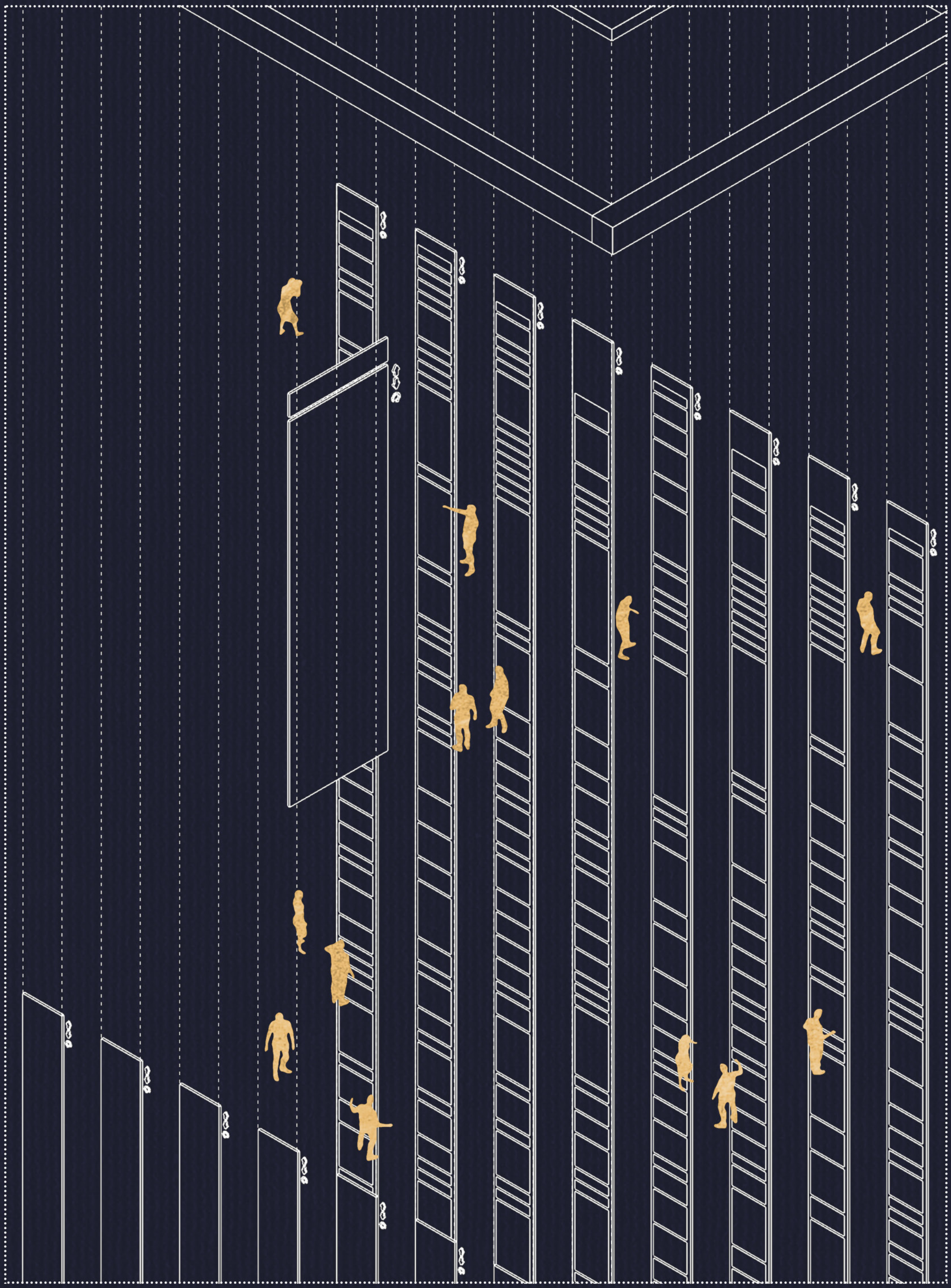

POST.SPACES

APPENDIX II: SYSTEMS DOCUMENTATION

1 Code Documentation

The Unity platform means that much of the design and programming work takes place across various objects, variables, and linkages, not to mention plugin support with other libraries and assets. Much of the programme structure is difficult to document due to the component hierarchies and connections between the various game objects and prefab instances. However, these are the bulk of the raw scripts that have been manually coded to support the VR environment. Note that due to time limitations they have not been fully cleaned, organised and refactored; this appendix exists for reference and as a proof of work.
Due to the several changes in direction over the course of the year, there have been several different systems and custom scripts that have since been scrapped or superseded; those scripts are not included here to avoid more confusion. All scripts here are active in the final release, but there might be some minor ones which I have left out.

Contents

2 Network Scripts

Network management has been built using the Photon suite of network tools, allowing both object and variable synchronization as well as an additional plugin that works together for voice communication management. Most operations are performed client-side and the relevant data regarding synced objects, players, etc are sent over the network and objects are moved or instantiated on the other cliient.

2.1 Network Manager

The Network Manager handles most of the general network syncing operations, such as joining/leaving the network and updating player stats (i.e. which subreddit they are currently in).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
using ExitGames.Client.Photon;
using UnityEngine.XR.Interaction.Toolkit;



public class NetworkManager : MonoBehaviourPunCallbacks
{
    public GameObject rig;
    public RedditController redditController;
    private string localSubreddit { get; set; }
    private ExitGames.Client.Photon.Hashtable playerProperties;
    private Dictionary<Player, NetworkAvatar> playerAvatarDict = new
        Dictionary<Player, NetworkAvatar>();
    // Start is called before the first frame update
    void Start()
    {
        rig = GameObject.Find("XR Rig");
        playerProperties = PhotonNetwork.LocalPlayer.CustomProperties;
        ConnectToServer();
    }

    void ConnectToServer()
    {
        PhotonNetwork.ConnectUsingSettings();
        Debug.Log("Try Connect To Server...");
    }

    public override void OnConnectedToMaster()
    {
        Debug.Log("Connected To Server.");

        base.OnConnectedToMaster();
        RoomOptions roomOptions = new RoomOptions();
        roomOptions.MaxPlayers = 16;
        roomOptions.IsVisible = true;
        roomOptions.IsOpen = true;

        PhotonNetwork.JoinOrCreateRoom("all", roomOptions, TypedLobby.
            Default);
    }

    public override void OnJoinedRoom()
    {
        Debug.Log("Joined a Room: " + PhotonNetwork.CurrentRoom);
        base.OnJoinedRoom();
    }

    public override void OnPlayerEnteredRoom(Player newPlayer)
    {
        Debug.Log("A new player joined the room.");
        base.OnPlayerEnteredRoom(newPlayer);
    }

    public override void OnDisconnected(DisconnectCause cause)
    {
        Debug.Log("PUN DISCONNECTED: " + cause);
        Invoke("ConnectToServer", 20f);
    }


    public override void OnPlayerPropertiesUpdate (Player targetPlayer,
        ExitGames.Client.Photon.Hashtable changedProps)
    {
        Debug.Log("player properties update: " + targetPlayer + " - " +
            changedProps);

        if(changedProps.ContainsKey("subreddit"))
        {
            if(targetPlayer != PhotonNetwork.LocalPlayer)
            {
                if(changedProps["subreddit"].ToString() == localSubreddit)
                {
                    playerAvatarDict[targetPlayer].EnableSelf();
                    Debug.Log(targetPlayer + " has entered your subreddit.");
                }
                else
                {
                    playerAvatarDict[targetPlayer].DisableSelf();
                    Debug.Log(targetPlayer + " has left your subreddit.");
                }
            }
        }
    }

    public void SetPlayerSubreddit(string subreddit)
    {
        localSubreddit = subreddit;
        rig.GetComponent<XRRig>().MatchRigUpRigForward(new Vector3(0, 3, 0), new Vector3(1, 0, 0));
        rig.transform.position = new Vector3(0, -3, 0);

        //Set network player property
        if (!playerProperties.ContainsKey("subreddit"))
        {
            playerProperties.Add("subreddit", subreddit);
            Debug.Log("Property added");
        }
        playerProperties["subreddit"] = subreddit;
        PhotonNetwork.LocalPlayer.SetCustomProperties(playerProperties);

        //Initialise posts
        redditController.Initialise(subreddit);
    }

    public void AddPlayerAvatar(Player player, NetworkAvatar avatar)
    {
        if(!playerAvatarDict.ContainsKey(player))
        {
            playerAvatarDict.Add(player, avatar);
            Debug.Log("Added " + playerAvatarDict[player] + " to list of player avatars.");
        }
    }

```

```
}
```

2.2 Network Player Spawner

The Network Player Spawner is responsible for spawning in player avatars.

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

public class NetworkPlayerSpawner : MonoBehaviourPunCallbacks
{
    private GameObject spawnedPlayerPrefab;
    private Color shirtColor;
    private Color pantsColor;
    private object[] avatarInitData = new object[6];

    void Start()
    {
        //Randomise clothes
        //shirt color
        avatarInitData[0] = Random.Range(80, 230);
        avatarInitData[1] = Random.Range(80, 230);
        avatarInitData[2] = Random.Range(80, 230);

        //pants color
        avatarInitData[3] = Random.Range(20, 70);
        avatarInitData[4] = Random.Range(20, 70);
        avatarInitData[5] = Random.Range(20, 70);


    }
    public override void OnJoinedRoom()
    {
        base.OnJoinedRoom();
        spawnedPlayerPrefab = PhotonNetwork.Instantiate("Network Avatar",
            transform.position, transform.rotation, 0, avatarInitData);
    }

    public override void OnLeftRoom()
    {
        base.OnLeftRoom();
        PhotonNetwork.Destroy(spawnedPlayerPrefab);
    }
}
```

2.3 Network Avatar

The Network Avatar script is an important script that controls the positioning, animation and behaviour of each player avatar. These avatars are spawned client-side for each player in the server and are synced over the Photon network.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR;
using Photon.Pun;
using Photon.Realtime;
using Photon.Voice.Unity;

public class NetworkAvatar : MonoBehaviour, IPunInstantiateMagicCallback
{
    public Transform head;
    public List<Transform> meshes;
    public Transform character;
    public bool hideSelf = true;
    public bool hideHead = true;
    public float cameraFrontOffset;
    public GameObject speaker;
    private Vector3 baseVector;
    private bool parented = false;
    private GameObject rig;
    private Animator animator;
    private NetworkManager networkManager;
    //private int interval = 1;
    //private float animUpdateTime = 0;

    private PhotonView photonView;

    // Start is called before the first frame update
    void Awake()
    {
        photonView = GetComponent<PhotonView>();
        rig = GameObject.Find("XR Rig");
        networkManager = GameObject.Find("NetworkManager").GetComponent<
            NetworkManager>();
        animator = GetComponentInChildren<Animator>();
        if(photonView.IsMine) MapCharacterPosition(character, XRNode.Head);
    }

    public void OnPhotonInstantiate(PhotonMessageInfo info)
    {
        object[] data = info.photonView.InstantiationData;
        if (data != null)
        {
            SetColor(new Color((int)data[0]/255f, (int)data[1]/255f, (int)
                data[2]/255f), new Color((int)data[3]/255f, (int)data[4]/255
                f, (int)data[5]/255f));
        }
        networkManager.AddPlayerAvatar(photonView.Owner, this);
    }

    // Update is called once per frame
    void Update()
    {
        if (photonView.IsMine)
        {
            if (hideSelf)
            {
                foreach (Transform mesh in meshes)
                {
                    mesh.gameObject.SetActive(false);
                }
            }
            if (hideHead || hideSelf)
            {
                head.gameObject.SetActive(false);
            }

            if(rig.GetComponent<MovementProvider>().inCenter && rig)
            {
                MapCharacterPosition(character, XRNode.Head);
            }
            //else if(parented) MapLocalPosition(character, XRNode.Head);
            Animate();
        }
    }

    void MapPosition(Transform target, XRNode node)
    {
        InputDevices.GetDeviceAtXRNode(node).TryGetFeatureValue(CommonUsages
            .devicePosition, out Vector3 position);
        InputDevices.GetDeviceAtXRNode(node).TryGetFeatureValue(CommonUsages.deviceRotation, out Quaternion rotation);

        target.position = position;
        target.rotation = rotation;
    }

    void MapCharacterPosition(Transform characterModel, XRNode node)
    {
        InputDevices.GetDeviceAtXRNode(node).TryGetFeatureValue(CommonUsages.devicePosition, out Vector3 position);
        InputDevices.GetDeviceAtXRNode(node).TryGetFeatureValue(CommonUsages.deviceRotation, out Quaternion rotation);

        Transform rigTransform = rig.GetComponent<Transform>();

        if(rigTransform.rotation.eulerAngles.x == 0)
        {
            Quaternion snapTurnOffset = rigTransform.rotation;
            Vector3 characterControllerOffset = rigTransform.position;
            //Vector3 headPosition = new Vector3(characterControllerOffset.x, characterControllerOffset.y, characterControllerOffset.z);
            Vector3 headPosition = new Vector3(characterControllerOffset.x + (snapTurnOffset * position).x, characterControllerOffset.y, characterControllerOffset.z + (snapTurnOffset * position).z);
            //Vector3 headRotation = new Vector3(0, rotation.eulerAngles.y + (snapTurnOffset.eulerAngles.y), 0);
            Vector3 headRotation = new Vector3(0, (snapTurnOffset.eulerAngles.y), 0);
            Vector3 headPositionOffset = Quaternion.Euler(headRotation) * new Vector3(cameraFrontOffset, 0, 0);
            //Debug.Log(headPositionOffset);

            characterModel.position = headPosition - headPositionOffset;
            characterModel.rotation = Quaternion.LookRotation(rigTransform.right, rigTransform.up);
            //Debug.Log(Quaternion.LookRotation(rigTransform.up, rigTransform.forward));
            //characterModel.rotation = Quaternion.Euler(headRotation);

            //Parent to XR rig and stop moving dynamically
            transform.SetParent(rig.transform);
            baseVector = characterModel.localPosition;
            parented = true;
            //Invoke("Parented", 5f);
        }
    }

    void MapLocalPosition(Transform characterModel, XRNode node)
    {
        InputDevices.GetDeviceAtXRNode(node).TryGetFeatureValue(CommonUsages
            .devicePosition, out Vector3 position);
        Transform rigTransform = rig.GetComponent<Transform>();

        Vector3 headposition = new Vector3(position.x, 0, position.z);

        characterModel.localPosition = baseVector + headposition;
    }

    private void Parented()
    {
        parented = true;
    }

    public void SetColor(Color shirt, Color pants)
    {
        transform.Find("Torso Male Short Sleeves").GetComponent<
            SkinnedMeshRenderer>().material.SetColor("_BaseColor", shirt);
        transform.Find("Pants Male").GetComponent<SkinnedMeshRenderer>().
            material.SetColor("_BaseColor", pants);
    }

    private void Animate()
    {
        // Blend between walk/run
        //float blend = rig.GetComponent<CharacterController>().velocity.
            magnitude;
        //animator.SetFloat("Move", blend);
    }

    public void MuteSelf()
    {
        speaker.SetActive(false);
    }

    public void UnmuteSelf()
    {
        speaker.SetActive(true);
    }

    public void DisableSelf()
    {
        foreach (Transform mesh in meshes)
        {
            mesh.gameObject.SetActive(false);
        }
        head.gameObject.SetActive(false);
        //MuteSelf();
    }

    public void EnableSelf()
    {
        foreach (Transform mesh in meshes)
        {
            mesh.gameObject.SetActive(true);
        }
        head.gameObject.SetActive(true);
        //UnmuteSelf();
    }

    public Player GetOwner()
    {
        return photonView.Owner;
    }

}
```

2.4 Avatar Random Customiser

Small script to handle randomisation of avatar shirt colors, so players can tell each other apart.

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

public class AvatarRandomCustomiser : MonoBehaviour
{

    // Start is called before the first frame update
    void Start()
    {


    }

    // Update is called once per frame
    void Update()
    {

    }

    public void SetColor(Color shirt, Color pants)
    {
        transform.Find("Torso Male Short Sleeves").GetComponent<
            SkinnedMeshRenderer>().material.SetColor("_BaseColor", shirt);
        transform.Find("Pants Male").GetComponent<SkinnedMeshRenderer>().
            material.SetColor("_BaseColor", pants);
    }
}
```

2.5 Grabbable Photon Ownership

This script handles the transfer of ownership when different players want to move synced objects.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR;
using UnityEngine.XR.Interaction.Toolkit;
using Photon.Realtime;
using Photon.Pun;

[RequireComponent( typeof( PhotonView ) )]
public class GrabbablePhotonOwnership : MonoBehaviourPun
{
    private XRGrabInteractable interactable = null;
    private void Awake()
    {
        interactable = GetComponent<XRGrabInteractable>();
    }
    private void OnEnable()
    {
        interactable.onSelectEnter.AddListener(RequestOwnership);
    }
    private void RequestOwnership(XRBaseInteractor interactor)
    {
        if( this.photonView.Owner == PhotonNetwork.LocalPlayer )
        {
            Debug.Log( "Not requesting ownership. Already mine." );
            return;
        }

        this.photonView.RequestOwnership();
        Debug.Log( "Requesting ownership" );
    }

}
```

3 Player and XR Scripts

The main behaviour for VR/XR comes from the Unity XR plugin and is configured via existing script components and options. However, custom scripts are still needed to support all the additional behaviours available to the player as well as some custom functions for controls and teleportation not supported by the base package.

3.1 Player Controller

Simple script that sets the player's base position, should reallly be merged into another script because the player rig has lots of different scripts attached to it.

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


public class PlayerController : MonoBehaviour
{
    private Vector3 startPosition;
    public int numVoiceRooms = 0;


    // Start is called before the first frame update
    void Start()
    {
        startPosition = transform.position;
    }

    // Update is called once per frame
    void Update()
    {

    }

    private void ResetPlayerPosition()
    {
        transform.position = startPosition;
    }

    public void TeleportPlayer(Vector3 target)
    {

    }

}
```

3.2 Movement Provider

Main script controlling player movement, gravity, etc.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR;
using UnityEngine.XR.Interaction.Toolkit;

public class MovementProvider : LocomotionProvider
{
    public float speed = 1.0f;
    public float gravityMultiplier = 1.0f;
    public bool gravity = true;
    public List<XRController> controllers = null;
    public TeleportationProvider teleportationProvider;
    private CharacterController characterController = null;
    private GameObject head = null;
    private bool resetClick = true;
    public bool relativeGravity = true;
    public bool inCenter = true;


    //CUSTOM GRAVITY VARIABLES
    float distanceToGround;
    Vector3 Groundnormal;
    bool OnGround = false;

    protected override void Awake()
    {
        characterController = GetComponent<CharacterController>();
        head = GetComponent<XRRig>().cameraGameObject;
    }

    // Start is called before the first frame update
    private void Start()
    {
        PositionController();
    }

    private void Update()
    {
        // Ground Control (GRAVITY SCRIPT NOT WRITTEN BY ME)
        RaycastHit hit = new RaycastHit();
        if (Physics.Raycast(transform.position, -transform.up, out hit, 10))
        {

            distanceToGround = hit.distance;
            Groundnormal = hit.normal;

            if (distanceToGround <= 0.2f)
            {
                OnGround = true;
            }
            else
            {
                OnGround = false;
            }


        }
    }

    // Update is called once per frame
    private void FixedUpdate()
    {
        PositionController();
        if (!relativeGravity)
        {
            if (gravity)
                ApplyGravity();
            else if (characterController.transform.position.y <= 9)
                gravity = true;
        }
        else
        {
            if(inCenter)
                ApplyGravity();
            //if relative gravity is on
            //ApplyRelativeGravity();
        }
    }

    public void PositionController()
    {
        //Get the head in local, playspace ground
        float headHeight = Mathf.Clamp(head.transform.localPosition.y, 0.5f,
             2);
        characterController.height = headHeight;

        //Cut in half, add skin
        Vector3 newCenter = Vector3.zero;
        newCenter.y = characterController.height / 2;
        newCenter.y += characterController.skinWidth;

        //Move the capsule in local space as well
        newCenter.x = head.transform.localPosition.x;
        newCenter.z = head.transform.localPosition.z;

        //Apply
        characterController.center = newCenter;
    }

    private void CheckForInput()
    {

    }

    public void StartMove(Vector2 position)
    {
        //Add forward vector
        Vector3 direction = new Vector3(position.x, 0, position.y);
        Vector3 headRotation = head.transform.eulerAngles; //new Vector3(0,
            head.transform.eulerAngles.y, 0);
        Vector3 bodyRotation = transform.up;

        direction = head.transform.rotation * direction; //raw direction
            pointing in direction player head
        direction = direction - bodyRotation * Vector3.Dot(direction,
            bodyRotation); //direction constrained to XY plane of player, by
             subtracting scalar projection of normal

        //Apply speed and move
        Vector3 movement = direction.normalized * speed;
        characterController.Move(movement * Time.deltaTime);
        XR_TestVariables.XRMoveCount += Vector3.Magnitude(movement * Time.deltaTime);
    }

    public void StartMove(Vector3 position)
    {
        //Add forward vector
        Vector3 direction = position;
        Vector3 headRotation = head.transform.eulerAngles; //new Vector3(0, head.transform.eulerAngles.y, 0);
        Vector3 bodyRotation = transform.up;

        direction = head.transform.rotation * direction; //raw direction pointing in direction player head
        //direction = direction - bodyRotation * Vector3.Dot(direction, bodyRotation); //direction constrained to XY plane of player, by subtracting scalar projection of normal

        //Apply speed and move
        Vector3 movement = direction.normalized * speed;
        characterController.Move(movement);
        XR_TestVariables.XRMoveCount += Vector3.Magnitude(movement * Time.deltaTime);
    }

    private void ApplyGravity()
    {
        Vector3 gravity = new Vector3(0, Physics.gravity.y * gravityMultiplier, 0);
        gravity.y *= Time.deltaTime;

        characterController.Move(gravity);
    }

    private void ApplyRelativeGravity()
    {
        //Vector3 sphericalGravDirection = new Vector3( transform.position.normalized.z, 0, transform.position.normalized.y);
        Vector3 relativeGravDirection = -transform.up;
        Vector3 gravityTime = -relativeGravDirection * Physics.gravity.y * gravityMultiplier * Time.deltaTime;

        /*
        if (OnGround == false)
        {
            characterController.Move(gravityTime);
        }
        */

        characterController.Move(gravityTime);

        //Quaternion toRotation = Quaternion.FromToRotation(transform.up,
            Groundnormal) * transform.rotation;
        //transform.rotation = toRotation;

    }

    private void HighJump(bool jump)
    {
        Vector3 highJump = new Vector3(0, 11 - characterController.transform
            .position.y, 0);

        if (jump)
        {
            characterController.Move(highJump);
            gravity = false;
        }
        else
        {
            characterController.Move(-highJump);
            gravity = true;
        }
    }

    public void VertMove(float magnitude)
    {
        //Debug.Log("VertMove " + magnitude);
        characterController.Move(transform.up * magnitude * Time.deltaTime);
        //var moveVector3 = new Vector3(0, magnitude, 0);
        //StartMove(moveVector3);
    }

    void OnTriggerEnter(Collider collider)
    {
        if(collider.gameObject.name == "CenterSphere")
        {
            inCenter = true;
            Debug.Log("in center!");
        }
    }

    void OnTriggerExit(Collider collider)
    {
        if(collider.gameObject.name == "CenterSphere")
        {
            inCenter = false;
            Debug.Log("out of center!");
        }
    }
```

```

}
```

3.3 XR Controller Manager

Custom script I wrote that handles higher-level operations with the XR controllers, allowing behaviour like toggling raycast lines instead of having them on permanently.

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

using UnityEngine.XR.Interaction.Toolkit.UI;
using UnityEngine.EventSystems;

namespace UnityEngine.XR.Interaction.Toolkit
{
    public class XRControllerManager : MonoBehaviour
    {
        public XRController LeftController;
        public XRController RightController;
        public XRController TeleportController;
        public List<XRController> controllers;
        private MovementProvider movementProvider;
        private XRRayInteractor TeleportInteractor;
        private XRInteractorLineVisual TeleportLine;
        private XRRayInteractor GrabRayInteractor;
        private XRInteractorLineVisual GrabRayLine;
        private GameObject rig;
        private GameObject cameraOffset;
        private GameObject teleportReticle;
        private GameObject teleportPlane;
        public bool TeleportMode = true;
        //public Dictionary<string, GameObject> spawnPrefabsList = new
            Dictionary<string, GameObject>();


        // Start is called before the first frame update
        void Start()
        {
            rig = GameObject.Find("XR Rig");
            movementProvider = transform.GetComponent<MovementProvider>();
            cameraOffset = rig.transform.Find("Camera Offset").gameObject;
            teleportReticle = cameraOffset.transform.Find("GroundTeleportReticle").gameObject;
            teleportPlane = cameraOffset.transform.Find("MagicTeleportPlane").gameObject;

            TeleportInteractor = TeleportController.GetComponent<XRRayInteractor>();
            TeleportLine = TeleportController.GetComponent<XRInteractorLineVisual>();
            GrabRayInteractor = RightController.GetComponent<XRRayInteractor>();
            GrabRayLine = RightController.GetComponent<XRInteractorLineVisual>();

            CheckTeleportMode();

            //spawnPrefabsList.Add("anchorBush", Resources.Load("Blobs/BlobAnchorDynamic_Bush", typeof(GameObject)) as GameObject);
        }

        // Update is called once per frame
        void FixedUpdate()
        {
            CheckForInput();
            CheckForTeleport();
            CheckInteractorRay();
        }

        private bool resetClick, resetClick2 = true;

        private void CheckForInput()
        {
            if (TeleportController.inputDevice.TryGetFeatureValue(CommonUsages.secondary2DAxisClick, out bool click))
            {
                if (click && resetClick)
                {
                    TeleportMode = !TeleportMode;
                    CheckTeleportMode();
                    resetClick = false;
                }
                else if (!click)
                {
                    resetClick = true;
                }
            }

            if (TeleportController.inputDevice.TryGetFeatureValue(CommonUsages.primary2DAxisClick, out bool click2))
            {
                if (click2 && resetClick2)
                {
                    TeleportMode = !TeleportMode;
                    CheckTeleportMode();
                    resetClick2 = false;
                }
                else if (!click2)
                {
                    resetClick2 = true;
                }
            }

            /*
            foreach (XRController controller in controllers)
            {
                if (controller.enableInputActions)
                    CheckForMovement(controller.inputDevice);
            }
            */

            CheckForMovement(LeftController.inputDevice);
        }

        private void CheckForMovement(InputDevice device)
        {
            if (device.TryGetFeatureValue(CommonUsages.primary2DAxis, out
                Vector2 position))
            {
                //movementProvider.StartMove(position);
                //Debug.Log(position);
            }
            if (device.TryGetFeatureValue(CommonUsages.secondary2DAxis, out
                Vector2 position2) && position2.magnitude > 0.2f) //check >
                0.2f for deadzone
            {
                movementProvider.StartMove(position2);
                //Debug.Log(position2.magnitude);
            }
            /*
            //"High Jump" float function
            if (device.TryGetFeatureValue(CommonUsages.secondaryButton, out
                bool click))
            {
                if (click && resetClick)
                {
                    if (characterController.transform.position.y <= 12)
                    {
                        HighJump(true);
                        resetClick = false;
                    }
                    else
                    {
                        HighJump(false);
                        resetClick = false;
                    }
                }
                else if (!click && !resetClick)
                    resetClick = true;
            }
            */
        }

        private void CheckForTeleport()
        {
            if (TeleportController.inputDevice.TryGetFeatureValue(CommonUsages.triggerButton, out bool triggerValue) && triggerValue)
            {
                TeleportInteractor.enabled = true;
                TeleportLine.enabled = true;
                if (TeleportMode == true)
                {
                    teleportReticle.SetActive(true);
                }
                else teleportReticle.SetActive(false);
            }
            else if (!triggerValue && TeleportInteractor.enabled)
            {
                TeleportInteractor.enabled = false;
                TeleportLine.enabled = false;
                teleportReticle.SetActive(false);
            }
        }

        private void CheckInteractorRay()
        {
            if (RightController.inputDevice.TryGetFeatureValue(CommonUsages.triggerButton, out bool triggerValue) && triggerValue)
            {
                GrabRayInteractor.enabled = true;
                GrabRayLine.enabled = true;
            }
            else if (!triggerValue && GrabRayInteractor.enabled)
            {
                GrabRayInteractor.enabled = false;
                GrabRayLine.enabled = false;
            }
        }

        private void CheckTeleportMode()
        {
            //Teleport mode 1 i.e. spatial teleport projectile curve
            if (TeleportMode == true)
            {
                SetTeleportProjectile();
            }
            //Teleport mode 2 i.e. straightline select
            else if (TeleportMode == false)
            {
                SetTeleportRaycast();
            }
        }

        private void SetTeleportProjectile()
        {
            TeleportInteractor.lineType = XRRayInteractor.LineType.ProjectileCurve;
            //TeleportLine.AttachCustomReticle(teleportReticle);
            //teleportPlane.SetActive(true);
            teleportReticle.SetActive(true);
        }

        private void SetTeleportRaycast()
        {
            TeleportInteractor.lineType = XRRayInteractor.LineType.StraightLine;
            //TeleportLine.RemoveCustomReticle();
            //teleportPlane.SetActive(false);
            teleportReticle.SetActive(false);
        }
    }
}
```

3.4 Teleportation Anchored Area

Another custom XR script that modifies teleportation behaviour, allowing the player to aim at any part of the collision object but always teleport to a fixed point.

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


namespace UnityEngine.XR.Interaction.Toolkit
{
    public class TeleportationAnchoredArea : BaseTeleportationInteractable
    {
        public GameObject TeleportationAnchor;

        protected override bool GenerateTeleportRequest(XRBaseInteractor
            interactor, RaycastHit raycastHit, ref TeleportRequest
            teleportRequest)
        {
            teleportRequest.destinationPosition = TeleportationAnchor.
                transform.position;
            teleportRequest.destinationUpVector = TeleportationAnchor.
                transform.up; // use the area transform for data.
            teleportRequest.destinationForwardVector = transform.forward;
            teleportRequest.destinationRotation = TeleportationAnchor.
                transform.rotation;

            GameObject.Find("XR Rig").GetComponent<MovementProvider>().
                PositionController();
            return true;
        }

    }
}
```

3.5 Comment Frame Teleporter

Script that handles the teleporting to a specific spot on the comment frame.

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

namespace UnityEngine.XR.Interaction.Toolkit
{
    public class CommentFrameTeleporter : BaseTeleportationInteractable
    {
        private float teleportBufferDistance = 8.0f;
        protected override bool GenerateTeleportRequest(XRBaseInteractor
            interactor, RaycastHit raycastHit, ref TeleportRequest
            teleportRequest)
        {
            teleportRequest.destinationPosition = raycastHit.point - (
                transform.forward.normalized * teleportBufferDistance) - (
                transform.up.normalized * 1);
            teleportRequest.destinationUpVector = transform.up; // use the
                area transform for data.
            teleportRequest.destinationForwardVector = -transform.right;
            teleportRequest.destinationRotation = Quaternion.identity;//
                transform.rotation;
            return true;
        }

    }
}
```

3.6 Custom Teleportation Provider

Custom script instance copied from the default teleportation provider that adds some additional functions

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

namespace UnityEngine.XR.Interaction.Toolkit
{

    public class CustomTeleportationProvider : LocomotionProvider
    {

        // the current teleportation request
        TeleportRequest m_CurrentRequest;
        // whether the current teleportation request is valid.
        bool m_ValidRequest = false;


        /// <summary>
        /// This function will queue a teleportation request within the
            provider.
        /// </summary>
        /// <param name="teleportRequest">The teleportation request</param>
        /// <returns>true if successful.</returns>
        public bool QueueTeleportRequest(TeleportRequest teleportRequest)
        {
            m_CurrentRequest = teleportRequest;
            m_ValidRequest = true;
            return true;
        }

        /// <summary>
        /// Update function for the Teleportation Provider
        /// </summary>
        private void Update()
        {
            if(m_ValidRequest && BeginLocomotion())
            {
                var xrRig = system.xrRig;
                if (xrRig != null)
                {
                    switch (m_CurrentRequest.matchOrientation)
                    {
                        case MatchOrientation.None:
                            xrRig.MatchRigUp(m_CurrentRequest.
                                destinationUpVector);
                            break;
                        case MatchOrientation.Camera:
                            xrRig.MatchRigUpCameraForward(m_CurrentRequest.
                                destinationUpVector, m_CurrentRequest.
                                destinationForwardVector);
                            break;
                        //case MatchOrientation.Rig:
                        //    xrRig.MatchRigUpRigForward(m_CurrentRequest.
                            destinationUpVector, m_CurrentRequest.
                            destinationForwardVector);
                        //    break;
                    }

                    Vector3 heightAdjustment = xrRig.rig.transform.up *
                        xrRig.cameraInRigSpaceHeight;

                    Vector3 cameraDestination = m_CurrentRequest.
                        destinationPosition + heightAdjustment;

                    xrRig.MoveCameraToWorldLocation(cameraDestination);
                }
                EndLocomotion();
                m_ValidRequest = false;
            }
        }
    }
}
```

4 Reddit General Scripts

Reddit data is loaded using the Reddit.NET API interface. Probably not the most ideal due to synchronous lag which I haven't been able to solve, but it is relatively easy to understand and implement.

4.1 Reddit Controller

The big, main component script that handles the reddit interface and spawns all the subsequent posts and geometry which then execute their own scripts.

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Reddit;
using Reddit.Controllers;
using Reddit.Controllers.EventArgs;
using TMPro;
using System.Linq;
using System.IO;

public class RedditController : MonoBehaviour
{
    public RedditClient r;
    //private string subredditTarget = "AskReddit";
    private GameObject testPrefab;
    private GameObject redPostPrefab;
    private GameObject textTemplate;
    private GameObject textTemplateColumn;
    private GameObject textTemplateComment;
    public GameObject torusTemplate;
    public GameObject subredditStar;
    public RedCentralSpaceMonobehaviour centralSpace;
    public GameObject subredditTitleMesh;
    public NetworkManager networkManager;
    public bool generatePosts = true;
    private string targetSubreddit;
    private string refreshToken;
    public List<GameObject> postObjects;
    private List<Post> allPostList;
    private List<(float, float)> postPropertiesList = new List<(float upvoteScale, float commentWeight)>();
    private List<string> subredditList;

    //PARAMETERS
    private float spawnRadius = 100.0f;
    private int numRings = 3;
    private int postsPerRing = 18;
    private int numPosts = 50;

    // Start is called before the first frame update
    void Start()
    {
        //testPrefab = Resources.Load("Blobs/BlobAnchorStatic_Block", typeof(GameObject)) as GameObject;

        redPostPrefab = Resources.Load("Reddit/RedPostPlatform", typeof(GameObject)) as GameObject;
        textTemplate = Resources.Load("Generic_Text", typeof(GameObject)) as GameObject;
        textTemplateColumn = Resources.Load("Generic_Text_Column", typeof(GameObject)) as GameObject;
        textTemplateComment = Resources.Load("Generic_Text_Column_Smaller", typeof(GameObject)) as GameObject;

        targetSubreddit = File.ReadAllText(Application.streamingAssetsPath + "/subreddit.txt");
        refreshToken = File.ReadAllText(Application.streamingAssetsPath + "/refreshtoken.txt");

        r = new RedditClient("DHawrgYdt3OhPg", refreshToken);
        Debug.Log("Username: " + r.Account.Me.Name);

        GenerateSubreddits();
        Initialise(targetSubreddit);
        networkManager.SetPlayerSubreddit(targetSubreddit);


        /*for (int i = 0; i < numRings + 1; i++)
        {
            GameObject ringDivider = Instantiate(torusTemplate, transform);
            ringDivider.transform.localPosition = new Vector3(0, i * 30f, 0);
        }*/
        //UnityEngine.Debug.Log(r.Account.Me);
    }

    public void Initialise(string subreddit = "")
    {
        DestroyGeometry();
        targetSubreddit = subreddit;
        centralSpace.Initialise(r.Subreddit(targetSubreddit));
        allPostList = GetTopPosts(targetSubreddit, numPosts);
        Debug.Log("Total posts:" + allPostList.Count);
        ProcessPostData();

        int oneRingCount = Mathf.FloorToInt(postsPerRing);

        //Calculate which posts in the ring first
        List<int> ringBreakpoints = new List<int>();
        ringBreakpoints.Add(0);
        float sumWeights = postPropertiesList.Sum(x => x.Item2 + 1);
        //Debug.Log("sum weights :" + sumWeights);

        float totalWeight = 0;
        for (int p = 0; p < allPostList.Count; p++)
        {
            if(totalWeight >= sumWeights/(numRings))
            {
                totalWeight = 0;
                ringBreakpoints.Add(p);
                //Debug.Log("breakpoint: " + p);
            }
            totalWeight += postPropertiesList[p].Item2 + 1;
        }
        if(ringBreakpoints.Count < numRings + 1)
        {
            ringBreakpoints.Add(allPostList.Count - 1);
        }
        //Debug.Log(ringBreakpoints.Count);

        for (int i = 0; i < numRings; i++)
        {
            //Debug.Log("Generating ring - " + ringBreakpoints[i] + "-" + ringBreakpoints[i+1]);
            if(generatePosts) GeneratePostGeometry(allPostList.GetRange(ringBreakpoints[i], ringBreakpoints[i+1]-ringBreakpoints[i]),
                                      postPropertiesList.GetRange(ringBreakpoints[i], ringBreakpoints[i+1]-ringBreakpoints[i]), 15f + (i * 30f));
            //Debug.Log("Ring " + i + " Generated");
            /*
            if ((i + 1) * postsPerRing <= postList.Count)
            {
                GeneratePostGeometry(postList.GetRange(i * postsPerRing, postsPerRing), 10f + (i * 30f));
            }
            */
        }
    }

    public void DestroyGeometry()
    {
        foreach (GameObject post in postObjects)
        {
            GameObject.Destroy(post);
        }
    }

    // Update is called once per frame
    void Update()
    {

    }

    public List<Post> GetTopPosts(string subreddit, int num)
    {
        return r.Subreddit(subreddit).Posts.Hot.GetRange(0, num);
    }

    public void GeneratePostGeometry(List<Post> postList, List<(float, float
        )> postProperties, float yoffset)
    {
        int numObjects = postList.Count;
        var postWeights = new List<float>();

        for (int i = 0; i < numObjects; i++)
        {
            //var weightCalc = Mathf.RoundToInt(Mathf.Clamp((postList[i].
                Listing.NumComments / 200f), 0f, 1f) * 5);
            postWeights.Add(1f + postProperties[i].Item2);
        }

        //postWeights = postList.Select(x => (5 + Mathf.RoundToInt(Mathf.
            Clamp((x.UpVotes / 40000f), 0f, 1f) * 5)) ).ToList<int>();
        float totalWeight = postWeights.Sum();
        //Debug.Log(totalWeight);

        var cumulativeAngle = 0f;

        for (int i = 0; i < numObjects; i++)
        {
            Post currentPost = postList[i];
            string postTitle = currentPost.Title;


            //Circle geometry math
            float circumference = 2 * Mathf.PI * spawnRadius;
            float postAngleShare = postWeights[i] / totalWeight;
            float angle = postAngleShare * 2 * Mathf.PI;
            int postLengthUnits = Mathf.FloorToInt(2 * spawnRadius * Mathf.
                Sin(angle / 2) / 4) - 1;

            //float angle = i * Mathf.PI * 2 / numObjects;
            cumulativeAngle += angle / 2;  //add half because you actually
                want the center of the angle
            float x = Mathf.Cos(cumulativeAngle) * spawnRadius;
            float z = Mathf.Sin(cumulativeAngle) * spawnRadius;
            Vector3 pos = transform.position + new Vector3(x, yoffset, z);
            float angleDegrees = -cumulativeAngle * Mathf.Rad2Deg;
            Quaternion rot = Quaternion.Euler(270, angleDegrees, 90);


            //Instantiate platform geometry
            GameObject currentPlatform = Instantiate(redPostPrefab, pos, rot
                );
            currentPlatform.GetComponent<RedPostMonobehaviour>().
                InitialisePost(currentPost, currentPost.Comments,
                postLengthUnits, postProperties[i].Item1);

            cumulativeAngle += angle / 2;  //add half again to get the
                actual cumulative angle


            postObjects.Add(currentPlatform);

        }
    }

    public void ProcessPostData()
    {
        postPropertiesList.Clear();
        for(int i = 0; i < allPostList.Count; i++)
        {
            Post tempPost = allPostList[i];
            int maxUpvotes = allPostList.Max(x => x.UpVotes);
            int maxComments = allPostList.Max(x => x.Listing.NumComments);

            float scaledUpvotes = (float)tempPost.UpVotes/(float)maxUpvotes;
            float scaledComments = tempPost.Listing.NumComments/(float)
                maxComments;

            scaledUpvotes = Mathf.Ceil(scaledUpvotes * 100) / 100f;

            postPropertiesList.Add((scaledUpvotes, scaledComments));
        }
    }

    public void GenerateSubreddits()
    {
        List<Subreddit> subredditList = r.GetSubreddits("popular", 100);
        int num = subredditList.Count;
        List<string> subredditNames = subredditList.Select(x => x.Name.ToString()).ToList();

        Vector3 center = new Vector3(0, 0, 0);
        for (int i = 0; i < num; i++)
        {
            //Vector sphere math to position stars at random point on constrained sphere
            float thetaAngle = Random.Range(5, 45);
            float phiAngle = Random.Range(0, 360);
            Quaternion rotationAngle = Quaternion.Euler(thetaAngle, phiAngle, 0);

            Vector3 pos = rotationAngle * new Vector3(0,2000,0);

            //Vector3 pos = Random.onUnitSphere * 2000.0f;

            Quaternion rot = Quaternion.FromToRotation(Vector3.up, center - pos);
            GameObject star = Instantiate(subredditStar, pos, rot);

            //Debug.Log(subredditList[i].Name);
            string subredditName = subredditList[i].Name;
            star.GetComponent<SubredditStarMonobehaviour>().InitialiseStar(subredditName);

        }

        GameObject allstar = Instantiate(subredditStar, new Vector3(0,2000,0), Quaternion.FromToRotation(Vector3.up, new Vector3(0,-2000,0)));
        allstar.GetComponent<SubredditStarMonobehaviour>().InitialiseStar("all");
    }

    public void GenerateOAuth()
    {
        //AuthTokenRetrieverLib authTokenRetrieverLib = new AuthTokenRetrieverLib(appId, appSecret, port);
    }
}
```

4.2 Reddit Central Space

Script controlling the behaviour of the central space, including scaling management.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Reddit;
using Reddit.Controllers;
using TMPro;
using System;
using System.Linq;

public class RedCentralSpaceMonobehaviour : MonoBehaviour
{

    public List<GameObject> nodeList = new List<GameObject>();
    public GameObject centralPlatform;
    public GameObject subredditTitleMesh;
    public GameObject boundaryRingPrefab;
    public GameObject boundarySphere;
    public float distanceMargin = 10;
    public float minScale = 20;

    //Boundary Parameters
    public float ringWidth = 3;
    public float ringHeight = 1;
    public int numSteps = 3;
    public float stepWidth = 1;
    private GameObject boundaryRing;
    private List<GameObject> stepList = new List<GameObject>();

    //adjustment helper parameters
    private float maxDistance;
    private float targetScale;
    private bool startTransform = true;
    private float scaleRate = 0.005f;
    private float nextUpdateTime = 0.0f;
    private float updateInterval = 2f;

    //Subreddit Data
    private Subreddit sr;
    private string subredditName = "all";
    //private List<string> infoModerators;
    private string infoDescription;
    private string infoSidebar;
    private List<string> sidebarPanelStrings;
    private List<GameObject> sidebarPanelObjects = new List<GameObject>();
    public GameObject subredditInfoParent;
    public GameObject sidebarPanelPrefab;
    private Reddit.Controllers.Internal.Lists infoLists;

    // Start is called before the first frame update
    void Start()
    {
        CreateBoundaryRing();
        GetAllNodes();
        InstantScale();
        //targetScale = centralPlatform.transform.localScale.x;
    }

    public void Initialise(Subreddit subreddit)
    {
        ClearAll();

        if (subreddit.Name != "all")
        {
            sr = subreddit.About();
            subredditName = sr.Name;

            if (sr.Description != null) infoDescription = sr.Description;
            if (sr.Sidebar != null)
            {
                infoSidebar = sr.Sidebar;
                sidebarPanelStrings = new List<string>(infoSidebar.Split(new
                     string[] { "***" }, StringSplitOptions.None));
                GeneratePanels();
            }
            if (sr.Lists != null) infoLists = sr.Lists;
        }
        else
        {
            subredditName = "all";
        }


        //Set geometry stuff
        RefreshGeometry();
    }

    public void RefreshGeometry()
    {
        subredditTitleMesh.GetComponent<TextMeshPro>().text = "r/" +
            subredditName;

    }

    public void ClearAll()
    {
        if (sidebarPanelStrings != null) sidebarPanelStrings.Clear();
        DestroyPanels();
    }

    public void GeneratePanels()
    {
        DestroyPanels();

        var cumulativeAngle = Mathf.PI / 6;
        var radius = 6.5f;

        for (int i = 0; i < sidebarPanelStrings.Count; i++)
        {
            float angleIncrease = Mathf.PI / 12;

            cumulativeAngle = -cumulativeAngle; //flip sides back and forth

            float x = Mathf.Cos(cumulativeAngle) * radius;
            float z = Mathf.Sin(cumulativeAngle) * radius;
            Vector3 pos = new Vector3(x, -2.25f, z);

            float angleDegrees = -cumulativeAngle * Mathf.Rad2Deg;
            Quaternion rot = Quaternion.Euler(0, angleDegrees + 90, 0);

            GameObject panel = GameObject.Instantiate(sidebarPanelPrefab,
                pos, rot);
            panel.transform.SetParent(subredditInfoParent.transform);

            panel.GetComponent<RedInfoPanelMonobehaviour>().SetText(
                sidebarPanelStrings[i]);

            sidebarPanelObjects.Add(panel);

            if (i % 2 == 1)
            {
                cumulativeAngle += angleIncrease;
            }
        }
    }

    public void DestroyPanels()
    {
        for (int i = 0; i < sidebarPanelObjects.Count; i++)
        {
            GameObject.Destroy(sidebarPanelObjects[i]);
        }
    }
```

```csharp

    // Update is called once per frame
    void Update()
    {

        if (Time.time > nextUpdateTime)
        {
            nextUpdateTime += updateInterval;

            CheckDistances();

            var platformRadius = (maxDistance + distanceMargin + stepWidth * numSteps);

            if (platformRadius * 2 > minScale && Mathf.Abs(platformRadius * 2 - centralPlatform.transform.localScale.x) > 1 && platformRadius < 80)
            {
                ChangeTargetScale(Mathf.Round((maxDistance + distanceMargin + stepWidth * numSteps) * 2));
            }

            scaleRate = 0.01f * Mathf.Clamp01(Mathf.Abs(platformRadius * 2 - centralPlatform.transform.localScale.x) / (distanceMargin * 2));
        }


        if (startTransform)
        {
            if (centralPlatform.transform.localScale.x - targetScale < -1f)
            {
                centralPlatform.transform.localScale += new Vector3(1, 0, 1) * scaleRate;
                float sphereScale = centralPlatform.transform.localScale.x + 2 * ringWidth + 2;
                boundarySphere.transform.localScale = new Vector3(sphereScale + 1f, sphereScale + 1f, sphereScale + 1f);
                UpdateBoundaryRing();
            }
            else if (centralPlatform.transform.localScale.x - targetScale > 1f)
            {
                centralPlatform.transform.localScale += new Vector3(-1, 0, -1) * scaleRate;
                float sphereScale = centralPlatform.transform.localScale.x + 2 * ringWidth + 2;
                boundarySphere.transform.localScale = new Vector3(sphereScale + 1f, sphereScale + 1f, sphereScale + 1f);
                UpdateBoundaryRing();
            }
            else
            {
                startTransform = false;
            }
        }
    }

    void CheckDistances()
    {
        float tempMax = minScale/4;
        Vector3 centerCoord = new Vector3(transform.position.x, 0, transform
            .position.z);

        nodeList.RemoveAll(i => i == null);

        foreach (GameObject i in nodeList)
        {
            Vector3 tempCoord = new Vector3(i.transform.position.x, 0, i.
                transform.position.z);
            float tempDist = Vector3.Distance(centerCoord, tempCoord);
            if (tempDist > tempMax)
            {
                tempMax = tempDist;
            }
        }

        maxDistance = tempMax;
    }

    void ChangeTargetScale(float target)
    {
        targetScale = target;
        startTransform = true;
    }

    void UpdateBoundaryRing()
    {
        boundaryRing.GetComponent<GeometryTorus>().SetParameters(
            centralPlatform.transform.localScale.x / 2 + ringWidth / 2,
            ringWidth, 64, 4, 0.5f, true);
        boundaryRing.transform.localScale = new Vector3(1, ringHeight /
            ringWidth, 1);

        for (int i = 0; i < stepList.Count; i++)
        {
            stepList[i].GetComponent<GeometryTorus>().SetParameters(
                centralPlatform.transform.localScale.x / 2 - (i * stepWidth
                * 1.3f) - ringWidth / 2, stepWidth, 64, 4, 0.5f, true);
            stepList[i].transform.localScale = new Vector3(1, (1 - (float)(i
                + 1) / (numSteps + 1)) * ringHeight / stepWidth, 1);
        }
    }

    void CreateBoundaryRing()
    {
        boundaryRing = GameObject.Instantiate(boundaryRingPrefab);
        boundaryRing.GetComponent<GeometryTorus>().SetParameters((
            centralPlatform.transform.localScale.x / 2) + (ringWidth / 2),
            ringWidth, 64, 4, 0.5f, true);
        boundaryRing.transform.localScale = new Vector3(1, ringHeight /
            ringWidth, 1);
        boundaryRing.transform.position = this.transform.position;
        boundaryRing.transform.SetParent(this.transform);

        for (int i = 0; i < numSteps; i++)
        {
            GameObject stepObject = GameObject.Instantiate(
                boundaryRingPrefab);
            stepObject.GetComponent<GeometryTorus>().SetParameters(
                centralPlatform.transform.localScale.x / 2 - (i * stepWidth
                * 1.3f) - ringWidth / 2, stepWidth, 64, 4, 0.5f, true);
            stepObject.transform.localScale = new Vector3(1, (1 - (float)(i
                + 1) / (numSteps + 1)) * ringHeight / stepWidth, 1);
            stepObject.transform.position = this.transform.position;
            stepObject.transform.SetParent(this.transform);

            stepList.Add(stepObject);
        }

    }

    public void InstantScale()
    {
        CheckDistances();

        var platformRadius = (maxDistance + distanceMargin + stepWidth *
            numSteps);

        if (platformRadius * 2 > minScale && Mathf.Abs(platformRadius * 2 -
            centralPlatform.transform.localScale.x) > 1 && platformRadius <
            80)
        {
            ChangeTargetScale(Mathf.Round((maxDistance + distanceMargin +
                stepWidth * numSteps) * 2));
        }

        centralPlatform.transform.localScale = new Vector3(targetScale, 0.3f
            , targetScale);
        float sphereScale = centralPlatform.transform.localScale.x + 2 * ringWidth + 2;
        boundarySphere.transform.localScale = new Vector3(sphereScale + 1f, sphereScale + 1f, sphereScale + 1f);

        UpdateBoundaryRing();

    }

    public void GetAllNodes()
    {
        nodeList.Clear();
        nodeList = GameObject.FindGameObjectsWithTag("NodeBlock").ToList();
        Debug.Log(nodeList.Count);
    }

    public void NodeListAdd(GameObject node)
    {
        nodeList.Add(node);
    }

    public void NodeListRemove(GameObject node)
    {
        nodeList.Remove(node);
    }

}
```

4.3 Reddit Central Sphere

Script controlling the collision events for the sphere area to determine if player is still in center area.

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

public class RedCenterSphereCollision : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

4.4 Reddit Cube Spawner

Script for the pedestal that spawns the node-block.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Realtime;
using Photon.Pun;

public class RedCubeSpawner : MonoBehaviour
{
    private bool touchingNode;
    public RedCentralSpaceMonobehaviour centralControl;
    public ColorChanger colorChanger;
    private List<Collision> collidingNodes = new List<Collision>();
    private List<GameObject> nodeBlockList = new List<GameObject>();
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void SpawnCube()
    {
        if(!IsTouchingNode())
        {
            object[] id = new object[1];
            id[0] = Random.Range(1, 255);

            GameObject cube = PhotonNetwork.Instantiate("Reddit/NodeBlock",
                transform.position + new Vector3(0, 1, 0), transform.
                rotation, 0, id);
            nodeBlockList.Add(cube);
            centralControl.NodeListAdd(cube);
        }
    }

    void OnCollisionEnter(Collision collision)
    {
        if(collision.gameObject.CompareTag("NodeBlock"))
        {
            collidingNodes.Add(collision);
            colorChanger.SetColor(1);
        }
    }

    void OnCollisionExit(Collision collision)
    {
        if(collision.gameObject.CompareTag("NodeBlock"))
        {
            collidingNodes.Remove(collision);
            if(!IsTouchingNode())
            {
                colorChanger.SetColor(0);
            }
        }
    }

    public bool IsTouchingNode()
    {
        if(collidingNodes.Count > 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public void HoverIn()
    {
        if(!IsTouchingNode())
        {
            colorChanger.SetColor(2);
        }
    }

    public void HoverOut()
    {
        if(!IsTouchingNode())
        {
            colorChanger.SetColor(0);
        }
    }


}
```

4.5 Reddit Node Block

Script controlling the node block - just simple sync functions and destroy if it falls off the map.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
using Photon.Voice.Unity;

public class RedNodeBlock : MonoBehaviour, IPunInstantiateMagicCallback
{
    public int voiceRoomID = 0;
    public RedNodeSphere sphere;


    // Start is called before the first frame update
    void Start()
    {

    }

    public void OnPhotonInstantiate(PhotonMessageInfo info)
    {
        object[] data = info.photonView.InstantiationData;
        if (data != null)
        {
            voiceRoomID = (int)data[0];
            sphere.voiceRoomID = voiceRoomID;
        }
        else
        {
            voiceRoomID = this.GetComponent<PhotonView>().ViewID;
            sphere.voiceRoomID = voiceRoomID;
        }

    }

    // Update is called once per frame
    void Update()
    {
        if(transform.position.y <= -100)
        {
            Destroy();
        }
    }

    void ConnectToRoom()
    {

    }

    public void Destroy()
    {

        PhotonNetwork.Destroy(this.GetComponent<PhotonView>());
    }
}
```

4.6 Reddit Node Sphere

Handles collision events for the projected sphere around each node block.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
using Photon.Voice.Unity;
using Photon.Voice.PUN;

public class RedNodeSphere : MonoBehaviour
{
    public int voiceRoomID = 0;
    public PhotonVoiceNetwork voiceNetwork;
    public Recorder recorder;
    public PlayerController playerController;
    private List<GameObject> collidingPlayers = new List<GameObject>();

    // Start is called before the first frame update
    void Start()
    {
        playerController = GameObject.Find("XR Rig").GetComponent<
            PlayerController>();
        voiceNetwork = GameObject.Find("VoiceManager").GetComponent<
            PhotonVoiceNetwork>();
        recorder = GameObject.Find("VoiceManager").GetComponent<Recorder>();
    }

    // Update is called once per frame
    void Update()
    {
    }

    void OnTriggerEnter(Collider collider)
    {
        if(collider.gameObject.CompareTag("Player"))
        {
            playerController.numVoiceRooms += 1;

            byte[] room = new byte[1];
            room[0] = (byte)voiceRoomID;
            voiceNetwork.Client.OpChangeGroups(null, room);
            recorder.InterestGroup = room[0];

            Debug.Log("Joined group " + room[0]);
        }
    }

    void OnTriggerExit(Collider collider)
    {
        if(collider.gameObject.CompareTag("Player"))
        {
            playerController.numVoiceRooms -= 1;

            byte[] room = new byte[1];
            room[0] = (byte)voiceRoomID;
            voiceNetwork.Client.OpChangeGroups(room, null);

            if(playerController.numVoiceRooms == 0 && recorder.InterestGroup
                 != 0)
            {
                recorder.InterestGroup = 0;
                Debug.Log("Recorder joined back group 0.");
            }

            Debug.Log("Disconnected from group " + room[0]);
        }
    }
}
```

4.7 Subreddit Star

Script to handle the behaviour of individual subreddit links.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using Photon.Pun;
using UnityEngine.XR.Interaction.Toolkit;

public class SubredditStarMonobehaviour : MonoBehaviour
{
    public GameObject titleMesh;
    private string subreddit;
    //private RedditController redditController;
    private NetworkManager networkManager;

    // Start is called before the first frame update
    void Start()
    {
        //InitialiseStar("");
        //redditController = GameObject.Find("RedditManager").GetComponent<
            RedditController>();
        networkManager = GameObject.Find("NetworkManager").GetComponent<
            NetworkManager>();
        HideTitle();
    }

    public void InitialiseStar(string subredditName)
    {
        subreddit = subredditName;
        //Debug.Log("subreddit = " + subreddit);
        titleMesh.GetComponent<TextMeshPro>().text = string.Concat("r/",
            subreddit);
    }

    public void ChangeSubreddit()
    {
        //networkManager.rig.GetComponent<XRRig>().MatchRigUpRigForward(new
            Vector3(0, 3, 0), new Vector3(1, 0, 0));
        networkManager.SetPlayerSubreddit(subreddit);
        //redditController.Initialise(subreddit);
    }

    public void ShowTitle()
    {
        //titleMesh.SetActive(true);
        titleMesh.GetComponent<TextMeshPro>().fontSize = 15;
    }

    public void HideTitle()
    {
        //titleMesh.SetActive(false);
        titleMesh.GetComponent<TextMeshPro>().fontSize = 6;
    }

    // Update is called once per frame
    void Update()
    {
        titleMesh.transform.rotation = Camera.main.transform.rotation;
    }
}
```

4.8 Reddit Info Panel

Handles the info panel boards

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using LogicUI.FancyTextRendering;
using NaughtyAttributes;

public class RedInfoPanelMonobehaviour : MonoBehaviour
{

    public GameObject mainText;
    private Camera XRcamera;
    public Canvas cameraCanvas;

    // Start is called before the first frame update
    void Start()
    {
        XRcamera = GameObject.Find("VR Camera Controller").GetComponent<
            Camera>();
        cameraCanvas.GetComponent<Canvas>().worldCamera = XRcamera;
        Invoke("FitContents", 0.1f);
    }

    public void Initialise()
    {
        Invoke("FitContents", 0.1f);
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void SetText(string str)
    {
        mainText.GetComponent<MarkdownRenderer>().Source = str;
        Invoke("FitContents", 0.1f);
    }

    void FitContents()
    {
        RectTransform textRect = mainText.GetComponent<RectTransform>();

        textRect.localPosition = new Vector3(0, - textRect.sizeDelta.y / 2,
            0);
    }
}
```

4.9 Geometry Torus

Script that generates a torus-shape dynamically at runtime, because otherwise scaling a pre-made torus would be uniform and not give the expected results. Used for the giant rings as well as the cylindrical boundaries of the center space.

```
/**
```

```
 * Based on a script by Steffen (http://forum.unity3d.com/threads/torus-in-
     unity.8487/) (in $primitives_966_104.zip, originally named "Primitives.
     cs")
 *
 * Editted by Michael Zoller on December 6, 2015.
 * It was shortened by about 30 lines (and possibly sped up by a factor of 2
     ) by consolidating math & loops and removing intermediate Collections.
 */
using UnityEngine;

[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]
public class GeometryTorus : MonoBehaviour
{

    public float segmentRadius = 1f;
    public float tubeRadius = 0.1f;
    public int numSegments = 32;
    public int numTubes = 12;
    public float tubeRotation = 0.0f;
    public bool hardEdges = false;

    void Start()
    {
        RefreshTorus();
    }

    public void SetParameters(float circleRadius, float thickness, int
        circleResolution, int tubeResolution, float tubeRotate, bool
        hardShading)
    {
        segmentRadius = circleRadius;
        tubeRadius = thickness;
        numSegments = circleResolution;
        numTubes = tubeResolution;
        tubeRotation = tubeRotate;
        hardEdges = hardShading;

        RefreshTorus();
    }

    public void RefreshTorus()
    {
        // Total vertices
        int totalVertices = numSegments * numTubes;

        // Total primitives
        int totalPrimitives = totalVertices * 2;

        // Total indices
        int totalIndices = totalPrimitives * 3;

        // Init the mesh
        Mesh mesh = new Mesh();

        // Init the vertex and triangle arrays
        Vector3[] vertices = new Vector3[totalVertices];
        int[] triangleIndices = new int[totalIndices];

        // Calculate size of a segment and a tube
        float segmentSize = 2 * Mathf.PI / (float)numSegments;
        float tubeSize = 2 * Mathf.PI / (float)numTubes;

        // Create floats for our xyz coordinates
        float x, y, z;

        // Begin loop that fills in both arrays
        for (int i = 0; i < numSegments; i++)
        {
            // Find next (or first) segment offset
            int n = (i + 1) % numSegments; // changed segmentList.Count to numSegments

            // Find the current and next segments
            int currentTubeOffset = i * numTubes;
            int nextTubeOffset = n * numTubes;

            for (int j = 0; j < numTubes; j++)
            {
                // Find next (or first) vertex offset
                int m = (j + 1) % numTubes; // changed currentTube.Count to numTubes

                // Find the 4 vertices that make up a quad
                int iv1 = currentTubeOffset + j;
                int iv2 = currentTubeOffset + m;
                int iv3 = nextTubeOffset + m;
                int iv4 = nextTubeOffset + j;

                // Calculate X, Y, Z coordinates.
                x = (segmentRadius + tubeRadius * Mathf.Cos((j + tubeRotation) * tubeSize)) * Mathf.Cos(i * segmentSize);
                z = (segmentRadius + tubeRadius * Mathf.Cos((j + tubeRotation) * tubeSize)) * Mathf.Sin(i * segmentSize);
                y = tubeRadius * Mathf.Sin((j + tubeRotation) * tubeSize);

                // Add the vertex to the vertex array
                vertices[iv1] = new Vector3(x, y, z);

                // "Draw" the first triangle involving this vertex
                triangleIndices[iv1 * 6] = iv1;
                triangleIndices[iv1 * 6 + 1] = iv2;
                triangleIndices[iv1 * 6 + 2] = iv3;
                // Finish the quad
                triangleIndices[iv1 * 6 + 3] = iv3;
                triangleIndices[iv1 * 6 + 4] = iv4;
                triangleIndices[iv1 * 6 + 5] = iv1;
            }
        }
        mesh.vertices = vertices;
        mesh.triangles = triangleIndices;

        mesh.RecalculateBounds();
        mesh.RecalculateNormals(); // added on suggestion of Eric5h5 &
            joaeba in the forum thread

        //HARD SHADING
        if(hardEdges)
        {
            Vector3[] oldVerts = mesh.vertices;
            int[] triangles = mesh.triangles;
            Vector3[] newVertices  = new Vector3[triangles.Length];
            for (int i = 0; i < triangles.Length; i++)
            {
                newVertices[i] = oldVerts[triangles[i]];
                triangles[i] = i;
            }
            mesh.vertices = newVertices;
            mesh.triangles = triangles;
            mesh.RecalculateBounds();
            mesh.RecalculateNormals();
        }


        mesh.Optimize();
        MeshFilter mFilter = GetComponent<MeshFilter>(); // tweaked to
            Generic
        mFilter.mesh = mesh;
        MeshCollider mCollider = GetComponent<MeshCollider>();
        if(mCollider != null)
        {
            mCollider.sharedMesh = mesh;
        }
    }
}
```

5 Reddit Post Scripts

Sub-scripts that handle various aspects of the reddit object behaviours such as posts, comments, scrolling, etc.

5.1 Reddit Post Controller

Script handling all the procedural generation of post space geometry, as well as interactions with interface and other stuff.

```
using System.Collections;
using System.Collections.Generic;
using System.Net;
using System.IO;
using UnityEngine;
using UnityEngine.Networking;
using Reddit;
using Reddit.Controllers;
using TMPro;
using Vuplex.WebView;
using System.Globalization;

public class RedPostMonobehaviour : MonoBehaviour
{

    //REDDIT POST DATA
    private Post post;
    private int numUpvotes;
    private string postText;
    private string postURL;
    private string postTitle;
    private string postSubreddit;
    private Comments postComments;
    private WebClient webClient;


    //GEOMETRY
    public GameObject platformGeometry;
    public GameObject platformCenter;
    public GameObject platformLeft;
    public GameObject platformRight;
    public GameObject platformSimple;
    public GameObject platformWall;
    public GameObject platformBase;
    public GameObject platformAreaCollider;
    public GameObject mainContent;
    public GameObject mainContentText;
    public GameObject mainTitle;
    public GameObject mainTitleText;
    public GameObject interfaceButtons;
    public GameObject commentFrame;
    public GameObject cylinderPlatform;
    public GameObject externalThumbnail;
    public GameObject externalTitle;
    public GameObject externalSubreddit;
    public GameObject externalUpvotes;

    private List<GameObject> commentFramesList = new List<GameObject>();
    private List<GameObject> cylinderPlatforms = new List<GameObject>();
    //private GameObject textTemplateColumn;
    //private GameObject textTemplateComment;
    public GameObject imagePlane;
    public GameObject webViewObject;
    public GameObject webView;
    public GameObject webViewFrame;
    public GameObject webViewLoadingText;
    public float frameTextureValue = 0f;
    WebViewPrefab webViewPrefab;
    private Texture thumbnailTexture;
    private Texture imageTexture;
    private Color platformColor;

    //PARAMETERS
    //private (int votesCeil, int minRadius, int maxRadius) upvoteScaling =
        (20000, 6, 12);
    private bool instantiateCommentFrames = true;
    private bool loadComments = true;
    private bool linkPost = false;
    private int platformLength;
    private float platformLengthUnits;
    private float unitLength = 4f;
    private float collisionHeight = 0f;

    // Start is called before the first frame update
    void Start()
    {
        //platformGeometry = transform.Find("Platform").gameObject;
        //platformAreaCollider = transform.Find("Collider").GetComponent<
            Collider>();
        //mainText = transform.Find("MainText").gameObject;
        //imagePlane = transform.Find("ImagePlane").gameObject;
        webViewPrefab = webView.GetComponent<WebViewPrefab>();

        platformColor = LibColor.RandomiseHSV(new Color(0.5f, 0.5f, 0.7f, 1.0f), 0.3f, 0.1f, 0.1f);
        RefreshGeometry();
        HideInterior();

    }

    // Update is called once per frame
    void Update()
    {
        var video = post.Listing.Media;
    }

    public void InitialisePost(Post p, Comments c, int length, float upvoteScale)
    {
        //POST DATA
        post = p;
        numUpvotes = p.UpVotes;
        postTitle = p.Title;
        postSubreddit = p.Subreddit;
        postComments = c;

        //PLATFORM GEOMETRY
        platformLength = length;
        platformLengthUnits = platformLength * unitLength;
        frameTextureValue = upvoteScale;

        if (post.Listing.IsSelf)
        {
            postText = ((SelfPost)post).SelfText;
            if (postText.Length == 0)
            {
                //postText = postTitle;
            }
        }
        else
        {
            postText = postTitle;
            postURL = ((LinkPost)post).URL;
            //Debug.Log("Not self post!");
            StartCoroutine(GetThumbnailTexture(p.Listing.Thumbnail));
            StartCoroutine(GetImageTexture(postURL));
            if (!post.Listing.IsRedditMediaDomain && postURL != null)
            {
                //Debug.Log(postURL);
                linkPost = true;
            }
        }


        RefreshGeometry();
        //RefreshComments();

    }

    void OnTriggerEnter(Collider collider)
    {
        //Debug.Log("Collision");
        if (collider.gameObject.tag == "Player")
        {
            ShowInterior();
            Debug.Log("Player entered post - refreshing comments...");
            RefreshComments();
            if (linkPost) InitialiseWebView(postURL);
        }
    }

    void OnTriggerExit(Collider collider)
    {
        //Debug.Log("Collision");
        if (collider.gameObject.tag == "Player")
        {
            HideInterior(true);
        }
    }

    private void RefreshGeometry()
    {
        //Calculate size of platform
        //platformRadius = upvoteScaling.minRadius + Mathf.Clamp((numUpvotes / upvoteScaling.votesCeil), 0, 1) * (upvoteScaling.maxRadius - upvoteScaling.minRadius);

        //Platform geometry
        platformCenter.transform.localScale = new Vector3(platformLengthUnits / 2, 0.5f, 9); //old transform without frame prefab
        platformLeft.transform.localScale = new Vector3(platformLengthUnits, 1, 4);
        platformRight.transform.localScale = new Vector3(platformLengthUnits, 1, 4);
        platformSimple.transform.localScale = new Vector3(platformLengthUnits, 18, 0.5f);

        platformGeometry.GetComponent<MeshRenderer>().material.color = platformColor;

        //Platform Text
        //var imagePos = new Vector3(platformLengthUnits / 3, 0, 0);
        //var textPos = new Vector3(platformLengthUnits / 3 + 0.2f, 3.5f, 0)
            ;
        var rot = Quaternion.identity;
        //imagePlane.transform.localPosition = imagePos;
        //mainText.transform.localPosition = textPos;

        mainContentText.GetComponent<TextMeshProUGUI>().text = postText;
        mainContent.GetComponent<RedTextFrameMonobehaviour>().Initialise();
        mainTitleText.GetComponent<TextMeshProUGUI>().text = postTitle;
        mainTitle.GetComponent<RedTextFrameMonobehaviour>().Initialise();

        externalTitle.GetComponent<TextMeshPro>().text = postTitle;
        externalSubreddit.GetComponent<TextMeshPro>().text = numUpvotes.
            ToString() + " | r/" + postSubreddit;

        externalTitle.GetComponent<RectTransform>().sizeDelta = new Vector2(
            platformLengthUnits - 3, 4);
        externalSubreddit.GetComponent<RectTransform>().sizeDelta = new
            Vector2(platformLengthUnits - 3, 4);
        externalUpvotes.GetComponent<RectTransform>().sizeDelta = new Vector
            2(platformLengthUnits - 3, 4);

        MaterialManager matManager = GameObject.Find("MaterialManager").
            GetComponent<MaterialManager>();
        Color frameColor = Color.HSVToRGB(0.1f, 0.7f, 0.8f);
        Material frameMaterial = matManager.InstantiateMaterialWithEmission(
            string.Concat("FrameMaterial", frameTextureValue), frameColor, 0
            .5f + frameTextureValue * 2);
        platformSimple.GetComponent<FrameDynamic>().ReplaceBaseMaterial(
            frameMaterial);
        platformSimple.GetComponent<FrameDynamic>().SetFrameMaterial();

        if (post.Listing.IsSelf)
        {
            //Resize title to be larger in absence of thumbnail
            externalTitle.GetComponent<RectTransform>().sizeDelta = new
                Vector2(platformLengthUnits - 3, 10);
            externalTitle.transform.localPosition = new Vector3(0, 1.3f, 1.8
                f);
        }

        if (instantiateCommentFrames)
        {
            foreach (GameObject i in commentFramesList)
            {
                GameObject.Destroy(i);
            }

            for (int i = 0; i < platformLength; i++)
            {
                float xpos = (platformLengthUnits / 2) - (i * unitLength) - (unitLength / 2);

                GameObject newComment = Instantiate(commentFrame, transform);
                newComment.transform.localPosition = new Vector3(xpos, 1f, -(18f / 2 + 3));
                newComment.transform.rotation *= Quaternion.Euler(new Vector3(0, 180, 0));
                commentFramesList.Add(newComment);

                //GameObject newPlatform = Instantiate(cylinderPlatform, transform);
                //newPlatform.transform.localPosition = new Vector3(xpos, 0.6f, -5.5f);
                //cylinderPlatforms.Add(newPlatform);

                newComment = Instantiate(commentFrame, transform);
                newComment.transform.localPosition = new Vector3(xpos, 1f, 18f / 2 + 3);
                //newComment.transform.rotation *= Quaternion.Euler(new Vector3(0, 180, 0));
                commentFramesList.Add(newComment);

                //newPlatform = Instantiate(cylinderPlatform, transform);
                //newPlatform.transform.localPosition = new Vector3(xpos, 0.6f, 5.5f);
                //cylinderPlatforms.Add(newPlatform);
            }

            instantiateCommentFrames = false;
        }

        RefreshCollisionHeight(20f);


        Invoke("RetrieveVotes", 1.0f);
        RefreshImage();
        RefreshThumbnail();

    }

    private void ShowInterior()
    {
        platformSimple.SetActive(false);
        externalThumbnail.SetActive(false);
        externalTitle.SetActive(false);
        externalSubreddit.SetActive(false);
        externalUpvotes.SetActive(false);

        platformCenter.SetActive(true);
        platformLeft.SetActive(true);
        platformRight.SetActive(true);
        platformWall.SetActive(true);
        platformBase.SetActive(true);

        if (linkPost) webViewObject.SetActive(true);
        else if (imageTexture != null) imagePlane.SetActive(true);
        else if (postText != "") mainContent.SetActive(true);

        mainTitle.SetActive(true);
        interfaceButtons.SetActive(true);

        foreach (GameObject i in commentFramesList)
        {
            i.SetActive(true);
        }
        foreach (GameObject i in cylinderPlatforms)
        {
            i.SetActive(true);
        }
    }

    private void HideInterior(bool hideComments = true)
    {
        platformSimple.SetActive(true);
        externalThumbnail.SetActive(true);
        externalTitle.SetActive(true);
        externalSubreddit.SetActive(true);
        externalUpvotes.SetActive(true);

        platformCenter.SetActive(false);
        platformLeft.SetActive(false);
        platformRight.SetActive(false);
        platformWall.SetActive(false);
        platformBase.SetActive(false);

        imagePlane.SetActive(false);
        webViewObject.SetActive(false);
        mainContent.SetActive(false);
        mainTitle.SetActive(false);
        interfaceButtons.SetActive(false);

        if (hideComments)
        {
            foreach (GameObject i in commentFramesList)
            {
                i.SetActive(false);
            }
        }
        foreach (GameObject i in cylinderPlatforms)
        {
            i.SetActive(false);
        }
    }

    private void RefreshComments()
    {
        if (!loadComments) return;

        int numComments = platformLength * 2;
        var topComments = RetrieveComments(numComments);

        for (int c = 0; c < commentFramesList.Count; c++)
        {
            if (c < topComments.Count)
            {
                commentFramesList[c].GetComponent<
                    RedCommentFrameMonobehaviour>().InitialiseComments(
                    topComments[c]);
            }
            else
            {
                //commentFramesList[c].GetComponent<
                    RedCommentFrameMonobehaviour>().Deactivate();
                commentFramesList[c].SetActive(false);
            }
        }

        loadComments = false;

        RefreshGeometry();
    }

    public void DeepRefreshAllComments()
    {
        if(!loadComments)
        {
            for (int c = 0; c < commentFramesList.Count; c++)
            {
                if(commentFramesList[c].activeSelf)
                {
                    commentFramesList[c].GetComponent<
                        RedCommentFrameMonobehaviour>().DeepRefreshComments(
                        );
                }
            }
        }
    }

    private async void InitialiseWebView(string url)
    {
        webViewPrefab = webView.GetComponent<WebViewPrefab>();

        //Set initial url of webview and initialise
        webViewPrefab.InitialUrl = url;
        webViewPrefab.Init();

        await webViewPrefab.WaitUntilInitialized();
        //bool initialLoad = true;
        //Wait until webpage is fully loaded, then check height of page and resize webview window accordingly.
        webViewPrefab.WebView.LoadProgressChanged += async (sender, eventArgs) =>
        {
            //Resize when finished
            if (eventArgs.Type == ProgressChangeType.Finished)
            {
                var heightString = await webViewPrefab.WebView.ExecuteJavaScript("document.documentElement.scrollHeight");
                var heightInPixels = float.Parse(heightString, CultureInfo.InvariantCulture);
                float heightInUnityUnits;
                bool tooLong = false;

                if (webViewPrefab.WebView.Size.x * webViewPrefab.WebView.Resolution * heightInPixels > 7000000)
                {
                    //Throw error if webpage too large to load
                    webViewLoadingText.GetComponent<TextMeshPro>().text = "WEBPAGE TOO LONG";
                    heightInPixels = 7000000 / (webViewPrefab.WebView.Size.x * webViewPrefab.WebView.Resolution);
                    tooLong = true;
                }

                ResizeWebView(heightInPixels, out heightInUnityUnits);

                //Hide scrollbar and loading text
                await webViewPrefab.WebView.ExecuteJavaScript("document.body.style.overflow = 'hidden';");
                if (!tooLong)
                {
                    webViewLoadingText.GetComponent<TextMeshPro>().text = "";
                    webViewLoadingText.gameObject.SetActive(false);
                }

                webViewLoadingText.transform.localPosition = new Vector3(0,
                    -heightInUnityUnits, 0);

            }
        };

    }

    private void ResizeWebView(float heightInPixels, out float
        heightInUnityUnits)
    {
        //Resize webview window to accommodate full webpage height
        Debug.Log("Webpage Height: " + heightInPixels);
        var existingWidth = webViewPrefab.WebView.Size.x;
        heightInUnityUnits = heightInPixels / webViewPrefab.WebView.
            Resolution;

        webViewPrefab.Resize(existingWidth, heightInUnityUnits);

        //Resize backing frame accordingly
        webViewFrame.transform.localScale = new Vector3(existingWidth + 0.2f
            , heightInUnityUnits + 0.2f, 0.09f);
        webViewFrame.transform.localPosition = new Vector3(0.051f, -(
            heightInUnityUnits / 2 - 0.5f), 0);

        RefreshCollisionHeight(heightInUnityUnits);
    }

    IEnumerator GetImageTexture(string url)
    {
        UnityWebRequest www = UnityWebRequestTexture.GetTexture(url);
        yield return www.SendWebRequest();

        if (www.isNetworkError || www.isHttpError)
        {
            //Debug.Log("Image retrieval error");
        }
        else
        {
            imageTexture = DownloadHandlerTexture.GetContent(www);
            //Debug.Log("Image downloaded");
            Invoke("RefreshImage", 5.0f);
        }


    }

    IEnumerator GetThumbnailTexture(string url)
    {
        UnityWebRequest www = UnityWebRequestTexture.GetTexture(url);
        yield return www.SendWebRequest();

        if (www.isNetworkError || www.isHttpError)
        {
            //Debug.Log("Thumbnail retrieval error");
        }
        else
        {
            thumbnailTexture = DownloadHandlerTexture.GetContent(www);
            //Debug.Log("Thumbnail downloaded");
            Invoke("RefreshThumbnail", 5.0f);
        }

    }

    private void RefreshImage()
    {
        //SET IMAGE TEXTURE
        if (imageTexture != null && imageTexture.height > 0 && imageTexture.
            width > 0)
        {
            imagePlane.GetComponent<MeshRenderer>().enabled = true;
            float imageRatio = (float)imageTexture.height / (float)
                imageTexture.width;
            imagePlane.transform.localScale = new Vector3(5f, 5f *
                imageRatio, 0.1f);
            imagePlane.transform.localPosition = new Vector3(7f, -(5f *
                imageRatio - 5f) / 2 - 2, 0);
            imagePlane.GetComponent<MeshRenderer>().material.SetTexture("
                _BaseMap", imageTexture);
            imagePlane.GetComponent<FrameDynamic>().RefreshFrame();
            //Debug.Log("Setting Texture");
        }
        else
        {
            imagePlane.GetComponent<MeshRenderer>().enabled = false;
        }
    }

    private void RefreshThumbnail()
    {
        //SET THUMBNAIL TEXTURE
        if (thumbnailTexture != null && thumbnailTexture.height > 0 &&
            thumbnailTexture.width > 0)
        {
            externalThumbnail.GetComponent<MeshRenderer>().enabled = true;
            float thumbnailRatio = (float)thumbnailTexture.height / (float)
                thumbnailTexture.width;
            //if (thumbnailRatio == 0) thumbnailRatio = 1;
            float platformRatio = 12f / platformLengthUnits;
            if (thumbnailRatio >= platformRatio)
            {
                externalThumbnail.transform.localScale = new Vector3(10f, 10
                    f, 1);
            }
            else
            {
                externalThumbnail.transform.localScale = new Vector3(
                    platformLengthUnits - 2, (platformLengthUnits - 2) *
                    thumbnailRatio, 1);
            }
            externalThumbnail.GetComponent<MeshRenderer>().material.
                SetTexture("_BaseMap", thumbnailTexture);
            //Debug.Log("Setting Texture");

            //Resize title to default with thumbnail
            externalTitle.GetComponent<RectTransform>().sizeDelta = new
                Vector2(platformLengthUnits - 3, 4);
            externalTitle.transform.localPosition = new Vector3(0, 1.3f, 4.3
                f);
        }
        else
        {
            externalThumbnail.GetComponent<MeshRenderer>().enabled = false;

            //Resize title to be larger in absence of thumbnail
            externalTitle.GetComponent<RectTransform>().sizeDelta = new
                Vector2(platformLengthUnits - 3, 10);
            externalTitle.transform.localPosition = new Vector3(0, 1.3f, 1.8
                f);
        }
    }

    private List<Comment> RetrieveComments(int numComments)
    {
        //var numComments = 4; //Mathf.CeilToInt(2 * Mathf.PI *
            platformRadius / 8);
        var topComments = postComments.GetComments(sort: "top", 0, 0, true,
            false, true, 2, 1000);
        return topComments;
    }

    public void RefreshCollisionHeight(float yheight)
    {
        //only fire if yheight is greater than current height
        if (yheight > collisionHeight)
        {
            yheight = Mathf.Ceil(yheight);
            platformWall.transform.localScale = new Vector3(
                platformLengthUnits + 4, 18 + 8, yheight + 20 + 4);
            platformWall.transform.localPosition = new Vector3(0, -((yheight + 20)/2), 0);
            platformBase.transform.localScale = new Vector3(platformLengthUnits + 4, 18 + 8, yheight + 20 + 4);
            platformBase.transform.localPosition = new Vector3(0, -(yheight + 20 + 2));
            platformAreaCollider.transform.localScale = new Vector3(platformLengthUnits + 4, yheight + 20 + 20, 25);
            platformAreaCollider.transform.localPosition = new Vector3(0, -((yheight + 20)/2), 0);

            foreach(GameObject c in commentFramesList)
            {
                c.GetComponent<RedCommentFrameMonobehaviour>().ResizeVerticalFX(yheight + 20 + 4 - 1);
            }

            collisionHeight = yheight;
        }

    }

    public void Vote(int vote)
    {
        if(vote == 1)
        {
            post.UpvoteAsync();
            platformSimple.GetComponent<FrameDynamic>().SetFrameMaterial(1);
            imagePlane.GetComponent<FrameDynamic>().SetFrameMaterial(1);
            mainTitle.GetComponent<RedTextFrameMonobehaviour>().frameGeometry.GetComponent<ColorChanger>().SetColor(1);
            mainContent.GetComponent<RedTextFrameMonobehaviour>().frameGeometry.GetComponent<ColorChanger>().SetColor(1);
            webViewFrame.GetComponent<ColorChanger>().SetColor(1);
        }
        else if(vote == -1)
        {
            post.DownvoteAsync();
            platformSimple.GetComponent<FrameDynamic>().SetFrameMaterial(2);
            imagePlane.GetComponent<FrameDynamic>().SetFrameMaterial(2);
            mainTitle.GetComponent<RedTextFrameMonobehaviour>().frameGeometry.GetComponent<ColorChanger>().SetColor(2);
            mainContent.GetComponent<RedTextFrameMonobehaviour>().frameGeometry.GetComponent<ColorChanger>().SetColor(2);
            webViewFrame.GetComponent<ColorChanger>().SetColor(2);
        }
        else if(vote == 0)
        {
            post.UnvoteAsync();
            platformSimple.GetComponent<FrameDynamic>().SetFrameMaterial(0);
            imagePlane.GetComponent<FrameDynamic>().SetFrameMaterial(0);
            mainTitle.GetComponent<RedTextFrameMonobehaviour>().
                frameGeometry.GetComponent<ColorChanger>().SetColor(0);
            mainContent.GetComponent<RedTextFrameMonobehaviour>().
                frameGeometry.GetComponent<ColorChanger>().SetColor(0);
            webViewFrame.GetComponent<ColorChanger>().SetColor(0);
        }

    }

    public void RetrieveVotes()
    {
        if(post.IsUpvoted)
        {
            interfaceButtons.GetComponentInChildren<
                VoteControllerMonobehaviour>().SetVote(1);
            platformSimple.GetComponent<FrameDynamic>().SetFrameMaterial(1);
            imagePlane.GetComponent<FrameDynamic>().SetFrameMaterial(1);
            mainTitle.GetComponent<RedTextFrameMonobehaviour>().
                frameGeometry.GetComponent<ColorChanger>().SetColor(1);
            mainContent.GetComponent<RedTextFrameMonobehaviour>().
                frameGeometry.GetComponent<ColorChanger>().SetColor(1);
            webViewFrame.GetComponent<ColorChanger>().SetColor(1);
        }
        else if(post.IsDownvoted)
        {
            interfaceButtons.GetComponentInChildren<
                VoteControllerMonobehaviour>().SetVote(-1);
            platformSimple.GetComponent<FrameDynamic>().SetFrameMaterial(2);
            imagePlane.GetComponent<FrameDynamic>().SetFrameMaterial(2);
            mainTitle.GetComponent<RedTextFrameMonobehaviour>().
                frameGeometry.GetComponent<ColorChanger>().SetColor(2);
            mainContent.GetComponent<RedTextFrameMonobehaviour>().
                frameGeometry.GetComponent<ColorChanger>().SetColor(2);
            webViewFrame.GetComponent<ColorChanger>().SetColor(2);
        }
    }
}
```

5.2 Reddit Comment Frame

Script handling all the procedural generation and calculation of comment frames, as well as associated interactions.

```csharp
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using Reddit;
using Reddit.Controllers;
using TMPro;
using UnityEngine.UI;
using NaughtyAttributes;
using LogicUI.FancyTextRendering;

public class RedCommentFrameMonobehaviour : MonoBehaviour
{

    //PLAYER
    private GameObject rig;
    private MovementProvider playerMovementProvider;

    //GAMEOBJECT / GEOMETRY
    private Camera XRcamera;
    public GameObject canvas;
    public GameObject frameGeometry;
    public GameObject textContainer;
    public GameObject mainText;
    public GameObject refreshButton;
    public GameObject voteArrows;
    public GameObject subTextPrefab;
    public GameObject helperContainer;
    public Material subCommentMaterial;
    public GameObject verticalFX;
    public int totalReplies = 0;
    private RedPostMonobehaviour parentPostController;
    private ScrollRect scrollRect;
    private MaterialManager materialManager = null;
    private List<string> subCommentStrings = new List<string>();
    private List<int> subCommentDepth = new List<int>();
    private List<GameObject> subCommentObjectList = new List<GameObject>();
    private List<Comment> subCommentList = new List<Comment>();

    //REDDIT COMMENT DATA
    private Comment mainComment = null;
    private List<Comment> directReplies;


    //PARAMETERS
    private float frameWidth = 2.4f;
    private float frameMargin = 0.2f;
    private float yMargin = 0.05f;


    private bool initialised = false;
    private bool deepRefresh = false;
    private Vector2 previousVector2 = new Vector2(0, 0);


    public void InitialiseComments(Comment c)
    {
        mainComment = c;

        if (!initialised)
        {
            //directReplies = RetrieveChildComments(c);
            RetrieveAllChildComments(c.replies, 1);
            totalReplies = c.NumReplies.Value;

            UpdateText();
            RetrieveVote();
            //FitContents();
            Invoke("FitContents", 0.1f); //need to delay fitcontents so '
                content fitter' has time to work
            initialised = true;
        }

        parentPostController = transform.parent.GetComponent<
            RedPostMonobehaviour>();
    }

    // Start is called before the first frame update
    void Start()
    {
        XRcamera = GameObject.Find("VR Camera Controller").GetComponent<
            Camera>();
        rig = GameObject.Find("XR Rig");
        playerMovementProvider = rig.GetComponent<MovementProvider>();
        canvas.GetComponent<Canvas>().worldCamera = XRcamera;
        scrollRect = textContainer.GetComponent<ScrollRect>();

        materialManager = GameObject.Find("MaterialManager").GetComponent<
            MaterialManager>();

        //canvas.SetActive(false);
    }

    // Update is called once per frame
    void Update()
    {

    }

    void UpdateText()
    {
        mainText.GetComponent<MarkdownRenderer>().Source = "";
        mainText.GetComponent<MarkdownRenderer>().Source = mainComment.
            UpVotes + " | u/" + mainComment.Author + "\n" + mainComment.Body
            ;


        for (int i = 0; i < subCommentStrings.Count; i++)
        {
            GameObject subComment = Instantiate(subTextPrefab, textContainer
                .transform);
            //subComment.GetComponent<TextMeshProUGUI>().text =
                subCommentStrings[i];
            subComment.GetComponent<RedCommentChildMonobehaviour>().
                Initialise(subCommentList[i], subCommentStrings[i], this);

            Color subCommentColor = Color.HSVToRGB(0.4f, 0.3f, 0.2f + 0.6f/
                subCommentDepth[i]);
            string subCommentColorName = string.Concat("CommentMaterial",
                subCommentDepth[i]);
            //Debug.Log(subCommentColorName + " " + subCommentColor + " " +
                subCommentMaterial);

            if(materialManager == null)
            {
                materialManager = GameObject.Find("MaterialManager").
                    GetComponent<MaterialManager>();
            }
            Material subCommentBackgroundMaterial = materialManager.
                InstantiateMaterialWithColor(subCommentColorName,
                subCommentColor, subCommentMaterial);
            subComment.transform.Find("ChildBackground").GetComponent<Image>
                ().material = subCommentBackgroundMaterial;
            subCommentObjectList.Add(subComment);
        }

    }

    void FitContents()
    {
        //canvas.SetActive(true);


        var mainRectTransform = mainText.GetComponent<RectTransform>();
        var mainHeight = mainRectTransform.sizeDelta[1];
        mainRectTransform.anchoredPosition = new Vector3(0, -
            mainRectTransform.sizeDelta[1] / 2, 0);


        var yOffsetTotal = 0.0f;
        for (int i = 0; i < subCommentList.Count; i++)
        {
            var subRectTransform = subCommentObjectList[i].GetComponent<RectTransform>();
            var subHeight = subRectTransform.sizeDelta[1];
            subRectTransform.anchoredPosition = new Vector3(0.1f, -(mainHeight + (subHeight / 2) + yMargin + yOffsetTotal), 0);

            yOffsetTotal += subHeight + yMargin;
        }

        var textContainerRect = textContainer.GetComponent<RectTransform>();
        var columnHeight = mainHeight + yOffsetTotal + 0.1f;
        textContainerRect.sizeDelta = new Vector2(frameWidth, columnHeight);
        textContainerRect.anchoredPosition = new Vector3(0, -columnHeight / 2, 0);

        frameGeometry.transform.localScale = new Vector3(frameWidth + frameMargin, columnHeight + frameMargin, 0.1f);
        frameGeometry.transform.localPosition = new Vector3(0, -columnHeight / 2, 0.1f);

        //Refresh total height
        parentPostController.RefreshCollisionHeight(yOffsetTotal);
    }

    List<Comment> RetrieveChildComments(Comment comment)
    {
        return comment.Replies;
    }

    void RetrieveAllChildComments(IList<Comment> comments, int depth)
    {
        if (comments != null)
        {
            foreach (Comment comment in comments)
            {
                subCommentStrings.Add(comment.UpVotes + " | u/" + comment.Author + "\n" + string.Concat(Enumerable.Repeat<string>(">", depth)) + " " + comment.Body);
                subCommentList.Add(comment);
                subCommentDepth.Add(depth);
                //Debug.Log(comment.replies.Count);
                if(comment != null) RetrieveAllChildComments(comment.replies, (depth + 1));
            }
        }
    }

    public void DeepRefreshComments()
    {
        if(!deepRefresh)
        {
            foreach(GameObject c in subCommentObjectList)
            {
                GameObject.Destroy(c);
            }
            subCommentObjectList.Clear();
            subCommentList.Clear();
            subCommentDepth.Clear();
            subCommentStrings.Clear();

            deepRefresh = true;

            RetrieveAllChildComments(mainComment.Comments.GetTop(limit:500),
                 1);
            UpdateText();
            Invoke("FitContents", 0.1f);

            refreshButton.GetComponent<RefreshCommentMonobehaviour>().
                SetRefreshed(true);
        }
    }


    public void Deactivate()
    {
        canvas.SetActive(false);
    }

    public void MovePlayer(Vector2 vector)
    {
        var velocity = scrollRect.velocity;
        playerMovementProvider.VertMove(-velocity.y);
    }

    public void ResizeVerticalFX(float height)
    {
        verticalFX.transform.localScale = new Vector3(2f, height, 1);
        verticalFX.transform.localPosition = new Vector3(0, -height/2, 0.1f)
            ;
    }

    public bool Refreshed()
    {
        return deepRefresh;
    }

    public void Vote(int vote)
    {
        if(vote == 1)
        {
            mainComment.UpvoteAsync();
            frameGeometry.GetComponent<ColorChanger>().SetColor(1);
        }
        else if(vote == -1)
        {
            mainComment.DownvoteAsync();
            frameGeometry.GetComponent<ColorChanger>().SetColor(2);
        }
        else if(vote == 0)
        {
            mainComment.UnvoteAsync();
            frameGeometry.GetComponent<ColorChanger>().SetColor(0);
        }
    }

    public void RetrieveVote()
    {
        VoteControllerMonobehaviour voteController = voteArrows.GetComponent
            <VoteControllerMonobehaviour>();
        if(mainComment.IsUpvoted)
        {
            voteController.SetVote(1);
            frameGeometry.GetComponent<ColorChanger>().SetColor(1);
        }
        else if(mainComment.IsDownvoted)
        {
            voteController.SetVote(-1);
            frameGeometry.GetComponent<ColorChanger>().SetColor(2);
        }
    }

    public int GetHiddenComments()
    {
        int numComments = subCommentList.Sum(x => x.NumReplies).Value;
        return numComments;
    }
}
```

5.3 Reddit Comment Child

Script attached to each child comment for finer control of attributes.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Reddit;
using Reddit.Controllers;
using TMPro;
using NaughtyAttributes;
using LogicUI.FancyTextRendering;

public class RedCommentChildMonobehaviour : MonoBehaviour
{
    private RedCommentFrameMonobehaviour parentCommentController;
    private Comment comment;
    private string commentString;

    //Geometry
    public TextMeshProUGUI textMeshComponent;
    public MarkdownRenderer markdownRendererComponent;
    public VoteControllerMonobehaviour voteController;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void Initialise(Comment c, string s, RedCommentFrameMonobehaviour
         parent)
    {
        comment = c;
        commentString = s;
        parentCommentController = parent;

        UpdateText(commentString);
        RetrieveVote();
    }

    public void UpdateText(string s)
    {
        markdownRendererComponent.Source = s;
    }

    public void Vote(int vote)
    {
        if(vote == 1)
        {
            comment.UpvoteAsync();
        }
        else if(vote == -1)
        {
            comment.DownvoteAsync();
        }
        else if(vote == 0)
        {
            comment.UnvoteAsync();
        }
    }

    public void RetrieveVote()
    {
        if(comment.IsUpvoted)
        {
            voteController.SetVote(1);
        }
        else if(comment.IsDownvoted)
        {
            voteController.SetVote(-1);
        }
    }
}
```

5.4 Vote Controller

Script attached to the upvote/downvote arrows that handle the API operation and toggle the arrow colors.

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

public class VoteControllerMonobehaviour : MonoBehaviour
{
    public RedPostMonobehaviour parentPostController;
    public RedCommentChildMonobehaviour parentCommentChildController;
    public RedCommentFrameMonobehaviour parentCommentController;
    public GameObject arrowUpvote;
    public GameObject arrowDownvote;
    private int vote = 0;

    // Start is called before the first frame update
    void Start()
    {

    }

    public void ToggleUpvote(bool live = true)
    {
        if(vote != 1)
        {
            vote = 1;
            arrowDownvote.GetComponent<ColorChanger>().SetColor(0);
            arrowUpvote.GetComponent<ColorChanger>().SetColor(1);

            if(parentPostController != null && live) parentPostController.
                Vote(1);
            if(parentCommentChildController != null && live)
                parentCommentChildController.Vote(1);
            if(parentCommentController != null && live)
                parentCommentController.Vote(1);
        }
        else
        {
            vote = 0;
            arrowUpvote.GetComponent<ColorChanger>().SetColor(0);
            arrowDownvote.GetComponent<ColorChanger>().SetColor(0);

            if(parentPostController != null && live) parentPostController.
                Vote(0);
            if(parentCommentChildController != null && live)
                parentCommentChildController.Vote(0);
            if(parentCommentController != null && live)
                parentCommentController.Vote(0);
        }
    }

    public void ToggleDownvote(bool live = true)
    {
        if(vote != -1)
        {
            vote = -1;
            arrowUpvote.GetComponent<ColorChanger>().SetColor(0);
            arrowDownvote.GetComponent<ColorChanger>().SetColor(1);

            if(parentPostController != null && live) parentPostController.
                Vote(-1);
            if(parentCommentChildController != null && live)
                parentCommentChildController.Vote(-1);
            if(parentCommentController != null && live)
                parentCommentController.Vote(-1);
        }
        else
        {
            vote = 0;
            arrowDownvote.GetComponent<ColorChanger>().SetColor(0);
            arrowUpvote.GetComponent<ColorChanger>().SetColor(0);

            if(parentPostController != null && live) parentPostController.
                Vote(0);
            if(parentCommentChildController != null && live)
                parentCommentChildController.Vote(0);
            if(parentCommentController != null && live)
                parentCommentController.Vote(0);
        }
    }

    public void HoverUpvote()
    {
        //Debug.Log("hover upvote");
        if(vote != 1)
        {
            arrowUpvote.GetComponent<ColorChanger>().SetColor(2);
        }
    }

    public void HoverDownvote()
    {
        //Debug.Log("hover downvote");
        if(vote != -1)
        {
            arrowDownvote.GetComponent<ColorChanger>().SetColor(2);
        }
    }

    public void HoverOutUpvote()
    {
        if(vote == 1)
        {
            arrowUpvote.GetComponent<ColorChanger>().SetColor(1);
        }
        else
        {
            arrowUpvote.GetComponent<ColorChanger>().SetColor(0);
        }
    }

    public void HoverOutDownvote()
    {
        if(vote == -1)
        {
            arrowDownvote.GetComponent<ColorChanger>().SetColor(1);
        }
        else
        {
            arrowDownvote.GetComponent<ColorChanger>().SetColor(0);
        }
    }

    public void ResetColors()
    {
        if(vote == 0)
        {
            arrowUpvote.GetComponent<ColorChanger>().SetColor(0);
            arrowDownvote.GetComponent<ColorChanger>().SetColor(0);
        }
        else if(vote == 1)
        {
            arrowUpvote.GetComponent<ColorChanger>().SetColor(1);
            arrowDownvote.GetComponent<ColorChanger>().SetColor(0);
        }
        else if(vote == -1)
        {
            arrowUpvote.GetComponent<ColorChanger>().SetColor(0);
            arrowDownvote.GetComponent<ColorChanger>().SetColor(1);
        }
    }

    public void SetVote(int i)
    {
        vote = i;
        Invoke("ResetColors", 0.1f);
    }
}
```

5.5 Refresh Comments

Script attached to the refresh arrow that handles the refresh operation.

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

public class RefreshCommentMonobehaviour : MonoBehaviour
{

    public RedPostMonobehaviour parentPostController;
    public RedCommentFrameMonobehaviour parentCommentFrameController;
    public GameObject refreshText;
    public GameObject refreshArrow;
    private bool refreshed = false;


    // Start is called before the first frame update
    void Start()
    {
        if(parentCommentFrameController != null)
        {
            //refreshText.GetComponent<TextMeshPro>().text = "Load More
                Comments\n>" + parentCommentFrameController.
                GetHiddenComments();
        }
        else if(parentPostController != null)
        {
            //refreshText.GetComponent<TextMeshPro>().text = "Load All
                Comments";
        }
        refreshText.SetActive(false);
    }

    public void Refresh()
    {
        if(parentCommentFrameController != null)
        {
            RefreshComments();
            SetRefreshed(true);
        }
        else if(parentPostController != null)
        {
            RefreshAllPostComments();
            SetRefreshed(true);
        }
    }

    public void RefreshComments()
    {
        if(!parentCommentFrameController.Refreshed())
        {
            parentCommentFrameController.DeepRefreshComments();
        }
    }
    public void RefreshAllPostComments()
    {
        parentPostController.DeepRefreshAllComments();
    }

    public void SetRefreshed(bool x)
    {
        refreshed = x;
        refreshArrow.GetComponent<ColorChanger>().SetColor(2);
        refreshText.SetActive(false);
        refreshArrow.SetActive(false);
    }

    public void HoverIn()
    {
        if(!refreshed)
        {
            refreshText.SetActive(true);
            refreshArrow.GetComponent<ColorChanger>().SetColor(1);
        }
    }

    public void HoverOut()
    {
        refreshText.SetActive(false);
        if(!refreshed)
        {
            refreshArrow.GetComponent<ColorChanger>().SetColor(0);
        }
    }
}
```

6 General Support Scripts

Some general scripts and libraries to handle miscellaneous operations, mostly regarding procedural geometry.

6.1 Scroll Surface

Key script that handles the reverse-scrolling of the player - uses an invisible helper scroll rectangle that moves the player in the opposite direction when scrolled.

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using Reddit;
using Reddit.Controllers;
using TMPro;
using UnityEngine.UI;

public class RedScrollSurface : MonoBehaviour
{

    //PLAYER
    private GameObject rig;
    private MovementProvider playerMovementProvider;

    //GAMEOBJECT / GEOMETRY
    private Camera XRcamera;
    public GameObject cameraCanvas;
    public GameObject scrollRectContainer;
    public GameObject scaleReference = null;
    private ScrollRect scrollRect;

    //PARAMETERS
    private float frameWidth = 2.4f;
    private float frameMargin = 0.2f;
    private float yMargin = 0.2f;


    private Vector2 previousVector2 = new Vector2(0, 0);


    // Start is called before the first frame update
    void Start()
    {
        XRcamera = GameObject.Find("VR Camera Controller").GetComponent<
            Camera>();
        rig = GameObject.Find("XR Rig");
        playerMovementProvider = rig.GetComponent<MovementProvider>();
        cameraCanvas.GetComponent<Canvas>().worldCamera = XRcamera;
        scrollRect = scrollRectContainer.GetComponent<ScrollRect>();

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void MovePlayer(Vector2 vector)
    {
        var velocity = scrollRect.velocity;
        if(scaleReference != null)
        {
            velocity = velocity * scaleReference.transform.localScale.y;
        }
        playerMovementProvider.VertMove(-velocity.y);
    }

    public void SnapPlayer()
    {

    }
}
```

6.2 Frame Dynamic

Creates the 3D frame around certain UI elements to make them less flat.

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


public class FrameDynamic : MonoBehaviour
{
    public float frameThickness = 0.1f;
    public float frameDepth = 0f;
    public Material frameMaterial;
    public List<Material> frameMaterials;
    private List<GameObject> frameObjects;
    private GameObject frameUpper;
    private GameObject frameRight;
    private GameObject frameLower;
    private GameObject frameLeft;

    // Start is called before the first frame update
    void Start()
    {
        CreateFrame();
    }

    // Update is called once per frame
    void Update()
    {

    }

    void CreateFrame()
    {
        if (frameUpper != null || frameRight != null || frameLower != null || frameLeft != null)
        {
            return;
        }

        frameMaterials.Insert(0, frameMaterial);

        frameUpper = GameObject.CreatePrimitive(PrimitiveType.Cube);
        frameRight = GameObject.CreatePrimitive(PrimitiveType.Cube);
        frameLower = GameObject.CreatePrimitive(PrimitiveType.Cube);
        frameLeft = GameObject.CreatePrimitive(PrimitiveType.Cube);

        frameUpper.transform.SetParent(transform);
        frameRight.transform.SetParent(transform);
        frameLower.transform.SetParent(transform);
        frameLeft.transform.SetParent(transform);

        RefreshFrame();
    }

    public void RefreshFrame()
    {
        if (frameUpper == null || frameRight == null || frameLower == null || frameLeft == null)
        {
            CreateFrame();
        }
        frameUpper.transform.localRotation = Quaternion.identity;
        frameRight.transform.localRotation = Quaternion.identity;
        frameLower.transform.localRotation = Quaternion.identity;
        frameLeft.transform.localRotation = Quaternion.identity;

        frameUpper.transform.localPosition = new Vector3(0, (transform.localScale.y / 2 + frameThickness / 2) / transform.localScale.y, 0);
        frameRight.transform.localPosition = new Vector3(- (transform.localScale.x / 2  + frameThickness / 2) / transform.localScale.x, 0, 0);
        frameLower.transform.localPosition = new Vector3(0, - (transform.localScale.y / 2  + frameThickness / 2) / transform.localScale.y, 0);
        frameLeft.transform.localPosition = new Vector3((transform.localScale.x / 2  + frameThickness / 2) / transform.localScale.x
            , 0, 0);

        frameUpper.transform.localScale = new Vector3( (transform.localScale
            .x + 2 * frameThickness) / transform.localScale.x,
            frameThickness / transform.localScale.y, (transform.localScale.z
             + frameDepth) / transform.localScale.z);
        frameRight.transform.localScale = new Vector3(frameThickness /
            transform.localScale.x, (transform.localScale.y) / transform.
            localScale.y, (transform.localScale.z + frameDepth) / transform.
            localScale.z);
        frameLower.transform.localScale = new Vector3( (transform.localScale
            .x + 2 * frameThickness) / transform.localScale.x,
            frameThickness / transform.localScale.y, (transform.localScale.z
             + frameDepth) / transform.localScale.z);
        frameLeft.transform.localScale = new Vector3(frameThickness /
            transform.localScale.x, (transform.localScale.y) / transform.
            localScale.y, (transform.localScale.z + frameDepth) / transform.
            localScale.z);

        SetFrameMaterial(0);
    }

    public void SetFrameMaterial(int i = 0)
    {
        if (frameUpper == null || frameRight == null || frameLower == null
            || frameLeft == null)
        {
            CreateFrame();
        }
        frameUpper.GetComponent<MeshRenderer>().material = frameMaterials[i]
            ;
        frameRight.GetComponent<MeshRenderer>().material = frameMaterials[i]
            ;
        frameLower.GetComponent<MeshRenderer>().material = frameMaterials[i]
            ;
        frameLeft.GetComponent<MeshRenderer>().material = frameMaterials[i];
    }

    public void ReplaceBaseMaterial(Material mat)
    {
        if (frameUpper == null || frameRight == null || frameLower == null
            || frameLeft == null)
        {
            CreateFrame();
        }
        frameMaterials[0] = mat;
    }
}
```

6.3 Frame Dynamic Planes

Same as frame dynamic but for planar elements.

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


public class FrameDynamicPlanes : MonoBehaviour
{
    public float frameDepth = 0f;
    public Material frameMaterial;
    private List<GameObject> frameObjects;
    private GameObject frameUpper;
    private GameObject frameRight;
    private GameObject frameLower;
    private GameObject frameLeft;

    // Start is called before the first frame update
    void Start()
    {
        CreateFrame();
    }

    // Update is called once per frame
    void Update()
    {

    }

    public void RefreshFrame()
    {
        if (frameUpper == null || frameRight == null || frameLower == null
            || frameLeft == null)
        {
            return;
        }
        frameUpper.transform.localRotation = Quaternion.Euler(270, 90, 0);
        frameRight.transform.localRotation = Quaternion.Euler(180, 90, 0);
        frameLower.transform.localRotation = Quaternion.Euler(90, 90, 0);
        frameLeft.transform.localRotation = Quaternion.Euler(0, 90, 0);

        frameUpper.transform.localPosition = new Vector3(0, (transform.
            localScale.y / 2) / transform.localScale.y, 0);
        frameRight.transform.localPosition = new Vector3(- (transform.
            localScale.x / 2) / transform.localScale.x, 0, 0);
        frameLower.transform.localPosition = new Vector3(0, - (transform.localScale.y / 2 ) / transform.localScale.y, 0);
        frameLeft.transform.localPosition = new Vector3((transform.localScale.x / 2) / transform.localScale.x, 0, 0);

        frameUpper.transform.localScale = new Vector3( (transform.localScale.x + frameDepth) / transform.localScale.x, 1, 1);
        frameRight.transform.localScale = new Vector3( (transform.localScale.x + frameDepth) / transform.localScale.x, 1, 1);
        frameLower.transform.localScale = new Vector3( (transform.localScale.x + frameDepth) / transform.localScale.x, 1, 1);
        frameLeft.transform.localScale = new Vector3( (transform.localScale.x + frameDepth) / transform.localScale.x, 1, 1);

        frameUpper.GetComponent<MeshRenderer>().material = frameMaterial;
        frameRight.GetComponent<MeshRenderer>().material = frameMaterial;
        frameLower.GetComponent<MeshRenderer>().material = frameMaterial;
        frameLeft.GetComponent<MeshRenderer>().material = frameMaterial;
    }

    void CreateFrame()
    {
        frameUpper = GameObject.CreatePrimitive(PrimitiveType.Quad);
        frameRight = GameObject.CreatePrimitive(PrimitiveType.Quad);
        frameLower = GameObject.CreatePrimitive(PrimitiveType.Quad);
        frameLeft = GameObject.CreatePrimitive(PrimitiveType.Quad);

        frameUpper.transform.SetParent(transform);
        frameRight.transform.SetParent(transform);
        frameLower.transform.SetParent(transform);
        frameLeft.transform.SetParent(transform);

        RefreshFrame();
    }
}
```

6.4 Text Frame

Some stuff to handle text panels.

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

public class RedTextFrameMonobehaviour : MonoBehaviour
```

```
{

    public GameObject frameGeometry;
    public GameObject textContainer;
    public GameObject mainText;
    public AlignEnum alignment;

    //PARAMETERS
    public float frameWidth = 4f;
    public float frameMargin = 0.2f;

    public enum AlignEnum
    {
        Top,
        Bottom
    };

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void Initialise()
    {
        Invoke("FitContents", 0.1f);
    }

    void FitContents()
    {

        var mainTextRect = mainText.GetComponent<RectTransform>();
        var mainHeight = mainTextRect.sizeDelta[1];

        var textContainerRect = textContainer.GetComponent<RectTransform>();
        textContainerRect.sizeDelta = new Vector2(frameWidth, mainHeight);

        frameGeometry.transform.localScale = new Vector3(frameWidth +
            frameMargin, mainHeight + frameMargin, 0.1f);


        if(alignment.ToString() == "Bottom")
        {
            textContainerRect.anchoredPosition = new Vector3(0, mainHeight / 2, 0);
            mainTextRect.anchoredPosition = new Vector3(0, mainHeight / 2, 0);
            frameGeometry.transform.localPosition = new Vector3(0, - (1 - mainHeight / 2), 0.051f);
            //transform.localPosition += new Vector3(0, 2f, 0); //moves entire gameobject to correct position
        }
        else
        {
            textContainerRect.anchoredPosition = new Vector3(0, - mainHeight / 2, 0);
            mainTextRect.anchoredPosition = new Vector3(0, - mainHeight / 2, 0);
            frameGeometry.transform.localPosition = new Vector3(0, (1 - mainHeight / 2), 0.051f);
        }


        /*
        var yOffsetTotal = 0.0f;
        for (int i = 0; i < subCommentList.Count; i++)
        {
            var subRectTransform = subCommentList[i].GetComponent<RectTransform>();
            var subHeight = subRectTransform.sizeDelta[1];
            subRectTransform.anchoredPosition = new Vector3(0, -(mainHeight + (subHeight / 2) + yMargin + yOffsetTotal), 0);

            yOffsetTotal += subHeight + yMargin;
        }

        var textContainerRect = textContainer.GetComponent<RectTransform>();
        var columnHeight = mainHeight + yOffsetTotal + yMargin;
        textContainerRect.sizeDelta = new Vector2(frameWidth, columnHeight);
        textContainerRect.anchoredPosition = new Vector3(0, -columnHeight / 2, 0);

        frameGeometry.transform.localScale = new Vector3(frameWidth + frameMargin, columnHeight + frameMargin, 0.1f);
        frameGeometry.transform.localPosition = new Vector3(0, -columnHeight / 2, 0.1f);

        //transform.localPosition += new Vector3(0, mainHeight + 0.5f, 0);
        if(!deepRefresh) transform.localPosition += new Vector3(0, 2f, 0);
        */
    }
}
```

6.5 Material Manager

Important script handling all dynamic material instances - ensures that procedural material generation goes through a central dictionary and uses the same materials if already instantiated, instead of creating a new instance for 10000s of comments.

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

public class MaterialManager : MonoBehaviour
{

    public Material defaultMaterial;
    public Material defaultNeon;
    public Material skyBox;
    public int dayInterval;
    private Dictionary<string, Material> materials = new Dictionary<string,
        Material>();

    // Start is called before the first frame update
    void Start()
    {

    }

    void Update()
    {
        AdjustDayNight();
    }

    public Material InstantiateMaterialWithColor(string materialName, Color
        materialColor, Material material = null)
    {
        //Debug.Log("Instantiating material - " + materialName);
        if(material == null)
        {
            material = defaultMaterial;
        }

        if(!materials.ContainsKey(materialName))
        {
            Material newMaterial = Material.Instantiate(material);
            newMaterial.SetColor("_BaseColor", materialColor);
            Debug.Log("Instantiating new material - " + materialName);

            materials.Add(materialName, newMaterial);
        }

        return materials[materialName];
    }

    public Material InstantiateMaterialWithEmission(string materialName,
        Color materialColor, float intensity, Material material = null)
    {
        //Debug.Log("Instantiating material - " + materialName);
        if(material == null)
        {
            material = defaultNeon;
        }

        if(!materials.ContainsKey(materialName))
        {
            Color emissionColor = new Vector4(materialColor.r * intensity,
                materialColor.g * intensity, materialColor.b * intensity, 1.
                0f);
            Material newMaterial = Material.Instantiate(material);
            newMaterial.SetColor("_EmissionColor", emissionColor);
            //Debug.Log("Instantiating new material - " + materialName);

            materials.Add(materialName, newMaterial);
        }

        return materials[materialName];
    }

    public void AdjustDayNight()
    {
        float currentTime = DateTime.Now.Minute + DateTime.Now.Second/60f;
        float scaledDay = (Mathf.Abs(currentTime / 59f - 0.5f) - 0.25f) * 3;

        float exposure = 1.1f + scaledDay;

        RenderSettings.skybox.SetFloat("_Exposure", exposure);
    }
}
```

6.6 Lib Color

I wanted more stuff in here but it's just a random color function at the moment that takes in HSV values (easier to control result).

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

public static class LibColor
{

    public static Color RandomiseHSV(Color inputColor, float varianceHue=0.0
        5f, float varianceSaturation=0.05f, float varianceValue=0.05f )
    {
        float[] HSV = { 0, 0, 0 };
        Color.RGBToHSV(inputColor, out HSV[0], out HSV[1], out HSV[2]);
        HSV[0] = Random.Range(HSV[0] - varianceHue, HSV[0] + varianceHue);
        HSV[1] = Random.Range(HSV[1] - varianceSaturation, HSV[1] +
            varianceSaturation);
        HSV[2] = Random.Range(HSV[2] - varianceValue, HSV[2] + varianceValue
            );

        Color newColor = Color.HSVToRGB(HSV[0], HSV[1], HSV[2]);
        return newColor;
    }


}
```

6.7 ColorRandomiser

Monobehaviour script that makes use of the aforementioned color randomising function.

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

public class ColorRandomiser : MonoBehaviour
{
    public Color platformColor;
    public float varianceHue = 0.05f;
    public float varianceSaturation = 0.05f;
    public float varianceValue = 0.05f;
    private float[] HSV = { 0, 0, 0 };


    // Start is called before the first frame update
    void Start()
    {
        Color.RGBToHSV(platformColor, out HSV[0], out HSV[1], out HSV[2]);
        HSV[0] = Random.Range(HSV[0] - varianceHue, HSV[0] + varianceHue);
        HSV[1] = Random.Range(HSV[1] - varianceSaturation, HSV[1] +
            varianceSaturation);
        HSV[2] = Random.Range(HSV[2] - varianceValue, HSV[2] + varianceValue
            );

        //Change platform material color
        Color newColor = Color.HSVToRGB(HSV[0], HSV[1], HSV[2]);
        GetComponent<MeshRenderer>().material.color = newColor;
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

6.8 ColorChanger

Clever universal script that allows you to set specific materials to an array that is unique to this component instance, then call their indices through another script to change object materials.

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

public class ColorChanger : MonoBehaviour
{
    Material redMat;
    Material blueMat;
    Material greenMat;
    Material clearMat;
    Material defaultMat;
    public List<Material> materialList = new List<Material>();

    void Start()
    {
        defaultMat = GetComponent<MeshRenderer>().material;
        materialList.Insert(0, defaultMat);

        redMat = Resources.Load("Materials/Select-Red", typeof(Material)) as Material;
        blueMat = Resources.Load("Materials/Select-Blue", typeof(Material)) as Material;
        greenMat = Resources.Load("Materials/Select-Green", typeof(Material)) as Material;
        clearMat = Resources.Load("Materials/Select-Clear", typeof(Material)) as Material;
    }

    public void SetMaterialDefault()
    {
        GetComponent<MeshRenderer>().material = defaultMat;
    }
    public void SetColorRed()
    {
        GetComponent<MeshRenderer>().material = redMat;
    }

    public void SetColorBlue()
    {
        GetComponent<MeshRenderer>().material = blueMat;
    }

    public void SetColorGreen()
    {
        GetComponent<MeshRenderer>().material = greenMat;
    }

    public void SetColorClear()
    {
        GetComponent<MeshRenderer>().material = clearMat;
    }

    public void SetColor(int colorIndex)
    {
        if(colorIndex < materialList.Count)
        {
            GetComponent<MeshRenderer>().material = materialList[colorIndex];
```

```
                }
        }
}
```