Assignment 1 Report

Che Chang
The University of Utah
che.chang@utah.edu

ABSTRACT
This is the write-up for assignment 1, which covers the implementation of the basic and some of the advanced AI agent movement algorithms.

KEYWORDS
Game AI steering behaviors, seek, arrive, align, face, flocking

INTRODUCTION
In this write-up, we will demonstrate the implementation of kinematic motion, arrive, wander, and flocking behaviors of AI agents. We will also analyze what might be causing deficiency in the way these behaviors looked, and how we may go about fixing them.

KINEMATIC MOTION
Screenshot of the Kinematic Motion Demonstration

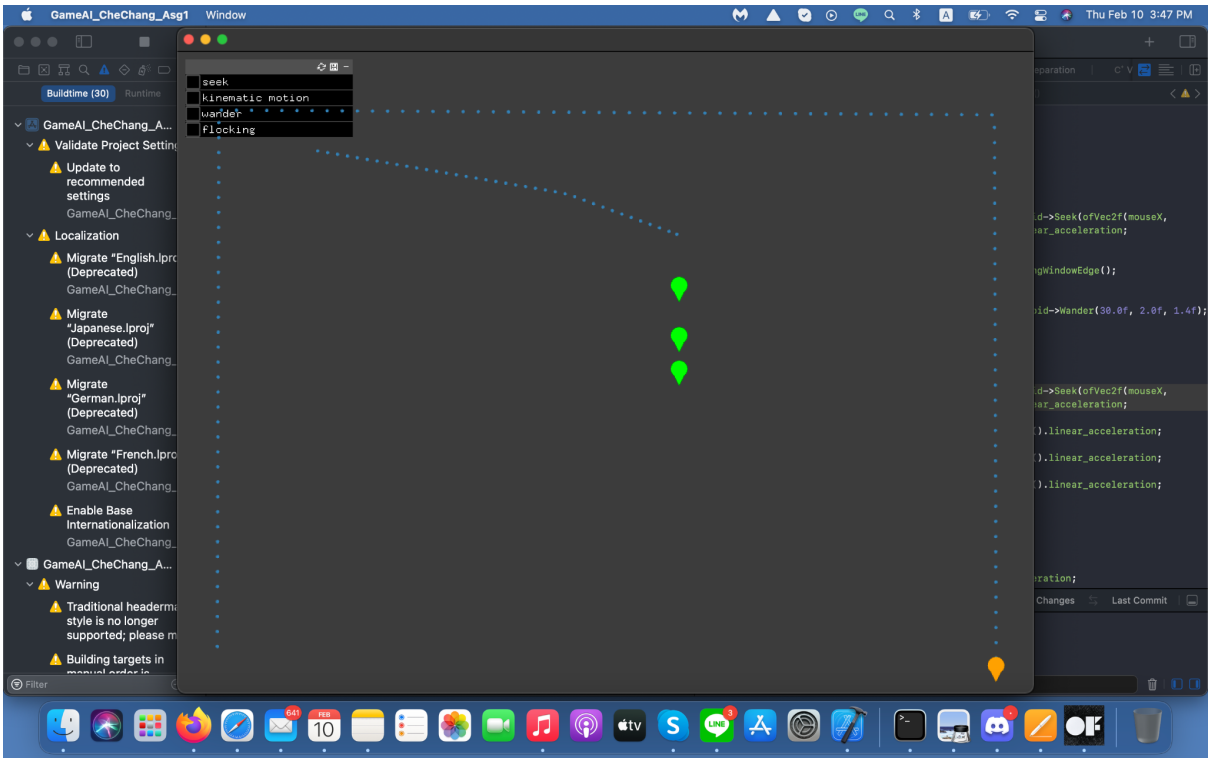

Figure 1: Kinematic motion along the application window edge

Observation
This simple kinematic behavior makes the character boid move along the application window's edge. It's implemented simply by changing the boid's direction when it enters the corner of the application window.

A thing worth noting is that this is a good practice for preparing us for the rest of the steering behaviors, because our steering behavior output is updated every frame, so coding the kinematic movement shown above in the same fashion helps us understand how steering behavior will be updated every frame.

To further explain this, please take a look at the pseudo code below:

```
SteeringOutput MoveAlongAppEdge() {
      case 1:
            return SteeringOutput {Vec2f(xxx, yyy), 0.0f};
      case 2:
            return SteeringOutput {Vec2f(xxx, yyy), 0.0f};
      ……
}
```

The steering behaviors discussed afterwards would be built with the same barebone.

BASIC STEERING BEHAVIOR: ARRIVE

Screenshot of the “Arrive” Steering Behavior

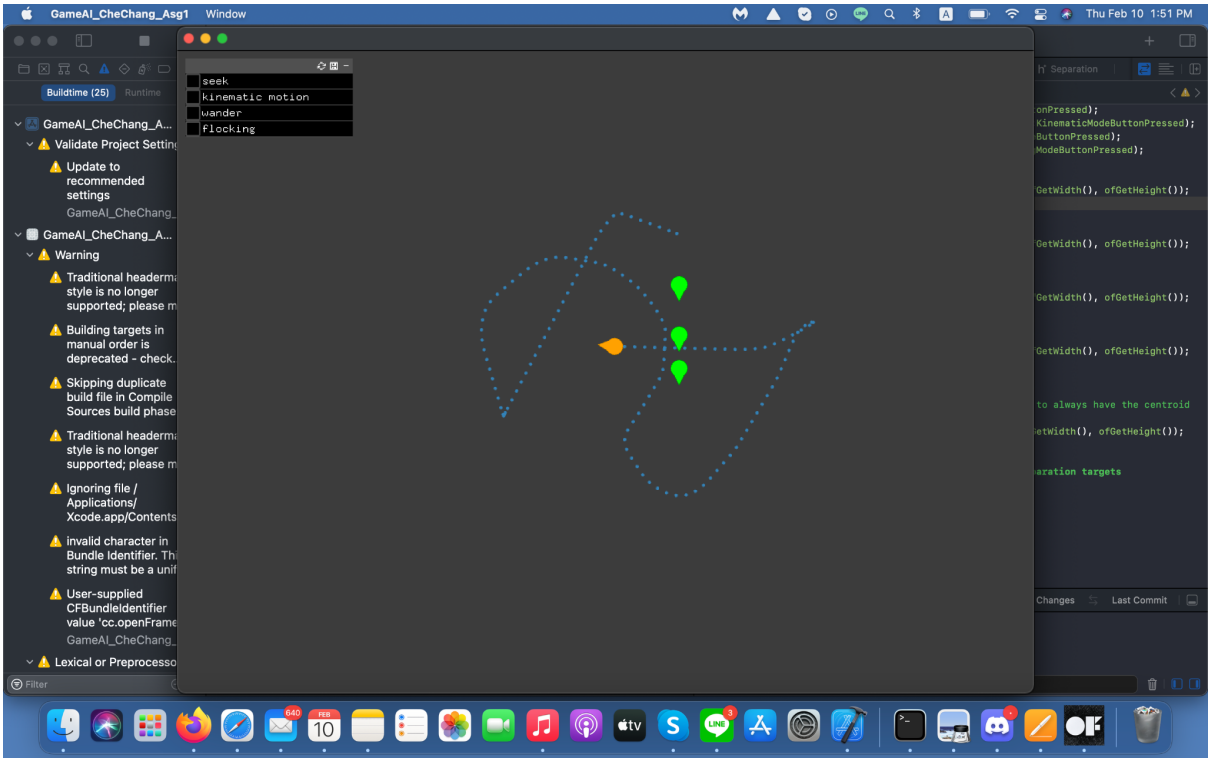

Figure 2: “Arrive” behavior

Observation

The “Arrive” shown above is implemented by defining a “slow radius” and “target radius”. When the character boid enters the slow radius, it interpolates its speed based on its

distance to the target position, and when it enters the target radius, it slows down until its speed is zero.

After calculating its steering linear acceleration, we then invoke a call to the function “LookWhereYoureGoing” to get the angular acceleration of our character boid. This behavior performs pretty well and does not have any weird-looking or unexpected movements.

I can’t compare the difference between multiple methods as I only managed to get only one working, also having a slow and target radius constraint in this implementation prevents a rigid body from acting unstable when approaching a target with high velocity and low acceleration and plus lower framerate.

ADVANCED STEERING BEHAVIOR: WANDER

Screenshot of the “Wander” Steering Behavior

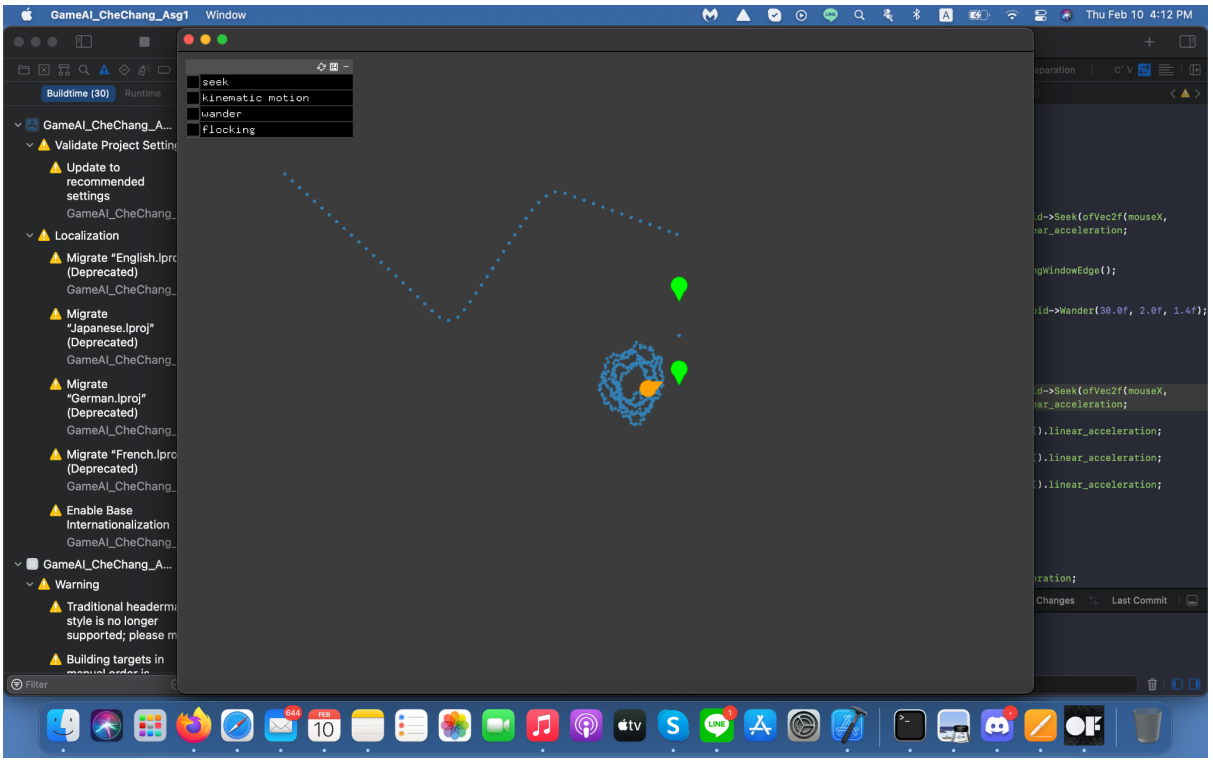

Figure 3: Jittering wandering behavior with smaller angular acceleration

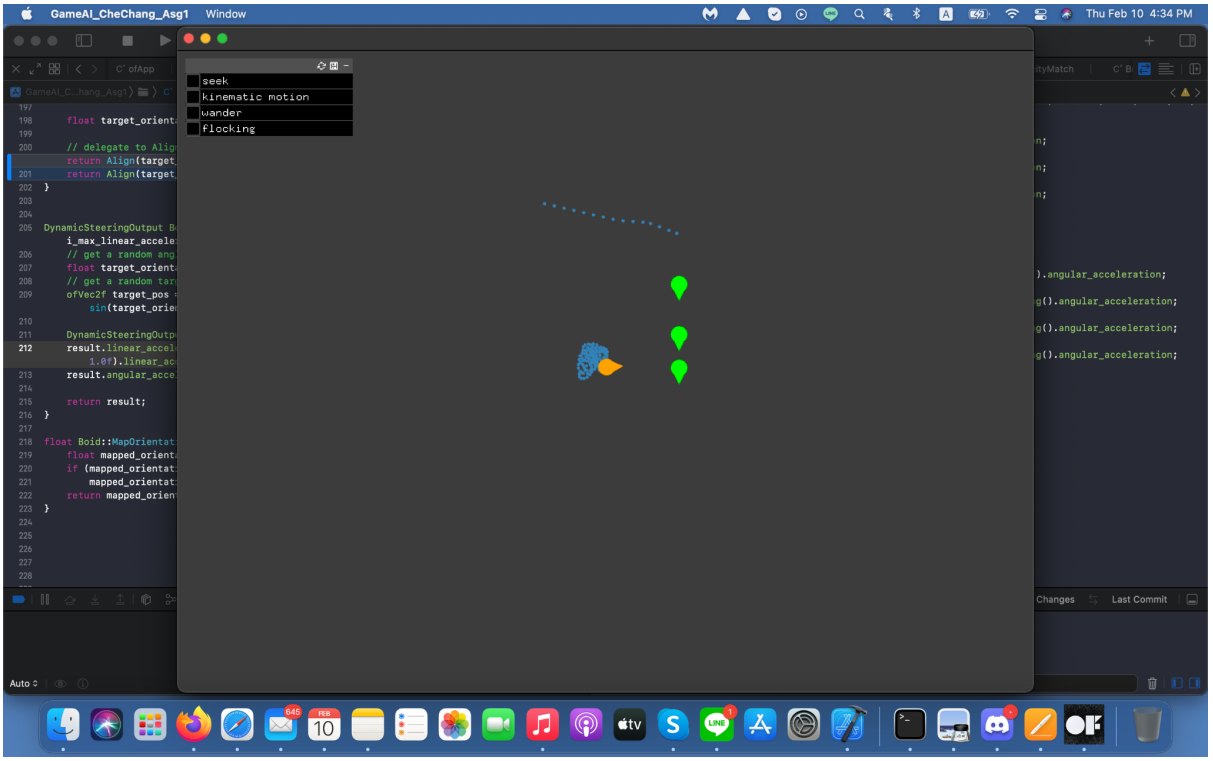

Figure 4: Smooth wandering behavior with bigger angular acceleration

Observation

The first deficiency I noticed in my algorithm is that when I calculated the character boid's target and delegate to seek, then invoke "LookWhereYoureGoing", the boid is always rotating the counterclockwise direction, no matter what the target orientation is.

The aforementioned deficiency caused the boid to appear to be running in circles. If I pass in a smaller angular acceleration to "LookWhereYoureGoing", the character boid would be jittering, due to its angular velocity is having trouble catching up to the seeking behavior; however, if I pass in a larger angular acceleration to "LookWhereYoureGoing", it would smooth out the boid's wander behavior, but still, the boid would appear to be circling.

To fix this issue, I would need to look into how to modify my alignment behavior algorithm to have the ability to figure out which way would be the fastest route to get to our target orientation, so it doesn't always output counterclockwise angular acceleration.

The rotation issue would be the main thing that caused the boid's wandering behavior to look weird. I also can't compare this implementation method with others because I only managed to get this one working.

BLENDING STEERING BEHAVIOR: FLOCKING

Screenshots of Flocking Behavior with Different Blending Weights

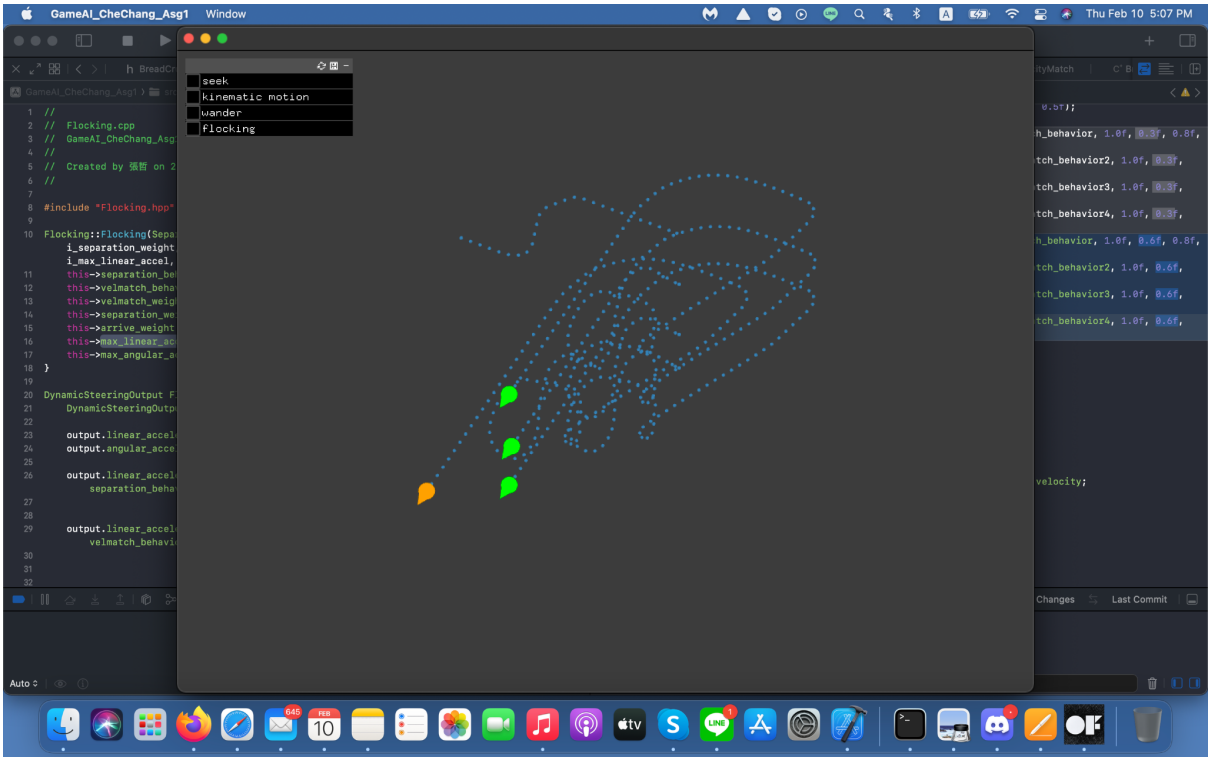

Figure 5. separation = 1.0, velocity_match = 0.6, arrive = 0.8, flock_boids = 3

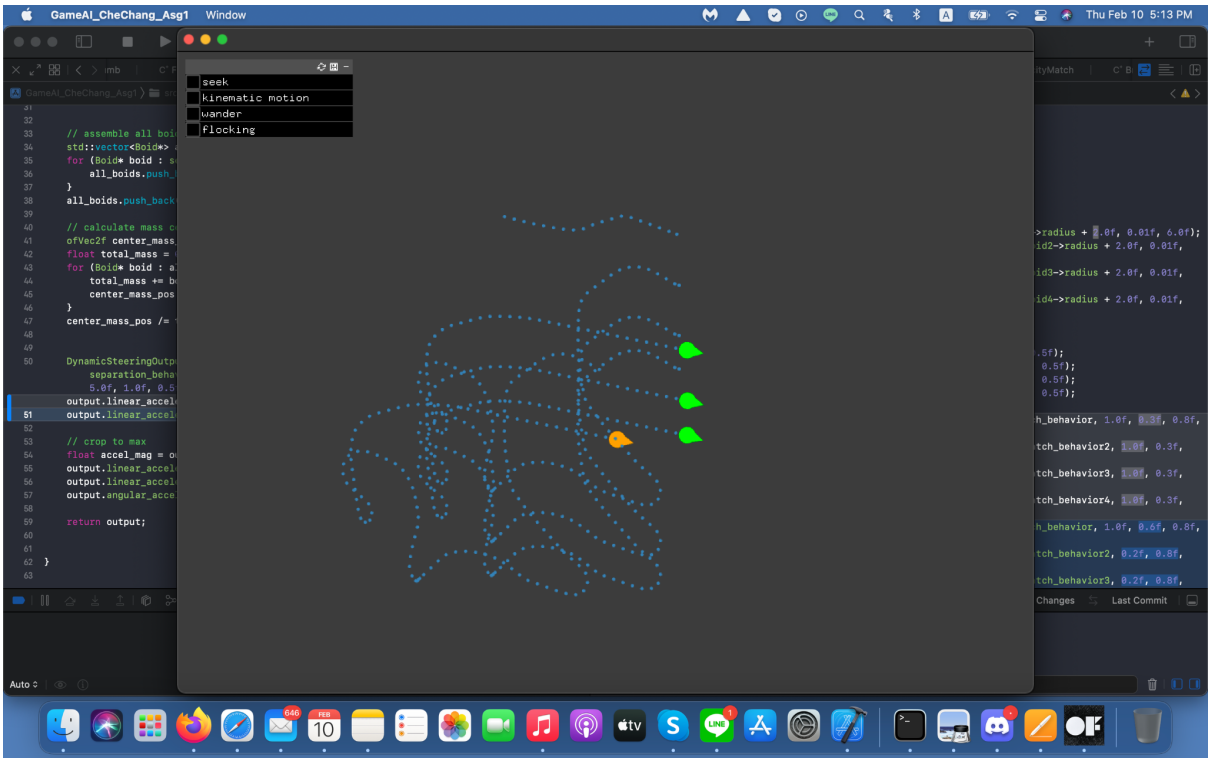

Figure 6. separation = 0.2, velocity_match = 0.8, arrive = 0.3, flock_boids = 3

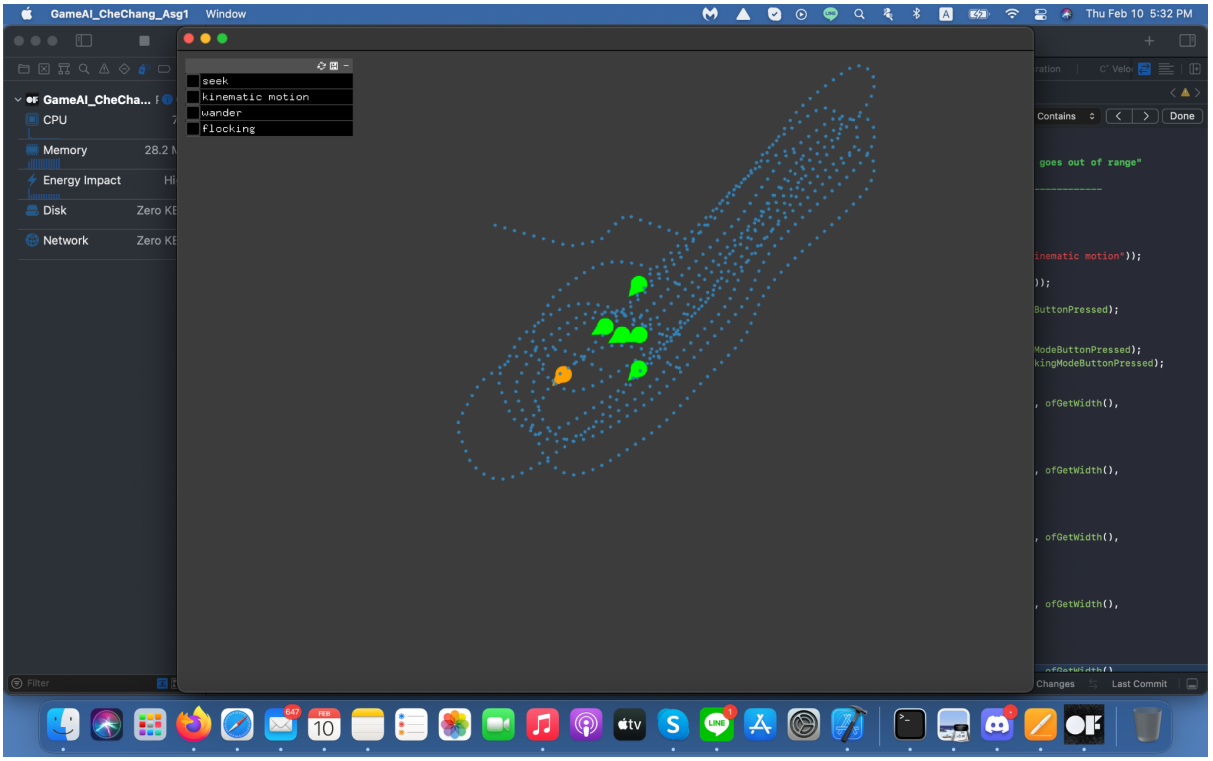

Figure 7. separation = 1.0, velocity_match = 0.6, arrive = 0.8, flock_boids = 5

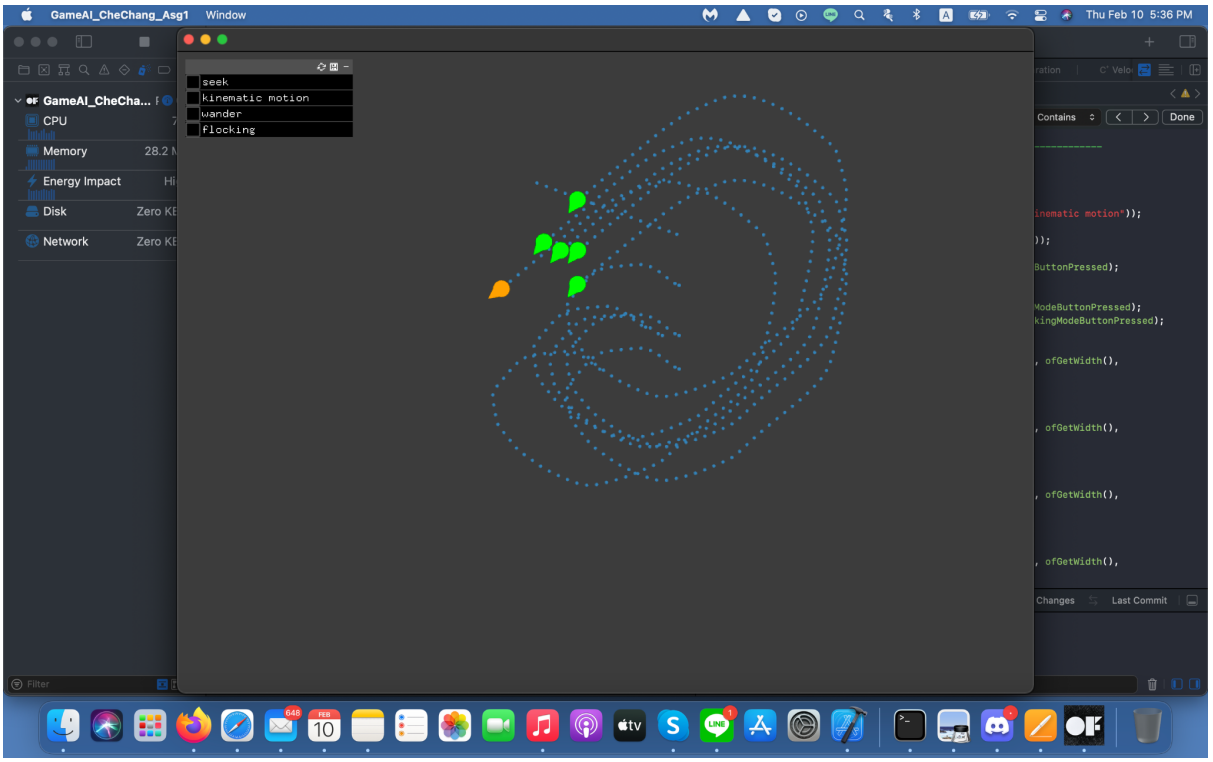

Figure 8. separation = 0.2, velocity_match = 0.8, arrive = 0.3, flock_boids = 5

Observation

1. What changes did you make to the algorithms to make flocking work?
    a. I had to multiply each behavior's linear acceleration output and accumulate them.
2. Are things easier or harder with more followers?
    a. It's harder with more followers, because the separation behavior becomes more unpredictable.
3. What happens if you have two wanderers and followers follow the closest wanderer?
    a. the followers would abandon the farther leader and form a flock behind the nearest leader.
4. Another thing I noticed is that my flocking behavior seems to be doing a great job velocity matching but does not perform well on "Arrive" behaviors (they don't seem to be arriving at the flock's center mass), this might be the velocity matching behavior limited these followers from going any faster, therefore not catching up to the center mass position.

REFLECTION

It was really fun implementing these steering behaviors, and these are great practices for service classes (Classes that provide output services for agents). These behaviors would surely be helpful for later on lecture materials, such as pathfinding, decision making. Movement algorithms serve as a base behavior for other more complex structures and algorithms, because the most fascinating thing for a game is to observe AIs move humanly and act like they have their own train of thoughts. I look forward to apply these movement algorithms to future assignments.