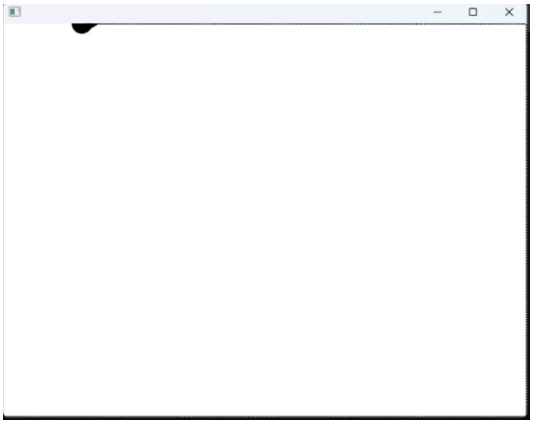
Process Letter

Kinematic Motion

The **Rigidbody** class encapsulates essential motion attributes, including position, velocity, acceleration, rotation, orientation, and movement constraints such as **maxSpeed** and **maxForce**. To ensure a visible trail, the update method in **ofApp** appends the shape's current position to a breadcrumbs vector as it moves. The shape starts at the bottom-left corner of the screen and follows a circuit path along the edges while maintaining its velocity and orientation.

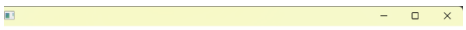


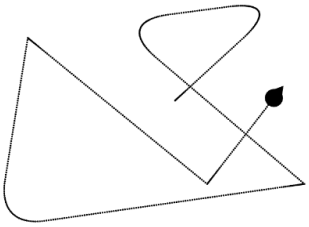
Dynamic Steering Behaviors

The seek algorithm is implemented in **Boid::seek**, where the shape calculates a force directed toward the target. The shape actively seeks mouse click locations, with **ofApp::mousePressed** updating the target position whenever the user clicks on the screen. To ensure proper orientation, **setOrientation** updates the shape's rotation based on its velocity vector, aligning its direction with its movement.

Two methods of arrival are implemented to ensure smooth deceleration near the target. The first method dynamically adjusts speed based on proximity, gradually reducing velocity within a defined radius to allow for a smooth stop. The second method integrates arrival behavior as an optional parameter in the **seek()** function, modifying speed within a slow radius—as the shape approaches the target, its speed decreases proportionally, ensuring controlled and natural movement. I think the first arrival method is better because it dynamically adjusts speed based on proximity, allowing the shape to smoothly decelerate as it approaches the target. Compared to the second method, the first method provides a smoother and more natural stop, as it continuously adjusts the speed based on the distance to the target, avoiding abrupt changes in velocity.

While working on the seek and arrival behaviors, I found that the dynamic speed adjustment method offered a smoother and more natural deceleration as the shape neared the target. Initially, I had difficulty achieving smooth stopping behaviors, but through testing and adjusting the radius values for both methods, I learned the importance of gradual transitions in motion.





Wander Steering Behaviors

I designed two wander methods, both methods create random wandering behavior for a Boid but differ in how they adjust direction. The first method determines a wander point ahead of the Boid based on velocity, then offsets it within a circular range, ensuring gradual directional shifts. A small random displacement is applied to **wanderTheta**, leading to smooth yet unpredictable movement. The second method relies on Perlin noise to generate

an angle for steering, resulting in rapid and fluid directional changes. However, it can cause the Boid's path to shift too quickly, making its motion appear less stable. The first method is preferable because it maintains a more controlled and steady wandering trajectory while still incorporating randomness.

Method1 (wanderTheta) :

Method 2 (Perlin Noise) :

The first method for changing orientation provided much more stable and controlled movement, which I found essential for the Boid's wander behavior to appear fluid. Initially, I thought the second method using Perlin noise would create more interesting, unpredictable movement, but it led to erratic behavior, making the Boid's path appear unstable. This forced me to reconsider the balance between randomness and stability in wander behaviors. hrough this process, I learned that small, continuous changes often result in a more satisfying visual experience than abrupt, unpredictable shifts.

Flocking Behavior and Blending

The Leader Boid is red, while others are black. It moves randomly but smoothly using **Boid::wander()**, projecting a point ahead with a randomly displaced angle for organic motion. Boundary violations are handled with toroidal mapping to wrap the leader seamlessly around the screen. The flock exhibits cohesion through **Boid::cohesion()**, where each boid moves toward the average position of nearby flock members within a perception radius, with weighted averaging for smooth movement. Followers are influenced by the center of mass, with **Boid::align()** ensuring alignment with the flock's general direction.

During implementation, I observed how alignment, cohesion, and separation forces affected the boids' movement. Initially chaotic, the movement became smoother after adjusting parameters like perception radius and weights. I fine-tuned the forces to prevent abrupt acceleration or deceleration. Experimenting with more followers made movement more fluid but caused crowding and collisions, while fewer followers led to smoother movement but less group cohesion. More followers required careful tuning of steering behaviors. Introducing two wanderers and having followers follow the closest one split the flock into two groups, creating chaotic movement. Further tuning was needed to maintain cohesion.

The process of blending these behaviors led me to experiment with adding separation to the flock, which improved the movement dynamics by allowing the Boids to avoid collisions more effectively. This made me realize how subtle adjustments to parameters can dramatically alter the overall behavior of the system, and the importance of testing combinations of different behaviors to achieve desired results.