IT3708 - Sub-symbolic AI Methods: Project 1

Guillaume Felley

February 2016

1 Introduction

This report is the result of my work for the first project of the course Sub-symbolic AI methods. The main objective was to simulate flocking behaviours of several boids moving together ruled by specifics forces in an environment. Secondly we had to add features like predators an obstacles. I will discuss first on technicals aspect like implementation method and architecture. Then I will describe how the different forces are calculated.

2 Implementation Architecture

This project was implemented in the only programming language that I know well enough: C++. I used in my whole project the Qt library. It's a very complete library that I used for almost everything: Windows, GUI, canvas, animations and 2D vectors.

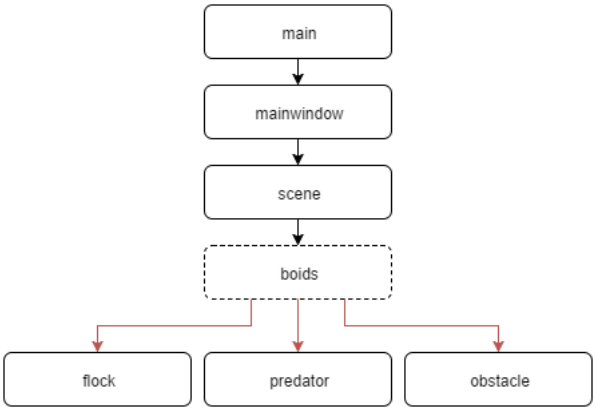

Figure 1: Architecture

My is project is composed of 7 modules : A simple main module. A main-window module where the GUI is displayed and from where the different signals from the interface are manage including the main timer for the refreshing of the display. A scene module where all the graphics are draw and where boids are manage and update. A boids module, an abstract class which is the base class for the flock, predator and obstacle module. So basically i will call a boid either a predator, an obstacle or a flock. On figure 1 we can see a simple dependence graph. The red arrow show an inheritance relation.

3 Forces

The following forces are calculated for each boid considering its neighbours. It mean that a specific neighbour radius is specified and a loop will look for the neighbours of the evaluated boid. This list of neighbours will be given as a parameter to the following functions.

Cohesion : The average position is calculated among the neighbours. Then a new vector is define from the boid position to the previous average position it is then weighted by relative force coefficient define in `constants.h`. We finally return this vector. Note that weight coefficient settable by the user come later.

Alignment: The average speed is calculated among the neighbours. Then we weight this vector with the relative coefficient and return it. The coefftient settable by the user come also later.

Separation: Here we will look among the neighbours for boids in the separation radius. This separation radius is smaller than neighbour radius and the user can define its size as the percentage of the neighbour radius through the `separationWeight` coefficient. If some boids are in the separation radius we calculate their average position and define vector from the average position to the position of the evaluated boid. We then weight it with a relative coefficient and return it.

Escape force: This is the force on the flock when there is a predator around. It's simply a vector directed from the predator to the flock evaluated weighted by the corresponding relative coefficient.

Predator force: We look for the closest flock in the neighbour radius and we take the weighted vector from the predator to the closest flock.

Obstacle force: This force is define in the abstract class boids so it's available for both predator and flock. This function take in parameters a list of obstacles in the neighbourhood. It will return a force if the boid is to close to the obstacle and considering if it's a big or a small obstacle. We take the vector from the obstacle to the boid $\overrightarrow{T_{ob}} = (x, y)$ and we transform it as follow to get the force : $\overrightarrow{F_{ob}} = (y, -x)$ This vector will be tangent to the obstacle. And finally we weight this vector by a relative coefficient very high because we don't want the boid to hit the obstacle.

For every kind of boid the speed is calculated as follow : `speed = speed + acceleration()` the acceleration function is a pure virtual and is implement differently for each kind of boid. For a flock when there is no predator around we have : `acceleration = separation() + cohesion()*cohesionWeight + alignment()*alignmentWeight + obstacleForce()` note that there is no separationWeight here because it's already include in the calculation of the separation. When there is a predator in the neighborhood the flock have to run

away and previous rules of flocking are not applied anymore: `acceleration = escape() + obstacleForce()` For the predator the implementation of the acceleration will be simply the previous predator force and for an obstacle the acceleration is obviously null.

4 Results

When we run the simulation with 100 flocks we can see clearly a good flocking behaviour. When we modify the weight coefficients trough the sliders we can see the change in the behaviour. The 6 different scenarios describe in the compendium have been successfully tested. The behaviour with the predator and obstacle is as expected with maybe one little flaw: When a group flocks have to get around a small obstacle they should split in two group on each side of the obstacle and merge after the obstacle. But instead of merging they stay in two different group.

5 Conclusion

This project was a first challenge for me as I don't have an IT background, I'm an international student studying mechatronics. I only programmed in C before and so OOP is a bit new for me. I probably have a lot of bad habits when it comes to code so that's why I would rely on a good feedback to improve for the future assignments.