David Dunnings  
13015063  
Simon Scarle

Simulated Worlds 2



# In this assignment I will be:

Developing a simulation of predators and prey using a code base provided by Simon Scarle.

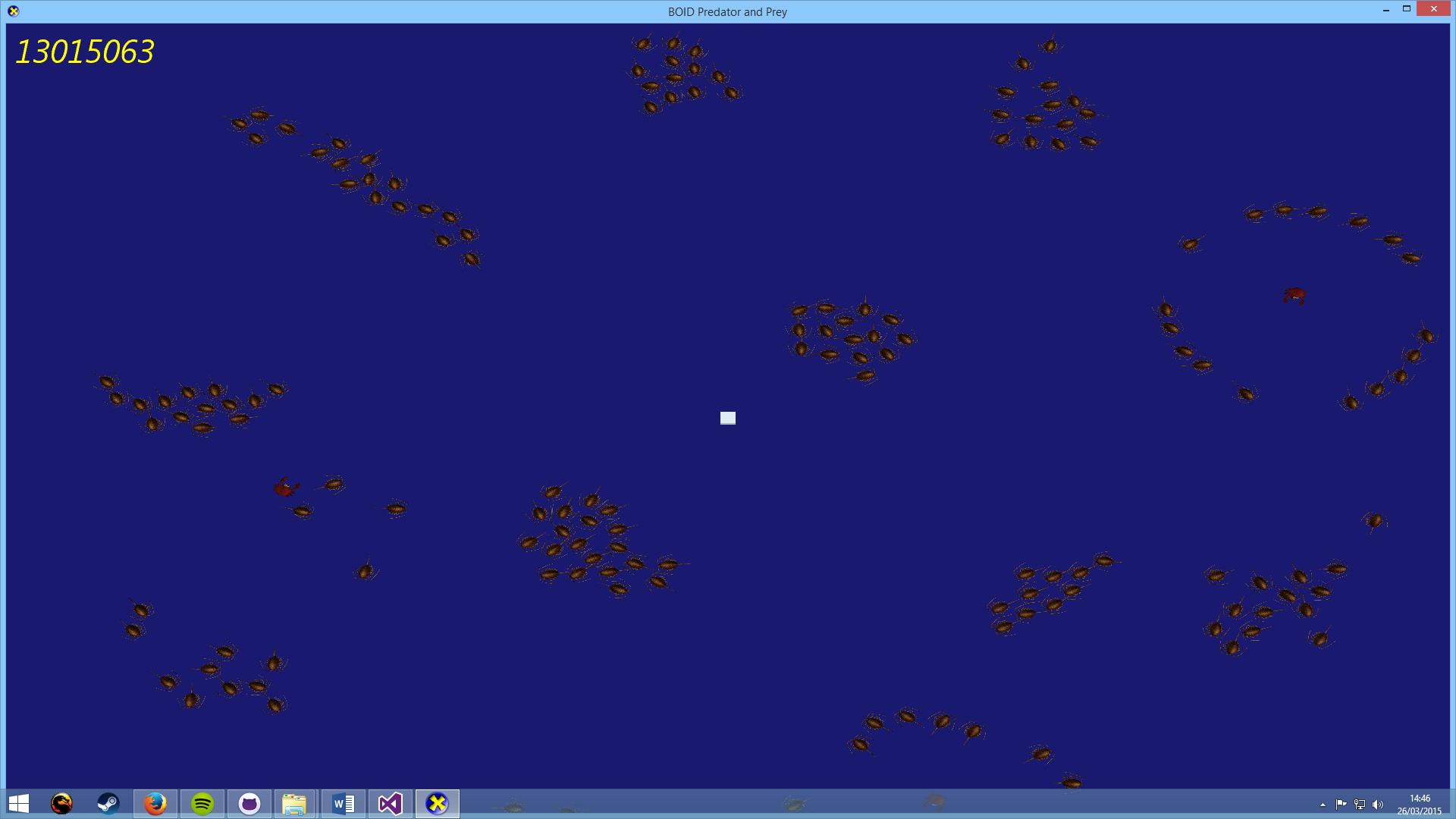
# Research

I have decided to split my research into two sub-sections. Firstly I will look at how animals react to their predators and how these predators hunt. I will then research previous boid simulations that may be relevant to my work.

## Hunting

During my research I discovered that a predominant feature that defines the effectiveness of predators is their vision. Their eyes are generally located in forward position and allow them to see large distances whilst having a narrow field of vision. This means that the predators can keep track of a prey from far away and that they have little or no care for things in their peripheral vision, such as other predators. From this, I have deducted that the predators in my simulation will have a parameter to represent vision distance which a prey needs to be located within for the predator to begin hunting them.

## boid simulations

In my research I discovered that the founder of boid simulations was Craig W. Reynolds. Craig published a paper in 1987 titled “**Flocks, Herds, and Schools: A Distributed Behavioral Model**” (REF 1). This was the first paper to talk about using computer graphics to simulate real-world flocking behavior. One specific section of his paper, which I think will heavily influence my boid system is the section labeled “**Simulated Flocks**”. In these few paragraphs Craig outlines the three behaviors which make up the flocking system:

Collision Avoidance: avoid collisions with nearby flockmates

Velocity Matching: attempt to match velocity with nearby flockmates

Flock Centering: attempt to stay close to nearby flockmates

For each of these three behaviors I designed a section of code to replicate the behavior. I concluded that to simulate collision avoidance, each boid will have a force applied to it for each nearby boid. The force will be created by calculating the vector from the nearby boid to the original boid. I can then apply a scaled down version of this vector to the original boid to push it away from the nearby boid.

As for the velocity matching I will most likely take an average velocity of every boid in a localized area around each individual boid. I will then apply a scaled-down version of this to each boid.

I will perform a similar calculation for the flock centering. I will take an average location of all nearby boids and create a vector from the target boid to the average group location. I can then apply a scaled-down version of this vector to the target boid.

In Craig’s paper he also mentions collision avoidance. He listed a couple of methods of making sure boids avoid obstacles. Steer-to-avoid is one method that Craig outlines:

“The boid considers only obstacles directly in front of it. Working in local perspective space, it finds the silhouette edge of the obstacle closest to the point of eventual impact. A radial vector is computed which will aim the boid at a point one body length beyond that silhouette edge.”

This was the chosen method for Craig’s system and I am considering using it in my simulation. However, Craig also mentions another method of collision avoidance which I believe may work better in my system:

“The force field model postulates a field of repulsion force emanating from the obstacle out into space; the boids are increasingly repulsed as they get closer to the obstacle.”

The force-field model is most likely going to be the model of choice for my collision avoidance. The reason for this is that it uses the same principle of attractive and repulsive forces as the rest of the simulation. Therefore I can easily have the same functions for avoiding obstacles as avoiding predators.

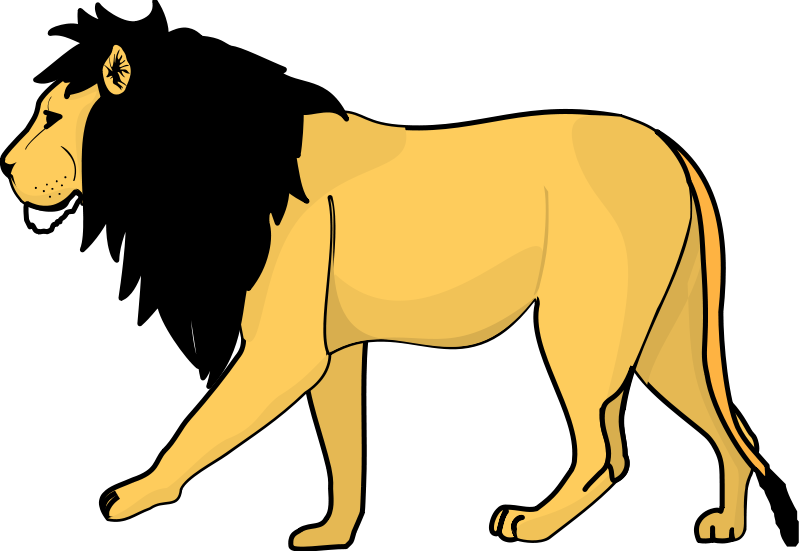
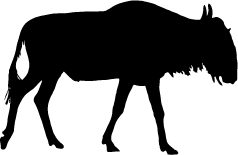
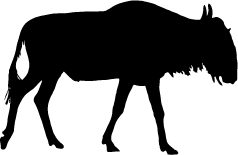
# Iterations

## Class iterations

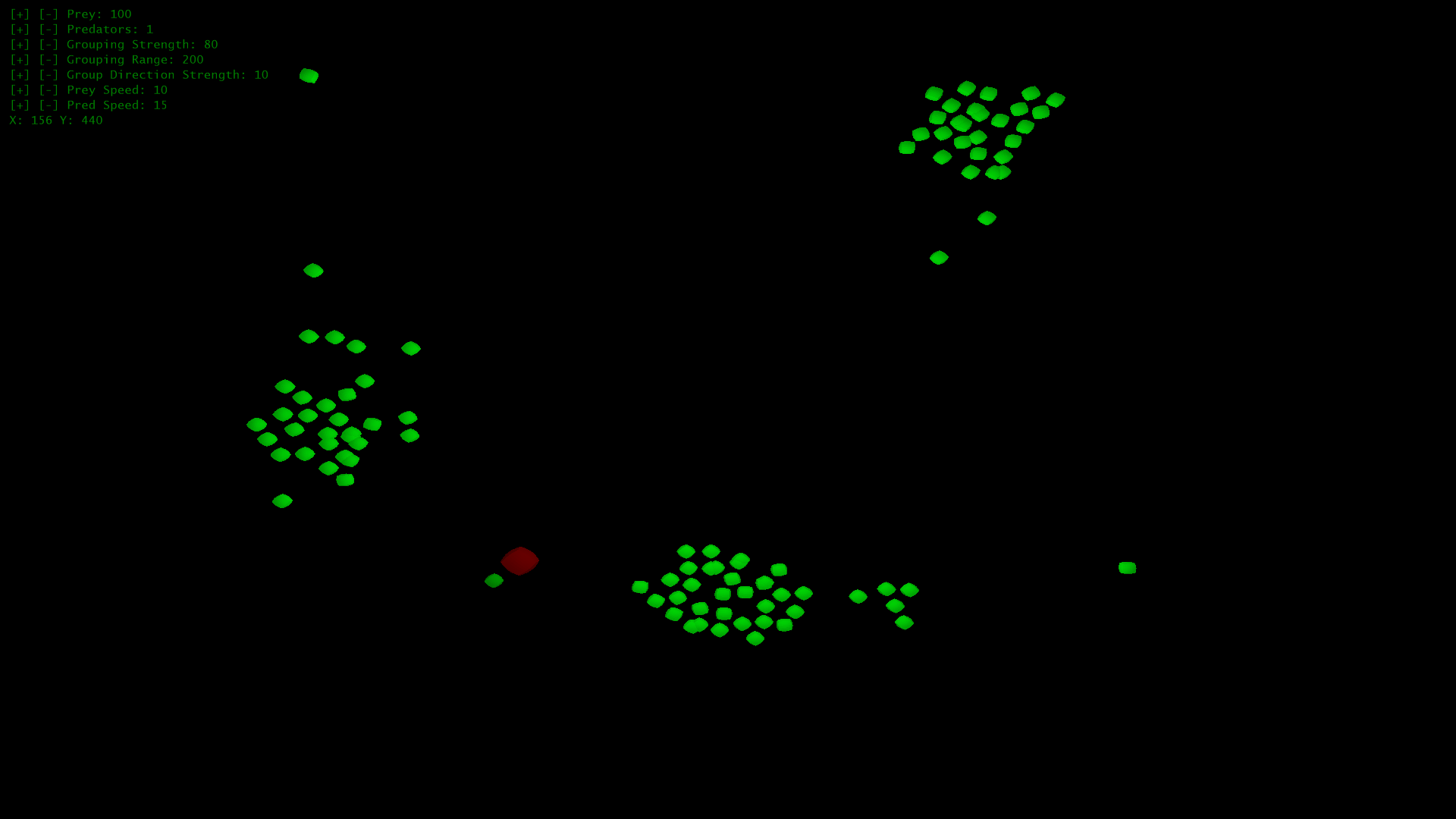
The first iteration of my boid simulation had a ***predatorBoid*** class and a ***preyBoid*** class. I quickly realized that this was unnecessary as both classes were almost identical, except for the polarity of the forces applied being reversed. Therefore I merged both classes into a parent class ***Boid***. This class had a ***BoidType*** enumerator class which dictates the hierarchy of the food chain. For example, one simple setup of the ***BoidType*** enumerator could be the following:

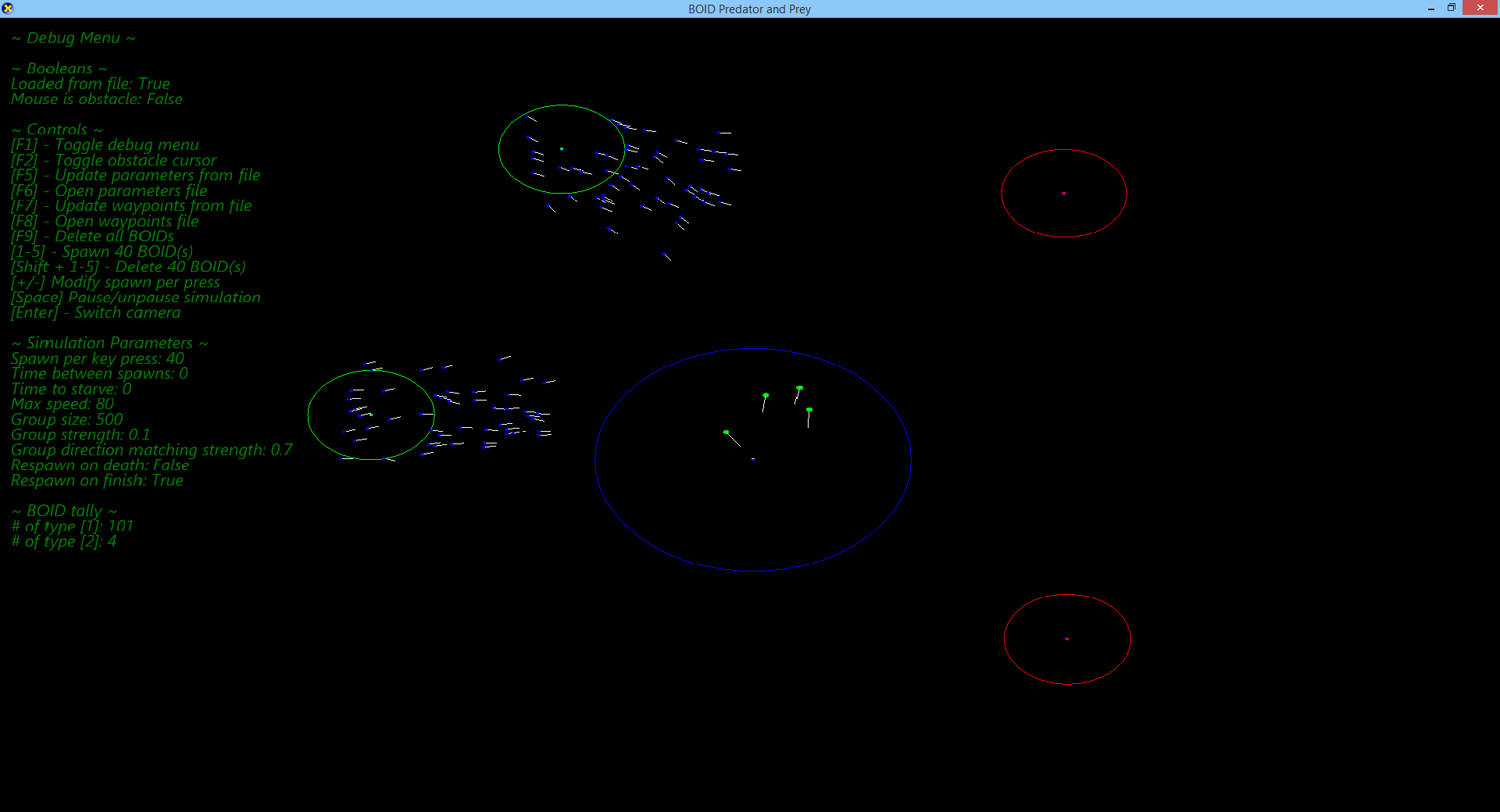
[0] boid\_OBSTACLE, [1] boid\_WILDEBEEST, [2] boid\_LION

In this instance, any boid with their BoidType set to [1] or [2] will avoid any boid with their BoidType set to [0]. Furthermore, boids with BoidType [2] will be attracted to boids of BoidType [1]. This attraction is directly inversed for the relationship between BoidType [1] and BoidType [2]. This ultimately results in BoidType [2] being the sole predator type, which hunts BoidType [1].



## Debug Iterations

The first version of my simulation to include a debug interface listed each simulation parameter along the left of the screen. There were plus and minus buttons next to each parameter which incremented/decremented their associated value. These were achieved by using a large nested if statement. The program would check the cursor’s position upon clicking and check if it was within the bounds for any of the “buttons”.



The latest version of the GUI removed the clickable “buttons” and instead opted for a more keyboard-centralized interface. The debug menu starts with a couple of true/false statements to display whether the simulation parameters have been loaded from file and if the mouse is being used as an obstacle. The next section lists all of the available controls for the simulation with a short description of what the action does. The next section lists all of the simulation parameters. This section no longer is intractable via mouse. The reason for this is that it was more hassle than it was worth to implement scaling clickable buttons. The user can now press a key to open the data file which holds the simulation parameters. The user can edit these parameters and close the text file to have the changes take effect instantly within the simulation.

Another noticeable change is the addition of force indicators for each boid. These use the boid’s current speed and current direction to create a line out from the boid. To create these the boids have a secondary vertex buffer which uses a line list format to draw a line between two vertices. Using the same line list vertex buffer the simulation also draws circles around each waypoint to indicate it’s area of effect.

# References

REF 1. http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/