Project Interim Report

“Me and da Boids”

A Processing Game Engine to Simulate Flocking Behavior

Daniel Hatakeyama, Ben Xiang, River Jordan

* This interim report represents approximately 50% functionality of the final product *

Deliverables

This report, in-progress code, interview.

Status Summary (10 points):

Work Done:

First Mock-Up Complete:

- Completed minimum viable boid simulation
    - Works with strategy pattern for arbitrary behavior and render functions
    - Boids are implemented as inheriting game objects, -> need for change

Completing final Deliverable

- Our first mock-up proved our ability to make simple game-like scenes; **we now focus on making an interface for arbitrary virtual object interaction and behavior. Our game engine will focus strictly on demonstrating complex CS architecture at a higher abstraction level**—graphical behavior will be left to third party libraries.
- Current work has solidly implemented designs for the entity system, entity manager, and base components.
- The design of the system / system manager is proving difficult to keep elegant, yet simple. This is our largest concern at the moment. Each time we find a solution, it lacks long term scalability. The current ideal solution to check all of our requirements of efficient runtime and loose coupling requires quite a bit of implementation, including multiple observers and listeners, memory caching, refactoring components, and careful planning.

Changes or Issues Encountered:

- Understand boid behavior and how to implement their behavior correctly via Processing canvas environment.
- Implementing the ECS pattern from scratch has been extremely difficult.
- Installing Processing 4 has been challenging to get it working on certain machines due to OS differences.
- Refactoring our current implementation to run at a more efficient way.

Patterns:

- Entity Component System - A design pattern not covered in our class
- Builder - EntityBuilder
- Extensive use of the strategy pattern
- Composition Pattern

Test Coverage Report:

- Professor Wright has given us the green light to use Processing and not strictly follow TDD.

Class Diagram (15 points)

A UML class diagram that shows all the classes and their relationships. Note which classes have been implemented and which still need to be created. Pattern use should be highlighted in this diagram.

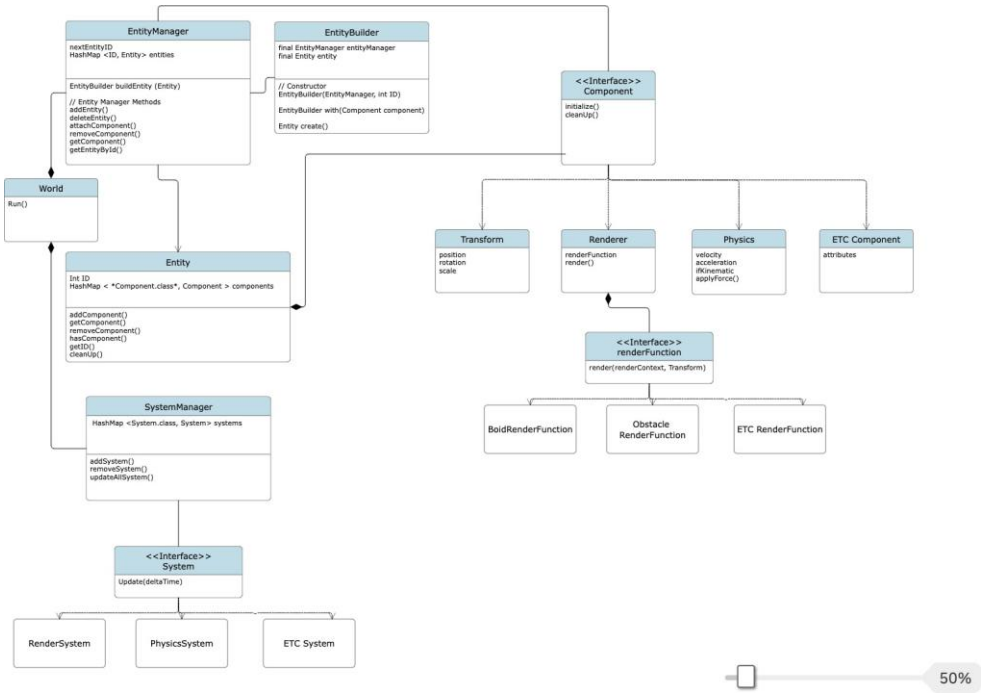

Fig 1. UML Diagram of ECS structured game engine, with scene agnostic capabilities.

Scenarios (10 points)

You must include at least one scenario for each major function in your application. You do not need to cover all variations of these functions. Just one variation is sufficient. This will force your team to agree on exactly what you are building and have one specific scenario to work towards implementing.

You DO NOT need to implement any step definitions for these scenarios. Meaning, they don't have to be executable tests. Implementing the step definitions and running the tests will be extra credit in the final submission.

Major Components:

- Entity
    - Must act as a UUID with a collection of components, works by composition over inheritance
- EntityManager
    - Handles all entities and adds and removes components as needed.
- EntityBuilder
    - Uses builder pattern to compose entities into complex game objects easily
- System
    - Given a subset of gameobjects with components that match a query, performs behavior / action responsibilities on the component datums.
- Component
    - Data that attaches to an entity that gives it the behavior
- ComponentManager
    - Handles all data attached to the entity, such as adding and removing data from an entity.
- PGraphics render contexts
    - Holds information about how the entity should be rendered, such as its color.
- Main Game Loop
    - Uses the 'world' or 'scene' to render all entities.

Plan for Next Iteration (5 points)

Provide an estimate of how much work still needs to be done. What are your plans for the final iteration to get to the project delivery?

Plan:

- Port boid code to the refactored ECS version code.

Recorded Demonstration (20 points)

The recorded video should be brief, 5 to 10 minutes; all team members should participate. Include the recording in your repo or provide an external link for viewing. Sections for the recording:

- Introduce all team members.
- Demonstrate your progress so far, showing the existing functionality.
- Identify the technologies and patterns used.
- Show some of your tests and at least two BDD scenarios.

The video for our demo is available inside of this attached folder.

NOTE: We did not show BDD because as we mentioned above, since we are using the Processing framework, We started our project with a graphical canvas interface to keep the scope reasonable over the timeframe.

Grading Rubric:

The point breakdown for Homework 7 is as follows:

Section	**Points**
Video Recording	20
Status Summary	10
Class Diagram	15
BDD Tests	10
Plan for next Iteration	5
Repository Submission with current code (includes required unit tests)	10
Total	70