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

/**
 * This script is the motor for the UFO. It controls all of the UFO's actions including attacks,
 * animations, and some ui. This script communicates the sound data from SoundCapture to
 * all of the UFO's lights in order to be analyzed and animated. It also communicates to PlayerMotor
 * and ScoreMotor when music starts playing and when it ends.
 *
 * @author Alleck Henrie
 */

public class UFOMotor : MonoBehaviour
{
    // Fields
    public GameObject player;
    public GameObject orb;
    public GameObject wall;

    private GameObject newLight;

    public Animator ui;

    public List<AudioSyncer> lights;

    private Dictionary<GameObject, List<GameObject>> orbs;

    public ScoreMotor scoreManager;

    private PlayerMotor playerMotor;

    private bool ascending = true;
    private bool switchDir = false;
    private bool setup = false;
    private bool orbSpawn = false;
    private bool attacking = false;
    private bool intro = true;
    private bool visualize = false;
    private bool tele = false;
    private bool scoresGone = false;
    private bool visualizer = false;

    public float rotateSpeed;
    public float riseSpeed;
    public float switchSpeed;
    public float maxLower;
    public float maxRaise;
```

```
public float teleStr;
public float teleSpd;
public float xVary;
public float yVaryMin;
public float yVaryMax;

private float jumpStr;
private float jumpTime = 0.0f;
private float minHeight;
private float maxHeight;
private float originalHeight;
private float prevSpeed;
private float maxFall = 0.0f;
private float maxRise = 0.0f;
private float newX;
private float newY;
private float newZ;

private int musicPlaying = 0;

private Vector3 prevPos;

// We will use SoundCapture to fill this with data
public static float[] m_audioSpectrum = new float[0];

private void Update()
{
   // if the UFO is not setup
   if (!setup)
   {
      // initialize the UFO
      Setup();
   }

   // Capture the computer's audio
   CaptureAudio();

   // Have the UFO play it's Idle animation
   Idle();

   // if tele,
   if (tele)
   {
      // calculate new location and lerp to said location
      Tele();
   }

   // if the player activates visualizer mode,
```

```
    if (playerMotor.Visualize() && !visualizer)
    {
        // reset UFO actions for visualizer mode.
        Visualizer(true);
    }

    // if the player does not have visualizer mode activated,
    if (!playerMotor.Visualize())
    {
        // reset UFO actions for game mode.
        Visualizer(false);
    }

    // if there is no music playing,
    if (!IsPlaying())
    {
        // set the UFO to idle.
        AudioPlaying(false);
    }

    // if there is music playing,
    if (IsPlaying())
    {
        // start the Run.
        AudioPlaying(true);
    }
}

// This method will initialize the UFO
private void Setup()
{
    setup = true;
    // UI Default
    ui.Play("NoAudio");

    //set playerMotor to the player's playerMotor script
    playerMotor = player.GetComponent<PlayerMotor>();

    // initialize data list
    m_audioSpectrum = new float[1024];

    // Used for UFO idle
    originalHeight = gameObject.transform.position.y;
    minHeight = originalHeight - maxLower;
    maxHeight = originalHeight + maxRaise;

    // List of orbs currently controlled by the UFO
    orbs = new Dictionary<GameObject, List<GameObject>>();
```

```
    // Used to properly calculate if the ufo is currently rising or falling,
    // and set the maxRise and maxFall speeds
    if (riseSpeed > 0)
    {
        maxRise = riseSpeed;
        maxFall = -riseSpeed;
        ascending = true;
    }
    else if (riseSpeed < 0)
    {
        maxRise = -riseSpeed;
        maxFall = riseSpeed;
        ascending = false;
    }
}

// UFO idle animation, call in Update()
private void Idle()
{
    // UFO rotation
    gameObject.transform.Rotate(0.0f, rotateSpeed, 0.0f, Space.Self);
    gameObject.transform.position += new Vector3(0.0f, riseSpeed * Time.deltaTime, 0.0f);

    // if the UFO is lower than the minimum height, switch directions
    if (gameObject.transform.position.y <= minHeight && !switchDir)
    {
        prevSpeed = riseSpeed;
        switchDir = true;
    }

    // if the UFO is higher than the maximum height, switch directions
    if (gameObject.transform.position.y >= maxHeight && !switchDir)
    {
        prevSpeed = riseSpeed;
        switchDir = true;
    }

    // Direction switch animation for the UFO in the y-axis
    if (switchDir)
    {
        // if ascending, slowly switch riseSpeed to maxRise speed
        if (ascending)
        {
            riseSpeed += switchSpeed;

            if (riseSpeed > maxRise)
            {
```

```
            riseSpeed = maxRise;
            switchDir = false;
            ascending = true;
        }
    }

    // if descending, slowly switch riseSpeed to maxFall speed
    if (!ascending)
    {
        riseSpeed -= switchSpeed;

        if (riseSpeed < maxFall)
        {
            riseSpeed = maxFall;
            switchDir = false;
            ascending = true;
        }
    }
  }
}

// Jump the UFO to a new location
private void Tele()
{
  jumpTime += Time.deltaTime;

  // newPos is calculated using newX, newY, and newZ from the action() method
  Vector3 newPos = new Vector3(newX, newY, newZ);

  gameObject.transform.position = Vector3.Lerp(prevPos, newPos, jumpTime / teleSpd);

  // recalculate minHeight and maxHeight based off new location
  if (gameObject.transform.position == newPos)
  {
    minHeight = newPos.y - maxLower;
    maxHeight = newPos.y + maxRaise;

    tele = false;
    jumpTime = 0.0f;
  }
}

// Put the UFO into or take it out of visualizer mode
private void Visualizer(bool visOn)
{
  if (visOn && !visualizer)
  {
    List<List<GameObject>> orbsToRemove = new List<List<GameObject>>();
```

```
        foreach (List<GameObject> orbLists in orbs.Values)
        {
            foreach (GameObject orb in orbLists)
            {
                Destroy(orb);
            }
        }

        orbs = new Dictionary<GameObject, List<GameObject>>();
        visualize = false;
        attacking = false;
        musicPlaying = 0;

        // if the scores are showing, remove scores
        if (!scoresGone)
        {
            ui.Play("RemovePrevScores");
            scoresGone = true;
        }

        visualizer = true;
    }
    else if (!visOn)
    {
        // Default UI
        ui.Play("NoAudio");

        List<List<GameObject>> orbsToRemove = new List<List<GameObject>>();

        foreach (List<GameObject> orbLists in orbs.Values)
        {
            foreach (GameObject orb in orbLists)
            {
                Destroy(orb);
            }
        }

        orbs = new Dictionary<GameObject, List<GameObject>>();
        scoresGone = false;
        attacking = false;
        intro = true;
        visualize = false;
        musicPlaying = 0;

        visualizer = false;
    }
}
```

```
// If there is audio detected, the UFO can start animating and attacking
// else the run is over, return to intro.
private void AudioPlaying(bool audio)
{
   if (audio)
   {
      // if attacking and visualizer mode is disabled
      if (attacking && !playerMotor.Visualize())
      {
         // if intro, play game start
         if (intro)
         {
            ui.Play("RUN");
            scoresGone = true;
         }

         // once the animation is over, the intro is over
         if (!ui.GetCurrentAnimatorStateInfo(0).IsName("RUN"))
            intro = false;
      }

      // if spawning an Orb,
      if (orbSpawn)
      {
         // spawn an Orb
         SpawnOrb();
      }
   }

   else
   {
      attacking = false;

      // if the UFO is not in visualizer mode, Default UI
      if (!visualizer)
         ui.Play("NoAudio");

      List<List<GameObject>> orbsToRemove = new List<List<GameObject>>();

      foreach (List<GameObject> orbLists in orbs.Values)
      {
         foreach (GameObject orb in orbLists)
         {
            Destroy(orb);
         }
      }
```

```
        orbs = new Dictionary<GameObject, List<GameObject>>();

        // if not visualizer mode, scores will show up
        if (!playerMotor.Visualize())
            scoresGone = false;

        intro = true;
    }
}

// Spawn and initialize an Orb
private void SpawnOrb()
{
    GameObject go = Instantiate(orb, newLight.transform);
    go.SetActive(true);
    go.GetComponent<OrbMotor>().setLight(newLight);

    if (!orbs.ContainsKey(newLight))
    {
        List<GameObject> newList = new List<GameObject>();
        orbs.Add(newLight, newList);
    }

    orbs[newLight].Add(go);

    for (int i = 0; i < orbs[newLight].Count; i++)
        orbs[newLight][i].GetComponent<OrbMotor>().sendBeat();

    orbSpawn = false;
}

// Get spectrum data from SoundCapture and fill the data into m_audioSpectrum
private void CaptureAudio()
{
    m_audioSpectrum = SoundCapture.barData;
}

// Set the UFO into visualizer mode
public void Visualize()
{
    visualize = true;
}

// Check if the UFO is in visualizer mode
// return bool
public bool GetVisualize()
{
    return visualize;
```

```
    }

    // This method is designed to tell the UFO what to do
    public void action(string newAction, GameObject light)
    {
        // Used to teleport the UFO into a random but closer location to the player
        if (newAction == "Teleport")
        {
            attacking = true;

            // Each jump is a random strength between 1/3 of teleStr, to 3/3 teleStr
            jumpStr = Random.Range(teleStr * 0.33f, teleStr);

            // Formulas to calculate new direction to jump to
            float theta = Mathf.Atan(Mathf.Abs((player.transform.position.z -
gameObject.transform.position.z)) / Mathf.Abs((player.transform.position.x -
gameObject.transform.position.x)));
            float xJump = Random.Range(-xVary, xVary) * jumpStr;
            float zJump = Mathf.Sqrt((jumpStr * jumpStr) - (xJump * xJump));
            float delta = Mathf.Atan(xJump / zJump);

            // using the direction, calculate the position with jumpStr
            newX = jumpStr * Mathf.Cos(theta + delta);
            newZ = jumpStr * Mathf.Sin(theta + delta);
            newY = Random.Range(yVaryMin + player.transform.position.y, (yVaryMax) +
player.transform.position.y);

            // direction calculation loses negative values, change to - if need
            if (player.transform.position.x < gameObject.transform.position.x)
                newX *= -1;

            // direction calculation loses negative values, change to - if need
            if (player.transform.position.z < gameObject.transform.position.z)
                newZ *= -1;

            // change to be in relation of the player's current position
            newX += gameObject.transform.position.x;
            newZ += gameObject.transform.position.z;

            // will be used in another block in the Update method
            prevPos = gameObject.transform.position;

            // Teleport UFO
            tele = true;
        }

        // Used to spawn an orb from the UFO
        else if (newAction == "SpawnOrb")
```

```
        {
            attacking = true;
            orbSpawn = true;
            newLight = light;
        }

        // Used to spawn a wall in front of the character
        else if (newAction == "SpawnWall")
        {
            attacking = true;

            // Formula used to calulcate a random position in front of the character to spawn the wall
            Vector3 wallSpawn = new Vector3(playerMotor.getWall().transform.position.x + Random.Range(-
10, 10), 0, playerMotor.getWall().transform.position.z + Random.Range(-10, 10));

            // Instantiate a wall at the new location
            GameObject go = Instantiate(wall, wallSpawn, Quaternion.identity);
            go.SetActive(true);
            go.transform.parent = null;
        }

        // Used to slow down the player's maximum speed
        else if (newAction == "SlowDown")
        {
            attacking = true;
            playerMotor.slowDown();
        }

        // action is not valid
        else
            return;
    }

    // Used to remove an orb from the UFO orbs list
    public void removeOrb(GameObject light, GameObject orb)
    {
        orbs[light].Remove(orb);
        scoreManager.orbAvoided();
    }

    // Used to inform the UFO on when a light is starting to listen music
    public void lightPlaying()
    {
        musicPlaying++;
    }

    // Used to inform the UFO when a light is not hearing music anymore
    public void lightOff()
```

```
    {
        musicPlaying--;
    }

    // Is the UFO hearing music
    // return bool
    public bool IsPlaying()
    {
        // if there are any lights listening to music, return true
        // else false
        return musicPlaying > 0;
    }

    // Is the UFO currently attacking
    // return bool
    public bool IsAttacking()
    {
        return attacking;
    }

    // Disable visualizer mode on the UFO
    public void StopVisualizing()
    {
        visualizer = false;
    }
}
```