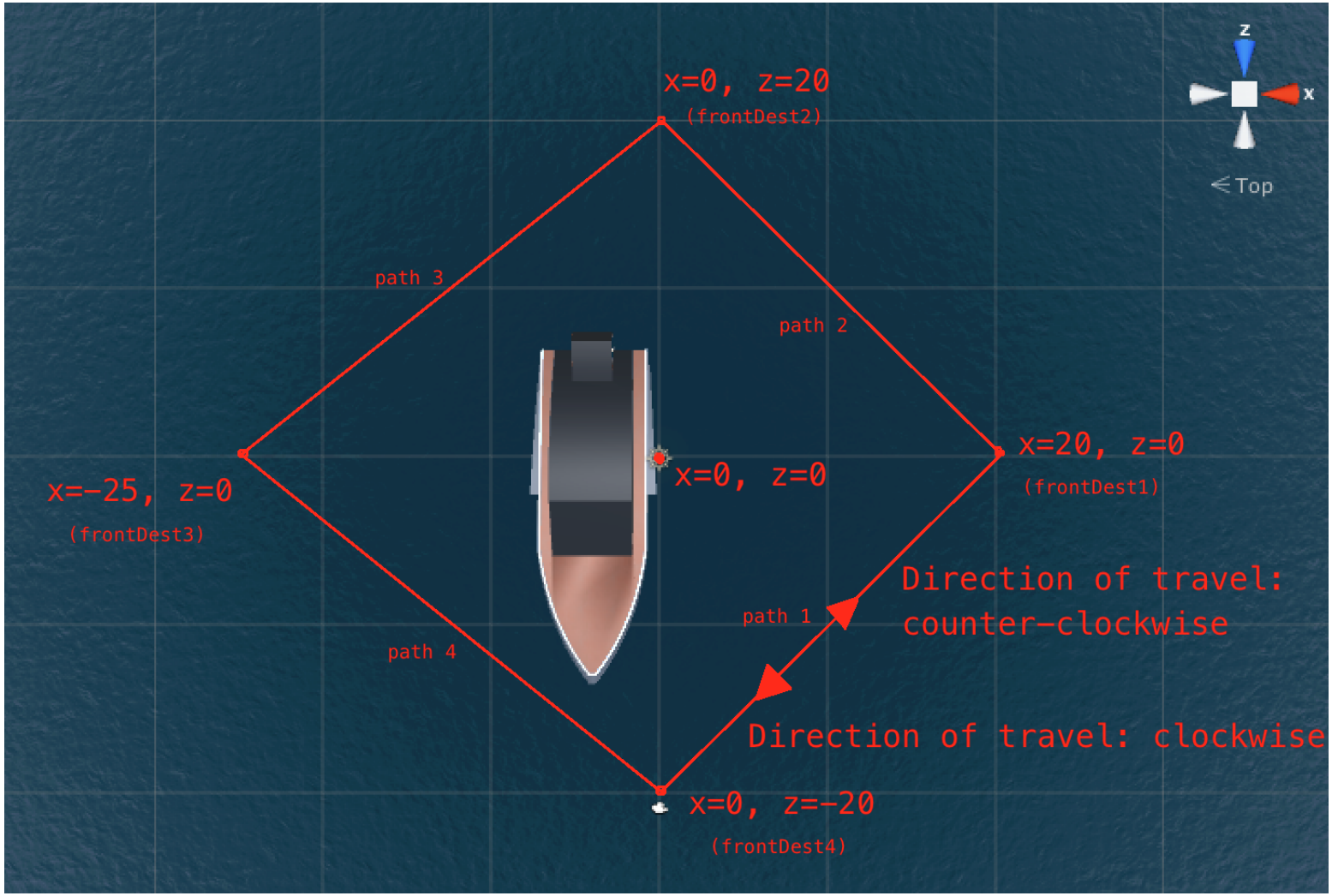
Code Review

There are only two C# files custom developed for this application: NewBehaviourScript and EventHandlerScript. Complete listing for them is available as part of the download package. Below, we give the review of those scripts.

Review of NewBehaviourScript

The NewBehaviourScript is attached to CardboardMain object, which contains the stereo camera in its object hierarchy. NewBehaviourScript is responsible for automatically navigating CardboardMain, hence the stereo camera, around the yacht object. It is also responsible for handling the pointer click event by changing direction of travel.

Automatic navigation is implemented using Unity navigation system. The navigation mesh, i.e. the navigation surface area, is created for the WaterProDaytime object, that is positioned at X=0, Y=0, Z=0, the origin. The CardboardMain object is initially at X=0, Y=2, Z=-20. The navigation of CardboardMain object will take place on (x,z) plane always maintaining Y=2. Geometric details of the navigation is shown below.



The code starts with various definitions.

```
public class NewBehaviourScript : MonoBehaviour {

        // Reference to CardboardMain object
        GameObject cardboardMain;

        // Reference to Nav Mesh Agent component in CardboardMain
        NavMeshAgent navmeshAgent;

        // Reference to yacht object
        GameObject yacht;
```

```
// Navigation agent is likely to stop when it gets very close to its destination,
// rather than exactly at the destination. We need to check if agent's current
// position is within a close margin of the destination. If so, we can assume it is
// at the destination. The margin is set below.
float margin = 1.5f;
```

We now define the destination points for the navigation. See the above figure for definitions.

```
// Approximately 3 o'clock position in (x,z)-plane
Vector3 frontDest1 = new Vector3 (20,2,0);

// Approximately 12 o'clock position in (x,z)-plane
Vector3 frontDest2 = new Vector3 (0,2,20);

// Approximately 9 o'clock position in (x,z)-plane
Vector3 frontDest3 = new Vector3 (-25,2,0);

// Navigation starting point; approximately 6 o'clock position in (x,z)-plane
Vector3 frontDest4 = new Vector3 (0,2,-20);

// This flag indicates rotation of travel.
// If true, travel is in counter-clockwise direction; otherwise,
// it is clockwise
bool ccw = true;

// Camera travels along 4 distinct paths:
// path 1: between frontDest4 (starting point) and frontDest1
// path 2: between frontDest1 and frontDest2
// path 3: between frontDest2 and frontDest3
// path 4: between frontDest3 and frontDest4
int path = 1; // Initial path
```

The Start method will be called at application start. Here, we obtain object references and set initial destination point for travel.

```
void Start () {
        // Obtain reference to CardboardMain object
        cardboardMain = GameObject.Find("CardboardMain");

        // Obtain reference to Nav Mesh Agent component
        navmeshAgent = GetComponent();

        // Obtain reference to yacht object
        yacht = GameObject.Find("yacht");
```

```
        // Set initial destination point for travel
        navmeshAgent.destination = frontDest1;
}
```

The Update method is called per frame. Here, depending on direction of travel, clockwise or counter-clockwise, we check if Nav Mesh Agent is within close proximity to its current destination. If so, we assume it reached its destination and set the next destination point and update the path.

```
void Update () {
        // Depending on whether direction of travel is counter clockwise or clockwise
        // check proximity to current destination and if we're within the margin set
        // next destination point.

        if (ccw) {// Current rotation is counter clockwise
                switch (path) {
                case 1: // Currently we're traveling from frontDest4 to frontDest1
                        if ((navmeshAgent.transform.position - frontDest1).magnitude <= margin) {
                                navmeshAgent.destination = frontDest2;
                                path = 2;
                        }
                        break;
                case 2: // Currently we're traveling from frontDest1 to frontDest2
                        if ((navmeshAgent.transform.position - frontDest2).magnitude <= margin) {
                                navmeshAgent.destination = frontDest3;
                                path = 3;
                        }
                        break;

                case 3: // Currently we're traveling from frontDest2 to frontDest3
                        if ((navmeshAgent.transform.position - frontDest3).magnitude <= margin) {
                                navmeshAgent.destination = frontDest4;
                                path = 4;
                        }
                        break;
                case 4: // Currently we're traveling from frontDest3 to frontDest4
                        if ((navmeshAgent.transform.position - frontDest4).magnitude <= margin) {
                                navmeshAgent.destination = frontDest1;
                                path = 1;
                        }
                        break;

                default:
                        break;
                }
```

```
            } else {// Current rotation is clockwise
                    switch (path) {
                    case 1: // Currently we're traveling from frontDest1 to frontDest4
                            if ((navmeshAgent.transform.position - frontDest4).magnitude <= margin) {
                                    navmeshAgent.destination = frontDest3;
                                    path = 4;
                            }
                            break;
                    case 2: // Currently we're traveling from frontDest2 to frontDest1
                            if ((navmeshAgent.transform.position - frontDest1).magnitude <= margin) {
                                    navmeshAgent.destination = frontDest4;
                                    path = 1;
                            }
                            break;

                    case 3: // Currently we're traveling from frontDest3 to frontDest2
                            if ((navmeshAgent.transform.position - frontDest2).magnitude <= margin) {
                                    navmeshAgent.destination = frontDest1;
                                    path = 2;
                            }
                            break;
                    case 4: // Currently we're traveling from frontDest4 to frontDest3
                            if ((navmeshAgent.transform.position - frontDest3).magnitude <= margin) {
                                    navmeshAgent.destination = frontDest2;
                                    path = 3;
                            }
                            break;

                    default:
                            break;
                    }
            }
```

As the CardboardMain object travels along its current path, it should always keep pointing to the yacht object. We complete the Update method by adjusting current direction of CardboardMain towards yacht.

```
            // Maintain current transform of CardboardMain object towards yacht
            cardboardMain.transform.LookAt(yacht.transform);
    }
```

We mentioned previously that when user points the gaze pointer to the yacht and sends a trigger input via the cardboard headset, rotation of travel would change, from clockwise to counter-clockwise and vice versa. That event is captured and dispatched by the yacht. The event handler is the ToggleRotation method. In that method we set the flag, ccw, to indicate new direction of travel. Then, depending on current path, we change the destination point.

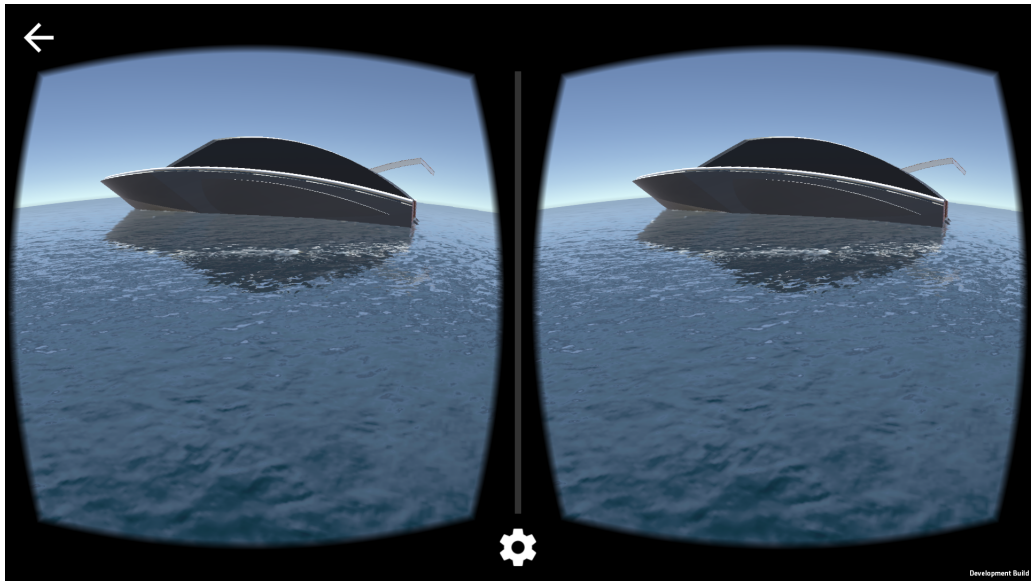
```
public void ToggleRotation(){
        // Change rotation from clockwise to counter-clockwise and vice versa.

        if (ccw) { // Current rotation is counter-clockwise
                ccw = false; // change direction
                switch (path) { // Depending on current path, change destination point
                case 1:
                        navmeshAgent.destination = frontDest4;
                        break;
                case 2:
                        navmeshAgent.destination = frontDest1;
                        break;
                case 3:
                        navmeshAgent.destination = frontDest2;
                        break;
                case 4:
                        navmeshAgent.destination = frontDest3;
                        break;
                default:
                        break;
                }
        } else { // Current rotation is clockwise
                ccw = true; // change direction
                switch (path) { // Depending on current path, change destination point
                case 1:
                        navmeshAgent.destination = frontDest1;
                        break;
                case 2:
                        navmeshAgent.destination = frontDest2;
                        break;
                case 3:
                        navmeshAgent.destination = frontDest3;
                        break;
                case 4:
                        navmeshAgent.destination = frontDest4;
                        break;
                default:
                        break;
                }
        }
}
```

This completes the review of NewBehaviourScript.

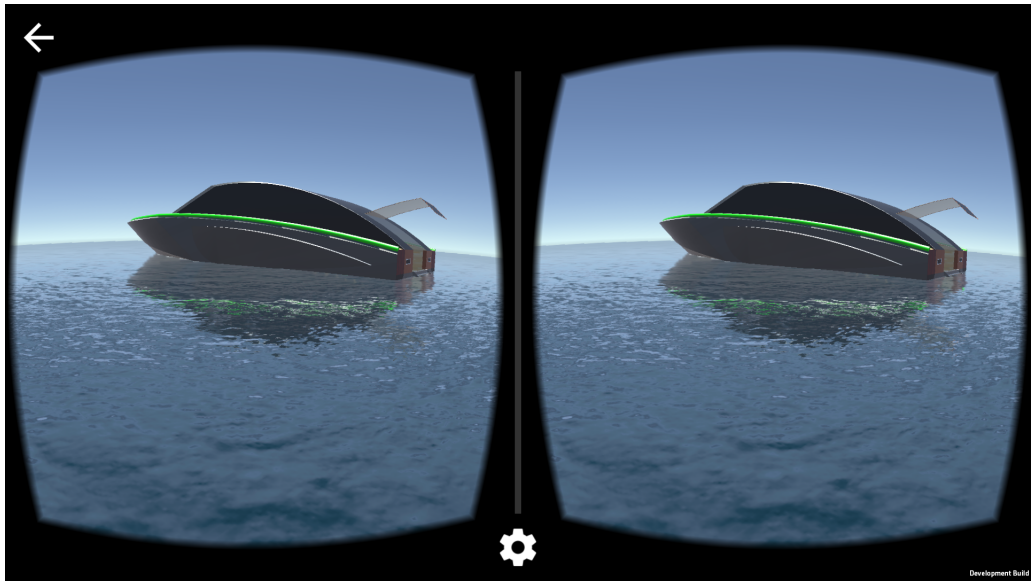
Review of EventHandlerScript

The EventHandlerScript is attached to the yacht object and is responsible for handling pointer enter and pointer exit events. The pointer enter event changes color of the gunwale of the yacht object to green from original silver color. The pointer exit event changes color of the gunwale to original silver from green.

The following figure shows the original gunwale color.



The following figure shows the gunwale color in green.



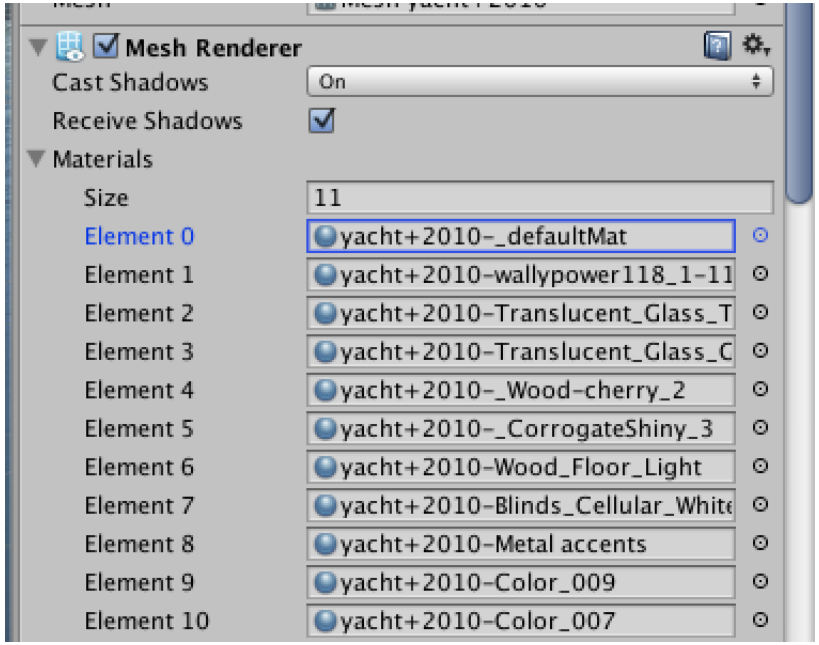
This class can only be attached to an object that is of type collider. For this reason the yacht object consists of a box collider component.

```
[RequireComponent(typeof(Collider))]
public class EventHandlerScript : MonoBehaviour {
```

We declare fields to hold the original color of the gunwale and the renderer for the yacht.

```
        // Original color of the yacht's gunwale
        Color originalColor;

        // Renderer component of the yacht object
        Renderer originalRenderer;
```

Note that the gunwale color is defined by the 0-th element in the materials array of the yacht model's renderer as shown below.



At initialization, we obtain a reference to the renderer and define the original gunwale color. Then, we reset the gunwale color to original.

```
        void Start() {
                // Obtain reference to originalRenderer
                originalRenderer = GetComponent ();

                // Obtain originalColor
                originalColor = originalRenderer.materials [0].color;

                // Reset gunwale's color
                SetGazedAt(false);
```

The SetGazedAt method below is called at pointer enter event with input parameter set to true and also at pointer exit event with input parameter set to false.

```
        public void SetGazedAt(bool gazedAt) {
                // If true is passed set gunwale's color to green; if false is passed set it
                // back to its original color
                originalRenderer.materials [0].color = gazedAt ? Color.green : originalColor;
        }
}
```

This completes the review of EventHandlerScript.