FLokk of Kurious Kreatures

Stuart Tett
stett@vrac.iastate.edu

Andres Reinot
areinot@vrac.iastate.edu

Daniel Shipton
dshipton@vrac.iastate.edu

December 11, 2004

Abstract

The FLokk of Kurious Kreatures (FLOKK) library provides an API for developing applications with several autonomous agents acting in communication with each other and their environment. FLOKK uses programmer defined classes to map out a hierarchical task network to control the flokk of autonomous agents.

Contents

Chapter 1

Introduction

1.1 Problem Statement

Many real-time applications such as games, simulations, and visualizations involve large groups of autonomous agents. Large groups often need to perform unscripted actions in real-time with very good performance. Agents in large groups need to interact with their environment and their fellow agents. Agent groups need a network to communicate goals and provide each of their agents with information needed to obtain collective goals. All of this agent behavior should be done by the agents autonomously, so that the developers can introduce a group of agents into an environment to simulate realistic behavior.

Games, simulations and visualizations that require such collections of agents are typically programmed in C++. To our knowledge, no such tools are available that provide an organizational scheme to fulfill the needs of these real-time applications. Some tools do exist to deal with specific requirements of autonomous agents, but none exist that provide an interface for dealing with many possible requirements.

Some developers have ventured to make applications that do simulations of artificial life. These, however, are specifically geared toward biological simulations. Biological simulation applications are just a subset of the type of real-time applications that would benefit from a system of multi-agent management.

Other types of simulations are limited to only dealing with motion among the multi-agent groups. These systems do not offer flexibility for systems which have many different goals and a dynamic environment affecting these goals.

In current real-time applications communication among a complex hierarchy of agents is often left to the developers. This extra time spent developing communication is costly. An Application Programming Interface (API) to encapsulate agent behavior would be beneficial in such situations, allowing developers to concentrate on the project overall and not how those goals are distributed among the agent network.

The FLokk of Kurious Kreatures library is a solution to the need for a

management system for autonomous agents in real-time applications. Before explaining how it works, the artificial intelligence concepts behind the system need to be addressed. Chapter 2 explains the design of the FLOKK library. For more detail on the use of FLOKK, consult the FLOKK API Reference Manual.

1.2 Artificial Intelligence Concepts

1.2.1 Continuous planning agent (CPA)

FLOKK treats agents as **Continuous planning agents**. A CPA creates new goals and reacts to changes in the environment in real-time. At each step the agent checks its available percepts. Then the agent checks the preconditions of the next action. If this precondition does not hold, then the next action must be changed. Say for example we have two CPAs playing a game tic-tac-toe. Each agent has an initial plan of where it wants to place its mark. If agent A places its mark in a square and B's plan was to place its mark there, then B must change its plan. B must execute a different action (place a mark in a different square). A precondition of the action of placing a mark in a square is that the square is currently unmarked.

1.2.2 Hierarchical task network (HTN) planning

In addition to handling individual problem solving, FLOKK deals with the hierarchical structure desirable in a autonomous multi-agent system. A HTN allows problems to be passed down through the hierarchy of agents.

Artificial intelligence planning in general works like this: an agent's percepts provide it with information about its environment. Using knowledge of the environment and knowledge of available actions, an agent can achieve its goals through AI planning. The plan usually gives a network of actions to perform to achieve the goal.

HTN planning is a specific type of AI planning which divides goals into sub-goals and actions into sub-actions. For example, a goal might be to get from point A to point B. There are different actions we could take. For instance, an agent might have the option to walk or swim to point B. Using its percepts, the agent detects either that the path is all solid ground, all water, or some water and some solid ground. HTN planning would deal with these constraints and possible interactions between the actions.

FLOKK uses HTN planning by setting up goals for the flokk and passing information to its agents in order to achieve its goal. Each agent knows only the information that it's parent flokk tells it. This may or may not include the ultimate goal of the flokk.

Say, for example, that the flokk is an army of soldiers. The army's goal is to take over a city. The army has a complex hierarchy of divisions. Lets say that the army allows all its descendants to know that they are taking over the city. A particular platoon might be given this information along with the subgoal

of capturing a particular building. The platoon figures out how to achieve this goal, and by further division of the goal, gives instructions to individual soldiers to carry out tasks.

Chapter 2

The FLOKK Library

2.1 What is a flokk?

The term **flokk** refers to a flock, herd, school, or any term referring to a collection of autonomous agents. An **agent** is a member of a flokk. An agent is (in our case) an animal, human, or some dynamic character. The idea of the flokk is to control multiple agents that interact together. Examples include a herd of sheep, pigs in a pig pen, birds in flight (in formation), or an army of soldiers.

2.2 Basis

The FLOKK Library is based on the AI concepts of Hierarchical task network planning and Continuous planning agent (Section 1.2). FLOKK uses these concepts to address the problem statement in Section 1.1. FLOKK is written in C++ and uses the General Math Template Library (GMTL) to handle the math required for movement within a three-dimensional space.

FLOKK provides the programmer with an API to develop an autonomous multi-agent system. FLOKK provides a set of abstract classes which developers can subclass to customize the flokk. Developers create their own set of problems to be distributed through the flokk hierarchy. The developers define how to solve problems and on what conditions they are considered solved. FLOKK works out the distribution of problems and communication among the agents.

FLOKK also provides concrete classes to deal with motion solving and collision detection, since these are common among most of the applications to which the problem statement (Section 1.1) refers.

2.3 Agent

2.3.1 Characteristics of an Agent

Each agent has characteristics which depend on the current problem of the agent. A problem, defined below in detail, determines an agent's behavior as the agent tries to solve it. Possible characteristics of an agent include the size of the agent geometry, the distance to maintain from other agents, the position of the agent in the environment, or the velocity (speed and direction) of the agent.

2.3.2 Structure of an Agent

An agent has a list of problems it needs to solve. For example, an agent might solve for the path to desired solution given a problem. An agent asks its parent for any available problems to solve. If the list of problems is empty, the agent solves its default problem (i.e., a problem defined by the programmer for an agent to solve when in an idle state).

2.3.3 Locational Agent

FLOKK defines a special type of agent called a locational agent. Locational agents solve for their desired location when given a locational problem. A locational problem simply refers to a problem in which an agent needs to move in the environment. Since practically any graphically represented system will need agents to move, FLOKK already has it built in. Locational agents know how to move among other agents and obstacles in the environment. They detect collisions and possibly try to avoid them.

2.4 Flokk

A flokk is a specific type of agent: an agent composed of many agents. Thus, the environment treats the flokk as an single agent. Since the flokk is actually many agents, it must handle them itself as its own "sub-environment." Note that because a flokk is an agent it inherits the characteristics and structure as noted in Section 2.3.

A **motion flokk** is a flokk that holds locational agents as defined in Subsection 2.3.3. The idea is that a motion flokk already has the intelligence to handle moving agents. It does this using a motion solver that FLOKK has built in. The motion solver deals with motion conflicts such as collision detection. It also aims to deal with path finding, and spline interpolation.

2.5 Problem

A problem causes an agent to behave in a manner in which the problem is solved. The problem could be that the agent needs to move to a certain position. Then

the agent solves the problem by getting to that position. Various other problems can be defined which may result in movement or yield some other result.

The idea is to make the flokk behave how the programmer wants. So the programmer defines problems and how to solve them. Problems may be given to an agent in one of three ways: the flokk assigns a problem to one/many of its agents, an agent asks the flokk for any available problems to solve, or the agent generates new problems as it tries to solve the current problem. A problem, when given to an agent, performs the solving with the agent following the instructions from the solver.

To create problems the programmer creates subclasses of the `Problem` class from the FLOKK library. The programmer overrides the `solve( float dt )`, `reset( )`, and `isSolved( )` methods. In `solve( float dt )`, the programmer defines how the object solves the problem. It takes a `float dt` representing the change in time since the last update. The `reset( )` function resets the problem back to being unsolved.

In a hierarchical system, it is beneficial for a problem to be split into subproblems. A problem splits into subproblems by the parent splitting the problem. For example, if the parent's problem is to get the flokk into a formation, the parent then generates a problem for each of its members. Each of those problems tells the agent exactly where to move.

Chapter 3

Related Work

OpenSteer

OpenSteer is an open source library that helps simulate "steering behavior" for autonomous agents in a multi-agent system. Steering behavior refers to how an agent determines its path for motion, which includes avoiding obstacles. Agents may be any type of moving character. OpenSteer is intended to be used as a toolkit by programmers of games and other simulations.

Although FLOKK is similar to OpenSteer, it is more general. OpenSteer only deals with one requirement that autonomous agents may have. Motion solving is a large part of FLOKK and steering could be simulated with FLOKK, but FLOKK is not limited to motion problems. The idea is that FLOKK can be customized to deal with any kind of problem that a programmer wants to assign to autonomous agents.

Chapter 4

Conclusion

FLOKK solves the problem of a lack of an open source API for creating an autonomous multi-agent hierarchical task network for real-time graphics applications. Much time was devoted to designing the system to allow for flexibility on both the programmers part and the extensibility of the API. Because so much time was devoted to designing the API, design flaws were few. Thus, the implementation closely follows the original design.

FLOKK began as a solution to dealing with flocks of animals in graphical real-time applications, but grew to be much more flexible. In non-graphical applications, developers could subclass the abstract classes to do data simulations using the hierarchical task network FLOKK provides.

4.1 Future Work

The motion solver still needs work. Collision detection is close to being fully implemented. Some rough motion still occurs when an object changes direction. Also, other real-world physics such as gravitational force may become a part of the motion solver. The samples library will grow to include examples with simple graphics to clarify the use of the API, and also examples with more complex structure (including a flying flock of birds, and an army). More extensive testing of the system will help improve the algorithms used. For instance, testing with very large flokks will determine the limits of the system and possibly aid in the improvement of the library.

4.2 Division of Labor

Stuart Tett -

- API design
- Website design and maintainence

- Documentation
- 3D modelling of fish and fishbowl for fishbowl sample
- Programming problems for fishbowl sample
- Presentation slides

Daniel Shipton

- API design
- API Implementation
  - Motion solver
  - Collision detection
- Documentation
- Build System and CVS setup/maintainence

Andres Reinot

- API design
- API Implementation
  - setup domain
- Programming fishbowl sample
- 3D modelling of fish

Bibliography

[1] Russell, S. J. and Norvig, P. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice-Hall International, Inc. 2003.

[2] American Association for Artificial Intelligence (AAAI). *Artificial Life*. `http://www.aaai.org/aitopics/html/alife.html`.

[3] Reynolds, Craig. *OpenSteer Documentation*. November 11, 2004. `http://opensteer.sourceforge.net/doc.html`.

[4] Reynolds, Craig. *Boids*. `http://www.red3d.com/cwr/boids`. September 6, 2001.

[5] Sloman, Aaron. *THE SimAgent TOOLKIT*. 23 September 2004. `http://www.cs.bham.ac.uk/%7Eaxs/cog_affect/sim_agent.html`

[6] X. Tu and D. Terzopoulos. "Artificial Fishes: Physics, Locomotion, Perception, Behavior". Computer Graphics. SIGGRAPH '94 Conference Proceedings, pp. 43-50. July, 1994.

[7] Nau, Dana S. *Automated Planning*. University of Maryland. Fall 2004. `http://www.cs.umd.edu/~nau/cmsc722/notes/chapter11.pdf`

[8] Cavazza, M., Charles, F. and Mead, S.J., 2002. *Character-based Interactive Storytelling*. IEEE Intelligent Systems, special issue on AI in Interactive Entertainment, pp. 17-24.