Sean Holdsworth

Agents

Introduction

This is probably the most complex of the applications in terms of logic but the UI is relatively simple; it displays the agents and allows for alterations in the size of the cell grid in which they exist and the speed at which they move. As with similar applications the animation can be paused by setting the moves per second slider to zero.

The original specification was that agents should move randomly and leave a colored trail behind them. This was relatively easy to implement, the only difficulty being making sure that when trails overlapped the most recent trail overlaid the others.

The idea of agents though is that they should have some independent behaviour within their environment. I therefore decided to have agents of three different colors; red, green and blue. The red agents hunt the green agents, the green agents hunt the blue agents, and the blue agents in turn hunt the red ones. When an agent reaches its target the targeted agent is destroyed together with its trail and a new agent of the same color is created at a random position. Agents have a perception radius of 10 cell widths radial distance. If no other agent, or only an agent of the same color, is within perception distance then the agent will move randomly. If a predator or prey agent is within perception radius then the agent will move so as to move towards or away from the other agents as appropriate. The algorithm that handles this can process multiple agents of different types and select the best move from the viewpoint of that agent. One final complication is that I've arranged things so that agents will not cross their own trails. If an agent finds itself unable to move without crossing its own trail it will stay in position waiting for its trail to decay sufficiently for it to move again.

Assumptions

Ideally agents would be running in independent threads rather than in lockstep with the Swing event loop but that would complicate implementation beyond what would be reasonable in the time available. Therefore I model the environment as class AgentWorld and in that context call the move method of each agent in turn. This call provides the agent with possible cells to move to and details of other agents within perception range.

Notes on UI

The application structure follows very closely that of the previous application with the constituent classes being slightly modified versions of that application. The top level JFrame is a super class of MainFrame. MainFrame then creates instances of two view components: the CellsPanel used to display the agents and the ControlPanel holding the control components All layout is done programmatically using the GridBagLayout class of Swing.

Table of Classes, Interfaces and Files

So as not to have to deal with more source files than you have to when writing and reviewing code I've placed utility classes and interfaces in the same Java source file as the main class that makes use of them. For ease of navigation though, here is a list of classes and the files in which they can be found.

Notice that for this application I have broken the logic into two classes to reflect the fact that agents should be independent within their environment and because it simplifies code structure.

Class / Interface	Source File
class AgentApp (main)	AgentApp.java
class MainFrame (controller)	MainFrame.java
class CellsPanel (view)	CellsPanel.java
class ControlPanel (view)	ControlPanel.java
interface RestartListener	ControlPanel.java
interface SpeedChangeListener	ControlPanel.java
interface SizeChangeListener	ControlPanel.java
class AgentWorld (logic)	AgentWorld.java
class Agent (logic)	Agent.java

Tools & Techniques

The application follows the pattern of similar UI focussed applications with the MainFrame class acting as a controller and allowing loose linkage of the view component classes. The logic of the application is encapsulated in the Agent and AgentWorld classes and the CellsPanel provides a graphical representation of the agents in their environment.

This is the only application that required use of a container class other than a simple ArrayList. Trails behave as FIFO queues with cell positions being added as the agent moves and others being removed as a trail of the maximum allowed length decays. For this purpose the Java Queue class works well.

If an agent is in perception range of other agents of other colors then it must choose how it is going to move rather than just moving randomly. It uses the following evaluation function as a method of the Agent class, scoring a move high if it is a move towards prey and low if it is towards a predator.

```
    double evaluateMove(int destination, List<Agent> nearby) {
        double value = 0.0;

        int x = destination % cellsPerSide;
        int y = destination / cellsPerSide;

        for (Agent other : nearby)
            if (isPrey(other))
                value += 1.0 / other.distance(x, y);
            else
                if (isPredator(other))
                    value -= 1.0 / other.distance(x, y);

        return value;
    }
```

This can then be used to examine all possible moves and select the best move. Note that often there will be multiple possible moves with the same score, in which case a destination is chosen by random from those positions.

```
    int chooseBestMove(List<Integer> moves, List<Agent> nearby) {
        ArrayList<Integer> bestMoves = new ArrayList<>();
        double bestValue = 0.0;

        for (int destination : moves) {
            double value = evaluateMove(destination, nearby);

            if (bestMoves.size() == 0) {
                bestMoves.add(destination);
                bestValue = value;
            }
            else if (value > bestValue) {
                bestMoves.clear();
                bestMoves.add(destination);
                bestValue = value;
            }
            else if (value == bestValue) {
                bestMoves.add(destination);
            }
        }

        return bestMoves.get(random.nextInt(bestMoves.size()));
    }
```

Critique

This application was a step up from the previous ones in terms of the complexity of the logic involved whilst reusing most of the UI code used in the previous two applications.