IT3708: Project 1 – Report

Author: Petr Zvonicek

Task of this project was to implement flocking behaviour with obstacle avoidance and predators following the non-predator boids.

Implementation

I've chosen Processing as a programming language and an environment to implement this task. Processing is build on the Java language and extends it with graphics programming model, which makes it easy to create programs focused on the visual output.

The program is divided into seven files:

- **Flocking.pde** - the initial ("bootstrap") file, creates the canvas, instantiates new *Flock* object, draws the GUI, handles GUI events and periodically calls the *run* method on *Flock* which redraws the world.
- **Flock.pde –** contains the class *Flock* that manages the flock. The class keeps reference to all boids and obstacles in the world and is responsible to draw them. The core is the *run* method that periodically redraws all the boids and obstacles.
- **Boid.pde** - contains the class *Boid* which represents one boid in the flock. The boid is determined by two vectors: position and velocity. The class handles movements of the boids and calculates the forces in order to compute the velocity. When computed, the boid's position is moved by the velocity with respect to maximum speed limit.
- **Obstacle.pde** - class *Obstacle* represents one obstacle in the world. It's defined by position and radius.
- **Predator.pde** - *Predator* is a subclass of *Boid*, changes some parameters and overrides method *updateBoid* with custom force calculation.
- **GUI.pde** - GUI drawing and event handling
- **Helpers** - some helper methods

Force calculation

Separation

Separation force is calculated as a sum of differences between position of the current boid and positions of nearby boids. This will point the boid away from its neighbors. Separation vector is then normalized.

Alignment

Alignment force is calculated as a sum of velocities of nearby boids, which is then normalized. This will set the force as an average velocity of its neighbours.

Cohesion

Cohesion force is calculated similarly as alignment, but instead of sum of *velocities* there is calculated a sum of *positions*. This sum will be a position of the center, but we want just a direction, so we'll subtract a boid's position from it and do the normalization.

Obstacle avoidance

The core part of obstacle avoidance is choosing the closest colliding obstacle. For this we'll create an ahead vector. Algorithm for circle-line intersect will then check the collision and we'll alternatively choose the closest obstacle. Ahead vector and position of the closest obstacle will be then subtracted and this vector will be normalized.

Fleeing from predators

Fleeing from predators is computed as a sum of differences between position of the current boid and positions of nearby predators. This ensures the boids will escape away from the preadator. The vector is then normalized.

Emergent behavior

Scenario 1 – Separation: low, Alignment: low, Cohesion: high

Boids fly close together, if alignment is too low (<0.3), they got stuck in a circle and not moving around the map ("blackhole").

Scenario 2 – Separation: low, Alignment: high, Cohesion: low

All boids fly in the same direction, they tend to group in a large clusters. They are flying straight and avoid turning.

Scenario 3 – Separation: high, Alignment: low, Cohesion: low

Boids tries to avoid colliding with each other and keeps a big distance between themselves.

Scenario 4 – Separation: low, Alignment: high, Cohesion: high

Boids fly close together in a long clusters ("stripes", "tails").

Scenario 5 – Separation: high, Alignment: low, Cohesion: high

Clusters are mid-large, boids are "not sure" with their direction and are not stable - they stay in a cluster for a short time.

Scenarion 6 – Separation: high, Alignment: high, Cohesion: low

There are big distances between each other, but they still follow the same direction. Boids are almost evenly distributed among the space.