作品名：DoragonHunter

作品GitURL：https://x.gd/v3Q1B

紹介動画URL：https://x.gd/SCnzI

制作人数：3人

担当箇所：プログラム全て

制作環境：C#/Unity

敵の様々な攻撃を回避しつつ
コンボ攻撃を叩き込め！

ジャンル:モンハン風**狩猟**アクション！

大食いプログラマー浜松廉斗

学校：福岡情報ITクリエイター専門学校

ポートフォリオ：https://x.gd/at8C6

GitHubURL：https://github.com/Hmmtrnt

言語経験年月
Unity/C#：2年、C++：2年等

ゲームループ

より高い難易度に挑戦！タイムを縮めよ！

クエスト受注

いざ出発！

モンスター討伐！

討伐成功！

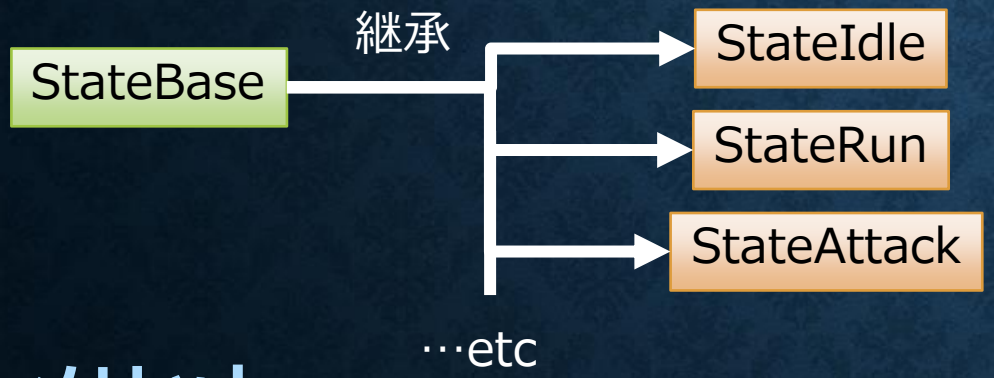
ランク更新！

未知の難易度に挑戦！

制作作品　プレイヤーのプログラムコード1

ステートマシンの導入

膨大なプレイヤーの状態を管理する際の
可読性向上や工数の短縮のために
ステートマシンを導入した。
右の図のクラスを継承し、状態クラスを
増やしていく仕組みになっている。



メリット

- 可読性の向上
- 状態クラスを追加する際の工数の短縮

デメリット

- 汎用的ではない

状態を管理するクラスの実装速度を
上げるためのクラス

```
1   // 状態管理の抽象クラス
2
    99+ 個の参照
3   public abstract class StateBase
4   {
5       /*プレイヤー*/
6       /// <summary>
7       /// ステート開始時呼び出し
8       /// </summary>
9       /// <param name="owner">アクセスするための参照</param>
10      /// <param name="prevState">ひとつ前の状態</param>
        24 個の参照
11      public virtual void OnEnter(Player owner, StateBase prevState) { }
12      /// <summary>
13      /// Update
14      /// </summary>
15      /// <param name="owner">アクセスするための参照</param>
        23 個の参照
16      public virtual void OnUpdate(Player owner) { }
17      /// <summary>
18      /// FixedUpdate
19      /// </summary>
20      /// <param name="owner">アクセスするための参照</param>
        23 個の参照
21      public virtual void OnFixedUpdate(Player owner) { }
22      /// <summary>
23      /// ステート終了時呼び出し
24      /// </summary>
25      /// <param name="owner">アクセスするための参照</param>
26      /// <param name="nextState">次に遷移する状態</param>
        22 個の参照
27      public virtual void OnExit(Player owner, StateBase nextState) { }
28      /// <summary>
29      /// ステート遷移の呼び出し
30      /// </summary>
31      /// <param name="owner">アクセスするための参照</param>
        22 個の参照
32      public virtual void OnChangeState(Player owner) { }
33
```

制作作品　プレイヤーのプログラムコード2

各状態クラスを内部クラスとして宣言

内部クラスにした**メリット**

・privateフィールドの関数、変数を呼び出すことができる
・状態制御関係のバグの発生率の低下

内部クラスによる**コードの肥大化**

膨大な数の状態クラスを内部クラスとして宣言したため**コードの肥大化**がおこった。
コードの肥大化が原因で可読性が大幅に低下することになったので**partial class**を使用することにした。
その結果プレイヤークラスを内部クラスの数だけ分けてコードを書くことができた。

```
/*プレイヤー行動全体の管理*/

using UnityEngine;

Unity スクリプト (3 件のアセット参照)|99+ 個の参照
public partial class PlayerState : MonoBehaviour
{
    //--納刀状態--//
    private static readonly StateIdle               _idle = new();              // アイドル.
    private static readonly StateAvoid              _avoid = new();             // 回避.
    private static readonly StateRunning            _running = new();           // 走る.
    private static readonly StateDash               _dash = new();              // ダッシュ.
    private static readonly StateFatigueDash        _fatigueDash = new();       // 疲労時のダッシュ.
    private static readonly StateRecovery           _recovery = new();          // 回復.
```

```
/*待機状態*/

using UnityEngine;

Unity スクリプト|99+ 個の参照
public partial class Player
{
    2 個の参照
    public class StateIdle : StateBase
    {
        3 個の参照
        public override void OnEnter(Player owner, StateBase prevState)
        {
            // アニメーション開始.
            owner._idleMotion = true;

            if(prevState == _sheathingSword)
            {
                owner._motionFrame = 0;
            }
        }
```

制作作品　戦闘面での工夫

モーションキャンセルの実装

このゲームはプレイヤーの攻撃の後隙は全体的に長くなっている。
なので、攻撃の後隙に一部の入力を行うことで (※1)**モーションキャンセル**が
できるような実装を行った。
このモーションキャンセルにより攻撃の後隙の時間にコンボ攻撃を続けるか回避を
行うかを判断する時間が生まれたため、強い攻撃を畳み込むか避けるかの駆け引きを
行えるようになった。
この駆け引きは戦闘面において単純なターン性の戦闘から複雑なターン性の戦闘になった。

攻撃開始

後隙

後隙をつぶして次の行動へ

※1**モーションキャンセル**：現在のモーションを中断し次のモーションに移ること