SOFT351

Assignment 2

Joseph Kellaway

# End User Guide

Run the “Boids.sln” Microsoft Visual Studio Solution file and click run (or the “Boids.exe” file in the “\x64\Debug” pathway) to run the game. Boids are artificial life agents designed to simulate flocking animals originally created by Craig Reynolds (see <https://www.red3d.com/cwr/boids/> for a full description). In this simulation, the boids’ physical form is represented by parasaur.

The user is presented with a bird’s-eye view of the winged-bear, which they can move around the screen as with the previous assignment. The bear controls remain the same, with the F1 key remaining the key pressed to display controls on-screen whilst in the program. New commands added to the software are as follows:

* ‘e’ key – toggle the leash effect on and off.
* ‘r’ key – increase the strength of the leash affect forcing boids to stay near the default camera.
* ‘d’ key – decrease the strength of the leash affect forcing boids to stay near the default camera.
* ‘t’ key – increase the distance that the boids can move away from the default camera.
* ‘f’ key – decrease the distance that the boids can move away from the default camera.
* ‘y’ key – increase the range at which boids see nearby boids (affects turning decisions).
* ‘g’ key – decrease the range at which boids see nearby boids (affects turning decisions).
* ‘u’ key – increase the strength of cohesion (boid’s desire to be the centre of the flock).
* ‘h’ key – decrease the strength of cohesion (boid’s desire to be the centre of the flock).
* ‘i’ key – increase the strength of alignment (boid’s desire to face the same direction as the flock).
* ‘j’ key – decrease the strength of alignment (boid’s desire to face the same direction as the flock).
* ‘o’ key – increase the strength of separation (boid’s desire to maintain a minimum distance between themselves and other boids).
* ‘k’ key – decrease the strength of separation (boid’s desire to maintain a minimum distance between themselves and other boids).
* ‘p’ key – increase the distance that the boids desire to be away from each other.
* ‘l’ key – decrease the distance that the boids desire to be away from each other.

The default values are set in such a way that the boids appear to move around the map together somewhat whilst moving apart if they get too close to each other, avoid travelling too far away from the default camera position and flee the bear if it gets too close. Using the above commands, the player can change the relevant force or length from 10% to 200% of the default value.

# Programmer’s guide

What I am looking at from a programmer’s point of view. How your program fits together. The flow through your program. This is not intended to be an exercise in formally documenting a software project.

On start-up 100 Boid objects are created and stored in an array called flock. The Boid class inherits from the base class Thing3D as the boids themselves are objects with coordinates, rotations, scales etc. in the same manner that the bear is. The spawnFlock() function uses a random number generator to place the boid within a specified radius of the default camera by randomly selecting a value between the negative and positive maximum distance for the x-coordinate and then selects a random number between the negative and positive remaining value to assign to the z-coordinate. A random rotation between 0 and 2 pi radians about the x-axis is also selected to ensures that the boids don’t all start facing the same way.

Each frame the program checks for any keyboard inputs and moves the bear in the same manner as the previous assignment. After this a loop starts to calculate how each boid will turn and then move based on their environment:

A range check is performed on every other boid in the world to see if it is within the local neighbourhood of the boid calculating its current movement:

1. If there are no boids nearby, the boid will move randomly. This consists of two actions:
   1. A random turning motion decided by a random number between 1 and 3 where 1 is turn left, 2 is turn right and 3 is neither (i.e. continue straight).
   2. A random speed adjustment decided by a random number between 1 and 3 that favours boids travelling at approximately half speed. If the boid is travelling under half speed then 1 and 2 speed the boid up, and 3 slows it down; if the boid is travelling over half speed then 1 speeds it up, and 2 and 3 slow it down.
2. If there are other boids nearby, then the boid flocks:
   1. It adjusts its speed as previously described (in 1. b.).
   2. The boid matches alignment with its nearby neighbours – it calculates the mean of their rotations about the x-axis and then contributes 1% of the difference between this direction and its own rotation towards its heading.
   3. The boid then steers towards where it perceives as the center of its neighbours – it calculates the mean x and z coordinates and steers 0.1% towards this position.
   4. The boid then steers away from other boids that are too close – it performs a range check to see if neighbours are too close and then steers 2.5% away from them.

Next the boid checks to see if the user’s character is too close to them and moves away accordingly:

1. If the bear is close enough to attack the boid, it “eats” the boid, roaring (if no other sounds are playing) and the boid is respawned using the standard spawning rules.
2. Otherwise it steers away from them in the same way as with other boids, except it steers 5% away per frame.
3. It then performs a movement in this direction in the same way as the bear moved for the previous assignment. It merges the current rotation values into the math library’s rotation matrix, calculates the current direction based on this matrix and the boid’s “spawn” direction and then normalizes the value and applies the speed in the current direction before setting the new x, y and z coordinates.

The boid’s distance from the default camera position is then calculated and it is steered 2.5% towards it if it is too far away.

The boid’s movement is then applied as previously described (in ii.) This movement is applied whether the bear is near or not, giving the effect that a boid with the bear near it is actually fleeing the bear since it moves twice in the same turn.

Finally, modulus division is performed on the boid’s rotation so that multiple turns aren’t remembered or calculated each time. This was implemented to prevent miscalculations if the simulation was run for a large amount of time.

# Additional Note to the Programmer

The random numbers generated to decide the boids spawning is done uniformly using a single random number generator in the main “Boids.cpp” file. This is so that the area permitted to the boids should be approximately evenly filled with boids.

The random numbers generated to decide boid behaviour are created uniformly by a random number generator within the boid class. This is so that whilst each boid should turn left, right and continue straight 1/3 of the time, each boid could roll to turn left at exactly the same time, but with a shared random number generator with a uniform distribution this would be less likely to happen.

Performing cohesion, separation, fleeing the bear and applying the leashing affect occurs in the same way. Acquiring the point at which to face or flee is simple enough – the x and z coordinates are available or calculated. Calculating the angle to turn requires two vectors:

1. The first vector is the vector desired to be travelling in i.e. towards or away from an object. These are calculated by the following pseudocode:

towardsVector = desiredPosition – currentPosition

awayVector = currentPosition – avoidancePosition

1. The second vector required is the current movement vector if no change is required. This works as previously described (in ii.)

The xnamath library has a function, XMVector3AngleBetweenNormals(), which returns the radian angle between two normalised vectors, though it is always the angle between the two vectors (i.e. the angle less than pi radians) and it does not return which direction to turn. To accommodate this, I have attempted to implement a solution, though it is currently not working. The function is currently in the project but not called with pseudocode as follows:

Simulate a turn clockwise towards the target.

Calculate the angle between the simulated heading and the target heading.

If this new angle is smaller (i.e. we are turning the correct way) then continue.

Else turn anticlockwise the same amount and continue.

This pseudocode makes logical sense and I am unable to find any issue with the implementation, but the boids act incorrectly when this is applied, so at present these rules (cohesion, separation, fleeing and leashing) all create a clockwise turn. This results in boids getting caught in spinning loops after a short time due to the leash turning them towards the center of the map and the boids themselves turning away from it based on nearby boid positions.

When the boids are respawned, the new operator is not called as it would be more wasteful than simply reusing the boid that has been caught by the bear. The initialDirection value needs to be updated otherwise the boid would potentially start walking sideways or backwards when it thinks it is walking forwards.

# Software Engineering Issues

Due to each boid having to react to each other boid the simulation is computationally expensive – just to cycle through the flock to create a list of local neighbours requires 9900 loop cycles per frame – with more loops being required later if there are any groups of boids within range of each other to perform flocking behaviour. It would likely be better to apply separation in the original neighbour calculation loop, since it could reduce the number of loops by 9900 per frame (if every boid was positioned within separation range of each other), however, separation is a mechanic that should take precedent over most other functions so to perform it correctly requires less computational efficiency.

Good programming practices have been followed whilst creating the simulation where possible. Each boid hold values that are universal to all boids – scales in each dimension and the mesh skin for example. Whilst the boids are all in fact the same size and skin, it was desirable to allow each instantiation of the class to maintain its own value to make it possible to simulate real-world differences between the boids in the future. Not even animals of the same species are identical sizes, and several different mesh designs could be rotated throughout the flock to add authenticity to the simulation at a later date.

Modulus division is performed on the rotation of each boid each frame (if required) to prevent erroneous calculations if the float overflows due to the simulation running for an extended period of time. Whilst it is unlikely to be a huge cause of computational waste on its own, it is easy to lose track of inefficient snippets of code and when calculation times are totaled (especially when done per frame) they do add up to surprising amounts of wasted time. A better solution would be to manually calculate the minimum time a boid can reach an overlap (which would require always turning in one direction constantly) and perform it once per time period on each boid then instead.

# Ownership

As anticipated in the proposal document, I started with my own submission to coursework part 1 (in fact, from a user perspective many of the functions affecting the world remain unchanged). This submission had been created using a combination of my own work and the demos available on the DLE.

Added to the above are the boids themselves and various global values that affect the boids behaviour. The boid class is an extension of the Thing3D class (since they all need positions and rotations etc. anyway) with modified movement, boid-like decision making (which was researched via the provided link <http://www.red3d.com/cwr/boids/> as well as various AI lectures given as part of the course at Plymouth University).

# Evaluation

Overall, I feel that the boids work well for a short period before being caught in turning loops at the edge of the leash due to their inability to calculate in which direction to turn to reach their desired location. The leash itself is essential to ensure that the boids don’t run off into the distance, but without the leash being used the boids that do group up tend to stay together as boids typically would. There are still occasional loops being performed due to the single turn direction of the separation and cohesion functions, but the boids largely appear to be moving naturally.

After more thoroughly considering what had been initially proposed and the intention of boids themselves, I felt the need to revise what was to be created. Initially I set out to allow the user to direct the boids by attracting them towards a playable entity, though boids having their own (albeit collective) “free will” conflicts with this idea in my opinion. Instead, the theme of predator and prey was used to allow the user to move an entity around the map and the boids will attempt to avoid it (or “die” trying). I feel that this improves the overall believability of how the boids flock together.

The assignment could be further improved by using extra DirectX functionality, and the project could feel more complete if more external help had been used, however as stated in the proposal document, I never intended for that to be the case. My personal goal for the project was to code as much from scratch as I could to get a better understanding of both boids and C++ as a language and I feel I have mostly done that. The exception would be using the xnamath library to do some of the heavier lifting, though understanding how the mathematics behind turning etc. works felt necessary before production began.