Homework 2 Reflection

I created a pure virtual class called “SteeringBehavior” that will be the class that each variable matching or combination of variable matching behaviors will have to extend. All that it has is a constructor and then a virtual function called “getSteering” that needs to be implemented. I’m not sure if I should try to have this class to house the maxAccel and maxAngularAccel fields that some SteeringBehaviors use although I have found success and enjoy that these are separate for now at least as some behaviors look nicer when these values are changed. (This could also just be changed with constructors to set these values but for now I just have them as constants that are defined in the header files). I then created a Kinematic and a SteeringData struct for easier access to these important variables as well as operators to help me with being able to utilize these structs in a way that is intuitive to me. I also created a KinematicBody class which extends both Kinematic and sf::Sprite as a way to communicate with sf::Sprite with the stuff that we are learning in class all in one location with an update function taking in SteeringData and a deltaTime. I then created SteeringBehavior classes for the four kinematic variables. The position and orientation matching behaviors were implemented in the form of arrive and align respectively, and the velocity and rotation matching behaviors were implemented as simply velocity match and rotation match. I was able to implement these largely from the book.

With my demonstration of velocity matching to the mouse pointer, I sampled the mouse pointer’s location gathering position and velocity data, putting that into a mouse kinematic to be sent with the character boid to calculate an acceleration on every frame (this style of sampling or calculating new SteeringData did change for later parts). Then an update is called onto the character boid demonstrating that it can match the mouse’s velocity.

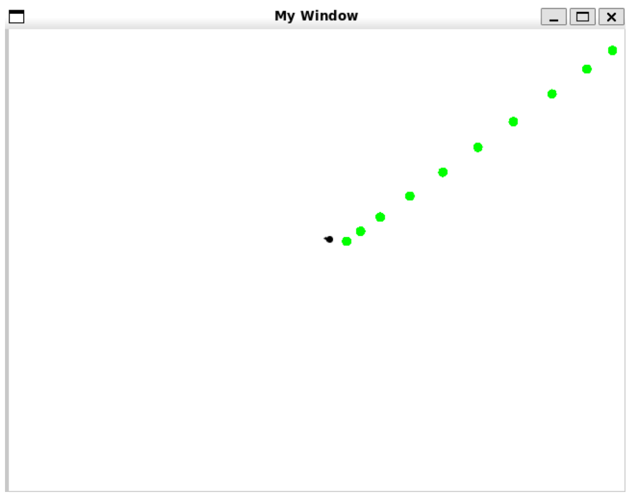
When implementing and testing the arrive algorithm, I feel like I ended up paying a lot of attention to the maxVelocity and slowRadius parameter a lot as that is what seemed to allow my character to go past the desired position and get pulled back and my two different parameter selections show that. The second one has more speed and a larger slowRadius but the speed ultimately ends up being too high for the algorithm to slow it down to stop on that target (SEE PART 2 PICTURES). The align behavior was implemented using radians which is how the book initially described it and thus I have had some issues where the orientation doesn’t end up exactly where I may want it to, most likely due to the final step of converting the orientation from radians to degrees when setting the rotation of the boid using the setRotation function from SFML that only takes degrees. The actual process or at least the way it takes to achieve this result is something that I do really like. It has a pretty fast start up then slows down just before it reaches its desired orientation and then stops. This allows for this behavior to finish very quickly after the click has been pressed allowing for the character to turn and face the direction that it will be traveling in, making for very believable movement behaviors that somewhat mimic how living creatures would start moving even though they are not facing their intended target but will face it soon after while they complete their movement to the target.

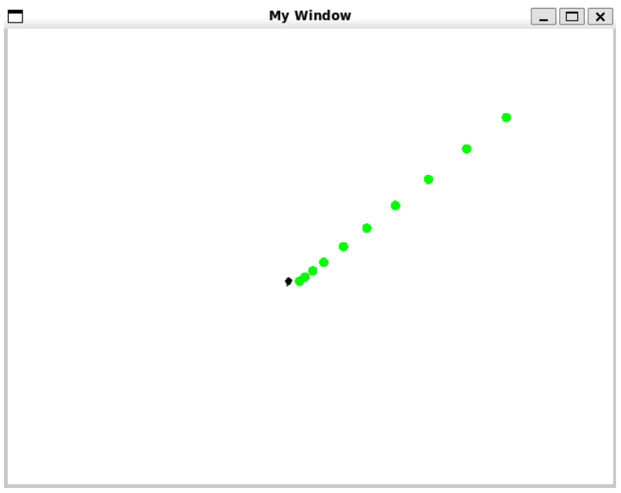
For the wander algorithm, I used the algorithm from class that uses the characters current orientation and makes a random adjustment to it given a random binomial and then a position in front of that target orientation is found and arrive and align is called onto these variables. The two different versions of wander I implemented were different only in the way that I would find the target orientation as every other part of the algorithm uses this orientation. Both were also some kind of random function but with one using the random binomial as represented in class and another using just one random 0-1 number that has a 50% chance on whether it is positive or negative. Both of these would be able to express the same values but the one that just has the random number from 0-1 would have the same chance for every number in the range from -1 to 1 however the random binomial allows for a greater chance for the result to be closer to 0, so that the character would mostly stay moving straight and making turns every so often. This behavior with the random binomial is the one that I prefer as it makes more sense for the character to move straight for a little bit before each turn while wandering, rather than making many small turns without making much progress in actual movement in any particular direction (SEE PART 3 PICTURES). Here is also where I implemented the ability for the boids to wrap around the screen and did so without tampering with the values that were located in the Kinematic fields as I merely just took those fields and made sure that they were positive (adding the window width and height if negative) and within the bounds of the window (using fmod) and putting that value into the setPosition and setRotation functions for SFML display.

Finally, I have implemented a flocking behavior. This flocking behavior is merely the implementation of a separation behavior that was implemented using the class textbook. Then, given a list of boids I could, for each boid, iterate through the list and find the center of mass of the boids that are nearby given some kind of radius, then for each that are within that radius, their position and velocity is added into a Kinematic and once I've iterated through the list, I average the data that has been added to the Kinematic to find the center of mass and the velocity of the center of mass. From there I also find an orientation that I gather by using the direction of the velocity of the center of mass as that's where the boids will ultimately be moving to as a group. Then this information is all delegated to arrive, align, and velMatch which is then blended together using differing weights with separation having a high weight, then velocity matching, and then arrive; align kinda gets left out here but angular acceleration isn't touched anywhere else so it doesn't really need any tampering. This new blended steering data then gets clipped to max acceleration to produce the final output. Some of the tampering that I found particularly useful to producing a desired output was to 1) have 30 boids to produce a nice amount of separation in order to get some movement going in the beginning and 2) to have the velocity match behavior be relatively powerful as to let the flock have some decent movement to it so that the group could be clearly seen moving together (SEE PART 4 PICTURE). One caveat that could make for some confusing viewing is that my implemented screen wrapping implementation doesn't tamper with the Kinematic fields located in the boids so when they wrap, if they were to come into contact with a boid that hasn't wrapped, (or not wrapped in the same direction the others had) that boid would not be considered as next to the other boids for the sake of calculating center of mass and whatnot as it would still technically be a screen length away from those boids however I think this is an ok implementation of such behavior.

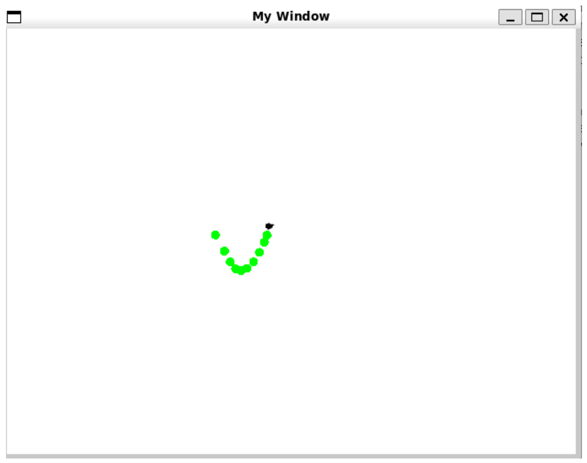
Appendix

Part 2: These two images were taken with the middle of the screen selected as the target position, however the one on the left was able to stop right at the middle while the one on the right goes past it.

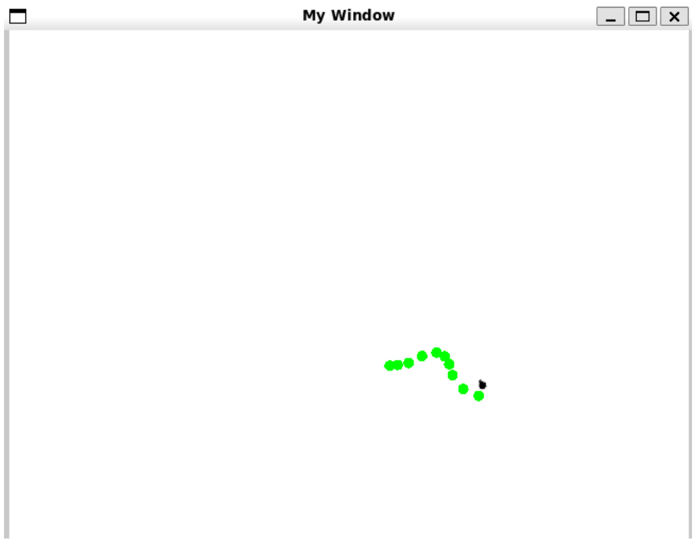




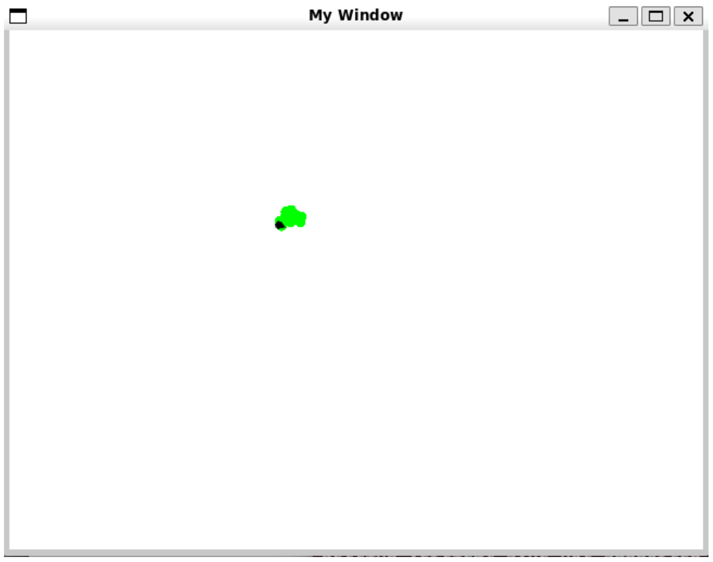
The arrive behavior when other acceleration is already present (from previous arriving)



Part 3: The first wander behavior demonstrating movement in different directions



While the second wander behavior is prone to turning too much and not enough movement in a direction



Part 4: The group flocks to the right while one gets left behind :(