A. Skraastad

PROJECT 1

IT3708

Swarm intelligence

Spring 2016

NTNU

Norwegian University of Science and Technology

Table of contents

1 Architecture and implementation

1.1 Architecture

The application is written in Java, as a JavaFX (MVC) application. The boids inherit from an abstract base class named *Physical*, which hold common properties like positional coordinates, radius and object ID. The boids, as well as an *Obstruction* class, inherit from the base class.

Predators inherit their properties from the *Boid* class, and applies appropriate overrides where necessary. This includes force calculation helper methods and the generic *applyForce* method that all movable objects implement.

The *Boid* class itself is structured in a way that it maintains a reference to the application controller (BoidController), and an *applyForce* method. The applyForce method delegates the different force calculations to different helper methods that calculate alignment, separation, cohesion, collision avoidance and fleeing from predator forces respectively.

All objects that deal with forces use a custom *Vector* class, that implements common methods such as addition, subtraction, multiplication, dot and cross product, as well as normalization, rotation and angular calculation.

1.2 Force calculation

All the different force categories are normalized before being applied to the boid's velocity vector. After normalization, the vector is multiplied by the value of the slider control in the GUI.

The boids *applyForce* method aggregates the different forces using the following steps: Calculate each individual force vector and apply it to the current velocity vector. Normalize the velocity vector before multiplying it with the velocity modifier from GUI slider control value. Finally apply the velocity vector to the position.

Separation is calculated by applying summing up force vectors from neighbors to the boid in question. The individual vectors are weighted by distance using the inverse square law before normalization. This results in a vector towards a point in space of lowest density as seen from the boid.

Alignment is calculated by calculating the average velocity vector of all neighboring boids.

Cohesion is calculated by calculating the positional sentroid of all neighboring boids, and creating a vector from the boid in question to that point.

Collision avoidance is calculated by using trigonometry. I determine the leg of a triangle formed by a line from the boid to center of obstacle, and a perpendicular line from obstacle intersecting the current velocity vector. The velocity vector representing the hypothenuse. If the right edge is shorter than the radius of the obstacle (plus a small buffer), we will have a collision. I then apply a force perpendicular to the line from boid to obstacle, on the side with the greatest angle between said line and a given 90 deg rotation of the velocity vector. The force magnitude is then weighted by distance.

Fleeing behaviour is calculated by applying a force vector from predator to boid, weighted by the distance, before being scaled by the *Fear* control slider in the GUI.

Hunting behaviour of predators re-use the cohesion, separation and obstacle avoidance from the boid class, as well as applying a small single target force vector from predator to closest boid. Cohesion uses boid flock, separation uses other predators.

2 Emergent behaviour

Scenario 1
The boids group on top of each other and the groups of boids gradually merge into larger and larger clusters. After some initial jittering the ball assumes a common trajectory and moves in unison as they merge into larger groups, given that there is at least a slight alignment weight.

Scenario 2
The separation and cohesion forces are in equilibrium, but the weight of the alignment force is so great that almost all boids move in unison across the map, with small jittering internally in neighbor structures.

Scenario 3
Depending on how low the alignment is set, the boids may be in standstill, each maintaining a position maximising the median distance from each other. A slight upward adjustment of alignment will maintain formation, but the boids will move in a common direction.

Scenario 4
The boids move in tight balls but with a common direction. In time they tend to group together in larger balls, still maintaining a common direction.

Scenario 5
Inter-boid distance is still in equilibrium, but since the alignment is low, directional jittering inside dense clusters occur. This prompts rapid separation from the flock, causing single boids to frequently disconnect and connect with dense areas.

Scenario 6
The great advantage of the separation force over the cohesion force cause the boids to attempt to assume optimal distance to each neighbor, while still maintaining a common direction. This gives rise to large scale wave patterns as influences propagate through this map-wide configuration.