THE FLOCKING DEMO

Overview

This coursework focuses on the development of AI algorithms relating to the flocking behavior of NPC units, and more specifically, implement a Bullet demo illustrating flocking forces. The world should be populated with a large number of identical objects “birds” as well as some obstacles that the birds should avoid while applying physical and flocking forces on them.

Implementation

For the implementation of this coursework I based on the Project Template with forces provided in moodle, as well as on ”AI for Game Developers”, David M. Bourg and Glenn Seemann, and other online sources which illustrate the fundamentals of flocking behavior.

.Methodology

The project consist of the following core files:

- **Boid**
- **flockingDemo**
- **MiscObstacles::Obstacles**

.The world:

The initialization of the word and creation of the obstacles and the boids, lies in the FlockingDemo file. For the creation of the plane, I used the PlaneObstacle from the MiscObstacles file with which I created a 700x700 btBoxShape plane. From the same file, I used the ColumnObstacle class to create the obstacles into the screen. After specifying the column positions, I create 9 different columns on the plane, in positions that will support the flocking illustration.

.The boids:

In order to implement the boids, I used the Boid file provided in moodle. For the model of the boid, I created a btConvexHullShape passing specific vertices to create a polyhedron. The boids are initialized in random positions, using the rand() function for the x and z values with a fixed height of 60.

.The flocking:

In order to achieve a realistic flocking implementation, I needed to apply three different type of forces on the boids, Physical forces, flocking forces and avoidance forces.

.The Physical Forces:

The Physical forces consist of the basic forces needed to be applied on a single boid in order to flock harmonically. These forces are Direction (the normalized velocity vector), lift force (negated gravity force), drag force (negated velocity), angular drag force and thrust. In the Boid::PhysicalForce function I also calculate the boundaries of the world. In order the keep the boid inside specific boundaries, I check the boid's distance from the transform origin and I reflect the direction of the boid on the point it reaches the boundaries. The reflection is calculated by the type :

$$\mathbf{r=d-2(d\cdot n)n}$$

where d·n is the dot product, and n must be normalized. In order to keep the boid inside specific height boundaries, I apply a negative force on the local Y axis of the boid (B frame). In order to complete the physical forces, I apply rotation on the boid according to thrust as well as rotation relating to the angular drag.

.The flocking forces:

The flocking forces include the three basic rules of flocking, separation cohesion and alignment. Based on the flocking techniques introduces by David M. Bourg and Glenn Seemann, I loop the boids and I check for every boid its neighbors based on a field of view. The field of view is implemented in the Boid::InView function, where I define a field with angle of 250 degrees. After getting the neighboring boids, I calculate the sum of positions and directions for each boid. In order to achieve the separation, I apply a separation force on the negated vector created from the distance of the boids' body centre to each neighboring boid. For the cohesion, I calculate the average positions of the flock and apply it on the boid. Simillarly, I calculate the average direction of the flock, and apply it on the boid in order to achieve the alignment behavior on the flock.

.The avoidance force:

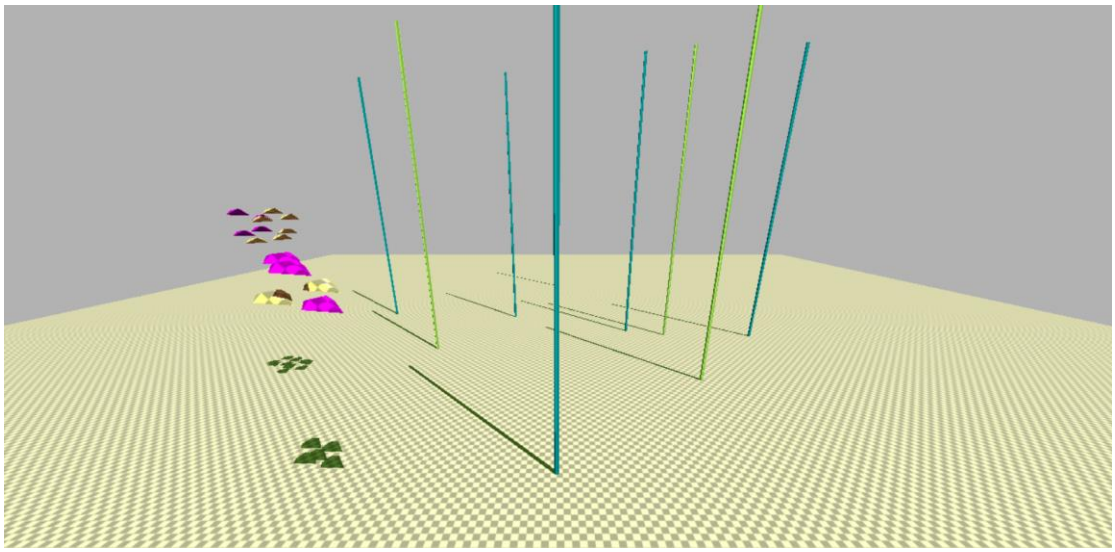
The last flocking rule I implemented on this application, is the avoidance rule. The boids are expected to avoid the obstacles whilst staying aligned and cohesive with in the flock and retaining the force balance. Firstly, when the function AvoidanceForce is called by a boid, I loop over the obstacle vector to see which obstacles are in the Path of the boid. After setting a bounding radius for the boid, I calculate the distance between the boid and each obstacle. When the distance become less than the sum of the radius of the boid and the obstacle, then the obstacle exists in the path of the boid. In order to make the boid turn correctly and retain the realism, I calculate the point where the bounding radius of the boid and the bounding radius of the obstacle will intersect. This point is given by the function:

collisionPointX = ((firstBall.x * secondBall.radius) + (secondBall.x * firstBall.radius)) / (firstBall.radius + secondBall.radius);

collisionPointY = ((firstBall.y * secondBall.radius) + (secondBall.y * firstBall.radius)) / (firstBall.radius + secondBall.radius);

After I calculate the avoidPoint, I reflect the boid's direction on this point, achieving a realistic avoidance effect.

.The result



REFERENCES:

AI for Game Developers: http://www.cse.unr.edu/~sushil/class/381/notes/AIGDch04.pdf

Reflection vector: http://math.stackexchange.com/questions/13261/how-to-get-a-reflection-vector

Calculate circle collision points: http://gamedevelopment.tutsplus.com/tutorials/when-worlds-collide-simulating-circle-circle-collisions--gamedev-769

Template with forces: http://moodle.city.ac.uk/course/view.php?id=16723