Ocean Simulator: Iteration 4

https://github.com/i12345/ocean-simulator/commit/c66a1a8c6b1573688c9a33a378faa0cb58b82449

Project Members: Shaina Ayer, Christian Blundell, Isaac Valdez

Vision:

The ocean simulator will be a video game that lets players virtually "swim" in an ocean with marine life using motion-control.

Target Users:

The target users will be an adolescent-focused group that enjoys things such as new technologies, games, science, animals, exercise, and exploration. A sample of this target audience is Jacob, a family member of group member Isaac Valdez, who falls into this target user category and is used for feedback. Jacob wants a simulator game that runs smoothly, has realistic graphics, and extensive gameplay.

Competitors:

Similar video games and interactive projects featuring marine life already exist, such as Ultimate Ocean Simulator, Ocean Mammals: Blue Whale Mari, Fish Abyss: Aquarium Simulator, and David Attenborough's Great Barrier Reef. They have many kinds of animals, and Ultimate Ocean Simulator lets the player "play" as one of the marine animals. David Attenborough's Great Barrier Reef has many educational videos and articles as well. Besides these, there are many non-interactive underwater experiences available on the Internet, such as regular and 3D videos. However, these games lack motion-controlled input.

Move! lets a person play a video game with motion-control, but it doesn't feature the ocean.

Requirements:

The major requirement for this project is to develop a motion control for the user to navigate through the simulator, with the user's webcam detecting movement and passing it out as movement in the simulation. The other major requirement is to design 3D models of marine life that can be discovered by the user while moving in the simulation. The marine life should have simple AI for generation and recognition of the type of marine life generated. Other requirements include generating AI patterns for certain types of marine life (schools of fish, shark behaviors, rarity of marine life) and dynamic generation of marine life. If possible, other requirements could include developing more in-depth patterns for certain types of marine life based on their natural behavior as well as fluidity of the water. Requirements will not include active interaction with marine life, or any simulation based outside of an ocean/sea environment.

Core Features:

- Navigation between main menu, "set up camera", and playing in the simulation
- 3D model / rendering for animals and diver
- Motion-control for the diver
- AI controls the animals' behavior

Business case:

- The free version lets you swim in only a few places.
- If the user wants to unlock all features (like visiting the Great Barrier Reef or dolphins around Hawaii) they must pay for the full version.
- We gather anonymous data on how people respond to ocean life for research purposes (not for ad-targeting methods), and we could sell this data to researchers.
- We can sell the game to universities or schools to use in science class

Core Features I/O Table:

Input	Output
User: Enters Home Screen	**Program: Begins calibration for motion control**
Webcam: User moves using motion control	**Program: Defines movement to mirror movement of simulator**
User: Selects marine life found in simulator	**Simulator: Identifies fish based on value defined based on its 3D model**
Program: Generates marine life	**User: Sees marine life in simulated environment**

Domain Model:

The domain model shows the scope of each element of the Ocean Simulator. These elements include the motion control figure that guides through the simulator, the player pose that defines the motion control of the game, and the creatures that are generated using simple AI models into the simulator.

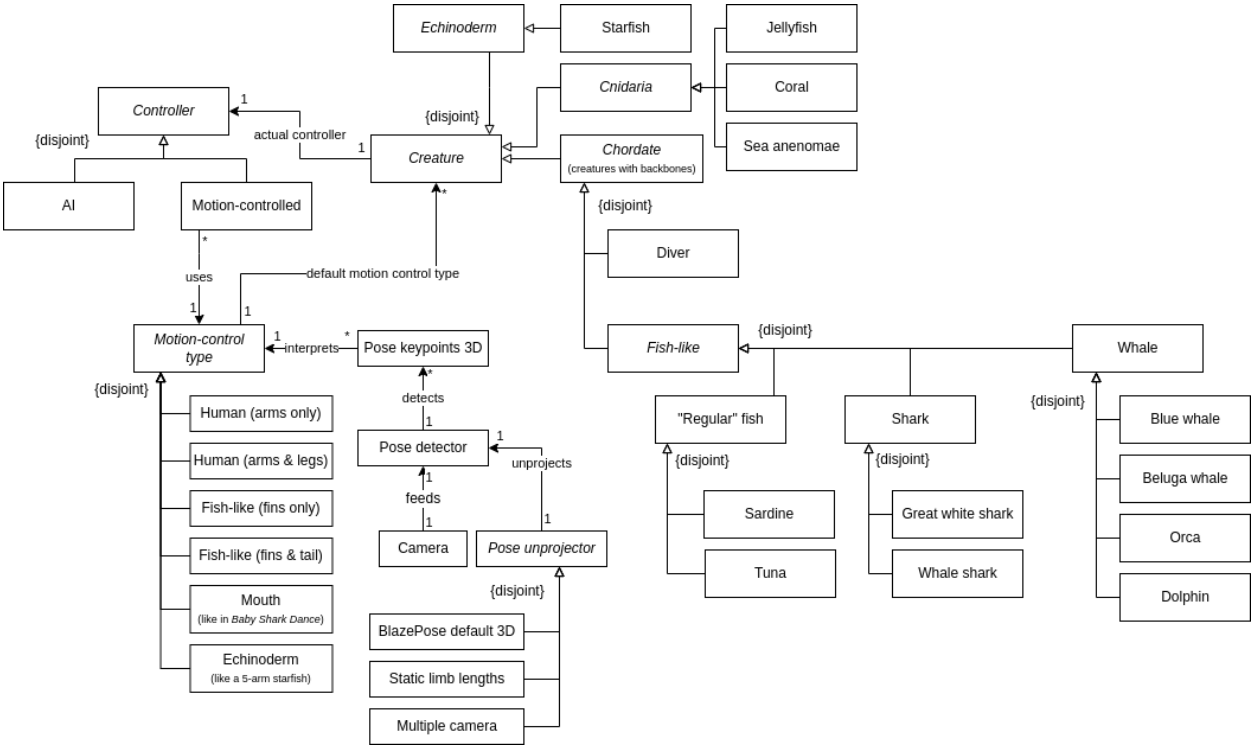

Figure 1: Domain Model for Ocean Simulator

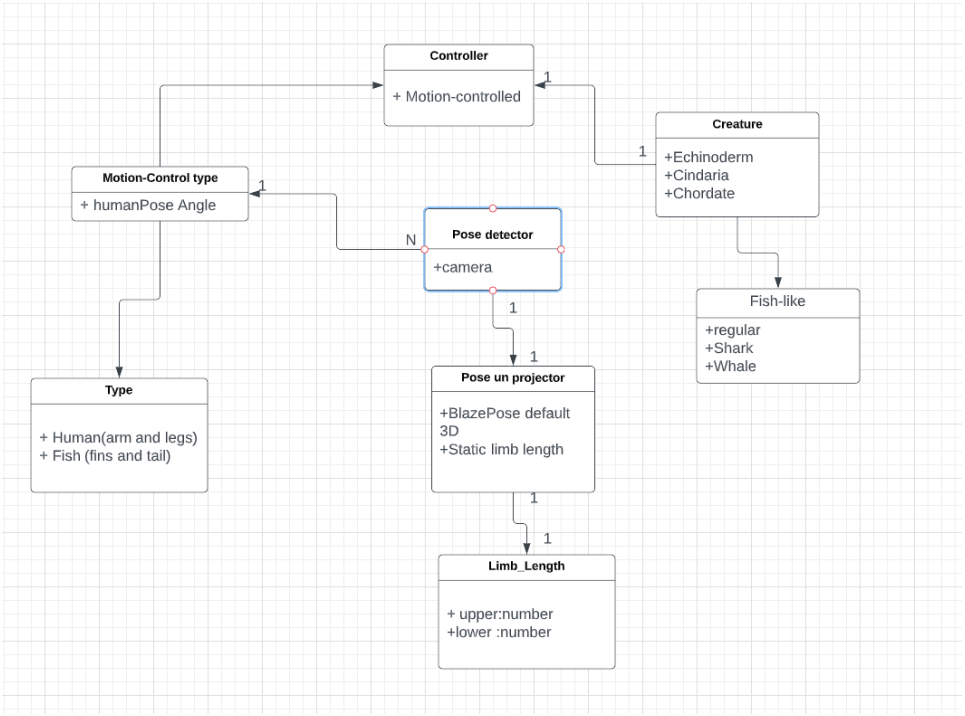

Figure 2: UML Diagram Based on Domain Model of Ocean Simulator Iteration Progress

Use Cases:

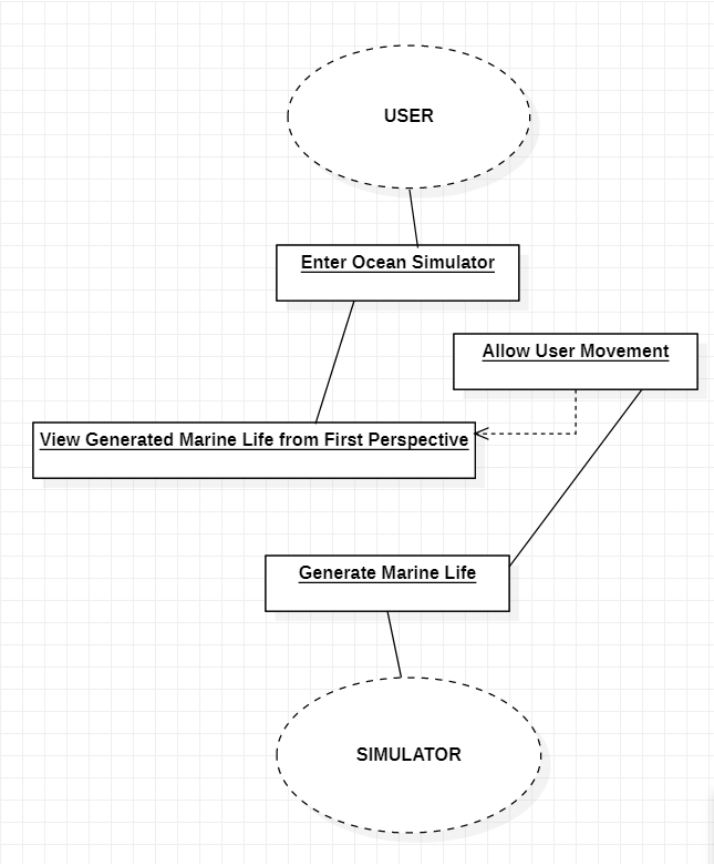

Figure 1: Use Case for General Ocean Simulator Project

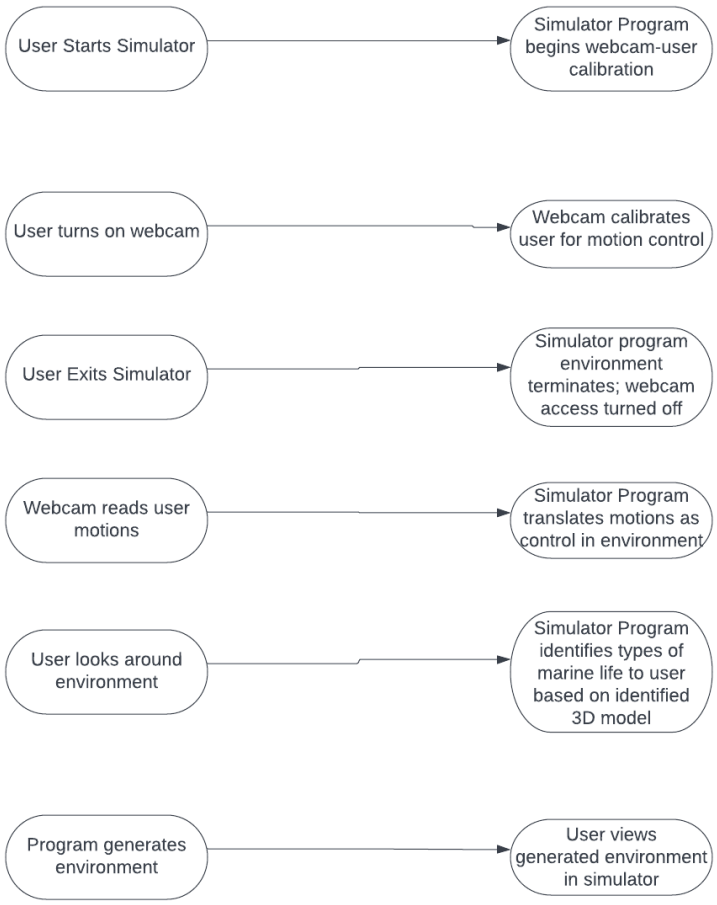

Figure 2: Use Cases for each aspect of simulator setup/gameplay

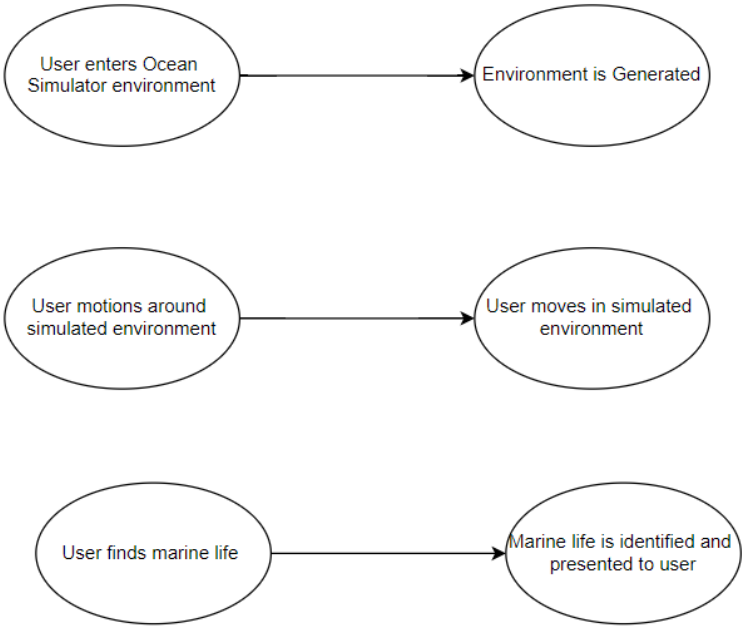

Figure 3: Use Cases for Ocean Simulator Gameplay

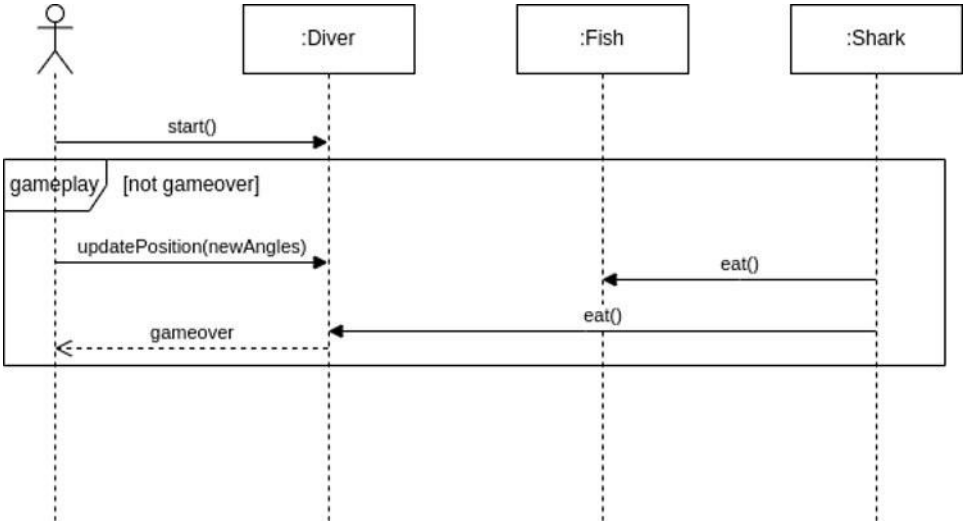

Figure 4: System Sequence Diagram for Ocean Simulator Gameplay

Risks and Risk Mitigation:

- **TV/Large monitor implementation: 100% x total project hours (80 hr) = 80 hours**
  TV/Monitor implementation difficult to achieve, only able to test and develop on single monitor setups.
- Market Study: 20 people that fit target audience demographic took market study; results confirmed that they do not have a setup that justifies the implementation of large monitor device for motion control

    - Mitigation: We will implement and test this project using a single webcam setup on a computer as it is the most available for our target audience. We will potentially develop a simple prototype for multi-device setups that work on all types of large monitors/TV that support screen casting. (We will not make large monitors a must-have requirement for our project's final delivery)
- **Unrealistic Schedule: 100% x total project hours (80 hr) = 80 hours**
    - The time frame for each iteration may not be enough to complete each feature to the highest quality anticipated. This will result in a potentially incomplete product for the final delivery.
    - Mitigation: Schedule each feature for each iteration properly and set realistic goals for each feature. In the project plan, we have anticipated that we will complete the most necessary features by iteration 3 so that the product will have the minimum functionalities we anticipate demonstrating.
- **Failure in Pose Detection Functionality: 100% x total project hours (80 hr) = 80 hours**
    - If pose detection technology is too technical and abstract to implement, core features of the product will not be available, leaving an unpolished product as the resulting effect.
    - Mitigation: Analyze pose detection and implementation in earlier iterations to ensure that pose detection remains a key part of the product. (Currently working on pose detection for this iteration)

Current Progress:

Highlights:

- Physical joints are used – environment could interact with creature just like creature could interact with environment
- Creatures are generic – easy to add new animals
- Pose controlling is generic – easy to extend in future

We have a foundation for the motion control diver, as we have set up the code and test cases for the math to decompose a set of key points (e.g., shoulder, elbow, etc.) into the corresponding rotations of limbs. The code uses MediaPipe BlazePose to detect poses.

However, this code is currently not running in the demo. This iteration showcases progress with the joints. Now, the diver uses hinge joints, which allow two connected entities to connect with one axis of freedom for rotation (like the elbow). Having joints would let the diver be controlled by the motion-control system using the camera, by an AI system, and/or by reaction forces with the environment.

However, in addition to hinge joints, the diver will need to use "cone-twist" joints. These would let a joint have three degrees of freedom for rotation, and they are needed for the shoulder, hip, and neck joints. The cone-twist joints are working in observational tests but have not yet been integrated into the diver.

The system for controlling the pose of a creature is now more abstract than from the previous iteration. `PoseController` is one key interface. A PoseController can update the pose of a `Rig` object, and generic code will take the current pose of a Rig object to rotate the joints for the `Creature` it is

controlling, regardless of the specific type of poses. This lets `PoseController`'s only have to think about the pose they would like the Rig to hold, not the underlying mechanism of moving the joints.

There is a `PoseController` for motion-control. This, in turn, takes a `MotionControlType` to project the camera's observed human pose into a `CreatureKind`-specific pose structure, so that, for example, humans, starfish, and whales could each have a different `MotionControlType` to project the human poses from the camera to control their individual creatures' specialized joints.

Besides the motion-control `PoseController`, there is also a `PoseController` for generating sine-wave motion. This might be used along with a motion-control `PoseController` so that, when the camera cannot see part of the player, it could interpolate rhythmic sine-wave motion to make up for the lack of direct player control. Eventually, there will need to be an intelligent deep learning `PoseController` to make smart AI fish, whales, sharks, other animals, and even AI human divers. This will need further research before implementation.

Also, now there is a fluid simulator. It is not very realistic, but it seemed to exert an opposite force on rigid bodies that move through it. It is slow sometimes, so it may be replaced by an entirely different algorithm during a future iteration.

A static environment is also generated using the PlayCanvas engine, which has an underwater setting which is where the user begins to move and interact.

The test can be run by opening https://localhost:8080/ after running the commands "`npm i && npm run dev`" in the project directory. Currently, the observational test shows the diver with joints working, although they are currently only hinge joints.

3D models have been found and may be implemented in future iteration(s). Using 3D objects extracted from zip files we can use these models and identify them before implementing them into the simulator. Each 3D model will have a form of identification that can be used to identify the type of marine species for the user. Various coral and marine plant 3D models will be accessed from the collection of coral 3D models developed by the Smithsonian Institution (Public domain access).

Different 3D environments have been found for the implementation of the simulator, which are found on open-source sites such as CadNav.

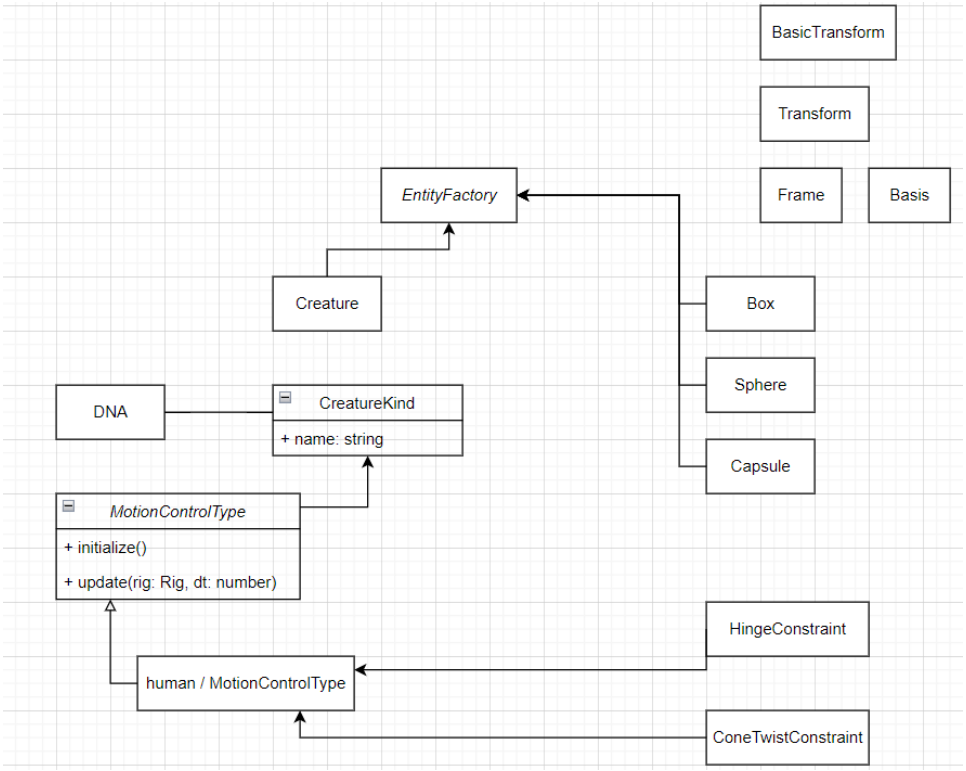

Figure 1: Class Diagram Current Progress Features

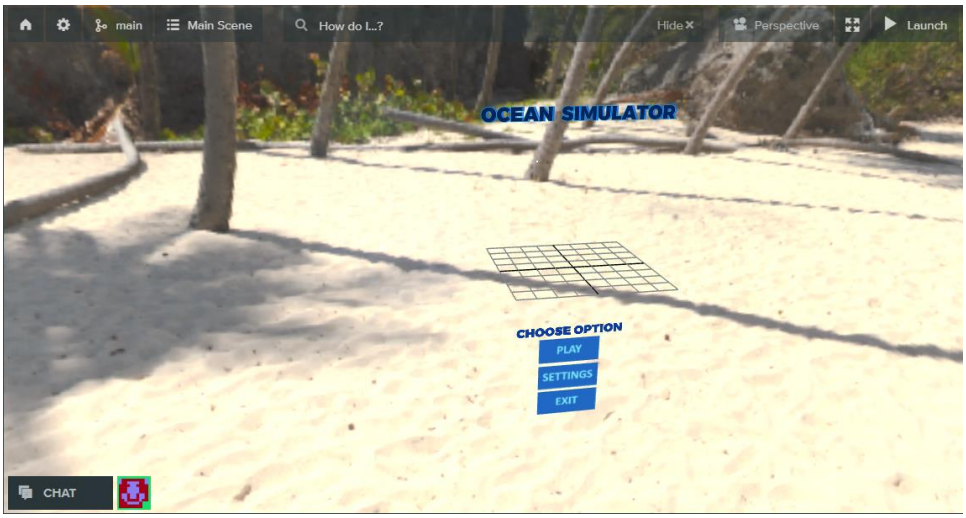

Figure 2: Play Canvas Development of Start Screen with Cube Map Generation

Current Use Case for Webcam Calibration:

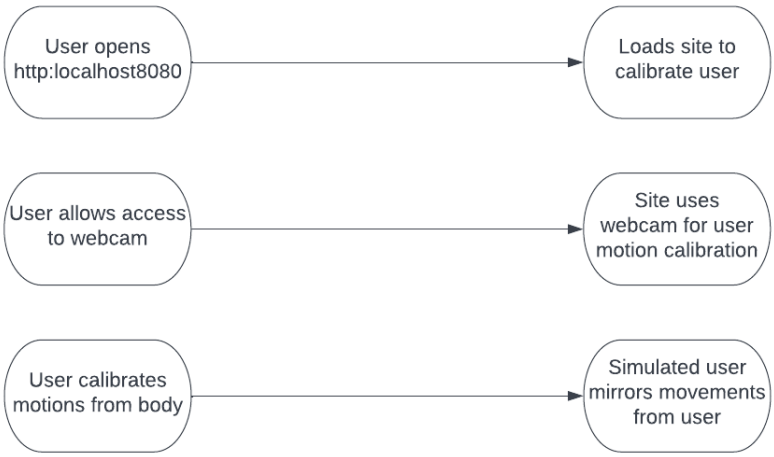

Figure 1: Use Cases for Webcam Calibration

Joel test:

1. Source control: yes
2. Build in one step: yes
3. Daily builds: no
4. Bug database: no
5. Fix bugs before writing code: sometimes
6. Up-to-date schedule: no
7. Have a spec? No
8. Quiet working conditions: yes
9. Best tools money can buy: no
10. Dedicated testers: no
11. New candidates write code: N/A
12. Hallway usability testing: no

Test Cases:

There are currently two kinds of tests: automated tests for math functions and observational tests for entity functions. The math tests just run functions and test whether the answers are close enough to expected results; the observational tests are run by commenting/uncommenting code that calls "playground" functions that make entities, and the tester observes whether the resulting 3D rendered scene and physics seem to match what the playground functions are supposed to do.

There are 42 automated math tests (39 currently passing). They test the functions dealing with Euler angle rotations (rotating a vector by Euler angles and decomposing one or two rotated vectors into their transforming Euler angles), functions dealing with (geometric) transformations (specifically, transforming a translation-rotation-scale transformation with another translation-rotation-scale transformation), and functions that help to decompose human pose key points into corresponding human pose angles. These tests are run with a testing framework like JUnit where inputs and expected

outputs are given for each test-runner function; also, these tests use a custom function to compare whether the given output is close enough to the expected output.

There are 40 observational tests in the “playground”, and they are tested by uncommenting lines from the game entry point function and re-running the project. (For this iteration, the “ground” and “diver” playground functions are shown.) The observational tests currently deal primarily with hinges and cone-twist joints. They also test code for capsule, sphere, and box entity generators, moving rigid bodies, the diver, components of him, and other functions. These tests are helpful for quickly observing the consequences of implementing a function one way or another when the function is difficult to understand, and they also make unintended consequences more quickly visible.

Instructions to Run Project:

1. Install nodejs.
2. Clone ocean simulator repository.
3. Run "npm install" in there.
4. Run "npm run dev" in there. Wait for it to give a URI like “https://localhost:8080”
5. Navigate to https://localhost:8080 or the URI that it gave and wait for the page to load.
6. Observe the diver falling; notice how the hinge joints are simulated.

Feedback:

A representative of the target audience reviewed the current iteration of the project. The information given to the representative was different from the information given to Jacob, another target demographic member. The basis of the feedback was not to ask about the experience of the simulation due to the missing features, but rather to ask about the foundation and quality of the motion control aspect and the fluid simulation in the test environment. The representative enjoyed the fluidity dynamic and suggested it shares physical attributes to moving underwater. The user suggests the motion control calibration should begin after the environment is generated (not shown in current demo).

Future Project Plans:

Based on the incompletion of certain aspects of the Ocean Simulator, the future development plans will include the features mentioned in previous iterations but not completed. This includes the creation of various dynamically generated ocean environments and various marine life 3D models to be generated. A running game navigation screen with seamless transition to gameplay will also need to be implemented, as well as the option to play as a marine animal. Features mentioned in the business model can also be implemented for potential commercial release, such as multiplayer and dynamic generation AI.