instantiation. Then the main program only needs to talk to the PNetwork, as its managing the list of people from instantiation on, and will take care of the movement of the Persons.

The Person and PNetwork also utilize another class I've written, the weightlist class. This class is a set of two array based, fixed size, looping lists; one for data, the other for the data's relative weighting. The important feature of the weightlist that other pre-written container classes don't offer is a summation function. This function effectively averages the data list, based on the relative weightings, and a decay weighting that favors the more recent entries in the list. This is crucial because the communication aspect of the PNetwork has to have a way of keeping track of each Person's communications. Therefore each Person in the PNetwork is assigned a weightlist. When the PNetwork is called to iterate, it calls the communicator functions in all the Persons, and adds the message to the weightlist of every other person, assigning the weighting based on the distance the message had to travel (the distance between the two agents). Then each weightlist is summed and the results are given to their respective Persons as their new prospective direction.

Class Diagram



Expected Results

This should create a real time multiple agent based, crowd dynamics computing artificial intelligence engine, which will be applied to a game to create realistic simulations of groups of people.

Seniors next year attempting to create a program that utilizes different artificial intelligence schemes, could use this as one of them to compare and contrast effectiveness and speed.