Jared Rivera

804603106

CEE/MAE M20

March 24, 2017

Final Project

1 Swarm Intelligence

1.1 Introduction

The goal of this program is to create an intelligent swarm of creatures that will closely mimic the movement patterns of swarms found in nature like schools of fish or, in the context of this particular code, a flock of swallows. The flock has no leader and no overarching goal in mind. It's motion as a whole is dictated by the motion of each individual swallow (called Boids in the code), and each swallow follows four instinctive rules that change its motion in terms of velocity and position.

First, each swallow will want to move toward the average position of its neighbors, this will be called cohesion. Second, each swallow will change its velocity to match that of the swallows its around, this will be called alignment. Third, each swallow will change its position in such a fashion that it doesn't collide with its neighbors, this will be called separation. Finally, each swallow will change its position in such a fashion that it strongly avoids the predator (a hawk named Charlotte), and this will be called predator avoidance. These rules are all very intuitive when put into a familiar context. Imagine walking in a group. You'll change your position and velocity in such a way that you'll stay with the group, and will also attempt to not bump into those around you. And in the unfortunate case you are being chased by something, chances are you'll run away.

The flock movement will be visualized over a specified domain in space and time with a plot similar to the one in The Game of life program, but will also be exported as a movie into an mp4 file at the end of the trial.

1.2 Model and Methods

The first step in this program is to initialize the time and space domains by setting up the boundaries and the time step data for the iteration later on. Next, the swallows are created and their positions and velocities are initialized randomly. After the swallows are created, Charlotte is created and her position is taken to be the center of the domain and her velocity is such that it will follow a different swallow in each iteration of the kinematic updates, as seen below.

```
Charlotte=struct('pos',[Nx/2,Ny/2],'vel',[Boids(g).pos(2),Boids(g).pos(1)]);
```

Now the plot is initialized and the mp4 is created and opened so that it can be written to in each iteration.

Next, the main loop is started. At the beginning of each loop, each swallow determines who its neighbors are. Using a double for loop, the swallow checks to see if each other swallow is within its horizon and if so, it's counted as a neighbor. A special case is set up to make sure the swallow doesn't consider itself as a neighbor, as seen on the following page. One can also see that an array containing the number of neighbors each swallow has is updated, to be used later in determining average position and velocity values.

```
            if k~=j
Neighbors(k).neighbors(j)=Boids(j);
Neighborcount(k)=Neighborcount(k)+1;
            end
```

Once the neighbors are determined, the cohesion and alignment acceleration components can be calculated. To make the motion realistic, a maximum value of velocity is taken and will be used to scale large values. These components, v1 and v2, are calculated in functions that are called in the main script. Each function has a very similar structure, so the cohesion(v1) function will be discussed here.

First, the neighbor's positions are summed into a variable named sumneighbors. Next, a conditional is set up to determine the average position in such a way that won't divide by zero in the case that a swallow has no neighbors, as seen below.

```
            if Neighborcount(l)~=0
avgpos1(l).pos=sumneighborsv1(l).pos/Neighborcount(l);
          elseif Neighborcount(l)==0
        avgpos1(l).pos=Boids(l).pos;
            end
```

Next, the vector that will contain the v1 value is created in accordance to the problem statement, with a similar conditional to prevent division by zero, and v1 is outputted to the main script. Again, v2 is done is a very similar fashion.

The separation(v3) component, is done slightly differently. Since the horizon for separation is smaller than that for cohesion and alignment, a new set of neighbors (called nosey neighbors in the code) is found with the exact same logic as the original neighbors. This is done inside the separation function, however, since it is only used in calculation of v3. Once again, the v3 vector is found in accordance to the method laid out in the problem statement and is outputted to the main script.

The final component, predator avoidance(v4), bears similarities to the separation. Charlotte's neighbors are determined in the function, and the main structure of the v3 calculation is copied over to allow the swallows to avoid Charlotte. The difference, however, is in the weight of their fear of Charlotte. In cohesion and alignment, the vectors are normalized by their magnitude, while in separation the vectors are divided by the square of their magnitude and then normalized, giving urgency and a higher weight to their separation in close distances. In the case of predator avoidance, the vectors are divided by the cube of their magnitude, giving yet more weight to their avoidance. Once the v4 vector is calculated in the same fashion as v1,v2, and v3, it is outputted to the main script.

Now the kinematic updates take place. In the kinematics function, the acceleration of each swallow is taken to be the weighted sum of the components outlined above, and is scaled to a maximum acceleration value if too large, similar to the scaling of the velocity values previously. Using semi-implicit Euler integration, the velocities of the swallows are then updated, which allows the position to then be updated by the same method. Once the positions are updated, the periodic boundary condition is enforced using a method similar to that discussed in class. An example is shown below for the upper boundary.

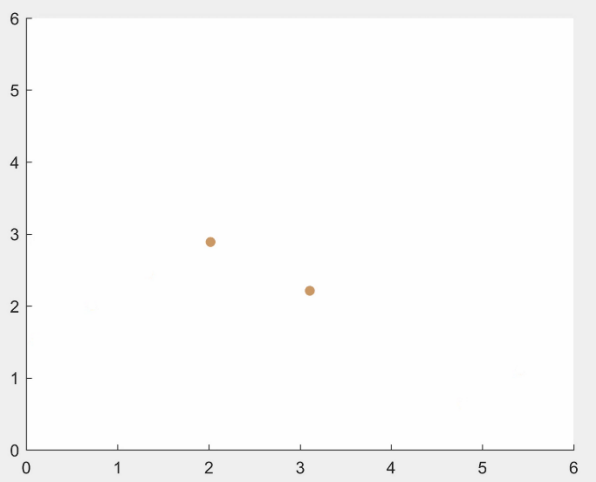
```
            if Boids(q).pos(2)>Ny
Boids(q).pos(2)=Boids(q).pos(2)-Ny;
            end
```

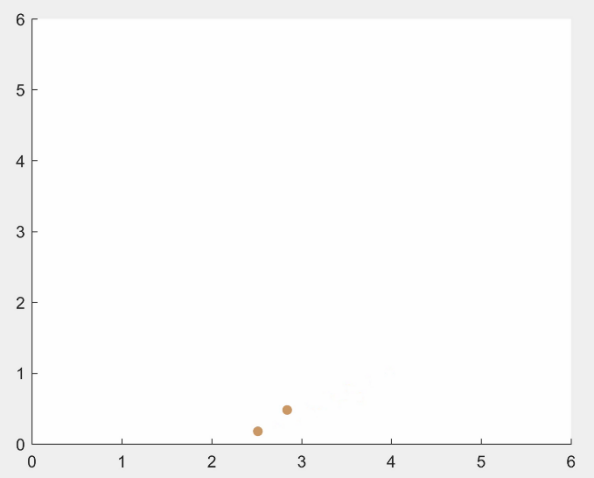
Finally, the plot handle is updated with the new positions of Charlotte and each swallow and the figure generated is written to the video file. Once all the iterations have been completed, the video closes and is rendered, showing the trajectories of the swallows and Charlotte through the entire trial.

1.3 Calculations and Results

Part a

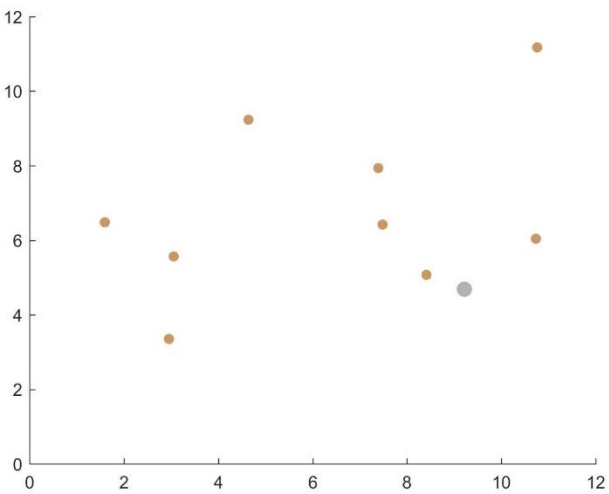
To showcase the functionality of the program, an example with only two swallows and no predator is shown below.



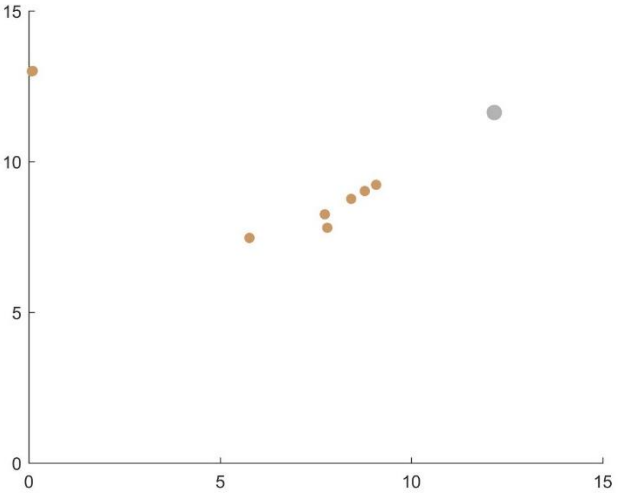


Part b

When the program is run, a figure is updated with the position of the swallows and Charlotte at each time step and displayed smoothly using the drawnow command. Each figure is also taken as a frame in the mp4 video file generated at the end of the script. An example of the initial and final positions of the swallows and Charlotte is shown in the figure below.



Initial



Final

1.4 Discussion

In part a above, the trial is initialized without the presence of a predator, and with only two swallows. Their position is random in the initial frame, and the final relative position is reached only a few seconds into the video included in this submission. They continue to translate once they've reached an equilibrium distance from each other and move with a uniform velocity and don't move any closer together or further apart. In part b, the program is shown to its full potential. The video is also included in this submission and shows Charlotte chasing the swallows across the domain. The initial frame has randomized positions, and the final frame shows the swallows in a formation that allows them to best escape Charlotte.

The way the different components of acceleration are weighted is with a weight vector who's components are designated by c_1 through c_4. Changing these adjusts how strongly the birds will cohere, align, separate, and avoid Charlotte. When cohesion is high but alignment is low, the swallows will group up, slow down and almost come to a stop at some point in the domain. When alignment is closer is value to cohesion, the swallows will stay closely grouped but keep a faster speed. Both of these situations assume a relatively low value for separation. When separation is high, especially when it's higher than both cohesion and alignment, the swallows will stay loosely grouped in a cloud shape but won't reach a very structured formation or fast speeds. This is similar to the predator avoidance factor in a sense. When the swallows strongly avoid Charlotte, they tend to stay relatively disbursed or in a line directed away from her. When predator avoidance is low, they cloud of swallows loosely forms near Charlotte's position and continuously moves away from her.

This program is collision free, in that it allows two swallows to occupy the same position in space at the same time. An odd thing I noticed while running trials, though, is that this isn't entirely the case for Charlotte. When she collides with a swallow, the swallow stays at almost her exact position, making it look like she's consumed it. Full discloser; I didn't mean for that to be the case but I'm glad it did and believe strongly that it has to do with the fact that the predator avoidance factor has a cubic in the calculation, which makes the values of acceleration for swallows at her position blow up and not be considered, forcing them to trace her path for the remainder of the trial. No set of c-values in my trials has been able to fully eliminate collisions, so I believe the way to do that would be through hard coding a reaction to collision. This could be done by changing the velocity of the swallows when they occupy the same point similar to the way the bots changed direction when encountering a wall in our previous programs. Or, a mass component could be added to each swallow and a conservation of momentum function could be applied, where the swallows bounce off of each other upon collision. This wouldn't be too complicated so long as the swallows are treated point-wise and thus oblique collisions aren't considered.

Currently, the swallows and Charlotte can't see past the boundary condition when checking for neighbors. In order to make that possible, we would have to specify a certain subset of positions near the boundary where such a condition is to be applied. One such method would be to check if the position of the swallow is within its horizon of the boundary, say, the left boundary. If so, calculate how far away it is form the boundary and subtract that value from the total width of the domain. This will give a subset of positions on the opposite wall on which the swallow will consider another object a neighbor. This would just have to be applied for each swallow and for Charlotte, and applied methodically in such a way that the tricky corner positions are checked properly.