FULL SAIL
UNIVERSITY.

Flocking

Table of Contents

Overview

Steering behaviors are algorithms that provide autonomous objects the ability to navigate their surroundings in a life-like or improvised manner. This lab will cover the three main algorithms used in flocking, a type of steering behavior. These three algorithms are alignment, cohesion and separation. Before jumping into the workings of the algorithms themselves however some background information on the application you will be utilizing to perform these behaviors is in order.

Classes

The following are some of the classes important to the flocking algorithms. This section may not list all the methods or members of these classes. For more information on a particular class please refer to its implementation.

Flock

This is the data structure that will represent a single group of **boids**, or steering agents. In this project, the boids are ship-like objects.

Important Properties:

`public float AlignmentStrength, CohesionStrength, SeparationStrength`

These properties represent the strength weights applied to each of the three simple behaviors that constitute the flocking behavior. These properties will be set by the user interface and should range between 0 and 5.

`public Vector3 AveragePosition, AverageForward`

Use these properties to hold the average position and direction vector of each boid respectively. You ***MUST*** use the AveragePosition property as it is used by derived classes (for example, to spawn new ships.)

`public float FlockRadius`

This property should be used in your cohesion algorithm to limit cohesion influence in certain situations and is editable in the user interface. For more specific details on its use please view the cohesion algorithm in the todo list.

MovingObject

This is the class representing all steering agents in this application. Each ***Flock*** consists of one or more of these objects or derivatives of these objects. This class is likened to a **boid** in more generic flocking algorithms. Please note that the terms moving object, ship, and boid are used interchangeably throughout this handout as they all refer to the same thing: an instance of the MovingObject class.

Important Properties:

`public Vector3 Position`

This property is derived from the ***BaseObject*** class and provides access to the location of an object (such as a boid).

`public Vector3 Velocity`
This property provides access to the velocity vector that determines the direction and speed of movement for the ship.

`public float SafeRadius`
This property represents the distance at which collision is more likely to occur between moving objects (such as ships / boids). Use this property in your separation algorithm. For more specific details on its use please view the separation algorithm in the todo list.

Important Methods:

`public Vector3 Update()`
This method is responsible for updating the position of the object. It prevents the magnitude of velocity from reaching values higher than the ship's max speed and uses the simple x' = x + vt algorithm to update the position.

Todo List

Flock.Update()

This is the only method that requires implementation for this lab. It is recommended that you use helper methods for each of the simple behaviors (alignment, cohesion, separation) and for the calculation of averages. The following are the steps needed for a flock update:

1. Compute the average position and velocity of all boids within the flock and store these values into the AveragePosition and AverageForward properties respectively.

2. For each boid (ship) in the flock:

    a. Compute the sum of each simple behavior result (algorithm for each detailed below)

    b. Scale this sum by the product of the boid's max speed and the change in time

    c. Add this sum to the velocity of the boid, limiting the speed to the boid's maximum speed, and update the boid

Alignment Algorithm

The alignment algorithm is rather simplistic as most of the work has already been done when calculating the average velocity of each boid in the flock. This average velocity will serve as the influencing vector. The only problem is that velocities range in magnitude from zero to the boid's max speed whereas we require an influence vector ranging in magnitude from zero to one. This is solved by simply scaling the average velocity vector by one over the boid's max speed. You must also insure that your influencing vector does not exceed unit length before applying any strength weights.

Cohesion Algorithm

The influence vector for the cohesion algorithm is a vector from the boid's position to the average position of the flock. As an additional step we can limit cohesion's influence based on distance from the average position. If the distance to the average position is under a threshold governed by the ***FlockRadius*** property then scale the influence vector by the distance over ***FlockRadius***. This will prevent cohesion from greatly influencing our boids when they are already close to the average position. This scaling should happen after you insure the influence vector does not exceed unit length. Don't forget to apply your strength weight at the end as well.

Separation algorithm

The separation algorithm consists of calculating a sum of vectors away from all other boids in the flock within a distance threshold that serves as a potential collision check. This distance threshold is calculated similar to when doing sphere to sphere collision; add the radius of both boids together. In this case it's the ***SafeRadius*** property of both boids that is used. If the boids' distance to each other violates this threshold then you should scale a unit vector in the direction away from the other boid in a manner similar to the cohesion algorithm. Only this time instead of the influence being less the closer we are; we want the opposite of separation having less effect the farther away from the other boid we are ((threshold – distance) / threshold). The sum of these vectors will become our influencing vector for separation. You must also insure that your influencing vector does not exceed unit length before applying any strength weights.

Turn-in Procedure

You will upload a compressed (Zipped) file named *<lastName>.<firstName>.FlockingLab.zip* which should include:

StudentAI (folder)
        \
        Flock.cs

If you finish this lab during the lab period, please ask a lab instructor for a check-off. This is the only lab in this class which has a check-off procedure.

Other References

Wikipedia: http://en.wikipedia.org/wiki/Flocking_(behavior)

Craig Reynolds: http://www.red3d.com/cwr/steer/