Introductory information: For all of the parts of the project, I utilized a modified version of the KinematicBreadCrumbs class provided by the course. The modifications introduce a deque that holds the previous 10 locations of breadcrumbs. This is to ensure that we don't have infinitely growing trails of breadcrumbs that slow down the program's ability to function.

Kinematic Motion (Part 1): In order to see this part of the project, you need to uncomment the "firstBasicMotion()" function found within "scenario.pde". For this portion of the project, I used a kinematic seek that goes through a cycling queue of locations to visit. It follows an imaginary rectangle's corners in a continuing cycle. Because of the way this behavior was implemented, this behavior cannot be mixed with any other behaviors since it directly sets the velocity and orientation of the Boid. This behavior also lacks any ability to slow down/accelerate due to it's direct manipulation on velocity and no usage of acceleration. I have provided it the ability to alter it's orientation based on a combination of smoothing and current velocity & orientation of the Boid. The lower the smoothing value is, the slower the turn the Boid has.

Arrive Steering Behaviors (Part 2): In order to see this part of the project, you need to uncomment either the "firstArrive()" or the "secondArrive()" functions found within "scenario.pde". The first implementation of arrive demonstrated is a dynamic version of arrive that goes to the target set by a click. Both implementations have a method to slow down its arrival, but the kinematic one is unable to control its starting speed, while the kinematic one can overshoot its target (it will only need one correction to fix). If a new destination is set mid-travel, the kinematic version will immediately start travel in the direction of the new destination. The dynamic version on the other hand,

slowly start moving in the direction. This allows an almost drift like behavior from the dynamic implementation.

The first of these two (the dynamic one) is far better a looking arrive implementation. This is because it maintains a far smoother motion of travel than the kinematic one. The kinematic (while it can make smooth rotations) cannot travel in a smooth manner when changing directions. While working on this portion of the project, I was playing around with which version of alignment I wanted to pair with arrive (face vs. look). In the end, I stuck with the behavior to face towards rather than look. This is because this makes it so that the Boid will face the direction it is trying to reach. The look behavior makes the Boid look in the direction it is traveling, which gives off a different feeling from something that actually faces its intended destination. (the steering behavior that alters the direction the Boid faces doesn't really matter in this case since it doesn't actually alter the trajectory of the Boid).

Wander Steering Behaviors (Part 3): In order to see this part of the project, you need to uncomment the "firstWander()", "secondWander()", or "thirdWander()" functions found within "scenario.pde". The first wander, although technically dynamic, is the most similar implementation to a kinematic implementation of the three wander algorithms I have.

The first wander implementation I have does not alter the orientation of the Boid. It only alters the acceleration in a particular direction for the Boid based on a target orientation. The alignment of the Boid is left to other behaviors added to the Boid. This strangely enough seems to be the most consistently "random" of the wander behaviors. This is likely because it is moving as though the wander radius is on top of itself, which makes it so that the new intended travel direction can change to any direction at any point.

The other two wanders have very similar implementations, just utilizing a different method to orient itself. The second one orients itself using a face steering behavior, while the third one utilizes a look steering behavior. Because of the implementation of these wander behaviors, they seek a target on a circle ahead based on an offset from the Boid. This the likely cause to the oscillation like behavior it has as it travels. If I were to set the offset of the circle to zero, the behavior of these two wander behaviors will be pretty much the same as the first wander behavior I implemented.

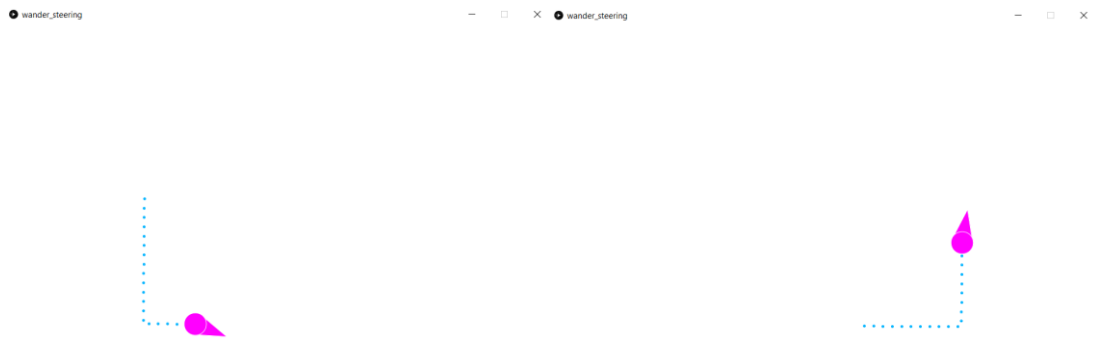
In general, I think the first wander algorithm looks nicer, but the other two wander algorithm have a lot more control I can exert to suggest directions of travel for wandering.

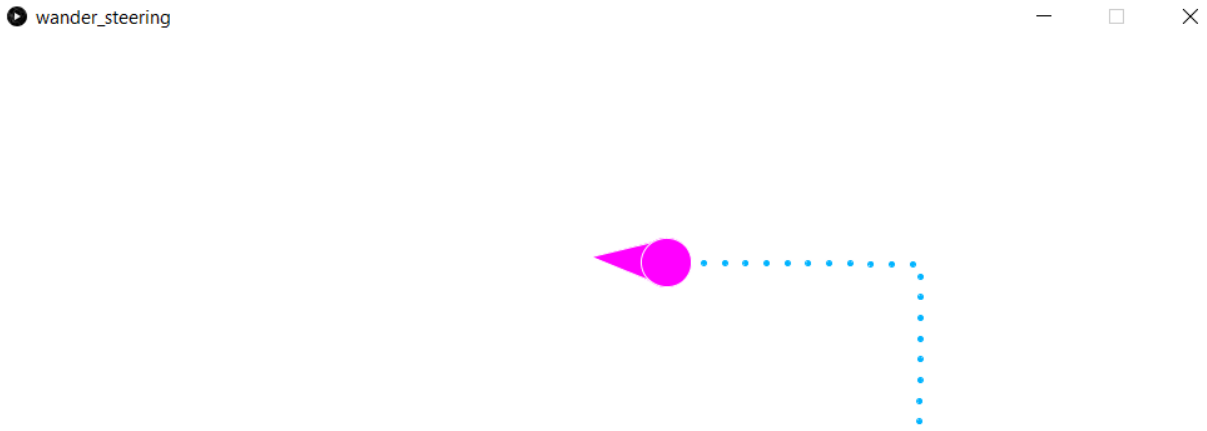
Flocking Behavior and Blending/Arbitration Behaviors (Part 4):

In order to see this part of the project, you need to uncomment the “flockCode()” function found within “scenario.pde”. For this implementation, I have chosen to put two leaders on the field, one using a wander behavior, and another that uses the arrive behavior. The main reason I did this was because I was having an unusual amount of trouble trying to get flocks to stick with the wanderer. Regardless of tweaks to behavior values I made, the flock refuses to stick with the wanderer. The leader with the arrive behavior however, is really good at keeping the flock together. As I was figuring out the best values to keep for flock behavior, I noticed that the more Boids there are in a flock, the more difficult it is for the “leader” to alter the course of flow. At the same time, however, these large flocks break apart pretty easily if there is another “leader” that is tries to steal members of a flock. The boundaries are the typical cause of my flocks falling apart.

Part 1:

basic_motion travel:

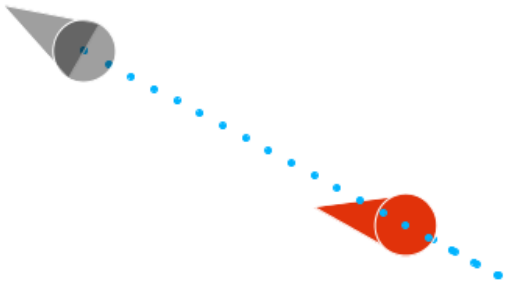


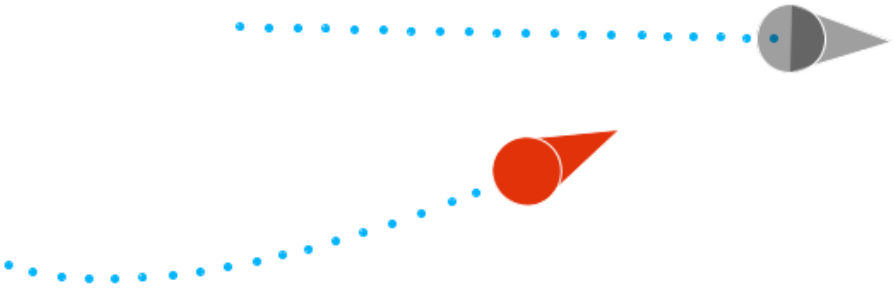


Part 2:

Gray: Kinematic Arrive

Orange: Dynamic Arrive



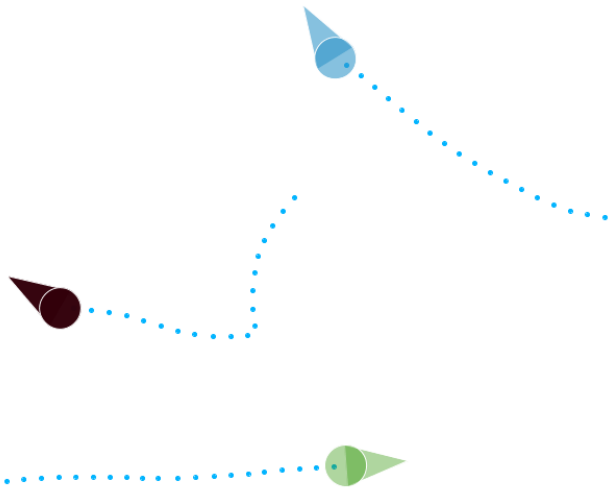


Part 3:

Green: firstWander()

Blue: thirdWander()

Brown: twoWander()



Part 4:

Two leaders: Left side has arrive leader, right side has wander leader