Contents

TECH

DESIGN

The Roman Fisherman

This report will describe the prototyping, evaluation and reflection of a Mixed Reality experience designed to be delivered by the British Museum, to explore the history of Ancient Rome. It is a Virtual Reality fishing game to let the user be put in the shoes of a Roman fisherman, after visiting the related exhibits in the museum. Before the game begins and the visitor puts the headset on (an Oculus Quest 2), they will be given a brief exposition on the game including: how to start the game, controls (pressing the grip button on the Oculus Quest 2 controllers), their role and objective in the game (that they are a Roman fisherman who is to catch as many fish as they can using a trident and fishing rod), and a warning on cybersickness, caused by VR and motion sickness as the game will involve some sensory misalignment. See the video for a walkthrough.

Beforehand, the visitor is shown how to hold the trident (over their shoulders), and the fishing rod (in front of them). They are also suggested to step or lean towards the edge of the boat to feel the boat tilt. They are not, however, told why there is a wire tied to the controller. No instructions are given during the game unless asked.

Prototyping

The game was made using Unity (3D Universal Render Pipeline), the Oculus XR Plugin and XR Interaction Toolkit by Unity Technologies. To start off, a rough terrain was built (Fig. 1.) with a river flowing through it (Fig. 2.), placing a boat on it (Fig. 3-5.).

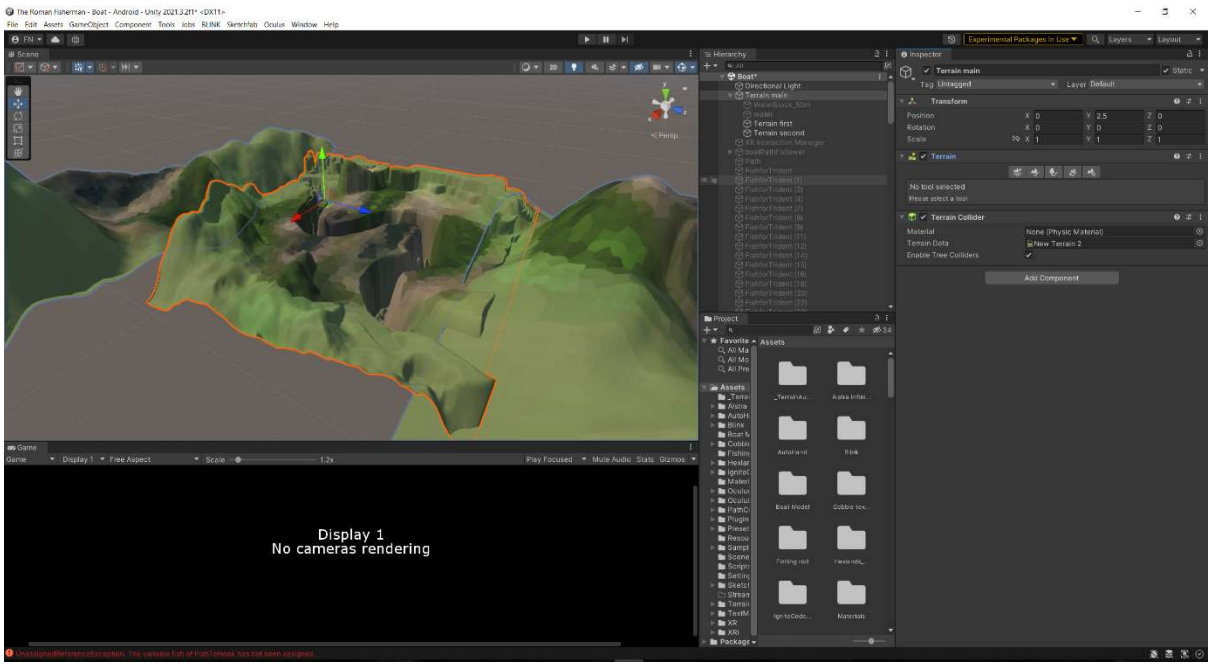


Fig. 1. A rough terrain, enough to cover the horizon from the player's perspective. Low poly texture painted over it found from [2]. Deep ridges in a snake-like shape made for the river to go in.

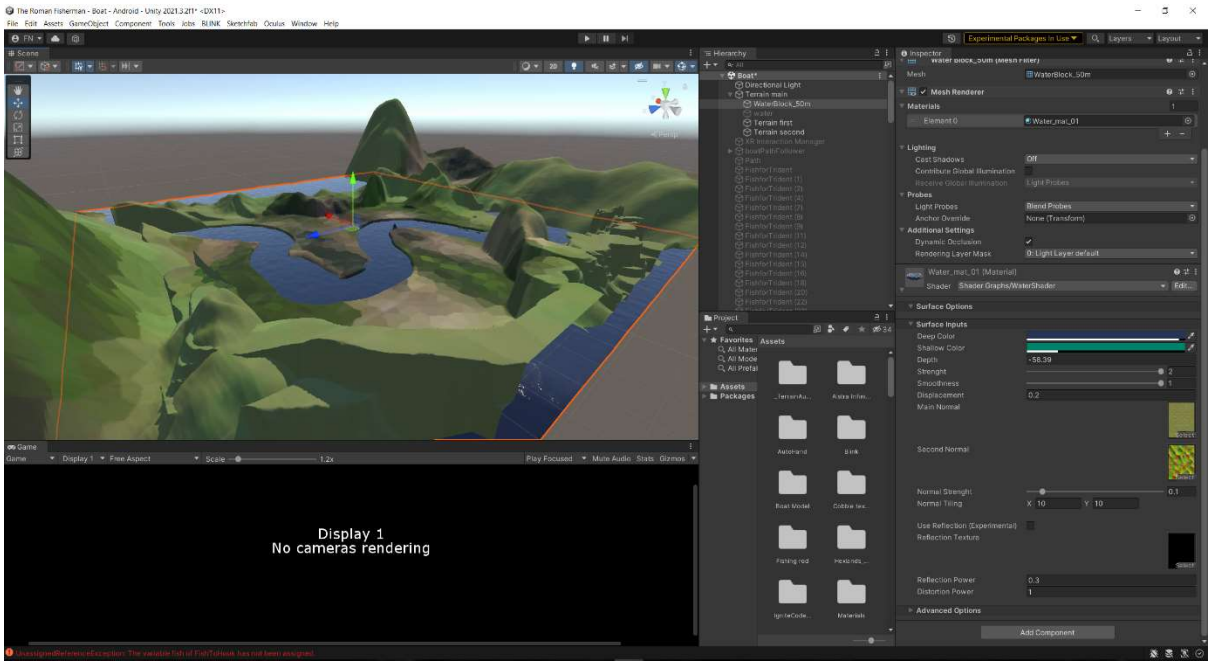


Fig. 2. Water shader found from [1] added to a plane which is then slid into the ridge to form a river, adjusting the displacement to get some vertical movement for the water.

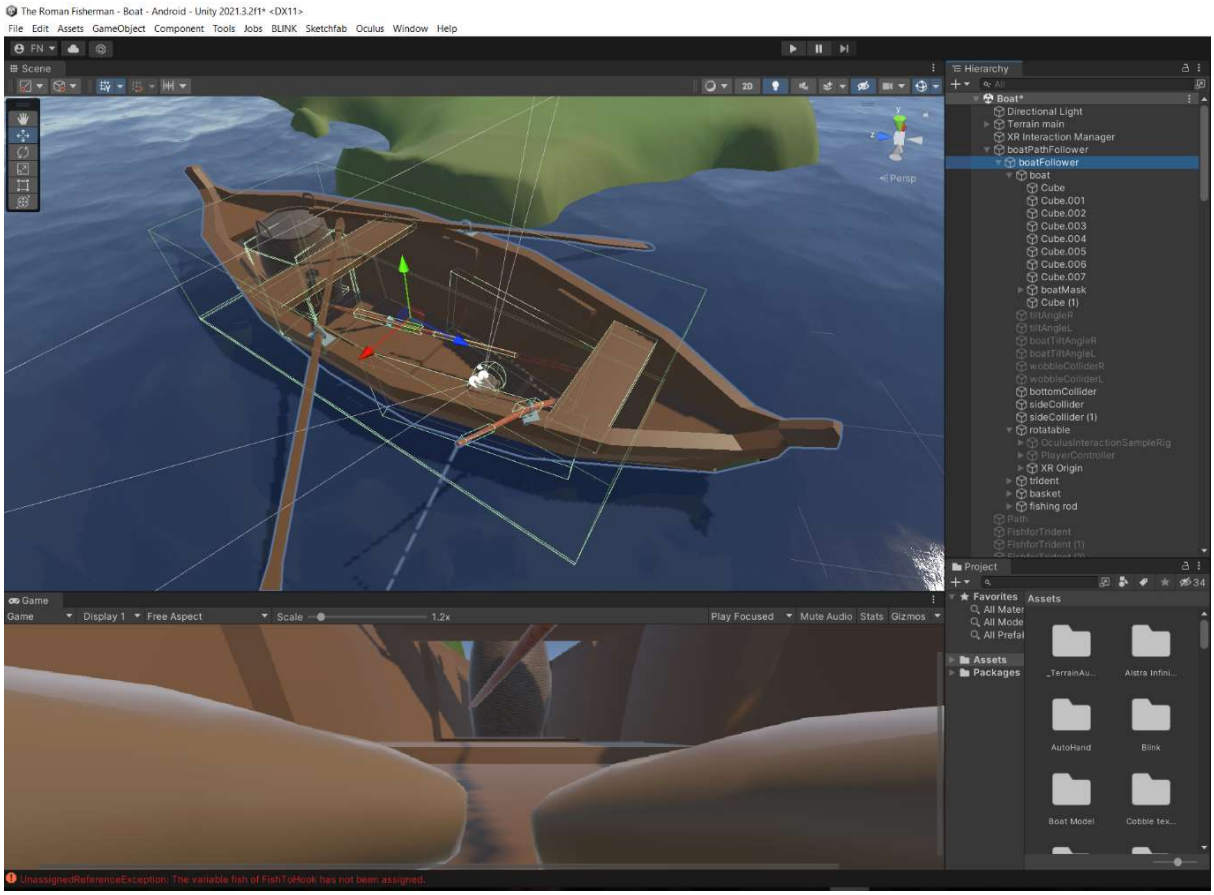


Fig. 3. A 3D boat model found from [3] and modified using Blender (see Fig. 5.) is placed on the water. The XR origin is placed on the boat, and made a parent of the boat so that the player moves with the boat (Fig. 4.).

Fig. 4. The XR origin is place on the floor of the boat, with Tracking Origin Mode set to “floor” so that the floor of the boat matches the physical floor, improving presence.

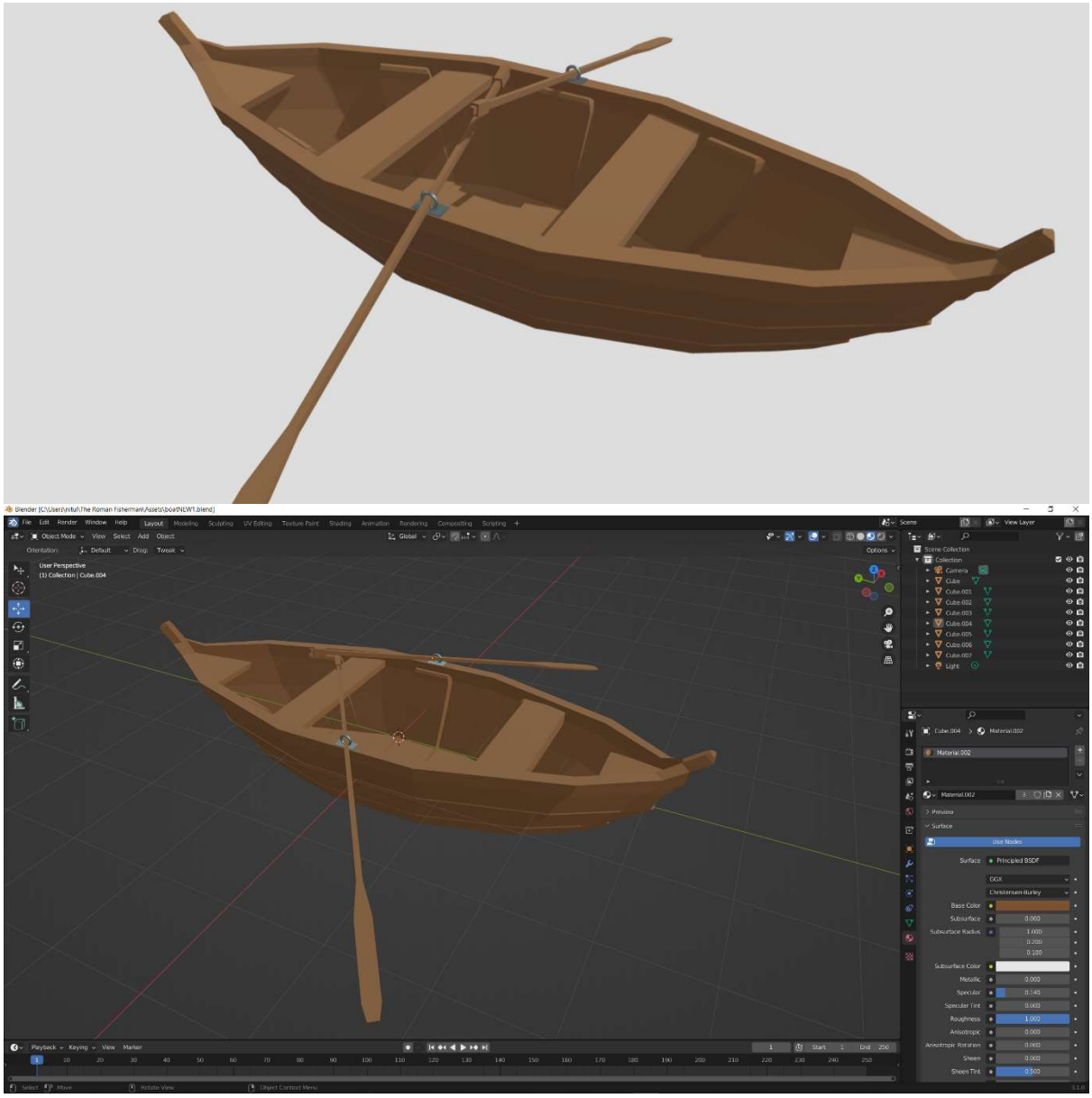
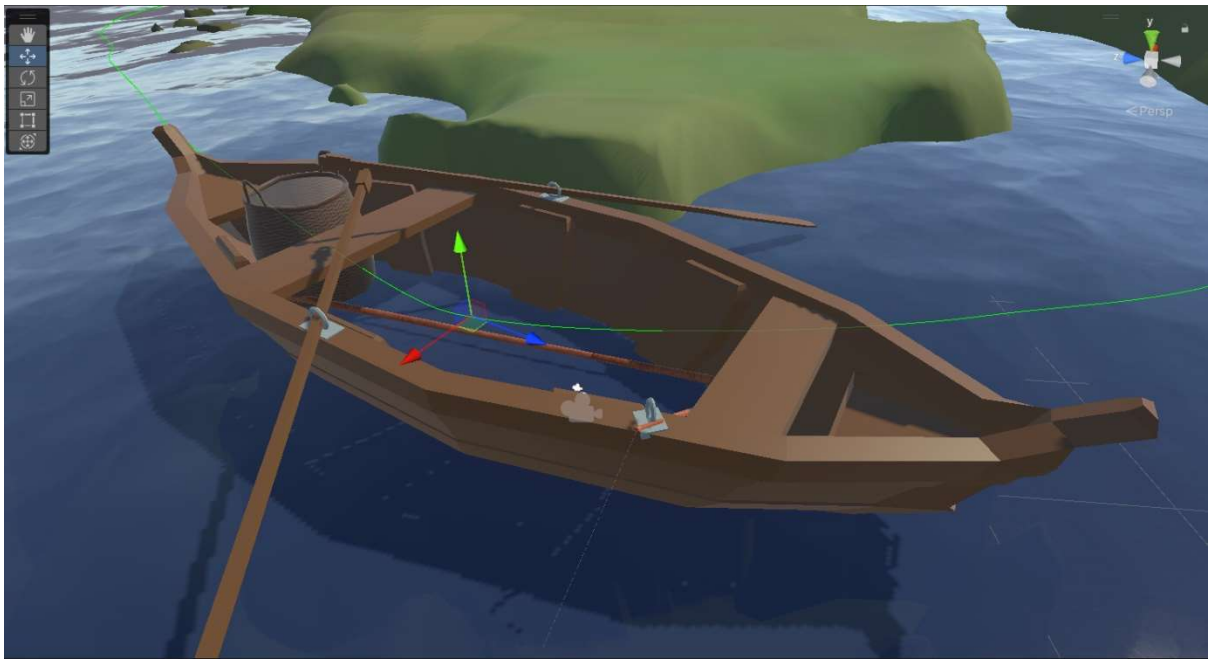


Fig. 5. Original model [3] (top) is modified using Blender (bottom) to slide a seat further along the boat to allow more space for the player to stand and move around in the boat.



Since the water is not actual water particles with physics, but a plane with a shader on it, the water can be seen inside where the plane intersects the boat. To solve this, a depth mask is made using Blender and a script imported from online [4] (see Fig. 6-8.).

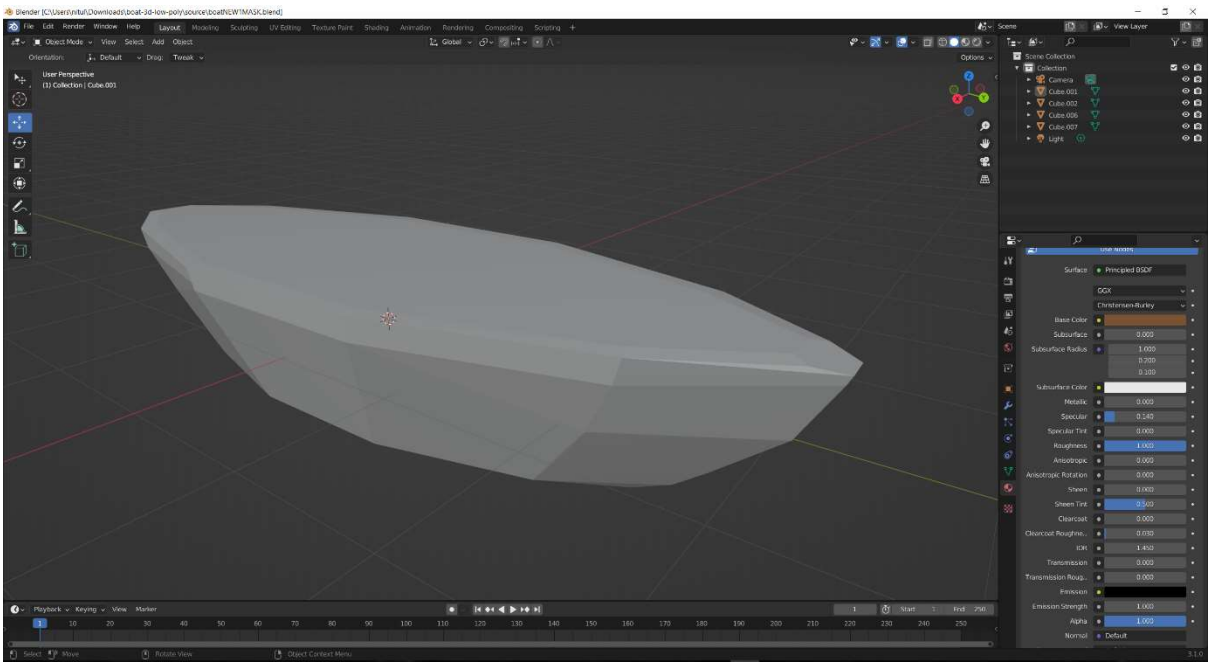


Fig. 6. Solid non-hollow volume in the shape of the boat is made using Blender, by creating a convex hull of the model

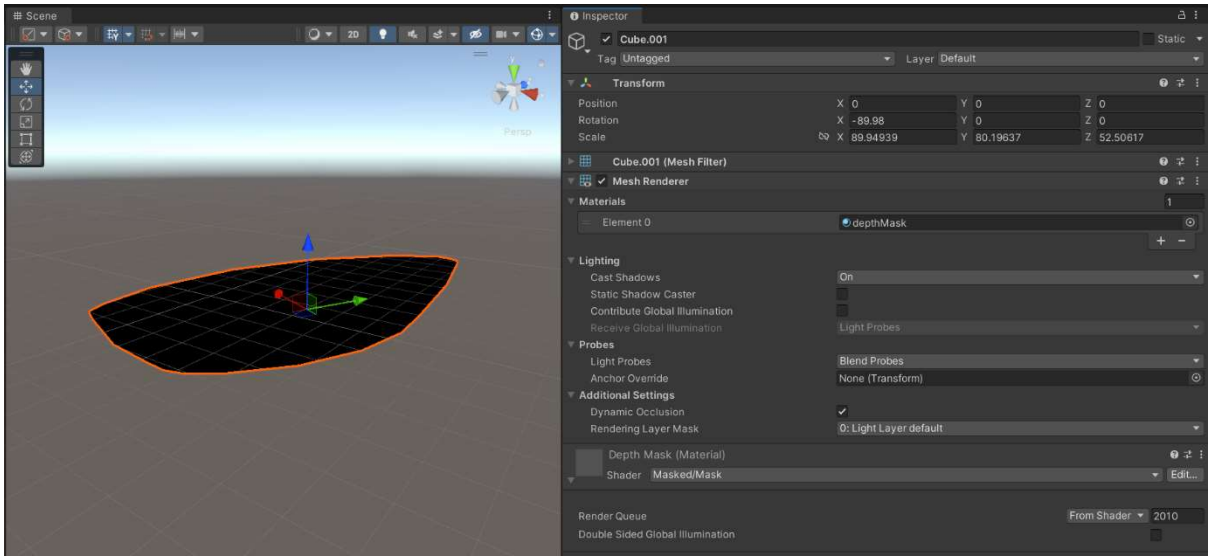


Fig. 7. Depth mask shader added to model in Fig. 4., making the volume “invisible”.

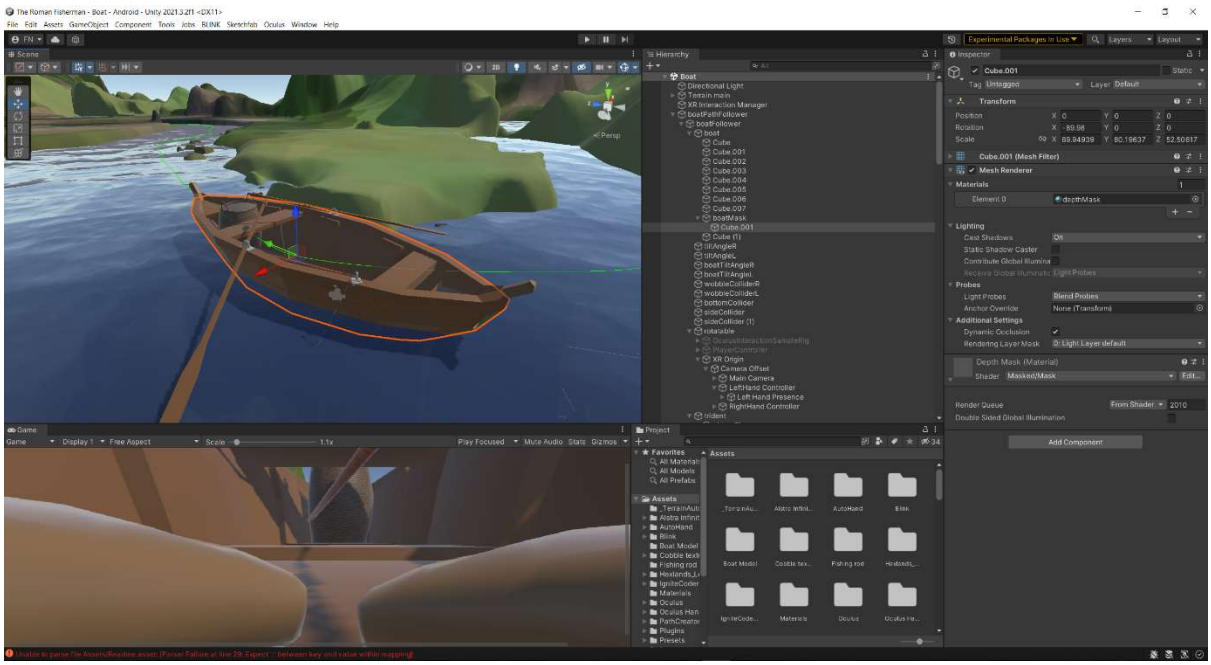


Fig. 8. Volume with depth mask now place inside the boat, and now the plane with the water shader cannot be seen inside where the depth mask volume is.

To hold the trident (Fig. 9-10.) and rod (Fig. 11.) with both hands, a script [7] and a Unity package is imported from [8] with everything set up for interacting with objects with controllers, using the XR Interaction Toolkit. (Fig. 12.).

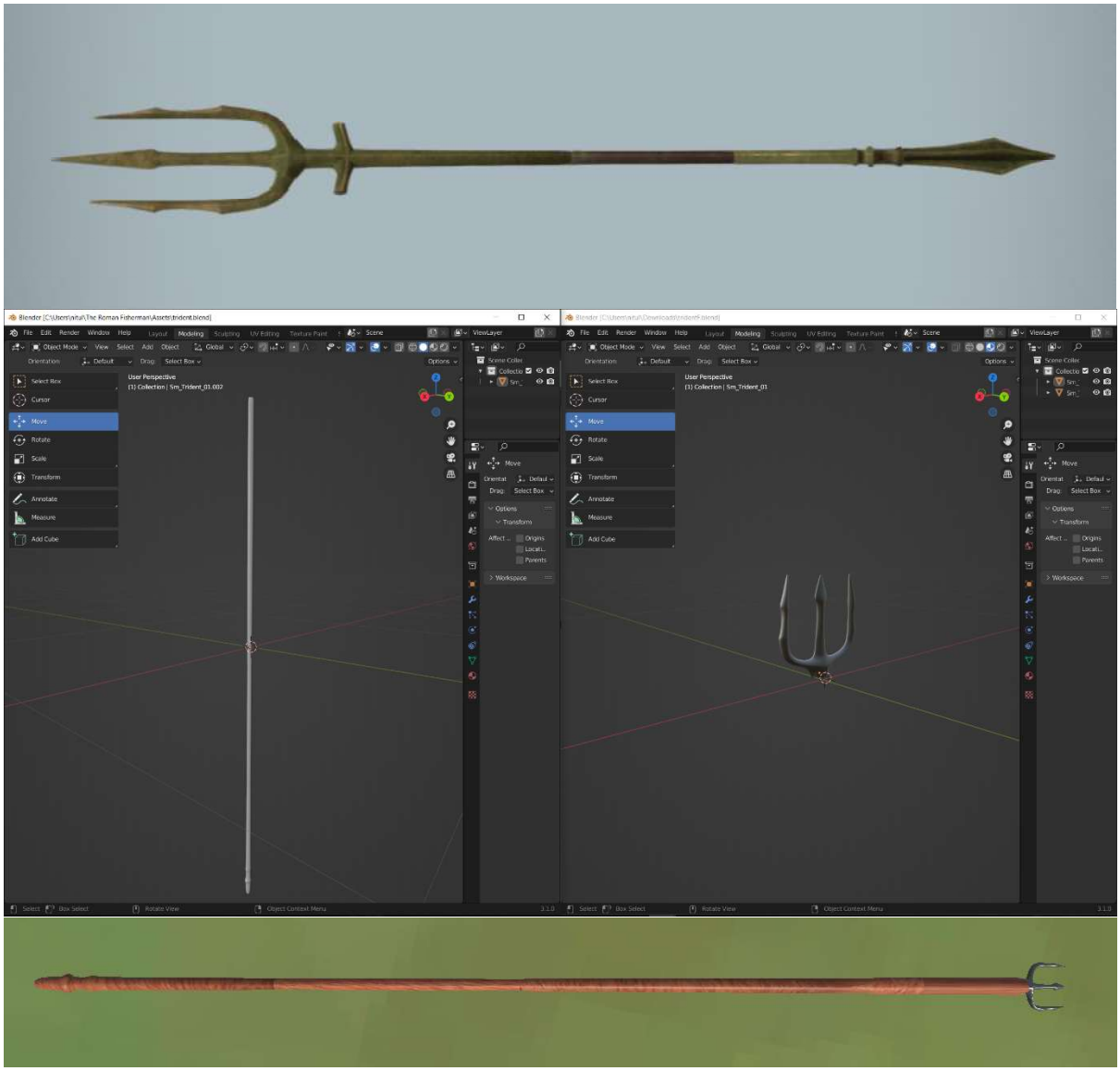


Fig. 9. Original trident model [5] (top) modified using Blender (middle) to separate the trident trip and the stick itself, and removing some parts to make the model simpler. This was done to make it look more historically accurate, as seen from a 3D reconstruction of an ancient Roman boat [6]. The tip and the stick is separated into two individual models so that separate materials can be added to each and resized accordingly (bottom).

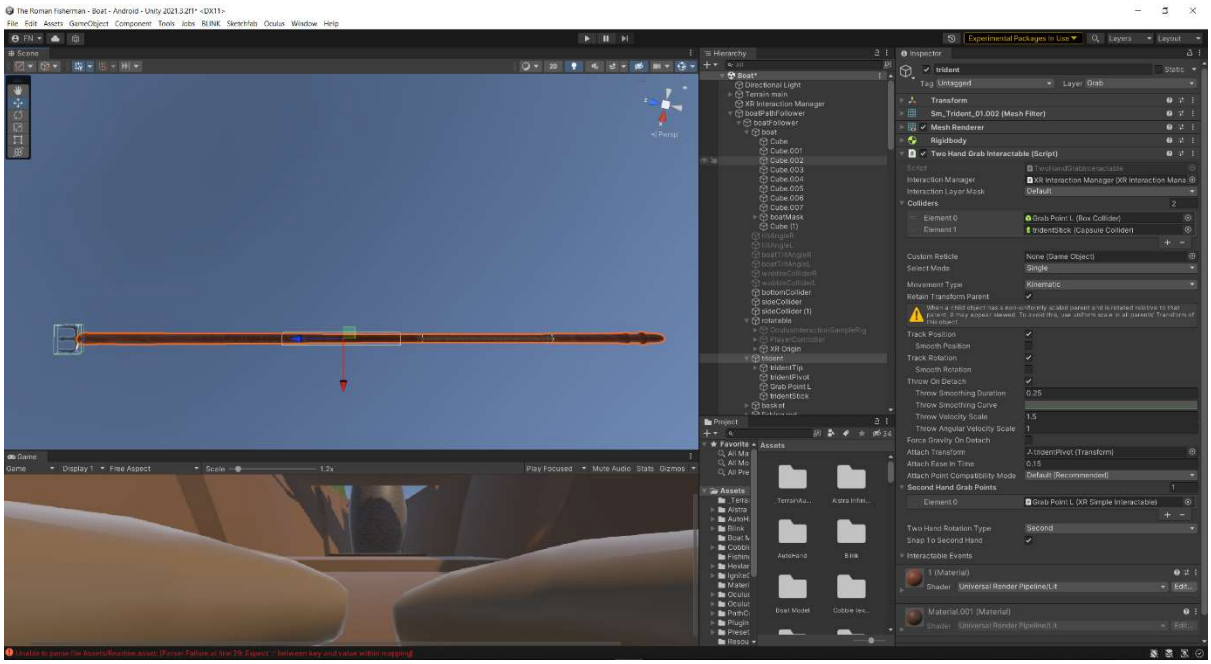


Fig. 10. “Two Hand Grab Interactable” script [6] (see Appendix A for full code) added to trident. Two separate colliders on the trident stick function as the grab points. The trident tip itself is given a collider so that it can collide with fish separately from the stick.

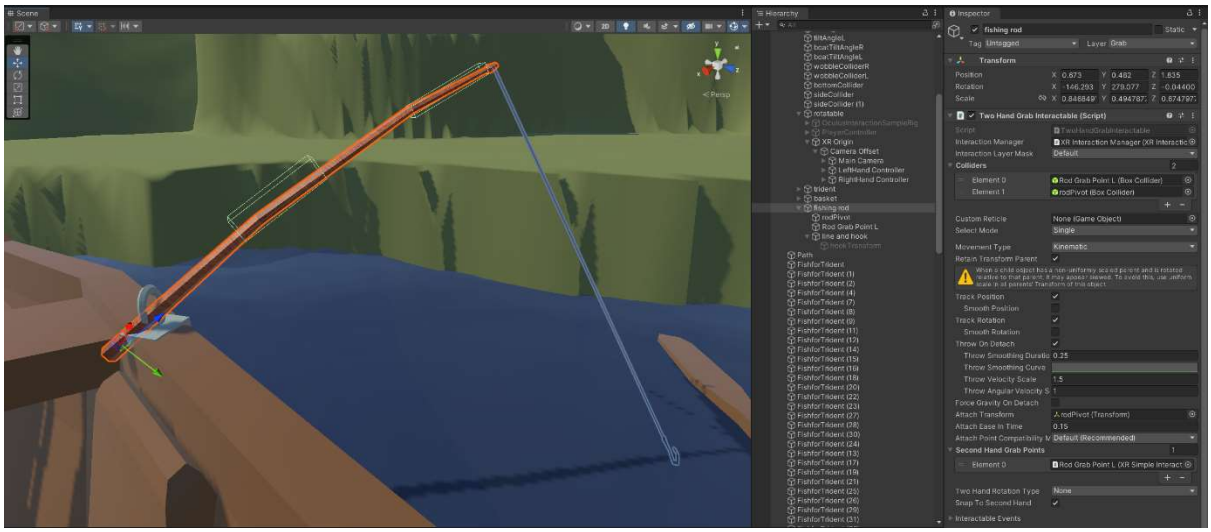


Fig. 11. Fishing rod [7] (with materials changed) with two colliders as the grab points, using the “Two Hand Grab Interactable” script

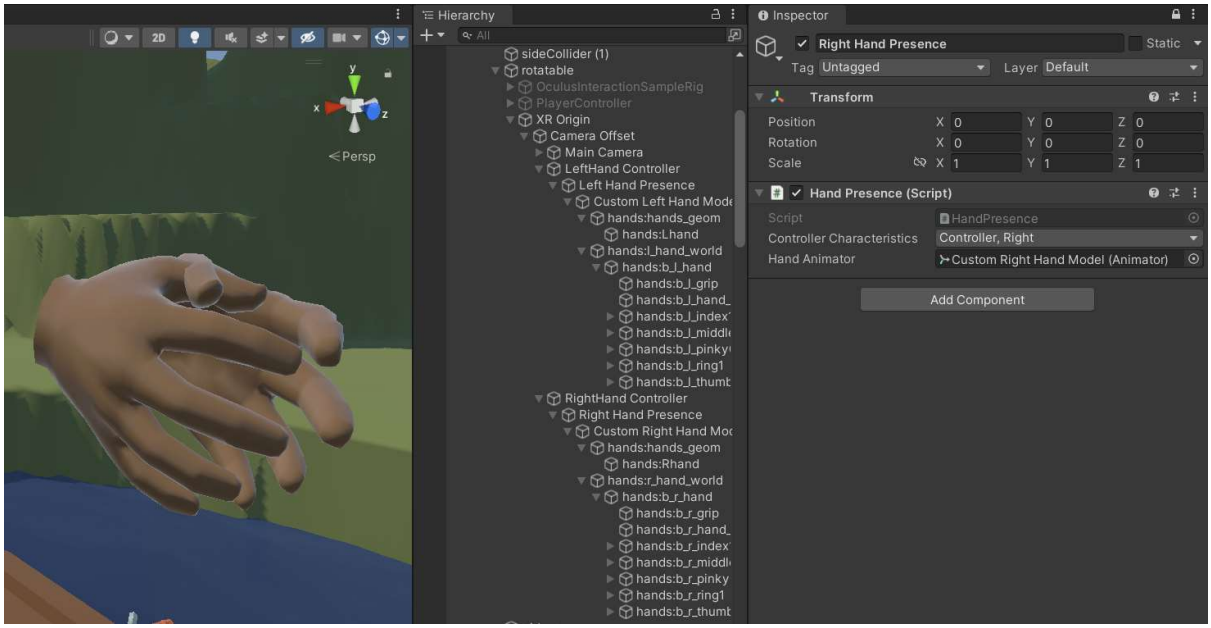


Fig. 12. XR origin with controllers set up with hand prefabs [9] (colour of material changed to a skin-tone), already with grip animations and grip controls, ready to use. These "grip" controls on the Oculus Quest 2 controllers are simply gripping the trigger buttons. The controllers are already ergonomically designed so that pressing the triggers already feels like gripping an object.

To make the boat lean in the direction the player moves, a sphere collider is created which acts as the player's centre of mass (Fig. 13.). Two trigger colliders are placed near each edge of the boat (Fig. 14.), and a script is written to detect a collision between centreOfMass and these colliders so that when there is a collision, the boat will tilt. To simulate the feeling of falling, the camera is tilted in the opposite direction of the boat tilt, accentuating the boat tilt and imbalance even more, an idea taken from observing the camera rotation in the Tightrope demo shown in the taster session [10].

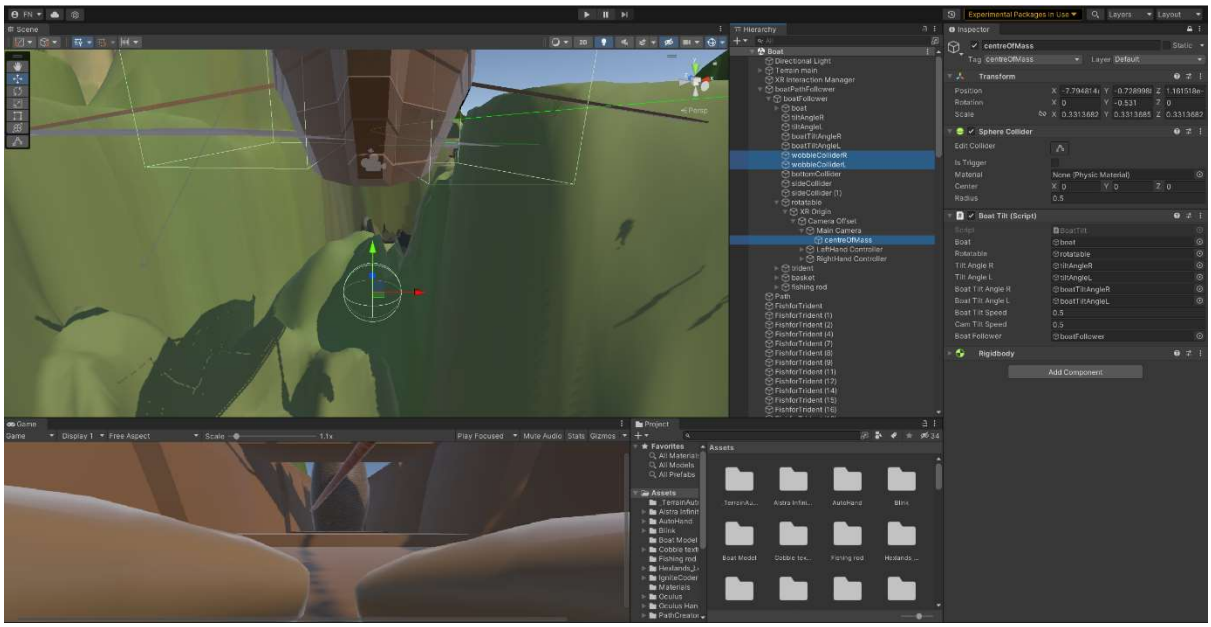


Fig. 13. "centreOfMass" sphere collider placed a distance below the camera that is approximately the distance from a person's head to their stomach. By making centreOfMass a child of the Main Camera, it will move with the camera keeping the same distance, matching the player's actual centre of mass. BoatTilt.cs script attached to centreOfMass (see next page and Appendix B).

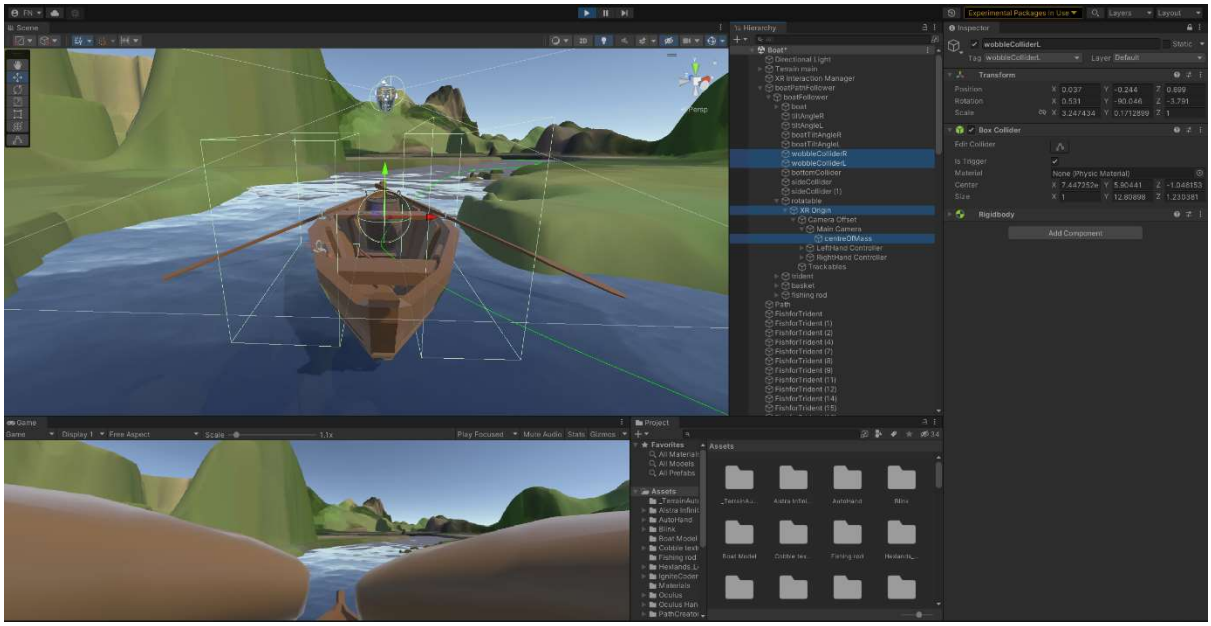


Fig. 14. Two box colliders, as triggers, (labelled wobbleColliderR and wobbleColliderL) on either edge of the boat. Centre of mass moving with the camera (player's head) in play mode.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (2 asset references) | 0 references
public class BoatTilt : MonoBehaviour
{
    public GameObject boat;
    public GameObject rotatable;
    public GameObject tiltAngleR;
    public GameObject tiltAngleL;
    public GameObject boatTiltAngleR;
    public GameObject boatTiltAngleL;
    public float boatTiltSpeed = 30f;
    public float camTiltSpeed = 50f;
    public GameObject boatFollower;
    private bool inTriggerZone;
```

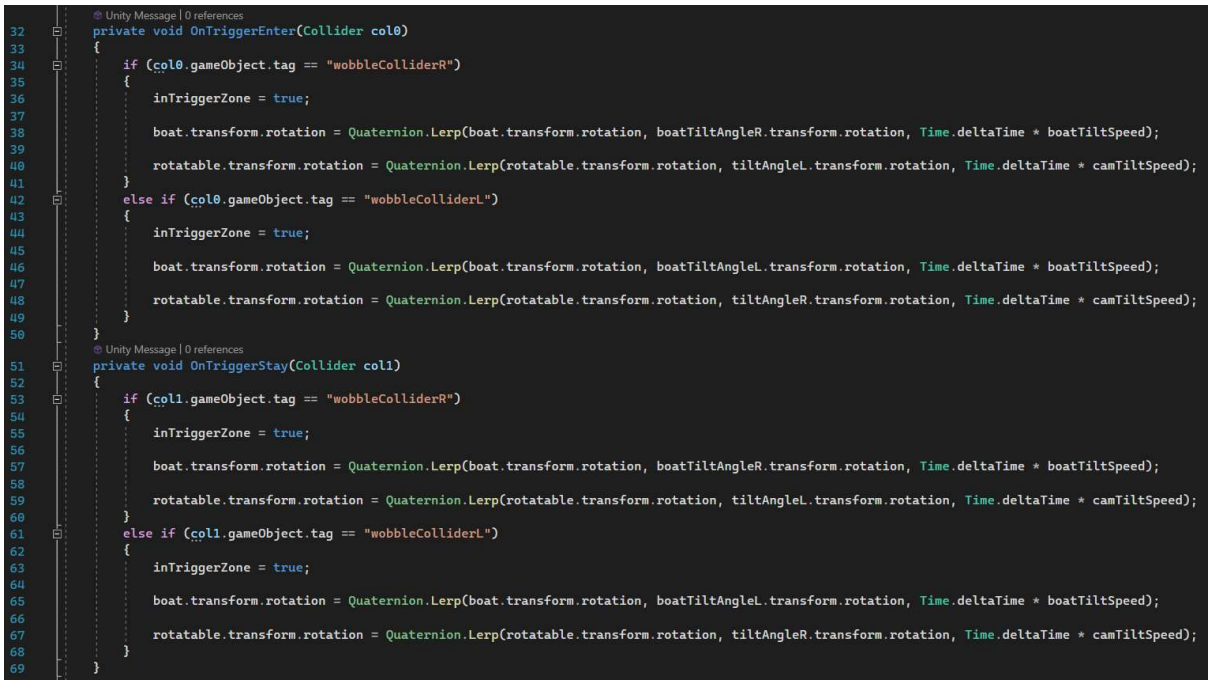
Fig. 15. BoatTilt.cs, Appendix B for full code. Referencing all required game objects and declaring variables. As the XR Origin won't follow any rotations directly (because of how it follows the rotation of the headset), to get the camera tilt, we create an empty game object, "rotatable" and make it a parent of the XR Origin so that the camera will follow the rotation of the parent game object, "rotatable". "tiltAngleR" and "tiltAngleL" are the camera tilt angles, and "boatTiltAngleR" and "boatTiltAngleL" are the boat tilt angles. These are made to be game objects instead of floats since for rotations, we are using quaternions. We want the boat and camera to rotate only in the direction of the sides of the boat (see Fig. 18-19.) so we define a rotated vector3 to which we want the boat and camera to rotate towards, and let these be the transforms of the tilt angle game objects.

To rotate the boat back to its original rotation when the player is not leaning to the side of the boat, Booleans are used to check if colliders are in collision, and code to rotate back to its original rotation is written in void Update() (Fig.16.).

```
Unity Message | 0 references
void Update()
{
    inTriggerZone = false;
    if (inTriggerZone)
    {
        boat.transform.rotation = Quaternion.Lerp(boat.transform.rotation, boatFollower.transform.rotation, Time.deltaTime * boatTiltSpeed);
        rotatable.transform.rotation = Quaternion.Lerp(rotatable.transform.rotation, boatFollower.transform.rotation, Time.deltaTime * camTiltSpeed);
    }
}
```

Fig. 16. BoatTilt.cs, Appendix B. In void Update(), we set the Boolean "inTriggerZone" to false, where centreOfMass is not in collision with the wobble colliders. The transform of the boat and "rotatable" is then rotated to its original rotation. The original rotation is defined by an empty game object, "boatFollower", which is made to be a parent of "boat" and everything on the boat, and its transform is the rotation when the boat is straight. Lerp is used to get a smooth change in rotation. "Time deltaTime * boatTiltSpeed" lets us define the speed of rotation, which can be changed in the Inspector since they are public floats.

To detect collision between centreOfMass and the wobble colliders, code is written under OnTrigger functions to rotate the boat and “rotatable” (Fig. 17.).



```
Unity Message | 0 references
private void OnTriggerEnter(Collider col0)
{
    if (col0.gameObject.tag == "wobbleColliderR")
    {
        inTriggerZone = true;

        boat.transform.rotation = Quaternion.Lerp(boat.transform.rotation, boatTiltAngleR.transform.rotation, Time.deltaTime * boatTiltSpeed);

        rotatable.transform.rotation = Quaternion.Lerp(rotatable.transform.rotation, tiltAngleL.transform.rotation, Time.deltaTime * camTiltSpeed);
    }
    else if (col0.gameObject.tag == "wobbleColliderL")
    {
        inTriggerZone = true;

        boat.transform.rotation = Quaternion.Lerp(boat.transform.rotation, boatTiltAngleL.transform.rotation, Time.deltaTime * boatTiltSpeed);

        rotatable.transform.rotation = Quaternion.Lerp(rotatable.transform.rotation, tiltAngleR.transform.rotation, Time.deltaTime * camTiltSpeed);
    }
}
Unity Message | 0 references
private void OnTriggerStay(Collider col1)
{
    if (col1.gameObject.tag == "wobbleColliderR")
    {
        inTriggerZone = true;

        boat.transform.rotation = Quaternion.Lerp(boat.transform.rotation, boatTiltAngleR.transform.rotation, Time.deltaTime * boatTiltSpeed);

        rotatable.transform.rotation = Quaternion.Lerp(rotatable.transform.rotation, tiltAngleL.transform.rotation, Time.deltaTime * camTiltSpeed);
    }
    else if (col1.gameObject.tag == "wobbleColliderL")
    {
        inTriggerZone = true;

        boat.transform.rotation = Quaternion.Lerp(boat.transform.rotation, boatTiltAngleL.transform.rotation, Time.deltaTime * boatTiltSpeed);

        rotatable.transform.rotation = Quaternion.Lerp(rotatable.transform.rotation, tiltAngleR.transform.rotation, Time.deltaTime * camTiltSpeed);
    }
}
```

Fig. 17. BoatTilt.cs, Appendix B. OnTriggerEnter and OnTriggerStay is used to detect collisions, so that the boat keeps tilting when the player is still leaning to the side. Here, the Boolean “inTriggerZone” is set to true, and Quaternion.Lerp is used to rotate the boat and “rotatable” smoothly to our desired rotation, using if statements so that they rotate only under the condition that the colliders’ tags are the wobble colliders (and so do not rotate upon collision with other colliders).

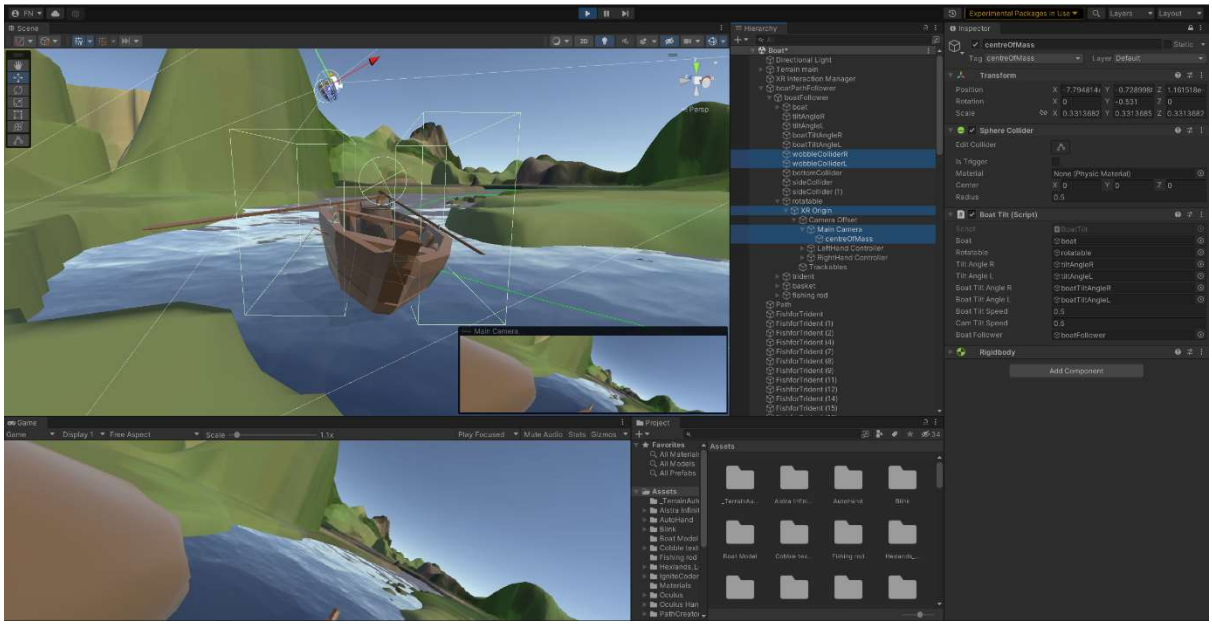


Fig. 18. Play mode: camera rotation offset to the left while boat tilting right when centre of mass is colliding with the collider on the right, i.e., player is leaning their body to the right. Offset between clockwise boat rotation and anti-clockwise camera rotation causes imbalance.

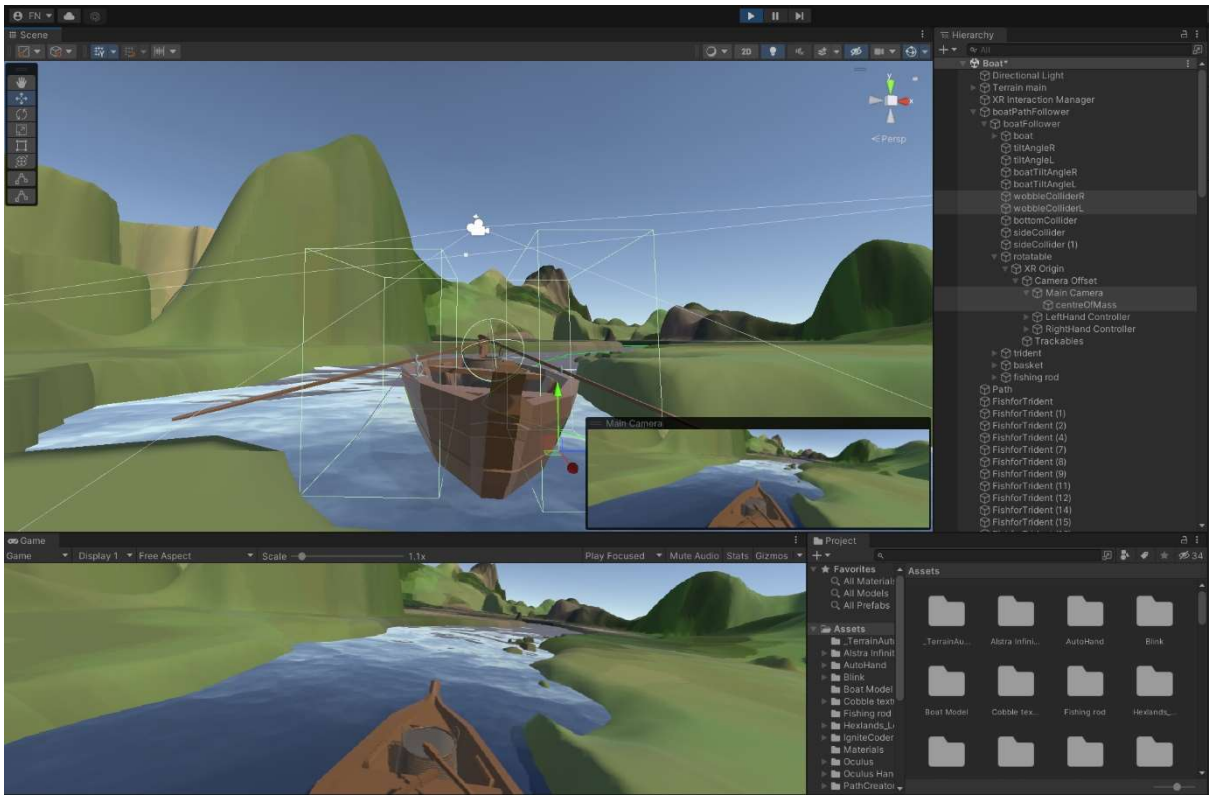


Fig. 19. Screenshot in play mode showing the tilt more clearly. It can be seen that the camera is tilted anti-clockwise as the boat tilts clockwise, the centreOfMass colliding with the wobble collider on the right. Note: this screenshot was taken without the headset on, which is why the XR Origin is shown to the left of the boat due to centreOfMass's collision with the wobble collider. If the headset was on, the XR Origin would be unaffected by this collision and would follow the player's head.

Fish (Fig. 20.), are placed along the shore and given some movement (Fig. 21.) so that they can be seen bouncing in and out the water.

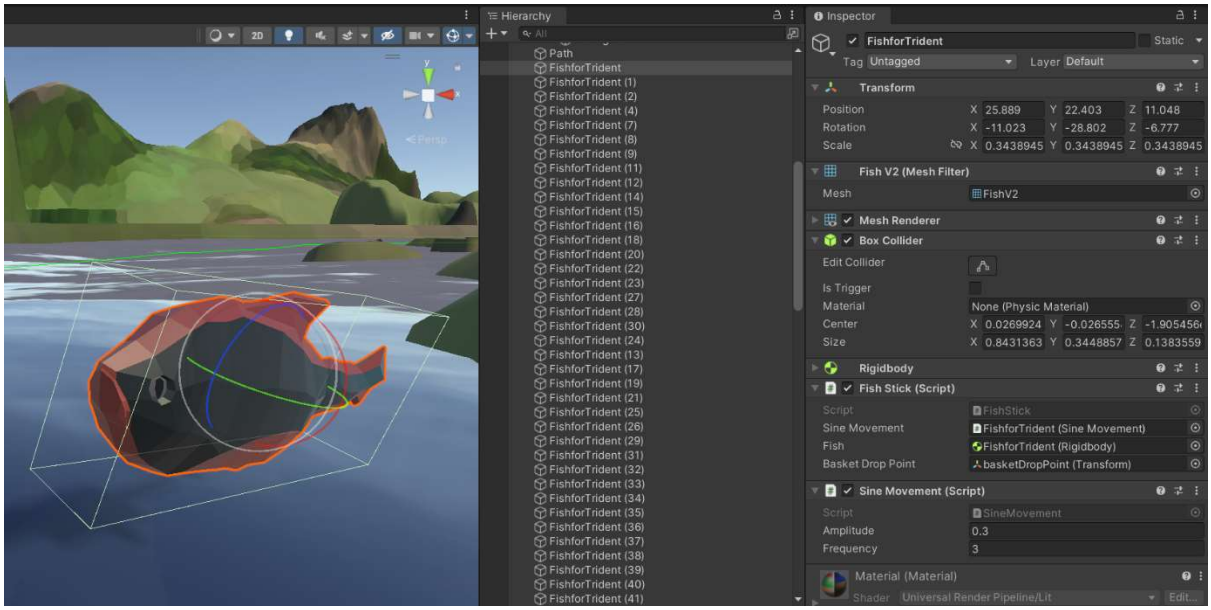


Fig. 20. Fish to be hit by trident. Multiple copies scattered along the shore. Model imported from [11].

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (42 asset references) | 1 reference
public class SineMovement : MonoBehaviour
{
    public float amplitude;
    public float frequency;
    Vector3 originalPosition;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        originalPosition = transform.position;
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        transform.position = new Vector3(originalPosition.x, Mathf.Sin(Time.time * frequency) * amplitude + originalPosition.y, originalPosition.z);
    }
}
```

Fig. 21. SineMovement.cs, Appendix C. The fish's transform is changed to a new vector3 with unchanged x and z components, and the y component following a sine wave so that the fish oscillates up and down. The amplitude and frequency of oscillation is given as public floats so that it can be changed in the Inspector, and randomly changed for different fish so that they're moving out of the water at different times.

To hit the fish, the trident tip is given a trigger collider (Fig. 10.) so that OnTriggerStay can be used for when the fish collide with the trident tip collider. A coroutine is started so that the caught fish automatically drops into the basket after 1 second. See Fig. 22.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (42 asset references) | 1 reference
public class FishStick : MonoBehaviour
{
    public SineMovement SineMovement;
    public Rigidbody fish;
    public Transform basketDropPoint;



    // Start is called before the first frame update
    Unity Message | 0 references
    private void Start()
    {
        SineMovement.enabled = true;
    }



    Unity Message | 0 references
    private void OnTriggerStay(Collider col1)
    {
        if (col1.gameObject.tag == "tridentTipCollider")
        {
            SineMovement.enabled = false;
            transform.position = col1.transform.position;
            StartCoroutine(putInBasket());

        }
    }

    1 reference
    IEnumerator putInBasket()
    {
        yield return new WaitForSeconds(1);
        transform.SetParent(basketDropPoint.transform);
        transform.position = basketDropPoint.position;
        fish.velocity = Vector3.zero;
        fish.useGravity = true;
        fish.isKinematic = false;

    }
}
```

Fig. 22. FishStick.cs, Appendix D. Script attached to fish (FishforTrident in Hierarchy, see Fig. 20.). Upon collision with the trident tip collider the position of the fish is set to the position of the trident tip. A coroutine is then started which waits 1 second, sets the fish as a parent of the basket and changes the fish's position to a point just above the basket, "basketDropPoint" (Fig. 23.). The fish's gravity is turned off, with its velocity set to zero, and non-kinematic, so that it drops straight into the basket. The fish was already moving so we disable the SineMovement.cs script during the collision, and enable it in the update function, so that they don't move when speared.

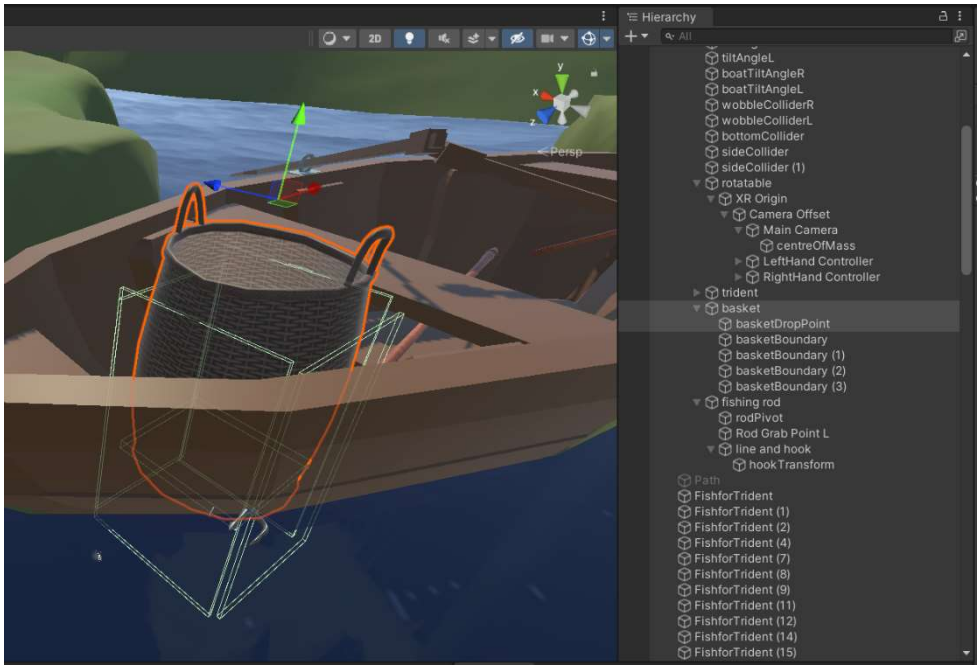


Fig. 23. Basket imported from [12]. basketDropPoint just above the basket. Non-trigger colliders covering all four sides, and the bottom, of the basket so that fish stay in the basket when dropped in.

To move the boat automatically along the river, a Bézier Path Creator is imported from online [13]. This lets us define a custom path for the boat to follow (Fig. 24.).

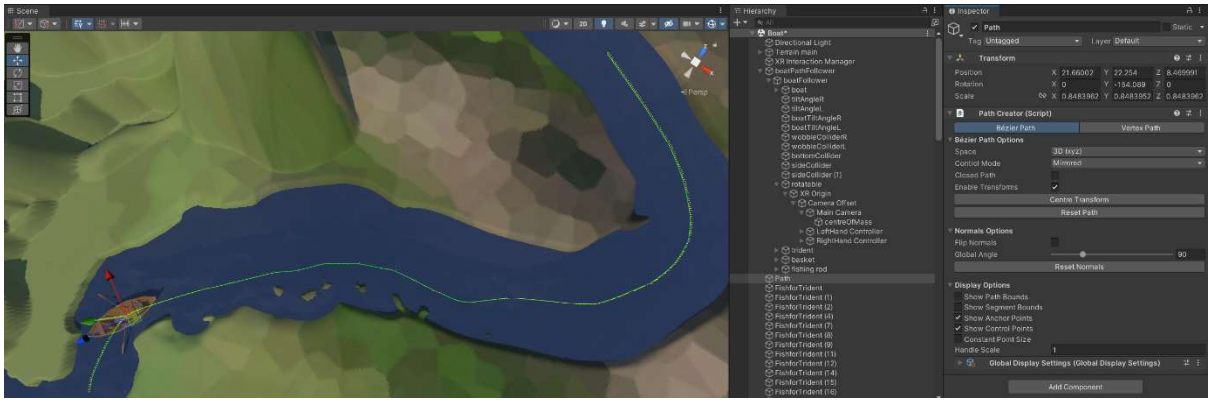


Fig. 24. Green line shows custom drawn path for the boat to follow. Starts off near the shore for the trident fishing, then towards the middle of the river for the next part of the game: using the fishing rod.

This asset comes with a script (Appendix E) that we attach to the boat (Fig. 25.). This script dictates the movement of an object along the custom path. The script was altered (Fig. 26.) so that the boat also oscillates up and down as it follows the path, to simulate buoyancy or water waves.

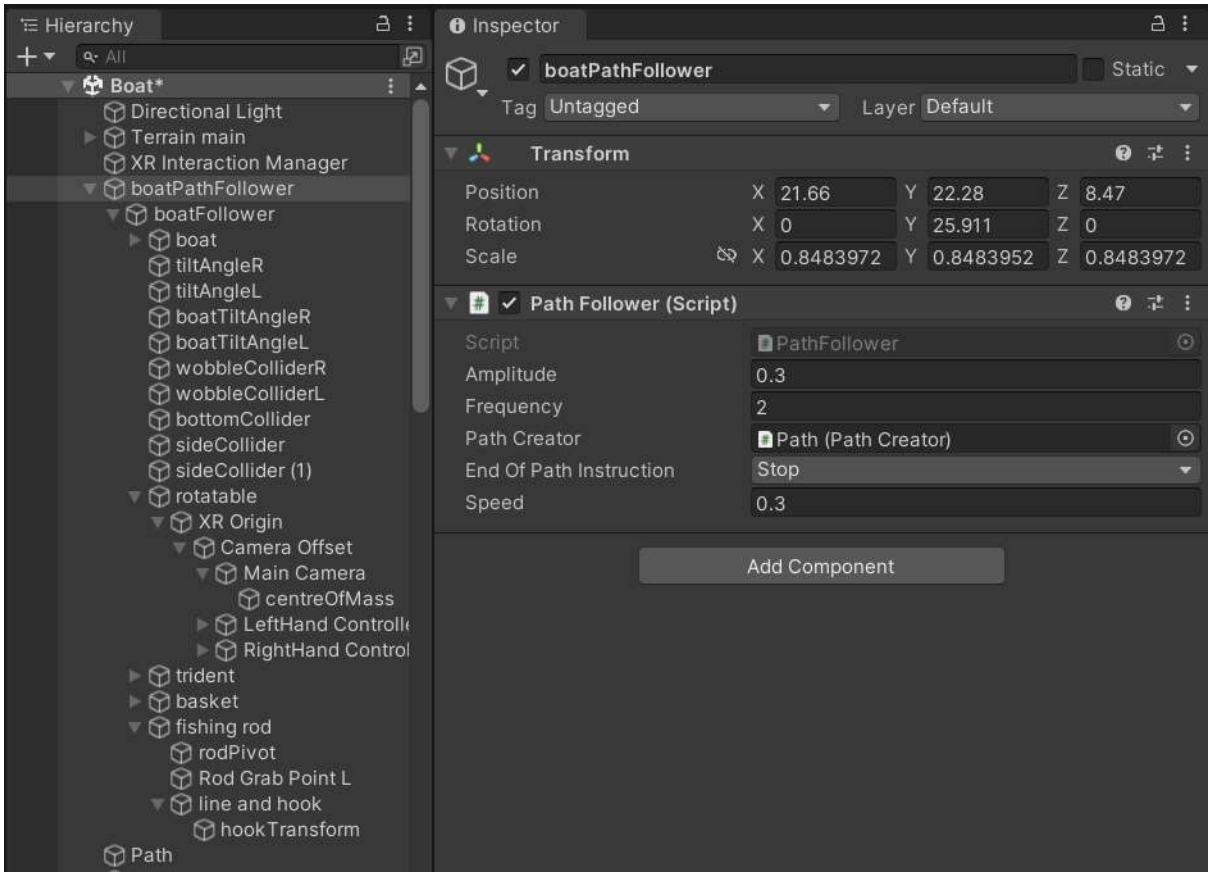
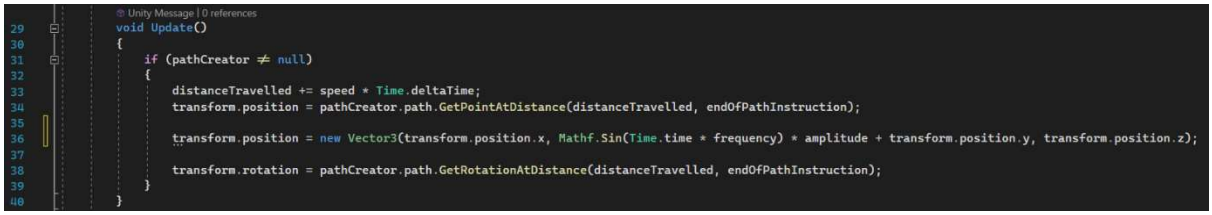


Fig. 25. PathFollower.cs attached to an empty game object, boatPathFollower, parent of the boat and everything on the boat. PathFollower is attached to this parent game object so that the rotations of the boat and other objects are not changed due to the path. Speed of the object on the path (hence the speed of the boat in this case) can be changed in the Inspector, as well as amplitude and frequency of oscillation (Fig. 26., Appendix E).



```
Unity Message | 0 references
29      void Update()
30      {
31          if (pathCreator != null)
32          {
33              distanceTravelled += speed * Time.deltaTime;
34              transform.position = pathCreator.path.GetPointAtDistance(distanceTravelled, endOfPathInstruction);
35
36              transform.position = new Vector3(transform.position.x, Mathf.Sin(Time.time * frequency) * amplitude + transform.position.y, transform.position.z);
37
38              transform.rotation = pathCreator.path.GetRotationAtDistance(distanceTravelled, endOfPathInstruction);
39          }
40      }
```

Fig. 26. PathFollower.cs, Appendix E. Line 36 added by me. transform.position changed so that the y component follows a sine movement, as in SineMovement.cs (Fig. 21., Appendix C). That way the boat oscillates vertically as it follows the path.

When the boat starts moving towards the middle of the river, it will reach an area where there are a lot of fish just under the surface of the water. Here the fish will attach to the hook (Fig. 27.). The first fish will attach to the hook when the boat reaches a bump in the terrain (Fig. 28.) at which point I will tug down on a wire connected to the controller (Fig. 30-31.) for under 7 seconds before letting go, acting as a Wizard of Oz. The player can then pull up the rod and see the fish caught on the hook. At the 7 second mark, the fish will automatically be dropped into the basket (Fig. 29.). This is repeated in 7 second increments until the boat reaches the end of the path, where the game ends.

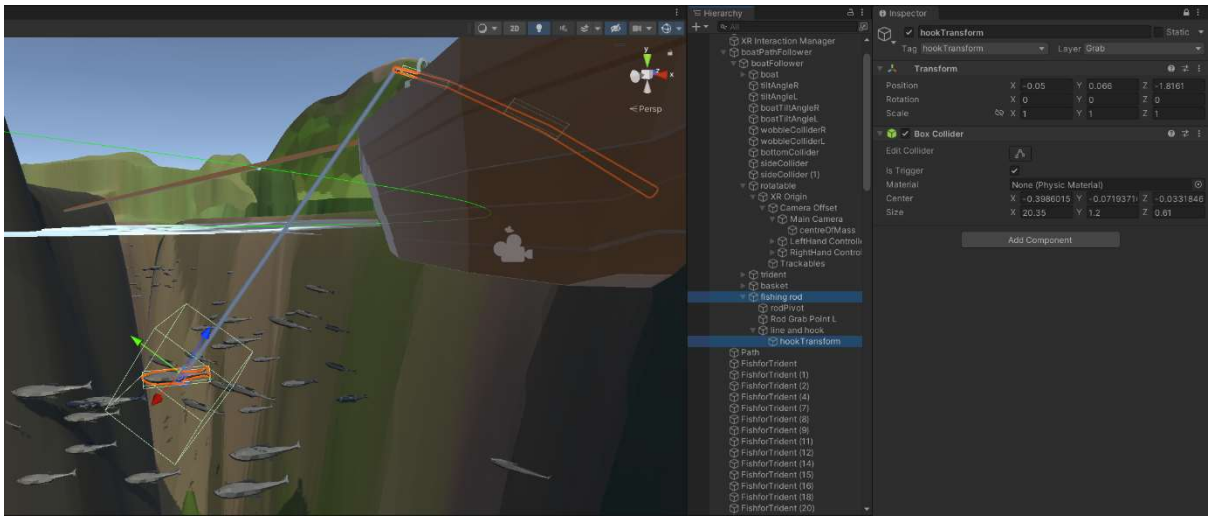


Fig. 27. Box collider around hook. When fish collides with this box collider, the fish's position is changed to the hook's position (Fig. 29.). Fish models imported from [11].

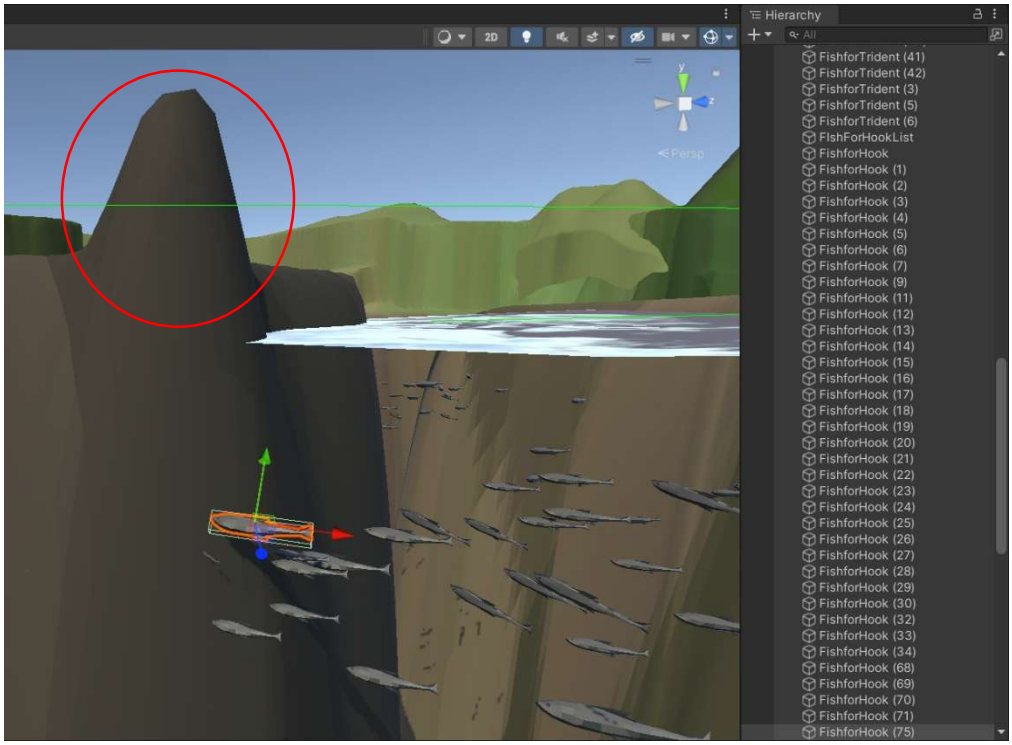
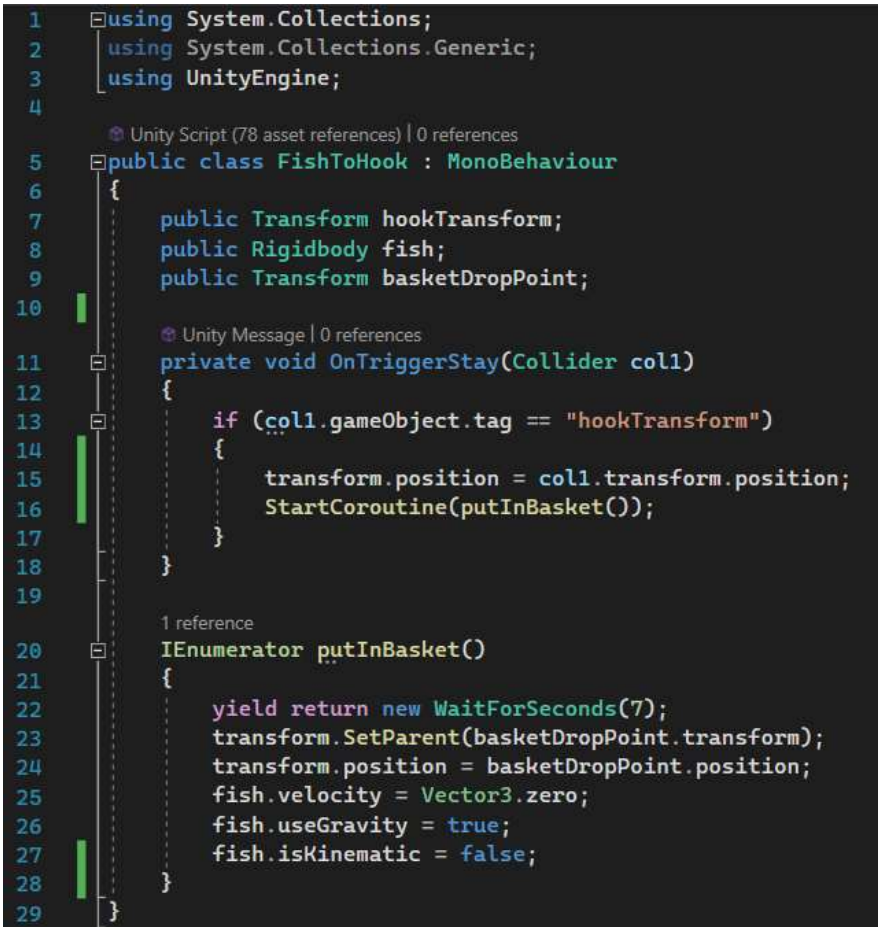


Fig. 28. Circled red is the bump in the terrain. As a Wizard of Oz, I'll be watching the player's screen from my phone (Fig. 31.). When I can see the boat, and the rod, is in line with the bump, I start pulling on the wire.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (78 asset references) | 0 references
public class FishToHook : MonoBehaviour
{
    public Transform hookTransform;
    public Rigidbody fish;
    public Transform basketDropPoint;

    Unity Message | 0 references
    private void OnTriggerStay(Collider col1)
    {
        if (col1.gameObject.tag == "hookTransform")
        {
            transform.position = col1.transform.position;
            StartCoroutine(putInBasket());
        }
    }

    1 reference
    IEnumerator putInBasket()
    {
        yield return new WaitForSeconds(7);
        transform.SetParent(basketDropPoint.transform);
        transform.position = basketDropPoint.position;
        fish.velocity = Vector3.zero;
        fish.useGravity = true;
        fish.isKinematic = false;
    }
}
```

Fig. 29. FishToHook.cs, Appendix F. Script is attached to fish (FishforHook in Hierarchy, see Fig. 28.). OnTriggerStay function used to detect collision between fish and box collider around the hook. In the function, the position of the fish is set to the position of the box collider (which is centred so that its centre is the hook). A coroutine is then started, waiting 7 seconds before dropping the fish into the basket, as in Fig. 22.

Evaluation

Three participants have tested this game, two in one location, one in another location. The locations are small indoor spaces acting as a mock area of the room the experience would be delivered in (Fig. 30). The Quest was cast to my phone to let me see the screen, allowing me to “Wizard of Oz” and time the tugs accordingly (Fig. 31-32.). No COVID restrictions as we did not attend the public museum.

Fig. 30. Participant testing the game in a small open area in an empty classroom. Area acts as a mock area of the small empty room in the museum it would be delivered in. Image taken at the beginning of the game where player is suggested to lean their body or step to the edge of the boat. Participant feels some imbalance causing them to fall on one foot as they feel the tilt, crouching to balance themselves. (Note: only the trident fishing portion of the game was available for testing, here. The full game including the fishing rod experience was only tested by one participant.)

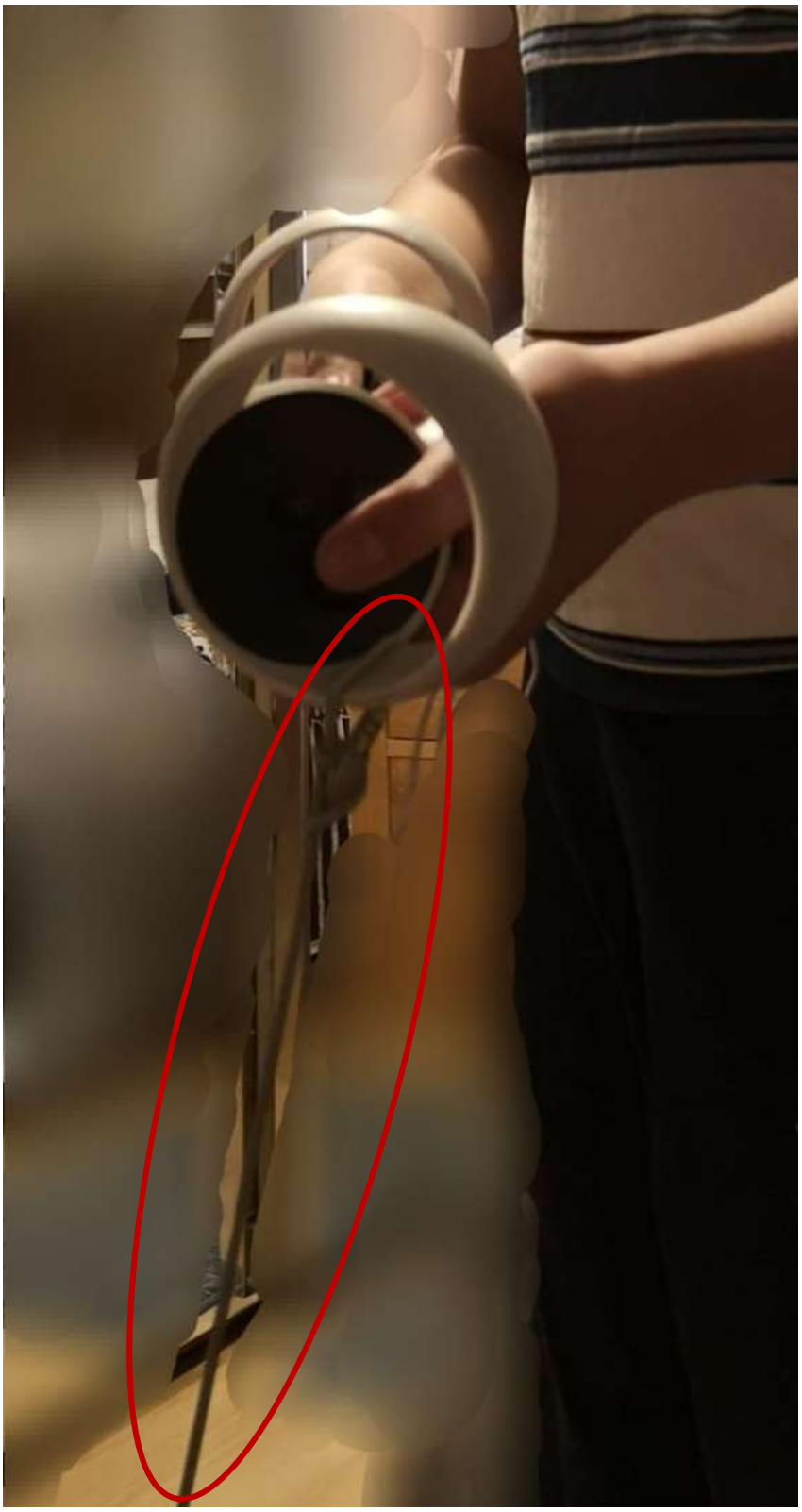


Fig. 31. Circled red is the wire tied around the controller – at a place where the player's fingers are not expected to touch, so the player would not notice the wire tied to the controller when playing. However, testing did show that the player does feel the wire swinging around during the trident fishing, could feel contact of the wire on their arms. (Background blurred for privacy.)

Fig. 32. Quest is casting to my phone (circled red), so that I can see the player's view, know when the player has picked up the rod, and begun fishing, from which I can time the pulls accordingly. Note: in the image, the phone is on the floor very close to the player's foot. This was just for a few seconds when I was taking the photo. During the experience, the phone was in my hand. (Background blurred for privacy.)

The idea is that simulating boat wobble should challenge the player as they will need to lean towards the edge of the boat often to reach the fish with the trident. I wanted to investigate the effect of the imbalance on the task difficulty (the task in this case being hitting the fish with the trident), and how this could cause immersion or presence, see Fig. 33. Table 1 shows results.

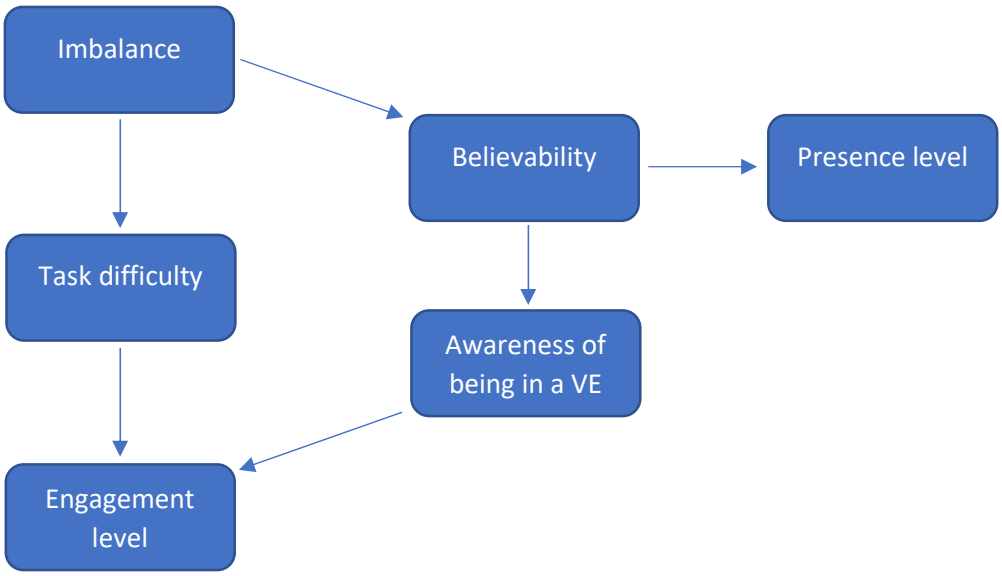


Fig. 33. Diagram showing the possible immersion process, with the root cause being the imbalance and how it may affect engagement levels by task difficulty or believability (which, can also affect presence). Direction of arrow represents causation.

Question	Participant A	Participant B	Participant C
[Visual] On a scale of 1 to 7, how much of the boat tilt were you able to *see* during the trident fishing? I.e., how much did you *visually* notice the edge of the boat being closer to the water as you lean or step towards the edge? *1 being you didn't see the boat tilt at all, and the boat looked exactly as it did before you leaned towards the edge. 7 being you saw the boat tilt a lot and can clearly see the edge of the boat is lower than before and almost touching the surface of the water.*	3 Comment: *Facing the side of the boat, you couldn't really tell.*	6 Comment: *The water looked like it was going to spill into the boat.*	7 Comment: *Water was in the boat at one point when I leaned really far out.*
[Proprioceptive] On a scale of 1 to 7, how much did you *feel* the boat tilt during the trident fishing? I.e., how much physical imbalance did your body feel when you leaned or took a step to the edge of the boat? *1 being you felt completely in balance, and felt no change in equilibrium as you leaned or stepped towards the edge of the boat. 7 being you felt like you were going to fall and felt the need to brace yourself by taking a step away from the edge, or in extreme cases, actually fall to the ground.*	3 Comment: *For a second at the start of the fishing, but you kinda get used to it pretty quick.*	3 Comment: *The flat floor made me feel it less.*	6
[Task Difficulty] On a scale of 1 to 7, how much did the feeling of being imbalanced make it difficult to hit the fish with the trident? Note: this isn't asking about how difficult the task was, but how much the imbalance affected the difficulty. *1 being the boat wobble made no difference to the difficulty and it would have been just as hard (or easy) without the wobble. 7 being you felt extremely frustrated, and the boat wobble severely affected your ability to hit the fish.*	2	4 Comment: *Sense of distance was a bit weird with the boat being so close to the water.*	6 Comment: *I kept having to step back away from the edge.*
On a scale of 1 to 7, during the trident fishing, how aware were you that you were in a virtual environment? *1 being you had full realisation that you were in the physical room. 7 being you did not realise or completely forgot that you were in a virtual environment.*	3	4 Comment: *I did forget about the flat floor when I was focusing on the fishing.*	4 Comment: *I could feel the wire that was connected to the controller.*

On a scale of 1 to 7, how involved did you feel in the activity of trident fishing? *1 being you did not feel involved at all, and you barely took part in the task. 7 being you were fully, actively, participating in the trident fishing and you were completely focused on the task at hand.*	3 Comment: *There was some shaking of the screen that was pretty distracting.*	5 Comment: *I was able to focus on the fishing, but it didn't exactly feel like I was hitting fish.*	6
How much did it feel like you were physically standing and wobbling on a boat on the river? *1 being none at all and it purely felt like you were just standing on the floor of the physical room you are in. 7 being the sensation was strong, and maybe you truly believed you were standing and wobbling on a real boat on a real river.*	5 Comment: *When you're facing the front, you feel it a lot more. You also notice the boat going up and down. But there's some shakiness that doesn't feel natural.*	4	6 Comment: *It felt like I was at sea at times with the boat moving up and down.*
How much did it feel like you were actually pulling fish up with the fishing rod? *1 being it didn't feel real at all and didn't feel like you were pulling up real fish with the fishing rod. 7 being it felt like real fishing, and it felt like there were actual real fish caught on the hook, tugging on the line.*	n/a	n/a	5 Comment: *At the start it was pretty surprising but then I remembered I felt the wire against my hand and my arm before so I figured it was you pulling the wire this time. That made it a bit less believable. Though the feeling felt real, in the back of my head I was aware.*

Did the environment around you in the game feel more like images that you saw or more like an actual place that you visited? *1 being it just felt like images and not real. 7 being it felt like visiting an actual place somewhere.*	5 Comment: *Graphics wasn't great, but because you're in VR, it feels a lot more real like you're in another place.*	3 Comment: *When a bit of water went into the boat, the water just went back out like it was flat. Also the trident looked weightless. At that point it really felt more like an image.*	6 Comment: *It was weird how my hands didn't look like it was gripped around the spear and the fishing rod. But it really felt like I was on a boat so I'd still say it felt like I was actually there.*
During the experience as a whole, to what extent were there times during the game when the virtual game environment physically felt like reality to you? *1 being at no times, and the physical room was always the reality for you. 7 being almost all the time the virtual game environment felt physically real, and was your reality.*	4 Comment: *I was talking to you and asking questions during the game so that kinda took me out of it for a bit.*	4 Comment: *I did almost forget that I was in VR during the fishing but I really noticed the flat floor when I was just standing there and you get reminded that you're not on a boat.*	5 Comment: *Only because I felt the wire brush against me. But the boat experience felt a lot more real like I said.*

Table 1: Results of questions (all measured on a 1-7 scale) with any comments of interest noted down. Participants were asked on how much they noticed the boat tilt when trident fishing, both from a visual and proprioceptive standpoint (proprioception being the body's sense of position, including sense of equilibrium) where the idea is: visual + proprioception ⇒ believability. Questions on real-world awareness and involvement/active participation to measure immersion, as these are good indicators of immersion [14]. Some questions adapted from [15] to measure presence. All questions were asked and answered vocally in an interview, and questions were asked using the exact words written in the table – questions that I had prepared beforehand. When needed, further explanation on the question was given. Participants, where they were together in the first place, were asked the questions individually but in the same room (they were not physically separated).

Testing has revealed that whether or not the participant feel the boat tilt is largely dependent on the direction they are facing. Participants when facing forward felt the boat tilt more than when facing the side (as is the case during the trident fishing). Another reason for not feeling the boat tilt is the flat floor they are standing on, as noted by Participant B. When the player steps side to side, their feet cover more surface area, so they feel the flat ground more. We can make them notice this less by making the boat floor narrower or decreasing the distance between the centreOfMass and wobble colliders. Perhaps on a future installation, a physical wobble board for players to stand on can be used to simulate the boat wobble.

On the note of the physical staging of the experience, Participant C was able to feel the wire against their hand. Being aware of the wire affected the believability of the fish pulling, but as the force feedback was still there, this still offered an engaging experience. So, a potential fix could be instead of a wire that would get in the way, attach something small with a hole to the controller, then use a long hook for the pulling, inserting the hook into the hole.

Results show that the idea, visual + proprioception ⇒ believability ⇒ presence, is mostly accurate, with participants (when facing forward) feeling more present when it was more believable that they were on a boat. The boat oscillation was also key in making the boat feel “real”.

Only Participant C noted the boat tilt as a significant challenge to the task. From observation, this is likely because their movement was much more exaggerated than the other two. Though from observing that all participants were able to catch the fish fairly easily, the boat tilt is perhaps insufficient for challenge-based immersion. But results show that immersion was more influenced by the sensation of being on a boat, than the challenge of fishing, except for Participant B who was more task focused than the other two.

Though despite the fishing not necessarily being challenging, the physical nature of the task itself was enough for engagement, as no participants commented on the fact that there is nobody rowing the boat.

The jittering was noticed by all participants, though only Participant A noted it as significantly distracting. This could potentially be fixed by referencing the colliders as rigidBodies when detecting the collisions in the boatTilt script, and adjusting the interpolation.

A lot of participants weren’t sure what to do in the game (e.g. when to pick up objects) despite being told beforehand. I was concerned about the effect of me giving them vocal instructions during the game on their presence and immersion, but I found that it didn’t affect them since they were engaged in the activity.

Reflection

The idea of centring the experience around the sensory misalignment came from the tightrope demo [10]. The idea came from me thinking of other ways I can utilise the feeling of imbalance. What, other than a tightrope, plays with your sense of balance? A boat.

Quoting Slater from [16] on how to achieve presence, “use knowledge of the perceptual system to find out what is important in our representations of reality”. Proprioception is important in our perception of reality, so the boat wobble being proprioceptively accurate should induce presence. Slater in states that a sign of presence is the behaviour of someone in a VE being similar to what their behaviour would be if that VE were real. By this logic, the boat wobble in this game should induce presence if the player attempts to rebalance themselves after leaning towards the edge of the boat, as if they were on a real boat. This concept of believability is what I used in Fig. 33. And the results agree with this. Participant C, who had the highest presence score, commented, “I kept having to step back away from the edge”.

From looking at the SCI-model [17]. I thought that the boat wobble could induce challenge-based immersion. But as seen from the testing, proprioceptive feedback was a bigger influencer (for this game) than the challenge created by the boat wobble. When thinking of sensory immersion, I was mainly focused on audiovisuality, as in the SCI-model, but this paper [18] shows that proprioceptive feedback is key in sensory immersion, with movement-based interaction which is prevalent in this game.

Participant B commented on how the trident looked weightless. This paper [19] looks at changing the perceived weight of a virtual object by offsetting the visual height of an object as it is being picked up, to make it look heavier. However, they find that proprioception provides a more reliable cue to the weight of an object than the visual height.

So, for the fishing rod, a kinaesthetic cue, having some force feedback, would be ideal. Originally, I thought of having some sort of equipment with masses and springs to simulate the tugging force. But since it would take too long to prototype, and cause more problems, I figured I’d just use my hands. And then thinking about it even more, I thought that may even be better. I could make it more random, and since it’s a real living being pulling, perhaps that’s more representative of an actual fish. I still had to wizard of oz the timing since it wasn’t automated.

One interesting thing to consider for future designers is exposition. Having an “exposition fairy” [20] (a character to provide exposition in the game such as controls and objectives) as the boat rower would eliminate the need for someone in the physical room to give instructions during the game. These types of characters are often criticised for giving exposition dumps in video games, but for this game, there is no intricate lore to provide, only simple controls, objectives, and perhaps a backstory to the rower the player can empathise with. This not only provides emotional engagement but also adds life to this virtual world, which is otherwise empty, perhaps making it feel more like Rome.

References

[1] IgniteCoders (2021), Simple Water Shader URP (shader), Available under Standard Unity Asset Store EULA license agreement, at: https://assetstore.unity.com/packages/2d/textures-materials/water/simple-water-shader-urp-191449#publisher (Accessed: 19 April 2022)

[2] RRFreelance / PiXelBurner (2019), HexLands - Low Poly Style (texture/material), Available under Standard Unity Asset Store EULA license agreement at: https://assetstore.unity.com/packages/2d/textures-materials/tiles/hexlands-low-poly-style-133586#publisher (Accessed: 25 April 2022)

[3] shevchenkomr29 (2021), Boat 3d low-poly (3D Model), Available under CC Attribution license, at: https://sketchfab.com/3d-models/boat-3d-low-poly-cc4e4619d8994b71b1f9230033cd1947 (Accessed: 7 April 2022)

[4] ShaderLab - DepthMask.shader (shader script), Available at: https://web.archive.org/web/20210831213650/http://wiki.unity3d.com:80/index.php/DepthMask (Accessed: 9 April 2022)

[5] MightyPinecone (2021), Mossy Golden Trident (3D Model), Available under CC Attribution license, at: https://sketchfab.com/3d-models/mossy-golden-trident-1894ef7418434c17b9860a518ad818d9 (Accessed: 25 April 2022)

[6] CNRS (2020), 3D reconstructions of boats from the ancient port of Rome, https://www.cnrs.fr/en/3d-reconstructions-boats-ancient-port-rome (Accessed: 30 March 2022)

[7] Valem (2020), “Introduction to VR in Unity - PART 10 : TWO HAND GRAB” (online video), Available at: https://www.youtube.com/watch?v=Ie0-oKN3Lq0 (Accessed 18 April 2022)

[8] Quentin Valembois (2022), Oculus Hands Physics.unitypackage (Unity package), Available at: https://drive.google.com/file/d/15LQ3egbh-dlVeDtXJShwwFxTH52d_kCC/view (Acceseed: 25 April 2022)

[9] Inuciian (2017), Fishing Rod (3D Model), Available under CC Attribution license, at: https://sketchfab.com/3d-models/fishing-rod-ab78aaf5ea154edcabdc118196916ddd (Accessed: 10 May 2022)

[10] Tightrope, VR taster session demo, COMP4036 Mixed Reality (2022)

[11] Alstra Infinite, Fish – PolyPack (3D Model), Available under Standard Unity Asset Store EULA license agreement, at: https://assetstore.unity.com/packages/3d/characters/animals/fish/fish-polypack-202232#publisher (Accessed 3 May 2022)

[12] koss2712 (2018), Wicker Basket Pack (3D Model), Available under CC Attribution license, at: https://sketchfab.com/3d-models/wicker-basket-pack-0783e48c8c71463582f4e0a4535aef9a (Accessed: 3 May 2022)

[13] Sebastian Lague (2019), Bézier Path Creator, Available under Standard Unity Asset Store EULA license agreement, at: https://assetstore.unity.com/packages/tools/utilities/b-zier-path-creator-136082#description (Accessed: 26 April 2022)

[14] Blumenthal, V., 2020. Consumer immersion in managed visitor attractions: The role of individual responses and antecedent factors. Scandinavian Journal of Hospitality and Tourism, 20(1), pp.4-27.

[15] Usoh, M., Catena, E., Arman, S. and Slater, M., 2000. Using presence questionnaires in reality. Presence, 9(5), pp.497-503.

[16] Slater, M., 2003. A note on presence terminology. Presence connect, 3(3), pp.1-5.

[17] Mäyrä, F. and Ermi, L., 2011. Fundamental components of the gameplay experience. Digarec Series, (6), pp.88-115.

[18] Pasch, M., Bianchi-Berthouze, N., Dijk, B.V. and Nijholt, A., 2009, June. Immersion in movement-based interaction. In International Conference on Intelligent Technologies for Interactive Entertainment (pp. 169-180). Springer, Berlin, Heidelberg.

[19] Samad, M., Gatti, E., Hermes, A., Benko, H. and Parise, C., 2019, May. Pseudo-haptic weight: Changing the perceived weight of virtual objects by manipulating control-display ratio. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1-13).

[20] TVTropes (2022), “Exposition Fairy” (Article), Available at: https://tvtropes.org/pmwiki/pmwiki.php/Main/ExpositionFairy (Accessed: 29 April 2022)

Appendix A - TwoHandGrabInteractable.cs [6]


```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.XR.Interaction.Toolkit;
5
6  public class TwoHandGrabInteractable : XRGrabInteractable
7  {
8      public List<XRSimpleInteractable> secondHandGrabPoints = new
         List<XRSimpleInteractable>();
9      private XRBaseInteractor secondInteractor;
10     private Quaternion attachInitialRotation;
11     public enum TwoHandRotationType { None, First, Second };
12     public TwoHandRotationType twoHandRotationType;
13     public bool snapToSecondHand = true;
14     private Quaternion initialRotationOffset;
15
16     // Start is called before the first frame update
17     [System.Obsolete]
18     void Start()
19     {
20         foreach (var item in secondHandGrabPoints)
21         {
22             item.onSelectEntered.AddListener(OnSecondHandGrab);
23             item.onSelectExited.AddListener(OnSecondHandRelease);
24         }
25     }
26
27     // Update is called once per frame
28     void Update()
29     {
30
31     }
32
33     [System.Obsolete]
34     public override void ProcessInteractable
         (XRInteractionUpdateOrder.UpdatePhase updatePhase)
35     {
36         if (secondInteractor && selectingInteractor)
37         {
38             //Compute the rotation
39             if(snapToSecondHand)
40                 selectingInteractor.attachTransform.rotation =
                     GetTwoHandRotation();
41             else
42                 selectingInteractor.attachTransform.rotation =
                     GetTwoHandRotation() * initialRotationOffset;
43         }
44         base.ProcessInteractable(updatePhase);
45     }
46
47     [System.Obsolete]
48     private Quaternion GetTwoHandRotation()
49     {
```

```
50          Quaternion targetRotation;
51          if (twoHandRotationType == TwoHandRotationType.None)
52          {
53              targetRotation = Quaternion.LookRotation
                  (secondInteractor.attachTransform.position -
                  selectingInteractor.attachTransform.position); ;
54          }
55          else if (twoHandRotationType == TwoHandRotationType.First)
56          {
57              targetRotation = Quaternion.LookRotation
                  (secondInteractor.attachTransform.position -
                  selectingInteractor.attachTransform.position,
                  selectingInteractor.attachTransform.up);
58          }
59          else
60          {
61              targetRotation = Quaternion.LookRotation
                  (secondInteractor.attachTransform.position -
                  selectingInteractor.attachTransform.position,
                  secondInteractor.attachTransform.up);
62          }
63
64          return targetRotation;
65      }
66
67      [System.Obsolete]
68      public void OnSecondHandGrab(XRBaseInteractor interactor)
69      {
70          Debug.Log("SECOND HAND GRAB");
71          secondInteractor = interactor;
72          initialRotationOffset = Quaternion.Inverse(GetTwoHandRotation
              ()) * selectingInteractor.attachTransform.rotation;
73      }
74
75      public void OnSecondHandRelease(XRBaseInteractor interactor)
76      {
77          Debug.Log("SECOND HAND RELEASE");
78          secondInteractor = null;
79      }
80
81      [System.Obsolete]
82      protected override void OnSelectEntered(XRBaseInteractor
          interactor)
83      {
84          Debug.Log("First Grab Enter");
85          base.OnSelectEntered(interactor);
86          attachInitialRotation =
              interactor.attachTransform.localRotation;
87      }
88
89      [System.Obsolete]
90      protected override void OnSelectExited(XRBaseInteractor interactor)
91      {
```

```
92          Debug.Log("First Grab Exit");
93          base.OnSelectExited(interactor);
94          secondInteractor = null;
95          interactor.attachTransform.localRotation =
              attachInitialRotation;
96      }
97
98      [System.Obsolete]
99      public override bool IsSelectableBy(XRBaseInteractor interactor)
100     {
101         bool isalreadygrabbed = selectingInteractor && !
              interactor.Equals(selectingInteractor);
102         return base.IsSelectableBy(interactor) && !isalreadygrabbed;
103     }
104 }
105
```

Appendix B – BoatTilt.cs

C:\Users\nitul\The Roman Fisherman\Assets\BoatTilt.cs 1

```
 1 using System.Collections;
 2 using System.Collections.Generic;
 3 using UnityEngine;
 4
 5 public class BoatTilt : MonoBehaviour
 6 {
 7     public GameObject boat;
 8     public GameObject rotatable;
 9     public GameObject tiltAngleR;
10     public GameObject tiltAngleL;
11     public GameObject boatTiltAngleR;
12     public GameObject boatTiltAngleL;
13     public float boatTiltSpeed = 30f;
14     public float camTiltSpeed = 50f;
15     public GameObject boatFollower;
16     private bool inTriggerZone;
17
18
19
20     // Update is called once per frame
21     void Update()
22     {
23         inTriggerZone = false;
24         if (inTriggerZone)
25         {
26             boat.transform.rotation = Quaternion.Lerp
                 (boat.transform.rotation,
                 boatFollower.transform.rotation, Time.deltaTime *
                 boatTiltSpeed);
27             rotatable.transform.rotation = Quaternion.Lerp
                 (rotatable.transform.rotation,
                 boatFollower.transform.rotation, Time.deltaTime *
                 camTiltSpeed);
28         }
29     }
30
31
32     private void OnTriggerEnter(Collider col0)
33     {
34         if (col0.gameObject.tag == "wobbleColliderR")
35         {
36             inTriggerZone = true;
37
38             boat.transform.rotation = Quaternion.Lerp
                 (boat.transform.rotation,
                 boatTiltAngleR.transform.rotation, Time.deltaTime *
                 boatTiltSpeed);
39
40             rotatable.transform.rotation = Quaternion.Lerp
                 (rotatable.transform.rotation,
                 tiltAngleL.transform.rotation, Time.deltaTime *
                 camTiltSpeed);
41         }
42         else if (col0.gameObject.tag == "wobbleColliderL")
43         {
44             inTriggerZone = true;
```

```
45
46              boat.transform.rotation = Quaternion.Lerp
                  (boat.transform.rotation,
                  boatTiltAngleL.transform.rotation, Time.deltaTime *
                  boatTiltSpeed);
47
48              rotatable.transform.rotation = Quaternion.Lerp
                  (rotatable.transform.rotation,
                  tiltAngleR.transform.rotation, Time.deltaTime *
                  camTiltSpeed);
49          }
50      }
51      private void OnTriggerStay(Collider col1)
52      {
53          if (col1.gameObject.tag == "wobbleColliderR")
54          {
55              inTriggerZone = true;
56
57              boat.transform.rotation = Quaternion.Lerp
                  (boat.transform.rotation,
                  boatTiltAngleR.transform.rotation, Time.deltaTime *
                  boatTiltSpeed);
58
59              rotatable.transform.rotation = Quaternion.Lerp
                  (rotatable.transform.rotation,
                  tiltAngleL.transform.rotation, Time.deltaTime *
                  camTiltSpeed);
60          }
61          else if (col1.gameObject.tag == "wobbleColliderL")
62          {
63              inTriggerZone = true;
64
65              boat.transform.rotation = Quaternion.Lerp
                  (boat.transform.rotation,
                  boatTiltAngleL.transform.rotation, Time.deltaTime *
                  boatTiltSpeed);
66
67              rotatable.transform.rotation = Quaternion.Lerp
                  (rotatable.transform.rotation,
                  tiltAngleR.transform.rotation, Time.deltaTime *
                  camTiltSpeed);
68          }
69      }
70
71
72  }
73
```

Appendix C – SineMovement.cs

C:\Users\nitul\The Roman Fisherman\Assets\SineMovement.cs 1

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class SineMovement : MonoBehaviour // object script is attached to will oscillate up and down
6  {
7      public float amplitude;
8      public float frequency;
9      Vector3 originalPosition;
10
11     // Start is called before the first frame update
12     void Start()
13     {
14         originalPosition = transform.position;
15     }
16
17     // Update is called once per frame
18     void Update()
19     {
20         transform.position = new Vector3(originalPosition.x, Mathf.Sin(Time.time * frequency) * amplitude + originalPosition.y, originalPosition.z);
21         // y component of transform follows a sine wave, so that object oscillates up and down, x and z component unchanged
22     }
23 }
24
25
26
```

Appendix D – FishStick.cs

C:\Users\nitul\The Roman Fisherman\Assets\FishStick.cs 1

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class FishStick : MonoBehaviour
6  {
7      public SineMovement SineMovement;
8      public Rigidbody fish;
9      public Transform basketDropPoint;
10
11
12
13     // Start is called before the first frame update
14     private void Start()
15     {
16         SineMovement.enabled = true;
17     }
18
19
20
21     private void OnTriggerStay(Collider col1)
22     {
23         if (col1.gameObject.tag == "tridentTipCollider") // detecting collision between fish and the tip of the trident/spear
24         {
25             SineMovement.enabled = false; //stop the fish from moving when the tip hits the fish
26             transform.position = col1.transform.position; // position of fish is now at the position of the trident tip
27             StartCoroutine(putInBasket());
28
29         }
30     }
31
32     IEnumerator putInBasket() // to put the caught fish in the basket
33     {
34         yield return new WaitForSeconds(1); // fish to automatically drop into basket after 1 second
35         transform.SetParent(basketDropPoint.transform); // make fish a child of the basket drop point so that the fish drops relative to the boat's movement
36         transform.position = basketDropPoint.position;
37         fish.velocity = Vector3.zero; // stop fish from moving so that it will drop straight down into the basket
38         fish.useGravity = true;  // turn gravity on for fish so that it drops into the basket
39         fish.isKinematic = false;
40
41     }
42 }
43
```

Appendix E – PathFollower.cs

...ssets\PathCreator\Examples\Scripts\PathFollower.cs 1

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  namespace PathCreation.Examples
6  {
7      // Moves along a path at constant speed.
8      // Depending on the end of path instruction, will either loop,
         reverse, or stop at the end of the path.
9      public class PathFollower : MonoBehaviour
10     {
11         public float amplitude;
12         public float frequency;
13
14         public PathCreator pathCreator;
15         public EndOfPathInstruction endOfPathInstruction;
16         public float speed = 5;
17         float distanceTravelled;
18
19         void Start() {
20             //StartCoroutine(gameStart());
21
22             if (pathCreator != null)
23             {
24                 // Subscribed to the pathUpdated event so that we're
                     notified if the path changes during the game
25                 pathCreator.pathUpdated += OnPathChanged;
26             }
27         }
28
29         void Update()
30         {
31             if (pathCreator != null)
32             {
33                 distanceTravelled += speed * Time.deltaTime;
34                 transform.position =
                     pathCreator.path.GetPointAtDistance
                     (distanceTravelled, endOfPathInstruction);
35                 transform.position = new Vector3
                     (transform.position.x, Mathf.Sin(Time.time *
                     frequency) * amplitude + transform.position.y,
                     transform.position.z);
36
37                 transform.rotation =
                     pathCreator.path.GetRotationAtDistance
                     (distanceTravelled, endOfPathInstruction);
38             }
39         }
40
41         // If the path changes during the game, update the distance
             travelled so that the follower's position on the new path
42         // is as close as possible to its position on the old path
43         void OnPathChanged() {
44             distanceTravelled =
                   pathCreator.path.GetClosestDistanceAlongPath
                   (transform.position);
```

```
45          }
46
47          //IEnumerator gameStart()
48          //{
49              //yield return new WaitForSeconds(5);
50          //}
51
52      }
53  }
```

Appendix F – FishToHook.cs

C:\Users\nitul\The Roman Fisherman\Assets\FishToHook.cs 1

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class FishToHook : MonoBehaviour
6  {
7      public Transform hookTransform;
8      public Rigidbody fish;
9      public Transform basketDropPoint;
10
11     private void OnTriggerStay(Collider col1)
12     {
13         if (col1.gameObject.tag == "hookTransform")
14         {
15             transform.position = col1.transform.position;
16             StartCoroutine(putInBasket());
17         }
18     }
19
20     IEnumerator putInBasket()
21     {
22         yield return new WaitForSeconds(7);
23         transform.SetParent(basketDropPoint.transform);
24         transform.position = basketDropPoint.position;
25         fish.velocity = Vector3.zero;
26         fish.useGravity = true;
27         fish.isKinematic = false;
28     }
29 }
30
```