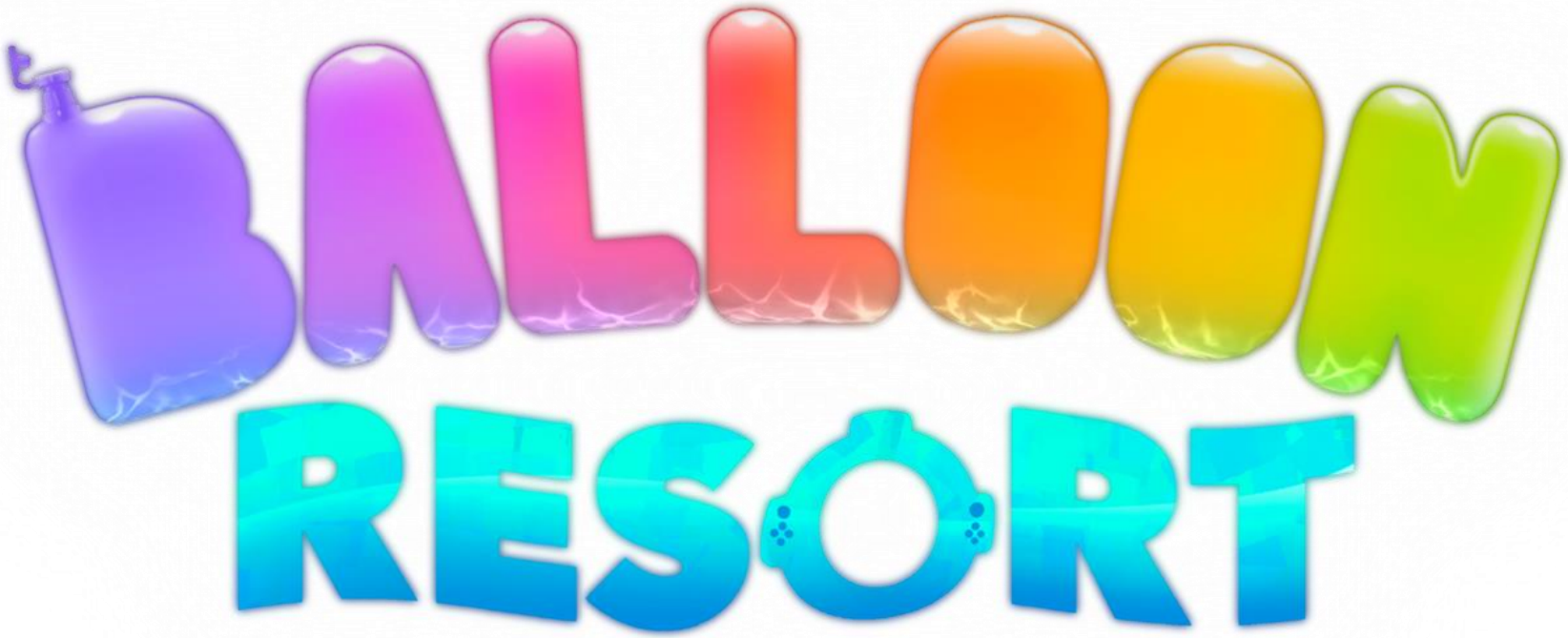
Balloon Resort ポートフォリオ

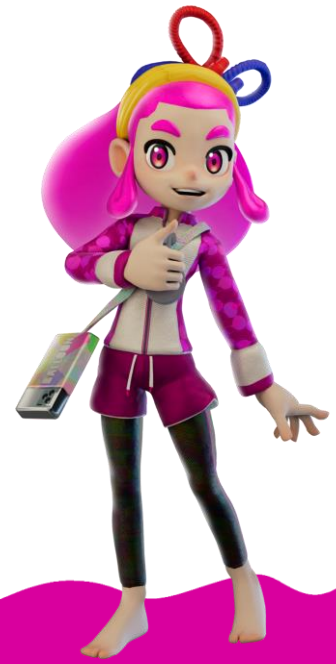


東京工芸大学 芸術学部
ゲーム学科 プログラム分野 3年
守本 拓翔

自己紹介

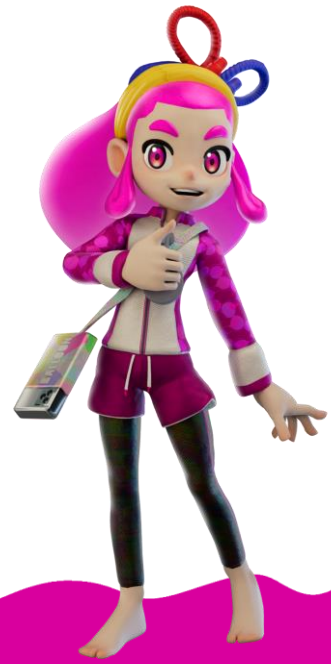
東京工芸大学 芸術学部 ゲーム学科 プログラム分野 3年 守本 拓翔

- プログラミング言語・ゲームエンジン等の経験
  - C# (2年半)
  - C (1年)
  - C++ (1年半)
  - Unity (2年半)
  - Unreal Engine 4 (半年)
  - DirectX12 (2ヶ月)
- 競技プログラミングの成績(2024年1月8日時点)
  - AtCoder 茶色 (最高値 : 758, 現在 : 758)
  - Paiza Sランク (最高値 : 2097, 現在 : 2097)



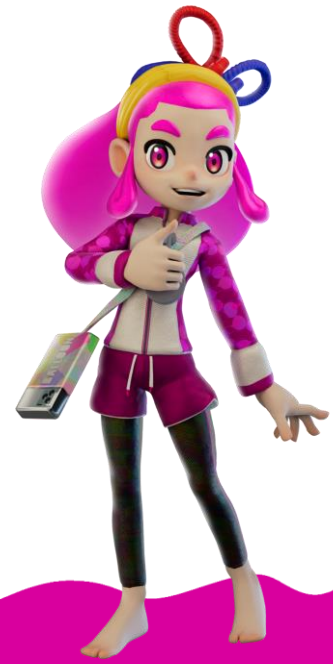
目次

1. 概要
2. 担当箇所
3. 本制作での目標
4. 担当ソースコード
5. 工夫点



1. 概要

- ジャンル
  - 3Dアクションゲーム
- 開発目的
  - 工芸ゲームショウならびにGFF AWARD 2024への出場
- 開発期間
  - 2023年 6月18日 ～ 2024年 1月6日 (6ヶ月19日)
- チーム構成
  - 企画 2名 デザイン4名 プログラム2名（計8名）

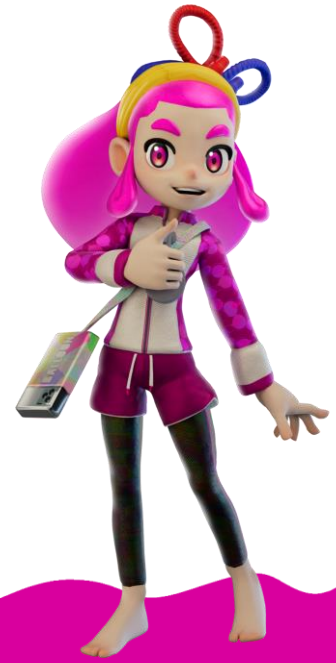


1. 概要

- 開発環境
  - Unity2022.3.3f1(URP)
  - VisualStudio2022
  - C#
  - GitHub
  - GitHub Desktop

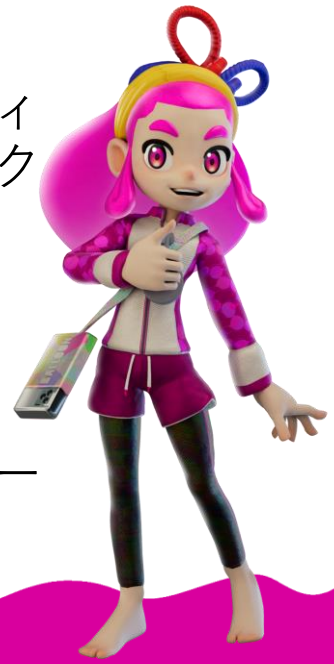
- 企画書 URL
  - 企画書

- GitHub URL
  - プロジェクトファイル



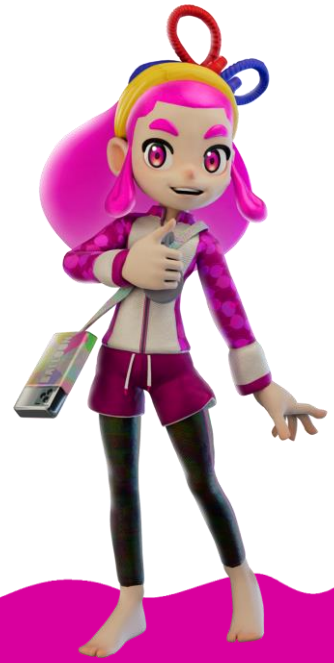
2. 担当箇所

- プレイヤー
  - プレイヤー
  - 風船
  - カメラ
- 敵
  - ハリセンボン
  - ヤドカリ
- ギミック
  - 風
  - 水
  - 空気栓（設計のみ）
- インゲーム
  - 流れ
- アウトゲーム
  - クリア画面
  - タイトル画面
- 隠し要素
  - サンタ
- UI
  - 制限時間
  - ゲーム開始時
  - ゲーム終了時
- サウンド
  - インタラクティブミュージック
- その他
  - エディタ拡張
  - コードレビュー



3. 本制作での目標

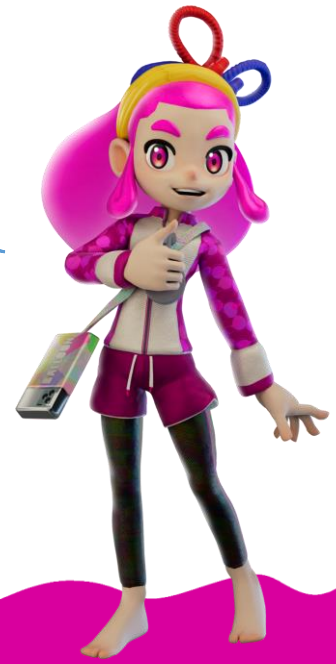
- チームのレベルに合った開発をする
  - ペアプログラミング、コードレビューの徹底、わかりやすい命名
- 担当間の連携を取りやすくする
  - コードレビューの徹底、インターフェース/基底クラスの実装、マニュアルの作成
- 拡張/変更/削除がしやすい設計にする
  - SOLID原則、カプセル化、疎結合、高凝集



4. 担当ソースコード

- 目次
  - Animationフォルダ
  - BalloonPlantフォルダ
  - Cameraフォルダ
  - Easingフォルダ
  - Editorフォルダ
  - Enemyフォルダ
  - Gimmickフォルダ
  - Gravityフォルダ
  - Ingameフォルダ
  - Joy-Conフォルダ
  - OutGameフォルダ
  - Playerフォルダ
  - Santaフォルダ
  - Scoreフォルダ
  - Soundフォルダ
  - Sunglassesフォルダ
  - TimeLimitフォルダ
  - Utilityフォルダ
  - Input

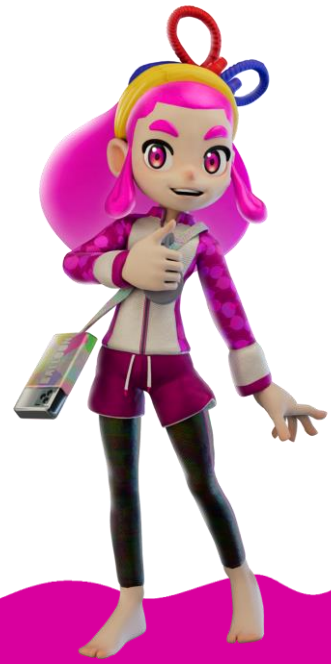
全58スクリプト＋αです。
特に見てほしいものは
目次上で赤字にしておきました！



4. 担当ソースコード

- Animationフォルダ

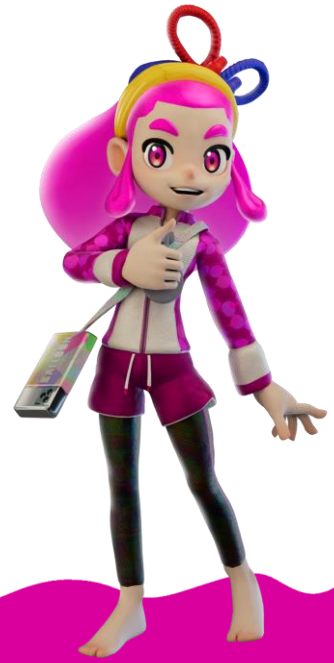
ファイル名	概要
AnimationChanger.cs	ジェネリックを活用したアニメーション変更クラス。 Unityが提供するAnimatorControllerを用いたアニメーションの変更では文字列指定があり、ヒューマンエラー防止のために作成。
AnimatorStateData.cs	AnimatorControllerに配置されたステート名を管理するスクリプタブルオブジェクト。 後述するエディタ拡張により自動生成される。
Enumフォルダ 例：E_Atii.cs	AnimationChanger.csのジェネリックで指定されるenumのみを定義したスクリプトが格納されたフォルダ。 後述するエディタ拡張により自動生成される。



4. 担当ソースコード

- BalloonPlantフォルダ

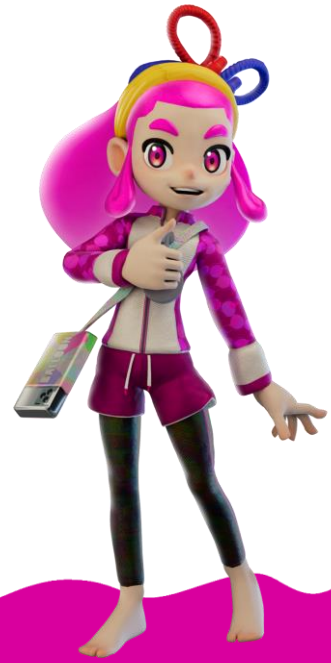
ファイル名	概要
BalloonPlantController.cs	景観オブジェクト用スクリプト。 プレイヤーが近づいた状態でプッシュすると風船モチーフの花が膨らむ。



4. 担当ソースコード

- Cameraフォルダ

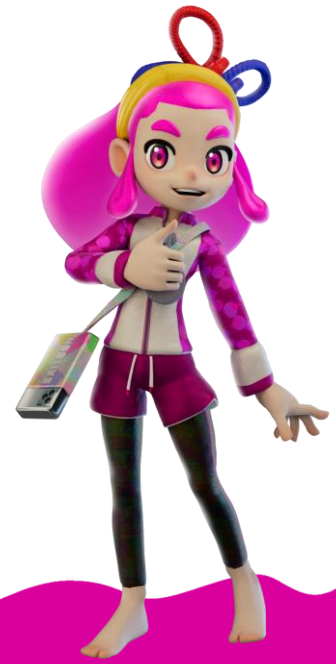
ファイル名	概要
CinemachineController.cs	カメラの動きを制御するクラス。 Unityが提供するCinemachineを用いている。



4. 担当ソースコード

- Easingフォルダ

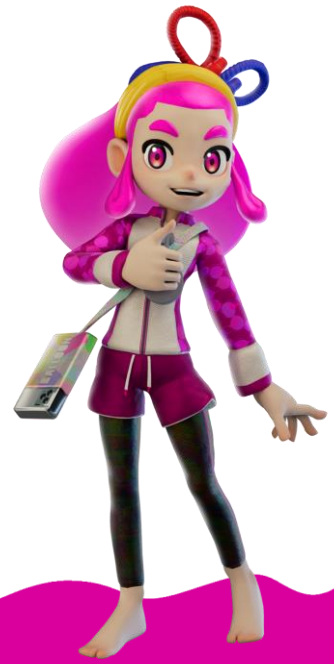
ファイル名	概要
EaseType.cs	イージングの種類を定義したenum型。 例としてInOutCubicがある。
EasingManager.cs	イージングの進行度[0-1]を返すメソッドがあるstaticなクラス。 計算式参考サイト



4. 担当ソースコード

- Editorフォルダ

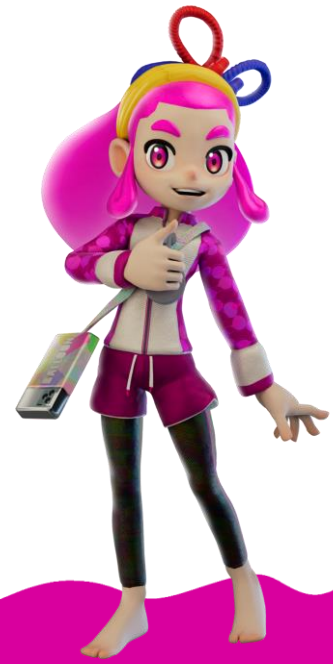
ファイル名	概要
AnimatorStateExporter.cs	ウィンドウを表示するエディタ拡張。 指定したAnimatorControllerにあるstate名を読み取り、前述のAnimationChanger.csで使用できるようにスクリプタブルオブジェクトとenumを自動生成する。
BaseEnum.txt	上記enumの自動生成のベースとなるテキスト
Curve01Drawer.cs	AnimationCurveを0-1の範囲のみで表示するようにするエディタ拡張。
SoundSettingsEditor.cs	視聴しながら音量、ピッチを調整できるエディタ拡張。



4. 担当ソースコード

- Enemyフォルダ

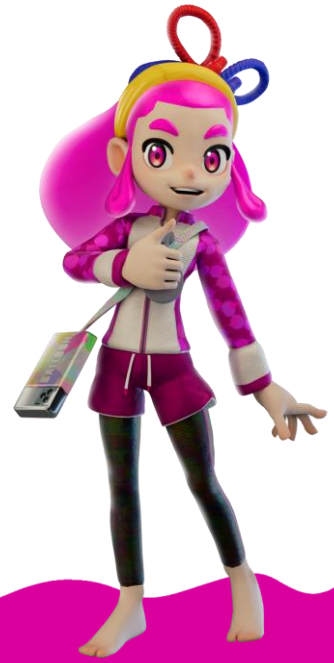
ファイル名	概要
HermitCrabController.cs	敵「ヤドカリ」の動きを管理するクラス。 ステートパターンを用いている。
HoneybeeController.cs	敵「ハリセンボン(旧:ミツバチ)」の動きを管理するクラス。



4. 担当ソースコード

- Gimmickフォルダ（1/3）

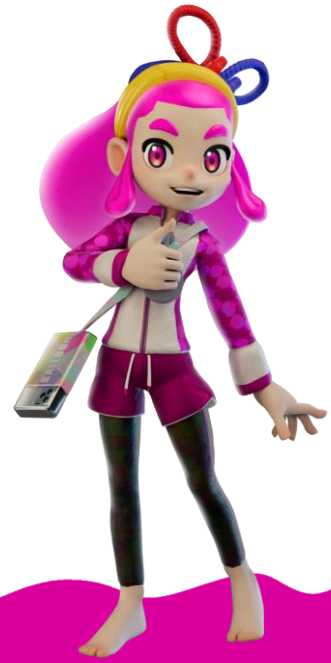
ファイル名	概要
AirVentInteractable.cs	空気栓でインタラクトできるものの基底クラス。 Inspectorで管理するために、interfaceではなく基底クラスとして管理している。 設計を担当した。
AirventEvent.cs	空気栓の周囲に出入りした際の通知を送るクラス。 オブザーバーパターン。 IObservable<T>やIObserver<T>、EventArgs、EventHandler<T>は用いていない。 用いていない理由は、チームメンバーがオブザーバーパターン、インターフェース等に慣れていないため、最もシンプルな形式で慣れてもらうためである。 以降に出てくるオブザーバーパターンを用いたものも上記項目は用いていないが、同じ理由のため割愛する。



4. 担当ソースコード

- Gimmickフォルダ（2/3）

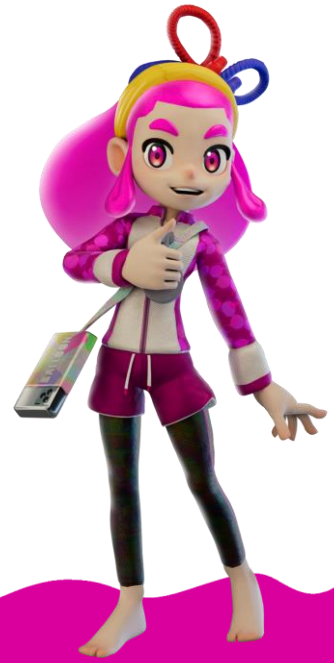
ファイル名	概要
FunController.cs	送風機の動きを制御するクラス。
FunEvent.cs	送風機の影響範囲内に出入りした際の通知を送るクラス。 オブザーバーパターン。



4. 担当ソースコード

- Gimmickフォルダ（3/3）

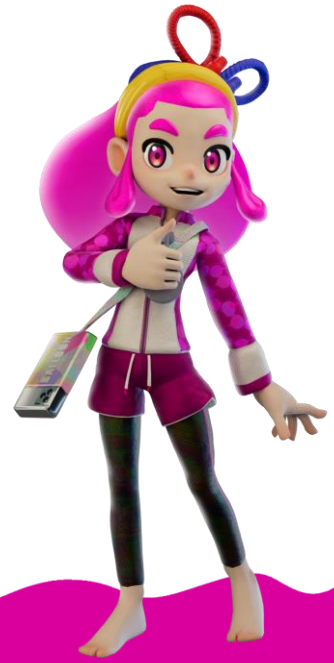
ファイル名	概要
Water.cs	水に関連する動作を集約したクラス。
WaterEvent.cs	水に出入りした際に通知を送るクラス。 オブザーバーパターン。



4. 担当ソースコード

- Gravityフォルダ

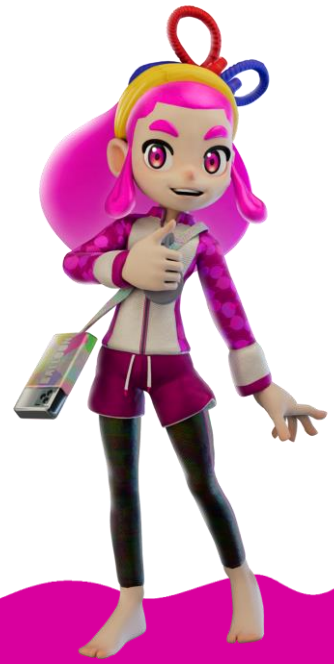
ファイル名	概要
IAdjustGravity.cs	重力を調整するインターフェース。



4. 担当ソースコード

- Ingameフォルダ

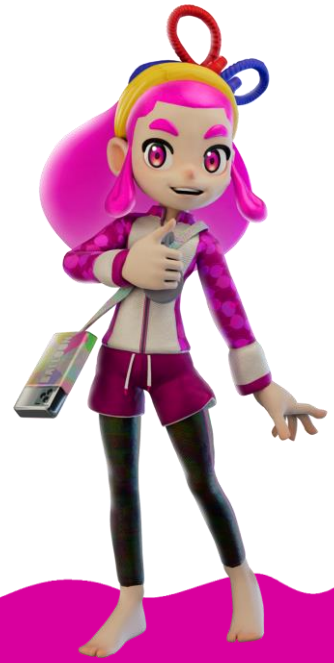
ファイル名	概要
GameFinishController.cs	ゲーム終了時処理を実装したクラス。
GameFinishView.cs	ゲーム終了時処理の描画部分を担当したクラス。
GameStartUIController.cs	ゲーム開始時のUIの処理を実装したクラス。
GameStartUIView.cs	ゲーム開始時のUIの描画部分を担当したクラス。
GameStarter.cs	ゲームを開始するクラス。
IngameController.cs	インゲームの流れを制御するクラス。
IngameStartEvent.cs	インゲームが開始したことを通知するクラス。 オブザーバーパターン。



4. 担当ソースコード

- Joy-Conフォルダ

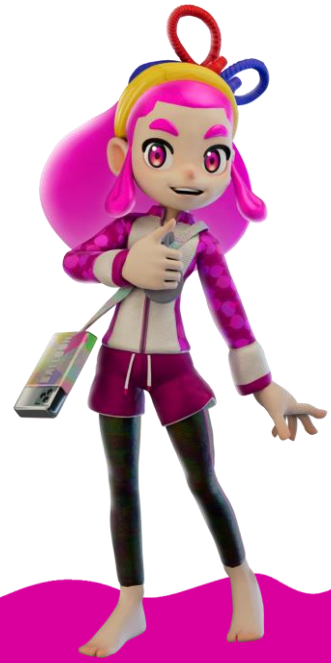
ファイル名	概要
JoyconHandler.cs	外部ライブラリの機能を使いやすくカプセル化したもの。



4. 担当ソースコード

- OutGameフォルダ

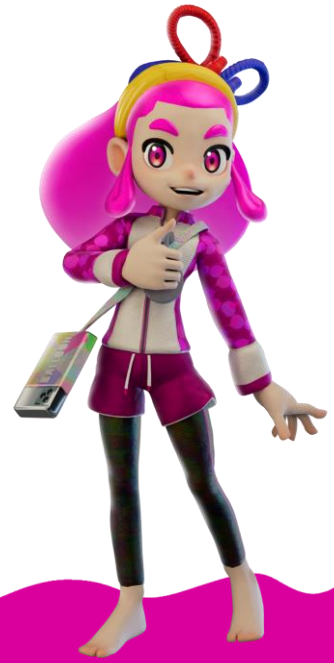
ファイル名	概要
SuccessSceneController.cs	クリア画面の処理を実装したクラス。
SuccessSceneView.cs	クリア画面の処理の描画部分を担当したクラス。
TitleUIController.cs	タイトルUIの処理を実装したクラス。
TitleUIView.cs	タイトルUIの処理の描画部分を担当したクラス。



4. 担当ソースコード

- Playerフォルダ（1/4）

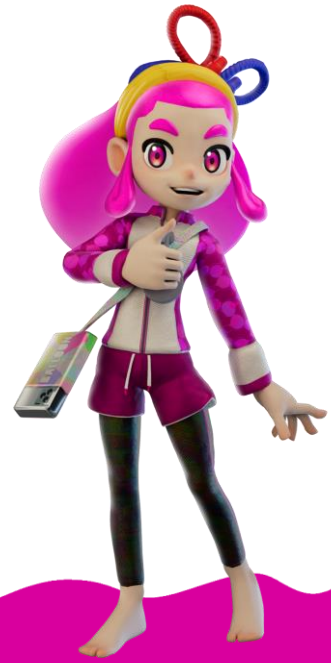
ファイル名	概要
BalloonController.cs	プレイヤーの風船部分の処理をカプセル化したクラス。



4. 担当ソースコード

- Playerフォルダ（2/4）

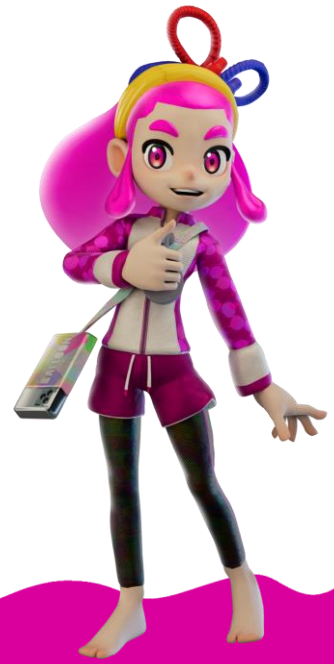
ファイル名	概要
BoostDashEvent.cs	プレイヤーの特定アクション時に通知を送るクラス。オブザーバーパターン。
IHittable.cs	プレイヤーとヒット時の処理を実装させるインターフェース。
PlayerGameOverEvent.cs	プレイヤーがゲームオーバー時に通知を送るクラス。オブザーバーパターン。



4. 担当ソースコード

- Playerフォルダ（3/4）

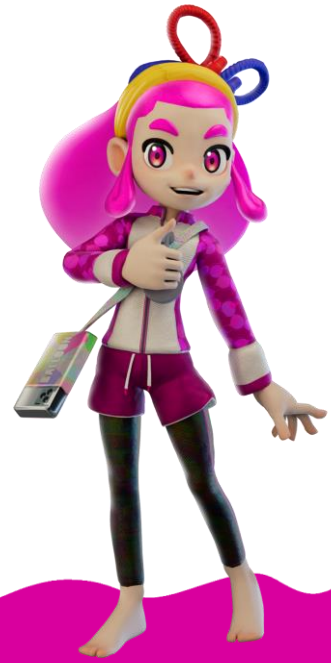
ファイル名	概要
DeflatablePlayer.cs	風船が萎んでいる際のプレイヤーの処理が実装されたクラス。IPlayer.csを継承している。
IPlayer.cs	プレイヤーが行えるアクションを実装させるインターフェース。
InflatablePlayer.cs	風船が膨らんでいる際のプレイヤーの処理が実装されたクラス。IPlayer.csを継承している。
PlayerController.cs	プレイヤーの動きの流れを制御するクラス。 IPlayer.csを所持し、風船の膨らみによりインスタンスを変更することで処理を呼び分けている。 ステートパターンを用いている。



4. 担当ソースコード

- Playerフォルダ（4/4）

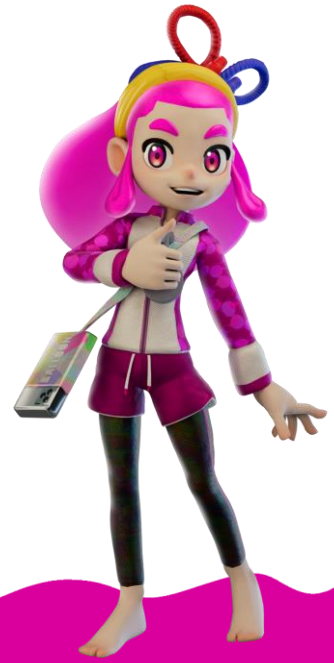
ファイル名	概要
PlayerFootStepSoundReplayer.cs	プレイヤーの足音再生処理を実装したクラス。Animationのキーフレームからイベントとして呼ばれる。
PlayerParameter.cs	プレイヤーのレベルデザイン情報となるパラメータがカプセル化されたもの。



4. 担当ソースコード

- Santaフォルダ

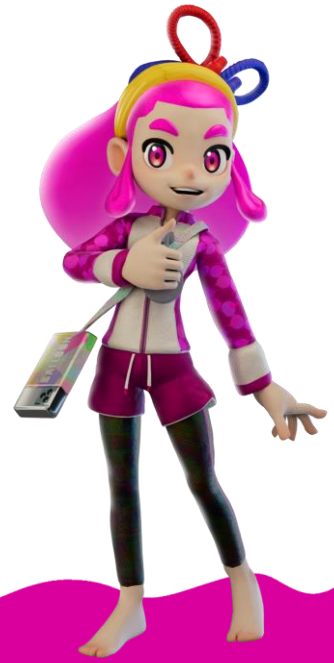
ファイル名	概要
SantaEventArea.cs	サンタに近づいた際にイベントが挟まるエリアに入った際の処理を実装したクラス。



4. 担当ソースコード

- Scoreフォルダ

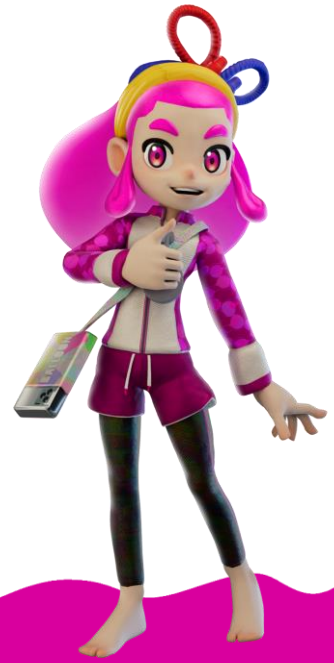
ファイル名	概要
ScoreManager.cs	スコアの管理クラス。 値オブジェクトに近い機能を持つ。



4. 担当ソースコード

- Soundフォルダ（1/2）

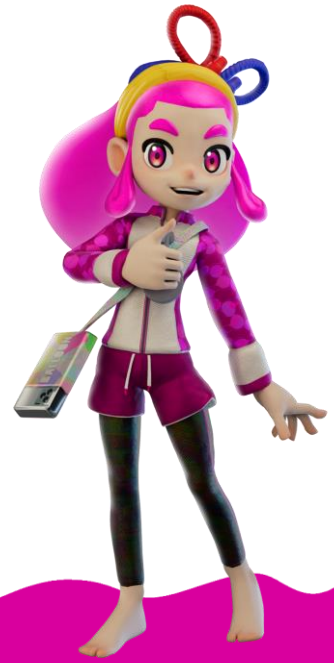
ファイル名	概要
BGMTrackChanger.cs	次のBGMを再生する機能を持つクラス。 インタラクティブミュージック。
BPMEvent.cs	BGMのビートを刻むタイミングで通知を送るクラス。
SoundKVPair.cs	サウンド管理で用いているstringと設定のペア。 シリアライズできないDictionary<TKey,TValue>の代わりのクラス。
SoundList.cs	サウンドのリストのファーストクラスコレクション。
SoundManager.cs	音源の再生、停止を担当するクラス。 シングルトンパターン。 アンチパターンとされているが、音源再生の頻度等を考えて利用。



4. 担当ソースコード

- Soundフォルダ（2/2）

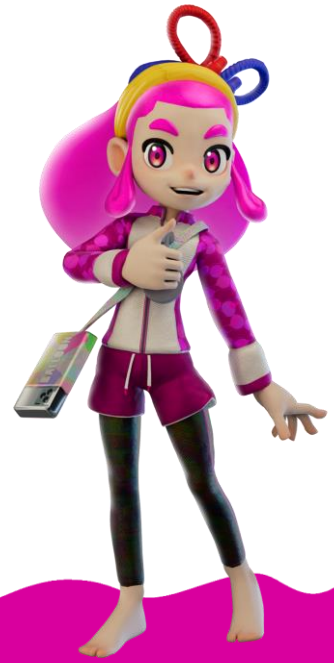
ファイル名	概要
SoundSetting.cs	SoundKVPair.csのValueとして使用しているサウンドの設定が保存されるクラス。
SoundSettingsData.cs	SoundList.csを所持するスクリプタブルオブジェクト。



4. 担当ソースコード

- Sunglassフォルダ

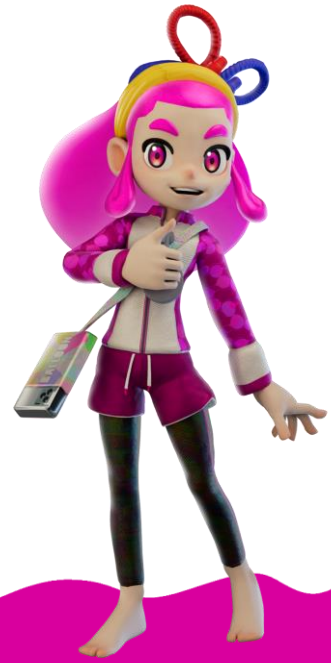
ファイル名	概要
SunglassController.cs	サングラスの処理を実装するクラス。



4. 担当ソースコード

- TimeLimitフォルダ

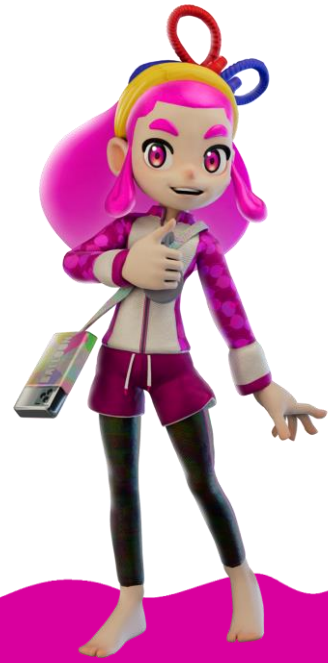
ファイル名	概要
TimeLimit.cs	制限時間の値オブジェクト。 普通は不変にするが、頻繁に書き換わる値であるためあえて可変にしている。
TimeLimitController.cs	制限時間の値オブジェクトを用い、処理を実装しているクラス。
TimeLimitView.cs	制限時間のUIの描画部分を担当するクラス。



4. 担当ソースコード

- Utilityフォルダ

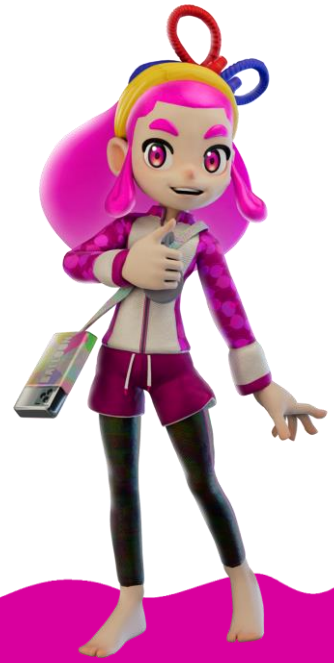
ファイル名	概要
Curve01Attribute.cs	プロパティの属性。詳細は前述のエディタ拡張を参照。 Curve01Drawer.cs
ImageUtility.cs	Imageの拡張メソッド。 よく用いられるフェードやフラッシュを実装している。
RequireAttribute.cs	Inspector上で値がnullの場合、警告を出す属性。
Vector2Utility.cs	Vector2の拡張メソッド。 入力値を4方向にスナップする機能を実装している。



4. 担当ソースコード

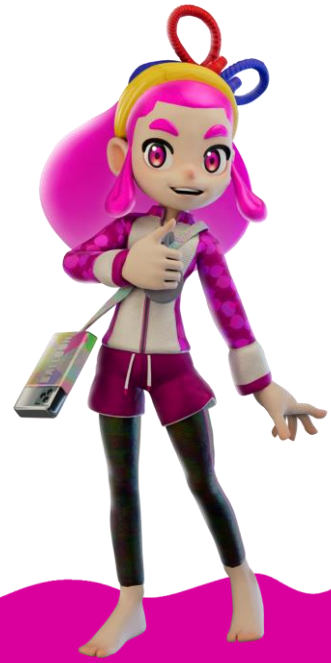
- Input

ファイル名	概要
InputSystemManager.cs	InputSystemのマップ切り替えをカプセル化したクラス。 通常のInputSystem以外にJoyconを用いたInputがあるため一括管理するために作成。



5. 工夫点

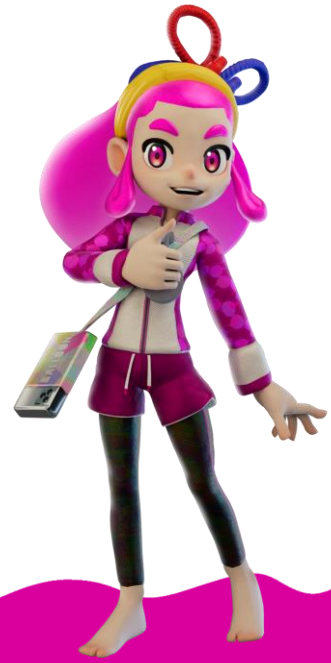
- コードレビューの徹底
- マニュアルの作成
- プレイヤーの設計
- 空気栓の設計
- ロードの時間を極力減らす
- 隠し要素



5. 工夫点

時間はかかるけど…
その分、同規模のプロジェクトより
バグ報告が半分に！（33件→15件）

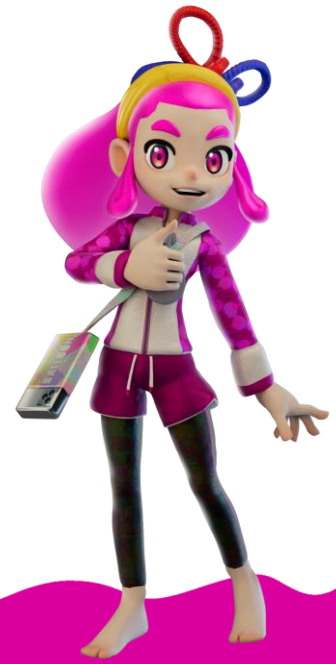
- コードレビューの徹底
  - バグの検出がしやすい
    - バグの見逃しを第三者がコードを見ることで減らせる。
  - 仕様のすり合わせができる
    - コードレビューの段階で確認することで、勘違いを防げる。
  - ソースコードが保守しやすくなる
    - 動きは合っているが、より軽くできる処理などを見つけられる。
    - 冗長なコードを少なくできる。
      - 将来的にバグの減少に繋がる。



5. 工夫点

マニュアルを書くことで
客観的に機能を見れて、
使いにくい部分を改善できた！

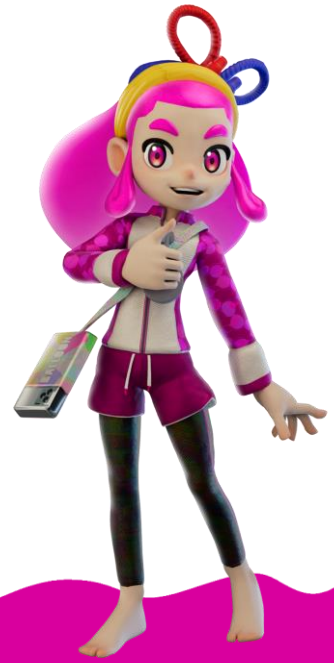
- マニュアルの作成
  - 難しい設定が必要な部分はマニュアルに
    - 一度書いておけば、この先自分が忘れても見返せる。
  - プログラム向けなら例としてソースコードも載せられる。
    - 使い方がわからないという質問を以前チームで多く受けたため、改善案としてマニュアルを作ってみたところ、同じような質問が減った。



5. 工夫点

以前は一つのスクリプトに
すべてを書いていて読みづらかった。
今回は内部処理が
ほぼ別スクリプトだからクリーン！

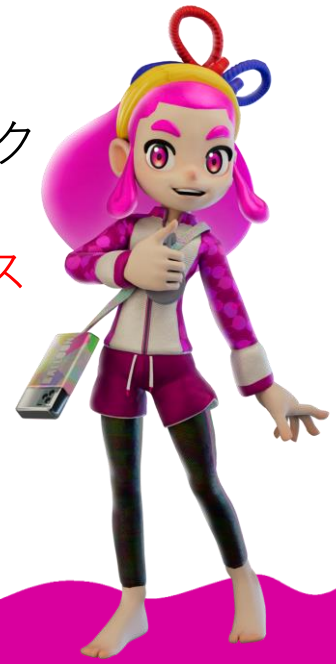
- プレイヤーの設計
  - 風船部分とプレイヤー本体で分けた
    - 風船は風船で独立している。
    - 分けることでコードの肥大化を防ぎ、可読性がアップした。
  - 風船の状態で呼ぶスクリプトを動的に変えた
    - インターフェースを継承させれば、仕様変更に対応しやすい。
  - プレイヤーと接触するインターフェースを作った
    - 以前インターンシップで教わった内容を実践してみた。
    - 相互に接触判定を行わずに済むため、担当間の受け渡しが簡単に。



5. 工夫点

チームメンバーから
とても使いやすいと評判でした！

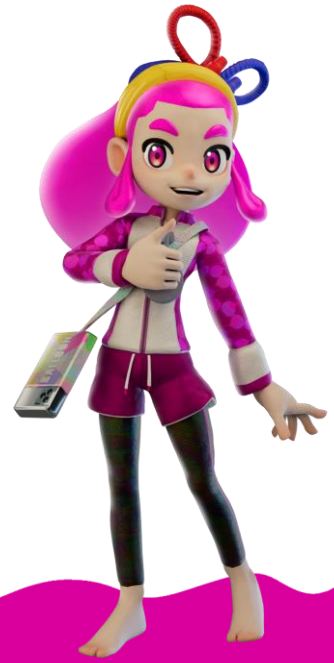
- 空気栓の設計
  - インターフェースではなく基底クラス
    - UnityはインターフェースをInspectorから設定することはできない。
    - そのため、基底クラスで管理している。
  - 処理を書いたらアタッチするだけ
    - どんなオブジェクトでも基底クラスを継承するだけで空気栓からインタラクト可能に
    - インタラクト時の処理もabstractなクラスは実装を強制させられるためミスがない。



5. 工夫点

- ロードの時間を極力減らす
  - 無駄なGetComponentを無くす
    - [SerializeField]で済むならそれで済ませる。
      - 開発の手間よりプレイヤー優先。
      - でも開発の手間も改善できる手法を考えたい。
  - 演出で誤魔化す
    - どうしてもロードなしにはできない。
    - トランジションの演出中に裏でロードするという形でカバー。

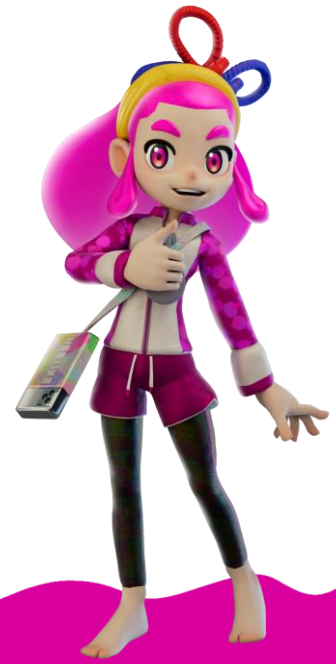
レベルデザイン班からは
「手間がかかる」と不評。
でも、ロード時間は気にならない！
改善するならワンボタンで
アタッチできる機能を作るとかですね。



5. 工夫点

自分で提案した演出が
そのまま採用されました！

- 隠し要素
  - 探す楽しさがある
    - 制限時間のあるゲームだから、演出中は制限時間を止めている。
  - 仕様になかった演出を追加した
    - 隠し要素に気づくには演出が必要だった。
    - カットシーンをいれることで、明らかな変化を演出した。



最後までご覧いただき、ありがとうございました！