IT3708 Assignment 1- Torgeir Skogen

Implementation

Architechture

The task has been implemented in Java. The source code consists of eight java files, contained in two packages to separate graphics from the underlying logic. For the graphics, I use Java Swing. In the graphics package, I have the files: BoidArea, Main and SettingsPanel. The main will set up a BoidArea and a SettingsPanel and put them in the same frame. The settings consist of buttons to add and remove obstacles and predators and sliders to adjust separation, alignment and cohesion. The BoidArea is implemented as a PropertyChangeListener to listen to events that happen in the settings. The SettingsPanel is listening to its components as a ChangeListener and ActionListener to capture events happening to the different kinds of input mechanisms. The BoidArea keeps lists of boids, predators and obstacles, and draws these in a rectangle of a given size. After being set up, a thread takes care of the main loop, which ticks on all the boids and predators, then paints the area.

In the logic package, I have the classes Boid, Obstacle, Parameters and Predator, as well as the interface Coordinates. The Coordinates interface exists so that a Boid can calculate the distance to and the vector a Boid or an Obstacle without needing a separate method for each one. The Predator class inherits from Boid as well. Each boid has a coordinate pair, a direction in the form of an angle and a velocity pair. The most important method of the Boid class is the tick, which calls methods to change the velocity and direction in separate function calls for separation, alignment, cohesion, obstacle evasion and predator evasion. Then the location is updated. The Boid class also has a method for finding the group it belongs to, for the purpose letting the predators find the group to chase. The Obstacle class is a simple class. Each obstacle simply has a coordinate pair and a radius. The Predator class inherits from the Boid class, so it can reuse a lot of the functionality. The key difference is that the tick method makes the predator follow a group of Boids rather than obeying all of the same rules as the boids. The predators use the same method for separation as the boids, but separates the predators from other predators rather than from the boids. Predators also avoid obstacles with the same method as the boids. The location of the predator is updated in same way as the boids, but the velocity is twice as big. The Parameters class contain only static parameters, some of which are changed runtime via the settings. The purpose of the file is to keep all the variables at the same place, so modifying them is not a hassle.

Calculation of movement

The tick method for the boids takes in the list of all the boids. It will filter out the boids that are too close for the boid to see before feeding this resulting list into the methods that calculate the various forces.

For the boids, the separation is calculated this way. The boid is fed a list of all the boids. It will iterate over all the boids. A unit vector representing the distance from the other boid to the current boid is added to the total in the method. After iteration this total is scaled by the separation weight and added to the velocity vector.

For alignment, the boid will iterate over its neighbours. This method will convert the directions of the neighbours to Cartesian coordinates, add up all of these. Then divide by the amount of neighbours. This will calculate the average direction of the neighbours. The current boid will then adjust its direction to be 0.01 radian closer to this direction. Then a unit vector based on this direction is scaled by the alignment weight and added to the velocity.

Cohesion is similar to separation, but the direction of the vectors is flipped around. And instead of adding unit vectors, the sum of the displacement vectors is divided by the amount of neighbours. The total gets scaled by the cohesion weight and added to the velocity.

To avoid obstacles, the boid will iterate over the list of obstacles. If it is closer to the obstacle than the look range divided by four, then the unit vector away from the obstacle is scaled by a big number then added to the velocity.

If the boid is closer to any predators than the look range, then both the direction and velocity of the boid gets overridden by the unit vector pointing away from the last predator it checked that was close enough.

When updating the location, the velocity is scaled to a unit vector. Then a random number is used to determine whether the boid should move in the respective directions based on the absolute value of the velocity in the direction.

The predators will find the biggest group of boids. Boids are considered to be in the same group if they are linked by boids that are close enough together. This means that if boids A and B are close enough, then all boids close enough to B are also in the same group as A. To find the biggest group, I use a list to keep track on all the boids we have to consider. As long as this list is not empty, we pick a boid from it, use the recursive method for finding the neighbours of that boid, remove all the boids in this group from the search space. If this group is bigger than the group we already found, then we replace the biggest group with this group. The predator then finds the average position of the group, similar to the way boid cohesion works. The vector to the average position of the group is added to the velocity.

Then the predator will reuse the functions for separation and obstacle evasion from the Boid class, before updating its direction to correspond with the velocity and then updating the position. The position is updated the same way as for the boids, except the velocity of the predator is scaled to 2 rather than 1.

Behaviour

Spearation	Alignment	Cohesion	Description
Low	Low	High	Boids group very tightly and move slowly
Low	High	Low	Boids form tight long lines in the direction they are headed
High	Low	Low	Boids spread out with little to no overall movement aside from wiggling
Low	High	High	Boids form tight round groups that move clearly
High	Low	High	Boids form groups that move slowly
High	High	Low	Boids spread out evenly and move in the same direction