Project 1 IT3708:
Flocking and Avoidance With Boids

Purpose: Implementing flock behavior with boids that avoid obstacles and flee from predators.

1 Background

A flock (or a school or a swarm) is a big group of individual organisms moving together. The behavior they exhibit is called flocking. In general, there is no group leader. There is also no global information, for instance about where they are heading. Each individual is just following what the closest other individuals are doing, so they behave only according to *local information*.

In 1986, Reynolds came up with a mathematical model for this flocking behavior. The model is just a set of simple rules that each individual follows. Reynolds gave objects following these rules the name *boids*, from "bird-oid object" or bird-like object. The three basic rules for a boid are:

- *Separation*, a boid will steer to avoid crashing with other boids close to it.
- *Alignment*, a boid will steer towards the average heading of other boids close to it.
- *Cohesion*, a boid will steer towards the average position of other boids close to it.

These rules are pictured in figure 1. These simple rules are good enough to model something that looks like a realistic flock and has been used for many applications, for instance birds in Half-Life and bats in Batman Returns.

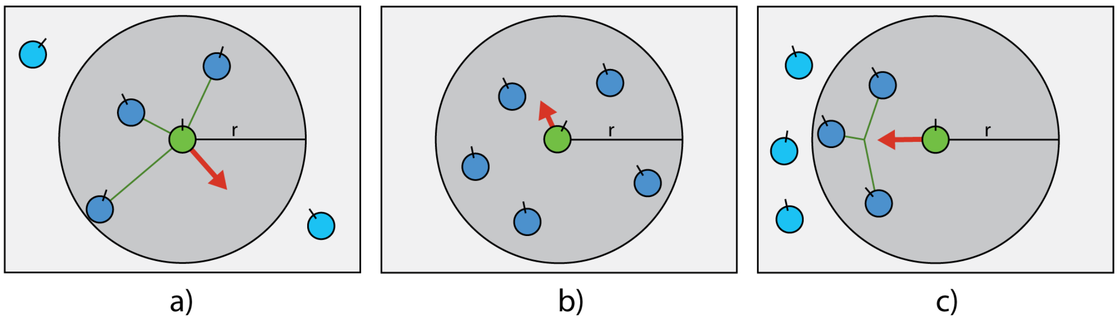

Figure 1: a) separation, b) alignment, c) cohesion

2 Assignment

For this project, you should implement boids flying around in a 2D world.

2.1 Basic flocking behavior

For the basic flocking behavior, you will need to implement the three rules discussed above. Note that all these rules should result in *forces* that will change the velocity (speed in x and y direction) that later will lead to an updated position. Pseudocode for this is given below. Note that most of the variables in the code are 2D vectors.

```
updateBoid (allBoids)
    neighbors = boids closer than r

    sep = separationWeight * calculateSeparationForce(neighbors)
    align = alignmentWeight * calculateAlignmentForce(neighbors)
    coh = cohesionWeight * calculateCohesionnForce(neighbors)

    velocity = velocity + sep + align + coh
    position = position + velocity
```

As you also can see in the pseudocode, the rules are weighted against each other. A requirement is that you should be able to change the weights while your simulation is running, for instance using a slider for each weight.

The world should wrap around, so a boid exiting the world on the right should return on the left. The same applies for up and down. For calculating neighbors for a boid, it is *not* a requirement to handle wrapping. So a boid on the right edge of the world doesn't need to consider a boid on the left edge of the world a neighbor.

For the rendering, each boid should have something showing its direction. A small line as in the figures above is good enough.

2.2 Obstacle avoidance

There should be stationary obstacles in the world which the boids should avoid. An obstacle has a point and a radius (so it's a circle). To implement this, you will need to extend the pseudocode above with a new calculation to get a force to avoid obstacles. It's important for a boid not to collide with an obstacle, so the weight of this force should probably be high.

To get full credit for avoidance it should be smooth, as shown in figure 2a and b. When encountering an obstacle, the boid should not just turn away, instead it should calculate an angle that makes it fly just past the obstacle. Ideally, a flock encountering a small obstacle should split, go smoothly around it on both sides and then the standard flocking behavior should end up grouping the flock back together. This is shown in figure 2b.

It should be possible to add and remove obstacles while the simulation is running. A button that adds a random obstacle to the world and a button to remove all obstacles is good enough.

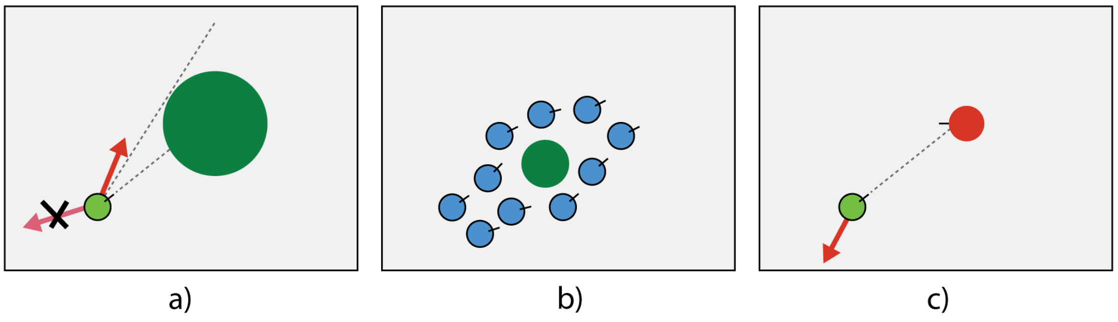

Figure 2: a) avoid an obstacle, b) a flock avoiding a small obstacle, c) fleeing from a predator

2.3 Predators

There should be some predators in the world which the boids should flee from. The fleeing should be desperate and chaotic when a predator comes close. So the behavior should be different from just avoiding an obstacle. This is shown in figure 2c.

For full credit, a predator should not just move randomly around. It should chase group of boids similarly to how boids follow each other. With a good architecture, predators can be implemented as subclass of boids. To check that a predator chases a group of boids, lower the weight of separation for the boids and see how the predator moves.

It should be possible to add and remove predators while the simulation is running. A button that adds a predator to the world with a random position and velocity, and a button to remove all obstacles is good enough.

2.4 Parameters and tips

To run a successful simulation, you will need to spend some time tweaking the various parameters. They are highly dependent on each other, so weights working for someone else may for instance not work if you use a different radius to calculate neighboring boids.

Your simulation should have a couple of hundred boids to properly show the flocking behavior. A good estimate for the neighbor-radius is 5-15 times bigger than the radius of a boid in your simulation. This ensures that each boid is only acting on local information.

It can also be a good idea to add a max limit for velocity to keep the simulation stable. Experiment with a max velocity for predators different from the max velocity of boids.

3 Report

You should write a report answering the points below. The report can give a maximum score of 8 points. Make sure your report does not exceed more than 2 pages. Bring a hard copy of your report to the demo session

- Document your implementation. (5p)
  - Write a short summary of your architecture and implementation.
  - Explain how you calculate the forces for separation, alignment, cohesion, obstacle avoidance and fleeing from predators. Document each one. Don't just paste in the code, explain the idea / general concept.
- Describe the emergent behavior. (3p)
  - With no obstacles and no predators in the world, vary the weights and explain what happens in your simulation. There are 6 scenarios, shown in table 1, that you should run.

Scenario	Separation	Alignment	Cohesion
1	Low	Low	High
2	Low	High	Low
3	High	Low	Low
4	Low	High	High
5	High	Low	High
6	High	High	Low

Table 1: How separation, alignment and cohesion should be weighted for 6 scenarios.

4 Demo

There will be a demo session where you will show us the running code and we will verify that it works. This session can give a maximum of 12 points.

- Show that the basic flocking behavior works as expected. (4p)
- Can change the weighting of separation, alignment and cohesion while the program is running. Can add and remove obstacles and predators. (3p)
- The boids avoid obstacles in a smooth manner. (2p)
- The boids flee from predators. (2p)
- Predators chase the boids. (1p)

5 Delivery

You should deliver your report + a zip file of your code on It's Learning. The deadline is given on the assignment on It's Learning. The 20 points total for this project is 20 of the 100 points available for this class. For this project you should work alone.