ポートフォリオ

総合学園ヒューマンアカデミー横浜校

ゲームカレッジ　プログラマー専攻

髙田　海渡

目次

プロフィール

氏名　:髙田海渡	性別　:男
年齢　:24歳	
希望職種　　:プログラマー	

所持スキル	
言語	C/C++, C#, ActionScript3.0, CSS, HTML
環境	Visual Studio 2022, Unity, GitHub
開発支援 ツール	Adobe Animate, Adobe illustrator, Office Word, Excel, PowerPoint

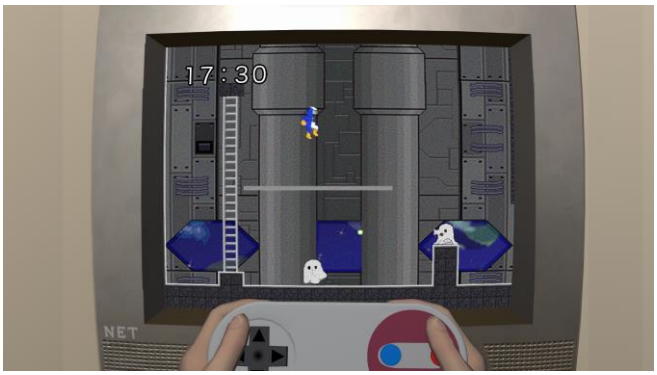
自己PR

大学を卒業後に専門学校に入学し、ゲーム制作とプログラミングを学習してきました。チーム制作では主にメインプログラマーを務め、新しいことに挑戦しつつ、わからないところは要点をまとめて先生などに相談し、作品のクオリティー向上に努めてきました。

現在は、DirectXの勉強をし理解を深めています。今後は、DirectXを用いたゲームの作成に挑戦しようと考えています。

Unity 2D横スクロールアクションゲーム

作品名：Tube Traveler



開発環境	Unity 2022 3.10, GitHub
使用ツール	Microsoft Visual Studio 2022
使用言語	C#
動作環境	PC
制作期間	3ヵ月(2023年9月～12月頃)
制作人数	プランナー2人, プログラマー3人, CGデザイナー3人
担当箇所	リードプログラマー

ゲーム概要	テレビの中と外の二つの操作を活用するシンプルな2Dアクションゲーム。内の操作では、テレビ画面に映っているキャラクターを操作し敵や障害物を飛び越え進んでいく。外の操作では、テレビを見ているプレイヤーを操作し、テレビを叩いたりチャンネルを切り替えることで、画面にうまく影響を与えてキャラクターの道行を有利にすることができる。

Unity 2D横スクロールアクションゲーム

◆ 落とし穴に落ちた際の復帰地点

落とし穴に落ちた際に、指定したタグの中で一番近いものを取得し、その地点にプレイヤーを移動させる機能を作成しました。

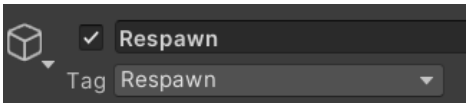


図.リスポーンタグ

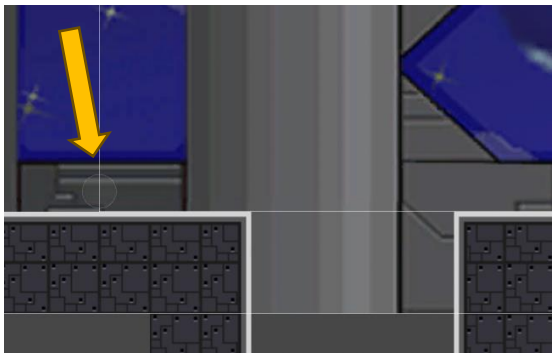


図.復帰地点(矢印の個所)

空のゲームオブジェクトを用意し、

respawnsにrespawnタグが付いたオブジェクトを格納する。

格納したものの中で、

一番playerに近いものを、

searchRespawnに代入する。

returnで値を返すことで、

一番近いrespawn情報を保存する。

```
// リスポーン
nearRespawn = Respawn();
```

```
// 一番近くのRrspawnを取得する
1 個の参照
public GameObject Respawn()
{
    float nearDistance = 0;
    GameObject searchRespawn = null;

    // Respawnタグを取得
    GameObject[] respawns = GameObject.FindGameObjectsWithTag("Respawn");

    if (respawns.Length == 0)
    {
        return searchRespawn;
    }

    foreach (GameObject respawn in respawns)
    {
        // 一番近いrespaenの座標をdistanceに代入
        float distance = Vector3.Distance(respawn.transform.position, transform.position);

        if (nearDistance == 0 || nearDistance > distance)
        {
            nearDistance = distance;

            searchRespawn = respawn;
        }
    }

    return searchRespawn;
}
```

```
// 一番近いRespwanタグに移動する
Vector3 position = transform.position;
transform.position = nearRespawn.transform.position;
```

図.プレイヤースクリプト

これによって落とし穴ごとに、別々のオブジェクトを用意する必要がなく、ステージ作成に割く時間が削減できたので、作品のクオリティーアップにつながったと考えている。

Unity 2D横スクロールアクションゲーム

◆ インターフェースを用いたダメージ計算

ダメージを与える側から容易にダメージの数値を変えることができる機能を作成した。

IDamageableというインターフェースを用意することによって、敵側の数値を変えることで、与えるダメージを変えることができる。

IDamageable

```
public interface IDamageable
{
    7 個の参照
    public void Damage(int damage);
}
```

図.Idamageableのインターフェース

```
//敵のダメージ値
[SerializeField]
private int enemyDamage = 1;
```

```
// ダメージ判定
var damaged = collision.gameObject.GetComponent<IDamageable>();
damaged.Damage(enemyDamage);
```

図.敵のスクリプトのダメージ判定部分

Player側でIDamageableを参照している部分。

ここで、敵側で入力した数値を呼び出しダメージ計算を行う。

```
// ダメージ判定
7 個の参照
public void Damage(int damage)
{
    //判定を遅らせる
    StartCoroutine(Wait());

    SEManeger.Instance.DamageSe();

    // 無敵状態
    player.layer = 8;
    rigidbody.velocity = new Vector2(0, 2);
    HPcount(damage);
    StartCoroutine(Invin());
}

// 体力カウント
2 個の参照
private void HPcount(int damage)
{
    //ダメージ
    currentHP -= damage;
    isDamaged = true;

    if (currentHP != 0)
    {
        foreach (GameObject HPFrame in HP)
        {
            // HPUIを表示
            HPFrame.SetActive(true);
        }
        StartCoroutine(HPtime(HP[currentHP].transform.GetChild(0).gameObject));
    }
```

図.プレイヤー側のスクリプト

Unity 2D横スクロールアクションゲーム

◆ ライティング

コンセプトが昭和時代だったので当時の色味に近いライティングを再現した。

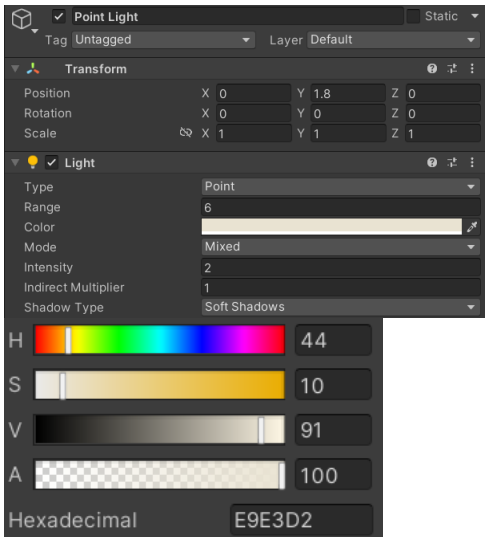


図.Point Light

3D空間を制作した後、　部屋の範囲を照らす少し黄色いライトを配置したところ全体的に暗く、光を強くしても白飛びしてしまった。

なので、Reflection Probeを設置し床から反射する光を再現することで、部屋が明るくなり実際の部屋に近づいた。

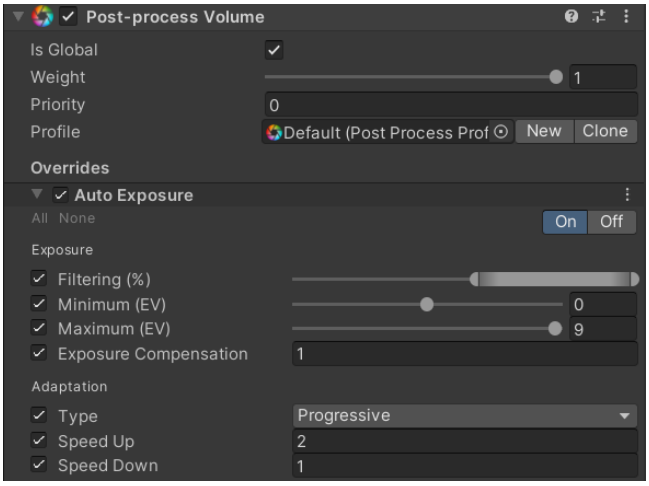


図. Post-process Volume(3D空間)

図.3D空間

上:Reflection Probe反映前 下:反映後

また、Post-process Volumeを設置し光がより室内の光に近づくように制作しました。

Unity 2D横スクロールアクションゲーム

◆ 画面エフェクト

遊んでいるテレビが、アナログテレビという設定だったため、画面がざらついているようなエフェクトを作成した。

Post-process VolumeのGrainというエフェクトを追加し、画面のざらつきを再現した。

図.ゲーム画面

上: Post-process Volume反映前 下:反映後

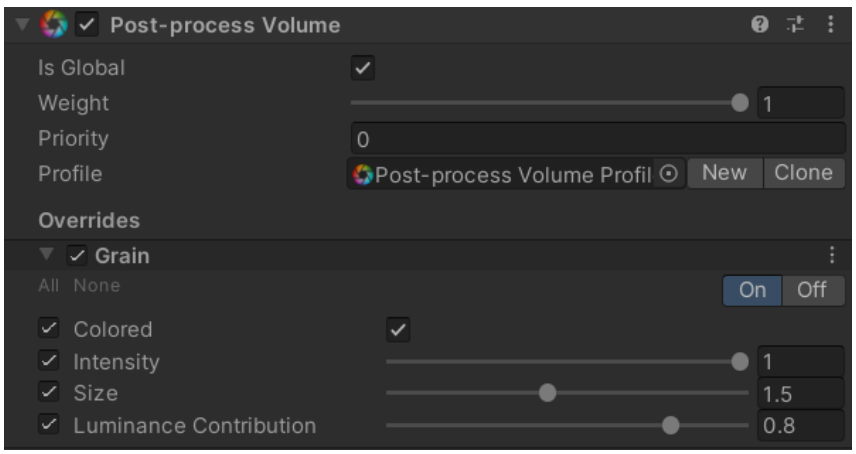


図. Post-process Volume　(ゲーム空間)

エフェクトやライティングなどは今回の制作で初めてだったので、今後も世界観に合うの設定を行えるように学習を進めたい。

制作後記

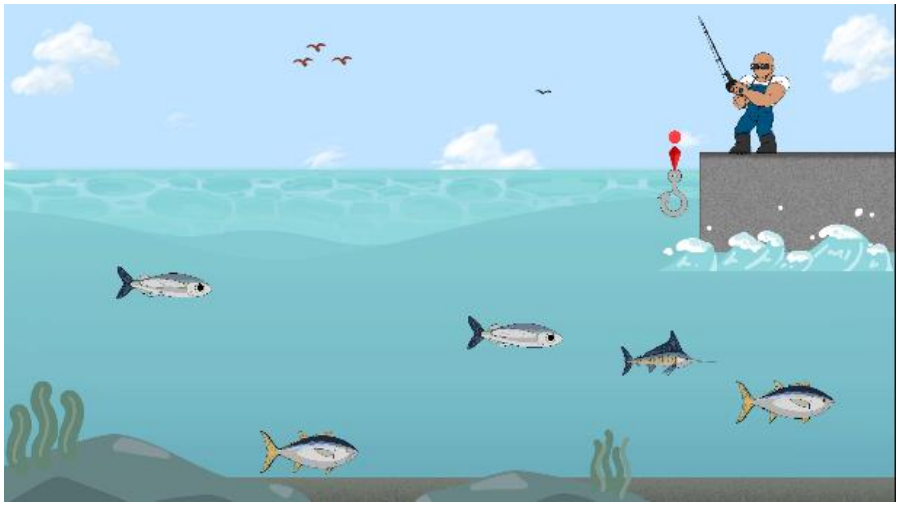
今回の制作では、プログラマーのメンバーが入院や技能五輪に出場するといったことがあり、一人で制作しなければいけない期間が数週間あった。

来れないメンバーの分も担当し、コードを工夫することでステージ作成の負担を減らし、完成に漕ぎ着けた。

また友人にゲームをプレイしてもらうことで、バグを発見し、すぐに直すことで作品のクオリティーアップにつながったと考えている。

Unity 2D釣りアクションゲーム

作品名：Fresh Fish



開発環境	Unity 2022 2.15, GitHub
使用ツール	Microsoft Visual Studio 2022
使用言語	C#
動作環境	PC
制作期間	1週間(2023年8月頃)
制作人数	プランナー1人, プログラマー2人, CGデザイナー2人
担当箇所	リードプログラマー

ゲーム概要	魚を釣り、釣った魚を操作して配達するゲームです。釣りフェーズと魚フェーズがあり、釣りフェーズでは釣りたい魚を狙って釣りあげ、ボタンを連打するミニゲームをし、魚フェーズへ。魚フェーズでは、釣りフェーズのミニゲームの記録で飛ぶ速度が変わり、障害物などをよけながらより長い距離を飛んで、寿司屋などの近くに着地し、高いスコアを目指しましょう。

◆泳ぐ魚の実装

指定した時間泳ぐと反転する魚を実装した。

図.指定時間で反転する魚

①魚の種類ごとの泳ぐ速度
②泳いでいる時間
③反転する時間

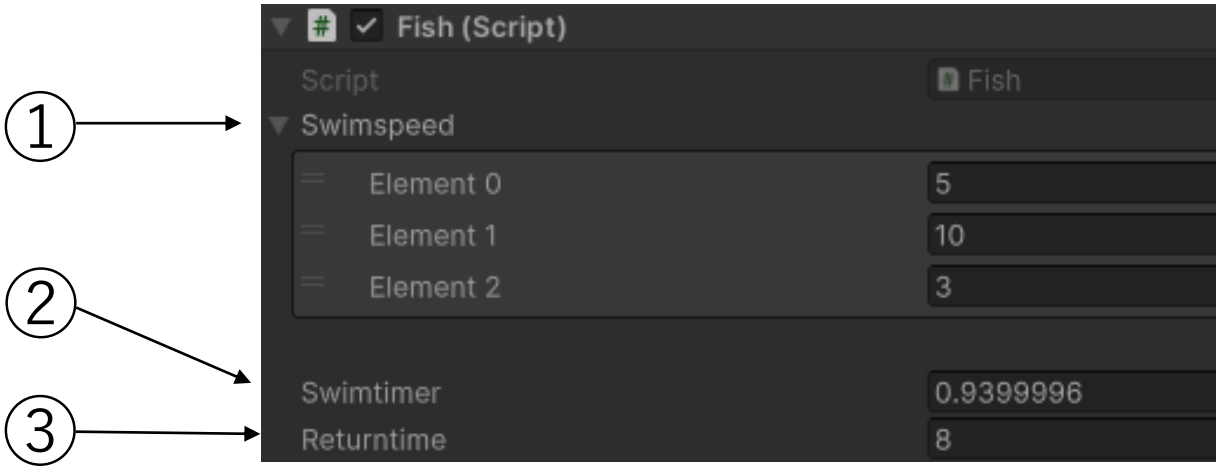


図.魚のインスペクター

プランナーがレベルデザインしやすいように、Unity上の数値を変えることで速さと往復するタイミングを簡単に調整できるようにしました。

◆ミニゲーム部分の実装

釣った際に発生するミニゲームを実装した。
このミニゲームの記録によって次の魚を操作するフェーズの速度が決まる。バーの座標を次のシーンに引き継ぎ、その数値によって速度を決めるようにした。

ミニゲーム中とそうでない状態を分けミニゲーム中は餌の操作などのほかの操作を受け付けないようにした。

```
// 衝突判定
Unity メッセージ | 0 個の参照
public void OnCollisionEnter2D(Collision2D collision2D)
{
    // 餌と魚がぶつかった際に呼び出される
    if (collision2D.gameObject.tag == "Feed")
    {
        // 泳いでいる状態をfalseにする
        swim = false;
        // isHitがtrueになるとミニゲームが開始される
        isHit = true;

        // ミニゲームのゲージを表示させる
        gaugeUI.Show();
```

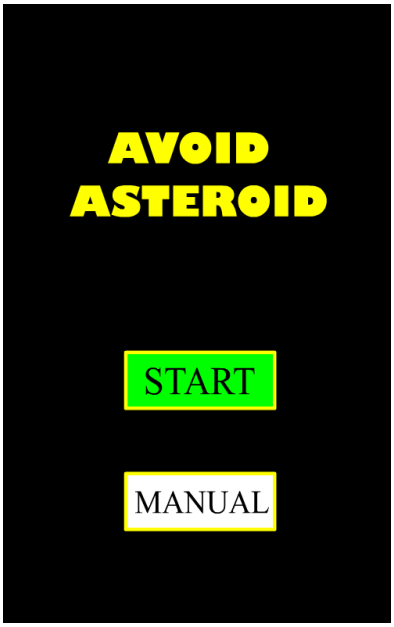
図.魚の状態を決めている部分のスクリプト

ミニゲームが終わった際に、斜め方向に力をかけることで自然に次のシーンにつながるようにした。

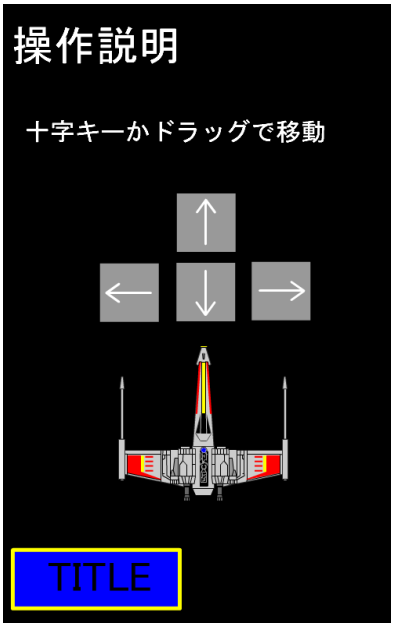
Animate制作

2Dシューティングゲーム

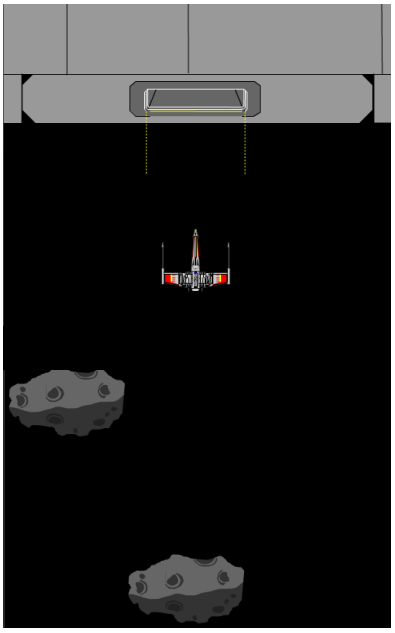
〈タイトル画面〉



〈操作説明〉



〈ゲーム画面〉



開発環境	Adobe Animate
使用言語	Action Script 3.0
制作期間	3日
制作人数	1人

【ゲーム概要】
宇宙船をマウスでドラッグするか十字キーで操作し、隕石をよけて目的地を目指すゲーム。
隕石にぶつかってしまうとゲームオーバーになる。
【補足】
このゲームはイラストを含め、すべて自分で制作した。
工夫した点は、タブレットなどでもプレイできるようにマウスとキーで操作できるようにしたこと。