

反射神経刀アクションゲーム

虫斬り

むしぎり



作品ポートフォリオ

チームメンバー

城脇 礼奈

プロジェクトリーダー
デザインリーダー / プランナー

プロジェクトマネージャー

新倉 滉祐 吉田 智哉

プランナー

有川 海斗 キムヒョンミン

UI デザイナー

村上 慧都 浅見 郁弥

内野 稀翠

エフェクトデザイナー

松田 拓道 守屋 佳祐

等々力 遊美

モーションデザイナー

杉森 風太

プログラムリーダー

横田 優作 菅原 渉

Switch 担当者

盛田 翔太

プログラマー

鳥海 穂菜実

大長 ルミナ

森 美夏

関口 栄哉

TEAM MUSHIGIRI



蟲斬

直感操作で爽快コンボ、
反射神経 刀アクション！

蟲斬は、TECH.C ゲームプロジェクト 2023 で企画・制作した和風アクションゲームです。

使用するのは Switch のジョイコン 1 つのみ、次から次へと表示される 3 種類の操作指示に上手く対応し敵を攻撃していきます。ミスせずコンボを重ねるたび、難易度が上がり大ダメージを与えることができますが、同時に受けるダメージも大きくなる「コンボシステム」によって、勝負の緊張と爽快感が楽しめます。

浮世絵を参考にした和モダンなデザインと、「蟲」たちによる一風変わった世界を気に入って頂けたら幸いです。

蟲斬



◆ ジャンル
◆ プラットフォーム
◆ 操作方法
◆ プレイ人数
◆ グラフィック
◆ 開発環境

アクション
Nintendo Switch
Joy-Con
1人
2D(Spine)
Unity

ジョイコンを構えて

斬る！

突く！

守る！



矢印の向きに合わせて
ジョイコンを振る



マークが出たら
前へ素早く押し出す



敵が攻撃してきたら
横向きにしてガードする

直感操作で爽快な和風・刀アクション！

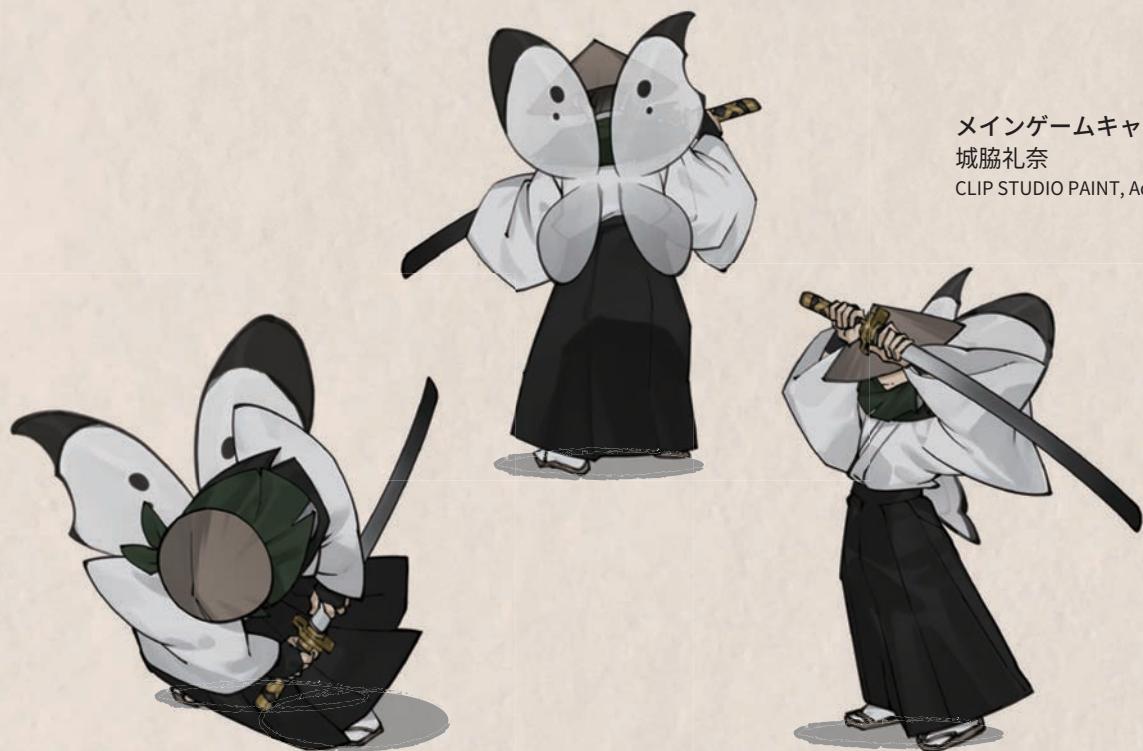
キャラクター



紋白 MONSHIRO

神の森を一人旅する蝶族の剣客。
悪しき蟲を討つ「蟲斬」と名乗る。
寡黙で自らの出自を語ることは殆どないが
穏やかで意外とノリもいい
大根の葉っぱとなんばばの蜜がすき。

メインゲームキャラクターモデル
城脇礼奈
CLIP STUDIO PAINT, Adobe Photoshop





土蜘蛛 TSUCHIGUMO

神の森の守りびとである蝶を食い、森を滅ぼす悪しき存在「蟲」の一種。
本ゲームのエネミーキャラクター。



紋白シンボルマーク

ゲーム中に使用されるシンボルマークです。
実際に存在する蝶の家紋を再現しながら、
紋白のモデルであるモンシロチョウを
かたどりました。 / 城脇

プランナー



プランナーリーダー

城脇 礼奈

プランナー

有川 海斗 キムヒヨンミン

プロジェクトマネージャー

新倉 涼祐 吉田 智哉

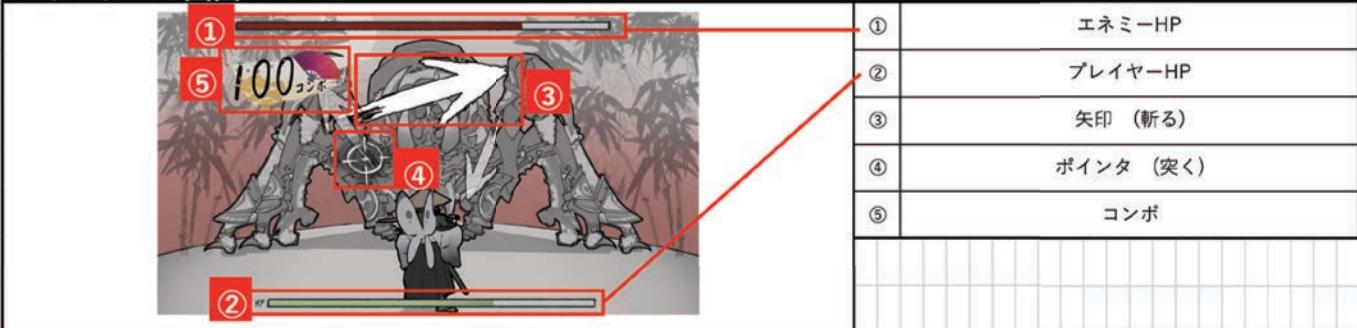
INDEX	スケジュール	企画概要	操作方法	ゲームフロー	emainゲーム概要	攻撃と防御	不意打ち	コンボ	難易度	勝利演出	背景演出	画面構成要素	UI	キャラクタ
-------	--------	------	------	--------	------------	-------	------	-----	-----	------	------	--------	----	-------

メインゲーム概要

ゲームルール

勝利条件	敵のHPを0にする。ステージボスを倒すとゲームクリア
敗北条件	プレイヤーのHPが0になる
攻撃・ダメージ条件	指示された動作を行うと攻撃を与えることができ、ミスした場合ダメージを受けHPが減少します。

メインゲーム画面



HP

プレイヤーのHPは難易度に関わらず一定に設定してください。エネミーは雑魚敵とステージボスでHP量が違います。

システム一覧

メインゲーム部分に実装される、重要なシステム・要素についての一覧です。詳細は各項目のリンク先、Excelシートをご覗ください。

システム名	記載シート先
メインゲーム概要	攻撃と防御 不意打ち コンボ 難易度 勝利演出 背景演出 画面構成要素 UI キャラクター BGM・SE チュートリアル

仕様書 / 城脇礼奈 新倉涼祐

ゲームプロジェクト『蟲斬』の仕様書です。
コンボシステムやジョイコンを使った操作など、
メインゲーム部分の方向性を定め、試遊を重ねて
詳細部分を練り直していきました。
演出やエフェクト等作品の見栄えや仕上がりも意識し、
“ゲーム体験をより豊かにできる作り込みの部分”の
可視化を心がけました。 / 城脇

城脇礼奈
PORTFOLIO



コンボ数演出とダメージ倍率

常時、ゆっくりと木の葉のエフェクトが画面に舞っています。コンボ数によって、花吹雪に変化します。※詳細はシート『背景演出』を参照

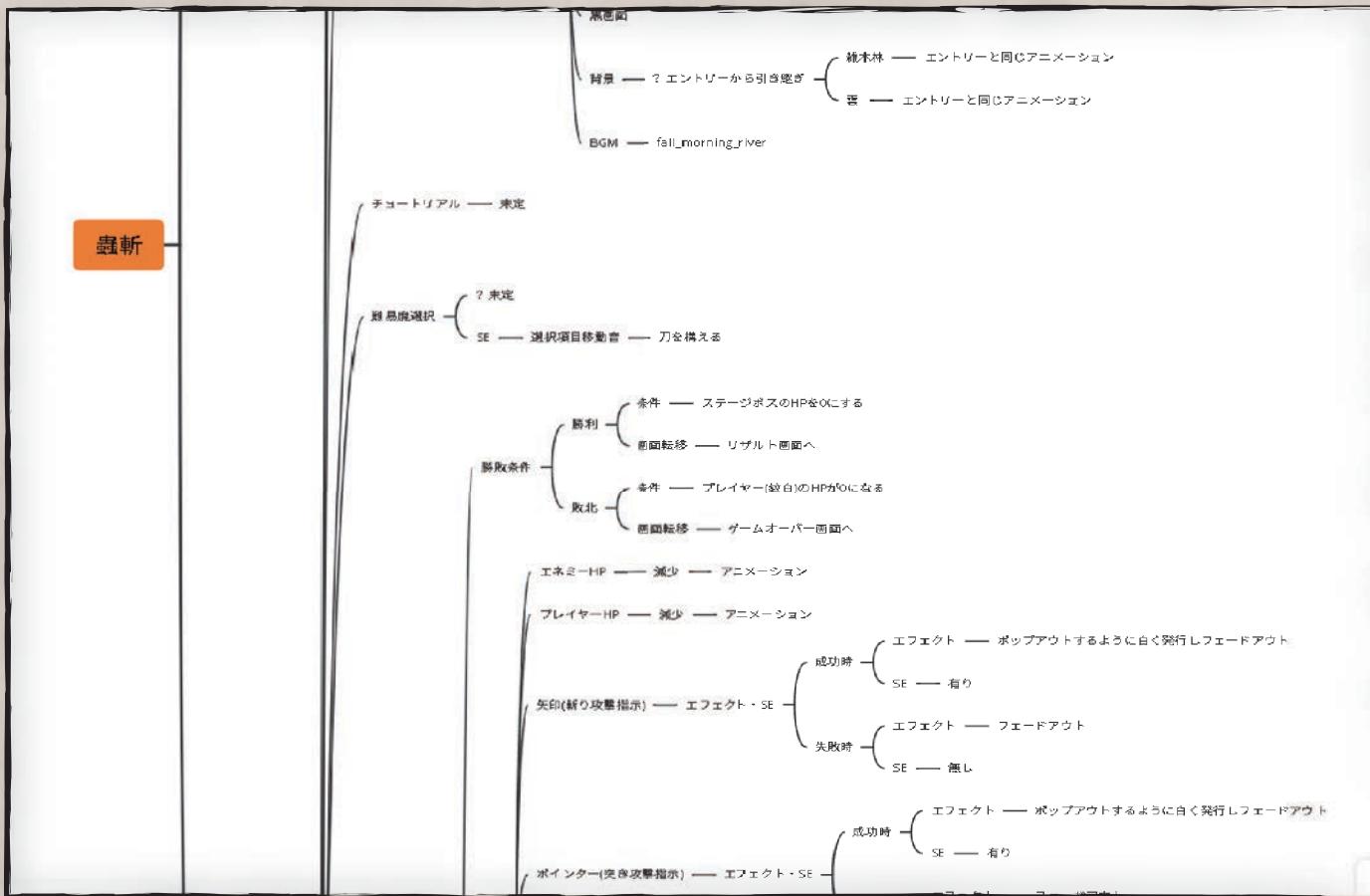


背景色の変化

コンボ数によって、背景グラデーション部分の色が変化します。※詳細はシート『背景演出』を参照



タスク管理書類 / 新倉滉祐

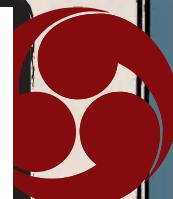


Gitmind

初期の仕様書の中から要素を全て出し、α版やβ版までの目標の設定タスクの割り振りを行いました。割り出しを行う中で不透明な仕様の発見やタスク総量が見えるようになり、チームでの作業分配を円滑に行うことができました。 / 新倉

タスクマトリクス								
要素	内容	EFかアニメーション	秒数	発注書が必要か否か	優先順位	発注担当	制作担当	完了
背景	背景コンボ演出(笛)	PCが動かす		不要	1			
	背景コンボ演出(紅葉)	PCが動かす		不要	1			
	背景コンボ演出(桜)	PCが動かす		不要	1			
UI	UI実装済出	EF	0.5秒	必要	6			余裕があれば制作
	攻撃変更時の点灯	PCが動かす		不要	2			
	斬り攻撃タイミングの線	EF	1秒	必要	1	有川		完了
	斬り攻撃成功時	EF	0.5秒	必要	1	有川		完了
	突き攻撃タイミングの線	EF	1秒	必要	1	有川		完了
演出	突き攻撃成功時	EF	0.5秒	必要	1	有川		完了
	難易度アッパー(紅葉)	EF	1秒	必要	3			画面上に表示される演出「あかよみ」
	難易度アッパー(桜)	EF	1秒	必要	3			画面上に表示される演出「みはしの」
攻撃	斬る失敗	PCが動かす		不要	3			暗くなってフェードインするようなSE
	突く失敗	PCが動かす		不要	3			
キャラクター	斬る1	EF		必要		キム		完了
	突く	EF		必要		キム		完了
IDリスト	斬る2(カウンター)	EF		必要	2			
	どめの一撃	EF		必要	4			

新倉滉祐
PORTFOLIO



ID リスト

発注書のタスク管理を行う時にプランナーが確認できるように ID リストを作成しました。

情報共有に役立ちタスクの効率化に成功しました。 / 新倉

ID 管理表 / 吉田智哉

7309	木の葉桜3	エフェクト用グラフィック
7310	斬る矢印	エフェクト
7311	斬る矢印(成功時)	エフェクト
7312	プレイヤー 突く	モーションエフェクト
7313	プレイヤー 斬る	モーションエフェクト
7314	プレイヤー 斬る2(カウンター)	モーションエフェクト
7315	プレイヤー 防御	モーションエフェクト
7316	プレイヤー ダメージ	モーションエフェクト
7317	ボスエネミー 登場	モーションエフェクト
7401	雑木林1	グラフィック
7402	雑木林2	グラフィック
7403	雑木林3	グラフィック
8501	ランク表示(背景)	アニメーション
8502	ランク表示(文字)	アニメーション
8503	キャラクター	アニメーション
8504	木の葉	アニメーション
8505	プレイヤー 走る	アニメーション
8506	プレイヤー 止まる	アニメーション
8507	プレイヤー 防御	アニメーション
8508	プレイヤー 斬る1	アニメーション
8509	プレイヤー ダメージ	アニメーション
8510	プレイヤー リザルト立ち絵	アニメーション

ID 管理表

必要なグラフィック素材
アセット等をまとめて
わかりやすくしました
/吉田

※ ? は機能を使用するかどうか不明		大項目	使用場面
ID	項目名		
_0001	タイトルロゴ	背景	タイトル画面
_0801	タイトルBGM	BGM/SE	タイトル画面
_0901	決定SE	BGM/SE	タイトル画面
_1101	?Yesボタン	UI	チュートリアル選択画面
_1102	?Noボタン	UI	チュートリアル選択画面
_1801	?チュートリアル選択画面BGM	BGM/SE	チュートリアル選択画面
_1901	選択時SE	BGM/SE	チュートリアル選択画面
_2801	?チュートリアルBGM	BGM/SE	チュートリアル画面
_2901	?ボタン押した時のSE	BGM/SE	チュートリアル画面
_2902			
_3101	難易度表記	UI	難易度選択画面
_3801	?難易度選択画面BGM	BGM/SE	難易度選択画面
_3901	難易度選択決定音	BGM/SE	難易度選択画面
_3902	難易度項目移動音	BGM/SE	難易度選択画面
_4201	プレイヤーHPバー	UI	メインゲーム画面
_4202	エネミーHPバー	UI	メインゲーム画面

発注書 / 有川海斗 キムヒョンミン

フロー

- 矢印が現れ始める（半透明）
- 矢印が完全に現れる。このタイミングから枠が迫り始める
- 枠がどんどん狭くなる（1秒程）
- 枠が完全に重なった時、金色になる

矢印が出現するまでの推移図です

矢印が出現するまでの推移図です

1. 半透明な矢印から完全な矢印になります
2. 完全な矢印になります

0.5秒後

（※1）斬撃成功エフェクト

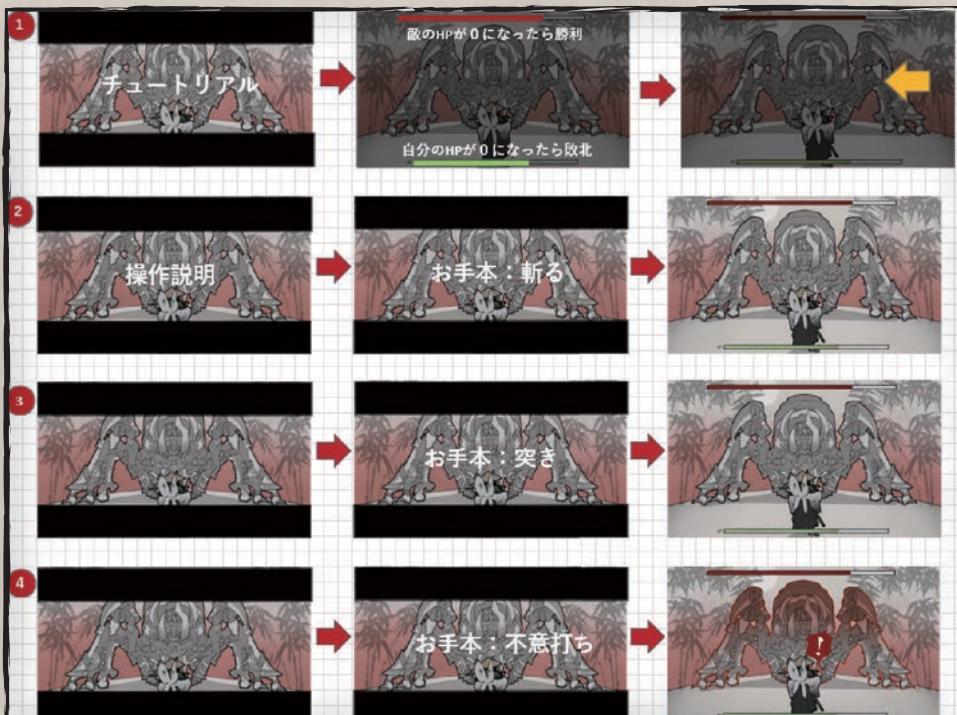
（※1）斬撃成功エフェクト

1. 矢印の通りに攻撃が成功する

概要
この一連の流れは0.5秒程にする

```

graph TD
    A[斬った時] --> B[成功した場合]
    A --> C[失敗した場合]
    B --> D["斬撃成功エフェクトと共に矢印が消滅する"]
    C --> E["エフェクトが必ずしも消滅する"]
  
```



エフェクト発注書 チュートリアル仕様書

メインゲーム中に表示される
斬り・突きエフェクトの発注を
中心に行い、チュートリアルの
フラグリスト制作も行いました。
/ 有川

エフェクト発注書

メインゲーム中に表示される
キャラクターのモーション
エフェクト発注を行いました。
具体的なイメージを添えて
わかりやすくしました。
/キム

0.00

黒い傷がぱっと現れる

0.05

中が白で満まると同時に光る。閃光のようなエフェクトも入る。

0.10

閃光は消える。白の光も消してどんどん小さくなる

0.15

白が消えてもっと小さくなる

0.20

完全に消える

0.30

白が消えてもっと小さくなる

0.35

エフェクトが消えて半分だけキラって光る

0.40

完全に消える

禁止事項

片方だけ伸ばしたり減らしたりはやめてください
中心から左右両方を同時に動いてください

デザイナー



デザイナーリーダー

UI デザイナー

エフェクトデザイナー

モーションデザイナー

城脇 礼奈

村上 慧都 浅見 郁弥 内野 稀翠

等々力 遊美 松田 拓道 守屋 佳祐

杉森 風太

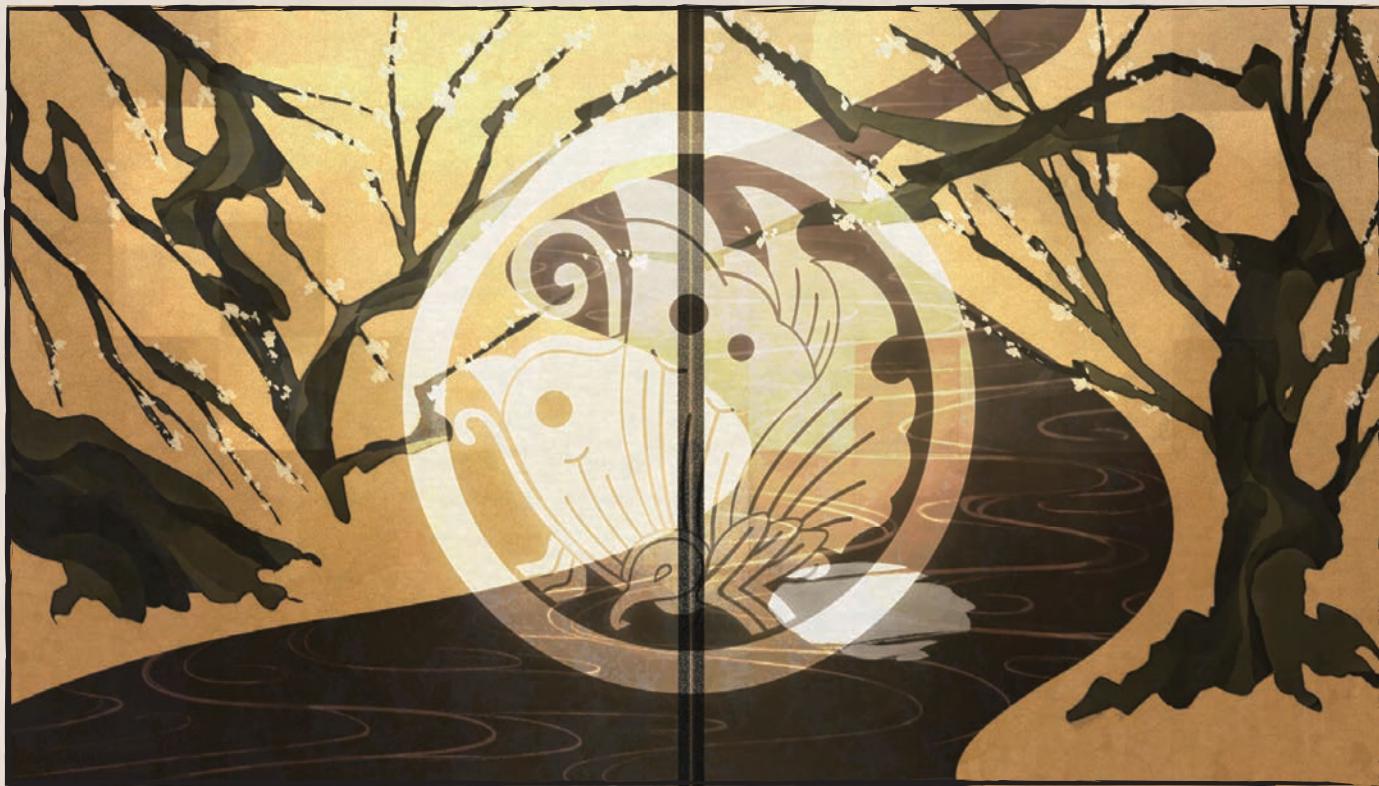


ゲームビジュアル / 城脇礼奈

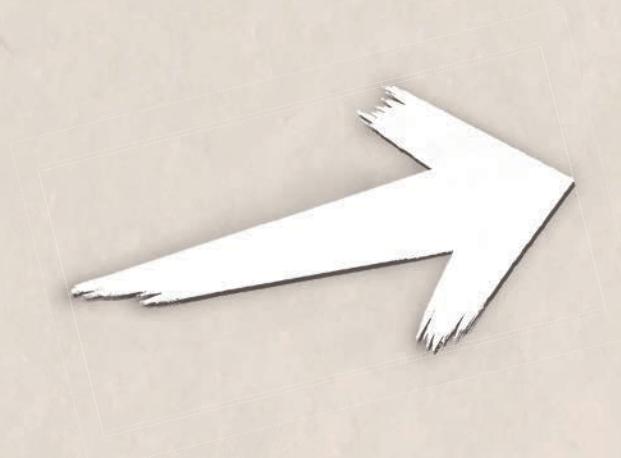
ゲーム全体を通してのビジュアルイメージを制作しました。
全体のデザインや解釈にばらつきが出ないよう、フォントや
UI のカラーコード統一など、作品のイメージを仕様書に
細かく記載し、各担当デザイナーとこまめなすり合わせを
行ってきました。

既存の和風ゲームにない違いを出すため、浮世絵をイメージした
抑えめの色調や家紋の幾何学デザインを参考にし『和モダン』な
印象が前面に出るように心がけました。 / 城脇





波紋 エフェクト素材 / 村上慧都



ポインター & スラッシュ UI

筆の筆跡を意識しながら制作しました。遠くからでも視認性を高める為、背面に灰色を敷いています。 /村上



波紋 パーティクル素材 / 浅見郁弥

ゲーム全体で仕様する花吹雪の元素材を制作しました。

ゲーム上のパーティクルアニメーションでひらひらと舞います。 /浅見

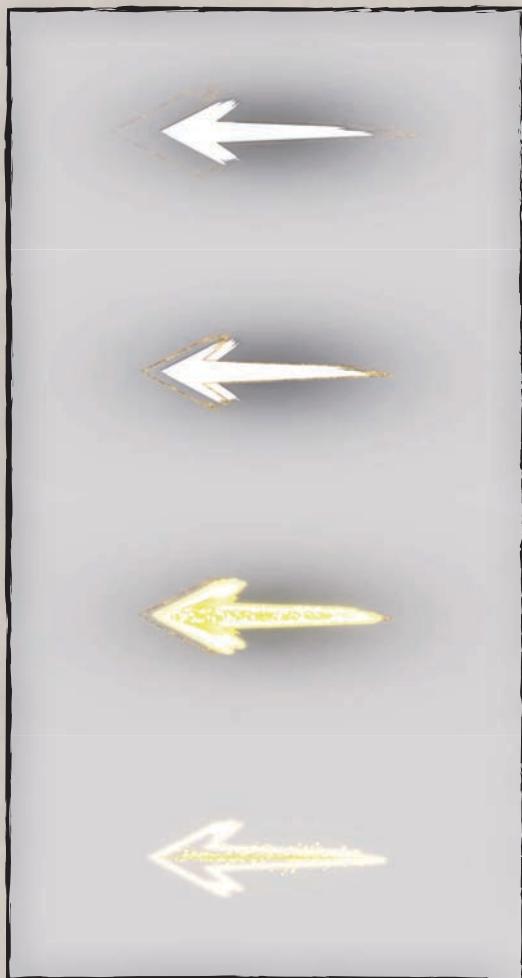




操作アイコン / 内野稀翠

チュートリアル、操作説明で使用されるアイコンです。
見ただけで何のモーションなのかわかりやすいように
制作しました。 / 浅見





bli斬りエフェクト / 等々力遊美

斬り成功時エフェクト

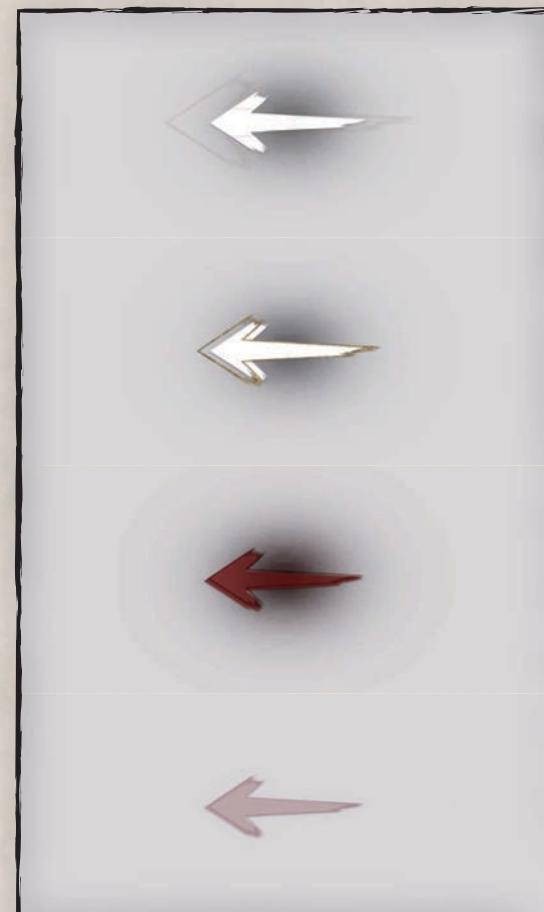
「斬り」攻撃時にプレイヤーが入力した攻撃動作が画面に指示された動作と一致した場合に、攻撃が成功であることを示すエフェクトです。

プレイヤーが「攻撃が成功した」という達成感をより大きく感じられるよう、豪華な仕上がりになるよう制作しました。 / 等々力



斬り失敗時エフェクト

「斬り」攻撃時にプレイヤーが入力した攻撃動作が画面に指示された動作と違った場合に、攻撃が失敗であることを示すエフェクトです。矢印の振幅が大きくなりすぎないように気をつけながら制作しました。 / 等々力



突きエフェクト / 等々力遊美



突き成功時エフェクト

「突き」攻撃時にプレイヤーに攻撃動作を指示するエフェクトです。

目立ちすぎてゲームの邪魔になることがないように制作しました。 /等々力

こちらから再生できます▶



ダメージエフェクト / 等々力遊美

ダメージを受けた際に、「ダメージを受けた」とプレイヤーに分かりやすく伝えるためのエフェクトです。見えづらさも目立ちすぎもしないように制作しました。 /等々力

こちらから再生できます▶



遷移アニメーション / 松田拓道

リザルト画面へ遷移する際に再生される花吹雪の画面遷移アニメーションを担当しました。
自然な花びらの動きを表現できるように意識して作りました。/ 松田

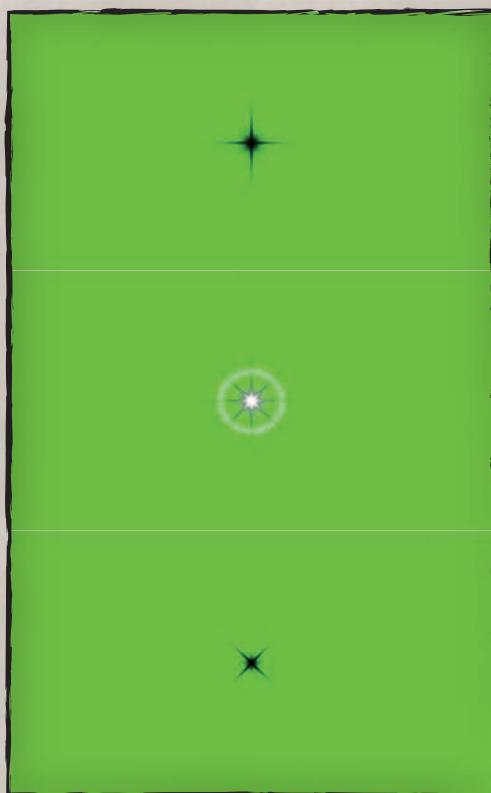
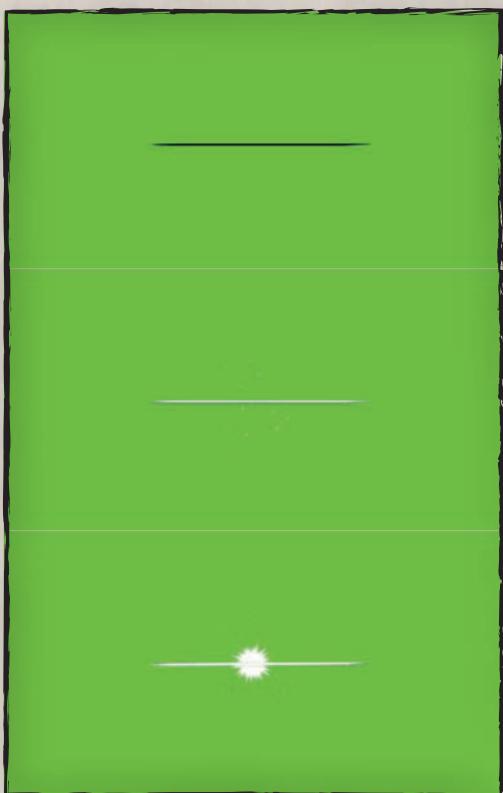
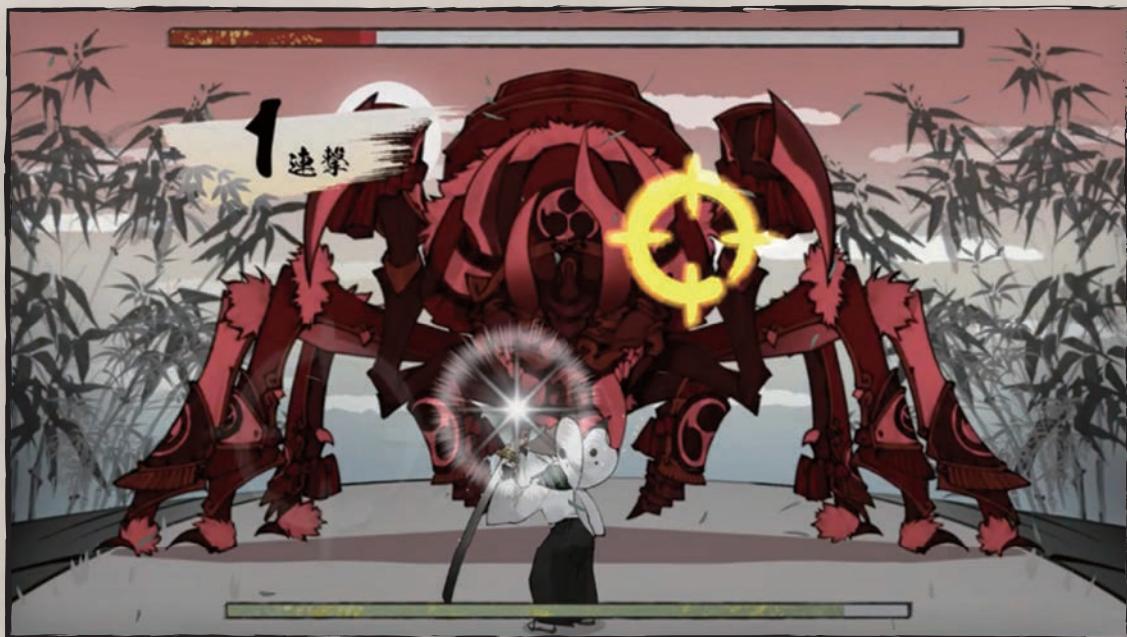


こちらから再生できます▶





キャラクターモーションエフェクト / 松田拓道



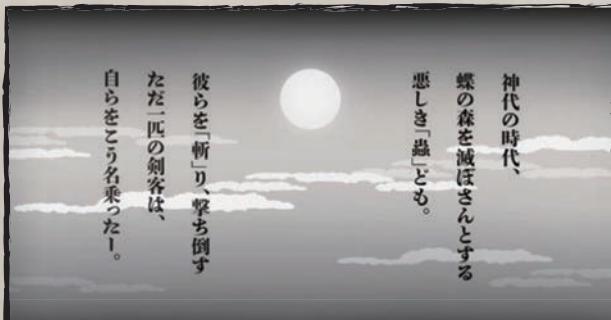
こちらから再生できます▶



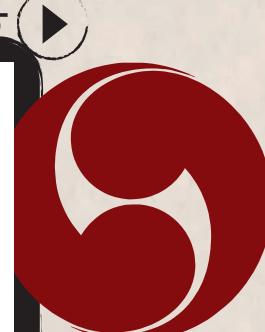
こちらから再生できます▶



 作品紹介 PV / 守屋佳祐



絵コンテ / キムヒョンミン 城脇礼奈



こちらから視聴できます ▶

プレイヤーキャラクターモーション / 杉森風太



キャラクターモーション『紋白』

①待機モーション

あまり派手な動きは無いものの、羽を動かしたり笠を揺らしたりすることで飽きさせないモーションにしました。



②斬り・突きモーション

斬りと突き 2 種類のモーションです。
再生後は自然に移行できるよう、最後は必ず待機状態へ戻るように制作しています。



③カウンターモーション

イベント機能を使い、防御の立ち絵からカウンターに変化するときに効果音がなるようにしました。



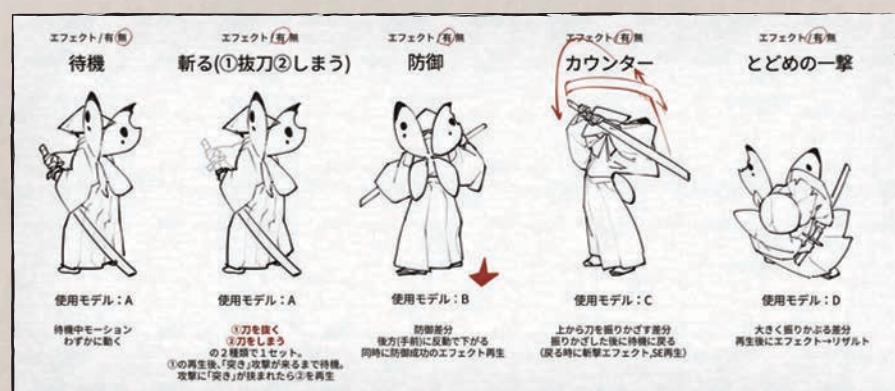
④とどめモーション

ダイナミックな動きから緩急のある動きにすることで迫力を出しました。手は IK を使いくつづけることでどの位置にあっても連動するようにしています。

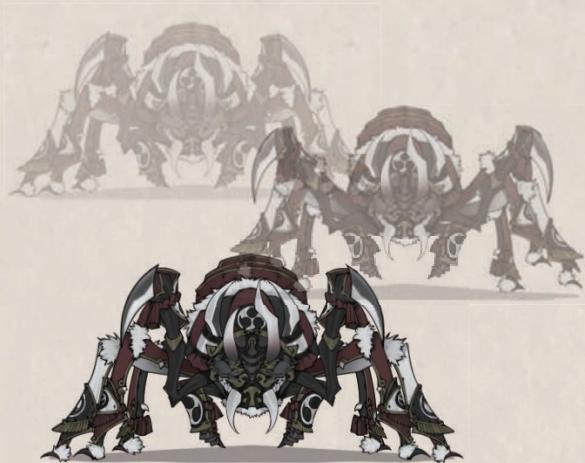
こちらから視聴できます



モーション発注書 / 城脇礼奈



エネミーキャラクターモーション / 杉森風太



②攻撃モーション

攻撃時のモーションです。鎌だけでなく全身を使って攻撃の意思が伝わるように表現しました。着地の足のバラバラ感は不気味さを強調させるためわざとずらしています。



キャラクターモーション『土蜘蛛』

①待機モーション

蜘蛛特有の気持ち悪い動きに緩急をつけて強調したアニメーションにしました。足はIKを使って動くようにしたため、体を上下させても違和感の少ない物になっています。



③ダメージモーション

ダメージを喰らったときののけ反りを感じさせるため、スケールを下げて奥行きを感じさせるようなアニメーションを心がけました。のけ反りから戻ってくるときのアニメーションは生き物らしさを感じられるように、実際の蜘蛛を見て特にこだわって作っています。

こちらから視聴できます▶



キャラクターカットイン / 杉森風太

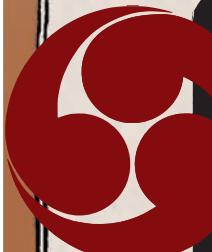


杉森風太
PORTEFOLIO



勝利時キャラカットイン

クリッピングマスク機能を使って目の瞬きや
腕の前後関係を作りました。
移動はモーションブラー等を入れるために
AfterEffects を使用しています。 / 杉森



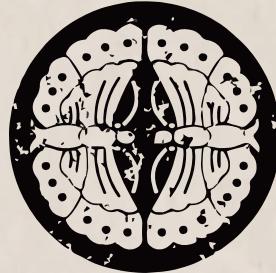


使用ツール

CLIP STUDIO PAINT , Adobe Photoshop

制作時間 /5 時間

プログラマー



プログラマーリーダー

横田 優作 菅原 渉

Switch 操作系担当

盛田 翔太

プログラマー

鳥海 穂菜実 森 美夏

大長 ルミナ 関口 栄哉

```
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
```

/// <summary>
/// 引数のギミックをリストに追加＆判定までの時間が早い順でソート
/// </summary>
/// <param name="item"></param>
3 個の参照
public void AddGimmickList(MainGimmickAbstract item)
{
 // 空の要素を追加
 generatedMainGimmickList.Add(new GimmickData());

 // 配列の後に参照用変数
 int listInnermostNum = generatedMainGimmickList.Count - 1;

 // アブストラクトを追加
 generatedMainGimmickList[listInnermostNum].gimmickAbstract = item;

 // 判定に必要な数値の取り出しだけ
 generatedMainGimmickList[listInnermostNum].gimmickMainValues
 = generatedMainGimmickList[listInnermostNum].gimmickAbstract.Get_JudgeNeedValue();

 // 保存
 generatedMainGimmickList[listInnermostNum].judgeTime
 = generatedMainGimmickList[listInnermostNum].gimmickMainValues.JudgeDraePeriod + elapsedTime;

 // ギミックソート
 SortGimmickList();
}

メインゲームシーン

ギミックの 生成後リスト追加とソート / 菅原渉

使用ツール

- Unity 2021.3.19f1
- Visual Studio 2022

制作時間 /1 時間

ギミックの生成後の管理をするためのリストに
ギミックの追加をしています。
その際判定までの時間が一律ではないので
判定が前後してしまわないように、
判定までの時間が少ない順でソートを
しています。 /菅原



The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** Debug, Any CPU, Unityにアタッチ, NXターゲット: デフォルト
- Solution Explorer:** Assembly-CSharp
- Code Editor:** GimmickGenerator.cs (highlighted)
- Code Content:** The code implements a selection system for gimmicks based on probabilities. It includes comments explaining the logic: selecting a gimmick from a pool, calculating total probability, and using cumulative probability to find the selected item.
- Status Bar:** 134%, 問題は見つかりませんでした

生成ギミックの選定システム / 菅原渉

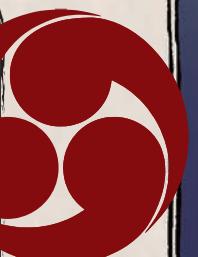
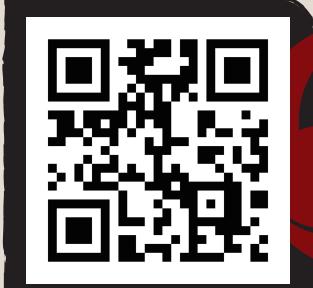
使用ツール

- Unity 2021.3.19f1
- Visual Studio 2022

制作時間 /1 時間

事前に設定した確率を元に、エネミーの攻撃とメインギミックを選び、その後早い・普通・遅い等のタイミングで分かれていれば、それも事前に設定した確率を元に選定し、最終的にギミックのオブジェクトプールを戻り値として渡しています。 /菅原

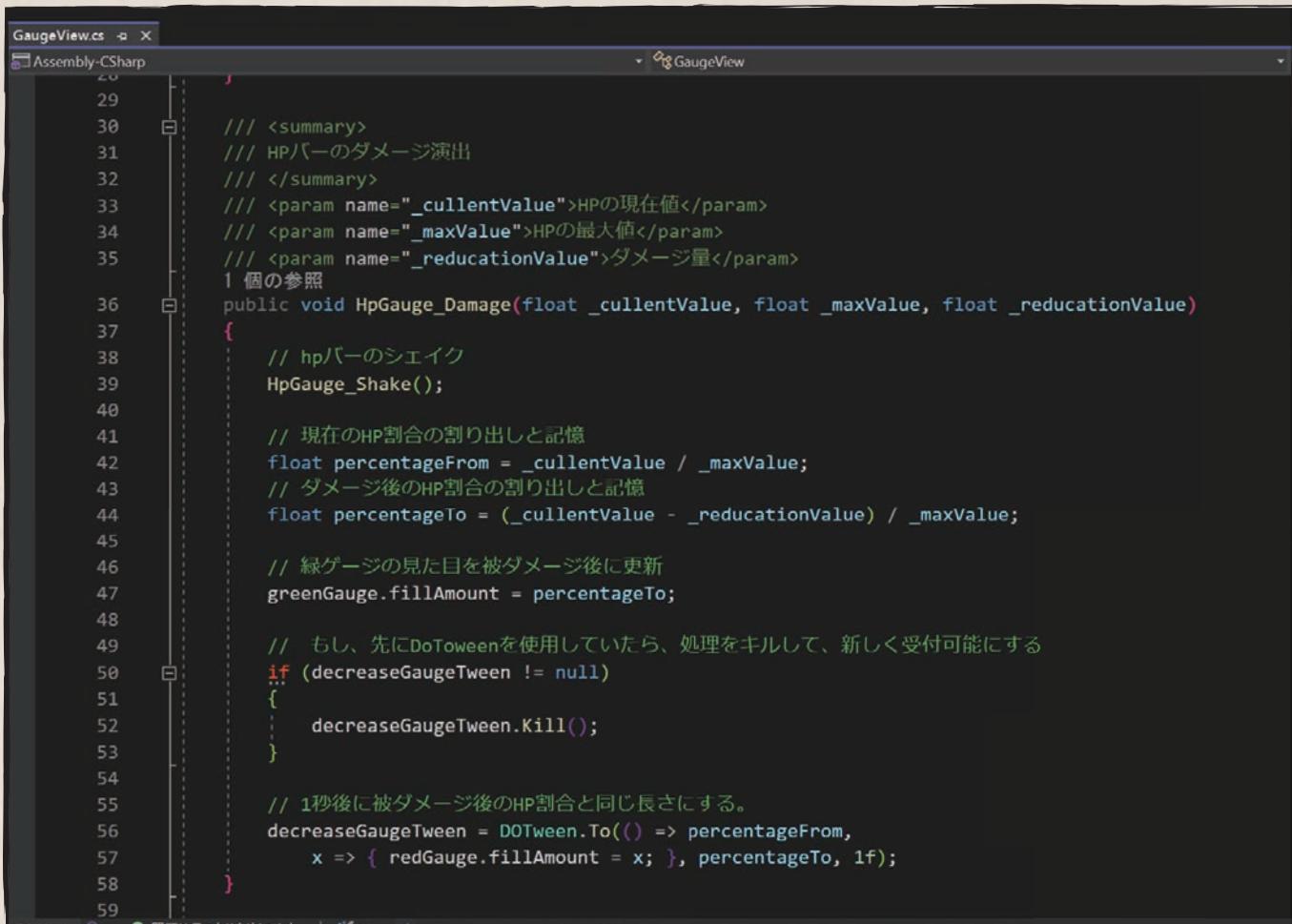
菅原渉
PORTFOLIO



HP バーのダメージ演出 / 横田優作

HP の現在値、最大値、受けたダメージから HP 割合を算出し HP バーを減らしています。

緑のバーを減らした後に赤いバーを 1 秒かけて徐々に減らすことで、ダメージを受けたことを分かりやすくしています。 / 横田



```
GaugeView.cs
Assembly-CSharp
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

```
/// <summary>
/// HPバーのダメージ演出
/// </summary>
/// <param name="cullentValue">HPの現在値</param>
/// <param name="maxValue">HPの最大値</param>
/// <param name="reductionValue">ダメージ量</param>
1 個の参照
public void HpGauge_Damage(float cullentValue, float maxValue, float reductionValue)
{
    // hpバーのシェイク
    HpGauge_Shake();

    // 現在のHP割合の割り出しと記憶
    float percentageFrom = cullentValue / maxValue;
    // ダメージ後のHP割合の割り出しと記憶
    float percentageTo = (cullentValue - reductionValue) / maxValue;

    // 緑ゲージの見た目を被ダメージ後に更新
    greenGauge.fillAmount = percentageTo;

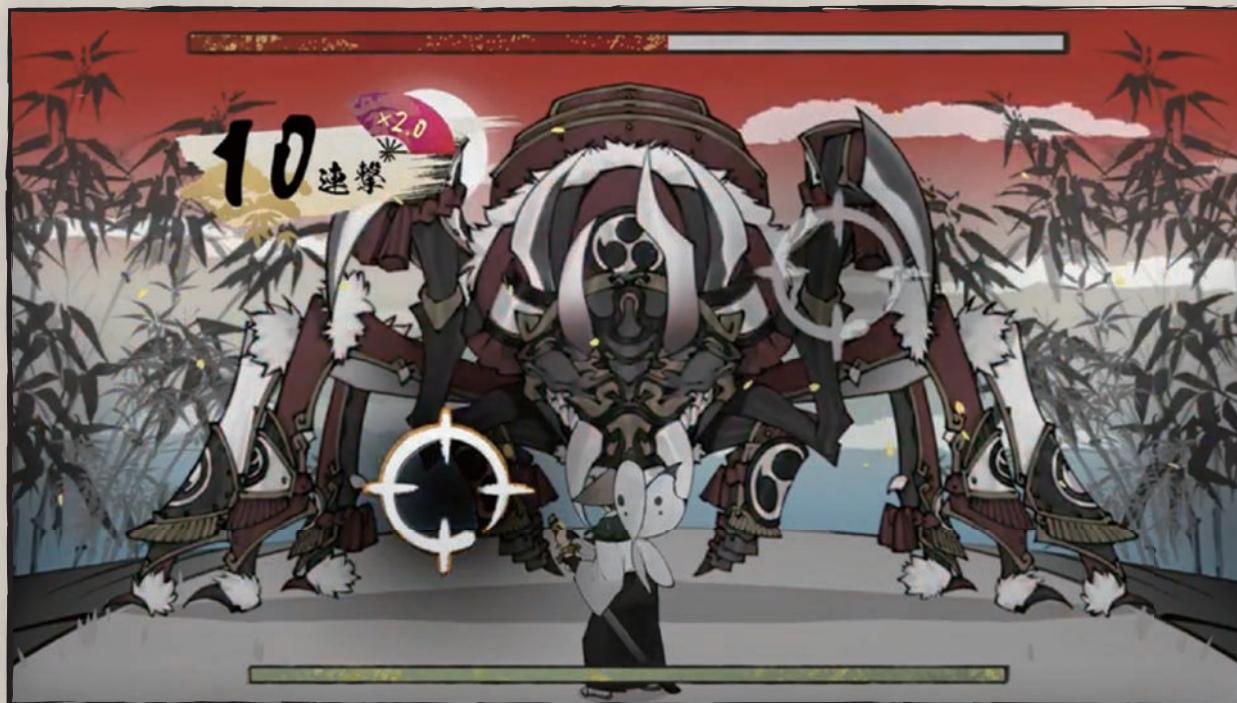
    // もし、先にDoTweenを使用していたら、処理をキルして、新しく受付可能にする
    if (decreaseGaugeTween != null)
    {
        decreaseGaugeTween.Kill();
    }

    // 1秒後に被ダメージ後のHP割合と同じ長さにする。
    decreaseGaugeTween = DOTween.To(() => percentageFrom,
        x => { redGauge.fillAmount = x; }, percentageTo, 1f);
}
```



ギミックのプールから貸出・回収 / 横田優作

事前に生成したギミックを必要な時に貸出して、ギミックが終了したときにプールに回収しています。 / 横田



```
GimmickPool.cs X
Assembly-CSharp GimmickPool
46     /// 貸出
47     /// <summary>
48     /// <returns></returns>
49     2 個の参照
50     public MainGimmickAbstract Launch_Gimmick(Vector3 newPosition)
51     {
52         // 貸出物の取得
53         MainGimmickAbstract slashGimmick = generatGimmickQueue.Dequeue();
54
55         // Queue の数が 1 以下のときにギミックを格納する
56         if(generatGimmickQueue.Count <= 1)
57         {
58             AddSlashGimmickQueue_AndStorage();
59         }
60
61         // 貸出物を返り値として返す
62         return slashGimmick;
63     }
64
65     /// <summary>
66     /// 回収
67     /// </summary>
68     /// <param name="returnGimmick"></param>
69     2 個の参照
70     public void CollectArrowGimmick(MainGimmickAbstract returnGimmick)
71     {
72         // 回収物の position を取得する
73         var _returnGimmickPosition = returnGimmick.transform.position;
74
75         // z 以外の座標を仮生成の場所に調整する
76         _returnGimmickPosition.x = tempPosition.x;
77         _returnGimmickPosition.y = tempPosition.y;
78
79         // 回収物を仮生成場所へ移動させる
80         returnGimmick.transform.position = _returnGimmickPosition;
81
82         // 回収物を Queue に格納する
83         generatGimmickQueue.Enqueue(returnGimmick);
84     }
}
```

Scene Load

Scene Load Management

/ Shota Matsuda

Used Tools

- Unity 2021.3.19f1
- Visual Studio 2022

Production time / 1 hour



When switching scenes, this class is used. When switching scenes, the curtain closes, and the next scene is loaded. / Matsuda

```
private IEnumerator LoadSceneAsync(string title)
{
    if (SceneManager.GetSceneByName(title).IsValid() == false)
    {
        Debug.LogError("シーンが見つかりません");
        yield break;
    }
    // 閉める
    sceneChangeAnimator.SetTrigger("Close");
    SoundManager.Instance.StopBGM();
    yield return new WaitForSeconds(1f); // 閉まるまで待つ

    // オペレーション
    AsyncOperation operation = SceneManager.LoadSceneAsync(title);
    // イベント登録
    operation.completed += OnLoadComplete;
    operation.allowSceneActivation = false;

    // ロードが完了したときにシーンのアクティブ化を許可する
    while (!operation.isDone)
    {
        // 現在のLoad/バーセンテージ変更
        LoadProgress = Mathf.Clamp01(operation.progress / 0.9f);
        if (operation.progress >= 0.9f)
        {
            operation.allowSceneActivation = true;
            LoadProgress = 1;
        }
        yield return null;
    }
    operation.completed -= OnLoadComplete;
}
```

攻撃チェック / 盛田翔太

```
/// <summary>
/// 加速度List更新用関数
/// </summary>
/// <param name="value">加速度</param>
1 個の参照
private void UpdateBuffer(Vector3 value)
{
    currentAccListCount++;
    currentAccListCount %= MAX_ACC_LIST_COUNT;

    if (JoyconInputManager.Instance.SixAxisHandle != null)
    {
        accBuffer[currentAccListCount] = value;
    }
}

/// <summary>
/// 加速度Yの平均値を返してくれる関数
/// </summary>
/// <returns>加速度Yの平均値</returns>
2 個の参照
private float Get_Y_AccAvg()
{
    float[] vec_y = new float[MAX_ACC_LIST_COUNT];
    for (int i = 0; i < accBuffer.Length - 1; i++)
    {
        vec_y[i] = accBuffer[i + 1 % MAX_ACC_LIST_COUNT].y - accBuffer[i].y;
    }
    return vec_y.Average();
}

/// <summary>
/// 加速度Xの平均値を返してくれる関数
/// </summary>
/// <returns>加速度Xの平均値</returns>
2 個の参照
private float Get_X_AccAvg()
{
    float[] vec_x = new float[MAX_ACC_LIST_COUNT];
    for (int i = 0; i < accBuffer.Length - 1; i++)
    {
        vec_x[i] = accBuffer[i + 1 % MAX_ACC_LIST_COUNT].x - accBuffer[i].x;
    }
    return vec_x.Average();
}
```

使用ツール

- Unity 2021.3.19f1
- Visual Studio 2022

制作時間 /4 時間

コントローラーの加速度の平均値を記録し、規定値を超えたら攻撃イベントを発行するようになっています。/ 盛田

```
private void FixedUpdate()
{
    var state = JoyconInputManager.Instance.SixAxisState;
    var acc = new Vector3(state.acceleration.x, state.acceleration.y, state.acceleration.z);
    //加速度List更新
    UpdateBuffer(acc);

    //攻撃中かどうか
    //if (isAttacking == true)
    //{
    //    if (Get_Y_AccAvg() <= reset_Y)
    //    {
    //        isAttacking = false;
    //        return;
    //    }
    //    return;
    //}
    resetTimer += Time.deltaTime;

    if (resetTimer >= attackWaitTime)
    {
        isAttacking = false;
    }

    if (isAttacking)
    {
        return;
    }

    //Debug.LogError(Get_Y_AccAvg()); //Debug
    AVG_Y = Get_Y_AccAvg(); //Debug, Inspector表示用
    AVG_X = Get_X_AccAvg(); //Debug, Inspector表示用

    AVG = AVG_X + AVG_Y;

    if ((Mathf.Abs(Get_Y_AccAvg()) + Mathf.Abs(Get_X_AccAvg())) > permissible)
    [
        Slash_AttackEvent.Invoke();
        isAttacking = true;
        resetTimer = 0;
    ]
}
```

ゲームオーバー画面 / 森美夏

ゲームオーバー時のリザルト画面です。蝶の絵をフェードインさせ、下にテキストを表示します。
リザルトはスキップさせることができます。 /森

```
GameOverManager.cs* ② X ComboManager.cs
Assembly-CSharp GameOverManager

62  ① Unity メッセージ 10 個の参照
63  private void Update()
64  {
65      //リザルトの表示スキップ
66      if (Input.GetKeyDown(KeyCode.Space)
67          || //Switch JoyconZRボタン入力)
68      {
69          //演出が終わりきったらシーン遷移
70          if (gameOverFinished)
71          {
72              //サウンドマネージャーからSE呼び出し
73              SoundManager.Instance.PlaySE(AssetService.SEType.Choice);
74
75              //ロードシーンマネージャーから_loadSceneWaitTime秒後シーン呼び出し
76              LoadSceneManager.Instance.LoadScene
77                  (LoadSceneManager.SceneName.TitleScene, _loadSceneWaitTime);
78          }
79          //終わっていなければ演出を完了させる
80          else
81          {
82              fadeTween.Complete();
83          }
84      }
85
86      //蝶の絵表示
87      ② 1 個の参照
88      IEnumerator GameOverDirectionDelay()
89      {
90          //gameOverDelayTime秒待つ
91          yield return new WaitForSeconds(gameOverDelayTime);
92
93          //DOTween.DOFade spiderFadeTimeの時間かけてアルファ値を1にする
94          //フェードの仕方、InとOutがあそめ
95          fadeTween = spiderImage.DOFade(1f, spiderFadeTime).SetEase(Ease.InOutQuad)
96          //終わった時画面下のテキスト表示
97          .OnComplete(() => GuideView_And_GameOverFinishedSetTrue());
98      }
}
90 % ③ 2 ④ 0 ⑤ ← → ⑥ ⇧ ⇩
```

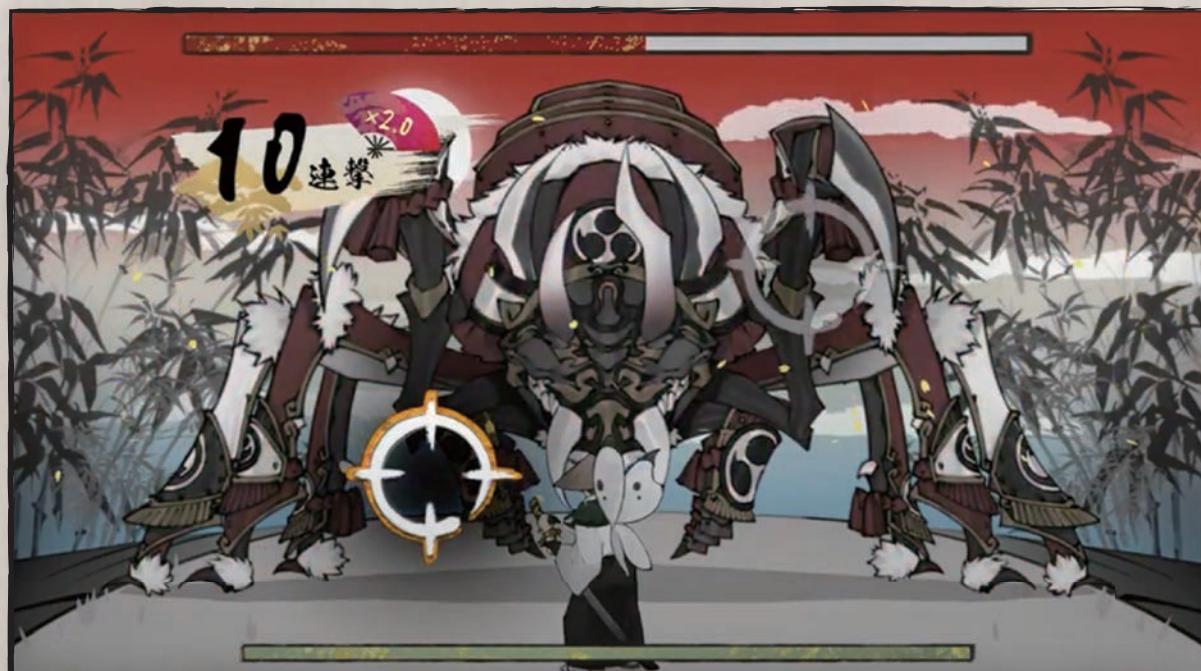


weathermap コンボの増減と演出 / 森美夏

コンボ数に応じて背景や演出を変更するスクリプトです。コンボの加算、リセットを行います。 / 森

```
ComboManager.cs  + X
Assembly-CSharp
ComboManager

117  /// <summary>
118  /// コンボが1増える
119  /// </summary>
120  3 個の参照
121  public void AddCombo()
122  {
123      //コンボ再開、フェードイン開始
124      if (_combo.Value == 0)
125      {
126          StartCoroutine(FadeIn());
127
128          //コンボ加算
129          _combo.Value += 1;
130
131          //コンボ数が倍率・梅の範囲なら
132          if (_combo.Value == _umeIndex)
133          {
134              //intに変換 ChinesePlumはもともとenumで名前を付けたので分かりやすい
135              _comChange.sprite = _comboOrnamentList[(int)comboOrnamentType.ChinesePlum]; //装飾：梅
136              _comboFanController.ChangeFanSprites(FanType.Fan1, 1.5f); //倍率：1.5倍
137              _comboBG.sprite = _comboBG_SpriteList[(int)comboBG_Type.second]; //背景：2番目
138
139          //メインゲームマネージャーのcomboStageの加算
140          _mainGameManager.AddStageNum();
141
142          //コンボ数が倍率・松の範囲なら
143          if (_combo.Value == _matuIndex)
144          {
145              _comChange.sprite = _comboOrnamentList[(int)comboOrnamentType.PineTree]; //装飾：松
146              _comboFanController.ChangeFanSprites(FanType.Fan2, 2f); //倍率：2.0倍
147              _comboBG.sprite = _comboBG_SpriteList[(int)comboBG_Type.third]; //背景：3番目
148
149          //メインゲームマネージャーのcomboStageの加算
150          _mainGameManager.AddStageNum();
151
152      }
153  }
```



アーキテクチャに基づいた扇 UI の制作 / 大長ルミナ

MVC のアーキテクチャに沿ったプログラム作成を行いました。仕様変更に強く、バグが発生した際の発見が早くなるようメソッドを分けました。MVC に機能ごとにメソッドを分けるのに苦労しました。
処理を呼ぶ流れを意識しながら作ることで処理が明確になり、格段に解読のしやすさが上がりました。 / 大長



```
public class ComboFanController : MonoBehaviour
{
    [SerializeField] private ComboFanView _fanView;
    private Coroutine _currentCoroutine;

    /// <summary>
    /// 扇スプライトの変更を指示する関数
    /// フェードを行い終了するまで待つ、その後スプライトの変更を行う
    /// </summary>
    /// <param name="fanType">変更したい扇のタイプ</param>
    /// <param name="powerIndex">倍率</param>
    10 個の参照
    public void ChangeFanSprites(FanType fanType, float powerIndex = 1.0f)
    {
        if (_currentCoroutine != null)
        {
            // 既にコルーチンが実行中の場合は終了まで待機する
            StartCoroutine(WaitForCoroutineFinish(fanType, powerIndex));
        }
        else
        {
            _currentCoroutine = StartCoroutine(ChangeFanSpritesCoroutine(fanType, powerIndex));
        }
    }

    1 個の参照
    private IEnumerator WaitForCoroutineFinish(FanType fanType, float powerIndex)
    {
        while (_currentCoroutine != null)
        {
            yield return null;
        }

        // 全てのコルーチンが終了した後に新たなコルーチンを開始する
        _currentCoroutine = StartCoroutine(ChangeFanSpritesCoroutine(fanType, powerIndex));
    }

    2 個の参照
    private IEnumerator ChangeFanSpritesCoroutine(FanType fanType, float powerIndex)
    {
        // Viewに変更するタイプを伝える
        _fanView.CurrentType = fanType;
        // フェード(は終わるまで待機
        yield return StartCoroutine(_fanView.FadeOutFan());

        // None(コンボミス)以外は次の扇のスプライトに変更する
        if (_fanView.CurrentType != FanType.None)
        {
            yield return StartCoroutine(_fanView.UpdateFanSprite(fanType, powerIndex));
        }

        _currentCoroutine = null;
    }
}
```

```
    /// <summary>
    /// 扇のスプライトを変更するコルーチン
    /// </summary>
    /// <param name="fanType">変更したい扇タイプ</param>
    /// <param name="powerIndex">倍率</param>
    1 個の参照
    public IEnumerator UpdateFanSprite(FanType fanType, float powerIndex)
    [
        // Noneの場合は透明なスプライトを割り当てる
        if (fanType == FanType.None)
        {
            _fanImage.sprite = _fanSpriteList[0];
        }
        else
        {
            List<Sprite> spriteList = GetFanSpriteList(fanType);

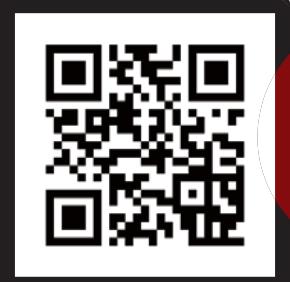
            if (spriteList == null)
            {
                Debug.LogError("スプライトのリストが存在しません！");
                yield break;
            }

            // 扇が開く前に倍率を設定しておく
            _tmp.text = $"×{powerIndex:0.0}";

            // 指定秒で開き終わるために一枚にかかる時間を計算する
            float waitTime = _displayTime / spriteList.Count;

            // 変更する
            foreach (Sprite sprite in spriteList)
            {
                _fanImage.sprite = sprite;
                yield return new WaitForSeconds(waitTime);
            }
        }
    ]
}
```

GitHub



ノーツのジャッジのプログラム作成 / 大長ルミナ

ノーツに対して判定処理が走った際に時間経過で成功か、失敗かを判定するプログラムを作りました。

コメントでは3分のXになっていますが変数の変更ですべて変えることができ、判定の厳しさを設定できるようになっています。 /大長

```
public JudgmentType Judge(GimmickType type)
{
    // 判定開始前
    if (WaitTime > _elapsedTime)
    {
        return JudgmentType.Yet;
    }

    // 中心のBad範囲に入った場合
    if (_elapsedTime >= ContractionTime)
    {
        Destroy(gameObject);
        return JudgmentType.Bad;
    }

    // Good判定内 && Good判定内の3分の1に達するまで
    else if (_elapsedTime <= ContractionTime && _elapsedTime >= ContractionTime - GoodContractionTime / 3)
    {
        Destroy(gameObject);
        return JudgmentType.Good;
    }

    // Good判定内 && Good判定内の3分の2に達するまで
    else if (_elapsedTime <= ContractionTime && _elapsedTime >= ContractionTime - GoodContractionTime + GoodContractionTime / 3)
    {
        Destroy(gameObject);
        return JudgmentType.Good;
    }

    // Good判定内 && Good判定内の3分の3に達するまで
    else if (_elapsedTime - GoodContractionTime / 2 <= ContractionTime && _elapsedTime >= ContractionTime - GoodContractionTime)
    {
        Destroy(gameObject);
        return JudgmentType.Good;
    }

    // Good判定外 && 判定を取り始めている場合
    else if (_elapsedTime <= WaitTime)
    {
        Destroy(gameObject);
        return JudgmentType.Bad;
    }

    // それ以外はBad
    else
    {
        Destroy(gameObject);
        return JudgmentType.Bad;
    }
}
```



Addressables を用いたデータ管理と サウンドシステムの作成 / 大長ルミナ

SE や BGM を必要なシーンで必要なものだけをロードするシステムを作りました。不需要にならメモリを開放し、処理を軽くするよう心掛けました。Addressables で保持したものから読み込んで音を鳴らすサウンドシステムも作りました。

今までプロジェクトで使う全ての SE や BGM、スクリプトなどをロードしていました。これにより不必要的メモリを保持し続けることになってしまい、メモリの管理を学ぼうと思いました。このプログラムをきっかけにメモリ管理について気にかけるようになりました。 /大長

```
/// <summary>
/// 引数と一致するBGMのClipを取得し再生する
/// </summary>
/// <param name="bgmName">再生したいBGMType</param>
/// 1 個の参照
public async void PlayBGM(AssetService.BGNTYPE bgmName, float volume = 0.5f)
{
    //None間違えて引数に入ってきた場合ははじく
    if (bgmName == AssetService.BGNTYPE.None)
    {
        Debug.LogWarning("[TypeName:None] を再生しようとされたため、処理を中断しました。");
        return;
    }
    else if (_bgm AudioSource.isPlaying && bgmName == _currentBGM)
    {
        _bgm AudioSource.Play();
        // フェード後は音量が0になっているため音量を調整する
        _bgm AudioSource.volume = volume;
        return;
    }
    else if (_bgm AudioSource.isPlaying && bgmName == _currentBGM)
    {
        return;
    }

    var newClip = await AssetService.Instance.LoadBGMAudio(bgmName);
    if (newClip == null)
    {
        Debug.LogWarning("Clipがないため再生できないよ！");
        return;
    }

    if (_bgm AudioSource.isPlaying)
    {
        // フェードアウトの開始
        await FadeOutBGM();
    }

    // AudioSourceに新しいBGMを設定する
    _bgm AudioSource.clip = newClip;
    // 音量を調整する
    _bgm AudioSource.volume = volume;
    // 再生する
    _bgm AudioSource.Play();
    // 現在再生しているBGMを更新する
    _currentBGM = bgmName;
}
```

```
/// <summary>
/// BGMのPathを作りアセットをロード
/// </summary>
/// <param name="orderBGM">再生したいBGM</param>
/// <returns></returns>
1 個の参照
public async UniTask<AudioClip> LoadBGMAudio(BGNTYPE orderBGM)
{
    //前のBGMのメモリを開放する
    if (_currentBGMClip.IsValid())
    {
        Addressables.Release(_currentBGMClip);
    }

    string orderPath = _soundCommonPath;

    switch (orderBGM)
    {
        case BGNTYPE.Entry:
            orderPath += _entryPath;
            break;
        case BGNTYPE.Ready:
            orderPath += _readyPath;
            break;
        case BGNTYPE.MainGame:
            orderPath += _mainGamePath;
            break;
        case BGNTYPE.Result:
            orderPath += _resultPath;
            break;
    }

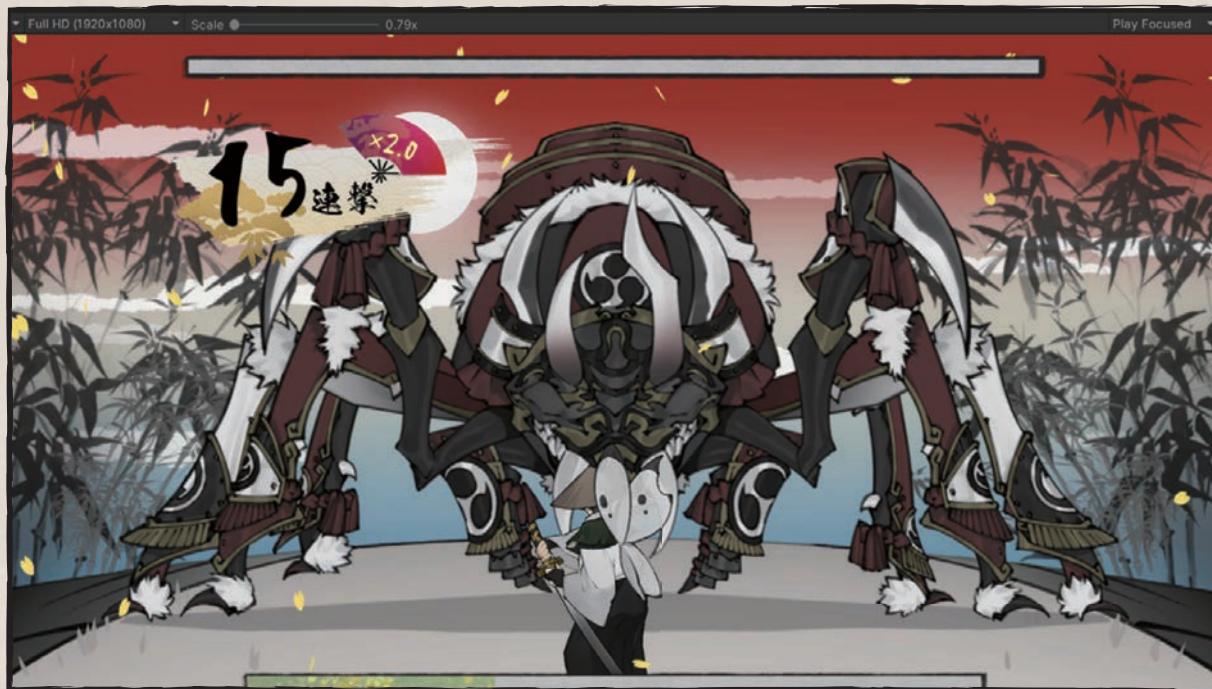
    // 用意したPathからBGMデータをロード
    _currentBGMClip = Addressables.LoadAssetAsync<AudioClip>(orderPath);
    await _currentBGMClip.Task;
    return _currentBGMClip.Result;
}

/// <summary>
/// SEのPathを作りアセットをロードする
/// </summary>
/// <param name="orderSE"></param>
```

❖ 篠の葉、紅葉、桜の花びらのパーティクル / 鳥海穂菜実

使用ツール

- Visual Studio 2022 パーティクル自体のサイズの変更を誰でも変えられるように書きました。
- 制作時間 /3 時間 任意でマテリアルが変えられるようにしました。 / 鳥海



1 個の参照

```
public void ChangeParticle_Bamboo()
{
    // 桜の花びらに変更
    particle.material = m_bamboo;

    // サイズ変更:桜
    transform.localScale = new Vector3(bambooSize, bambooSize, 1);
}
```

1 個の参照

```
public void ChangeParticle_Momiji()
{
    // 紅葉の葉っぱに変更
    particle.material = m_momiji;

    // サイズ変更:紅葉
    transform.localScale = new Vector3(momijiSize, momijiSize, 1);
}
```

1 個の参照

```
public void ChangeParticle_Sakura()
{
    // 桜の花びらに変更
    particle.material = m_sakura;

    // サイズ変更:桜
    transform.localScale = new Vector3(sakuraSize, sakuraSize, 1);
}
```

Youtube



主人公の Spine 制御 / 鳥海穂菜実

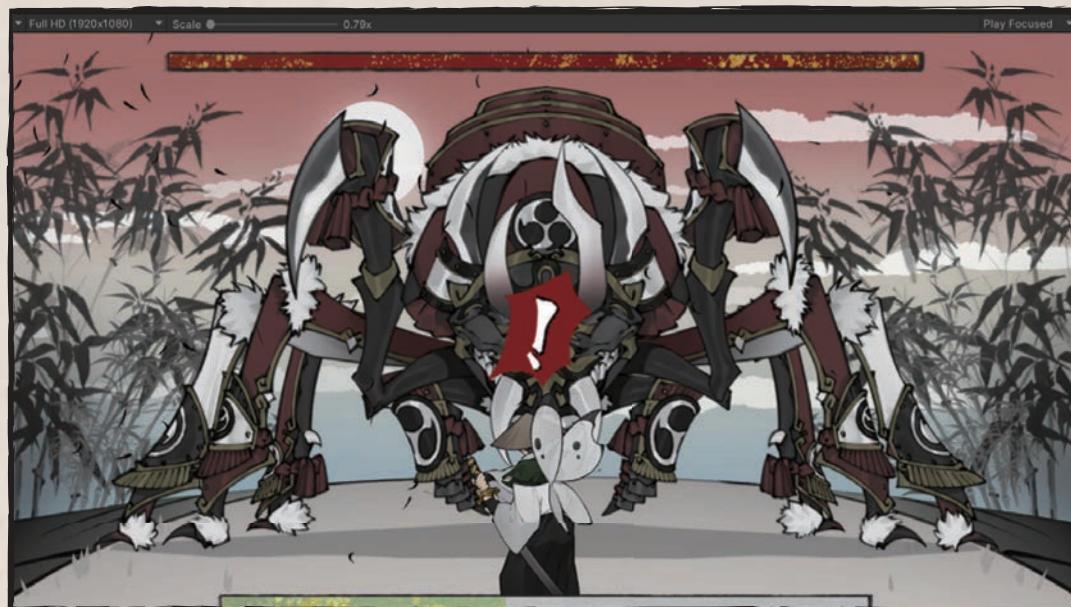
使用ツール

・Visual Studio 2022

制作時間 /3 日

Spine の導入、動作確認、スクリプトを作成しました。

導入が難しく、何度も試行錯誤を繰り返して作成しました。 / 鳥海



```
[SerializeField]
private string testAnim0 = "Idle",
            testAnim1 = "Attack";
//testAnim2 = "run";

private SkeletonAnimation skeltonAnim = default;
private Spine.AnimationState player_SpineAnimState = default;
private SkeletonData skeletonData;

TrackEntry trackEntry;

④Unity メッセージ10 個の参照
void Start()
{
    skeltonAnim = GetComponent<SkeletonAnimation>();
    player_SpineAnimState = skeltonAnim.AnimationState;
    //
    PlayerSkeltonAnim_Idle();
}

1 個の参照
private void PlayerSkeltonAnim_Idle()
{
    //前のアニメを消去(残っていると再生終了後におかしくなる事がある)らしい
    skeltonAnim.state.ClearTrack(0);

    player_SpineAnimState.SetAnimation(0, testAnim0, true);
    skeltonAnim.skeleton.SetToSetupPose();
}
```

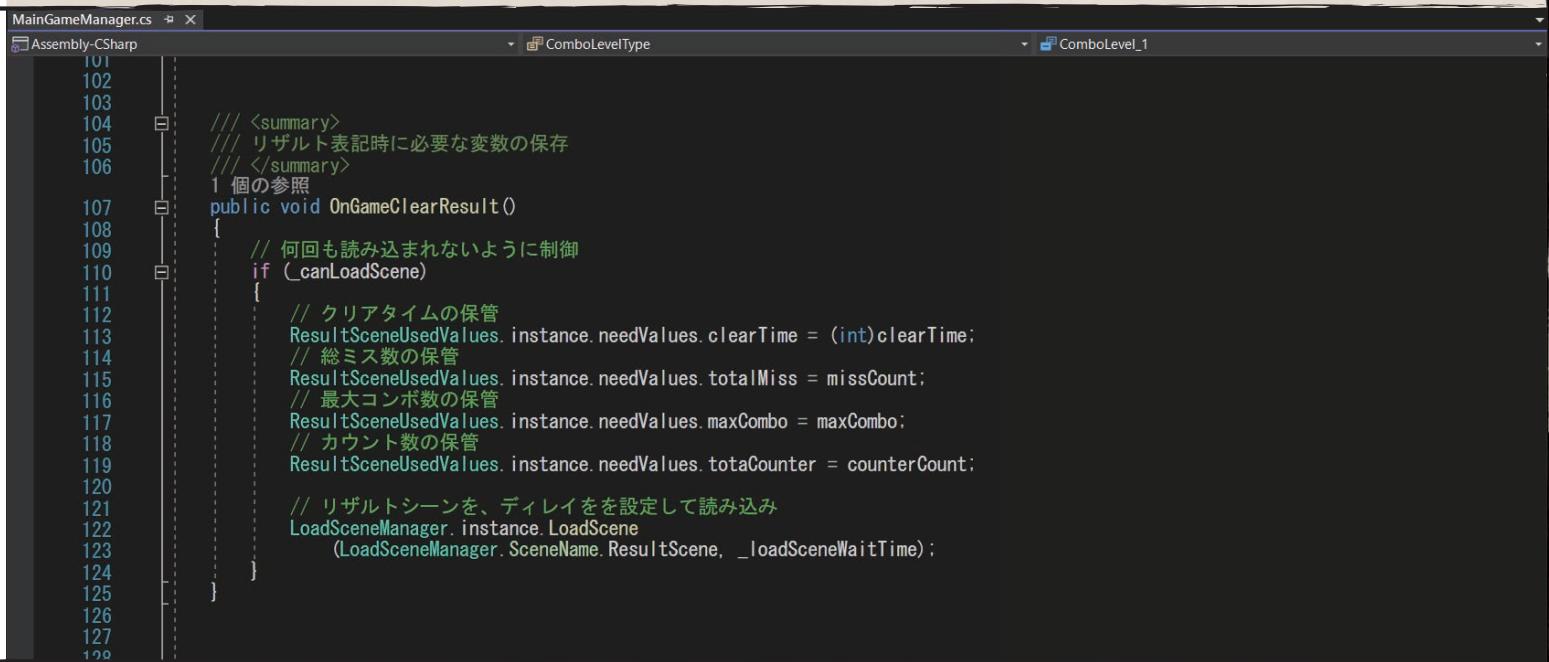
リザルトに必要な値の引き渡し / 関口柊哉

使用ツール

・Visual Studio 2022

制作時間 /1 時間

リザルトに必要な数値を、シーンを超えるオブジェクトに保存して
次のシーンで使用できるようにしています。 /関口



```
MainGameManager.cs 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133

101
102
103
104  /// <summary>
105  /// リザルト表記時に必要な変数の保存
106  /// </summary>
107  1 個の参照
108  public void OnGameClearResult()
109  {
110      // 何回も読み込まれないように制御
111      if (_canLoadScene)
112      {
113          // クリアタイムの保管
114          ResultSceneUsedValues.instance.needValues.clearTime = (int)clearTime;
115          // 総ミス数の保管
116          ResultSceneUsedValues.instance.needValues.totalMiss = missCount;
117          // 最大コンボ数の保管
118          ResultSceneUsedValues.instance.needValues.maxCombo = maxCombo;
119          // カウント数の保管
120          ResultSceneUsedValues.instance.needValues.totaCounter = counterCount;
121
122          // リザルトシーンを、ディレイを設定して読み込み
123          LoadSceneManager.instance.LoadScene
124              (LoadSceneManager.SceneName.ResultScene, _loadSceneWaitTime);
125      }
126  }

127
128
129
130
131
132
133
```

α版ギミックの生成と削除 / 関口柊哉

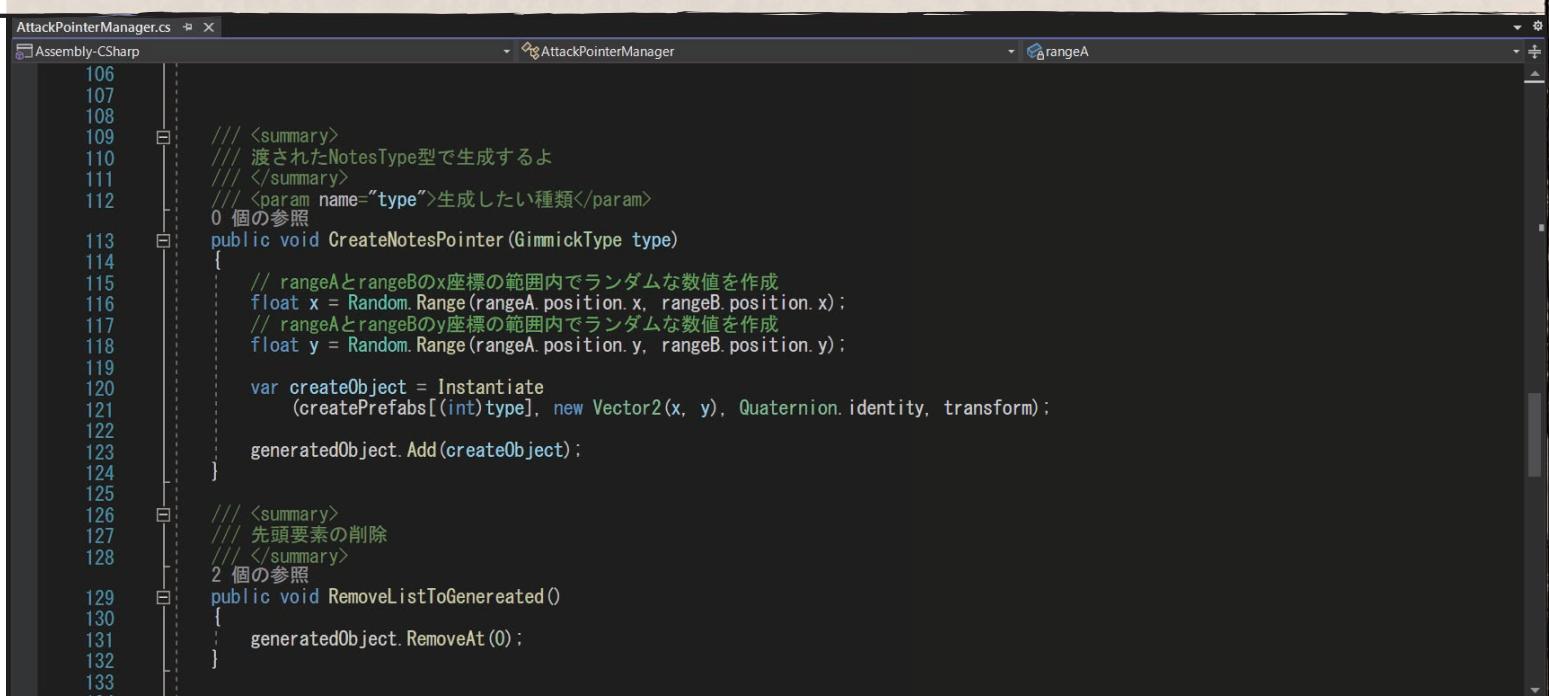
使用ツール

・Visual Studio 2022

制作時間 /4 時間

ギミックをランダムな位置に生成して管理のリストに追加しています。

ギミックが一種だったため、ソート無しでも先頭を削除します。 /関口



```
AttackPointerManager.cs 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133

106
107
108
109  /// <summary>
110  /// 渡されたNotesType型で生成するよ
111  /// </summary>
112  /// <param name="type">生成したい種類</param>
113  0 個の参照
114  public void CreateNotesPointer(GimmickType type)
115  {
116      // rangeAとrangeBのx座標の範囲内でランダムな数値を作成
117      float x = Random.Range(rangeA.position.x, rangeB.position.x);
118      // rangeAとrangeBのy座標の範囲内でランダムな数値を作成
119      float y = Random.Range(rangeA.position.y, rangeB.position.y);

120      var createObject = Instantiate
121          (createPrefabs[(int)type], new Vector2(x, y), Quaternion.identity, transform);
122
123      generatedObject.Add(createObject);
124  }

125
126  /// <summary>
127  /// 先頭要素の削除
128  /// </summary>
129  2 個の参照
130  public void RemoveListToGenerated()
131  {
132      generatedObject.RemoveAt(0);
133  }
```

TEAM MUSHIGIRI



蟲斬