Exercise 1
IT3708

Simon Borøy-Johnsen
MTDT

February 3, 2016

1 Architecture and implementation

The simulation was implemented using Java. Java FX was used to form the graphical user interface.

A hierarchy of different objects was used. On the top of the hierarchy is the WorldObject. This only has a Cartesian position, a radius, and some methods to calculate distances to other objects.

The Obstacle class inherits directly from the WorldObject and doesn't have any other functionality.

Below the WorldObject is the MovableWorldObject. This class adds logic used for movement; velocity, forces (abstract method), and methods for checking whether other objects can bee seen or are hitting the object. A set of neighbouring objects is also added.

The Boid and Predator classes inherits from MovableWorldObject, and only implements the abstract methods for calculating forces and clearing list of neighbours (used when calculating forces).

The forces are implemented as their own classes. At the top is a general abstract Force class. This holds the force's weight, and the method for calculating the force. All the other forces inherits from the Force class.

In addition the force's different weights, another weight is also used when calculating some of the forces. This is the distance weight. This is used for increasing the weight of the closest neighbours. The weight is calculated using the following formula; $1 - (distance/neighbourhood\ radius)$.

2 Forces

2.1 Separation

For each neighbouring boid, the distance vector between the current boid and neighbour is calculated. All the distance vectors are normalized and scaled by the distance weight mentioned above. The vectors are then reversed, and they are all summed to form the total force. This makes the force point away from the most crowded are.

2.2 Alignment

For each neighbouring boid, its normalized velocity is fetched. The velocities are scaled by the distance vector, and then summed. The total alignment force is this sum normalized. The sum is normalized in order to avoid the total force being too large.

2.3 Cohesion

For each neighbouring boid, its current position is fetched. The positions are summed, and then normalized. The normalized sum is then used as the force. The force points towards the center of gravity of all the visible neighbours.

2.4 Obstacle avoidance

The avoidance force is calculated using the position of all visible obstacles and the boid's current velocity. For each obstacle; If the velocity is not in a direction that will cause a collision, the obstacle is ignored. If the velocity will cause a collision, a vector normal the the velocity is used to form the force to apply. The vector is scaled by the distance vector, and then adjusted to point in the best direction in order to avoid the obstacle. The sum of these vectors form the total force.

2.4.1 Fleeing from predators

The flight force is calculated in the exact same manner as the separation force. The distance vector to each visible predator is calculated. The vectors are then normalized, scaled by the distance weight, and then summed to form the total force.

3 Emergent behavior

High = 20 * Low

1. The boids gather in huge, tight flocks, eventually merging into one big tight , wobbly ball of boid.
2. The boids gather into one big stream, all going the same direction.
3. The boids spread out evenly around the map, all going the same direction, moving across the map.
4. Like scenario 1, but less wobbly. The velocities are more even.
5. The boids gather in a large swarm, moving along the map. Like scenario 1, but the boids are more spread.
6. The boids all move in the same direction, in a big stream. Like scenario 3, but not as evenly distributed.