Colin Shaw
100628526

CSCI 4110 Final Project

Flocking Simulation

December 13, 2019

Colin Shaw

100628526

Overview

In this assignment I created a 3D flocking motion simulation in OpenGl based on the boid model I used assignment 3 as a base. The boids start in random positions and move around. The boids move by following 6 rules:

1. Avoid collisions with obstacles
2. Avoid leaving the designated area
3. Avoid being too close to friends
4. Move towards the goal position
5. Match your direction with other nearby boids
6. Avoid predators

The program drawing is centered on (0,0,0). The program draws the obstacles and boids in random locations within a specified radius. The colors are chosen in the frag shader.. An int passed to the shader works as an Enum to decide the boid's colour or if it is not currently drawing a boid.

Colin Shaw
100628526

To Build and Run

If you unzip the submitted file and open the .sln file in visual studio 2019 everything should work just fine.

Boid Class

A class Boid holds all the info we need to move a boid.

- posx, posy, posz : three floating points values to describe the position
- basespeed: a floating point used to reset speed after a boid is frightened
- speed: the speed a boid moves at
    - Predator boids have a higher speed
- direction: a vec3 that hold or current heading
    - this value used used for most of our math
- isPred: a Boolean determining if predator or prey
- atGoal: a Boolean to tell if a boid has reached the goal
- nearEnemy: a Boolean that is used to emulate fear of a predator
- getPos(): returns a vec3 representation of our position
- getMagnitude(): returns the magnitude of our position vector

distanceBetween()

distanceBetween() is a helper function that takes vectors and returns the Euclidian distance between them. It is overloaded for convenience.

Update() Function

Colin Shaw
100628526

```
172     //make sure all weight sum to 1
173     float totalWeight = rule1Weight + rule2Weight + rule3Weight + rule4Weight + rule5Weight + rule6Weight;
174     rule1Weight = rule1Weight / totalWeight;
175     rule2Weight = rule2Weight / totalWeight;
176     rule3Weight = rule3Weight / totalWeight;
177     rule4Weight = rule4Weight / totalWeight;
178     rule5Weight = rule5Weight / totalWeight;
179     rule6Weight = rule6Weight / totalWeight;
180
181     totalWeight = rule1Weight + rule2Weight + rule3Weight + rule4Weight + rule5Weight + rule6Weight;
```

These first few lines makes sure the weights sum to 1. The ratio between the weights is maintained. For example if weight 1 is 10 and weight 2 is 40 and all other weights are 0, after this code weight1 will be .2 and weight 2 will be .8.

Next we loop through all the boids. We created a vector for every rule that will store the direction that rule wants us to go. These are initialized with the current direction the boid is going.

We then enter the main part of the update code. If the boid is a predator it run a different block or if it is at the goal it skips all of the movement code since its already where it wants to be.

If the boid is prey and it is not at the goal it computes the rule directions.

Turning (This took me forever to figure out in 3d)

In the following rules turning is determined in the following way.

- First determine if you need to turn.
- Find the cross product of your current direction and direction to object
  - This gives a normal vector that defines a plane
  - Our new direction exist on this plane
- Rotate by a given angle, with the normal vector as the axis of rotation and save this vector
- Rotate by the negative of that angle and save this vector
- Compare these vector with dot product to determine which is more/less similar to the direction we want

Colin Shaw
100628526

Rule 1 Avoid Obstacles

This rule loops through every obstacle and finds the closest to the boid (I previously considered all obstacle but it worked better when only considering the closest). If this obstacle is within the range we care about the boid will turn away, turning increases with the square of the distance to the obstacle.

Rule 2 Avoid Going Out Of Bounds

Since we are centered on (0,0,0) to tell if we are out of bounds we can simply compare the magnitude of our position vector to see if it is less than the radius of our area. If the boid is found to be out of bounds it is guided back to the center.

If a boid is close to going out of bounds it is encouraged to turn just as in rule 1.

Rule 3 Avoid being too close to friends

This rule prevents boid from stacking up on each other when they flock. Just as in the first two rules if a boid is too close to another boid it will turn away.

Rule 4 Go towards the goal

This rule simply encourages the boid to turn to the goal.

Rule 5 Flock with friends

This is definetly the most interesting rule. The boid looks at all nearby boids and tries to match its direction with it.

Rule 6 Run from Predators

This rule encourages a boid to get way from a predator boid in the same way as previous rules. If a predator boid is nearby a “fear” response is activated that increases the boids speed.

Colin Shaw
100628526

TurnRate

The six vectors obtained from the rules are now normalized and combine by a weighted sum to determine the new direction. We then see if this direction is larger than our turn rate. If it is we turn as far as we can in that direction

Predator Boids

The predator boid ignores all the rules and simply tries to follow the nearest prey boid. It also does have its turning speed bounded by turnrate.

Misc.

Next all the boids are moved. If they are “scared” they move faster. Also we check if all prey boids reached the goal and exit the program if they have.

Init and load functions

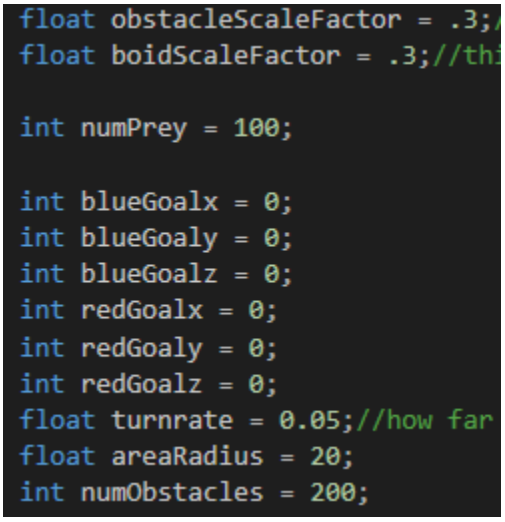
Boids and obstacles are load from obj files in the load functions. Arrays are populated in the init function.

Display()

This function applies the needed translation, rotation and scale transformation to each boid and obstacle and then draws it.

Parameters

Colin Shaw
100628526



```
float obstacleScaleFactor = .3;
float boidScaleFactor = .3;//thi

int numPrey = 100;

int blueGoalx = 0;
int blueGoaly = 0;
int blueGoalz = 0;
int redGoalx = 0;
int redGoaly = 0;
int redGoalz = 0;
float turnrate = 0.05;//how far
float areaRadius = 20;
int numObstacles = 200;
```

These Scale Factor are for visual apeal

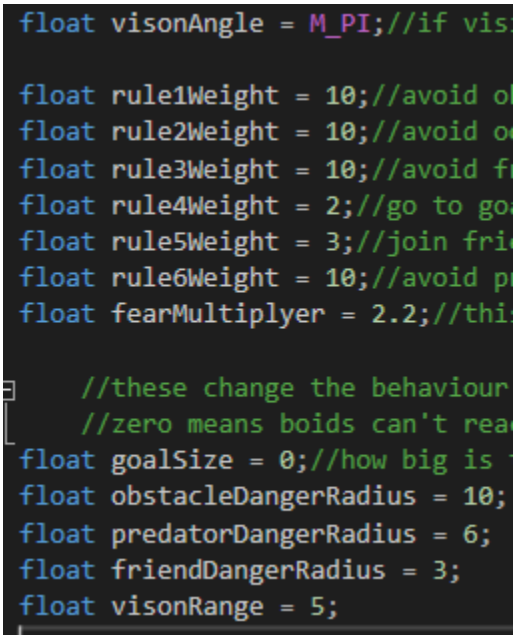
The number of prey boids

Goal locations

How fast the boids turn

The bounding radius

How many obstacles to create



```
float visonAngle = M_PI;//if vis

float rule1Weight = 10;//avoid o
float rule2Weight = 10;//avoid o
float rule3Weight = 10;//avoid f
float rule4Weight = 2;//go to go
float rule5Weight = 3;//join fri
float rule6Weight = 10;//avoid p
float fearMultiplyer = 2.2;//thi

    //these change the behaviour
    //zero means boids can't rea
float goalSize = 0;//how big is
float obstacleDangerRadius = 10;
float predatorDangerRadius = 6;
float friendDangerRadius = 3;
float visonRange = 5;
```

For rule 5 if vision is enabled

The weights of each rule. Changing these can lead to strange interactions

How big is the goal(if zero goal can't be reached)

These are the sense ranges of the boids

When Running

By default the camera will follow the predator boid you can press F to gain control of the camera with mouse. The keys Z, X, C, V, B, N, can be used to toggle runs to see how the boids behave differently. You can also zoom with the scroll wheel. The arrow keys and W S can move the camera but tis is not intuitive and should not be necessary. Finally pressing G will hide all obstacles to let you have a better view.

Rule 5 can be change to have a field of view by uncommenting line near 405 so the boids will only following monkeys in front of them.

Conclusion

In conclusion I believe Boids are a very good way to mimic natural hearding behaviour. This could also be used for not only birds but also fish and land animal heards. Additionally swarm robotics could use a similar type of algorithm to keep groups of robots together without having to send a command to all of them. Each bot could simply look at nearby neighbors.