製作物について

今回の製作物としては、レースゲームを作成した。プレイヤーは山間部を舞台に設定されたコースを駆け抜け、スコアを稼ぎながらゴールを目指すことに加え、様々な障害物を回避しながらゴールを目指す。本作の特徴としては、Unity の Terrain 機能を用いて一から作り上げてリアルな山の地形を作成したり、細かく設計した障害物、視点操作、スコア表示システム等プレイヤーの没入感を高める仕組みを充実させた。ゲーム内では山を再現するため高低差を持たせた地形編集を行い、道路や障害物を丁寧に配置した。さらに、滝や霧などの自然要素も取り入れることで山の風景も再現した。また、画面遷移や効果音、アニメーションを組み合わせることで、より楽しめるような空間を作成した。

1.1 ステージ全体について

図 1：ステージ全体

今回のステージでは、山を土台として製作を行った。Asset ではなく、unity の terrain という機能を使用しいちから編集を行い作成した。加えて、レースゲームの作成にはコースを整える必要があったため、道路も一つ一つプロットすることにより作成した。作成したものは左の図 1 のようになった。

1.1.1 山の製作について

前述のとおり、山自体の作成には asset を使用せず、terrain という unity の 3Dobject を編集することにより作成した。Terrain とは一枚の薄い板のようなものであり、現実世界における山を再現するためにはその高低差を編集する必要があった。山の高低差を編集するには、terrain の編集における raise and low というツールを使用することで、より高さを付けたい地点やどれくらいの高さにするかなどはカーソルを合わせることで自身の力加減で細かい編集が可能であった。ここまでを行うと図 2 のようになる。山には当然、ただの緑色というわけではなく、草やコケ、木、土が生えているためそれを再現するために、paintTexture というツールを使用し、設置や色付けを行うことよりリアルに仕上げた。しかし、単に塗るだけであると簡素であったため以下の様に自身で asset を一本一本設置した。その製作過程が図 3 である。

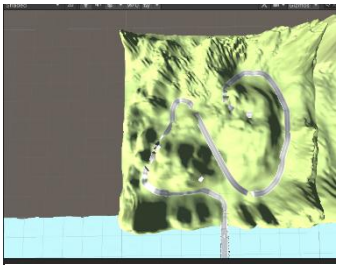

図 2:山の高低差を付ける

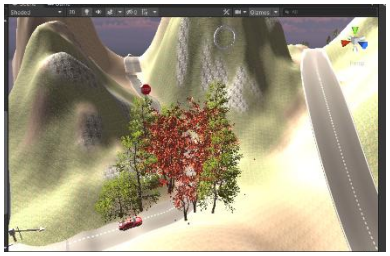

図 3 ：山の装飾（木などの設置過程）

1.1.2 道路の作成

道路の作成については、Easy Road 3D という asset を使用することにより、自身が建設したい道路を一つ一つプロットすることで自由自在にコースを作成することができた。図 4 について、プロットしている様子を撮った物であり、青い点から次の青い点まで道路をプロットすることができ、プロットした点は軸を編集することも可能である。これらの編集を行うことにより図 5 の様に完成した。

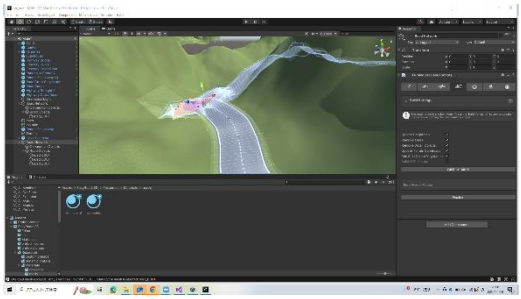

図 4:道路の製作過程

図 5 :道路の完成図

1.2 プレイヤー操作について

今回、プレイヤーは asset を用いてプログラムを組むことにより動かした。プレイヤーはキーボードキーで随時動くため、その動きを認識するために追跡カメラも必要となった。

1.2.1 プレイヤーの動きについて

プレイヤーについては矢印キーで前と後ろに進め、左右の矢印により自身の軸を中心に、水平方向に 360 度回転することができる。アニメーションから、動きを抜粋しスケートボードに乗りながらプレイしているという様子を再現した。ゲームが開始すると start メソッドが一度だけ呼び出され、コンポーネントを取得する。ゲームが進行する間、毎フレーム update メソッドを呼び出し、矢印キーを使いプレイヤーの向きを変更するためには changeDirection を使用した。スペースキーを押すと jump 処理が呼び出され、キャラクターが地面にいる場合のみにこのメソッドは呼び出し可能となる。速度については、プレイヤーの moveDirection を使い、キャラクターに速度を与えることで実現した。この速度については、Rigidbody 内の velocity を設定することにより適応される。キャラクターの向

きについては RoatedSpeed に基づき回転速度が調整され、左右矢印キーにより変更可能である。ゲーム中はこの流れを繰り返すことによりプレイが可能となっている。以下図 6 が使用したスクリプトである。

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

public class PlayerController : MonoBehaviour
{
    private Animator animator;
    private Rigidbody rb;

    private Vector3 moveKeyInput;
    private Vector3 moveDirection;
    private bool isGround = true;
    private Vector3 playerPos;

    [SerializeField] private float moveSpeed = 6.0f;
    [SerializeField] private float jumpPower = 10.0f;
    [SerializeField] private float rotateSpeed = 100f;

    private float adjustment = 2.0f;
   void Start()
    {
        rb = GetComponent<Rigidbody>();
        animator = GetComponent<Animator>();
        playerPos = GetComponent<Transform>().position;

    }

    private void Update()
    {
        ChangeDirection();
        MovePlayer();

    }
```

```
    void FixedUpdate()
    {
        Jump();
    }

    private void MovePlayer()
    {
        moveKeyInput = new Vector3(Input.GetAxis("Vertical"), 0.0f, 0.0f);
        moveDirection = (transform.forward * moveKeyInput.x).normalized;

        rb.velocity = moveDirection * moveSpeed * Time.deltaTime;
    }

    private void ChangeDirection()
    {
        if (Input.GetKey(KeyCode.RightArrow))
        {
            transform.Rotate(new Vector3(0, 1, 0) * rotateSpeed *
Time.deltaTime);
        }
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            transform.Rotate(new Vector3(0, -1, 0) * rotateSpeed *
Time.deltaTime);

        }
    }

    private void Jump()
    {
        if (Input.GetKeyDown(KeyCode.Space) && isGround)
        {
            isGround = false;
            Vector3 jumpForce = new Vector3(0.0f, jumpPower, 0.0f);
            rb.AddForce(jumpForce, ForceMode.VelocityChange);
        }
```

```
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Ground"))
        {
            isGround = true;
        }
    }
}
```

図 6 ：PlayerController（プレイヤーの動作に関するスクリプト）

1.2.2 追跡カメラについて

プレイヤーをコースで動かす中で、プレイヤーの動きに合わせてカメラでとらえる必要がある。図 7 のスクリプトを使用することにより、I キーと L キーにより垂直に 360 度カメラを回転させることができる仕組みとなった。I キーを押すと上方向にカメラを向けることができ、K キーを押すと下方向に向けることができる。これによりゲーム内でプレイヤーが視点を自由に操作することができる。コース自体が山であるため高低差も激しいことからこの動作は必要不可欠である。

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

public class CameraController : MonoBehaviour
{
    public float rotateSpeed, rotateAngle;
    private void Update()
    {
            if (Input.GetKey(KeyCode.I))
            {
                transform.Rotate(-rotateAngle * Time.deltaTime * rotateSpeed, 0, 0);
            }
            if (Input.GetKey(KeyCode.K))
            {
                transform.Rotate(rotateAngle * Time.deltaTime * rotateSpeed, 0, 0);
```

```
            }
    }
}
```

図 7: CameraController

（プレイヤー追従カメラにおける、視点の操作を行うためのスクリプト）

1.3 障害物について

今回コース内には 5 つの障害物を設置し、それにぶつかるとゲームオーバーとなる仕組みとした。以下、1.3.1~1.3.5 で 5 つの障害物について説明を行う。障害物に接触するとゲームオーバー画面へと遷移する仕組みを作成するため、図 8 スクリプトを使用した。Player が衝突したことを検知するには、オブジェクトがものとして認識されなければならないため、boxCollider もしくは capuselCollider を付ける必要がある。加えて、IsTrigger タグをつけることにより、プレイヤーがトリガー範囲内に入ると OnTriggerEnter メソッドが呼び出され、If 文によりそのオブジェクトに指定のタグが付いていれば GameOver メソッドが呼び出され、ゲームオーバー画面へと遷移する仕組みとした。

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

public class GameOverTrigger : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Obstacle"))
        {
            GameOver();
        }
        else if (other.gameObject.CompareTag("FallenTree"))
        {
            GameOver();
        }
        else if (other.gameObject.CompareTag("Butterfly"))
        {
```

```
            GameOver();
        }
        else if (other.gameObject.CompareTag("EarthAndSand"))
        {
            GameOver();
        }
    }
    private void GameOver()
    {
        Debug.Log("Game Over!");
        SceneManager.LoadScene("GameOver");
    }
}
```

図 8: GameOverTrigger　（ゲームオーバー判定）

1.3.1 回る鶏

ある点を軸に鶏が回るという障害物を一つ目として作成した。ある点として、3Dobjrct で cube を作成し、機能はさせたいが可視化できないようにするため MeshRenderer を false にした。そして、この周りを回転するように図９のスクリプトを使用した。中心点を orbitCenter という先に設定しておいた cube のオブジェクトの位置で初期化を行い、指定した軸に沿って指定した周期で回転する動作を実現した。これにより作成したものが以下図 10，図 11 である。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class RotateAround : MonoBehaviour
{
    [SerializeField] private GameObject orbitCenter;
    private Vector3 _center;
    [SerializeField] private Vector3 _axis = Vector3.up;
    [SerializeField] private float _period = 2;
    [SerializeField] private bool _updateRotation = true;
    [SerializeField] private float _radius = 100.0f;

    private void Start()
    {
```

```
        _center = orbitCenter.transform.position;
    }
    private void Update()
    {
        var tr = transform;
        var angleAxis = Quaternion.AngleAxis(360 / _period * Time.deltaTime, _axis);
        var pos = tr.position;

        pos -= _center;
        pos = pos.normalized * _radius;

        pos = angleAxis * pos;
        pos += _center;

        tr.position = pos;
        if (_updateRotation)
        {
            tr.rotation = tr.rotation * angleAxis;
        }
    }
}
```

図 9：RotateAround（オブジェクトを中心点に対して回転させる）

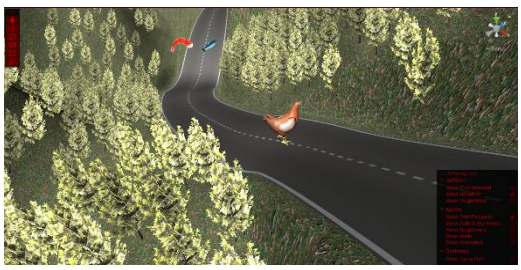

図 10:回転する鶏 1

図 11：回転する鶏 2

尚、図 10、11 を見比べることにより、オブジェクトが回転し位置が移動している事が確認できる。

1.3.2 上下する鹿について

二つ目の障害物として、上下する鹿を作成した。この動作を実現するため、オブジェクトを Y 軸に上下移動させる処理を図 12 のスクリプトで行った。Move は移動の速さであり、毎フレーム move 分だけ Y 軸方向に移動し、count が 50 に達すると移動方向を反転さ

せる仕組みである。一定のフレームごとに移動方向を反転させることにより永続的に上下運動を繰り返すことができる。これにより、図 13 のように上下する鹿を作成した。

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

public class MoveCylinder : MonoBehaviour
{
    private float count;
    private float move = 0.5f;
    void Update()
    {
        Vector3 position = new Vector3(0, move, 0);
        transform.Translate(position);
        count++;

        if(count == 50)
        {
            count = 0;
            move = -move;
        }
    }
}
```

図 12 : MoveCylinder（オブジェクトを永続的に上下運動させる）

図 13：上下する鹿（障害物）

1.3.3 土砂崩れ

三つ目の障害物として土砂崩れを作成した。土砂崩れを再現するために、岩を数十個用意し、そのそれぞれに対して図 14 のスクリプトをアタッチすることにより実現した。また、ゲーム開始時に落下し始めるとプレイヤーがその地点に来ると既に土砂が下に落ちている状態になってしまうため、図 14 のスクリプトにより、プレイヤーが特定のトリガーエリアに衝突すると指定された土砂オブジェクトが 1 秒後に重力で落下するという演出を制作した。これを実装したものが図 15 である。

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

public class LandslideTrigger : MonoBehaviour
{
    [SerializeField] GameObject[] debrisObjects;
    float triggerDelay = 1.0f;

    private void Start()
    {
    }
    private void OnCollisionEnter(Collision collision)
```

```
    {
        if(collision.gameObject.CompareTag("Player"))
        {
            TriggerLandslide();
        }
    }

    public void TriggerLandslide()
    {
        StartCoroutine(StartLandslide());
    }
    private IEnumerator StartLandslide()
    {
        yield return new WaitForSeconds(triggerDelay);
    foreach (GameObject debris in debrisObjects)
        {
            Rigidbody rb = debris.GetComponent<Rigidbody>();
            if (rb != null)
            {
                rb.useGravity = true;
            }
        }
    }
}
```

図 14：LandslideTrigger（ある地点を追加すると岩が落下してくる）

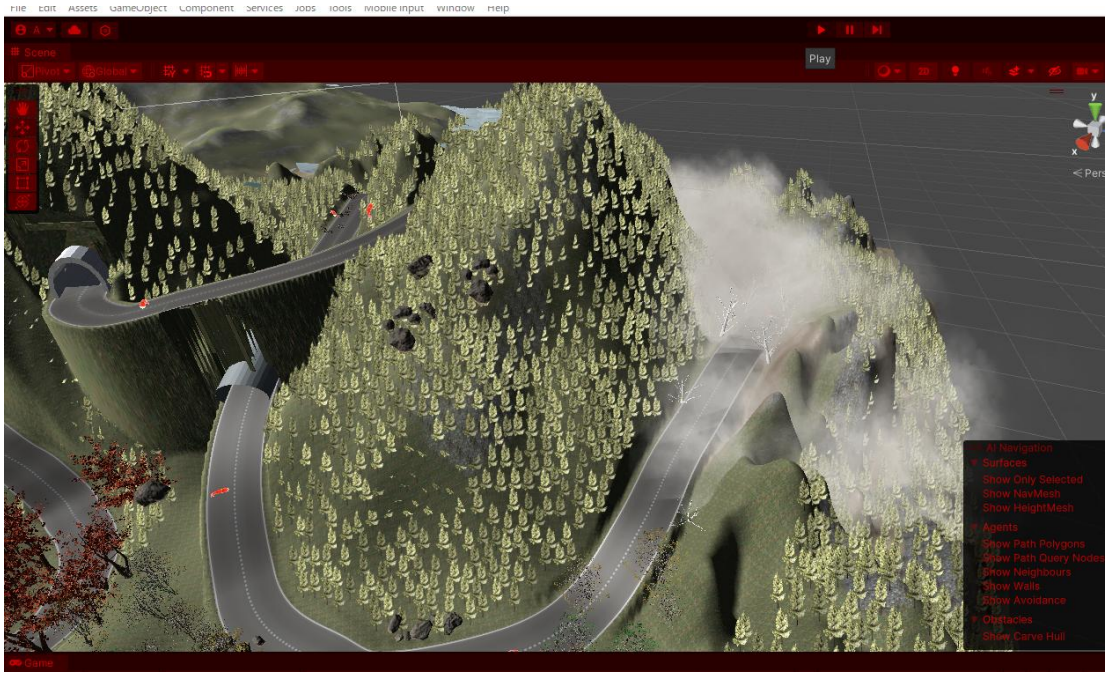

図 15 ：土砂の作成（画面中央部に岩があるとわかる）

1.3.4 倒木

4 つ目の障害物として、通ると木が倒れ始めるというものを作成した。土砂崩れと同様に、ある地点に差し掛かると木が倒れる仕組みを作成した。動作の実現には図 16 のスクリプトを使用した。このスクリプトにより、プレイヤーと接触した時に回転や傾きの動作を制御する機能を実現することができた。ランダムな角度でアニメーション的に回転し、DoTween を活用することによりスムーズな回転アニメーションを実現することができた。これを実行したものが図 17 である。

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

public class RotateObject : MonoBehaviour

{
    public float fallSpeed = 2.0f;
    private bool isFalling = true;
    private Quaternion initialRotation;
    private Quaternion targetRotation;
```

```
    void Start()
    {
        initialRotation = transform.rotation;
        targetRotation = Quaternion.Euler(initialRotation.eulerAngles.x + 90,
    initialRotation.eulerAngles.y, initialRotation.eulerAngles.z);
    }

    void Update()
    {
        if (isFalling)
        {
            transform.rotation = Quaternion.RotateTowards(transform.rotation,
    targetRotation, fallSpeed * Time.deltaTime);
            if (Quaternion.Angle(transform.rotation, targetRotation) < 0.1f)
            {
                isFalling = false;
            }
        }
    }
    public void StartFalling()
    {
        isFalling = true;
    }
}
    public float duration;
    private bool isRotate = false;
    private void OnTriggerEnter(Collider col)
    {
        if (col.gameObject.tag == "Player" && isRotate == false)
        {
            Rotate();
            isRotate = true;
        }
    }
    private void Rotate()
    {
```

```
            RandomAngle
            transform.DORotate(new Vector3(0, 0, RandomAngle()), duration);
        }
        private float RandomAngle()
        {
            int value = Random.Range(0, 2);
            if (value == 0)
            {
                return 70.0f;
            }
            else
            {
                return -70.0f;
            }
        }
}
```

図 16: RotateObject（プレイヤーがある地点に差し掛かると木が倒れ始める）

図 17：倒木

1.3.5 泥沼

五つ目の障害物として泥の沼（図 18）を設置した。Unity 内の Cilynder によりうすい円上の板のようなものを用意し、泥の texture を付けることで泥沼を再現した。
capuselCollider を付け当たり判定を行い、Object タグをつけることによりプレイヤーが触れるとゲームオーバーとなる仕組みを作成した。

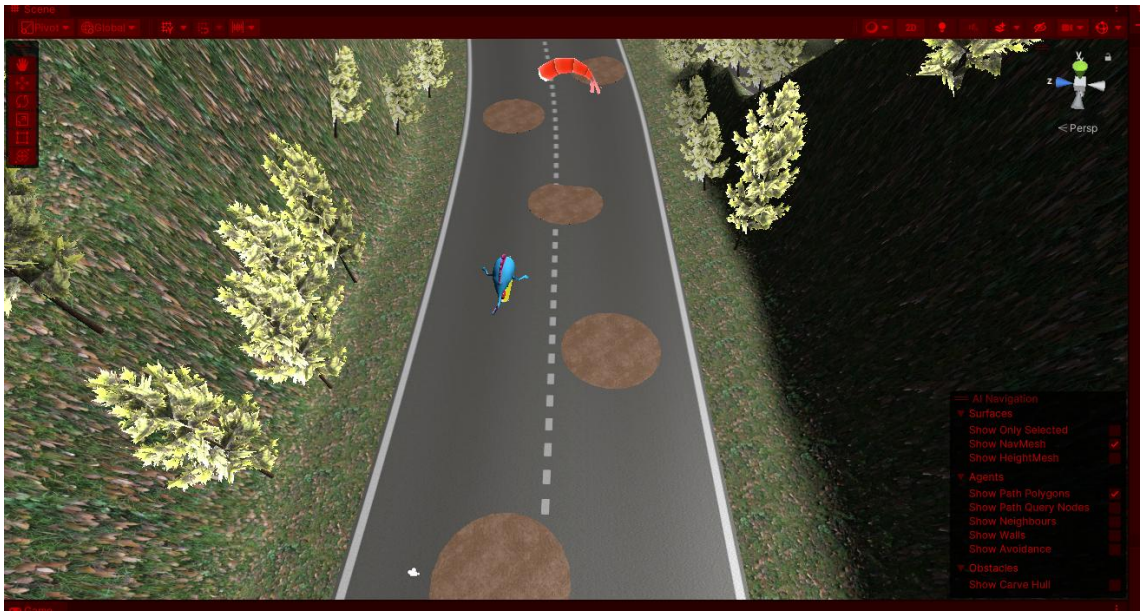

図 18：泥沼

1.4 スコア（点数表示）について

今回のステージでは、障害物を設置するだけでなく、当たるとスコアがプラスになるものとマイナスになるオブジェクトを設置し、そのスコアが随時表示される仕組みを作成した。図 19 の様に、ゲーム画面左上にスコア表示を行いそのスコアの状況が随時表示される仕組みを作成した。触れると音を鳴らしながらオブジェクトが消え、点数が加算・減算されていく仕組みである。

図 19：ゲーム内のスコア表示

1.4.1 スコアの加算・減算について

具体的には、エビのオブジェクトが 10 点、魚のオブジェクトが - 5 点という設定にした。スコアの変動に関係するオブジェクトであるエビと魚のオブジェクトにそれぞれ名前の tag を設置する。そして、shrimp のタグが付いたエビと衝突した場合には addScore メソッドを呼び出し 10 点加算する。Fish のタグが付いた魚と衝突すると substract スコアメソッドを呼び出し、スコアを 5 点減算するという仕組みを図 20 のスクリプトで実現した。加えて、プレイヤーが既にふれたスコアのオブジェクトに関しては二重で獲得しないためにプレイヤーがオブジェクトに触れるとオブジェクトが消える仕組みとした。図 21 でのスクリプトでは、Player タグをつけた Player に衝突すると onCollision メソッドを呼び出し Destroy メソッド内で触れたオブジェクトを呼び出し削除するという動作を行い、衝突した場合にはそのオブジェクトを消すことが可能となっている。

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

public class CollisionHandler : MonoBehaviour
{
    private ScoreManager scoreManager;

    private void Start()
    {
        scoreManager = FindObjectOfType<ScoreManager>();
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Shrimp"))
        {
            scoreManager.AddScore(10);
        }
        if (collision.gameObject.CompareTag("Fish"))
        {
            scoreManager.SubtractScore(5);
        }
    }
```

```
}
```

図 20：CollisionHandler（特定のオブジェクトに触れると加算・減算される）

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

public class ShrimpBehavior : MonoBehaviour
{
    void Start()
    {
    }
    void Update()
    {
    }
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Player")
        {
            Destroy(gameObject);
        }
    }
}
```

図 21：ShrimpBehavior（プレイヤーが触れると消える）

1.4.2 スコア表示について

スコア表示については、図 22 のスクリプト ScoreManager を UI の Text に添付することで随時更新されたスコアが表示されるようにした。図 20 における CollisionHander で衝突イベントに応じて addscore や substractScore を呼び出し値を更新する。加算、減算における変動が行われた場合にはそのポイント数を score に加算し、UpdateText メソッドを呼び出し値を更新し、表示するスコアの更新が可能となる。また、score が 0 以下の場合は図 23 の様に文字を赤色にして置き、0 以上になれば図 19 のように白色という様にマイナス点が分かりやすいように可視化した。

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

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class ScoreManager : MonoBehaviour
{
    public TMP_Text scoreText;
    private int score = 0;
    private void Start()
    {
    }
    public void AddScore(int points)
    {
        score += points;
        UpdateScoreText();
    }
    public void SubtractScore(int points)
    {
        score -= points;
        UpdateScoreText();
    }
    private void UpdateScoreText()
    {
        if (scoreText == null)
        {
            Debug.LogError("ScoreText ");
            return;
        }
        if (score < 0)
        {
            scoreText.color = Color.red;
        }
        else
        {
            scoreText.color = Color.white;
        }
```

```
            scoreText.text = "Score: " + score.ToString();
        }
}
```

図 22：ScoreManager　（ゲーム中のスコアの管理をおこなう）

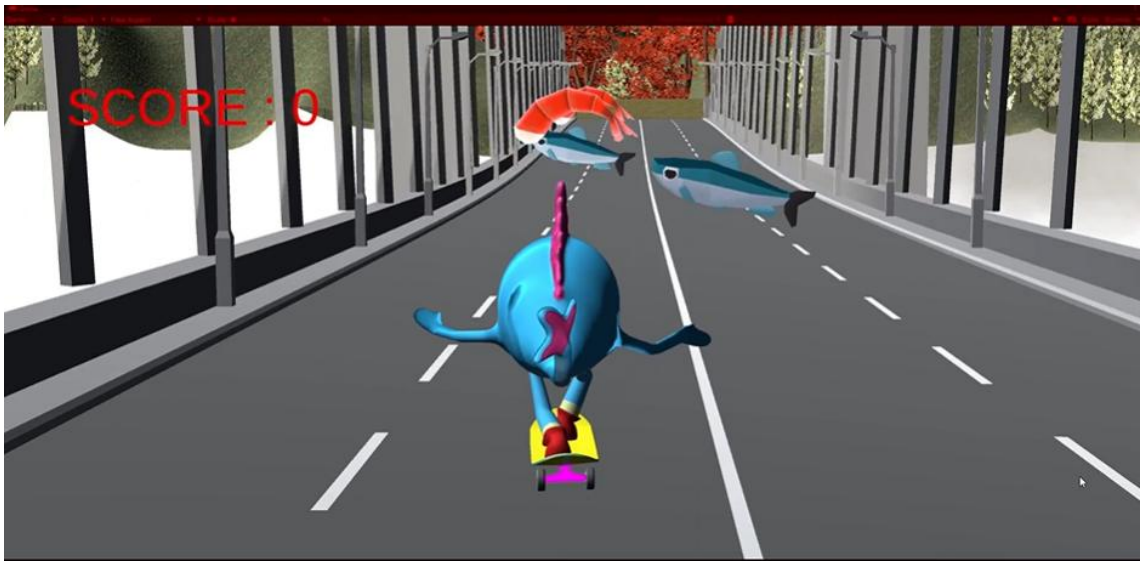

図 23：スコア表示（0 点以下の場合）

1.4.3 衝突時に音を鳴らす

Shrimp と fish に衝突した時に、点数を獲得したということが分かるように音を鳴らす仕組みを作成した。以下、図 24 がそれに使用したスクリプトのプログラムである。指定された効果音を一回再生する機能を持ち、衝突した場合に PlaySE メソッドを呼びだし、引数で渡された AudioClip を、PlayOneShot を使うことで再生する。audioSorce を新しくコンポーネントし、その中で audio を指定した曲で挿入することにより可能となる。

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

public class PlaySound : MonoBehaviour
{
    private AudioSource audio;
    void Start()
    {
        audio = GetComponent<AudioSource>();
    }
    void Update()
```

```
    {
    }
    public void PlaySE(AudioClip audioClip)
    {
        audio.PlayOneShot(audioClip);
    }
}
```

図 24：PlaySound（衝突時に音を鳴らす）

1.5 風景のオブジェクトの設置

今回、障害物だけでなく、山ならではのものとして滝、霧、温泉なども作成した

1.5.1 滝

Unity 内の機能の particleSystem を利用し、滝を作成した。particleSystem は粒の大きさや量、速さ、色などのパラメータ値を変えることにより実現した。

Emission,Trails,Renderer モジュール内パラメータ変更した。StartSpeed で水の勢いを変更でき、StartSize で水の幅を変えることができる。加えて、GravityModifier を編集すると流れ落ちる速さを変えることができ、これらを微細に編集することによりリアルな滝に近づけることができる。

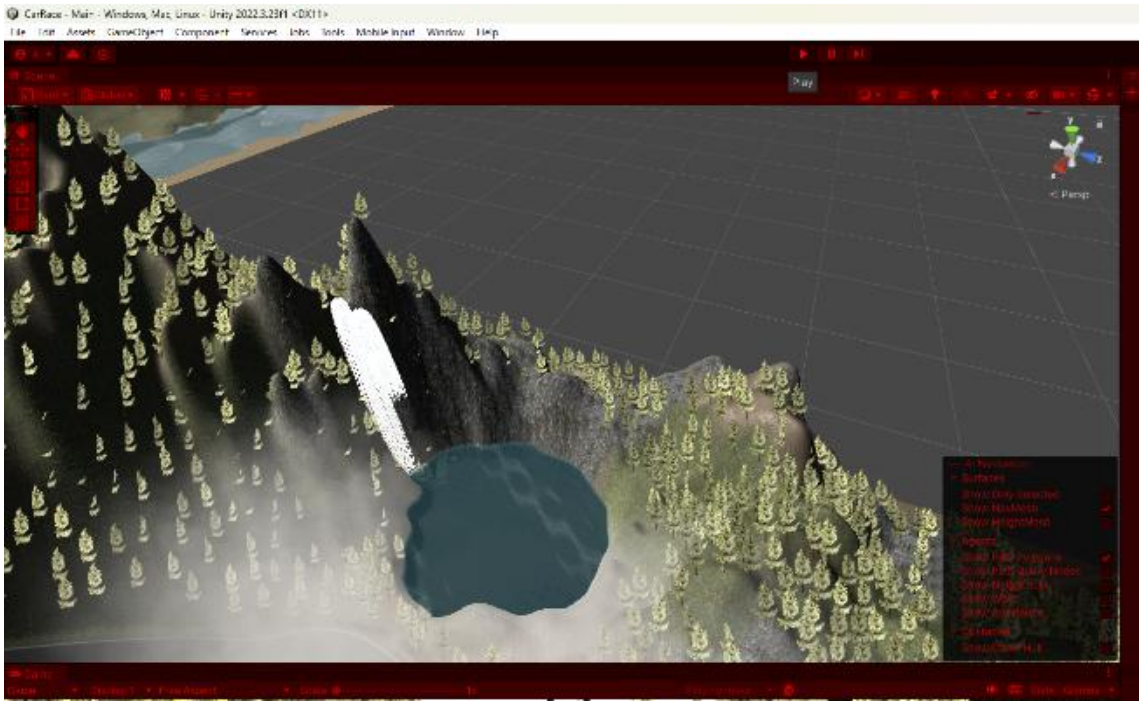

図 25：滝の作成

1.5.2 温泉（ゴール地点）

今回、滝を作成するのに用いた ParticleSystem を利用し、温泉の湯気も再現した。湯気については図 26 の中心部分が白くなっていて、作成したことが分かる。湯気の場合は

Emission,Shape,ColorOverLifetime,RotationOverLifetime,Renderer の値を微調整し編集し水蒸気の粒を再現した。また、山のくぼみを利用することで海の asset をその間に組み込み温泉の湯を再現した。岩の asset を実際の温泉の様に並べることで実際の温泉を再現することができた。

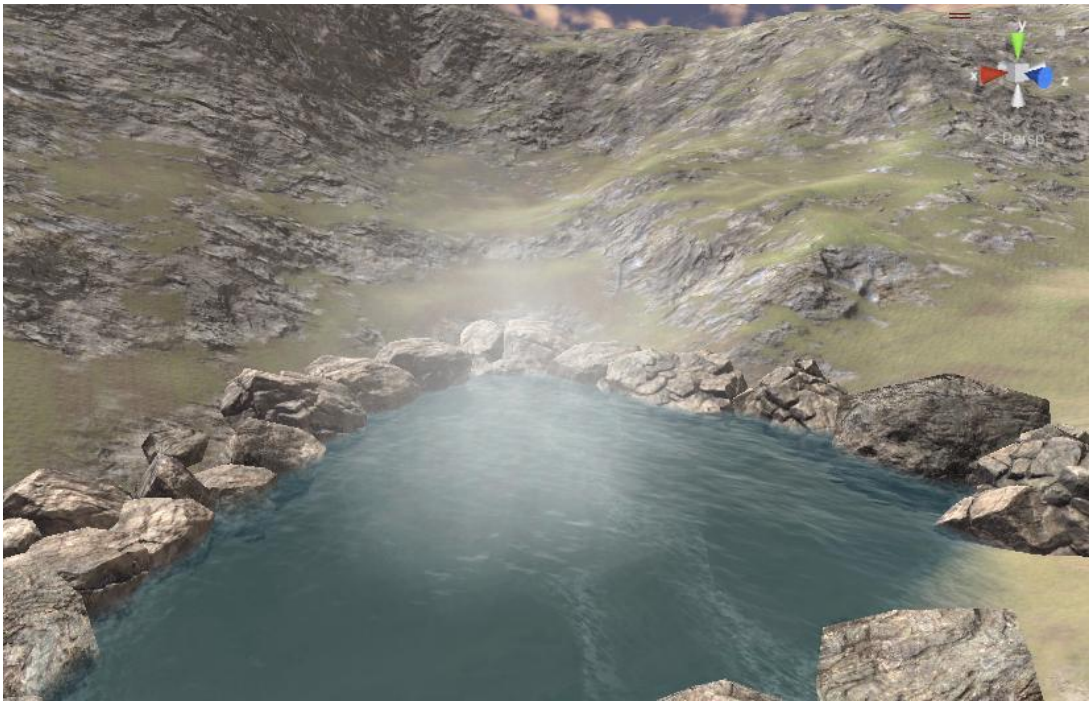

図 26：温泉

1.5.3 霧

上記二つと同様、Unity 内における ParticleSystem という機能を利用し、霧を作成した。霧を作成したことにより、ゲーム中に視界が悪くなることで進みにくくするというギミックとしての効果もあった。霧の作成にはモジュールにおけるパラメータ値を編集することで空気の粒を再現した。霧のマテリアルを作成し挿入することで、よりリアルさを追求した。Emmision,shape,color of time などを編集することで可能である。

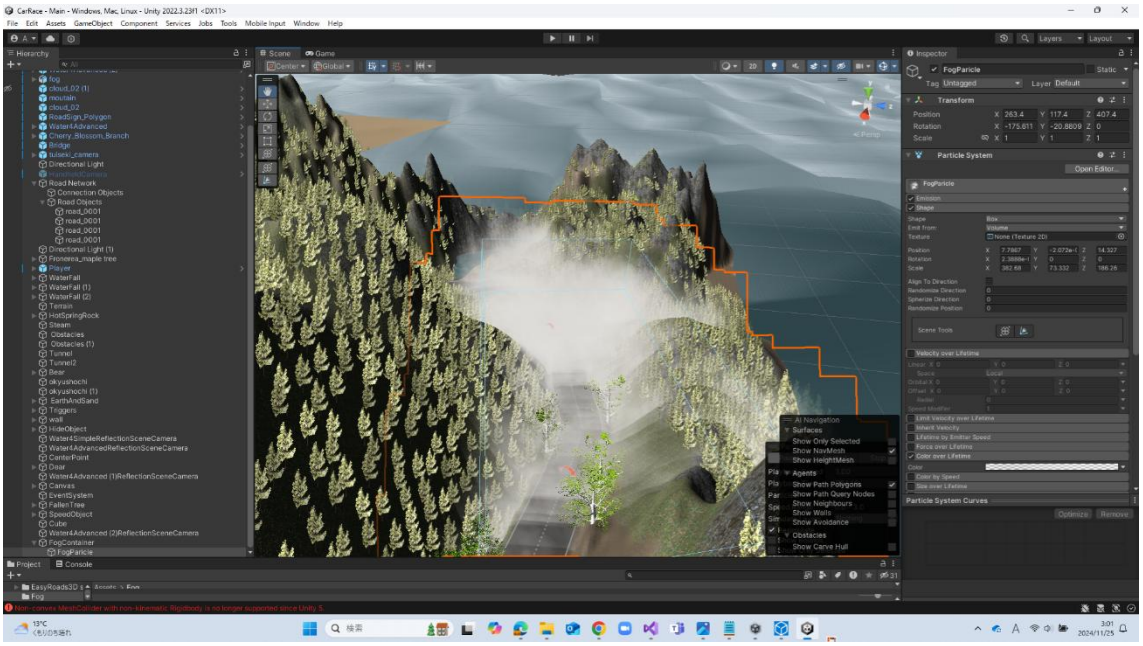

図 27：霧

1.6 画面切り替えについて

今回の実験内では、main シーンに加え、ゲームを始めるためにゲームを開始するタイトル画面、ゲームクリア画面、ゲームオーバー画面を作成した。それぞれを作成するため、それぞれの画面に応じてシーンを新しく、Title シーン、ゲームをクリアした場合に表示される clear シーン、ゲームオーバー時に表示する gameover シーンを作成した。

1.6.1　Title シーンについて

今回のタイトル画面には画像を使用した。UI の Panel の機能を使用し、ゲーム中に撮影した画像を Image に挿入することでゲーム背景画面を作成した。また、Text でゲーム名の追加も行った。加えて、Start ボタンを設置し、このボタンを押すと main シーンに遷移す

る仕組身を実現し図 28 のようなタイトル画面を作成した。

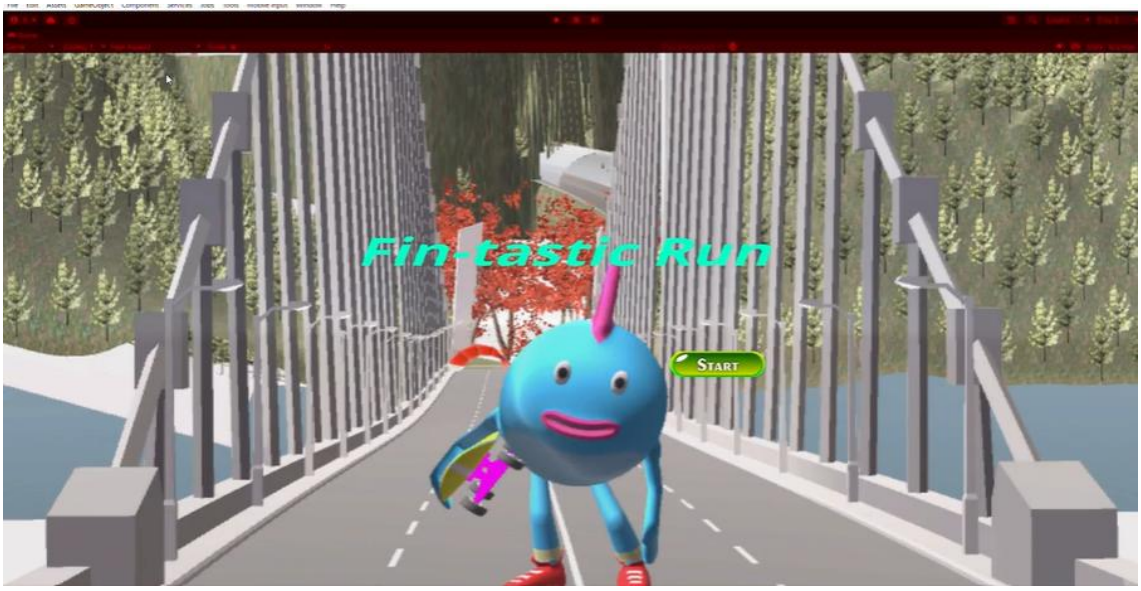

図 28：ゲームタイトル画面

Start ボタンを押すと、ゲーム実行画面へと遷移するようにプログラムを組んだ。Start ボタンについては unity エディター内の button を使用し作成した。図 29 のスクリプトをアタッチした。インスペクターウィンドウで scenName の値を切り替えたいシーン名(Main)を設定した。クリック時(On Click)にスクリプトの changeScene メソッドが呼び出されるようにし、画面が遷移するようにした。

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

public class button_game_start : MonoBehaviour

{
    public string sceneName ="Main";
    public void ChangeScene()
    {
        if (!string.IsNullOrEmpty(sceneName))
        {
            SceneManager.LoadScene(sceneName);
        }
```

```
        else
        {
            Debug.LogError("ゲームをスタートします");
        }
    }
}
```

図 29：button_game_start（スタートボタンクリック時の画面遷移）

1.6.2 Clear シーンについて

カメラの位置を調節し、アニメーションを組み合わせることによりプレイヤーが飛び跳ねる映像を背景としてクリア画面を作成した。ゲームクリアについて、今回ではこの温泉地に到着するとゴールするという判定にしたため以下の図 30 のようなスクリプトを使用した。具体的には、温泉地に透明のコライダーを設置することにより、触れるとゲームクリア画面に遷移する仕組みである。加えて、Retry ボタンを押すと、start ボタン同様クリック時に画面が遷移するためのスクリプトが呼び出されることでタイトル画面へ遷移するようにした。コードは図 29 と同様であり、「public string sceneName ="Main";」の部分を「 public string sceneName ="Title";」に書き換えることで遷移する画面がタイトル画面になる。これを実装したものが以下図 31 の図である。

```
using UnityEngine;

public class GoalChecker : MonoBehaviour
{
    // ゴールしたときに表示するメッセージ
    public string goalMessage = "ゴールしました！";
    public string targetTag = "Player";

    private void OnTriggerEnter(Collider other)
    {
        // ゴールエリアに特定のタグのオブジェクトが入った場合
        if (other.CompareTag(targetTag))
        {
            Debug.Log(goalMessage);
            GoalAchieved();
        }
    }
```

```
    private void GoalAchieved()
    {
        Debug.Log("ゴール判定の処理を実行中");
    }
}
```

図 30：GoalChecker(ゴール判定のスクリプト)

図 31：ゲームクリア画面（clear シーン）

1.6.3 GameOver シーン

障害物にぶつかるとゲームオーバー画面に遷移する仕組みを作成し、これについての具体的な仕組みについては、1.3 において説明し、図 8 における GameOverTrigger スクリプトにより実装している。Retry ボタンについては 1.6.2 のクリア画面の説明と同様である。ゲームオーバー画面についてはプレイヤーのアニメーションを用いて、立っていたプレイヤーが倒れるという動きを付けた。また、ParticleSystem を利用し、背景には雨を降らすことでより悔しさを感じられるようにした。これらにより完成したゲームオーバー画面が以下図 32,33 である。これら二つを見比べることによりプレイヤーに動きがある事が確認できる。

図 32：GameOver シーン
（プレイヤーが倒れる前）

図 33：GameOver シーン
（プレイヤーが倒れた後）