Demo

Ocean Simulator

Iteration 1

Group 4

Target User

Defined Target User: Adolescent Demographic

Interested in Sciences and Animals, New Technology, and Exploration

Target Audience assumed to be able to play this simulator using an accessible device, and do not need any external installation of hardware (already has a webcam)

Plan

- Assumed 60 hours of Development for total remaining project timeframe
- Develop Motion Control for Diver (Iteration2 - Iteration 3)
- Create 3D models for the marine life to be generated into simulation environment (Iteration 2 - Iteration 3)
- Start screen and simulator navigation (Iteration 3 - Iteration 4)
- After main features: Various marine life generation, simple AI patterns, polished motion control recognition, dynamically generated marine life, multi-device setup (Iteration 5)

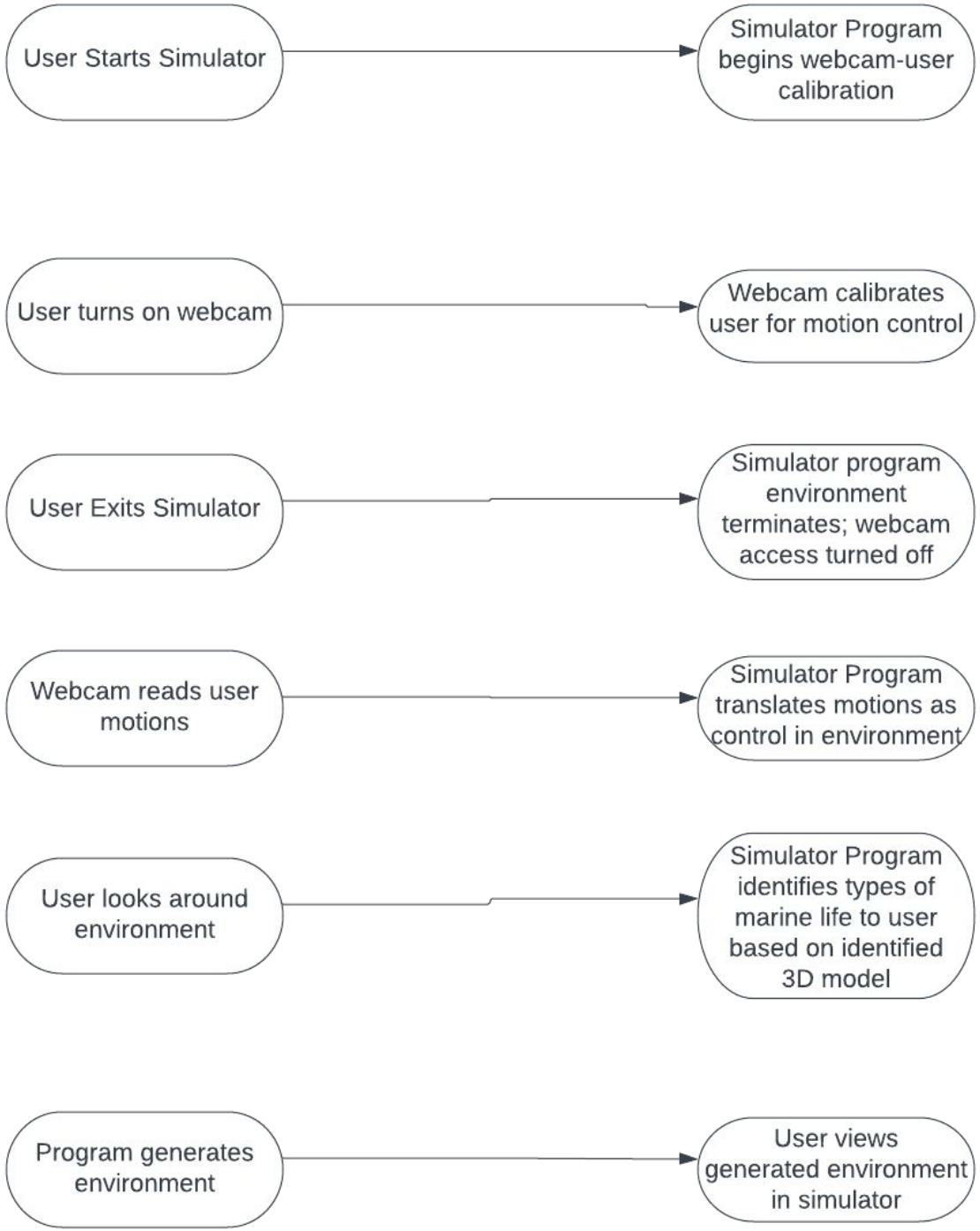
Risks

- **TV/Large Monitor Multi-Device Implementation: 100% x 60 hr = 60 hr**
  - Effect: Difficult Implementation for multi-device setups
  - Mitigation: Begin computer webcam implementation, then begin simple prototype; classify project as single device so this is not classified as a major risk – Justified with market study that shows most target users only have access to single device setup

- **Unrealistic Schedule: 60% x 60 hr = 36 hr**
  - Effect: Incomplete or lower quality product
  - Mitigation: Create incremental delivery schedule for main features, focus on supplemental features in later iterations; ensure there is core feature functionality

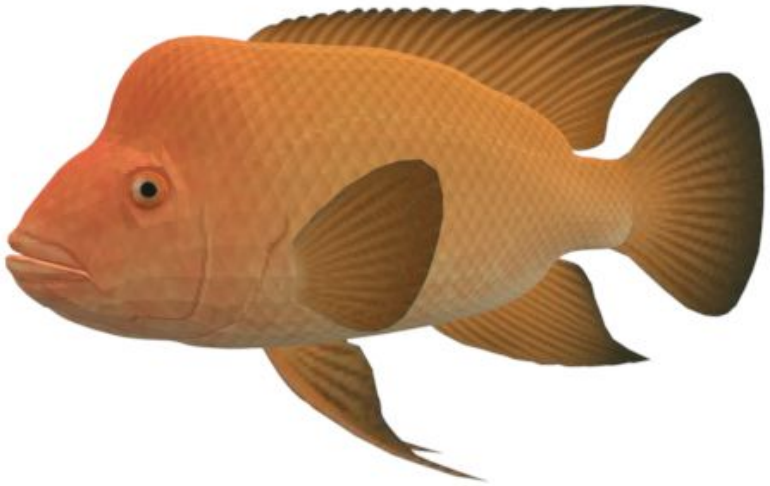
- **Failure of Pose Detection Functionality: 40% x 60 hr = 24 hr**
  - Effect: Main feature not properly implemented, product is different to its initial description
  - Mitigation: Focus on this functionality first to ensure the core feature is included in the project

Use cases

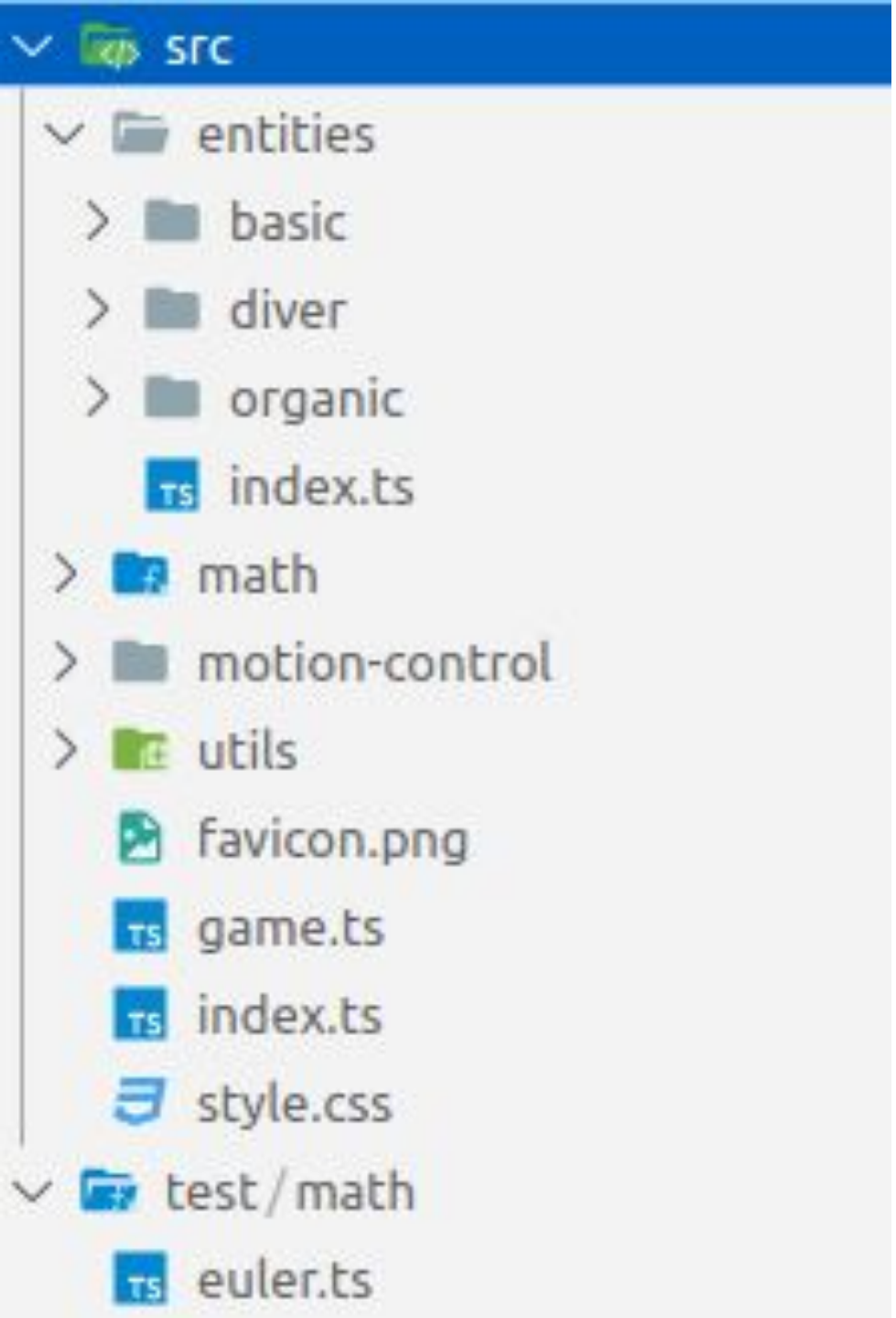

User Starts Simulator
Simulator Program begins webcam-user calibration
User turns on webcam
Webcam calibrates user for motion control
User Exits Simulator
Simulator program environment terminates; webcam access turned off
Webcam reads user motions
Simulator Program translates motions as control in environment
User looks around environment
Simulator Program identifies types of marine life to user based on identified 3D model
Program generates environment
User views generated environment in simulator

Current Progress

- Implementation of motion control calibration/recognition (Demo)
- 3D modeling sources identified for generation



3D model that will be used in the Ocean Simulator



Code Organization

- **Organized by Topic**
  - **"entities" may get an "animals" folder**
- **Key Data Structures:**
  - **Keypoints Map**
  - **Pose Angle**
- **Key code:**
  - **Euler Angle Decomposition**
  - **Pose Angle Decomposition**

Running the Code

- **May want to use multiple devices for a single game so multiple cameras can give higher-resolution pose detection.**

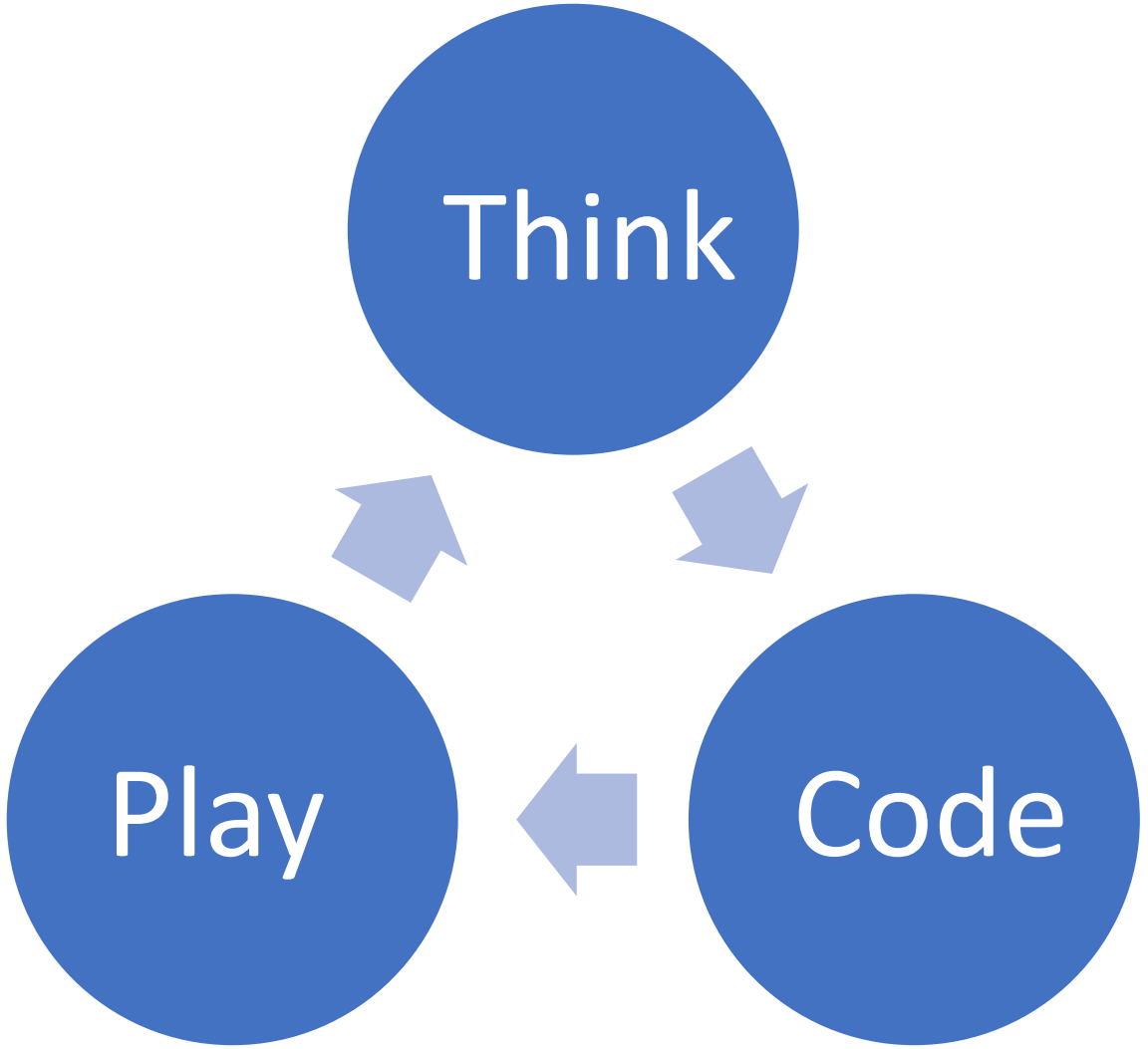
- **In that case, a certificate for https will be needed.**

1. Install nodejs
2. Clone ocean simulator repository
3. Run "npm install" in there
4. Run "npm run dev" in there
5. Navigate to http://localhost:8080 from development computer and wait for it to load
6. Let web page access camera
7. Back up so the camera can see your arms
8. Swim!

Pose Angle Decomposition

Testing

- Colored cubes visualized positions of key points
- Blue rectangle visualized rotation of component currently being debugged
- console.log()
- Debug visualizations were removed for the demo version



Test Cases

13 Test Cases for motion control calibration / webcam detection

9 Test cases for decompose of identity rotation

4 Test cases for Euler angle rotation

Customer Feedback

- **Isaac showed the demo to his brother Jacob.**
- **Jacob represents the target audience as he is a young male who is interested in new technology and exploring ocean life**
- **After interacting with the demo, Jacob gave these comments:**

Positive

- Responsive to joints, especially elbows and shoulders.

Negative

- Needs to be responsive to all other joints, like wrists, legs, and possibly hands as well.

Questions?

https://github.com/i12345/ocean-simulator/tree/f0d4c7b4b86193571ed0fc8349a64b3a0f2f9c85

Keypoints Map (1/3)

keypoints.ts

src > motion-control > keypoints.ts > ...

```
7   export type KeypointID = number
8
    You, 2 days ago | 1 author (You) | 5 references |
9   export interface KeypointIDs {
        0 references
10      nose: KeypointID
        0 references
11      left_eye: KeypointID
        0 references
12      right_eye: KeypointID
        0 references
13      left_ear: KeypointID
        0 references
14      right_ear: KeypointID
        7 references
15      left_shoulder: KeypointID
        5 references
16      right_shoulder: KeypointID
        3 references
17      left_elbow: KeypointID
        3 references
18      right_elbow: KeypointID
        3 references
19      left_wrist: KeypointID
```

Keypoints Map (2/3)

keypoints.ts

src > motion-control > keypoints.ts > ...

```ts
export type KeypointName = keyof KeypointIDs

2 references
export const modelKeypoints = {
    COCO: {
        nose: 0,
        left_eye: 1,
        right_eye: 2,
        left_ear: 3,
        right_ear: 4,
        left_shoulder: 5,
        right_shoulder: 6,
        left_elbow: 7,
        right_elbow: 8,
        left_wrist: 9,
        right_wrist: 10,
        left_hip: 11,
        right_hip: 12,
        left_knee: 13,
        right_knee: 14,
        left_ankle: 15,
        right_ankle: 16
    },

    BlazePose: {
        nose: 0,
        left_eye_inner: 1,
        left_eye: 2,
        left_eye_outer: 3,
        right_eye_inner: 4,
        right_eye: 5,
```

Keypoints Map (3/3)

```
export type COCOKeypointIDs = typeof modelKeypoints.COCO
2 references
export type BlazePoseKeypointIDs = typeof modelKeypoints.BlazePose

11 references
export type KeypointMap<T, IDs extends KeypointIDs = KeypointIDs> = { [K in keyof IDs]: T }
3 references
export type KeypointPoseMap<IDs extends KeypointIDs = KeypointIDs> = KeypointMap<Keypoint, IDs>
```

Pose Angle

pose-angles.ts

src > motion-control > pose-angles.ts > ...

```ts
export type PoseAngle = EulerAngle

You, 2 days ago | 1 author (You) | 4 references | 1 implementation
export interface LimbAngles {
    14 references
    upper: PoseAngle
    8 references
    lower: PoseAngle
    6 references
    end: PoseAngle
}

You, 2 days ago | 1 author (You) | 4 references | 1 implementation
export interface MirroredAngles<T> {
    12 references
    left: T
    11 references
    right: T
}

You, 2 days ago | 1 author (You) | 4 references | 2 implementations
export interface PoseAngles {
    1 reference
    head: PoseAngle
    25 references
    arms: MirroredAngles<LimbAngles>
    1 reference
    legs: MirroredAngles<LimbAngles>
}
```

Euler Angle Decomposition (1/2)

euler.ts

src > math > euler.ts > ...

```
20  /**
21   * Computes the euler XY angles needed to rotate [0, 0, 1] to get to {@link v}
22   * @param v a vector to find the euler XY angles to get to from [0, 0, 1]
23   */
    9 references
24  export function decomposeEulerXY(v: Vec3): EulerAngle {
25      // The returned x-component will rotate around the x-axis, and
26      // then the resulting vector will be rotated around the y-axis.
27
28      v = new Vec3().copy(v).normalize()
29
30      const y_plane = zx(v)
31
32      const angle_x = Math.atan2(-v.y, y_plane.length()) * 180 / Math.PI
33      const angle_y = Math.atan2(y_plane.y, y_plane.x) * 180 / Math.PI
34
35      return new Vec3(angle_x, angle_y, 0)
36  }
```

Euler Angle Decomposition (2/2)

euler.ts M

test > math > euler.ts > EulerTests

```
27    @test "decompose identity rotation"({ vector, eulerAngle }:
28        { vector?: Vec3, eulerAngle: EulerAngle }) {
29        vector ??= Vec3.BACK
30        const rotated_byParameter = rotate(vector, eulerAngle)
31        const decomposed: EulerAngle = decomposeEulerXY(rotated_byParameter)
32
33        const rotated_byDecomposition = rotate(vector, decomposed)
34
35        this.assertEq(rotated_byParameter, rotated_byDecomposition)
36    }
```

Pose Angle Decomposition (1/2)

pose-angles.ts

src > motion-control > pose-angles.ts > ...

```ts
export function calcPoseAngles(pose: Partial<KeypointMap<Vec3>>): DeeplyPartial<PoseAngles> {
    let angles: DeeplyPartial<PoseAngles> = {}
    const blazePose = pose as Partial<KeypointMap<Vec3, BlazePoseKeypointIDs>>

    // L arm
    if (pose.left_shoulder && pose.right_shoulder && pose.left_hip) {
        const y = new Vec3().sub2(pose.left_shoulder, pose.left_hip)
        const z = new Vec3().sub2(pose.left_shoulder, pose.right_shoulder)
        const x = new Vec3().cross(y, z)
        const basis_upper = { x, y, z }

        angles.arms ??= {}
        angles.arms.left ??= {}

        if (pose.left_elbow) {
            angles.arms!.left!.upper = decomposeEulerXY(
                projectToBasis(
                    new Vec3().sub2(pose.left_elbow, pose.left_shoulder),
                    basis_upper
                )
            )
```

Pose Angle Decomposition (2/2)

pose-angles.ts

src > motion-control > pose-angles.ts > ...

```
46
47          if (pose.left_wrist) {
48              const basis_lower = rotateBasis(basis_upper, angles.arms!.left!.upper! as EulerAngle)
49
50              angles.arms!.left!.lower = decomposeEulerXY(
51                  projectToBasis(
52                      new Vec3().sub2(pose.left_wrist, pose.left_elbow),
53                      basis_lower
54                  )
55              )
56
57              if (blazePose.leftThumb) {
58                  const basis_hand = rotateBasis(basis_lower, angles.arms!.left!.lower! as EulerAngle)
59
60                  angles.arms!.left!.end = decomposeEulerXY(
61                      projectToBasis(
62                          new Vec3().sub2(blazePose.leftThumb, pose.left_wrist),
63                          basis_hand
64                      )
65                  )
66              }
67          }
68      }
69  }
```

(Similar for the right arm)