# Programming Exercise (PEX) 2

***Due: Lesson 19 (start of class on 5 Oct 2020)***

(100 points)

## Help Policy

**AUTHORIZED RESOURCES:** Course texts, course slides, lesson notes, on-line Java tutorials and language references, your instructor, and other cadets currently in CS330.

**NOTE:**

- Never copy another person's work and submit it as your own.
- Do not jointly solve any of the exercises.
- You must document all help received from any sources other than your instructor or instructor-provided course materials (including your textbook).

## Learning Objectives:

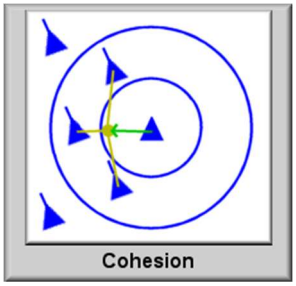
- Instantiate and launch an object of your application class from the static main() method.
- Create a Java program with collaborating classes using inheritance, association, aggregation, and/or composition.
- Use instance and static attributes appropriately.
- Use appropriate Java data structures for implementing aggregations.
- Use appropriate error handling so that your program will not crash.
- Use DrawingPanel for animating graphics and getting keyboard and mouse inputs from the user.
- Provide appropriate directions to facilitate ease of use for the user.
- Utilize the Vector330 class from PEX1 for position and velocity vectors.
- Document your Java code to provide comprehensive JavaDoc documentation for your program.

## Instructions:

This Comp Sci 330 PEX 2 assignment allows you to explore the creation of a more complex Java program that use several collaborating classes while reusing the Vector330 class that you created in PEX1.

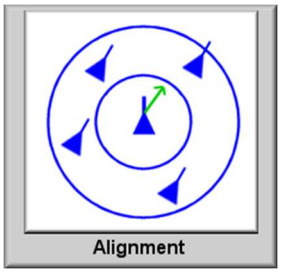
You will be exploring flocking behavior of autonomous entities, called boids. Flocking can be characterized by the three distinct component behaviors of Cohesion, Alignment, and Separation. Each behavior is based upon individual boids observing and reacting to their environment characterized by the behavior of near neighbors (those within a specified radius).

Cohesion[1] is defined as a steering towards the average position of neighbors and is perceived as an attraction to others of its kind. For Cohesion, each boid will look at the location of other boids of its species within a cohesion radius of itself, compute the average position of those boids, and create a unit cohesion vector directing it to that average location.

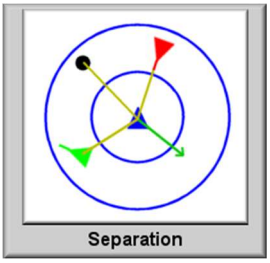



[1] Cohesion, Alignment, and Separation images taken from https://www.lalena.com/AI/Flock/.

Alignment is a steering towards the average heading of neighbors. For Alignment, each boid will look at the velocity vectors of the other boids of its species within an alignment radius, compute an average velocity vector, and create a unit alignment vector in the direction of the average velocity vector.




Separation is a short range repulsion to avoid crowding neighbors. For Separation, each boid will look at the positions of neighboring boids of its species within a small separation radius, create vectors moving it directly away from each of these near neighbors, and average these vectors to get the unit separation vector. Note, recommend you scale the distinct separation vectors to move further away from the closest neighbors.




Combine the Cohesion, Alignment, and Separation unit vectors scaling each one by an easily configurable weight and adding them up with the boid's current velocity vector to get its new velocity.

You will also implement an Evasion behavior moving away from a point where the user clicks in the simulation window. Only those boids within a specified radius will respond to the disruption by moving to a point directly away from the disruption that is the specified radius away from the disruption. The Evasion behavior takes precedence over the normal flocking motion with a one-time move (basically, it's a jump), but then the normal behavior resumes. A video demonstration is at: https://youtu.be/W2GHkH5eDd0.

**Flocking Simulation Functional Requirements**

1. The flocking simulation shall be displayed within a DrawingPanel window that can be easily resized under program control.

2. The flocking simulation shall include at least two distinct flocks of different species of boids. (Note, "Designing with extensibility in mind" is a great class design guideline to consider here).

3. Each boid in a species shall have common characteristics that can be distinct from the other species. These common characteristics shall include: size, color or image, speed, cohesion radius, alignment radius, separation radius, and a set of weights for combining the cohesion, alignment, separation, and current velocity unit vectors).

4. A right mouse click in the flocking simulation shall terminate and close the simulation.

5. A left mouse click in the flocking simulation shall cause a disruption such that any boid near the point of the disruption (within a given radius of the disruption point) shall evade by moving directly away from the point of disruption to the given radius distance (or window boundary whichever is the shorter distance).

6. The spacebar shall be used to toggle between a pause and resume of the flocking simulation.

7. All boids in the flocking simulation shall be randomly placed throughout the DrawingPanel window with random initial velocity vectors.

8. Individual boids shall move in an animated manner bouncing off the top, bottom, left, and right walls and responding to 'near' neighbors with cohesion, alignment, and separation behaviors in accordance with the settings for its species.

9. Boids shall be graphically represented in a manner that readily depicts its current heading.

**Flocking Simulation Non-Functional Requirements**

1. You shall generate and submit a comprehensive JavaDoc site for your PEX2 application that addresses all of the classes. Use the JavaDoc folder in your repository to hold the JavaDoc web site for your PEX2 program.

2. Your code shall comply with the following Java programming standards:

    a. camelCase notation for variables and methods.

    b. All CAPS for constants.

    c. CamelCase with the first letter capitalized for class names.

    d. Variables, constants, methods, and classes names so as to describe their purpose.

    e. Liberal use of white space, via skipped lines, for readability.

    f. JavaDoc header comments for each class and each method to include the @author attribute for classes and @param and @return attributes for methods as appropriate.

    g. Inline comments to describe the purpose of key chunks of code; a good rule of thumb would be to have and in-line comment for at least every 3-7 lines of code.

    h. The scope of variables shall be confined to the least possible to help manage complexity.

3. The design of the classes for this flocking simulation is part of the creative aspect of this assignment, but you shall have distinct classes for the Main (that launches the flocking simulation), the flocking simulation that controls the user's interactions and animations, a flock that aggregates boids by species, and individual boids. JavaDoc for a reference solution is provided for reference.

4. The class design of your flocking simulation shall comply with the Class Design Guidelines discussed in Comp Sci 330 Lesson 9.

5. The flocking simulation shall be implemented with appropriate error handling to catch and report errors while avoiding program crashes.

6. Create variables for each flock's radii and vector weighs for each of Cohesion, Alignment, and Separation so that you can easily modify them for both testing and tailoring of behaviors. (For PEX3, the flocking simulation extension will have you provide access to these parameters on a GUI so that the user can adjust during the simulation).

## Bonus Points

Up to 5 bonus points may be awarded for additional features that can include, but not be limited to, obstacles to avoid, image(s) for background and/or boids, predator species that chase, and/or prey species that evade.

## Hints

- Carefully consider and develop your class design for the flocking simulation explicitly referring back to the Class Design Guidelines discussed in class before ever starting to code the flocking simulation.

- Plan out an incremental implementation and test strategy so that you can build-a-little and test-a-little on your way to a complete implementation. Below is one such incremental strategy that you could use.

    1. Create a flock for one species of boids and have it render with each boid in its initial location with the right click termination logic.

    2. Have the boids of the one flock move randomly bouncing off of the ways.

    3. Add in the Alignment computations.

    4. Add in the Cohesion computations.

    5. Add in the Separation computations

    6. Add in a second flock to represent a distinct species.

    7. Add in the disruption and evasion behavior.

    8. Consider adding in additional bonus features as time and interest allow.

- This assignment is intentionally designed to be fun. Please be creative and enjoy.

## Notes on Cohesion, Alignment, and Separation Calculations:

- **Cohesion**

For each other boid in the flock *(Note: It's good to compare the boid to itself.)*

    If the distance to the other boid is less than the cohesion radius then

        Add the other boid's position vector to the sum of position of vectors

        Increment the number of nearby boids

Compute the average position vector by scaling the position sum vector by 1 over the number of nearby boids.

Compute the cohesion vector as the average position vector minus the current boid's position, then make this a unit vector.

- **Alignment**

> For each other boid in the flock *(Note: It's good to compare the boid to itself.)*
>
> If the distance to the other boid is less than the alignment radius then
>
> Add the other boid's velocity vector to a sum of velocities vector.
>
> Compute the alignment vector as the unit vector of the sum of velocity vectors.

- **Separation**

> For each other boid in the flock *(Note: It's good to compare the boid to itself.)*
>
> If the distance to the other boid is less than the separation radius then
>
> Compute an offset vector as the current boid's position vector minus the other boid's position vector.
>
> Compute the unit vector of the offset vector and scale it by the separation radius minus the distance between the current boid's position and the other boid's position.
>
> Add the vector from the previous calculation to a running sum vector.
>
> Compute the separation vector as the unit vector of the average separation vector.

- **Combining Cohesion, Alignment, Separation, and Current Velocity**

> For each boid in the flock
>
> Create the new velocity vector by multiplying each of the following by their respective weights, adding the resulting vectors together, and then scaling the sum vector by the flock's speed.
>
> - Boid's current velocity vector (recommended initial weight: 0.4)
> - Boid's cohesion vector (recommended initial weight: 0.2)
> - Boid's alignment vector (recommended initial weight: 0.2)
> - Boid's separation vector (recommended initial weight: 0.2)
>
> After all boids in the flock have their new velocities, make their new velocities into their current velocities, move them, and draw them.

## Submission Instructions

- Your PEX2 program will be submitted via your PEX2 GitHub cloud repository. The link to create this repository is posted with the PEX2 assignment in Blackboard.

- As back-up, you can email your PEX2 zip files to me at steven.hadfield@afacademy.af.edu.