FIT3094 - Assignment 3 Documentation

Pavle Spasic 29691931

Outline of Algorithm

Reynold's Boids

The First part of the assignment to be developed was the BoidAgent class for Reynold's boid system.

Similar to the base predator actor, although modified to fit the requirements and use of a boid. Each tick will do a SphereOverLap trace at the position of the boid for detecting if the boid is over something that could kill the boid, it also uses a USphereComponent to form the comfort zone of the boids

Once any collisions or overlaps had been detected, it was time for the boid to move around and know where to go. Each boid follows a priority list as follows…

Predator Avoidance > *Static Collision Avoidance* > *Flocking* > *Random Movement*

Predator Avoidance is done by using the collision sphere/comfort zone, after detecting any predators, it will get the vector that points away from each predator and use that as a direction to follow.

Static Collision avoidance is detected by a line trace going in the direction of the targetLocation, with a length of about 3 frames distance. When the boid approaches the wall, it will check for walls every X degrees to the left and right of it.
This is done by getting the vector between the current position and the point at which the wall was found, and rotating it and doing more LineTraceSingleByChannel.
Once a trace returns false indicating no wall, it will move to that direction.

Flocking implements Reynolds 3 rules, Collision Avoidance with other Boids, Velocity Matching, and flock centering.
This implementation goes over every boid in the comfort zone and does vector maths to get resulting vectors.

If any of these have not occurred, the Boid will move to a random target.

Fitness Function

Fitness of a Boid is determined by how long it has survived for and is reduced by long it dies. A value which takes on an Enum set of values controls the modifier for how much fitness is lost.

Evolution Controller

The nontrivial spawning code for the evolution controller is the same found in PredatorSpawner, although slightly adapted for Boids.

When a Boid is spawned, a variable inside the boid is set as a pointer to its spawner/controller. This is used when the Boid declares it has died, it will call the function boidDeath in the EvolutionController, this will simply move the Boid into a different array and move it off the map and stop it from ticking.

When the count of the active boids reaches 40, the Evolution controller initiates childGeneration. This function selects two parents and calls their Crossover function to create 2 children, this function takes half of one parent's genomes and half of the others. Then calls the children's Mutate function to randomise one of the genomes. It does this until 100 new Boids are created. After that, it removes all older alive and dead Boids from the world and continues looping.

Genomes

The genomes that vary from each Boid are as follows:

- Speed, ranging from 320 to 400
- Radius of the Comfort zone, ranging from 150 to 250
- How Quickly they will run from predators (predatorRunModifier), ranging from 0.5 to 3
- How many degrees between every wall check (wallavoidanceDegrees), ranging from 10 to 90.

Predator

The predator now every 10 seconds or when they have no target, will do a OverlapSphereActors in a radius of 400.0f. It will go over all the overlapping actors and save which BoidAgent which is the closest to the predator. If it manages to find one, it will chase that target until the 10 seconds finds a different, closer BoidAgent.

Reasoning

Reynold’s Boids

The initial design of the BoidAgent class had only the USphereComponent, during a problem that occurred during development, the possibility of using a SpherOverlap trace instead was

considered. In the end, the USphereComponent was left as it was not the cause of the problem at the time, and upon small research found that both had similar effects on the cost on performance, and the USphereComponent was kept due to familiarity.
However, the SphereOverlap trace was brought back for a different use with a smaller radius, as getting an actors overlap from the USphereComponent wouldn't tell if the Boids static mesh was over the object. So the trace was used to solve that.

The Action priority list for boids…

Predator Avoidance > *Static Collision Avoidance* > *Flocking* > *Random Movement*

Predator avoidance takes priority of others as dying from a predator can happen easily when moving slowly when flocking. This will ensure it runs away from any predators and will make any other boids potentially flocking together also avoid it.

Although static collision has a larger effect on fitness, the objects are static, so avoiding will be easier and less frequent than predators. So it has a lesser priority than predator, but more than flocking as it can kill the Boid.

Evolution Controller

Parents selecting for child generation are randomly chosen from alive Boids. The reason for this is, if the fitness of a boid is related to how long it is alive for, we will know that any Boid that is alive for when the Child Generation occurs, will have the same fitness. So randomly choosing parents will choose from the best.

Genomes

The genomes were selected to show potential highlights in the different movement aspects of boids. To see if running away from predators quicker, keeping closer to walls, or having a larger comfort zone radius would have better effects on survival.

Predator

Looking for a target will only happen every 10 seconds to reduce the cost they may have on performance, it will also give them a time to actually catch up and *eat* the Boid Agent.

Problems Encountered

An early rendition saw Boids spawning at the location 0,0,0, although this would not happen to all of them. This was attributed to initial location not being set in the begin play function for the lerp movement functions.

When implementing line traces for static object collision detection, the vectors/resulting lines were not in intended directions. This was due to incorrect vetor maths when finding the endpoint of the trace in the world.

When the Boid encountered walls in line traces in a circle to find which direction to travel in next, a problem that occurred during development was that the rotation wasn't being applied to the vector each loop. This ended up being the cause of the wrong use of the function RotateAngleAxis, i believed it changed the vector, although, it rather returned a new one. This not rotating of a vector also caused infinite loops when I wanted to rotate a number of times, although that loop was removed when redesigned during debugging.

There was a problem with boids where once they would start flocking, they would only go towards the top right of the world. This ended up being the cause of using world location vectors instead of direction vectors for the flocking rules; once the maths had been correctly changed to vectors of the target location from the current location, boids were moving fine again. This was a problem as my flocking implementation used..

*targetLocation = GetActorLocation() + vFinal * speed * DeltaTime;*

Rather than VInterpConstantTo to move.

Existing Bugs

Boids may initially spawn in walls or collide with other boids or predators. If they spawn within a predator or wall, they will die.

For boids get stuck in a wall, they get stuck their in a calculation finding a way out, once they fail they die.