Zans Gurskis
Acsv250

INM377: Game Physics and AI – Coursework Report

Overview

This project is focused on implementation of physics and AI to achieve realistic flocking behaviour. Flocking behaviour is a sum of several basic rules: first of - an agent must have basic navigational physics and usually some area where it is confined to. Then the additional steering rules can be applied- those are: alignment, cohesion and separation. Although they are seemingly basic rules, they take time and a lot of consideration to be implemented properly, since changing a single variable usually has a significant effect on other behaviours. The project was build using Bullet Physics Engine, building on top of the provided template.

Implementation

My project was continuously uploaded to GitHub. Each update can be seen on this link:
https://github.com/TheCuttlefish/PhysicsCoursework/commits/master (log in to view)

I started working on the provided template, which include a box and a boid with some forces. First of all, I've started adjusting variables to understand how the forces work in the Bullet Physics Engine. Once I had some understanding of the functionality, I begun adding more boids. Because of the inappropriate use of a forloop I have managed to separate the physics engine from graphics engine – and I produced 10 boids but only 1 of them obeyed physics. This issue was quickly solved, when I've created a Boid class which would encapsulate the behaviour, that way I could control my agents with an array in the main class.

After that I wanted to use a better debugging tool for visualising vectors and positions- I have found a book that explains how to render basic lines using embedded OpenGL library. (Learning Game Physics with Bullet Physics and OpenGL by Chris Dickenson, page 57). This discovery sped up my development process, as now I could draw the lines between needed vectors and quickly correct myself if my calculations had errors. I later made it into a function that can be toggled by pressing button- **[1]** on the keyboard.

When working on the boid object, I have moved most of the variable instantiations and functions into the header, so that the values can be quick to find and easy to modify. Then I've created a bounding volume, which prevents the boids from moving out of the certain radius. The radius is the length between the middle of the stage and the boid, once the boid is outside of the boundary, the boid slowly changes torque to point towards the middle. If the boid is in the boundary and it is not pointing towards the middle – it simple proceeds floating towards the concurrent direction. This adds some realism, as the agent rotates to different angles depending how far it is from the limit. Personally, I found it better to balance the physics of movement with multiple instances of the boids, as I could see how they behaved in general, and I avoided basing the movement on a single boid- which can possibly be an outlier.

At this point I was ready to implement 3 rules of flocking. First of all, I've added 3 empty functions to the direction vector, which would eventually process behaviours and output appropriate vectors. Then I started constructing the core functionality for all functions- by adding the neighbourhood – which is a radius of how far a single agent can see from its position. Then processing an array of other boids –except self. And adding a visibility angle- so that the boids can only see/react to the boids directly ahead, ignoring the ones behind.

This structure provided me with enough functionality, to implement the 3 rules. For alignment – I had to get the average orientation of boids in the neighbourhood, and add torque in that direction. Cohesion, was based on moving towards the average position of the boids in the surrounding area, with one little detail- there has to be some personal space. Therefore, the boid only moves towards the average when it can see others, but stops moving when it's too close. In addition to torque I also applied some central force to move towards the flock. And finally the boids are separating when other boids are in personal space, by rotating from intruding boids and also moving away with some central force. Moreover, in order to tweak different forced I've added scalar strength variables for each of the rules- that way- I could adjust and improve all behaviours for the desired outcome.

Avoidance is in the essence separation behaviour, with wider range of vision- so that the boids can react earlier. One major difference with avoiding cylinders, is that they extend indefinitely on the y-axis (unlike boids, which can be avoided by flying above or below the transform origin). My solution was to take the y-position of the cylinder and replace it with current boid's y-position, that way the boid is reacting, as if cylinder's y-position is always directly ahead of the boid.

A side note about obstacles – I did not create a class for obstacles, since their variables are always the same, I didn't find a reason to encapsulate any functionality.

In order to test, the ability to flock I've also added wind – which can be turned on and off by pressing button **[2]** – by default it is turned off.

I have also programmed functionality to change speed of the delta time – pressing button **[3]** will speed up delta time, pressing it again will return its value to the initial state. Changing delta time – is an interesting way of visualising the flock, as when they move slower they look like a school of fish, whereas when they move faster they look like a flock of birds.

Finally, I have added some randomness for the way my obstacles and boids spawn, every time the simulation is started (or restarted by pressing **[space]**) boids and obstacles are placed in different positions. Demonstrating that they can flock independently of their initial position.

Conclusion

This project put my programming skills to the limit, as in my experience I have not programmed real physics. Although having a lot of experience with pseudo-physics, made it easier to focus on what needed to be done. Another thing, I struggled with was the fact that Bullet Physics Engine is not well documented. I had to find a lot of functionality by trial and error or by asking my peers. Interestingly enough the simulation starts behaving in expected way once the alignment is added, cohesion and separation improve the overall appearance. I had a major improvement in flocking when I realised that neighbourhood should be set: between maximum visibility range and after personal space, as agents stop over-crowding and slow down when they are too close to one another. Overall I have enjoyed this module, I was always curious about swarm-logic and I'm glad that I finally got to program it. It is fascinating how agents rely only on local information to generate global intelligent behaviour.

Zans Gurskis
Acsv250

Additional input:

Button	**Functionality**
[1]	Toggle debug lines
[2]	Toggle wind force
[2]	Toggle delta time

Screenshots of flocking: