Name : Ajinkya S. Gavane Unity Id : asgavane

CSC 584/484 Spring 17 Homework 1: Movement

Due: 2/20/17 by the start of the class

Aim/Description:

The task of this assignment is to explore various movement algorithms. There are 4 parts to this assignment. Each based on Movement algorithms. The first task is to implement the basic kinematic motion by moving a character around the perimeter of the rectangular screen. Second task is to implement Arriving behaviour by directing the character at the location of mouse click. Third task is implementing the Wander behaviour. The final task is to implement Flocking behaviour using the Boids algorithm. Each task has been addressed in details and a report with images has been provided below.

Task 1: Kinematic Motion

This task required us to create a data structure to hold steering & kinematic variables. Updating these variables appropriately by getting the orientation from the direction of motion, direction of motion from orientation.

My character starts from the lower left corner of the screen. From there it traverses along the edge of the screen in anti-clockwise direction while leaving breadcrumbs and the after image of its motion enabling us to see distinctively its motion. My character is made of a circle and a triangle and triangle points in the direction of motion.

I have implemented it using 2 methods. First by making the character traverse around the perimeter by monitoring it and second, by making it seek each corner one after the other in an order. Thus, I have implemented the intuitive approach and the kinematic seek algorithm to complete this task. I found the later find to be more simple, since it showcases the reusability of the kinematic seek algorithm.

While implementing the Seek algorithm, I have tried two methods of changing the character's orientation. One by using a method named 'getNewOrientation()' which orients the character immediately towards the direction of motion. And other method orients the character smoothly towards the target location. Although this transition of smooth rotation of character towards orientation looks good, but if we have a large max speed, the character reaches the location before orienting itself towards target location. Below are the two images Fig1. and Fig2. of the methods I discussed. Left one is the first method and the right image is the second method.

I played around with these variables and came up with the following setting of variables which made the animation looked good: maxSpeed = 20, maxRotation = 0.3 rad.

Further I implemented the fusion of the two methods discussed above, the character uses smooth transition, until it gets near to target location where it uses the 'getNewOrientation()' method. Thus, answering the problem regarding smooth transition discussed above. You can see the third image.

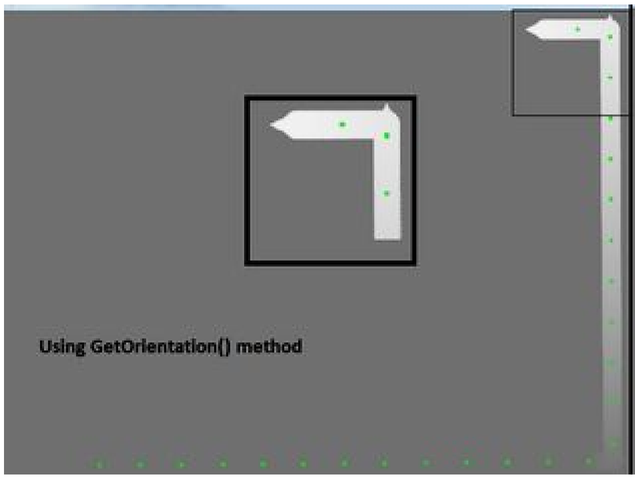

Figure 1.

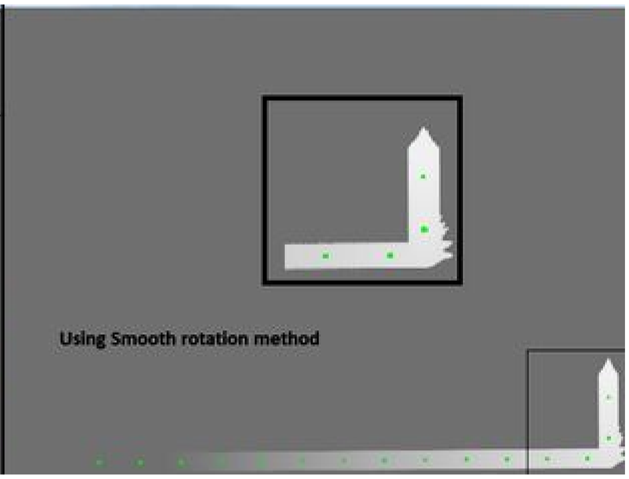

Figure 2.

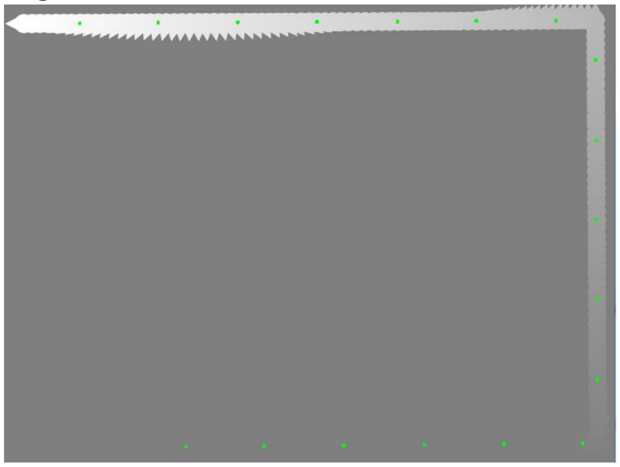

Figure 3.

Task 2: Arrive Steering Behavior

In this task we have to implement arrive movement algorithm by making the character follow the location of mouse click. While doing that, we also have to orient the character in the direction of motion.

My character starts from the top left corner and as a default it tries to arrive somewhere at the center of the screen. The target locations are dynamic and can be change anytime between any arrival, the character will adjust its kinematics accordingly.

I've tried different kind of approach for this task. In first I used simple kinematic seek. In this approach the character used to vibrate when reached at the target location. To solve this problem, I stop updating the character within some range of target location. In this approach I also played with orientation of the character, trying smooth change in orientation of character and direct changing the character's orientation in the direction of motion.

In second approach I introduced radius of satisfaction (ROS) and radius of deceleration (ROD). The values of variables that made animation look good are are: ROS = 0.5 * size of character, ROD = 80 and time to target acceleration = 0.5.

In the next approach I introduced ROS and ROD for the orientation components. Better looking values came out to be: ROS_rot = 0.01 rad, ROD_rot = 0.2 rad and time to target acceleration = 0.5.

Finally, I did the fusion of the last approach with getNewOrientation() method. In the previous approach I observed that when the orientation is within ROS_rot, the character's final orientation after reaching the target was a little bit off. I introduced one more radius of alignment (ROA), which is smaller than ROS. When the character is within ROA, it orients itself in the direction of motion. In all the above experiments, maxSpeed = 20 and maxRotation = 0.4 rad. Also I've kept the after image of the character. So that it is easier to see the the character's path, that it took along with its direction of rotation while orienting itself towards the target location. Below are the final images of this task. In figure 4, we can see the smooth translation and rotation of the character. Since the time and distance for the character to get to the target location is sufficient. In figure 5, the distance is small and the target has to orient itself quickly towards the direction. This is where ROA comes into picture. It checks if the object is very near to the target location and aligning it to the final orientation.

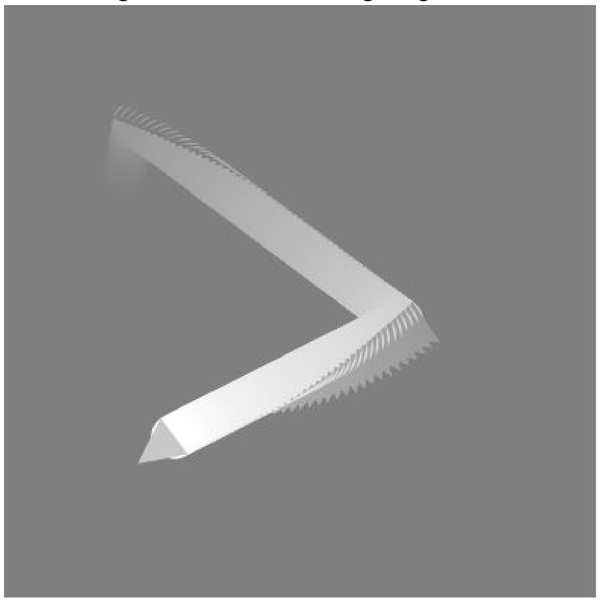
Figure 4.

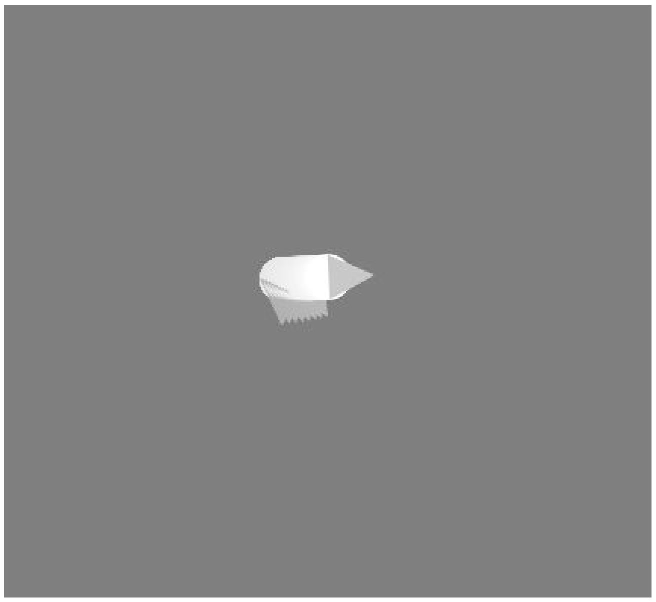
Figure 5.

Task 3: Wander Steering Behavior

In this task we have to implement wander movement algorithm and implement 2 different methods.

In my first approach, I randomly change the orientation direction of velocity vector using normal and gaussian distribution to generate the velocity components. If we use normal random number generator to generate orientation value and then updating the velocity component using cosine and sine rule then the character follows a circular motion. On the other hand if we use normal or gaussian distribution then we get a snake kind of motion. Below in figure 6, are the images describing this situation.

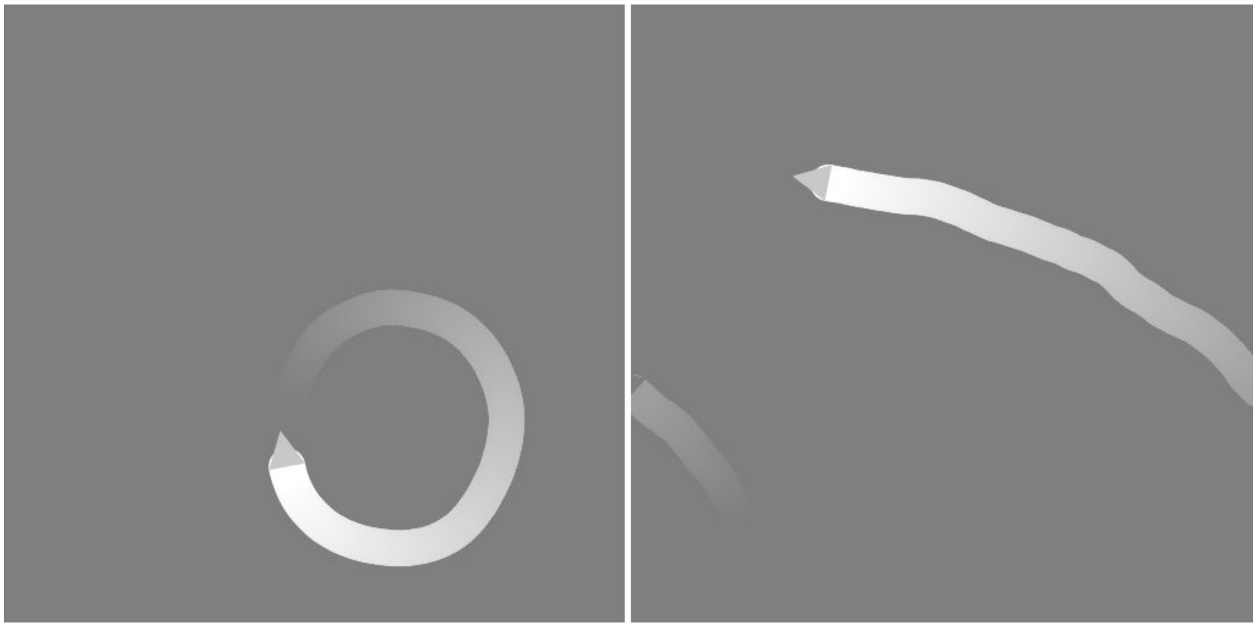

Figure 6.

In next approach I implemented the wander as a complex seek behavior. I projected a small circle at some distance in front of the character and randomly picked a point on that circle. Then I passed this point to the arrive algorithm. In this method, if your maxSpeed is low, then your character vibrates a lot along the z-axis (coming out of the plane) with some small angle. Also you need to have a large enough projected distance, i.e. the distance of the projected circle from the character, and appropriate wander radius. The following image in figure 7, is the result of this implementation, with wander radius = 2 and projected Distance = 25.

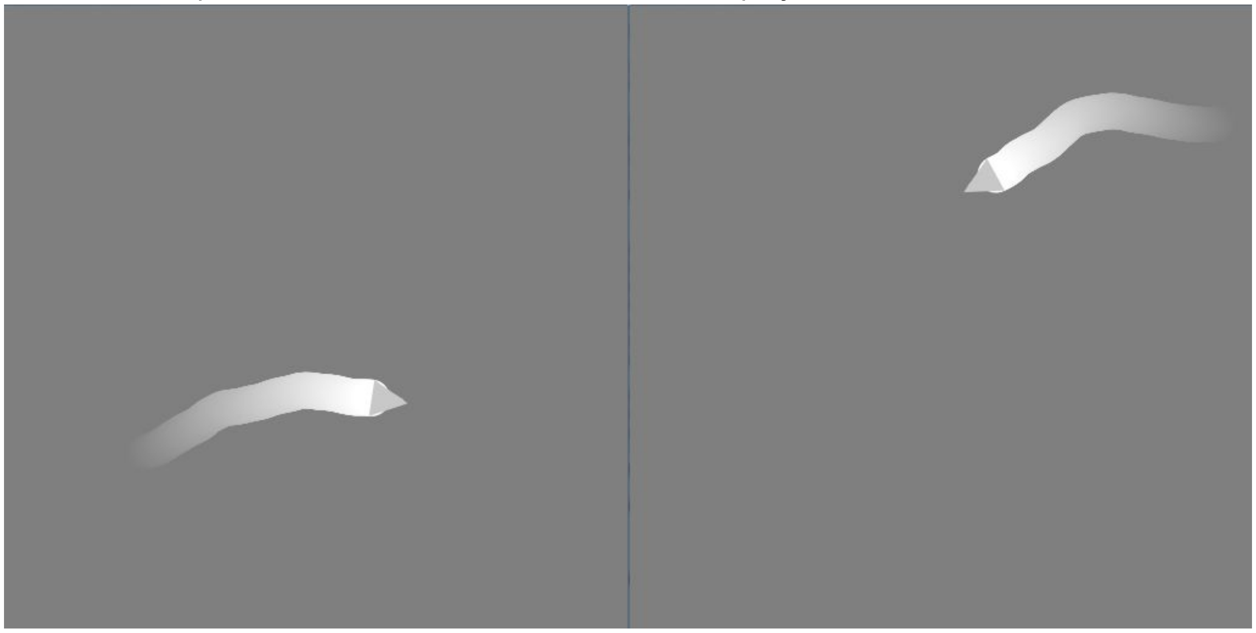

Figure 7: Wander using projected circle method

Task 4: Flocking Behavior and Blending/Arbitration

In this task we have to implement the boids flocking algorithm and do the analysis based on different parameters. In this section I'll refer boids as a bunch of characters.

My boids are generated with random position on the screen. I first added alignment (make character fly towards the center of mass of the flock) in the algorithm. After implementation I observe that the boids keeps going in a single direction. Then I added separation (collision avoidance) and cohesion (arriving at average position of the flock). I tried different sets of weight for each of these components algorithm. The weights that I found to be more settling are: separation weight: 0.4, cohesion = 0.3 and alignment = 0.3.

After implementation I observed that the velocity of boids was increasing steadily and there was no cap to it. So I put the cap on each character of the boid to the max speed which is 30. Also calculated the orientation based on the velocity direction of the boid.

To see the difference between the bounded flocking and unbounded flocking, I introduced one more parameter to my move() method. Which takes integer parameter and if the parameter is 0 then the flock is not bounded by the walls of screen, else they are contained in the screen space and the collision with the walls makes them bounce in the opposite direction.

The results of the two approaches are shown below with 100 boids. Figure 6, shows the unbounded version and figure 7 shows the flocking contained in the screen space.

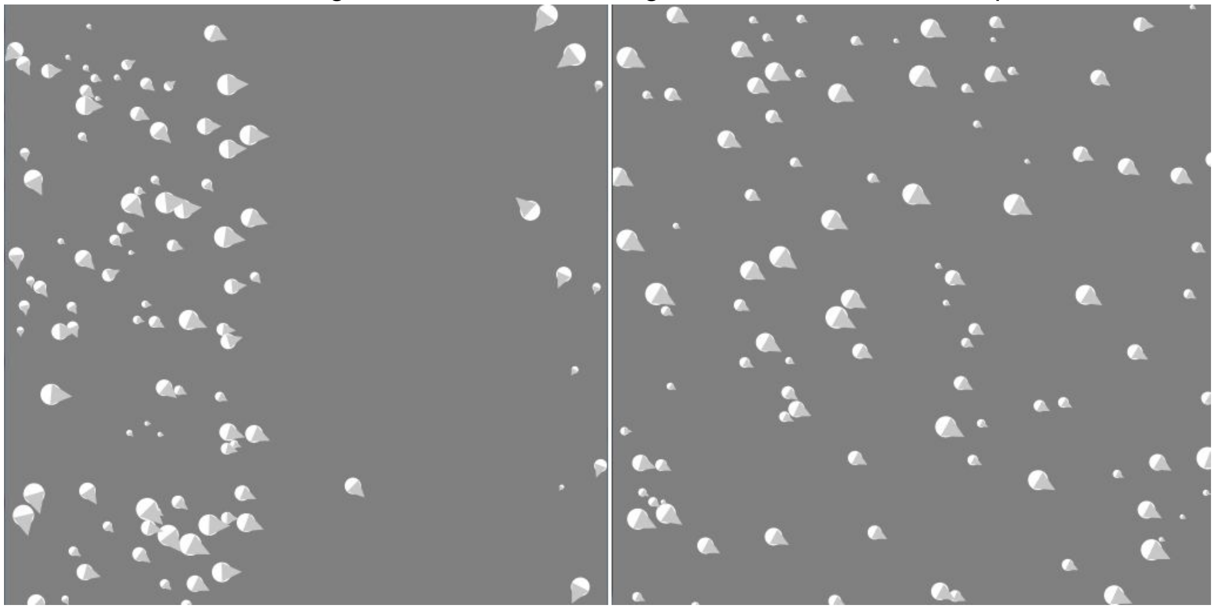

Figure 8. Bounded flocking. Time taken for the flock to orient in single direction decreases with increase in number of boids.

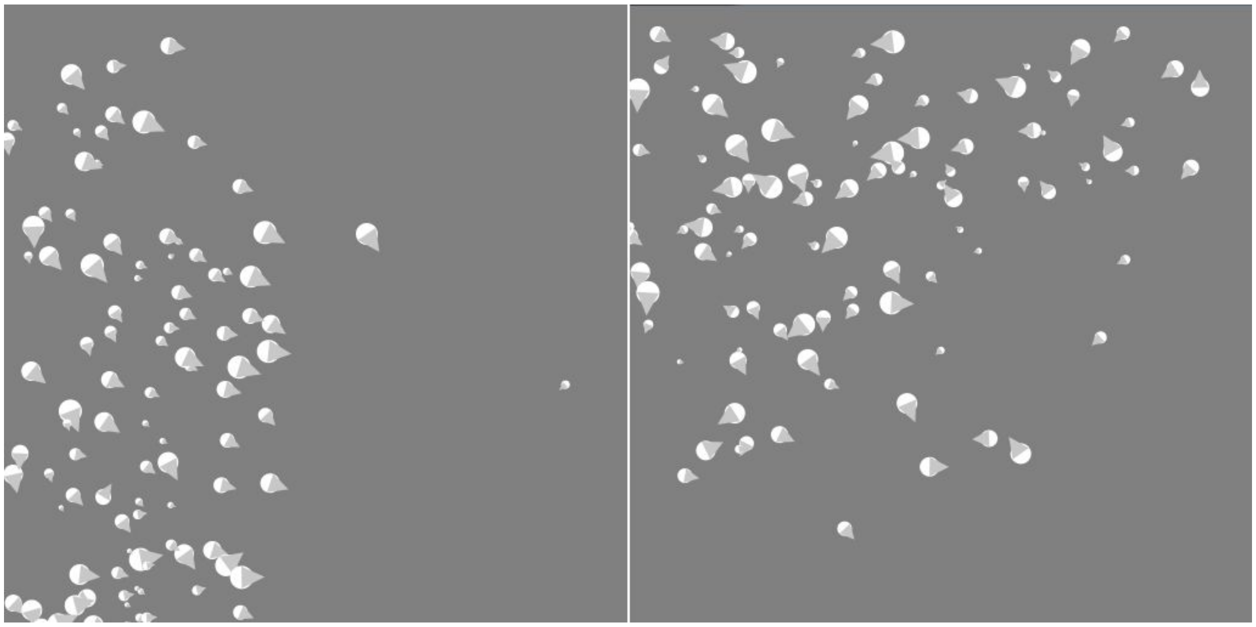

Figure 9: Bounded flocking. Things becomes harder if the number of boids are increased.

Here is the image of boids following the wanderer (the biggest boid is the wanderer). The weights of the components are: separation weight: 0.3, cohesion = 0.2 and alignment = 0.2 and towards Wanderer = 0.3.

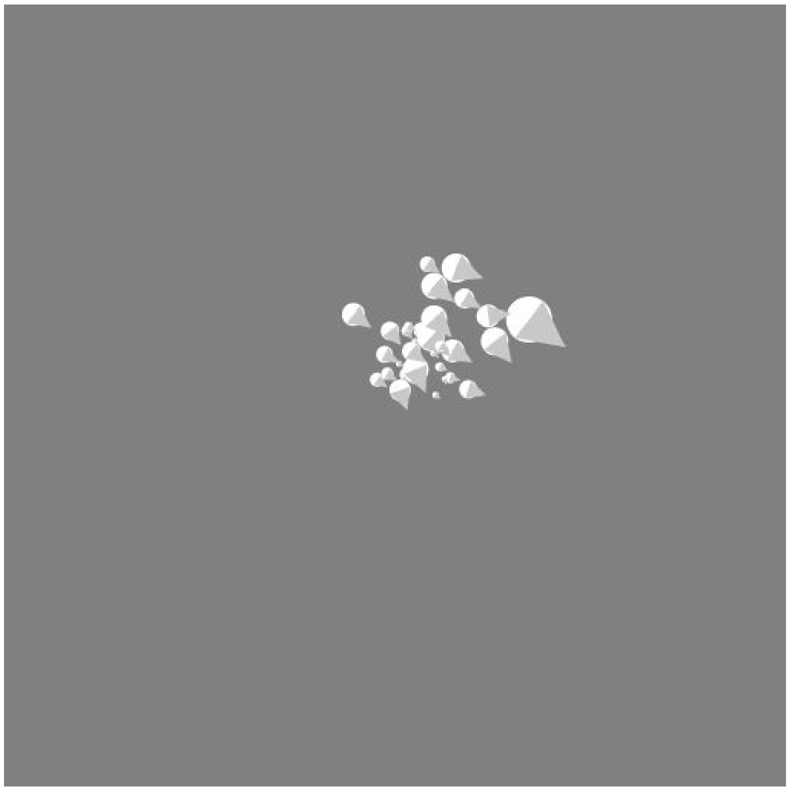

Figure 10: Flocking with one wanderer.

Here is the image of boids following closest of two wanderers (the biggest boids are the wanderer). The weights of the components are: separation weight: 0.3, cohesion = 0.2 and alignment = 0.2 and towards Wanderer = 0.3.