The boids' flocking algorithm is easier to implement overall then I expected. When we watched the videos of flocking programs coded in different rendering software, I was a little intimidated at first, I can't lie. This was also compounded by the fact that I’ve personally been having trouble with the intensive C++ coding skills that this class requires. However, our coding days with the whole class and then conversing with classmates and scouring different forum threads for different solutions really helped put everything together. Unfortunately, I was unable to get a working 3dworld integration with the flocking program, but I think the main issue I was having was rendering the agents in 3d vs 2d by using the "glvector3f" instead of "glvector2f" may have given better results when it came to the compilation of both series of files. I also had trouble with scaling the window down to follow a single agent, as when I would actually use the "f" key to begin the follow function, all of the agents being rendered currently would stop as if I had pressed the spacebar.

Various features have been added with the help of the guided coding presentations as a base. In the Agent’s, we can initialize the various forces that are acting upon our agents as Vector3d objects in the private section of our Agent class. Separation force, which acts as a force that governs how far the distance is maintained between the different boid actors that are being rendered on the display. Cohesion force is also represented by a Vector3d object that is in charge of the amount of force is applied to the boids when they are too far from each other and "want" to be closer. This is calculated by the number of agents on the screen and how far they are from each other. Alignment force updates the agents based on the velocity of the other agents that are able to be "seen" by the other agents on the display. In the controllingAgent.cpp file a few different features were added or changed. An adversarial force was added as well in the same way, by initializing a force by the Vector3d class, which after seeing its uses in this project, has proved itself to be very useful in a number of situations.

I liked the window itself a little larger, so I scaled it up to 700 by 700 units. Using the help from our coding in class, I added the integer variable "agentsToAdd" which will later help in adding agents to the rendering of our flocking program. Also, by changing the background color of the window to black, I found it easier to see the different behaviors of the boids as they respond to the different values of forces I tested on them. Unfortunately, I didn't really find a sweet spot of the flocking program that produced boid behaviors that I was completely satisfied with. Next, a menu and different options for the menu were created to add different numbers of boids to the render while it was being ran. This is accomplished by using "glutAddMenuEntry" and "glutAttachMenu" and setting the menu to be initialized using the right mouse button. The menu entries then use a string to identify what the option will do, and then an integer that selects the number of agents to add based on the integer number one through four. They add a single agent, five, ten, and 25 agents at a time depending on which option you select. Adding an adversary was premade for the most part and had to be done by initializing a set number of agents to be “adversaries” by setting them to be an adversary if there were no other agents being rendered at the time. This uses the update function in the Simulator class found in simulator.cpp.

I was unable to get the mouse button to create new agents based on location of the mouse button relative to the render window, but the agents are still being created as expected otherwise. The call to create the agents is located in the void mousebutton function, which I didn't really modify too much other than using the gSim variable to add agents, and then pass the number of agents to the functions with agentsToAdd that is selected from the menu. Again, with help from our coding sections, GetForceFromWall and GetResolvedPosition were added to the Environment.cpp file. Setting the force from the wall the same as we did the other forces with initializing them as variables from the Vector3d class, we find the distance of the agents from the closest wall to them and apply a force based on that distance. GetResolvedPosition takes the position of an agent and finds its position over time based on where it was last located. In this file I was getting a weird stoppage every other compile, stating that "Thread 1: signal SIGABRT" which I’m still not completely sure of what it means, but either way would not show up every single time, but also would not completely break my program. It was located on line 69 in Environment.cpp. In the Simulator.cpp file not much was changed as a lot of this part of the program was done already.

Altogether I really enjoyed working on this program, as with the last C++ heavy project, it definitely showed me how much I need to learn to be a competent programmer in not only C++ but any programming language. Adding various changes and tweaks on the fly in the middle of the program is exciting and I hope to do more of those kinds of creative projects in the future.