ポートフォリオ

総合学園ヒューマンアカデミー横浜校
ゲームカレッジ研究生プログラマー専攻

安藤周冴

目次

自己紹介

・スキルシート

Unityチーム制作作品

・コンテスト応募作品
クマさんのスライム掃除

・ゲーム大賞応募作品
ANOMAREA~アノマリア~

・GFF2023応募作品
ドーナツは食べられたい
~シェフに見つかるな~

Animate作品

・フリーシュート
~ここが正念場だ！~

・有害UFO殲滅隊

自己紹介

氏名	：安藤周冴（あんどうしゅうご）	性別	：男
生年月日	：2003年11月5日	年齢	：20歳
希望職種	：プログラマー		

対応可能言語

C++/Direct 3D11,
C#,Action Script,
HTML/JavaScript,
PHP, SQL

使用開発環境

Unity, Visual Studio,
Animate, Photoshop
Illustrator,Fire Alpaca64
GitHub

Office

PowerPoint / Excel / Word

主な情報共有ツール

Slack, Discord, GitHub, Google Drive

Unity

コンテスト応募作品

一次審査通過 !!

ワイナリーを管理するクマさん
冬眠から覚めたら
汚いスライムに汚されてた！！

作品名　　：クマさんのスライム掃除
使用言語：C#
動作環境：Xboxコントローラー/キーボード・マウス
制作人数：プランナー３人 / プログラマー４人 /
　　CGデザイナー５人　　（計１２人）
制作期間：３ヶ月
担当場所：メインプログラマー
使用したツール：Microsoft Visual Studio 2022
　　Unity ver.2022.3.10f1 , GitHub

Unity

制作過程について

制作メンバー
期間前半
PL３名・PG４名・CG５名

期間後半
PG４名・CG４名

制作期間はカリキュラムとして行っているため３か月となっています。
しかし、制作開始後PLとPG/CGとでの口論が起こり、企画は廃案、PLの脱退となりました。
上記の口論等は１か月半に及び、残ったPG/CGで新案を練り、PGリーダーである私が責任者となり１か月半で制作を行いました。

前ページの制作人数は、初期のメンバーを記させていただきましたが。

本作品の紹介動画はCGリーダーが撮影し、元PLに編集を依頼させていただきました。

Unity

作品概要

このゲームは？

クマさんが冬眠から目覚めると、管理していたワイナリーが汚いスライムたちに汚されてしまっていた！！
汚いスライムたちを掃除して綺麗なワイナリーを取り戻そう。

制作でのこだわりは？

ナビメッシュを使用したスライムの移動処理。
プレイヤーのアニメーション調整。
ライティングの調整。

１つのステージで構成されており、現在２つ目のステージを作成中です。１つ目のステージをクリアした後に解放されるようにする予定です。

操作方法

キーボード＆マウスの場合

移動：WASD
攻撃：Space
ポーズ：Escape
視点移動：マウス

Xboxコントローラーの場合

Unity

担当箇所

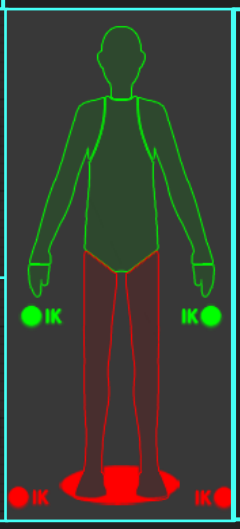
・プレイヤー
・スライム
・カメラ

・ライティング
→Lightmapping Setting
・GlobalVolume
→Bloom

の調整

・UI全般管理（←制作後半に担当）
・ミニマップ
・音量設定
・ナビメッシュ
・オブジェクト配置
・タイル

プレイヤーアニメーション



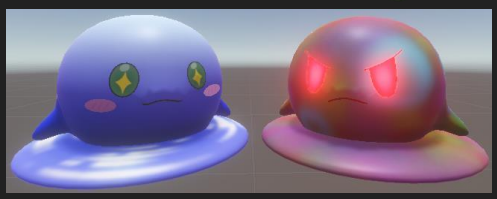
プレイヤーの攻撃時、移動中または停止中のどちらの場合でもアニメーションが違和感なく動くように、レイヤーを活用しました。

Humanoidの画像は、BaseBodyレイヤーで使用しています。

スライムの２つの状態

綺麗なスライム

汚れた床に侵入すると汚くなってしまう。



汚いスライム

綺麗な床を汚していく

攻撃を当てることで綺麗になる

Unity

担当箇所

ステージは室内ですが、室内ライティングをリアルにしてしまうと世界観に合わないと判断したため、グラフィカ—と絵作りについて相談し設定しました。
天井を付けただけで雰囲気が理想に近づきました。

天井
左有　　右無

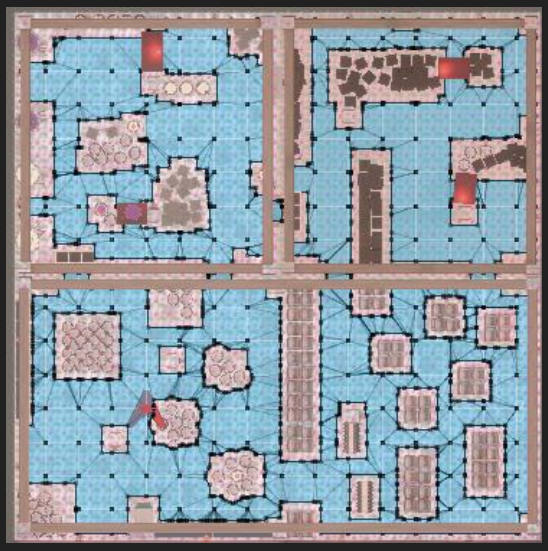
こだわりとして、スライムが汚れる、汚いスライムがタイルを汚すときはどちらも、
OnTriggerExitを使用しています。

すべてのタイルにばらつきを持たせるため、ゲーム開始時にランダムにタイルを回転させています。

```
Unity メッセージ|0 個の参照
void Start()
{
    // ランダムなY軸回転を加える
    var random = Random.Range(0, 4);
    dirtyTile.localRotation = Quaternion.Euler(90f, 90 * random, 0f);
    Dirty();
```

Unity

担当箇所



NavMesh[ナビメッシュ]

ナビメッシュを使用したスライムの移動。
右の画像の**水色のエリア**がナビメッシュで設定した**スライムの移動区域**です。

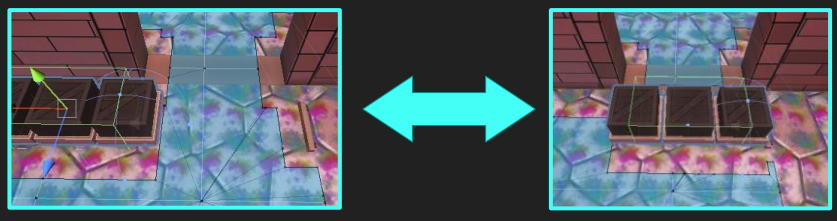
右上のステージにナビメッシュ用のターゲット（ゴール地点）が**５６個**あり、到着したらランダムに次のターゲットに進む処理を記述しています。
今回のエージェントはスライムであり、ナビメッシュではオブジェクトが一定の速度で進むことから、**スライムらしさ**を出すために**一定時間ごとに２つの速度を交互に代入する**処理を作りました。

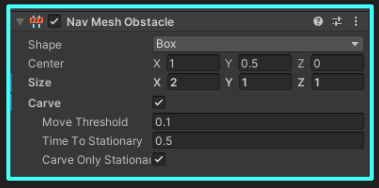
```
/// <summary>ランダムにターゲット（移動地点）を指定</summary>
2 個の参照
void SetRandomTarget()
{
    if (targets.Length >= 0)
    {
        // 次のターゲットの位置を目的地に設定。
        navMeshAgent.SetDestination(targets[currentTargetIndex].transform.position);

        // 次のターゲットのインデックスを更新。
        currentTargetIndex = Random.Range(0, targets.Length-1);
    }
}
```

```
/// <summary>数値を交互に切り替える。</summary>
1 個の参照
void SwapValues()
{
    navMeshAgent.speed = slow;
    slow = fast;
    fast = navMeshAgent.speed;
}
```

```
void Update()
{
    // 補間の進捗を更新。
    time += Time.deltaTime * speed;

    // 一定の間隔ごとに処理を行う。
    if (time >= interval)
    {
        // パラメータを0に戻す。
        time = 0.0f;

        // 数値を交互に切り替える。
        SwapValues();
    }

    // 数値を補間して変数に代入。
    float lerpedValue = Mathf.Lerp(fast, slow, time / interval);

    if (navMeshAgent.remainingDistance < 0.6f)
    {
        SetRandomTarget();
    }
}
```

スライムが進む道を阻むために用いるオブジェクトに**NavMeshObstacle**コンポーネントをアタッチすることで、右の画像のように**水色のエリア**が途切れ、ギミックとして機能させることに成功しました。





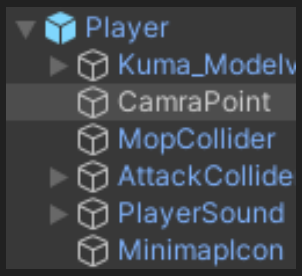
その他Staticオブジェクトにもアタッチしていて、めり込む等のバグが起こることを未然に回避しています。

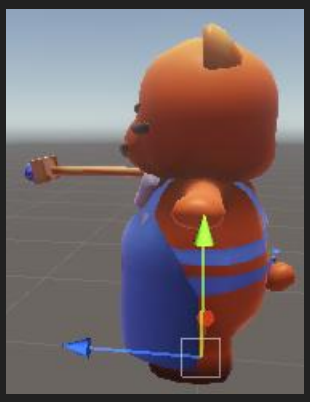
Unity

担当箇所

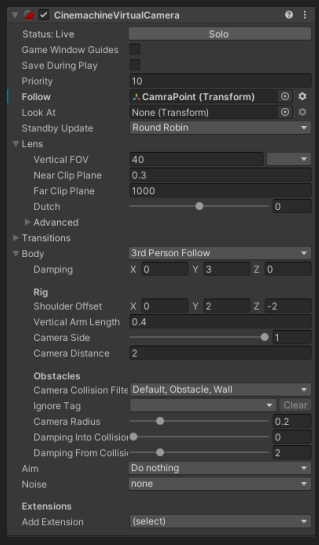
Cinemachineを用いたTPS視点カメラの実装

UnityにCinemachineをインストールし、TPS視点を実装しました

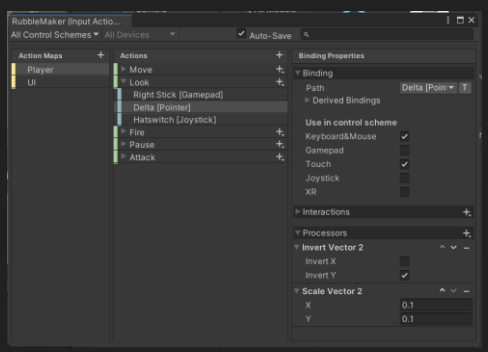




バーチャルカメラのインスペクターウィンドウ



Lookの中が、カメラ操作用のデバイスです。画像左下はカメラの感度です。



マウス・ゲームパッド操作でカメラが向いている方向を正面とし、違和感なくプレイ出来るようにする処理

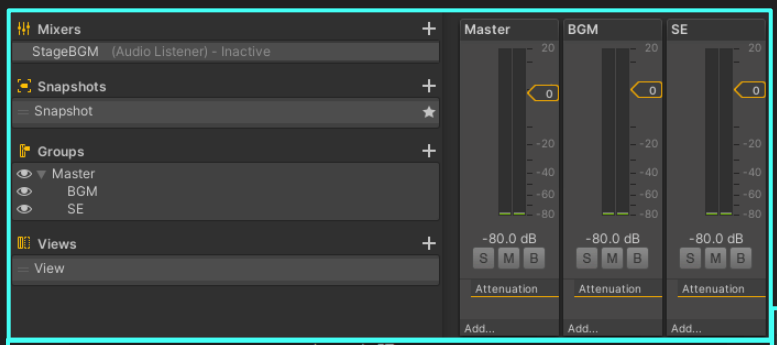
```
void Update()
{
    if (!isSleeping)
    {
        // ユーザー入力の大きさを計算[0, 1]。
        var speed = moveInput.magnitude;

        // ユーザーからの入力を3次元化。
        var inputDirection = new Vector3(moveInput.x, 0f, moveInput.y);

        // 移動入力がある場合。
        if (moveInput != Vector2.zero)
        {
            // カメラの方位角を基準にして、入力指示された方位角(Degree)を計算。
            targetHeading = Mathf.Atan2(inputDirection.x, inputDirection.z) * Mathf.Rad2Deg;
            targetHeading += Camera.main.transform.eulerAngles.y;
        }
        // カメラの正面を基準に移動。
        var targetDirection = Quaternion.Euler(0.0f, targetHeading, 0.0f) * Vector3.forward;
        moveBehaviour.Move(targetDirection * speed);
    }
}
// Updateの後に処理される。
0 Unity メッセージ | 0 個の参照
void LateUpdate()
{
    if (!isSleeping)
    {
        // マウス操作かゲームパッド操作かに応じた処理。
        bool isMouseDevice = (playerInput.currentControlScheme == "Keyboard&Mouse");
        float deltaTimeMultiplier = isMouseDevice ? 1 : Time.deltaTime;
        // カメラの左右上下角度。
        cameraRotationY += lookInput.x * deltaTimeMultiplier;
        cameraRotationX = Mathf.Clamp(cameraRotationX, bottomClamp, topClamp);
        targetCamera.rotation = Quaternion.Euler(cameraRotationX, cameraRotationY, 0);
    }
}
```

Unity

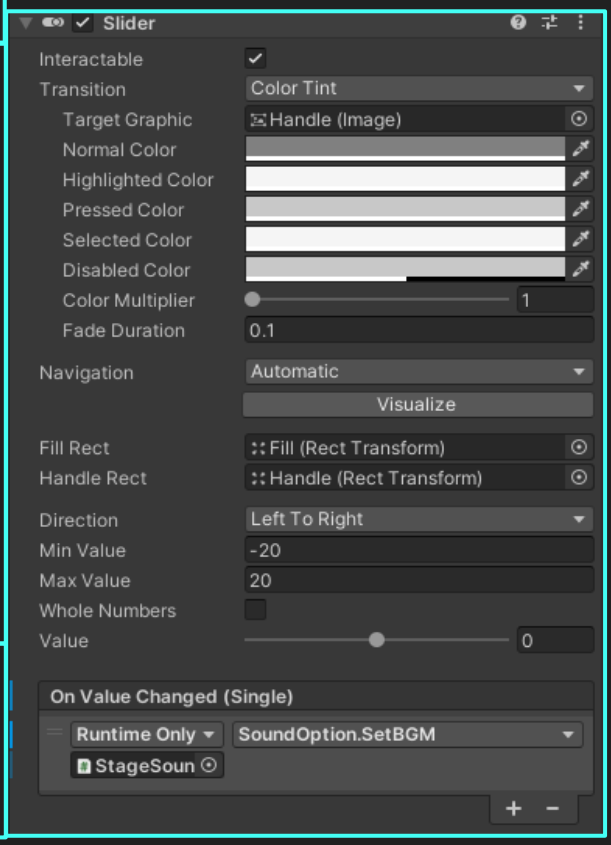
担当箇所

AudioMixer
[オーディオミキサー]

音量設定画 ↓




```
Unity メッセージ|0 個の参照
void Start()
{
    Time.timeScale = 1;
    audioMixer.GetFloat("BGM_Volume", out float bgmVolume);
    bGMSlider.value = bgmVolume;
    audioMixer.GetFloat("SE_Volume", out float seVolume);
    sESlider.value = seVolume;
}

/// <summary>BGMを調整</summary>
/// <param name="volume"></param>
0 個の参照
public void SetBGM(float volume)
{
    audioMixer.SetFloat("BGM_Volume", volume);
}

/// <summary>SEを調整</summary>
/// <param name="volume"></param>
0 個の参照
public void SetSE(float volume)
{
    audioMixer.SetFloat("SE_Volume", volume);
}
```



オーディオミキサーを使用し、
シーンをまたいでも設定を
引き継げるようにしました。

タイトルシーン

ステージシーン

Unity

担当箇所

ミニマップ

スライムの条件分岐でミニマップ内のアイコンの切り替えを行っています。


綺麗なスライムのアイコン


汚いスライムのアイコン

```
/// <summary>ミニマップ用プレイヤー追尾カメラ</summary>
Unity スクリプト (1 件のアセット参照)|0 個の参照
public class MinimapChaseCamera : MonoBehaviour
{
    [SerializeField, Tooltip("Playerを指定")]
    private Player player;

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {
        var posi = player.transform.position;
        posi.y = transform.position.y;
        // カメラ位置の更新。
        transform.position = posi;
    }
}
```

```
private void Update()
{
    DispIcon();
    iconTransform.rotation = Quaternion.Euler(0, 0, 0);
    if(slime.IsDirty)
    {
        icon[0].SetActive(true);
        icon[1].SetActive(false);
    }
    else
    {
        icon[0].SetActive(false);
        icon[1].SetActive(true);
    }
}
```

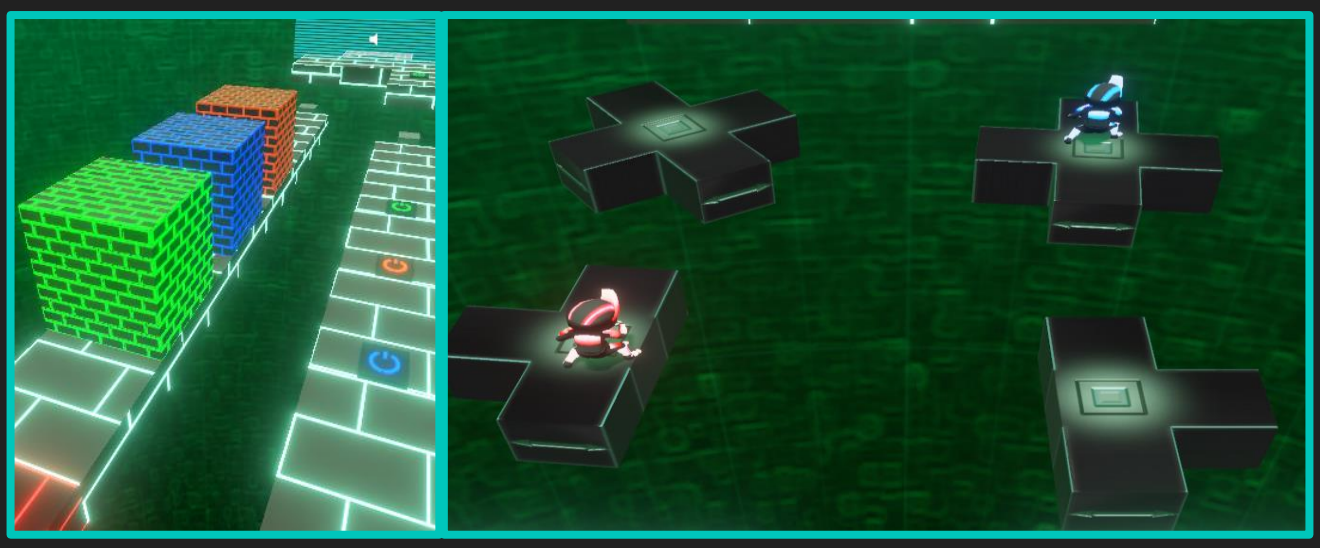
```
/// <summary>アイコン表示を更新する</summary>
1 個の参照
private void DispIcon()
{
    // アイコンを表示する座標。
    var iconPos = new Vector3(iconTarget.position.x, defaultPosY, iconTarget.position.z);
    var centerPos = new Vector3(minimapCamera.transform.position.x, defaultPosY, minimapCamera.transform.position.z);
    var offset = iconPos - centerPos;
    transform.position = centerPos + Vector3.ClampMagnitude(offset, minimapRangeRadius - rangeRadiusOffset);
}
```

Unity ゲーム大賞応募作品

３Dアクションゲーム

１人協力プレイ！？

１人で２体のキャラを操作しよう！！



作品名　：ANOMAREA~アノマリア~
使用言語：C#
動作環境：Xboxコントローラー / PC（キーボード）
制作人数：プランナー３人 / プログラマー２人 /
　　CGデザイナー４人　　（計９人）
制作期間：３ヶ月
担当場所：メインプログラマー
使用したツール：Microsoft Visual Studio 2022
　　Unity　ver.2021.3.18f1
GitHub, Fire Alpaca64

Unity

作品概要

このゲームは？

このゲームは『**１人協力プレイ**』をキャッコピーとし、
１人で２つのキャラを操作することで、難易度を高めに
設定しています。

制作でのこだわりは？

CGチームが考えた世界観を実現させるために
ライティングにより力を入れて制作を進めてきたこと。

二人操作を実現させるために、移動やジャンプの動きを
詳細に作りました。

操作方法

一人操作時の操作方法 <Entity状態>	二人操作時の操作方法 <Holo状態>
	

Unity

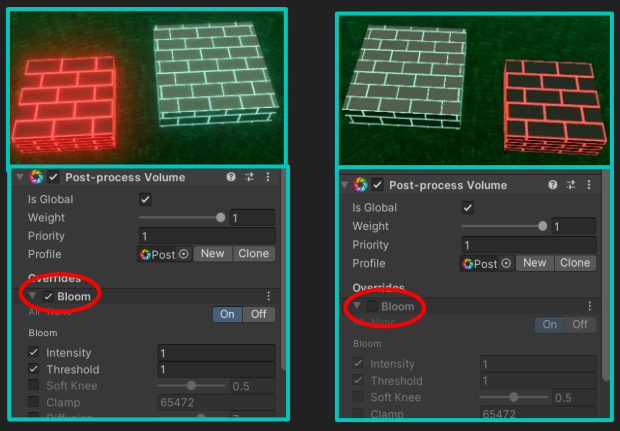
担当箇所

・ライティング
・ポストプロセス
　→Bloom
・ステージの配置
・デッドエリア
・ギミック等
・ゴール
・見えない壁のすべての配置

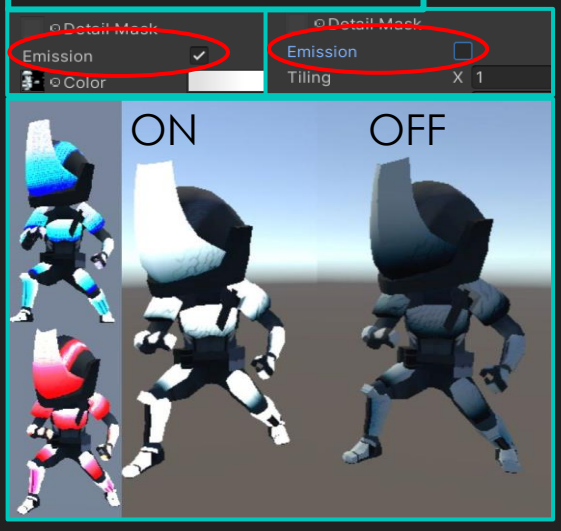
ライティングを真上に近い位置にして、ジャンプ時に影が足元にくるようにしました。
Bloomを調整して、想定しているバーチャル空間により近付けられるように意識しました。

ポストプロセス（Bloom）

ON ⟷ OFF

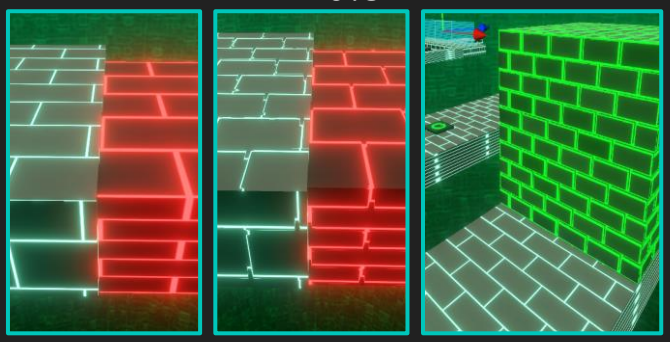


エミッション（マテリアル）



ハイトマップ（マテリアル）値＝X

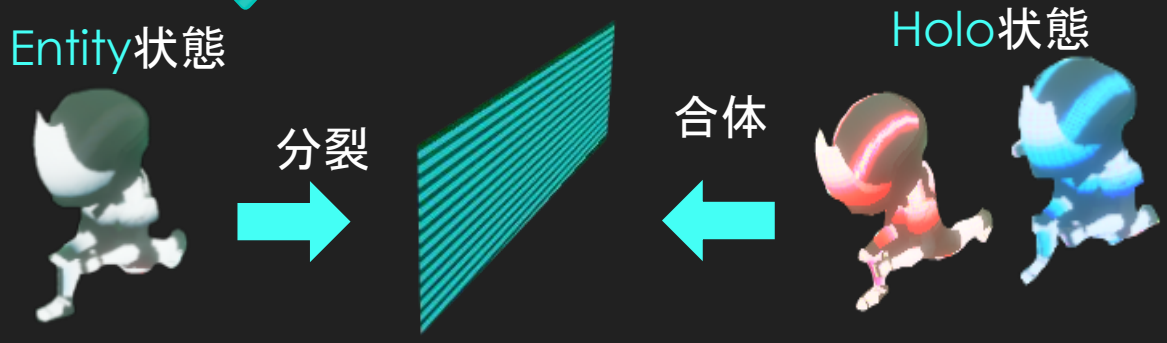
X ＝ 0　X ＝ 0.8　採用



[ポストプロセス](#)を導入しての[Bloom](#)と、マテリアルの[エミッション](#)を入れることにより、よりバーチャル空間に近づけることが出来ました。
[ハイトマップ](#)を導入しリアルなタイルを演出しましたが、このゲームでプレイヤーを目立たせるために、廃案となりました。
ギミックとして目立たせるために採用しれたものもあります。

Unity

担当箇所



プレイヤーの分裂

Entityがホログラムの壁をくぐり抜けたらこの関数の処理を開始。

分裂してHoloが0.3秒後に出現する。

```
/// <summary>分裂</summary>
1 個の参照
public async void Split()
{
    Entity.SetActive(!Entity.activeSelf);
    MB[1].PassageWall();
    MB[2].PassageWall();
    await Task.Delay(asyncData.playerModi
    Player.isSplittable = false;
    Holo1.SetActive(!Holo1.activeSelf);
    Holo2.SetActive(!Holo2.activeSelf);
    StageScene.instance.ChangeCamera();
}
```

プレイヤーの合体

Holoが２体ともホログラムの壁をくぐり抜けたらこの関数の処理を開始。

合体してEntityが0.3秒後に出現する。

```
// <summary>ホロ体合体</summary>
 個の参照
ublic async void HoloCombine()

    Holo1.SetActive(!Holo1.activeSelf);
    Holo2.SetActive(!Holo2.activeSelf);
    MB[0].PassageWall();
    await Task.Delay(asyncData.playerMod
    Player.isSplittable = true;
    Entity.SetActive(!Entity.activeSelf)
    StageScene.instance.ChangeCamera();
```

Unity

担当箇所

async - await

前回の制作で使った **Coroutine**
（IEnumerator-yield return）
より綺麗にまとめることの
できるasync – awaitを使用して
コンパクトにまとめました。

```
1 個の参照
public async void HoloCombine()
{
    Holo1.SetActive(!Holo1.activeSelf);
    Holo2.SetActive(!Holo2.activeSelf);
    MB[0].PassageWall();
    await Task.Delay(asyncData.playerModifications);
    Player.isSplittable = true;
    Entity.SetActive(!Entity.activeSelf);
    StageScene.instance.ChangeCamera();
```

分裂時の処理

- １人操作状態の座標を指定
- Holo体がそれぞれEntityの座標を参照し出現

合体時の処理

- 2人操作状態それぞれの座標を指定
- １人操作のとき、2人操作時の２体中間座標を計算し、そこの出現

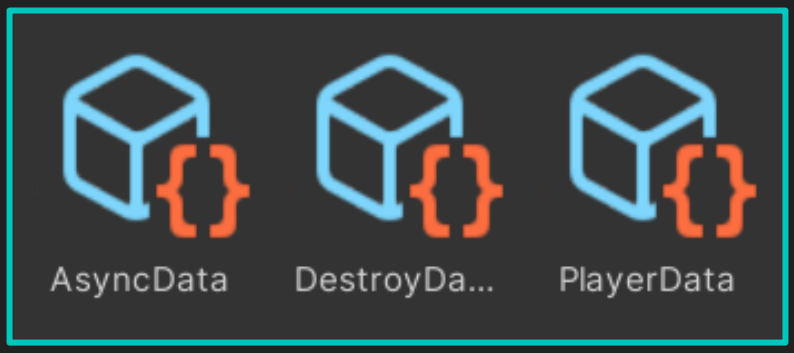
```
if (SharedVariables.isSplittable)
{
    Eposi = EntityRenderer.transform.position;
    Holo1.transform.position = new Vector3((Eposi.x - 1), Eposi.y, (Eposi.z + 1));
    Holo2.transform.position = new Vector3((Eposi.x + 1), Eposi.y, (Eposi.z + 1));
}
else
{
    Vector3 PositionB = Holo1.transform.position;
    Vector3 PositionC = Holo2.transform.position;
    Vector3 newPosition = (PositionB + PositionC) / 2f;

    // MoveBehaviourScriptのMoveTo関数を呼び出して、Entityを計算した座標に移動する。
    MB[0].MoveTo(newPosition);
}
```

```
MB[0].MoveTo(newPosition);
```

void MoveBehaviour.MoveTo(Vector3 position)
Entityの座標を保存

Unity

ScriptableObject



AsyncData　…　async awaitの時間を指定

DestroyData …　エフェクトの消滅の時間を指定

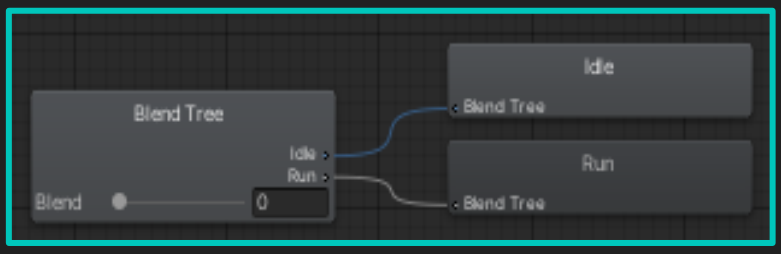
PlayerData　…　プレイヤーのパラメーターを

いくつか**マジックナンバー**が存在し、変更するたびすべてのスクリプトを修正しなければいけないのが大変だったので使用しました。
プランナーも**パラメーターを簡単にいじることができる**点がとても助かりました。


```
[AddComponentMenu("【プレイヤーのパラメーター】")]
[CreateAssetMenu(fileName = "PlayerData", menuName = "ScriptableObject/Player")]
Unity スクリプト | 1 個の参照
public class PlayerData : ScriptableObject
{
    public string PlayerName = "Entity";
    [SerializeField, Tooltip("速度の指定")]
    public int walkSpeed;

    [SerializeField, Tooltip("ジャンプ力の指定")]
    public int jumpPower;

    [SerializeField, Tooltip("回転速度の指定")]
    public int rotateSpeed;

    [SerializeField, Tooltip("判定の半径の指定")]
    public float chackRadius;

    [SerializeField, Tooltip("ジャンプアニメーションのジャンプ力の指定")]
    public float animationJumpPower;
}
```

Unity

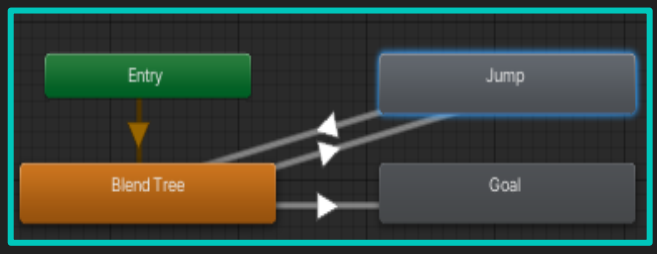
担当箇所



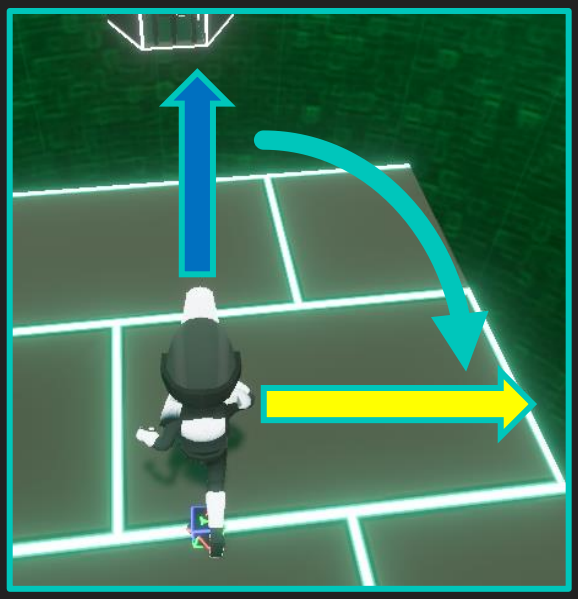
プレイヤーのアニメーション待機中と移動中を
ブレンドツリーで切替
今回の制作で初めて活用してみました。

```
// Idleアニメーション。
animator.SetFloat("Blend", 0f);
```

```
// Runアニメーション。
animator.SetFloat("Blend", input);
```



クォータニオンの活用

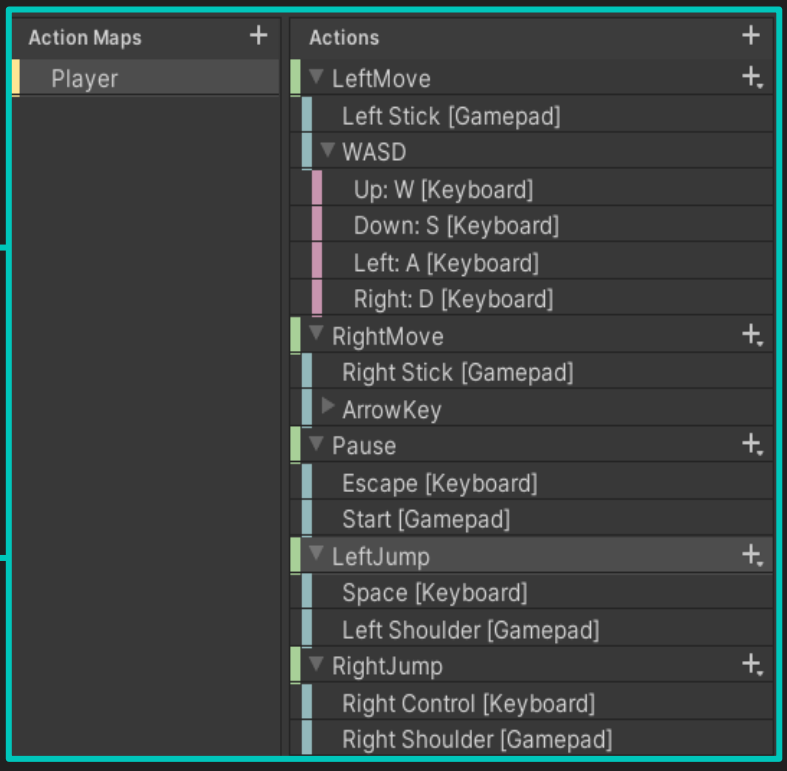


```
// プレイヤーの回転。
var rotation = Quaternion.LookRotation(moveInput);
transform.rotation = Quaternion.Slerp(transform.rotation, rotation, rotateSpeed * Time.deltaTime);
```

Unity

担当箇所

Input Actionを
使用した操作振り分け
キーボード
ゲームパッド
での操作に対応



２人操作の時、１つのコントローラーで２人を操作するのでそれぞれの左右の処理を分けて作っています。

```
/// <summary>Holo2ジャンプ動作</summary>
0 個の参照
public void OnRightJump(InputAction.CallbackContext context)
{
    if (context.phase == InputActionPhase.Started)
    {
        MB[2].Jump();
    }
}

/// <summary>Entity.Holo1ジャンプ動作</summary>
0 個の参照
public void OnLeftJump(InputAction.CallbackContext context)
{
    if (context.phase != InputActionPhase.Started) return;
    if (StageScene.instance.UInow && StageScene.instance.first) return;
    if (player.isSplittable)
    {
        MB[0].Jump();
    }
    else
    {
        MB[1].Jump();
    }
}
```

Unity

GFF応募作品

実写２Dアクションゲーム

本校のパフォーミングアーツカレッジ声優専攻の生徒に許可を得て掲載しています。

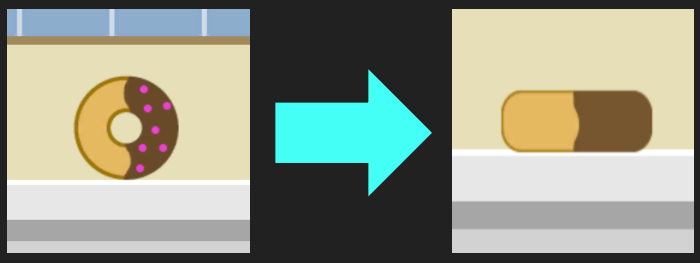
作品名　：ドーナツは食べられたい
　　~シェフに見つかるな~
使用言語：C#
動作環境：コントローラー
　　PC（キーボード/マウス）
制作人数：プランナー１人 / プログラマー２人
担当場所：サブプログラマー
開発期間：２ヶ月
使用したツール：Microsoft Visual Studio 2022
　　Unity　ver.2021.3.11f1, GitHub

Unity

操作説明

コントローラーでの操作は右の画像のようになっています。

Bボタンのしゃがむは倒れる動作を意味しています。



倒れると隠れられるオブジェクトがあるかも？？

難易度Easyの主人公クッキー

小さいオブジェクトにも倒れずに隠れられる。

難易度Normalの主人公ドーナツ

小さいオブジェクトに倒れて隠れることができる。

難易度Hardの主人公マカロン

倒れることができず、大きいオブジェクトにしか隠れられない。

Unity スクリプト

カウント用処理

画像の切り替え→

```
void Start()
{
    var totalItemCount = PlayerPrefs.GetInt("totalItemCount", 0);
    for (int index = 0; index < values.Length; index++)
    {
        values[index].sprite = numbers[totalItemCount % 10];
        totalItemCount /= 10;
    }
}
```

UIの表示切替

ボタンを押した際UIの
表示 / 非表示を切り替える

UIでは基本これを活用している

```
//このUIを非表示にします。
2 個の参照
public void Hide()
{
    backgroundImage.enabled = false;
    //子オブジェクトをすべて非アクティブ化
    foreach (Transform child in transform)
    {
        child.gameObject.SetActive(false);
    }
}

//このUIを表示します。
1 個の参照
public void Show()
{
    backgroundImage.enabled = true;
    //子オブジェクトのアクティブ化
    foreach (Transform child in transform)
    {
        child.gameObject.SetActive(true);
        selectedButton.Select();

    }
}
```

コルーチンを活用

IEnumerator
+
yield return

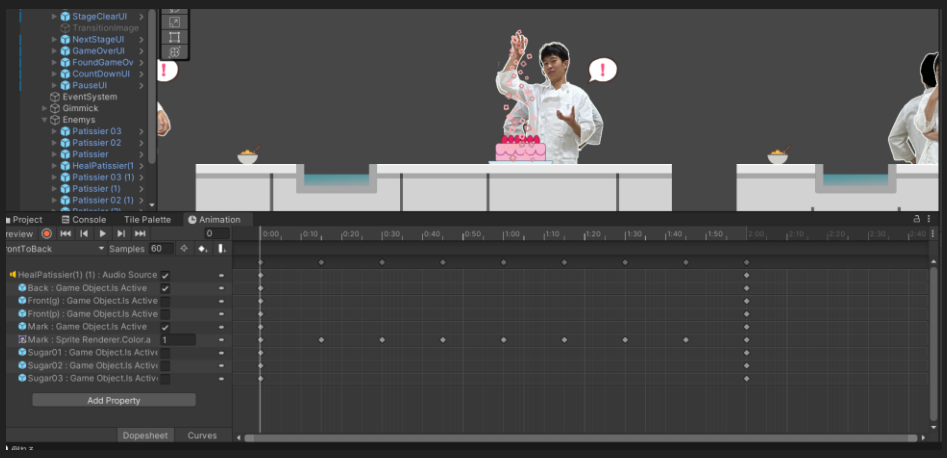
で時間差の処理

```
//  2秒待機後に次のシーンを読込み可能にします。
1 個の参照
IEnumerator OnStart()
{
    //2秒間待機
    yield return new WaitForSeconds(2);
    isLoadable = true;
}
```

Unity アニメーション

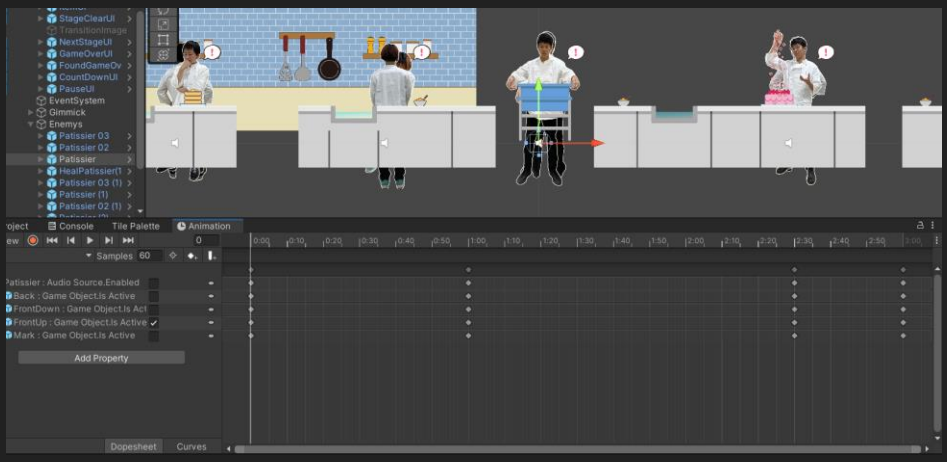
回復時

砂糖をかけているようにうまく見せるために、砂糖の画像を３枚使用しています。



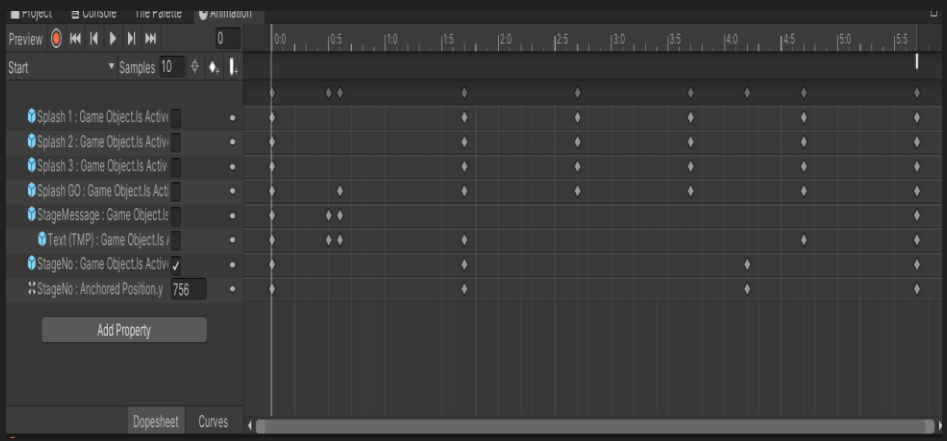
足場設置時

エネミーが振り向くタイミングに合わせるのに苦労した。



カウントダウンUI

前のページの看板の動きもここで行っています。

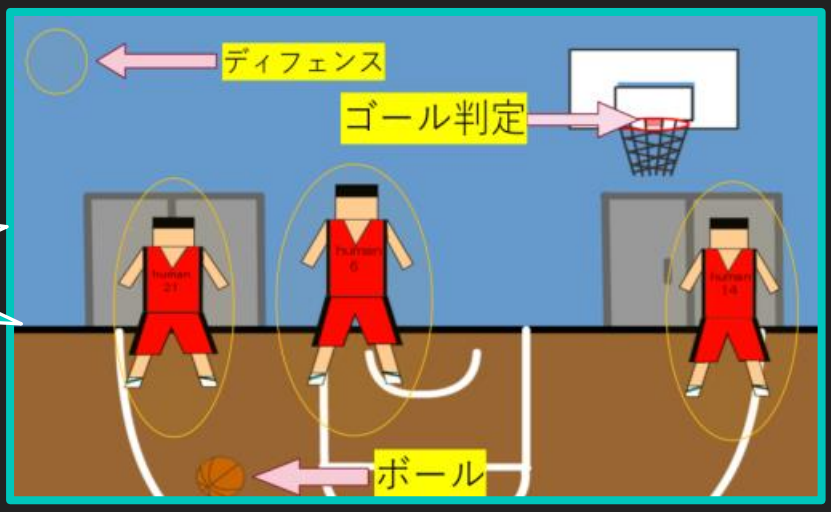


Animate

Animate作品①

『フリーシュート』
～ここが正念場だ！～

個人制作初作品



・使用言語
Action Script 3.0

画像内の<u>ボール</u>をマウス操作し、３人の<u>ディフェンス</u>をうまく躱して<u>ボールをゴールに持っていこう</u>。

Animate

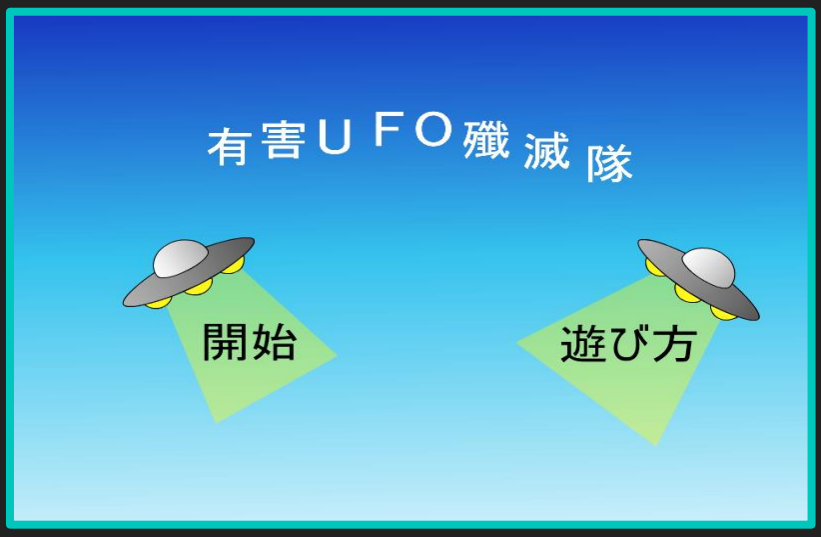
Animate作品②

『有害ＵＦＯ殲滅隊』

・制作時間
　合計10時間

・制作人数
　１人

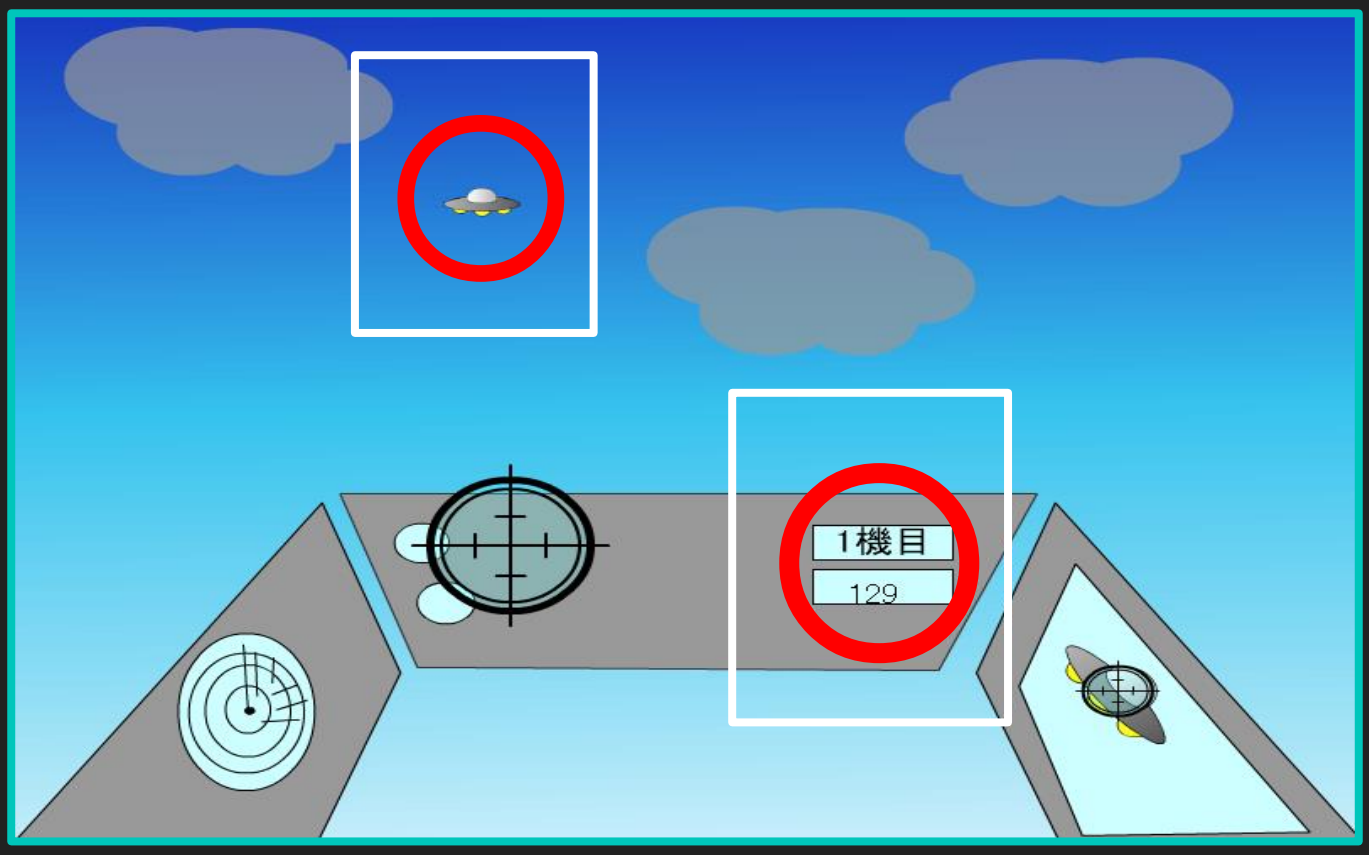
・使用言語
　Action script 3.0



ボタンの形を無難ではなく、UFOからの光にすることで、このゲームに適したものが完成したと思います。
こだわりでもあります。

操作方法：タッチ操作

Animate
プレイ画面



全10ラウンド 画面に出てる動くUFOを 殲滅していゲーム。
画面右中央付近にある数字は ラウンド数と、UFOの座標を表している。