During the development of my "boids" simulation program, I maintained a Github repository to track the changes I made. One aspect that underwent several revisions was the representation of boid movement. At first, I experimented with using the variables "speed" and "angle," and then transitioned to "dx" and "dy" to denote velocity along the x and y axes, respectively. Ultimately, settling on "dx" and "dy" proved to be more descriptive and intuitive for capturing velocity in the x and y directions. This naming convention significantly enhanced the comprehensibility and maintainability of the code, resulting in a cleaner and more readable implementation of boid movement.  
The usage of "dx" and "dy" in coding conventions aligns with common industry practices and has emerged as a standard convention in many programming communities. These terms, derived from "delta x" and "delta y," are widely adopted to signify changes or differences in values along the x and y axes. Their descriptive and intuitive nature makes it simpler for developers to work with code related to coordinate systems or two-dimensional movement.

By adhering to standard naming conventions like "dx" and "dy" promotes consistency across different projects and codebases. By following a shared convention, developers can facilitate collaboration, code sharing, and mutual comprehension. This consistency also contributes to improved code readability and maintainability.  
The documentation of coding conventions, including the usage of "dx" and "dy," varies across programming languages and frameworks. However, many programming communities and organizations offer guidelines and documentation on coding conventions, encompassing variable naming practices. These resources aid developers in understanding and effectively adhering to standard conventions.  
While the level of documentation may vary, it is generally advisable to comprehensively document code, including the meaning and usage of variables such as "dx" and "dy." This documentation can be integrated into the code itself, through comments, or in separate documentation files. By documenting the code, developers can simplify comprehension and future maintenance of the codebase.  
Another thing that I tested thoroughly was the settings to use to configure the simulation; ultimately, I decided on the 7 settings: pause, highlight, avoid walls, draw lines to neighboring boids, draw the vision cone, draw avoidance cone, and draw force vectors. I chose this many settings as they seemed necessary to get a deeper understanding of how the boids worked.  
I also thought about the idea of using a "target spot" that showed where a particular boid was going, but this was fairly benign so I decided to leave it out.

Early on in the project, I experimented with different boid shapes, the usual case is to use triangles as it clearly shows the direction the boid is travelling to the user. I didn't like this approach however as the boid in code is a single point in space and the triangle didn't clearly illustrate this. When I saw Jesse Lee's Java Script boids (<https://jumpoff.io/blog/implementing-boids-in-javascript-canvas>) I got the idea to use circles,