Manual: Flocking

Overview

Steering behaviors are algorithms that provide autonomous objects the ability to navigate their surroundings in a life-like or improvised manner using vector calculations and incremental change. This exercise will cover the three ***simple steering behaviors*** – alignment, cohesion, and separation - used in flocking, a type of ***compound steering behavior***. Students will implement each of these component behaviors and then combine them into the flocking behavior.

Structure

The exercise is broken into two main parts:

1) Implementation of the simple steering behaviors – alignment, cohesion, and separation
2) Implementation of the compound steering behavior, flocking, from the component simple behaviors

User Interface

This flocking simulator provides an interface that can be used to see the behavior of multiple flocks at the same time while also allowing their parameters to be changed in real-time (**Figure 1**). In this application, users may select flocks individually and update general settings as well as the strength of the component behaviors. Users may also switch between the example implementation and the student-developed implementation for testing.

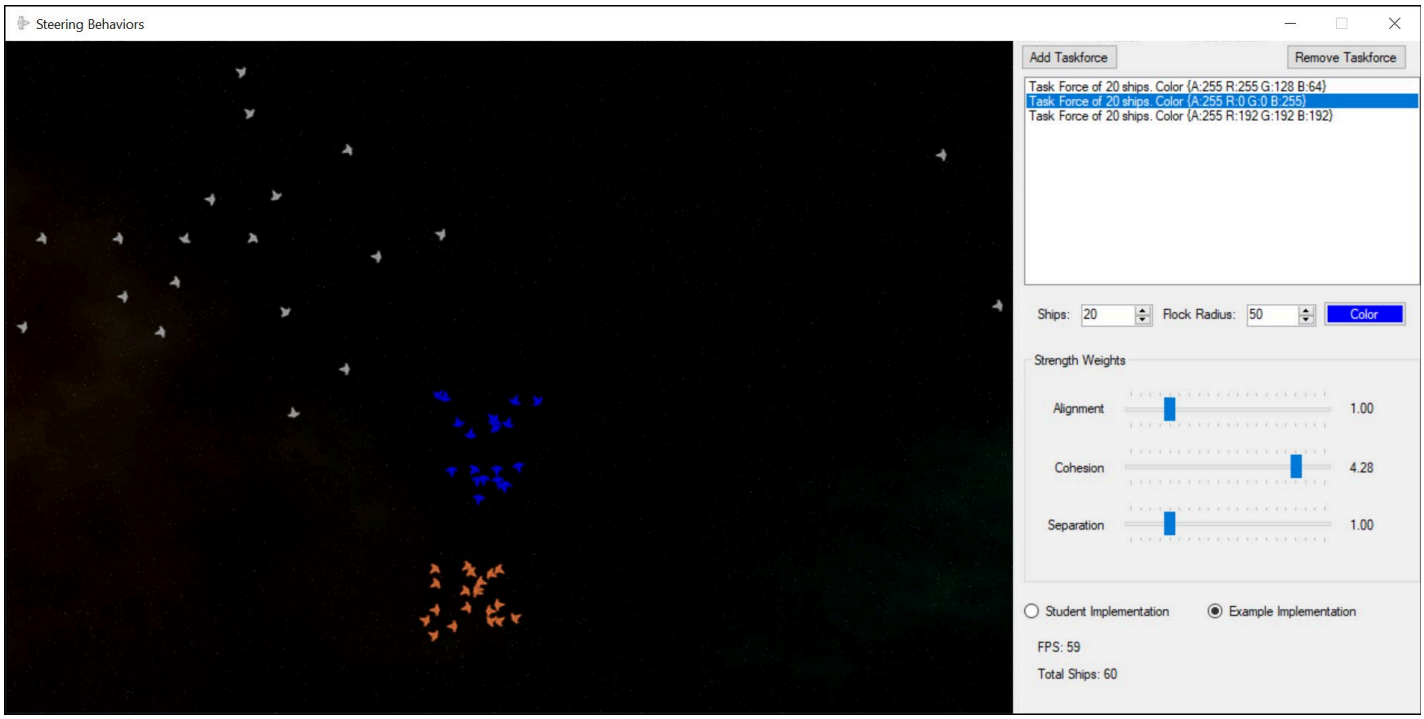

Figure 1. User interface of flocking application; flocks are visualized on the left, with the control panel on the right.

Development Interface

In order to implement the flocking behavior and its component behaviors, students will make use of the class representing agents within the application, which is found in the `AI.SteeringBehaviors.Core` namespace.

MovingObject

Objects of this class represent all agents in the engine (***boids*** in flocking terminology).

Properties

```
public Vector3 Position { get; set; }
```
Derived from *`BaseObject`*; provides the current position of this game object (e.g., an agent / boid).

```
public Vector3 Velocity { get; set; }
```
Holds the velocity vector of the object (which identifies its direction and whose magnitude is the speed).

```
public float SafeRadius { get; set; }
```
When another game object is closer than this distance, a collision is considered imminent.

Methods

```
public void Update(float deltaTime)
```
When called, this method updates the position of the object using its velocity and `deltaTime`. If its speed is above the object's maximum speed, this method reduces the speed to the maximum.

Assignment Interface

Students will complete missing elements of the class(es) in this section as part of the assignment. These classes are found in the `AI.SteeringBehaviors.StudentAI` namespace.

Flock

Objects of this class represent the group of ***boids*** (agents) that constitute flocks or swarms.

Properties (Provided)

```
public List<MovingObject> Boids { get; protected set; }
```
Provides a *`public`* readable (*`protected`* write) list of all agents as *`MovingObject`* instances.

```
public Vector3 AveragePosition { get; set; }
```
Average position of all boids in the flock.

```
public Vector3 AverageForward { get; set; }
```
Average of the velocity (forward) vectors of all boids in the flock.

```
public float AlignmentStrength { get; set; }
```
Alignment behavior strength for this *`Flock`*; updated by the UI.

```
public float CohesionStrength { get; set; }
```
Cohesion behavior strength for this *`Flock`*; updated by the UI.

```
public float SeparationStrength { get; set; }
```
Separation behavior strength for this *`Flock`*; updated by the UI.

```
public float FlockRadius { get; set; }
```
Distance from the center of flock at which cohesion behavior maximizes its influence; updated by the UI.

Methods (TODO)

```
public Flock()
```
Constructor; default implementation provided by runtime, but an explicit constructor is recommended.

```
public void Update(float deltaTime)
```
This method is called frequently to update the flock and its boids. It should:

1. Recompute all average values;
2. Update each boid's velocity based on the influence of the various component behaviors; and
3. Call each boid's `Update()` method (which will update the boid's position).

While only the parameterless constructor and `Update()` methods are required, it is highly recommend that students add additional *private* (not public or protected) helper methods.

Testing

An example implementation of the flocking behaviors is provided for students. It is recommended that students carefully study the behavior of this implementation, as well as the pseudocode implementations for flocking, when implementing the behaviors. The behaviors should remain the same when switching between implementations in the correctly completed application.

Submissions

Students will submit a **zip file** containing the following file(s) at the end of this exercise on Canvas:

- Flock.cs

Place them in the **root directory** of your zip file, not in a subdirectory. Do not submit any other files.