Flocking Behavior:
A comparison of languages, implementations, and real-time adaptability

Team Members:
Adam Storm
Javeed Ghorbani
Jeremy Crowe
Alex Fincham
Joseph Bennett

Project Purpose:
Flocking behavior is exhibited by a group of birds (similar in fish, insects, or herds) when foraging in flight. Computer simulations have been found to accurately emulate the behavior of such animals. We will implement a flocking behavior algorithm in both Haskell and Javascript, and compare the experiences of both projects. The goal will be to represent flocking with a realistic graphical representation and some form of user interaction.

The computer model representation of flocking behavior was developed in 1986 by Craig Reynolds. He coined the term boid to represent each “element” of a flock.

Flocking behavior consists of three key functions:
Separation: Each boid avoids crowding close flockmates.
Alignment: Each boid steers in the “average” heading of a close neighborhood of flockmates.
Cohesion: Each boid steers towards the average position of local flockmates.

Realistic Code Goals:
(Javascript)
- Implement flocking behavior algorithm
- Represent algorithmic evolution via GUI
(Haskell)
- Implement flocking behavior algorithm

Stretch Code Goals:
(Javascript)
- Implement predator-prey dynamic where user can interact with flock and present obstacles
(Haskell)
- Represent algorithmic evolution via GUI
- Implement predator-prey dynamic where user can interact with flock and present obstacles

Project Comparison:
Perform code / algorithm analysis of both implementations.
Comparison metrics include:
Code length
Ease of understanding
Code portability (level of abstraction)
Development time

Conclusions:
  Effectiveness of language in performing tasks
  Pros and Cons of both languages
  How well both languages cater to individual / group coding environments

**Time Estimation (assuming all members are working concurrently):**
Learning javascript : **6hrs** (Javascript subteam)
Learning GUI libraries in Haskell: **6hrs** (Haskell subteam)
Overall algorithm pseudo-code : **5hrs** (Haskell subteam)
Javascript algorithm implementation : **10hrs** (Javascript subteam)
Haskell algorithm implementation : **10hrs** (Haskell subteam)
Javascript Visual Implementation : **8hrs** (Javascript subteam)
Haskell Visual Implementation : **10hrs** (Haskell subteam)
Code comparison and analysis : **2hrs**
Project write-up and summary : **4hrs**

Total time commitment from each member : ~**35hrs**

**Basic Timeline:**
- **Week 1:** Algorithm Outlining / Learning libraries/language
- **Week 2:** Algorithmic Implementation
- **Week 3:** Algorithmic Implementation / GUI Implementation
- **Week 4:** GUI Implementation / Dynamic Behavior Implementation
- **Week 5:** Dynamic Behavior Implementation / Code Analysis / Project Write-up

**Risks:**
- The entire team understanding the algorithm in a timely manner
- How to portray each boid graphically
- Failure to implement GUI representation in Haskell
- Failure to create user interactivity (dynamic behavior)
  - What user interactivity is a realistic goal for our time constraints
- Teams reaching different points of development and becoming disjoint.
- Overextending our abilities and time constraints