```
1  using UnityEngine;
2
3  public class Chunk : MonoBehaviour
4  {
5      // Main chunk generation script that is in use
6      public static int chunkSize = 16; // Size of each chunk in units
7      public static float steepScale = 10f, steepAmplitude = 20f, amplitude =
           3f, scale = 2f, yOffset = -100f; // Related perlin constants
8      [SerializeField] Material sand; // Ground sand material
9
10     public GameObject CreateChunk(Vector2Int chunkPos) // Creates a single
         chunk at a given chunk position (different from player position)
11     {
12         Vector2 offset = new Vector2(chunkPos.x * chunkSize, chunkPos.y *
             chunkSize);
13
14         int chunkIndex = 0;
15
16         GameObject chunk;
17
18         Mesh chunkMesh;
19
20         MeshFilter chunkMeshFilter;
21
22         MeshRenderer chunkMeshRenderer;
23
24         CombineInstance[] combineInstance;
25
26         combineInstance = new CombineInstance[chunkSize * chunkSize];
27         chunk = new GameObject("Chunk(" + chunkPos.x + ", " + chunkPos.y +
             ")");
28
29         for (int z = 0; z < chunkSize; z++) // Here a bunch of quads
             (singular 1x1 planes) are created
30         {
31             for (int x = 0; x < chunkSize; x++)
32             {
33                 CreateQuad(new Vector3(x + offset.x, z + offset.y),
                       chunkIndex, combineInstance, chunk);
34                 chunkIndex++;
35             }
36         }
37
38         // After creating each individual quad, they must be combined
             together for optimization and treated as a single mesh
39         chunkMesh = new Mesh();
40         chunkMeshFilter = chunk.AddComponent<MeshFilter>();
41         chunkMeshFilter.mesh.Clear();
42         chunkMeshFilter.mesh = chunkMesh;
```

```
43          chunkMeshRenderer = chunk.AddComponent<MeshRenderer>();
44          chunkMesh.CombineMeshes(combineInstance);
45          chunkMeshRenderer.material = sand;
46          chunk.AddComponent<MeshCollider>();
47          chunkMeshFilter.mesh.RecalculateNormals();
48          chunkMeshFilter.mesh.Optimize();
49
50          foreach (Transform t in chunk.transform) // Destroys all previous
              quads as they were a reference and thus no longer needed
51          {
52              Destroy(t.gameObject);
53          }
54
55          return chunk;
56      }
57
58      // In short, makes a singular 1x1 plane mesh by adjusting the vertices,
           triangles, and uv values
59      private void CreateQuad(Vector2 pos, int index, CombineInstance[]
          combine, GameObject chunkObject)
60      {
61          GameObject quad = new GameObject("Quad", typeof(MeshFilter), typeof
              (MeshRenderer));
62          quad.transform.SetParent(chunkObject.transform);
63          Mesh mesh = new Mesh();
64          MeshFilter mf = quad.GetComponent<MeshFilter>();
65          MeshRenderer mr = quad.GetComponent<MeshRenderer>();
66          mf.mesh = mesh;
67          Vector3[] vertices = new Vector3[]
68          {
69              new Vector3(pos.x - 0.5f, yOffset + (amplitude *
                  Mathf.PerlinNoise((float)(pos.x - 0.5f) / chunkSize * scale,
                  (float)(pos.y - 0.5f) / chunkSize * scale)) /*+
                  (steepAmplitude * Mathf.PerlinNoise((float)(pos.x - 0.5f) /
                  chunkSize * steepScale, (float)(pos.y - 0.5f) / chunkSize *
                  steepScale))*/, pos.y - 0.5f),
70              new Vector3(pos.x - 0.5f, yOffset + (amplitude *
                  Mathf.PerlinNoise((float)(pos.x - 0.5f) / chunkSize * scale,
                  (float)(pos.y + 0.5f) / chunkSize * scale)) /*+
                  (steepAmplitude * Mathf.PerlinNoise((float)(pos.x - 0.5f) /
                  chunkSize * steepScale, (float)(pos.y + 0.5f) / chunkSize *
                  steepScale))*/, pos.y + 0.5f),
71              new Vector3(pos.x + 0.5f, yOffset + (amplitude *
                  Mathf.PerlinNoise((float)(pos.x + 0.5f) / chunkSize * scale,
                  (float)(pos.y + 0.5f) / chunkSize * scale)) /*+
                  (steepAmplitude * Mathf.PerlinNoise((float)(pos.x + 0.5f) /
                  chunkSize * steepScale, (float)(pos.y + 0.5f) / chunkSize *
                  steepScale))*/, pos.y + 0.5f),
72              new Vector3(pos.x + 0.5f, yOffset + (amplitude *
```

```
                Mathf.PerlinNoise((float)(pos.x + 0.5f) / chunkSize * scale,
                (float)(pos.y - 0.5f) / chunkSize * scale)) /*+
                (steepAmplitude * Mathf.PerlinNoise((float)(pos.x + 0.5f) /
                chunkSize * steepScale, (float)(pos.y - 0.5f) / chunkSize *
                steepScale))*/, pos.y - 0.5f)
        };
        int[] triangles = new int[] { 0, 1, 3, 3, 1, 2 };
        Vector2[] uv = new Vector2[] { new Vector2(0, 0), new Vector2(0,
          1), new Vector2(1, 1), new Vector2(1, 0) };
        mesh.vertices = vertices;
        mesh.triangles = triangles;
        mesh.uv = uv;
        combine[index].mesh = mf.sharedMesh;
        combine[index].transform = mf.transform.localToWorldMatrix;
    }

    // Static method used for the string, GameObject dictionary, allowing
      for infinite generation
    public static string Vector2IntToChunkPos(Vector2Int xzPos)
    {
        return "Chunk(" + xzPos.x + ", " + xzPos.y + ")";
    }
}
```

```
1  using System.Linq;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ChunkManager : MonoBehaviour
6  {
7      // While this procedular generation system does work, it will not for instances where you travel more than chunkSize in a single frame.
8      // With a chunkSize of 16, if the fish has such a high movement speed that you can travel 32 units for example, one chunk will go missing.
9      // With the above example, you technically should still have the chunk work as intended if you move slower and go back near the missing chunk.
10     public static int renderDistance = 10; // Determines how many chunks ahead of the player should be rendered
11
12     [SerializeField] Transform player;
13     [SerializeField] Chunk chunk; // Chunk script that must be on the same GameObject in scene in order to create chunks
14     Dictionary<string, GameObject> chunks = new Dictionary<string, GameObject>(); // Infinite generation dictionary
15     private Vector2Int oldPos, newPos;
16     private List<Vector2Int> oldCoords, newCoords;
17     private Queue<Vector2Int> loadQueue = new Queue<Vector2Int>(); // What to load/create
18     private Queue<Vector2Int> unloadQueue = new Queue<Vector2Int>(); // What to unload
19     private float chunkTimer;
20     private bool loadQueueActive;
21     private bool unloadQueueActive;
22
23     private void Start()
24     {
25         // Creates all of the starting chunks around the player
26         oldPos = playerPosToChunk();
27         oldCoords = new List<Vector2Int>();
28         newCoords = new List<Vector2Int>();
29         chunkTimer = 5f / renderDistance * 0.1f;
30
31         for (int z = -renderDistance + oldPos.y; z <= renderDistance + oldPos.y; z++)
32         {
33             for (int x = -renderDistance + oldPos.x; x <= renderDistance + oldPos.x; x++)
34             {
35                 Vector2Int pos = new Vector2Int(x, z);
36                 GameObject newChunk = chunk.CreateChunk(pos);
37                 oldCoords.Add(pos);
38                 chunks.Add(newChunk.name, newChunk);
```

```
39              }
40          }
41      }
42
43      private void Update()
44      {
45          newPos = playerPosToChunk();
46
47          if (newPos != oldPos) // If the player has for sure moved to a new chunk position, then we will bother doing anything at all
48          {
49              ChunkCheck(); // Main script
50
51              if (!loadQueueActive) // Since we are 100% going to have new chunks to load, we must turn on the queue method if not on already
52              {
53                  LoadQueue();
54                  loadQueueActive = true;
55              }
56
57              if (!unloadQueueActive) // Since we are 100% going to have old chunks to unload, we must turn on the dequeue method if not on already
58              {
59                  UnloadQueue();
60                  unloadQueueActive = true;
61              }
62          }
63      }
64
65      private void LoadQueue() // Either creates completely new chunks or loads previously loaded chunks that are in queue (in view of player)
66      {
67          Vector2Int currentQueue = loadQueue.First();
68
69          if (chunks.TryGetValue(Chunk.Vector2IntToChunkPos(currentQueue), out GameObject value)) // Case when chunk was once loaded before
70          {
71              value.GetComponent<MeshRenderer>().enabled = true;
72              value.GetComponent<MeshCollider>().enabled = true;
73          }
74
75          else // Case when no such instance of a chunk exists and thus must be created
76          {
77              GameObject newChunk = chunk.CreateChunk(currentQueue);
78              chunks.Add(newChunk.name, newChunk);
79          }
```

```
80
81          loadQueue.Dequeue();
82
83          if (loadQueue.Count == 0) // Turns off automatically if queue is
              empty
84          {
85              loadQueueActive = false;
86          }
87
88          else // If queue is not empty, begin working in chunkTimer seconds
               (recursion)
89          {
90              Invoke("LoadQueue", chunkTimer);
91          }
92      }
93
94      private void UnloadQueue() // Since in order to unload a chunk there
          must be a chunk loaded, we always unload the chunk at a coordinate
95      {
96          Vector2Int currentQueue = unloadQueue.First();
97
98          string key = Chunk.Vector2IntToChunkPos(currentQueue);
99          chunks[key].GetComponent<MeshRenderer>().enabled = false;
100         chunks[key].GetComponent<MeshCollider>().enabled = false;
101
102         unloadQueue.Dequeue();
103
104         if (unloadQueue.Count == 0) // Turns off automatically if queue is
               empty
105         {
106             unloadQueueActive = false;
107         }
108
109         else // If queue is not empty, begin working in chunkTimer seconds
               (recursion)
110         {
111             Invoke("UnloadQueue", chunkTimer);
112         }
113     }
114
115     private void ChunkCheck()
116     {
117         newCoords.Clear();
118
119         // Here we look for all of the chunks around the new position of
              the player using the given render distance
120         for (int z = -renderDistance + newPos.y; z <= renderDistance +
              newPos.y; z++)
121         {
```

```
122             for (int x = -renderDistance + newPos.x; x <= renderDistance +
                    newPos.x; x++)
123             {
124                 Vector2Int pos = new Vector2Int(x, z);
125 
126                 // Any coordinates that are new, and not part of the old
                      coordinates are added to the load queue
127                 if (!oldCoords.Contains(pos))
128                 {
129                     loadQueue.Enqueue(pos);
130                 }
131 
132                 newCoords.Add(new Vector2Int(x, z));
133             }
134         }
135 
136         // Any coordinates that are old, and not part of the new
              coordinates are added to the unload queue
137         List<Vector2Int> coords = oldCoords.Except(newCoords).ToList();
138 
139         foreach (Vector2Int coord in coords)
140         {
141             unloadQueue.Enqueue(coord);
142         }
143 
144         // Once the process is over, the old values become the new for
              later iterations
145         oldCoords = new List<Vector2Int>(newCoords);
146         oldPos = newPos;
147     }
148 
149     private Vector2Int playerPosToChunk() // Method that converts a player
        position to a chunk position using the static int chunkSize
150     {
151         int xOffset = 0, zOffset = 0;
152 
153         if (player.position.x < 0)
154         {
155             xOffset = -1;
156         }
157 
158         if (player.position.z < 0)
159         {
160             zOffset = -1;
161         }
162 
163         int x = (int)(player.position.x / Chunk.chunkSize) + xOffset;
164         int z = (int)(player.position.z / Chunk.chunkSize) + zOffset;
165 
```

```
166             return new Vector2Int(x, z);
167         }
168     }
```

```
1  using UnityEngine;
2
3  public class ChunkOld : MonoBehaviour
4  {
5      // Note: this was the original chunk creation script, and while with no material it looked the exact same (and more optimized)
6      // -it had issues with uvs, pretty much it stretched out the entire sand material across the chunk rather than per 1x1 plane in that chunk
7      // Though looking back now, we probably could've modified some things to make this script also work if we adjusted each coordinate to
8      // display the entire sand using the global equation we already determined in lines 55-60
9      public static int chunkSize = 16;
10
11     public static float amplitude = 3f, scale = 2f;
12
13     private Mesh mesh;
14
15     private MeshFilter meshFilter;
16
17     private MeshRenderer meshRenderer;
18
19     private int[] triangles;
20
21     private Vector2[] uv;
22
23     private Vector3[] vertices;
24
25     private void Start()
26     {
27         mesh = new Mesh();
28         meshFilter = gameObject.AddComponent<MeshFilter>();
29         meshRenderer = gameObject.AddComponent<MeshRenderer>();
30         meshFilter.mesh = mesh;
31         GenerateQuads();
32         UpdateMesh();
33     }
34
35     private void GenerateQuads()
36     {
37         triangles = new int[chunkSize * chunkSize * 6];
38         uv = new Vector2[(chunkSize + 1) * (chunkSize + 1)];
39         vertices = new Vector3[(chunkSize + 1) * (chunkSize + 1)];
40
41         for (int i = 0, z = 0; z <= chunkSize; z++)
42         {
43             for (int x = 0; x <= chunkSize; i++, x++)
44             {
```

```
45                  float y = amplitude * Mathf.PerlinNoise(scale * ((float)x / chunkSize), scale * ((float)z / chunkSize));
46                  vertices[i] = new Vector3(x - 0.5f, y, z - 0.5f);
47                  uv[i] = new Vector2((float)z / chunkSize, (float)x / chunkSize);
48              }
49          }
50
51          for (int i = 0, z = 0; z < chunkSize; z++)
52          {
53              for (int x = 0; x < chunkSize; i += 6, x++)
54              {
55                  triangles[i] = (chunkSize + 1) * z + x;
56                  triangles[i + 1] = (chunkSize + 1) * (z + 1) + x;
57                  triangles[i + 2] = (chunkSize + 1) * z + x + 1;
58                  triangles[i + 3] = (chunkSize + 1) * z + x + 1;
59                  triangles[i + 4] = (chunkSize + 1) * (z + 1) + x;
60                  triangles[i + 5] = (chunkSize + 1) * (z + 1) + x + 1;
61              }
62          }
63      }
64
65      private void UpdateMesh()
66      {
67          mesh.vertices = vertices;
68          mesh.triangles = triangles;
69          mesh.uv = uv;
70          mesh.RecalculateNormals();
71      }
72  }
```

```
1  using UnityEngine;
2
3  public class EatBehaviour : StateMachineBehaviour
4  {
5      // Attached to the fish animator, and pretty much helped play the
         eating animation every time something was eaten
6      // (as long as the animation was not already playing)
7      public override void OnStateUpdate(Animator animator, AnimatorStateInfo
         stateInfo, int layerIndex)
8      {
9          if (stateInfo.normalizedTime > 1)
10         {
11             animator.SetBool("isEating", false);
12         }
13     }
14 }
```

```
using UnityEngine;

public class EnemyBehaviour : MonoBehaviour
{
    // The script attached to each enemy fish
    public int size;
    private PlayerController pc;
    [SerializeField] Transform readjustments;
    [SerializeField] CharacterController cc;
    [SerializeField] TextMesh score;
    [SerializeField] Animator animator;

    private bool newRotation;
    private float movementSpeed;
    private float defaultXSmooth, defaultYSmooth;
    private float defaultSmoothTime = 0.125f;

    // When an enemy fish is created, it is assigned a random size. Based on this size, scale, speed, the score text and other values are determined
    // Furthermore, the fish begins traveling in a random direction
    private void Start()
    {
        transform.localScale += Vector3.one * (0.02f * size);
        movementSpeed = Mathf.Clamp(6 + 0.005f * (-size + FindObjectOfType<EnemySpawnManager>().difficulty), 4, 10);
        score.text = size.ToString();
        cc.detectCollisions = true;
        pc = FindObjectOfType<PlayerController>();
        transform.eulerAngles = new Vector3(0, Random.Range(0, 360), 0);
        score.GetComponent<FollowRotation>().centerTransform = pc.center; // The score will always rotate to the player in the scene for convenience
        animator.SetFloat("SizeScale", Mathf.Clamp(1 / transform.localScale.x, movementSpeed / 6, 1));
    }

    // Runs if two controller colliders were hit
    private void OnControllerColliderHit(ControllerColliderHit collision)
    {
        if (collision.transform.gameObject == pc.gameObject) // Checks to see that it was the player and not another enemy fish
        {
            // Determines whether the player or enemy "dies" based on the size difference
            if (pc.score >= size) // Player is bigger or the same size, player wins and grows
            {
                pc.Grow();
```

```
41                  Destroy(gameObject); // Fish is destroyed
42                  FindObjectOfType<EnemySpawnManager>().UpdateFish(); // To
                      make up for one dead fish, another one instantly spawns
43              }
44
45              else // Player is smaller, game ends
46              {
47                  FindObjectOfType<GameOver>().ShowText(); // Displays game
                      over screen
48                  pc.enabled = false; // Disables player controller
49                  Time.timeScale = 0; // Game "freezes"
50              }
51          }
52      }
53
54      private void Update()
55      {
56          float disToPlayer = (pc.transform.position -
              transform.position).magnitude;
57
58
59          if (disToPlayer >= 125)
60          {
61              Destroy(gameObject);
62              FindObjectOfType<EnemySpawnManager>().UpdateFish();
63          }
64
65          else if (disToPlayer <= 17.5f)
66          {
67              newRotation = true;
68
69              int direction = 1;
70
71              if (size <= pc.score)
72              {
73                  direction = -1;
74              }
75
76              Vector3 targetAngle = Quaternion.LookRotation(direction *
                  (pc.transform.position - transform.position)).eulerAngles;
77              Vector3 currentAngle = readjustments.eulerAngles;
78              float x = Mathf.SmoothDampAngle(currentAngle.x, targetAngle.x,
                  ref defaultXSmooth, defaultSmoothTime);
79              float y = Mathf.SmoothDampAngle(currentAngle.y, targetAngle.y,
                  ref defaultYSmooth, defaultSmoothTime);
80              readjustments.eulerAngles = new Vector3(x, y, 0);
81              cc.Move(readjustments.forward * movementSpeed *
                  Time.deltaTime);
82          }
```

```
83
84          else
85          {
86              if (newRotation)
87              {
88                  transform.eulerAngles = new Vector3(0, Random.Range(0,
                      360), 0);
89                  readjustments.localEulerAngles = Vector3.zero;
90                  newRotation = false;
91              }
92
93              cc.Move(transform.forward * movementSpeed * Time.deltaTime);
94          }
95      }
96  }
```

```
1  using UnityEngine;
2
3  public class EnemySpawnManager : MonoBehaviour
4  {
5      // This script manages the enemy spawns at playtime
6      [SerializeField] GameObject[] enemyFishes; // Types of enemy fishes
         (two in this case)
7      [SerializeField] PlayerController pc; // Player's script ref
8
9      [SerializeField] int totalSpawns; // Total number of spawns, (50 in
         this case)
10     public int difficulty;
11
12     private void Awake() // Runs at load time, sets difficulty and spawns
         all of the fish
13     {
14         difficulty = pc.score / 30;
15
16         if (difficulty < 1)
17         {
18             difficulty = 1;
19         }
20
21         for (int i = 0; i < totalSpawns; i++)
22         {
23             Spawn();
24         }
25     }
26
27     private void Spawn() // Spawn behaviour of fish
28     {
29         int randomSize = Random.Range(pc.score - 4, pc.score + 3 +
           difficulty); // Size of fish, more likely to be bigger with
           harder difficulties
30         int directionX = 1, directionZ = 1;
31
32         // Randomizes direction in the X and Z direction (Y is always the
           same range)
33         if (Random.Range(0, 2) == 1)
34         {
35             directionX = -1;
36         }
37
38         if (Random.Range(0, 2) == 1)
39         {
40             directionZ = -1;
41         }
42
43         // Random position is created
```

```
44          float randomX = Random.Range(pc.transform.position.x + directionX *
               75, pc.transform.position.x + directionX * 50);
45          float randomY = Random.Range(-95f, -45);
46          float randomZ = Random.Range(pc.transform.position.z + directionZ *
               75, pc.transform.position.z + directionZ * 50);
47
48          Vector3 newPos = new Vector3(randomX, randomY, randomZ);
49
50          // Instantiates a fish based on previous quantities
51          GameObject newFish = Instantiate(enemyFishes[Random.Range(0,
              enemyFishes.Length)], newPos, Quaternion.identity);
52          newFish.GetComponent<EnemyBehaviour>().size = randomSize; // Sets
              fish size
53      }
54
55      public void UpdateFish() // Whenever an enemy fish dies, difficulty is
            adjusted and a fish is spawned again
56      {
57          difficulty = pc.score / 30; // Trunucation means that difficulty
              increments with every 30 size increases
58          Spawn();
59      }
60  }
```

```
1  using UnityEngine;
2
3  public class FollowRotation : MonoBehaviour
4  {
5      public Transform centerTransform;
6
7      private void Update() // Based on a transform, the script has the same
         angles as that transform (used for the scores facing the camera)
8      {
9          transform.eulerAngles = centerTransform.eulerAngles.y * Vector3.up;
10     }
11 }
```

```
1  using UnityEngine;
2  using UnityEngine.UI;
3  using UnityEngine.SceneManagement;
4
5  public class GameOver : MonoBehaviour
6  {
7      [SerializeField] Text finalScore, endText;
8      [SerializeField] PlayerController pc;
9
10     // Disables all of the UI and the script because the game has just
         started
11     private void Start()
12     {
13         finalScore.gameObject.SetActive(false);
14         endText.gameObject.SetActive(false);
15         enabled = false;
16     }
17
18     // When necessary, script turns on to show text and enable its update
         method
19     public void ShowText()
20     {
21         enabled = true;
22         finalScore.text = "FINAL SCORE: " + pc.score;
23         finalScore.gameObject.SetActive(true);
24         endText.gameObject.SetActive(true);
25     }
26
27     // Based on a key press, game is unpaused and either the game or the
         menu has its scene loaded
28     private void Update()
29     {
30         if (Input.GetKeyDown(KeyCode.Return))
31         {
32             SceneManager.LoadScene(1);
33             Time.timeScale = 1;
34         }
35
36         else if (Input.GetKeyDown(KeyCode.Escape))
37         {
38             SceneManager.LoadScene(0);
39             Time.timeScale = 1;
40         }
41     }
42 }
```

```
 1  using UnityEngine;
 2  using UnityEngine.SceneManagement;
 3
 4  public class Menu : MonoBehaviour
 5  {
 6      private void Update() // Simply goes to the game scene when anything is ↵
           pressed
 7      {
 8          if (Input.anyKeyDown)
 9          {
10              SceneManager.LoadScene(1);
11          }
12      }
13  }
```

```
1  using UnityEngine;
2  using UnityEngine.UI;
3
4  public class PlayerController : MonoBehaviour
5  {
6      public int score; // Player size
7
8      [SerializeField] Animator animator; // Animation
9
10     [SerializeField] CharacterController controller; // Movement with
         collision detection
11
12     [SerializeField] Color startColor, endColor; // Sprint meter start and
          end colors (green --> red)
13
14     // Various float values used for different things. . .
15     [SerializeField] float moveAcc, sprintCooldown, animAcceleration,
         cameraDistance, defaultSmoothTime, defaultTimer, fastMove, normMove,
          rotationSmoothTime, rotationSpeed;
16
17     [SerializeField] Image sprintMeter; // The front image that is scaled
         up/down to visually display the cooldown
18
19     [SerializeField] KeyCode downKey, sprintKey, upKey; // Keys that
         trigger certain events determined in the editor
20
21     [SerializeField] Text sprintText; // The text that displays the
         cooldown of sprinting
22
23     [SerializeField] TextMesh scoreMesh; // The text that displays the
         score of the player
24
25     [SerializeField] Transform fish, fishCameras; // Transforms of the
         fish and the camera of the fish, used for movement
26
27     public Transform center;  // Center transform (kind of like the camera
          arm)
28
29     private int direction; // Sprinting acceleration direction
30
31     // Several boolean values used as conditionals for iteration
32     private bool cooldownEnabled, isSprinting, shouldDefault;
33
34     // Used for animations, sprinting, etc. . .
35     private float sprintTimer, sprintMoveTimer, animVelocity,
         initialMoveSpeed, movementSpeed, currentDefaultTimer,
         defaultSmoothVelocityX, sizeScale = 1, turnSmoothVelocityX,
         turnSmoothVelocityY;
36
```

```
37      private Vector3 centerAngle; // Camera arm angle
38
39      private float moveVel; // Control sprint movement, works in tandem
          with move acceleration
40
41      private void Start() // Sets values declared in inspector
42      {
43          Cursor.lockState = CursorLockMode.Locked;
44          scoreMesh.text = score.ToString();
45          initialMoveSpeed = normMove;
46          movementSpeed = normMove;
47          fishCameras.transform.localPosition = cameraDistance *
              Vector3.back;
48          currentDefaultTimer = defaultTimer;
49      }
50
51      private void Update()
52      {
53          if (Input.GetKeyDown(sprintKey) && !cooldownEnabled) // If you
              click the sprinting key and it's not on cooldown, things happen
54          {
55              direction = 1;
56              moveVel = 0;
57              cooldownEnabled = true;
58              isSprinting = true;
59              normMove = movementSpeed;
60              movementSpeed = fastMove;
61              sprintTimer = sprintCooldown;
62              sprintMoveTimer = 0.5f;
63              Sprint(); // The cooldown/UI management
64              SprintMove(); // The actual movement
65          }
66
67          float xRot = -Input.GetAxis("Mouse Y") * rotationSpeed *
              Time.deltaTime;
68          float yRot = Input.GetAxis("Mouse X") * rotationSpeed *
              Time.deltaTime;
69
70          // Sets center/camera angles based on mouse X and mouse Y movement
71          centerAngle.x = Mathf.Clamp(centerAngle.x + xRot, -89.9f, 89.9f);
72          centerAngle.y = (centerAngle.y + yRot) % 360;
73          center.eulerAngles = centerAngle;
74
75          // Sets movement values based on axis movements and up/down key
              presses
76          float xTranslate = Input.GetAxisRaw("Horizontal");
77          float yTranslate = 0;
78          float zTranslate = Input.GetAxisRaw("Vertical");
79
```

```
80          if (Input.GetKey(downKey))
81          {
82              yTranslate--;
83          }
84
85          if (Input.GetKey(upKey))
86          {
87              yTranslate++;
88          }
89
90          Vector3 moveDir = xTranslate * center.right + yTranslate *
              transform.up + zTranslate * center.forward;
91
92          if (moveDir.magnitude > 0 && !isSprinting) // Moves and rotates if
               not sprinting and moving at all
93          {
94              if (shouldDefault)
95              {
96                  shouldDefault = false;
97              }
98
99              if (currentDefaultTimer != defaultTimer) // Idle timer reset
100             {
101                 currentDefaultTimer = defaultTimer;
102             }
103
104             Vector3 newRotation = Quaternion.LookRotation
                  (moveDir).eulerAngles; // New angle fish should be facing
105
106             if (xTranslate == 0 && zTranslate == 0) // Makes vertical only
                   movement slightly less iffy
107             {
108                 newRotation.x = -Mathf.Sign(yTranslate) * 89.9f;
109                 newRotation.y = fish.eulerAngles.y;
110             }
111
112             // Smoothly rotates rather than instantaneously
113             float angleX = Mathf.SmoothDampAngle(fish.eulerAngles.x,
                  newRotation.x, ref turnSmoothVelocityX, rotationSmoothTime);
114             float angleY = Mathf.SmoothDampAngle(fish.eulerAngles.y,
                  newRotation.y, ref turnSmoothVelocityY, rotationSmoothTime);
115
116             // Blends from idle to moving and sets angles/positions
117             animVelocity = Mathf.Clamp(animVelocity + animAcceleration *
                  Time.deltaTime, 0, 1);
118             fish.eulerAngles = new Vector3(angleX, angleY, 0);
119             controller.Move(movementSpeed * Time.deltaTime *
                  fish.forward);
120             transform.position = new Vector3(transform.position.x,
```

```
                Mathf.Clamp(transform.position.y, transform.position.y,
                -50), transform.position.z);
121         }
122
123         else if (!isSprinting) // Runs timer, which when reaching zero,
              defaults the rotation of the fish
124         {
125             animVelocity = Mathf.Clamp(animVelocity - animAcceleration *
                Time.deltaTime, 0, 1);
126
127             if (!shouldDefault && currentDefaultTimer != 0)
128             {
129                 currentDefaultTimer = Mathf.Clamp(currentDefaultTimer -
                    Time.deltaTime, 0, defaultTimer);
130             }
131
132             else if (!shouldDefault)
133             {
134                 shouldDefault = true;
135             }
136
137             else if (fish.eulerAngles != new Vector3(0,
                fish.eulerAngles.y, 0))
138             {
139                 float x = Mathf.SmoothDampAngle(fish.eulerAngles.x, 0, ref
                    defaultSmoothVelocityX, defaultSmoothTime);
140
141                 fish.eulerAngles = new Vector3(x, fish.eulerAngles.y, 0);
142             }
143         }
144
145         animator.SetFloat("Velocity", animVelocity); // Idle --> moving or
              moving --> idle based on whether you moved or not this frame
146     }
147
148     private void SprintMove() // Sprint movement managed,  uses recursion
149     {
150         sprintMoveTimer = Mathf.Clamp(sprintMoveTimer - Time.deltaTime, 0,
              0.5f);
151
152         if (sprintMoveTimer != 0)
153         {
154             moveVel += moveAcc * direction;
155             controller.Move(movementSpeed * moveVel * Time.deltaTime *
                fish.forward);
156             animVelocity = Mathf.Clamp(animVelocity + animAcceleration *
                30 * Time.deltaTime, 0, 1);
157             transform.position = new Vector3(transform.position.x,
                Mathf.Clamp(transform.position.y, transform.position.y,
```

```
                -50), transform.position.z);
158             Invoke("SprintMove", Time.deltaTime);
159         }
160
161         else if (direction == 1)
162         {
163             direction = -1;
164             sprintMoveTimer = 0.5f;
165             SprintMove();
166         }
167
168         else
169         {
170             movementSpeed = normMove;
171             isSprinting = false;
172         }
173     }
174
175     private void Sprint() // Sprint cooldown management, uses recursion
176     {
177         sprintTimer = Mathf.Clamp(sprintTimer - Time.deltaTime, 0,
              sprintCooldown);
178         Color newColor = Color.Lerp(startColor, endColor, sprintTimer /
              sprintCooldown);
179         sprintMeter.transform.localScale = new Vector3((1 - sprintTimer /
              sprintCooldown), 1, 1);
180         sprintMeter.color = newColor;
181
182         if (sprintTimer == 0)
183         {
184             sprintText.text = "Sprint ready (Q)";
185             cooldownEnabled = false;
186         }
187
188         else
189         {
190             sprintText.text = "Sprint ready in " + ((int)(sprintTimer +
                  1)).ToString() + "...";
191             Invoke("Sprint", Time.deltaTime);
192         }
193     }
194
195     public void Grow() // Runs when the fish grows, and sets values of the
          fish based on the new size
196     {
197         animator.SetBool("isEating", true);
198         transform.localScale += 0.02f * Vector3.one;
199         fishCameras.transform.localPosition += 0.0075f * Vector3.back;
200         rotationSmoothTime = Mathf.Clamp(rotationSmoothTime + 0.0008f,
```

```
            rotationSmoothTime, 0.5f);
201
202         if (isSprinting)
203         {
204             normMove = Mathf.Clamp(normMove - 0.005f, 4, normMove);
205             sizeScale = Mathf.Clamp(1 / transform.localScale.x, normMove /
                initialMoveSpeed, sizeScale);
206         }
207
208         else
209         {
210             movementSpeed = Mathf.Clamp(movementSpeed - 0.005f, 4,
                movementSpeed);
211             sizeScale = Mathf.Clamp(1 / transform.localScale.x,
                movementSpeed / initialMoveSpeed, sizeScale);
212         }
213
214         animator.SetFloat("SizeScale", sizeScale);
215         score++;
216         scoreMesh.text = score.ToString();
217     }
218 }
```