Lim Zong-Yi MID:28696328

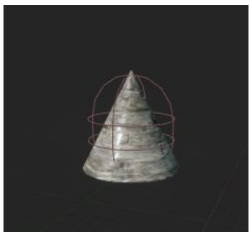
FIT3094 A3 Documentation

Repository: https://github.com/zlim0009/FIT3094_A3_Code_LimZongYi_28696328

Boids

Using the base code from Week 8, I used the Asteroids as my dynamic walls and have the 3000 float limit as to teleport the Boids to the other side of the bounds. The Boids are made using an Actor class.

Firstly, I set up the Boid's visual components. The mesh would be a cone shape as I wanted the tip of the cone to be where the Boid is pointing. The material of the cone would be one of the Starter Content Materials, Brick_Cut_Stone. During this time, I wanted to use a collision capsule to initiate a contact with other actors in the area. Therefore, I created a blueprint of the C++ class and in the Viewport, added a collision capsule to the RootComponent.



In the constructor, I attached the Collision Component to the BoidVisual and added the OnBeginOverlap dynamic.

I created a function so that the Boid can move. Before any functions are called, I will need to empty any previously stored arrays. I created two different types of scanning methods to find other actors in a personal space range and a visual range. The actors are then sorted in terms of the Actor type (Boid, Asteroid or Predator) and stored in either a Personal space array or a visual array. These Arrays are then used in different vector calculations for the Boid movement. To move the Boid, I would need to calculate the Collision Avoidance Vector, Velocity Matching Vector, the Flocking center vector and the Predator avoidance Vector.

Collision Avoidance Vector

The Collision vector will use the personal space arrays and will take in the directions of this Boid to other Boids surrounding it and add the opposite direction to a FVector variable. This process is done similarly with the Asteroids as they are treated as a dynamic wall and will need to be avoided. This is done so that the Boid would move away from the obstacle in the opposite direction. The cumulative Collision Vector is then normalized and multiplied by a Collision weight depending on the Boids genotype (more on this later).

Velocity Matching Vector

The Velocity Matching vector will use the visual Boid array and will take in the velocities of the other Boids surrounding it, by using GetVelocity(), and add that velocity to a FVector variable. The cumulative Velocity Vector will be divided by the number of visual Boids as of that time to get the average velocity of the Boids in the area. The velocity is then normalized and multiplied by a Velocity weight depending on the Boids genotype (more on this later).

Flocking Center Vector

The Flocking Center vector will use the visual boid array and will take in the locations of the other boids surrounding it, by using Boid->GetActorLocation(), and add that location to a FVector variable. The cumulative Location Vector will be divided by the number of visual boids as of that time to get the average location of the boids in the area and will be deemed as the Flocking Centre. The direction to the flocking centre is then normalized and multiplied by a Centering weight depending on the boids genotype (more on this later).

Predator Avoidance Vector

The Predator avoidance vector will use the visual predator array and will take in the location of the predators surrounding it and add the opposite direction to a FVector variable. The direction is then normalized and multiplied by a Predator weight depending on the Boids genotype (more on this later).

All the different directions and velocity are then added to a Resultant Vector that will be used to update the rotation of the Boid, using Quaternion rotation and slerp, and the position of the Boid. Additionally, I have added the direction of the Migration Target to the Resultant vector so that the Boids have a migratory urge.

The Boid uses the collision capsule to get any overlaps between other Boids, a predator or an asteroid. When there is an overlap, the Boid is “Dead” and therefore needs its boolean IsEaten to be false, it is hidden in game and if the fitness is not calculated during its death, the Fitness of the Boid reduces in terms of what hit it (50% if it was a predator and 25% if it was a Boid or wall).

Evolution (Boid) Generator

I created the Evolution Generator using an actor class. This class will hold the Arrays of Boids, DeadBoids, Migration target for the Boids and the Parents that will be selected for repopulation. Other variables are the static constant number of boids, predators and asteroids that need to be spawned.

In the BeginPlay functions, the Generator will call for an initial spawning of the actors in the bounds. First the generator will spawn Boids in random location in the bounds of 3000 to -3000, then they are each assigned a value of Weights to alter the behaviour of the Boids. These weights are of value float 0.00 to 1.00 as to simulate a percentage of the FVector they

are multiplied on when calculating distance and velocity. Lastly, the Boids are added to the Boid Array and a counter of Boids is increased until 100 Boids are created.
The Predators and the Migratory Urge target are spawned in with just its position at a random location in the bounds. These would act on their own and do not require a count or array to keep track of their behaviour in the generator.

The Asteroids are spawned in at the origin and they are to be dispersed to random locations in the bounds. Similarly with the Predators and Migratory Urge, would act on their own and do not require a count or array to keep track of their behaviour in the generator.

In the Tick function of the generator, the actor will keep track of the number of Boids that are alive in the scene. If the number of Boids is reduced to 20%, the need to repopulate begins again; a boolean value will turn to true, the parental selection will begin and the repopulation function will start after the parents are selected. If there are more than 20% of the Boids alive, the generator will check if the Migration Target has moved from its previous location, check if it is alive and then move the Boid. If the target has moved, the generator will update all the Boids of the new location. If the Boid has died, by colliding with something, it will be updated into the DeadBoidList and the Boid count is reduced.

To select the parents of the next generation of Boids, I decided to get the average fitness of the remaining alive Boids. This is done by checking if they are alive and adding their fitness together and dividing by the number of Boids left. To get the selection of the parents, I will take any Boids that have a fitness higher than the average alive Boid and add them to the Parent list.

To repopulate the Boids, I got 2 random numbers to be used as the indexes of the ParentList and use the Parents to get their Weights for evolution. Parent 1 will give its Collision and Vector Matching Weights and Parent 2 will give its Centering Weight and Predator Weight. This was decided by me as I wanted half of each parents genes to be passed down to their children. During evolution, there would be a 25% chance that there would be a mutation of a genotype on each gene. This is controlled by the MutationRate variable. The Boids' Migration Target is then reset, and all of its factors such as Fitness, IsAlive and IsFitnessCalculated are reset to the default values. The dead Boids will then become visible in game by switching the SetActorHiddenIngame as false. Lastly, the NeedToPopulate is false, the number of Boids is returned to the number initially spawned to and the DeadBoidList is emptied to get the newly evolved batch.

Predators, Asteroids and Migration Urge

Predators

The Predators are visualised the same way as Boids however they are using a different material, Concrete_Grime. Here I wanted them to first find Boids in the bounds. Therefore, using the same way to find other actors in the Boids, each Predator will produce a ray sphere and cast actors into Boids and if they are boids add them to a Visual Boid Array. This array will then go through each Boid to find whether they are alive and the nearest to the Predator. This will give us the TargetBoid. The Predator will move to the Boid's location and using a speed of 20% of the Boid. I created a collision capsule for the Predator for when the TargetBoid and the Predator overlap. There I will take in any OtherActors and cast them into a Boid and then check if the Boid that was hit is the TargetBoid. When that happens, the IsEating turns true and a timer is set for 5 seconds later. In the Tick, when IsEating is false, the MoveToBoid function is running. When IsEating is true, if the Timer reaches the TargetTime, IsEating is set back to false stating that the time period to eat has been reached and therefore the Predator can move to the next Boid.

Asteroids

The Asteroids are mainly left untouched from the Week 8 lab. However, I had to change some codes for the DynamicMaterial to StaticMaterial as it was giving me some Errors in UE4.

Migratory Urge

The Migration Target actor is visualised by using the Cube Shape and the Concrete_Poured material. This actor does not have a collision component as it will only be a guide for the Boids to move to. I created a Timer so that it will change to a random position in the bounds every 30 seconds.