ポートフォリオ

日本電子専門学校 ゲーム制作科 2年

プログラマー志望

照岡隆誠

プロフィール

氏名　　照岡隆誠(てるおかりゅうせい)
生年月日 2006年1月2日(19歳)
趣味　　ゲーム、筋トレ、漫画
連絡先　24ci0224@jec.ac.jp
github　https://github.com/Tel1urium/teru_
職種　　プログラマー志望

資格

2024年1月
セキュリティマネジメント試験 合格
2025年1月
情報検定 情報活用試験 3級 合格

自己PR

私の強みは、めげない心と最後まで諦めない粘り強さです。

プログラムでバグが生じても、その解決まで徹底的にやり遂げる根気があります。この諦めない姿勢と、複雑な技術課題への探求心を活かし、プロジェクトの品質向上に貢献します。

さらに、チームの雰囲気が悪くなった際には、積極的に声をかけ鼓舞することで、チーム全体を前向きな状態へ導き、プロジェクトの目標達成に向けて牽引する推進力となります。

開発ツール

GitHub

目次

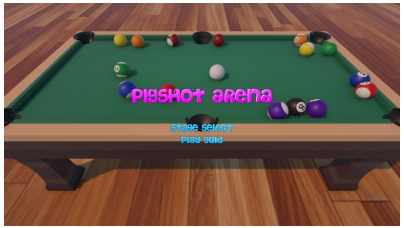
作品タイトル
ルセティカ
～ココのお料理天国～

開発環境
Unity

制作期間
2カ月

開発人数
14人

担当箇所
敵行動制御



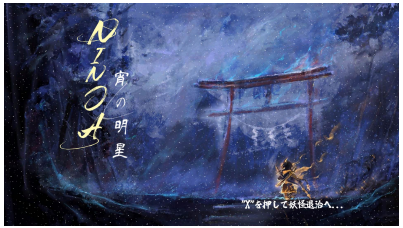
作品タイトル
PigShotArena

開発環境
Unity

制作期間
1カ月

開発人数
4人

担当箇所
プレイヤー行動制御
コントローラーのエントリー



作品タイトル
NINJA~宵の明星～

開発環境
Unity

制作期間
2カ月

開発人数
9人

担当箇所
敵行動制御、SE、BGM

01

ルセティカ~ココのお料理天国~

作品概要

食材を武器にして敵と戦うアクションゲームです。
敵を倒すと食材をドロップし、
それぞれ特性の違う食材を駆使してボスを倒します。

作品URL
https://github.com/Tel1urium/teru_/tree/main/Lucetica

ステートマシンの実装

敵の行動制御をするにあたってステートマシンを組みました。
敵の行動を複雑にするとコードの見通しが悪くなる課題があったため実装しました。
また敵の行動制御だけでなく汎用性を持たせる設計にしました。

```
EStateMachine<TOwner>
using System.Collections.Generic;
using UnityEngine;

public class EStateMachine<TOwner>
{
    /// 各ステートクラスはこのクラスを継承する
    public abstract class StateBase
    {
        public EStateMachine<TOwner> StateMachine;
        protected TOwner Owner => StateMachine.Owner;

        public virtual void OnStart() { }
        public virtual void OnUpdate() { }
        public virtual void OnEnd() { }
    }
    private TOwner Owner { get; }
    private StateBase _currentState; // 現在のステート
    private StateBase _prevState;    // 前のステート
    private readonly Dictionary<int, StateBase> _states = new Dictionary<int, StateBase>();
    public StateBase CurrentState => _currentState;
    public virtual void OnCollision(Collision collision) { }
    /// コンストラクタ
    public EStateMachine(TOwner owner)
    {
        Owner = owner;
    }
    /// ステートマシン初期化後にこのメソッドを呼ぶ
    public void Add<T>(int stateId) where T : StateBase, new()
    {
        if (_states.ContainsKey(stateId))
        {
            return;
        }
        // ステート定義を登録
        var newState = new T
        {
            StateMachine = this
        };
        _states.Add(stateId, newState);
    }
```

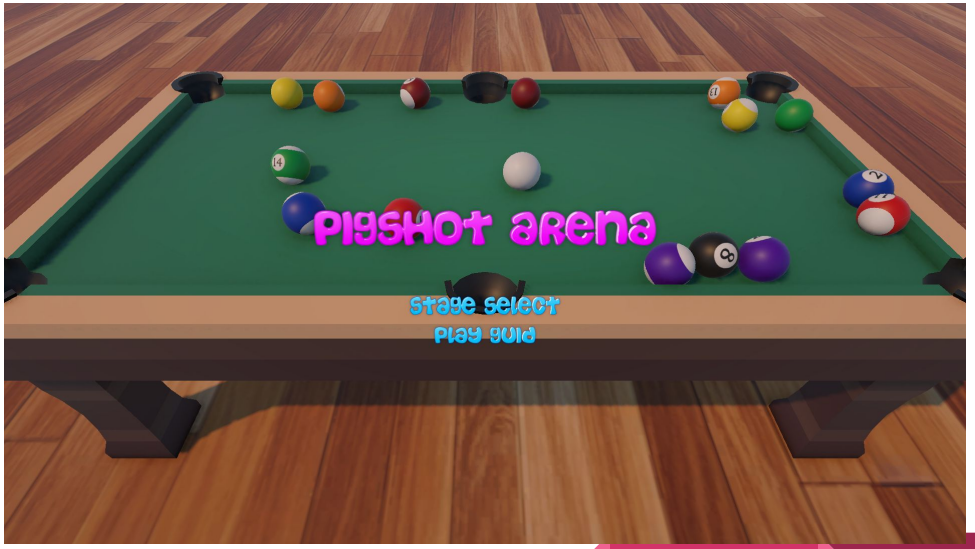
```
EStateMachine<TOwner>
public void OnStart(int stateId)
{
    if (!_states.TryGetValue(stateId, out var nextState))
    {
        return;
    }
    // 現在のステートに設定して処理を開始
    _currentState = nextState;
    _currentState.OnStart();
}
/// ステート更新処理
public void OnUpdate()
{
    _currentState.OnUpdate();
}
/// 次のステートに切り替える
public void ChangeState(int stateId)
{
    if (!_states.TryGetValue(stateId, out var nextState))
    {
        return;
    }
    // 前のステートを保持
    _prevState = _currentState;
    // ステートを切り替える
    _currentState.OnEnd();
    _currentState = nextState;
    _currentState.OnStart();
}
/// 前回のステートに切り替える
public void ChangePrevState()
{
    if (_prevState == null)
    {
        return;
    }
    // 前のステートと現在のステートを入れ替える
    (_prevState, _currentState) = (_currentState, _prevState);
}
}
```

各行動を独立したクラスにカプセル化することで、状態遷移のロジックを整理しました。

PigShotArena

作品概要

跳ねて転がる豚からはじまりました。
互いにステージ上の豚をはじいて、転がして跳ねる。そういったカオスを楽しむ対戦ゲームです。



作品URL
https://github.com/Tel1urium/teru_/tree/main/PigShotArena

物理挙動の追加

ゲームの特性上たくさん跳ねなければならないためunity既存の物理エンジンでは理想の挙動を作ることができなかったため追加しました。
トンネリングや逆に跳ねなくなったりしたので実装が大変でした。

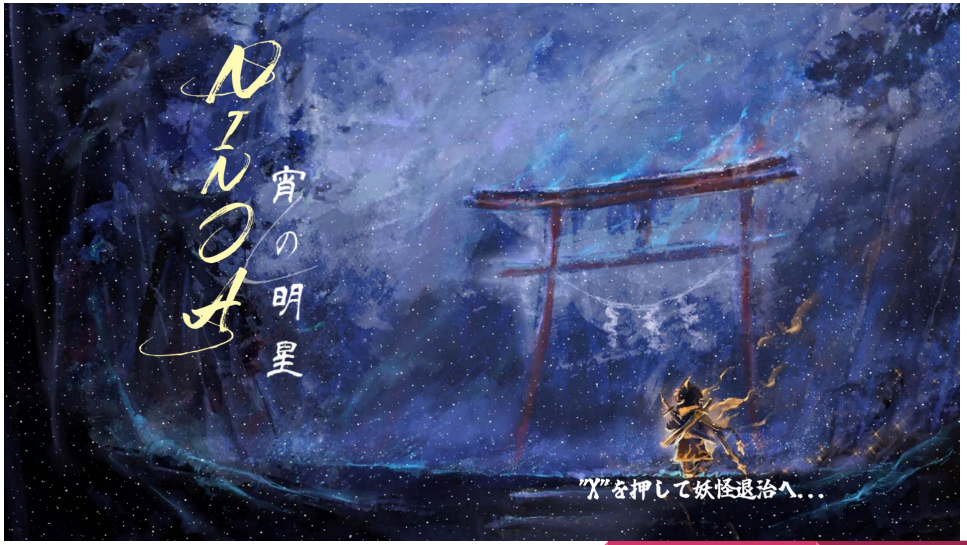
```
void CollisionPredictionAndReflect()
{
    Vector3 velocity = rb.linearVelocity;
    float speed = velocity.magnitude;

    if (speed < 0.01f) return;

    Vector3 direction = velocity.normalized;
    Ray ray = new Ray(transform.position, Vector3.zero);
    var sphereRadius = 0.7f;
    RaycastHit hit;
    var rayLength = 0.00000f;
    if (Physics.SphereCast(ray, sphereRadius, out hit, rayLength, collisionMask))
    {
        Vector3 hitNormal = hit.normal;
        Vector3 reflected = Vector3.Reflect(velocity, hitNormal);

        rb.linearVelocity = Vector3.zero; // 一度停止
        rb.AddForce(reflected.normalized * 50, ForceMode.VelocityChange);

        Debug.DrawRay(transform.position, direction * hit.distance, Color.red, 0.2f);
        Debug.DrawRay(hit.point, hitNormal, Color.yellow, 0.2f);

    }
    else
    {
        Debug.DrawRay(transform.position, direction * rayLength, Color.green, 0.1f);
    }
}
```

レイキャストで判定することでトンネリングを防止

NINJA~宵の明星~

作品概要

回避を主軸にして戦うアクションゲームです。
ジャスト回避を成功させると次の攻撃が強化され、ボスに大ダメージを与えることができます。



作品URL
https://github.com/Tel1urium/teru_/tree/main/NINJA

敵の行動パターン

敵の行動パターンをプレイヤーとの距離、角度によって変化しました。
プレイヤーと敵の座標を取得し、Vector3.Distance()関数を用いて距離を計算し、
条件分岐式で行動パターンを変えました。
敵キャラクターのコンセプトに合うように設計するのが難しかったです。

```
EnemySensor
private void OnTriggerStay(Collider other)
{
    if(inField.GetInField()) {
        var playerDir = other.transform.position - transform.position;
        var angle = Vector3.Angle(transform.forward, playerDir);//プレイヤーと自身との角度
        var dis = Vector3.Distance(other.transform.position, transform.position);//距離
        if (other.gameObject.tag == "Player")
        {
            ctrl.transform.position = Vector3.Lerp(ctrl.transform.position,
                other.gameObject.transform.position, 0.1f);
            ec.SetTarget(other.transform);
            ec.SetLookPlayer(other.transform.position);
            if (EnemyController.now / maxTime == 1)
            {
                maxTime = Random.Range(1, 4) * 80;

                if (dis <= search.radius * 0.15f)//半径の0.1～0. 15
                {
                    if (Probability(5))
                    {
                        ec.SetState(EnemyController.EnemyState.Chase);
                    }
                    else if (Probability(15))
                    {
                        ec.SetState(EnemyController.EnemyState.Atk3);
                    }
                    else if (Probability(30))
                    {
                        ec.SetState(EnemyController.EnemyState.Atk2);
                    }
                    else if (Probability(40))
                    {
                        ec.SetState(EnemyController.EnemyState.Atk1);
                    }
                    else if (Probability(75))
                    {
                        ec.SetState(EnemyController.EnemyState.Atk4);
                    }
                    else
                    {
                        ec.SetState(EnemyController.EnemyState.Leave);
                    }
                }
                else if (dis <= search.radius * 0.7f && dis > search.radius * 0.15f)//それ以降
                {
                    if (Probability(10))
                    {
                        ec.SetState(EnemyController.EnemyState.Chase);
                    }
                    else if (Probability(30))
                    {
                        ec.SetState(EnemyController.EnemyState.Atk3);
                    }
                    else if (Probability(60))
                    {
                        ec.SetState(EnemyController.EnemyState.Atk2);
                    }
                    else if (Probability(80))
                    {
                        ec.SetState(EnemyController.EnemyState.Atk1);
                    }
                    else
                    {
                        ec.SetState(EnemyController.EnemyState.Leave);
                    }
                }
                else if (dis <= search.radius * 1f && dis > search.radius * 0.7f)//それ以降
                {
                    ec.SetState(EnemyController.EnemyState.Chase);
                }
            }

        }
    }
}
```

制作当時はVector3.Distanceを使用していましたが、後に平方根計算の負荷を知り、現在ではsqrMagnitudeを使ってパフォーマンスを最適化する知識を習得しました。次の作品ではこの知見を活かします。

以上となります
ご覧いただきありがとうございます。

日本電子専門学校 ゲーム制作科 2年
プログラマー志望
照岡隆誠