CSC 484 – Game AI
Kuangyao Mai – kmai

Part 1:
Part 1's task is to create four steering behaviors that match the kinematic variables - position, orientation, velocity, and rotation. These variables would be stored in the Kinematic class. The first thing I did was create a pure virtual class called SteeringBehavior. This class will have a function called calculateAcceleration, and the subclasses of SteeringBehavior will have to override it. It returns a SteeringOutput object, which contains acceleration for linear and angular.

1. Velocity: To match the velocity of a target, a VelocityMatching class will be created, which will override the pure virtual function of the SteeringBehavior class and return a new acceleration. The calculation for the desired velocity will involve subtracting its current velocity from the target's velocity.
2. Rotation: Similar to velocity matching, the RotationMatching class will be created to match the rotation of a target. The calculation will just be the target's rotation - its rotation.

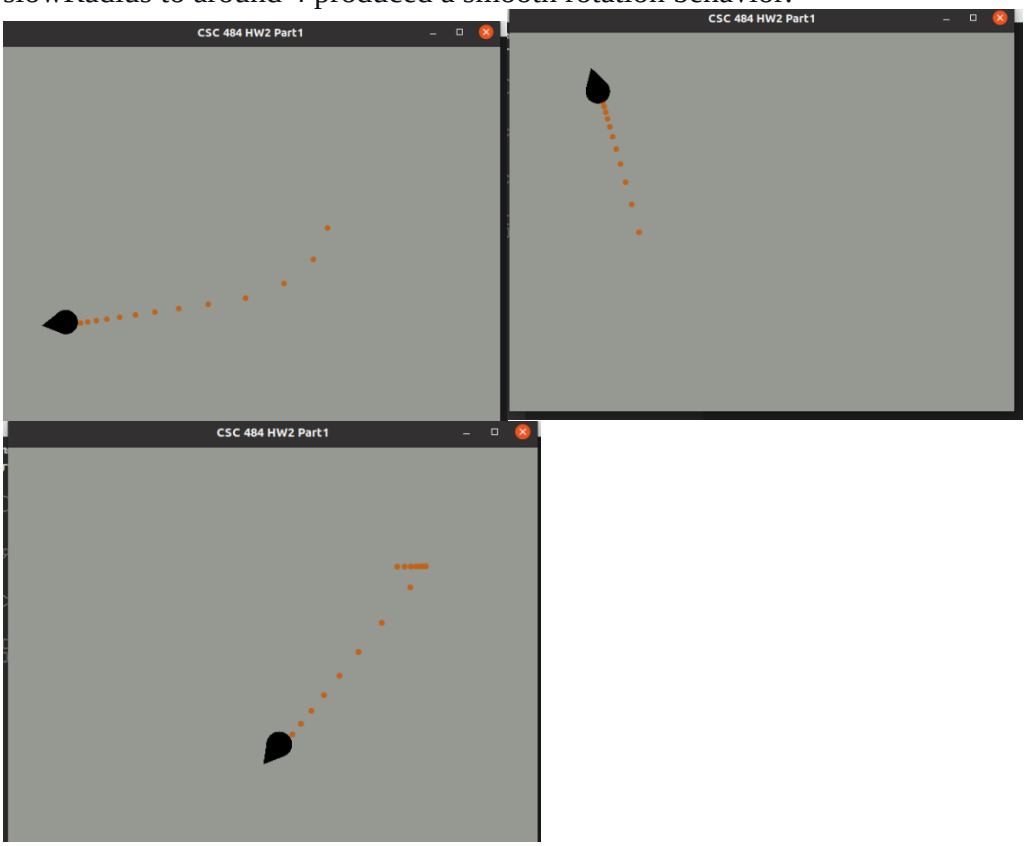
Position and Orientation matching will be explained in the next part.
To match the velocity of the mouse, I have to sample the mouse pointer's position in every iteration. The displacement of the mouse pointer will be calculated by subtracting the previous position from the current position. Then velocity will be calculated by position divided by elapsed time. Since the fps is set to 60 fps, the elapsed time will always be 1 / 60. After calculating the velocity of the mouse pointer, the data will be stored in a Kinematic object and passed into the calculateAcceleration function in VelocityMatching class. Then, the boid will update its current Kinematic variables based on the newly calculated SteeringOutput. Inside this update function, the position and orientation will be updated first based on the current velocity and rotation. Then, new velocity and rotation will be calculated using the acceleration from SteeringOutput multiples the elapsed time. In addition, the Kinematic variables will be clipped if they exceed the maximum speed.
For this part, there is not much parameter tuning as the boid is simply accelerating to the mouse's velocity. The main parameter to be set is the maximum velocity of the agent, which can be set high enough to accommodate the velocity of the mouse pointer. In this case, a maximum velocity of 10000 is sufficient as the mouse velocity is unlikely to exceed that number if the pointer is within the window.

Part 2: Arrive and Align
In this part of the assignment, the requirement is to implement the Arrive and Align Steering behaviors, which cover position and orientation matching. For both of the classes, the implementation was similar to the previous part, and I mainly followed the pseudocode from the textbook and lecture. These two behaviors will use the mouse click location as the target position for the boid to arrive and rotate at.
Arrive:
To get the position of a mouse click, I used two functions called isButtonPressed and getPosition in the SFML library. When I left click on the screen, these two functions will acquire the position, and the data will be put in a Kinematic object called "mouse." This object will become the target for calculating the new acceleration. In the calculateAcceleration function, the

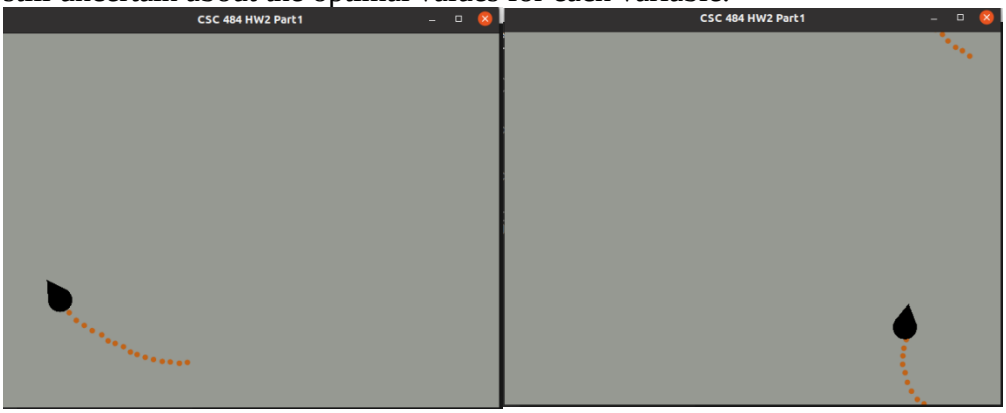
variables that I had to experiment with were maxSpeed, maxAcceleration, slowRadius, and targetRadius. The two parameters I adjusted the most were maxSpeed and slowRadius. Adjusting the maxSpeed parameter will directly affect the time it takes for the boid to arrive at the target. For example, when the maxSpeed is set to 100, the boid will become extremely slow when it is inside of the slow zone. Conversely, when the maxSpeed is set to anything over 400, the boid moves too quickly and can give the impression of being thrown from one place to another. After experimenting with different values, I found that a good range for maxSpeed is between 200 to 250. For slowRadius, I initially set the value to 100. However, the movement of the boid felt like it was slamming on the brakes when it reached the target location. As a result, I continued to increase the number until I found that a value around 380 worked well for the boid to decelerate. Making the value higher would just increase the time for reaching to target location. For targetRadius, I started with 1, but the boid will go past the target location and oscillate around it. So, I continued to decrease the value to around 0.1 and realized that anything below 0.5 seemed to work fine.

Align:

For Align, I used the mouse position to determine the orientation of where the boid should rotate to. Specifically, the target orientation of the boid is determined by calculating the angle between its current location and the target location of the mouse click. For this behavior, the parameters I had to experiment with the most were maxRotation, targetRadius, and slowRadius. Similar to the arrival behavior, adjusting the maxRotation and targetRadius parameters also had a noticeable impact on the rotation speed. High maxRotation and low targetRadius values can increase the rotation speed significantly. In addition, I had to consider the boid's speed to ensure smooth rotation and movement. Through experimentation, I found that setting the maxRotation to 10 and slowRadius to around 4 produced a smooth rotation behavior.

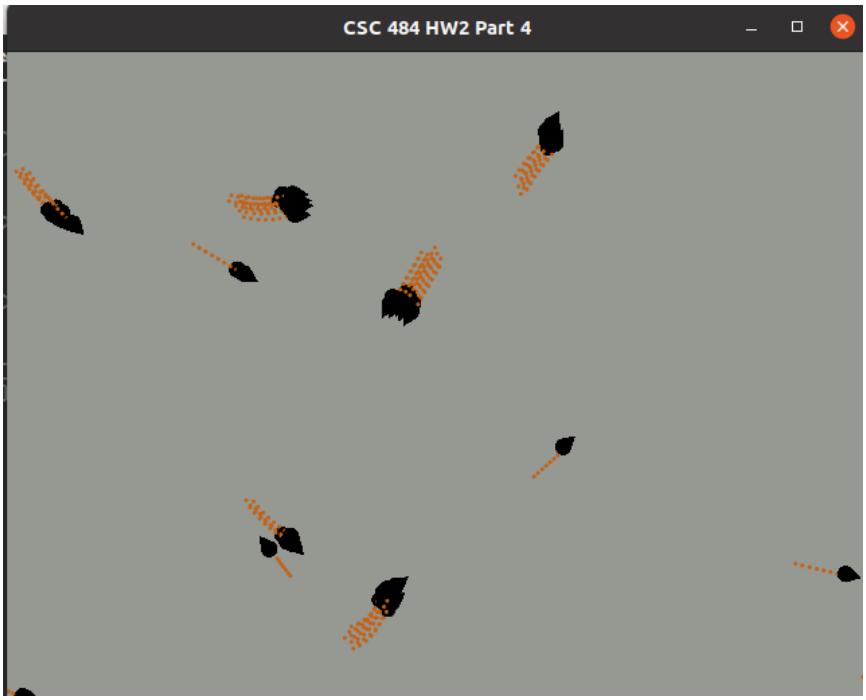


Part 3:
To implement the Wander behavior, I followed the design presented in our lecture and textbook but omitted the Face behavior. Throughout the process, I had to adjust several parameters, including those in the last part of the assignment. For the Wander behavior alone, I think the most important variables wanderOffset and wanderRadius. These two variables essentially determine where the boid will head next. wanderOffset determines how far away from the boid's current position the target will be, while wanderRadius determines the radius of the circle the target will move around. I found that if I set both values to be somewhat higher than 50, the boid would wander the way I wanted. However, I didn't observe any significant differences in behavior when I gradually increased both to 250. Another parameter that did not seem to have a significant impact is wanderRate. I found that as long as the wanderRate was set to a value higher than 10, the boid would randomly rotate in different directions. If the value is something like 1, the boid will just move straight the entire time. In addition, I also adjusted the maxSpeed to a lower value so that the boid moves slower and appears to be more in control during its wandering behavior. The boid will become shaky and unstable if the speed is too high. Lastly, a variable that I had to change was the timeToTarget in AlignBehavior class. In other parts of the assignment, I had set it to be 1 / 60 because of the fps limit. However, this produced an oscillating behavior where the boid frequently changed its rotation, which looked unnatural. Therefore, I tested multiple values and found that 0.1 was the closest value to what I wanted, even though there was still some slight oscillation. This behavior was the hardest to test as I am still uncertain about the optimal values for each variable.

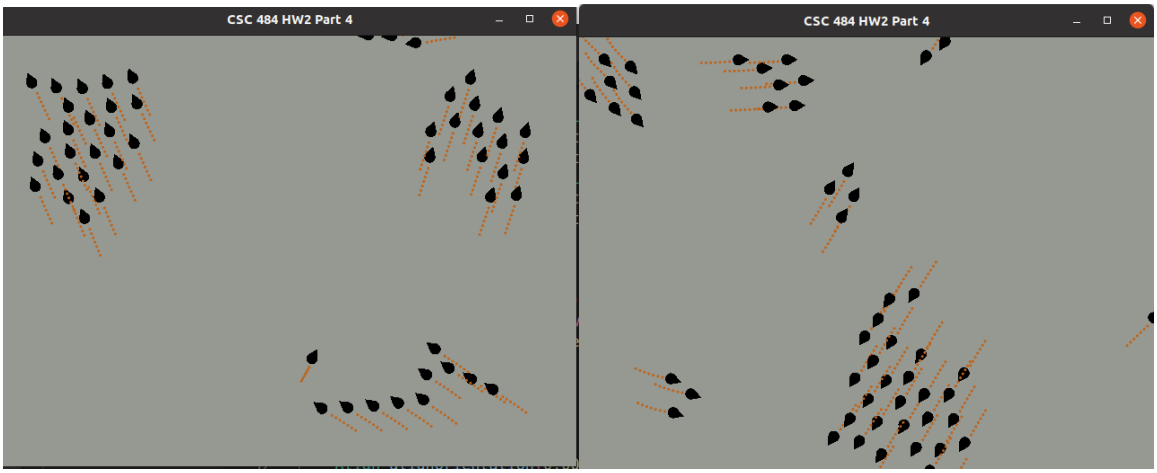


Part 4:
For Part 4 of the assignment, I implemented Cohesion and Alignment behaviors in the Boid algorithm, using the design outlined on https://eater.net/boids. In addition, I also incorporated Separation from the textbook. Unlike other Steering behaviors, I had to add a vector to hold all the boids to their constructor for them to calculate the desired acceleration. Each boid had to loop through this vector to determine if any other boids were within their visual range or threshold. If a target is within range, it is considered a neighbor. Then, it will try to match the average velocity of its neighbors while also moving toward the center of mass. In addition, Separation will keep them from getting too close to each other. In terms of implementation, it was not particularly difficult, and the most challenging part was determining the optimal values for the visual range and threshold. The visual range is used in both Cohesion and Alignment behaviors. If the visual range is too high, the boid will have more neighbors, leading to an overly cohesive behavior. There will always be a big group of boids. If the value is too low, then the boids will

be on top of each other. In addition, I observed that unless the visual range is set to a value that is about the size as the sprite, the boids tend to form a large group until they reach the screen boundary and reappear on the other side. For Separation, the variable I had to adjust around was the threshold. The threshold also determines the distance between boids, and I realized that this value must be smaller than the visual range. If the threshold is higher, then boids will repel each other and not perform the flocking behavior. Therefore, finding the right balance for these parameters was difficult to achieve the behaviors that I am looking for.



When threshold and visual range are small



When the variables are slightly higher