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　2年

髙田　海渡

目次

プロフィール

氏名　:髙田海渡	性別　:男
年齢　:25歳	
希望職種　　:プログラマー	

所持スキル	
言語	C/C++, C#, ActionScript3.0, HTML /CSS, JavaScript
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 引っ越しアクションゲーム

作品名：スペース・ムービング

開発環境	Unity 2022.3.19f1, GitHub
使用ツール	Microsoft Visual Studio 2022
使用言語	C#
動作環境	PC
制作期間	4ヵ月(2024年3月～6月)
制作人数	プランナー1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	『家具を投げ入れ、高得点を狙うスコアゲーム』 ボトル投げのようにびっちり立つことを快感にする 引っ越しアクションゲーム。

Unity引っ越しアクションゲーム

◆ スティックの回転方向の判定

回転方向の判定

コントローラーの
スティックが時計周りか
反時計回りかを判定する
機能を作成した。

図.動かしている画面

過去の入力値と現在の入
力値の角度の差を計算し、
指定した値より大きいか
小さいかで判定を行う。

この機能を作成したことによりクレーンの振り子部分
の操作性が格段に上がった。

```
void DoJudgeAngle()
{
    // 過去の角度を計算
    float beforAngle = Mathf.Atan2(beforemoveInput.y, beforemoveInput.x) * Mathf.Rad2Deg;
    // 現在の角度を計算
    float nowAngle = Mathf.Atan2(nowmoveInput.y, nowmoveInput.x) * Mathf.Rad2Deg;
    // 2つの差を計算
    float AngleDifference = Mathf.DeltaAngle(beforAngle, nowAngle);

    // 差が指定した値より大きかったら反時計回り
    if(AngleDifference > rotateJudge)
    {
        // 時計回りの判定をfalse
        if(clockwise)
        {
            clockwise = false;
        }
        // 反時計回りの判定をtrue
        counterclockwise = true;
    }
    // 差が指定した値より小さかったら時計回り
    else if (AngleDifference < -rotateJudge)
    {
        // 反時計回りの判定をfalse
        if (counterclockwise)
        {
            counterclockwise = false;
        }
        // 時計回りの判定をtrue
        clockwise = true;
    }

}
```

図.回転方向を判定するスクリプト

Unity引っ越しアクションゲーム

◆ 配列を用いた投げる家具の管理

オブジェクトの管理

配列を用いて、投げる家具の管理を行った。

クレーンが家具を投げ終えて所持していない場合のみ、こちらで指定した家具をクレーン側で呼び出す。

この機能を作成したことにより、Unity上で投げる家具を簡単に設定することができたので、新し家具やステージの追加が容易になった。

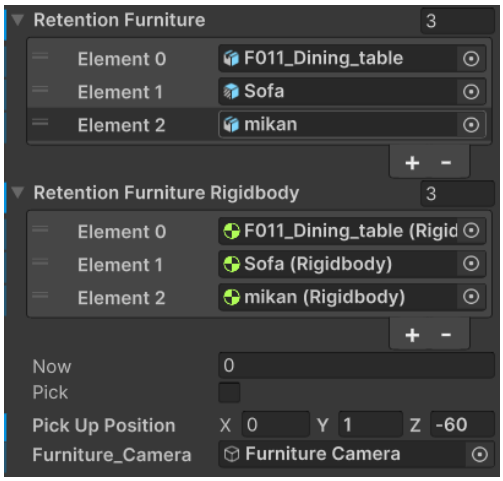


図.Unity上で配列に格納

```
// 現在持っている家具
[SerializeField]
public GameObject[] retentionFurniture;
// 現在持っている家具のRigidbody
[SerializeField]
public Rigidbody[] retentionFurnitureRigidbody;

// 現在持っている家具の配列用の番号
[SerializeField]
public int now;

// 家具を拾うエリアに入っているかの判定
[SerializeField]
public bool pick = false;

// 家具を拾う地点
[SerializeField]
private Vector3 pickUpPosition;

// 家具を追従するカメラ
[SerializeField]
private GameObject furniture_Camera;

// 他スクリプトで変数を呼び出せるようにする宣言
19 個の参照
public static PickUpFurniture pickUpFurniture { get; private set; } = null;
```

```
void Start()
{
    // sceneRootScriptを事前に参照
    sceneRootScript = GameObject.Find("SceneRoot").GetComponent<SceneRoot>();
    // sceneRootに現在何番目の家具かを送る。
    sceneRootScript.nowFurnituregetset = now;


    pickUpFurniture = this;
}
```

```
// 家具が土台か家にあり、家具を拾う範囲にいるときのみ反応
if (Furniture_.furniture_.onFurniture && pick)
{
    // 現在持っている家具の配列の番号をプラス
    now += 1;

    // 家具を持っている状態をtrueにする
    Crane._Crane.haveFurniture = true;
    // 家具を投げた時に家具に追従するカメラの状態をtrueにする
    Crane._Crane.targetFurniture = true;

    // 次の家具を呼び出す
    retentionFurniture[now].gameObject.SetActive(true);

    // sceneRootに現在何番目の家具かを送る。
    sceneRootScript.nowFurnituregetset = now;

    // SEを再生
    SEManeger.Instance.PickUpFurnitureSE();

    // 遅延させないとgameObjectが指定した地点に移動しないのでコルーチンで遅延
    StartCoroutine(transformFPosition());
}
```

```
IEnumerator transformFPosition()
{
    // 判定を遅らせる
    yield return new WaitForSeconds(0.01f);

    // 次の家具を指定した地点に移動させる
    retentionFurniture[now].transform.position = pickUpPosition;
}
```

図.家具を管理するスクリプト

Unity引っ越しアクションゲーム

◆ 家具を投げたときの成功・失敗判定

クリア判定

家具が家か地面に設置した時に成功か失敗か判定するスクリプトを作成した。

家具を投げている仕様上、家具が家から跳ねて地面に落ちてしまうことが多発したため、家具が静止してから判定を行うように変更した。

```
private void OnCollisionEnter(Collision collision)
{
    // 地面に接地
    if (collision.gameObject.CompareTag("Ground"))
    {
        // 家や土台に接地した後で地面に接地した時用に判定
        isGronded = true;
        isHoused = false;
        onFurniture = false;

        // タイマーをリセット
        collision_Timer = 0;

        // HingeJointをtrueにして家具をもう一度接続する
        _Hand.gameObject.SetActive(true);
        // クレーンの投げている状態をfalse
        Crane._Crane.isthrowed = false;

        // リトライ機能のために親オブジェクトを空にする
        this.gameObject.transform.parent = null;
    }
    // クリア範囲か家具に接地
    if (collision.gameObject.CompareTag("Onfurniture") ||
        collision.gameObject.CompareTag("Furniture"))
    {
        // 地面に接地した後で家や土台に接地した時用に判定
        isGronded = false;
        isHoused = true;

        // タイマーをリセット
        collision_Timer = 0;

        // クレーンの投げている状態をfalse
        Crane._Crane.isthrowed = false;

        // FixedJointを破壊
        Destroy(_fixedjointComponent);

        // リトライ機能のために親オブジェクトを空にする
        this.gameObject.transform.parent = null;
    }
}
```

```
// 家具が動いておらず地面に接地している場合
if (this.rigidbody.IsSleeping() && isGronded)
{
    onGronded = true;
}
// 家具が動いておらず家か土台に接地している場合
if (this.rigidbody.IsSleeping() && isHoused)
{
    if (collision_Timer > 0)
    {
        onHoused = true;
    }
}
```

図.家具が接地した際に成功・失敗 を判定するスクリプト

図.成功・失敗のUI

Unity引っ越しアクションゲーム

◆ 家具に追従するカメラ

家具用のカメラ

家具を投げたら、投げた家具に追従するカメラを作成した。

家具によって大きさが違うため、個々で家具とカメラの距離を調整できるように作成した。

この機能により、家具がどこに飛んで行ったかのわかりやすさと、ゲームの映像の面白さにつながった。

```
// カメラとの距離
private Vector3 offset;

// カメラから離れる距離
[SerializeField]
private float xPosition = 0;
[SerializeField]
private float yPosition = 2.5f;
[SerializeField]
private float zPosition = -5;

// Start is called before the first frame update
void Start()
{
    // カメラの家具からの距離を指定
    offset = new Vector3(xPosition, yPosition, zPosition);
}

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

private void FurnitureTargetCamera()
{
    if (Crane._Crane.targetFurniture)
    {
        // カメラを指定した地点に移動
        transform.position =
            PickUpFurniture.pickUpFurniture.retentionFurniture
            [PickUpFurniture.pickUpFurniture.now].transform.position + offset;
    }
}
```

図.家具を追従するカメラ

Unity引っ越しアクションゲーム

◆ チュートリアル

ステート管理を用いて、チュートリアルを作成した。

指定された動きしかできないようにすることで、素早くゲームの操作方法が理解できるように意識して作成した。

チュートリアルに関しては自分から作りたいと感じたので、仕様の作成、CGへの依頼、実装までを一人で担当した。

```
17 個の参照
enum TutorialState
{
    // 右に動かす
    MoveRight,
    // 左に動かす
    MoveLeft,
    // 収集地点に向かう
    GoPickPosition,
    // 家具を拾う
    PickUpFurniture,
    // 家具を投げる
    ThrowFurniture,
}
// 現在のチュートリアルの状態
[SerializeField]
TutorialState tutorialState = TutorialState.MoveRight;
```

```
void Update()
{
    switch (tutorialState)
    {
        case TutorialState.MoveRight:
            UpdateForMoveRightState();
            break;
        case TutorialState.MoveLeft:
            UpdateForMoveLeftState();
            break;
        case TutorialState.GoPickPosition:
            UpdateForGoPickPositionState();
            break;
        case TutorialState.PickUpFurniture:
            UpdateForPickUpFurnitureState();
            break;
        case TutorialState.ThrowFurniture:
            UpdateForThrowFurnitureState();
            break;
        default:
            break;
    }
}

1 個の参照
private void UpdateForMoveRightState()
{
    if (cranePosition.transform.position.z > moveRightTutorial)
    {
        // UIを表示
        checkUI.SetActive(true);
        phoneUI.SetActive(true);

        // クレーンを動けなくする
        Tutorial_Crane.tutorial_crane.canMove = false;
        Tutorial_Crane.tutorial_crane.moveInput =new Vector3(0,0,0);

        // ステートを次の状態に変更
        tutorialState = TutorialState.MoveLeft;
    }
}
```

図.チュートリアルのスクリプト

図.チュートリアル画面

制作後記

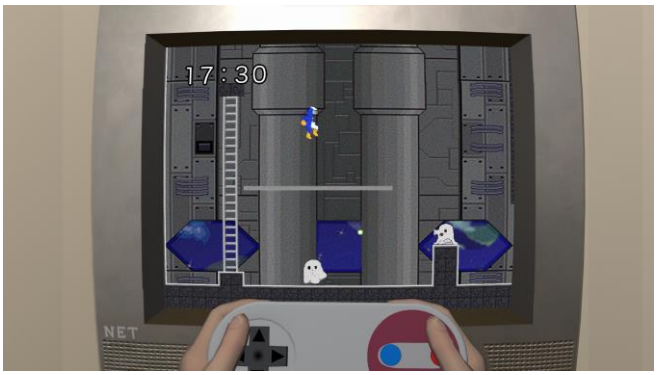
今回の制作では、リードプログラマーを担当した。前回の制作でもリードプログラマーを担当しており、その際に一人で担当した箇所が多かったと感じたため、今回は他のメンバーの手が空かないように能力に応じて均等に仕事を割り振ることを意識した。

制作の終盤には、プランナーが不在となってしまい、代わりに追加の仕様を考え、それに伴い必要なUI・家具・エフェクトをCGに発注した。

また、残っているメンバーを集め、足りない要素を出し合いクオリティーアップに努めた。

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

復帰地点の作成

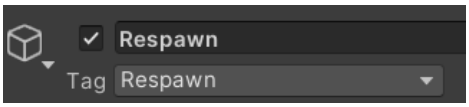


図.リスポーンタグ

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た。

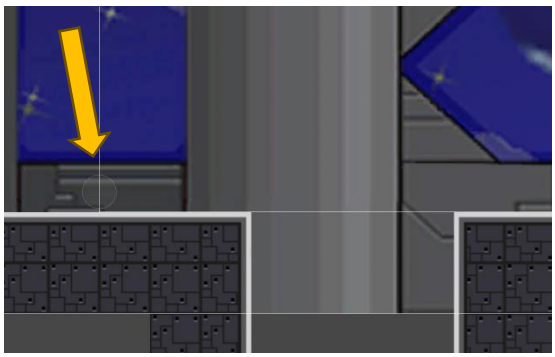


図.復帰地点(矢印の個所)

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

この機能を作成したことで、与えるダメージを簡単に変更することができ、新しい敵を容易に作成できる。

Unity 2D横スクロールアクションゲーム

◆ タイトル画面での1人称視点のゲームカメラ

タイトル画面での一人称視点カメラを作成した。

仕様書にはなかった機能だが、手が空いた時に作成しプランナーに見せたところ、プレイヤーへの没入感が深まるとの結論になり、正式に採用された。

図.視点移動させた画面

```
public class PlayerCamera : MonoBehaviour
{
    [SerializeField]
    private Transform targetCamera = null;

    // 視点の上下制限角度を調整する
    [SerializeField]
    private float topClamp = 80;
    [SerializeField]
    private float bottomClamp = -80;

    // 現在のMove入力値
    Vector2 lookInput;

    // 現在のカメラの角度
    [SerializeField]
    private float cameraRotationX = 0;

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {
        float deltaTimeMultiplier;

        // ゲームパッド用
        deltaTimeMultiplier = Time.deltaTime;

        // キャラクターの方角を変更
        transform.Rotate(0, lookInput.x * deltaTimeMultiplier, 0);
        //カメラの上下角度
        cameraRotationX += lookInput.y * deltaTimeMultiplier;
        // 上下角度
        cameraRotationX = Mathf.Clamp(cameraRotationX, bottomClamp, topClamp);

        targetCamera.localRotation = Quaternion.Euler(cameraRotationX, 0, 0);
    }
```

図.1人称カメラのスクリプト

Unity 2D横スクロールアクションゲーム

◆ 母親に怒られて操作不能になる機能

操作不能になる機能

画面を短時間で複数回叩くと母親が来て怒ることで、操作不可能になるスクリプトを作成した。

叩くごとに怒り値が上昇し、指定した数値を上回ると操作不能になる。

プランナーがUnity上で調整しやすいように意識して作成した。

作成したことでテレビを叩きすぎることへの抑制となり、ゲームバランスの調整につながった。

図.怒られている画面

```
void Update()
{
    timer += Time.deltaTime;

    // 怒り値を減少させる
    if (timer >= resetTimer)
    {
        anger -= calmDown;
        timer = 0;
    }
    // 怒り値が0以下にならないようにする
    if (anger <= 0)
    {
        anger = 0;
    }
    // 怒り値がmaxAngerの値まで行ったら0にする
    if (anger >= maxAnger)
    {
        anger = 0;

        StartCoroutine(canControl());
    }
}
```

```
IEnumerator Mash()
{
    // 怒り値が上昇
    anger += calm;
    // 連打防止
    cantMash = true;

    // 怒り値が100以上になったら操作不能状態にする
    if (anger >= maxAnger)
    {
        isSleep = true;
        MotherSE.Instance.PlayAngerMother();
        HandAnimation.Instance.SetShockAnime();
    }

    yield return new WaitForSeconds(3f);

    // 操作不能状態を解除
    cantMash = false;
    isSleep = false;
}
```

図. Unity上で設定できる数値

図. 母親が怒るスクリプト

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。

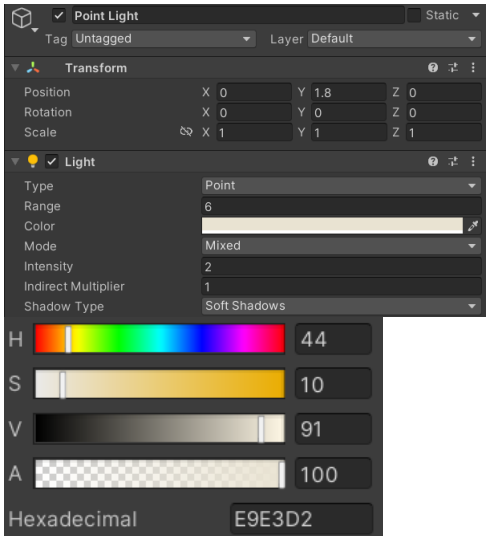


図.Point Light

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。

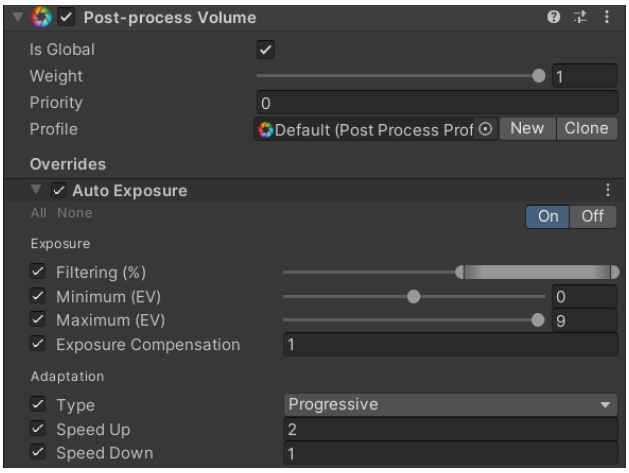


図. Post-process Volume(3D空間)

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、アナログテレビ特融の画面のざらつきを再現した。

作成したことでアナログテレビ感が増し、ゲームへの没入感が深まった。

図.ゲーム画面

上: Post-process Volume反映前 下:反映後

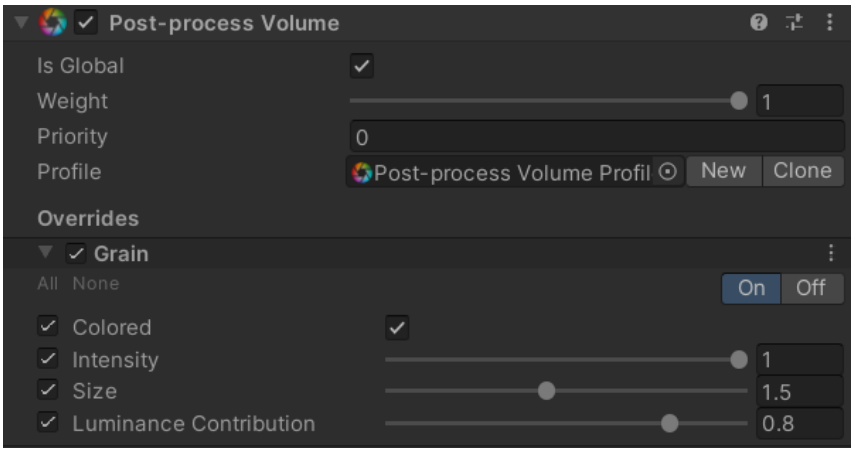


図. Post-process Volume　(ゲーム空間)

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

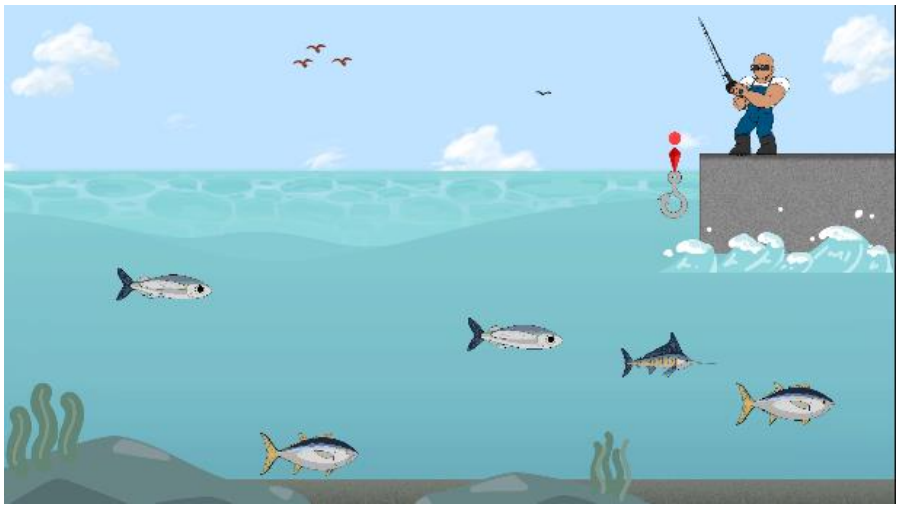
今回の制作では、プログラマーのメンバーが入院や技能五輪に出場するといったことがあり、一人で制作しなければいけない期間が数週間あった。

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

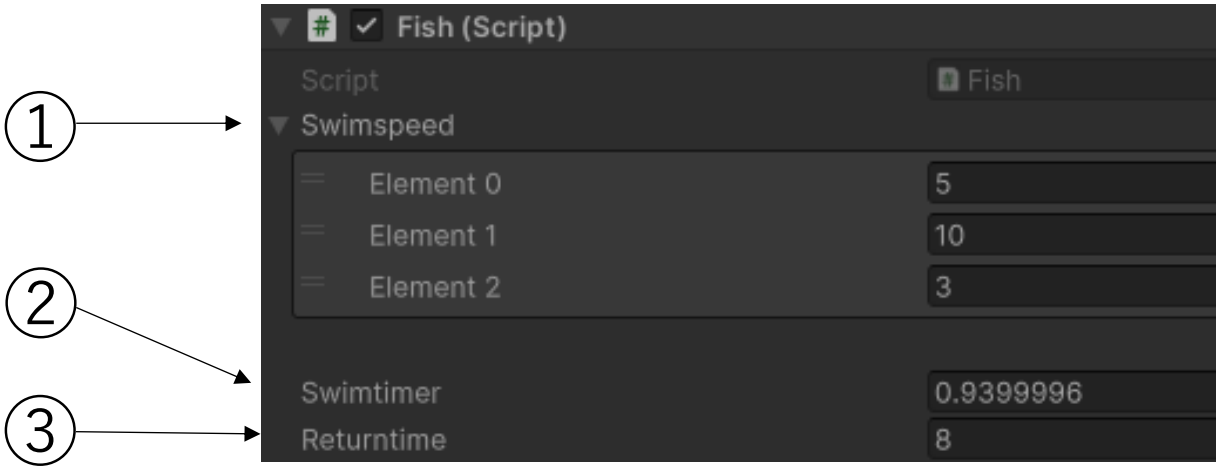


図.魚のインスペクター

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。

DirectX

◆DirectXを用いた円の描写

```
// 分割数
constexpr int splitCount = 64;
// 頂点データの配列
constexpr UINT vertexCount = splitCount + 1;
VertexPositionColor vertices[vertexCount] = {};
// 原点
vertices[0] = VertexPositionColor{ {  0.0f, 0.0f, 0.0f }, { 1.0f, 1.0f, 1.0f, 1.0f } };

// 円周上の点
float svc = 0.015625f;
float circleColor = svc;
for (int vertex = 0; vertex < splitCount; ++vertex) {
    constexpr float radius = 0.5f;
    float angle = 2 * 3.14f / splitCount * vertex;
    float x = radius * sinf(angle);
    float y = radius * cosf(angle);
    vertices[1 + vertex] = VertexPositionColor{ { x, y, 0.0f }, { circleColor, circleColor, circleColor, 1.0f } };

    circleColor += svc;
}

// インデックスデータの配列
constexpr UINT indexCount = splitCount * 3;
UINT32 indices[indexCount] = {};
int first = 0;
int second = 1;
int third = 2;
for (int i = 0; i < splitCount; ++i) {
    indices[first] = 0;
    indices[second] = 1 + i;
    indices[third] = 2 + i;

    first += 3; second += 3; third += 3;
}
indices[splitCount * 3 - 3] = 0;
indices[splitCount * 3 - 2] = splitCount;
indices[splitCount * 3 - 1] = 1;
```

図.図形の表示する座標を指定している箇所

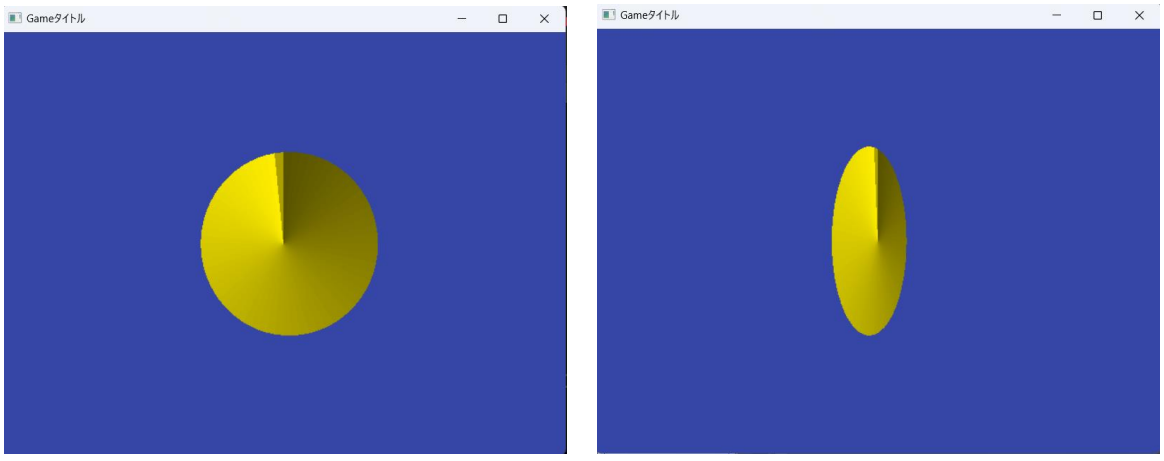


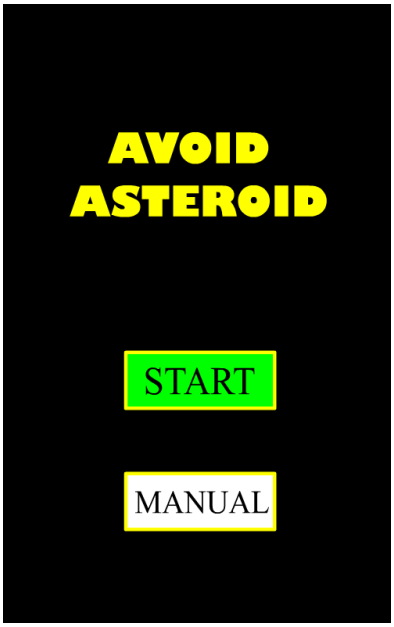
図.表示した円

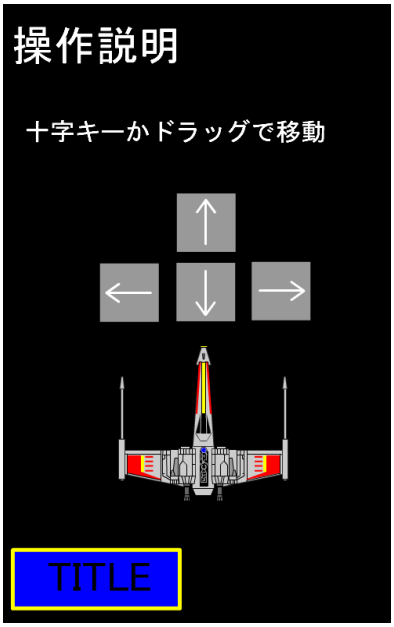
DirectXを用いて円を表示し、回転させた。
三角関数を用いて、複数の三角形を表示させることで円に見えるようにしている。

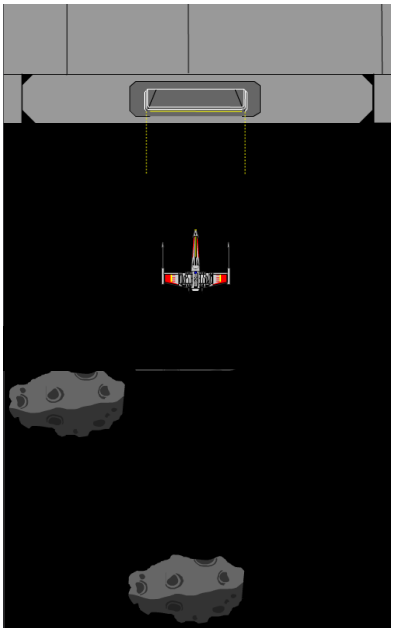
Animate制作

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〉







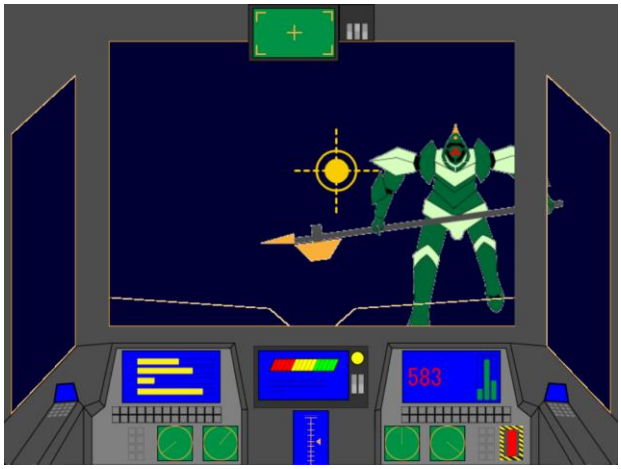
開発環境	Adobe Animate
使用言語	Action Script 3.0
制作期間	3日
制作人数	1人

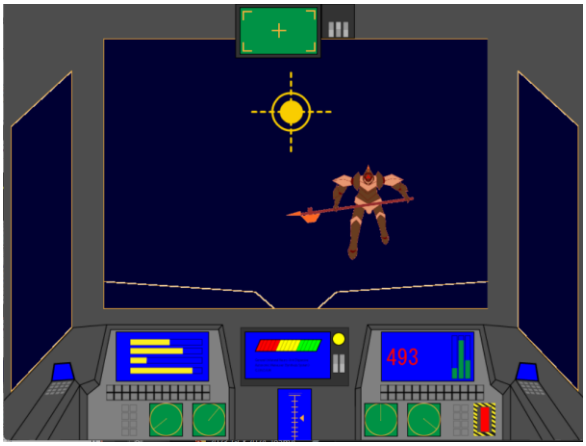
【ゲーム概要】
宇宙船をマウスでドラッグするか十字キーで操作し、隕石をよけて目的地を目指すゲーム。
隕石にぶつかってしまうとゲームオーバーになる。
【補足】
このゲームはイラストを含め、すべて自分で制作した。
工夫した点は、タブレットなどでもプレイできるようにマウスとキーで操作できるようにしたこと。

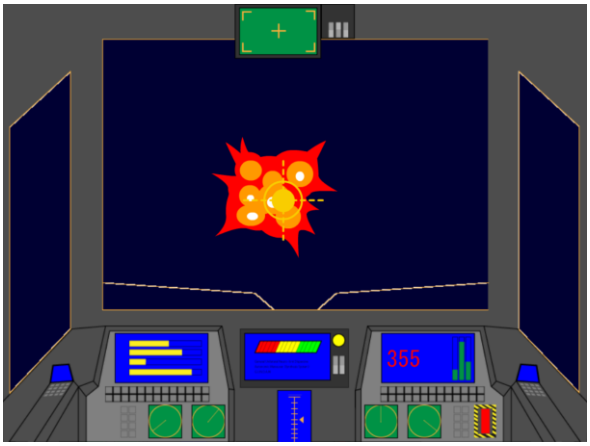
Animate制作

2Dシューティングゲーム

〈ゲーム画面〉







開発環境	Adobe Animate
使用言語	Action Script 3.0
制作期間	2日
制作人数	1人

【ゲーム概要】
カーソルを敵機に合わせてクリックし撃破するゲーム。敵機が近づきすぎるかクリックできないとゲームオーバー。

【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、敵機の出現位置をランダムで出現させることと、クリックしてから数秒後に爆発演出を加えたこと。