作品説明書

（プログラマ）

日本電子専門学校 ゲーム制作科

ジョ　スウ　イ

徐　崇　瑋

HSU CHUNG WEI

目次

1.個人作品(メインアピール作品)
時間：2022年5月～現在
・作品概要
・説明

2.チーム作品Ⅰ
時間:2022年2月～2022年5月
・作品概要
・担当箇所説明

3.チーム作品Ⅱ
時間:2021年12月～2022年2月
・作品概要
・担当箇所説明

個人作品ー概要

github:eaffic/LightLines (github.com)
＊他のチーム作品動画はこのgiturlの動画フォルダ内です

開発概要：

・タイトル：LightLines
・開発期間：2022/05~（制作中）
・開発人数：一人
・開発エンジン、言語：Unity（2020.3.0f1）、C#
・解像度：1920*1080
・2022東京ゲームショウの出展作品に向けて制作中

ゲームコンセント：

簡単で華麗なアクションパズルゲーム

ゲーム概要：

このゲームは倉庫番の概念を基にして作りました。
プレイヤーはキャラを操作し、「プッシュ」動作を使い、
ブロックを移動させて、レーザ（光の線）をつないで
ゴールまで導くゲームです。

個人作品ー概要

現時点の実装内容：

・プレイヤー（移動、動作）
・カメラ制御
・音声制御
・エフェクト（シェーダーグラフ利用）
・ステージ、環境（背景）配置
・セーブ機能、データ管理
・UI
    ・タイトル
    ・ステージ詳細画面
    ・メニュー
・ギミック関連
    ・箱（移動、プッシュ）
    ・レーザー（反射、クリア判定）
    ・ボタン
    ・移動ブロック
    ・回転ブロック

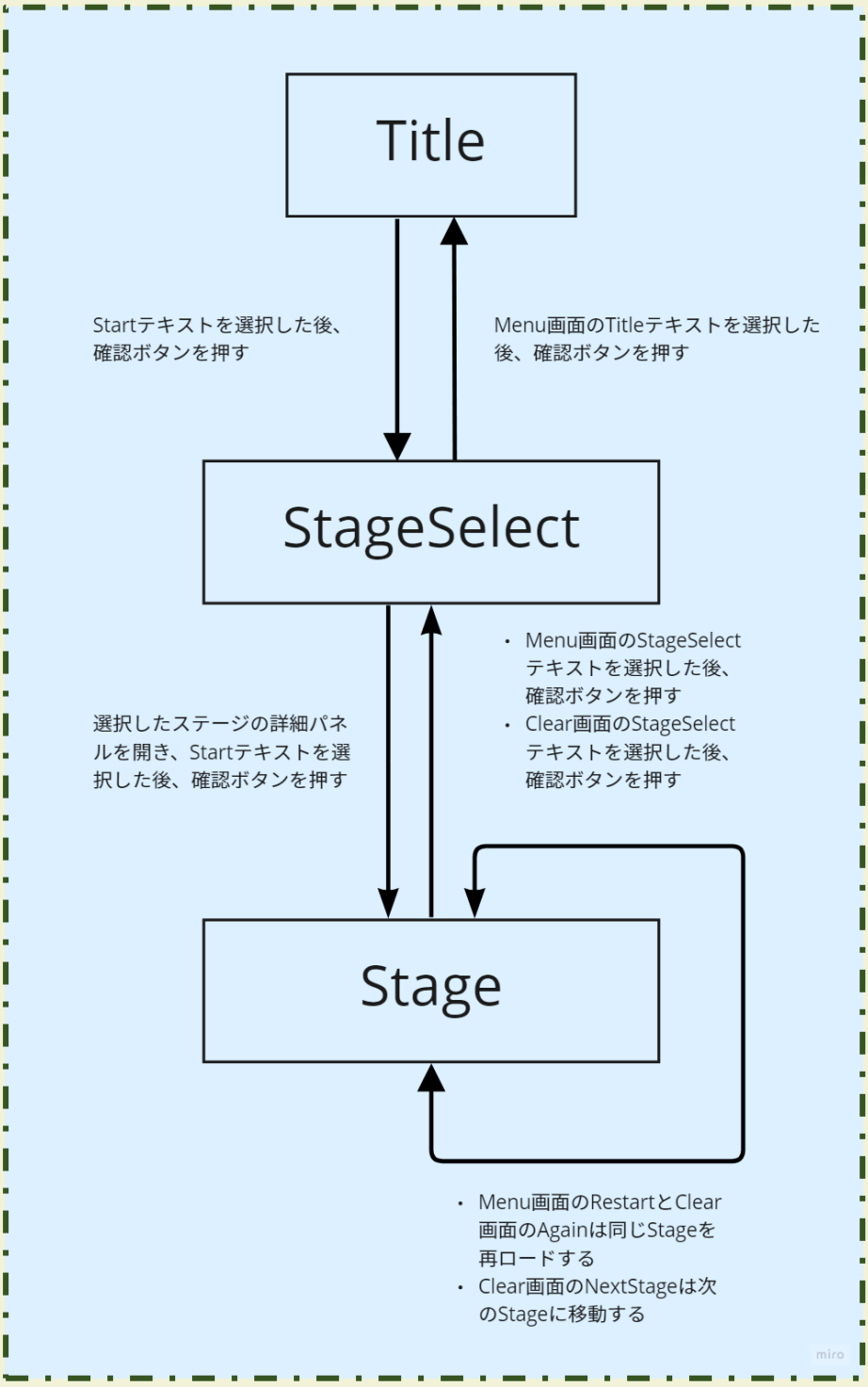
個人作品ー概要

操作方法:

個人作品ー説明

・ゲーム全体のシーン構成

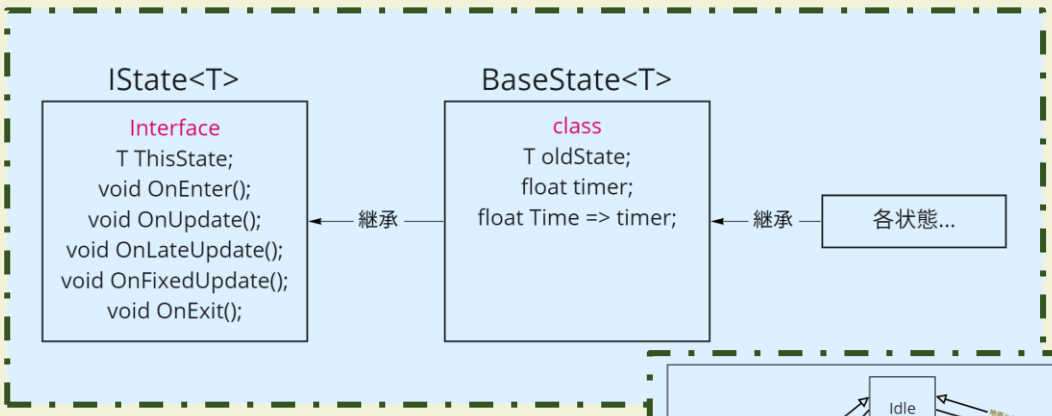
▼シーンの流れ図



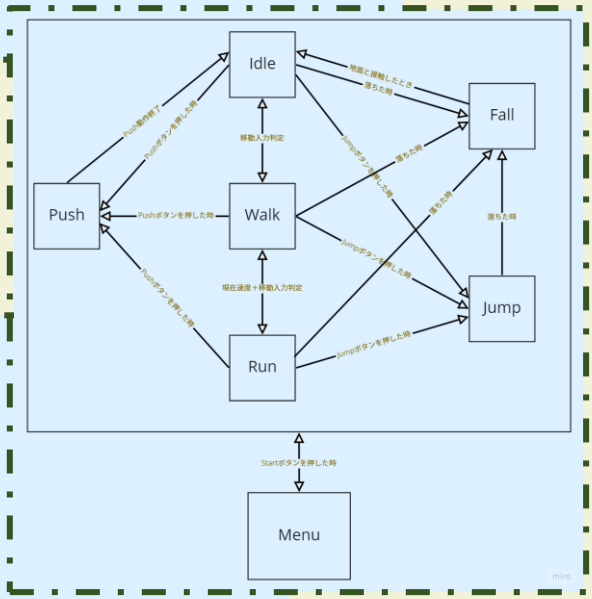
個人作品ー説明

・プレイヤー構成概要

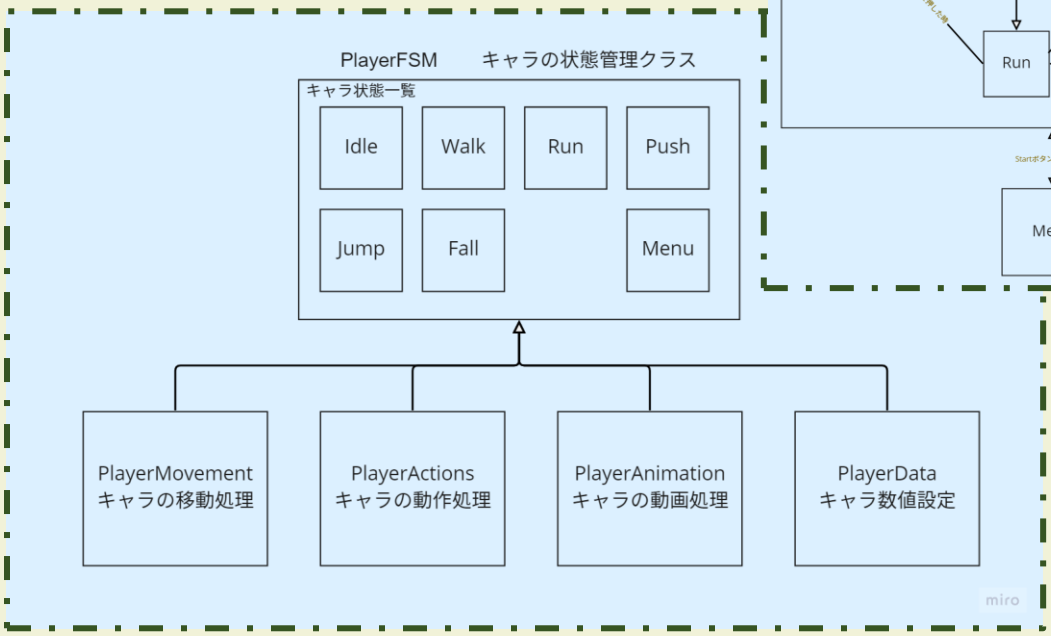
▼ステート構造



プレイヤーのステート図▶



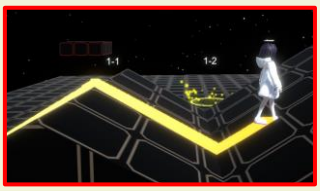
▼プレイヤーのステート構成



・プレイヤーはステートマシンを利用しています。各ステートはBaseStateクラスから継承し、管理クラスでまとめて管理しています。

個人作品ー説明

・プレイヤー構成詳細



▼キャラ状態管理クラス

```cs
public class PlayerFSM : MonoBehaviour {
    public CharacterData_SO PlayerData; //キャラデータ
    public PlayerMovement PlayerMovementController; //移動制御
    public PlayerAnimation PlayerAnimationController; //動画制御
    public PlayerActions PlayerActionsController; //動作制御

    private IState<PlayerState> _currentState;
    public PlayerState CurrentState => _currentState.ThisState;
    private Dictionary<PlayerState, IState<PlayerState>> _states = new Dictionary<PlayerState, IState<PlayerState>>();

    private void Awake() {
        PlayerData.PlayerInputSpace = Camera.main.gameObject.transform;

        TryGetComponent(out PlayerMovementController);
        TryGetComponent(out PlayerAnimationController);
        TryGetComponent(out PlayerActionsController);

        _states.Add(PlayerState.Idle, new PlayerIdleState(this, PlayerState.Idle));
        _states.Add(PlayerState.Walk, new PlayerWalkState(this, PlayerState.Walk));
        _states.Add(PlayerState.Run, new PlayerRunState(this, PlayerState.Run));
        _states.Add(PlayerState.Jump, new PlayerJumpState(this, PlayerState.Jump));
        _states.Add(PlayerState.Fall, new PlayerFallState(this, PlayerState.Fall));
        _states.Add(PlayerState.Push, new PlayerPushState(this, PlayerState.Push));
        _states.Add(PlayerState.Menu, new PlayerMenuState(this, PlayerState.Menu));

        TransitionState(default, PlayerState.Idle); //初期状態
    }

    private void Update() {
        PlayerActionsController.SearchBox();
        _currentState.OnUpdate(Time.deltaTime);
    }

    /// <summary>
    /// 状態遷移
    /// </summary>
    /// <param name="type"></param>
    public void TransitionState(PlayerState now, PlayerState next){
        if(_currentState != null){
            _currentState.OnExit();
        }
        _currentState = _states[next];
        _currentState.OnEnter(now);
    }
}
```

▼状態のデモ構成

```cs
public class PlayerIdleState : BaseState<PlayerState>
{
    private PlayerFSM _fsm;

    public PlayerIdleState(PlayerFSM manager, PlayerState state)
    {
        base.ThisState = state;
        _fsm = manager;
    }

    public override void OnEnter(PlayerState oldState)
    {
        //この状態に入った時
    }

    public override void OnUpdate(float deltaTime)
    {
        //状態移行確認
    }

    public override void OnExit()
    {
        //この状態から離脱した時
    }
}
```

↑ステートクラスはMonoBehaviorを継承していないため、キャラに直接な影響は出ません。各ステートはゲームの状態（ポーズ…）、入力（ジャンプ…）、キャラの数値（速度…）から判断し、次の状態を決まります。

↑キャラの状態とキャラの表現制御をまとめて管理するクラス。現在ステートのUpdateを呼び出します。

▼キャラ移動クラス（切り抜き）

```cs
/// <summary>
/// 移動制御クラス
/// </summary>
public class PlayerMovement : MonoBehaviour
{
    private void Update()
    {
        switch (_fsm.CurrentState)
        {
            case PlayerState.Walk:
            case PlayerState.Run:
            case PlayerState.Jump:
            case PlayerState.Fall:
                InputCheck();
                break;
            default:
                break;
        }
    }

    private void FixedUpdate()
    {
        switch (_fsm.CurrentState)
        {
            case PlayerState.Idle:
            case PlayerState.Walk:
            case PlayerState.Run:
            case PlayerState.Jump:
            case PlayerState.Fall:
                _velocity = _rigidBody.velocity;
                SnapToGround();
                AdjustVelocity();
                ClearState();
                _rigidBody.velocity = _velocity;
                break;
            default:
                break;
        }
        _fsm.PlayerData.Velocity = _rigidBody.velocity;
        CheckGravity();
    }
```

←キャラの移動はUnityの物理エンジン（RigidBody）を使用しています。
キャラの状態によって数値（最大速度、加速度…）を更新する。

↓キャラの数値クラス、各ステートの数値は細かく設定しています。

```cs
public class CharacterData_SO : ScriptableObject {
    //......
    [Header("MaxSpeed")]
    [Tooltip("最大地面歩行速度"), SerializeField, Range(0f, 100f)]
    private float _maxWalkSpeed = 5f;
    public float MaxWalkSpeed => _maxWalkSpeed;
    //......

    [Header("Accleration")]
    [Tooltip("歩行加速度"), SerializeField, Range(0f, 100f)]
    private float _walkAcceleration = 2f;
    public float WalkAcceleration => _walkAcceleration;
    //......

    [Header("Jump")]
    [Tooltip("最大ジャンプ高さ"), SerializeField, Range(0f, 10f)]
    private float _jumpHeight = 2f;
    //......
}
```

個人作品ー説明

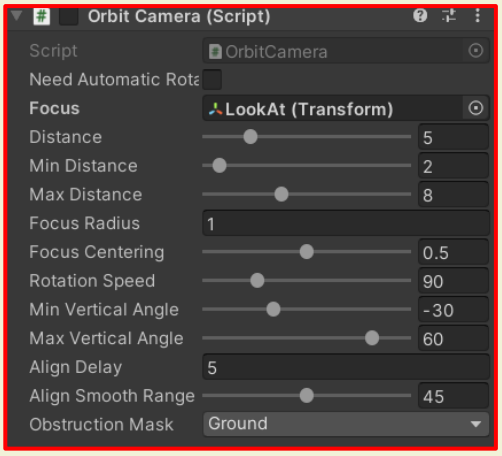
・カメラ構成

▼カメラの数値設定

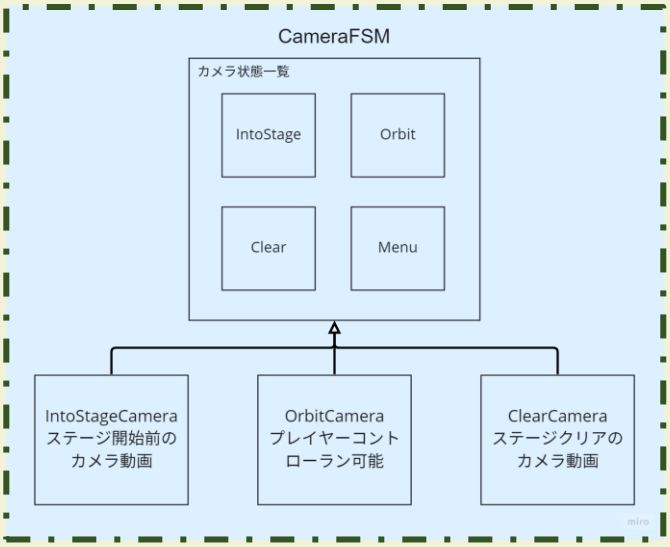
```
01. [Tooltip("注視点"), SerializeField] Transform _focus = default;
02. [Tooltip("距離"), SerializeField, Range(1f, 20f)] float _distance = 5f;
03. [Tooltip("最小距離"), SerializeField, Range(1f, 20f)] float _minDistance = 2f;
04. [Tooltip("最大距離"), SerializeField, Range(1f, 20f)] float _maxDistance = 8f;
05. [Tooltip("注視点半径"), SerializeField, Min(0f)] float _focusRadius = 1f;
06. [Tooltip("注視点にそろえる速度"), SerializeField, Range(0f, 1f)] float _focusCentering = 0.5f;
07. [Tooltip("回転速度/秒"), SerializeField, Range(1f, 360f)] float _rotationSpeed = 90f;
08. [Tooltip("垂直回転最小角度"), SerializeField, Range(-89f, 89f)] float _minVerticalAngle = -30f;
09. [Tooltip("垂直回転最大角度"), SerializeField, Range(-89f, 89f)] float _maxVerticalAngle = 60f;
10. [Tooltip("自動追尾遅延時間"), SerializeField, Min(0f)] float _alignDelay = 5f;
11. [Tooltip("自動追尾速度"), SerializeField, Range(0f, 90f)] float _alignSmoothRange = 45f;
12. [Tooltip("障害物Layer"), SerializeField] LayerMask _obstructionMask = -1;
```

・角度、距離、回転速度などの操作制限は細かく設定可能です。
・障害物の判定注視目標からカメラの方向にcubeレイを使って探測して、目標をはっきり見えるために位置を調整する
・一定の時間未操作の場合、自動追跡、転向になります。追跡と転向の速度は目標の速度の繋がっています。

▼インスペクターの表示



▼ステージ内のカメラ構成（実装予定）

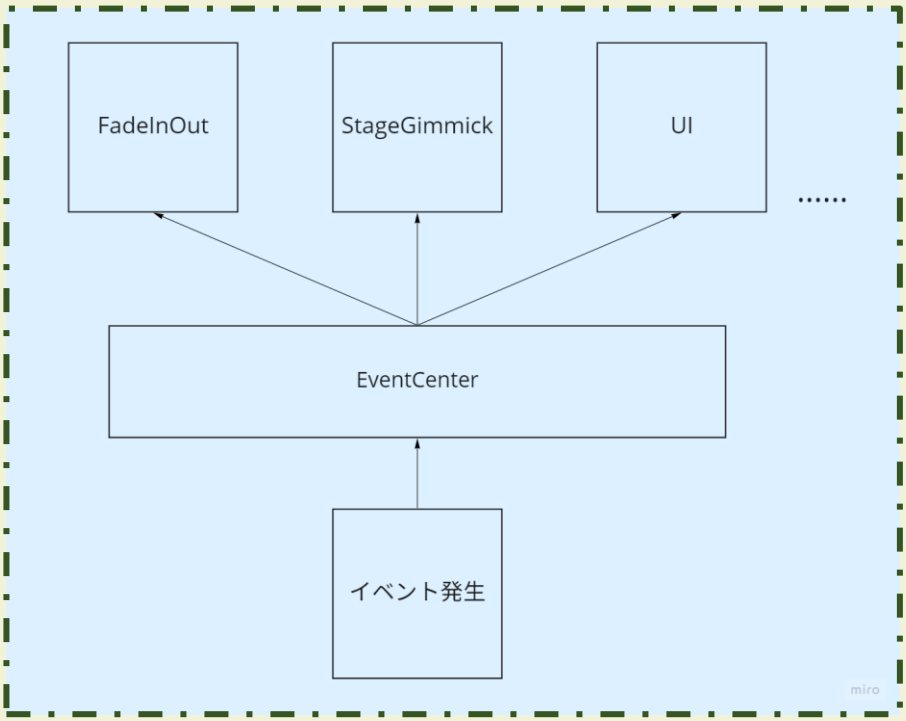


（実装予定）
・ステートマシンを利用して、ステージ開始とクリア時のカメラ動画を実装する

個人作品ー説明

・イベントセンター

▼イベントセンターの構成



・イベントの観察者（ボタンに反応しているギミックなど…）はイベントセンターから提供されたActionに登録する。特定の事件が発生した時、事件の発生側（ボタンなど…）からイベントセンターを呼び出し、観察者に伝える。
・事件の発生側と観察側の直接の繋がりをなくして、管理しやすくなる。

▼イベントセンターの構成

```
01. public class EventCenter
02. {
03.     public static bool Enabled = true;
04. 
05.     //クリア目標リスト
06.     private static List<IStageGimmick> _laserTargetList = new List<IStageGimmick>(16);
07. 
08.     #region 委託リスト
09.     private static List<Action> _stageSecretItemObserver = new List<Action>(8); //ステージ内の隠しアイテム
10.     private static List<Action<string>> _uiObserver = new List<Action<string>>(8); //ui
11.     private static List<Action<bool>> _stageClearObserver = new List<Action<bool>>(8); //クリア時
12.     private static List<Action<int, bool>> _buttonObserver = new List<Action<int, bool>>(256); //サイズは先に決まります
13.     private static List<Action<SceneType>> _fadeInOutObserver = new List<Action<SceneType>>(8); //フェード効果
14.     private static List<Action<SceneType>> _stageInfomationObserver = new List<Action<SceneType>>(8); //ステージ情報
15.     #endregion
16. 
17.     #region ボタンと連動するギミック
18.     /// <summary>
19.     /// ギミック登録
20.     /// </summary>
21.     /// <param name="action"></param>
22.     public static void AddButtonListener(Action<int, bool> action)
23.     {
24.         if (!_buttonObserver.Contains(action))
25.         {
26.             _buttonObserver.Add(action);
27.         }
28.     }
29.     #endregion
30. 
31.     #region フェードインアウト効果
32.     //...
33.     #endregion
34. 
35.     #region ステージクリア確認
36.     //...
37.     #endregion
38. 
39.     //....
40. }
```

```
01. EventCenter.ButtonNotify(ID, IsOpen);
02. EventCenter.FadeNotify(SceneType.StageSelect);
03. EventCenter.UINotify("Menu");
```

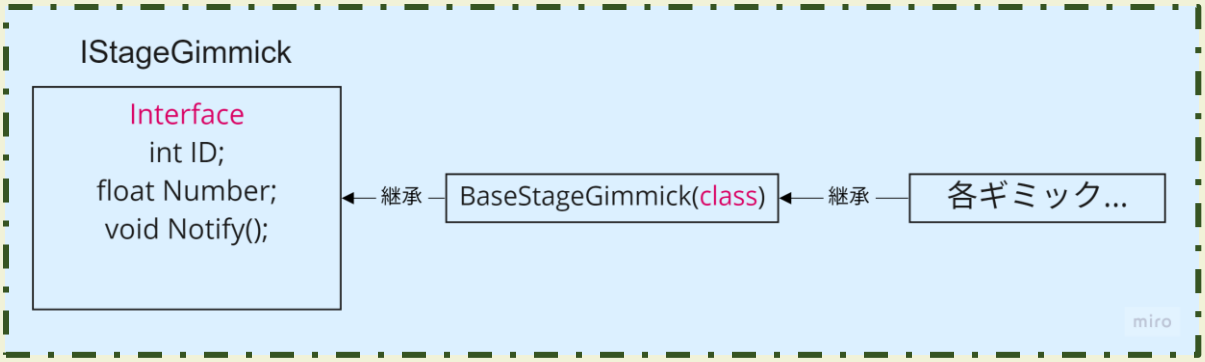
↑呼び出し例

←イベントセンターはGameObjectにアタッチする必要はありません。

個人作品ー説明

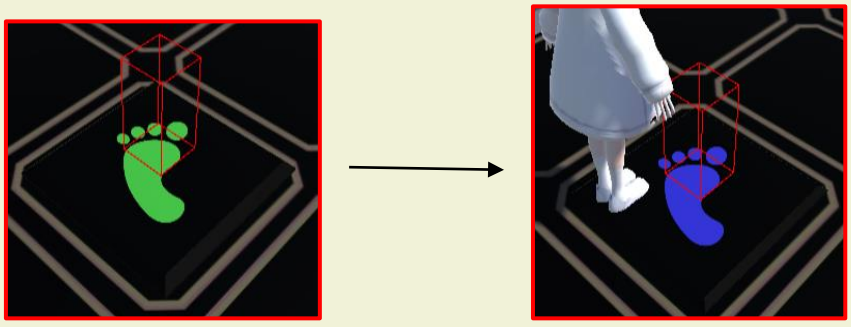
・ステージギミック

▼ギミック構成

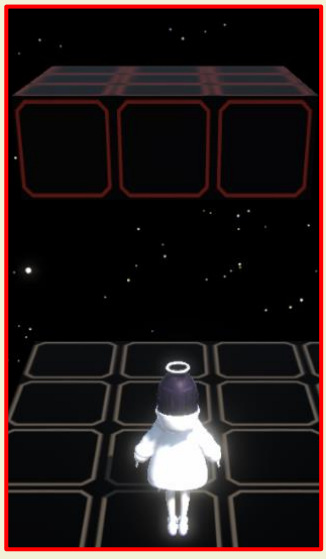


・他のギミックと連動する必要があるのギミックはBaseStageGimmickを継承する

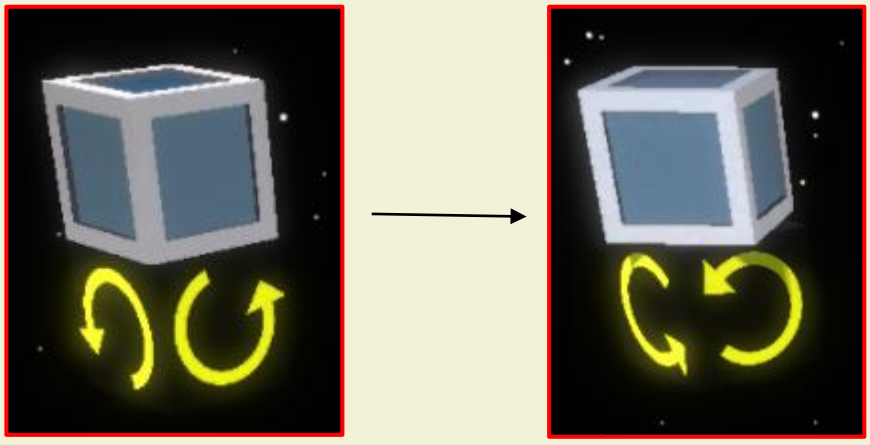
▼ギミックーーボタン



▼ギミックーー移動ブロック



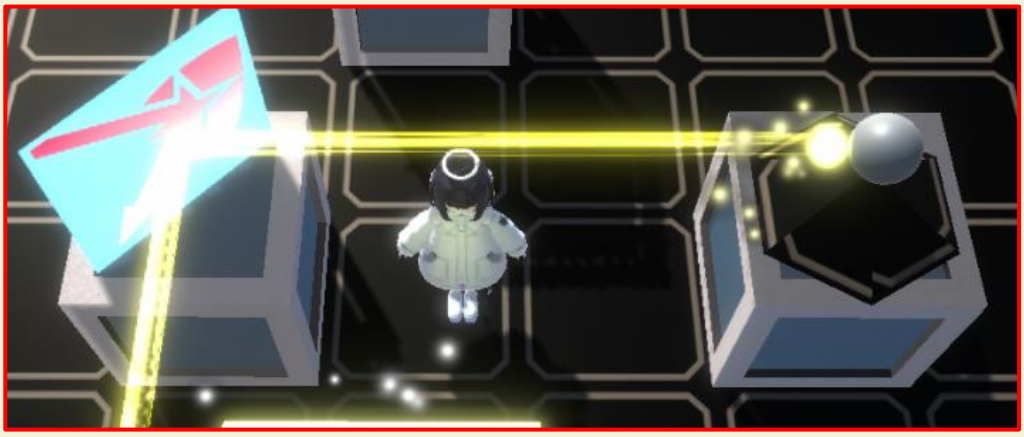
▼ギミックーー回転ブロック



個人作品ー説明

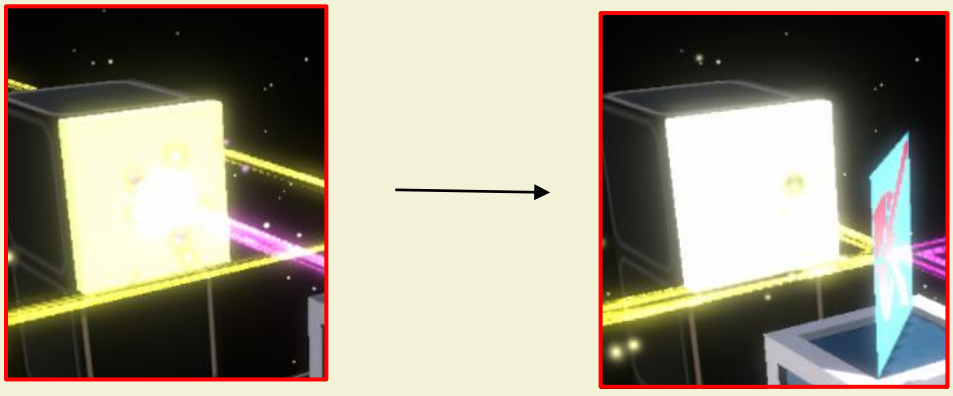
・ステージギミック

▼レーザーとミラー

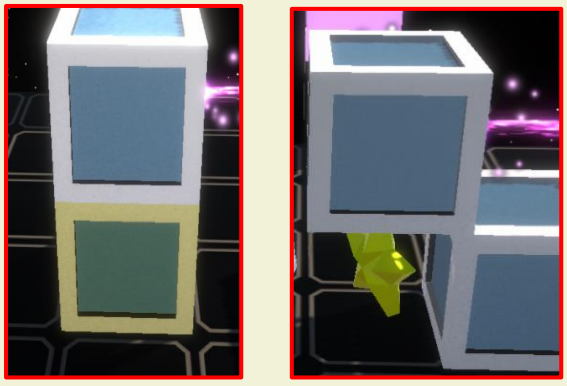


・レーザーはミラーに当たった時反射する

▼レーザーターゲット



・ターゲットは指定のレーザーと同じ色を持っている、状態によって色を変更して、起動しているかどうか確認やすくなります。



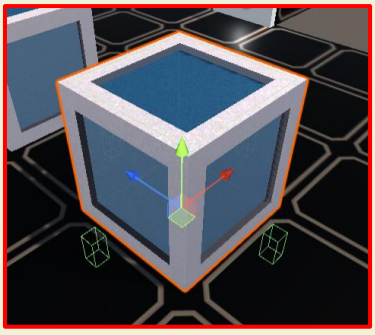
◀ステージ内の取集アイテム

・アイテムは箱の中に隠している場合、箱は黄色の点滅光があります。

個人作品ー説明

・箱構成

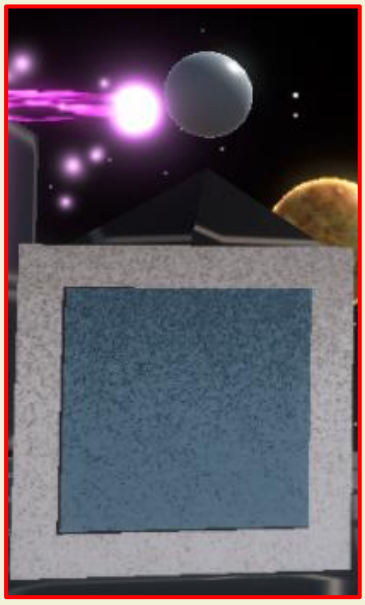
▼箱



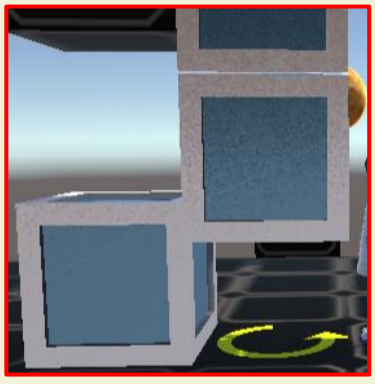
・「プッシュ」の条件
1. 緑判定エリアに立っていること
2. プレイヤー前方のレイが箱にヒットすること
・プッシュ可能の状態に入ると、箱は赤色の点滅光があります。
・箱の中にアイテムが隠している場合、箱は黄色の点滅光があります。

▼プッシュ可能状態

▼箱と他のギミックの結合

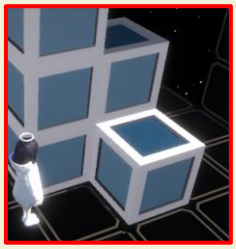


・他のギミックは箱の上に乗ると、箱と一緒に移動できます。



◀箱は重なっていても、下の箱は移動できます（だるま落とし）。この場合、上の箱は落ちます。

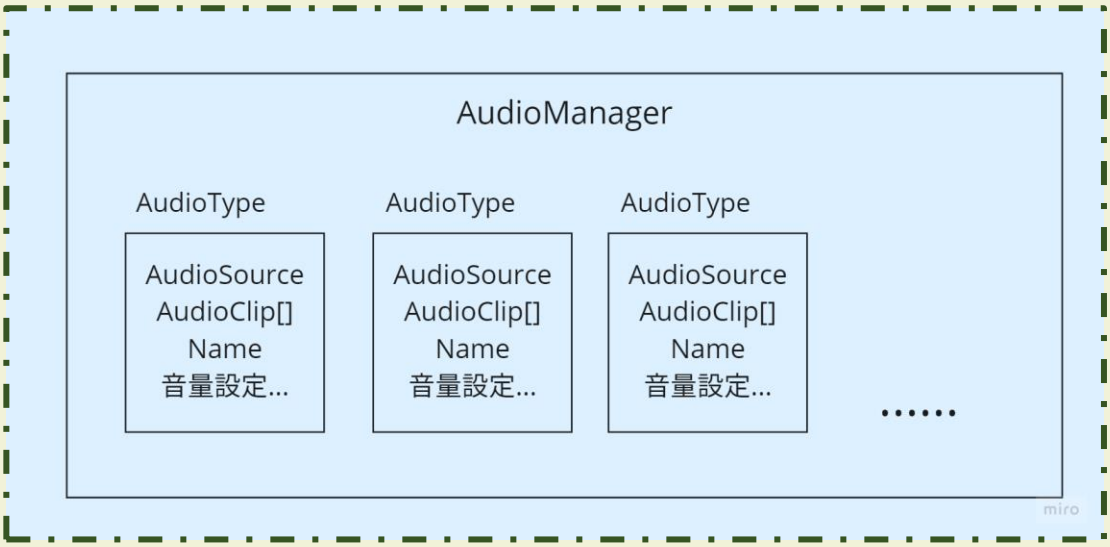
階段のような構造ができます▶

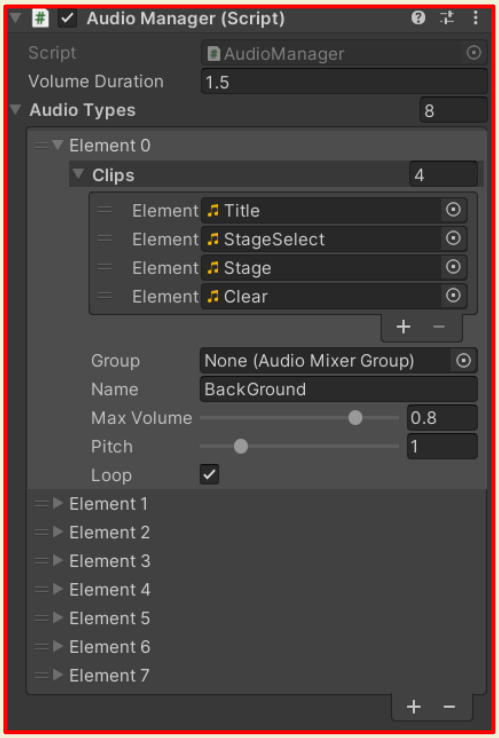


個人作品ー説明

・音声制御

▼AudioManager構成





▲インスペクターの表示

・複数音源をまとめて管理しています。
・一つの音源は複数のclipを持つことが可能です。
・同じタイプのclipは同じ音源を使用可能、
音源の総数を減少する
・このゲームのステージ範囲は制限
されているので、距離の減衰はありません。

▼使用例

```
01. AudioManager.Instance.Play("UI", "Select", false);
02. AudioManager.Instance.Play("BackGround", "StageSelect", true);
```

音源管理のクラス単位▶

```
01. [System.Serializable]
02. public class AudioType {
03.     [HideInInspector]
04.     public AudioSource Source;
05.     public AudioClip[] Clips;
06.     public AudioMixerGroup Group;
07.
08.     public string Name;
09.
10.     [Range(0f, 1f)]
11.     public float MaxVolume;
12.     [Range(0.1f, 5f)]
13.     public float Pitch;
14.     public bool Loop;
15. }
```

個人作品ー説明

・参考サイト

・プレイヤー、カメラ制御参考
Unity Movement Tutorials (catlikecoding.com)

・魔法陣参考
Unity Learning Materials (unity3d.jp)

・レーザーエフェクト参考
How To Create and Shoot a 2D Laser in Unity – YouTube

・フェード効果参考
Scene Transitions Shader in Unity Tutorial - YouTube

・素材元

・モデル（キャラ、背景など）
Muryotaisu | Characters | Unity Asset Store
Sci-Fi Styled Modular Pack | 3D Sci-Fi | Unity Asset Store
Low Poly Space Rocks | 3D Sci-Fi | Unity Asset Store
Simple Gems Ultimate Animated Customizable Pack | 3D Props | Unity Asset Store

・音源（BGM、SE）
© Unity Technologies Japan/UCL
HURT RECORD：ハートレコード
DOVA-SYNDROME
ポケットサウンド
キラキラ効果音工房

チーム作品Ｉ－概要

github:21CI0521shimokawa/GAMETAISYOU (github.com)

開発概要：

Labolimes
ラボライムズ

・タイトル：Labolimes
・メンバー数：5人
・開発期間：2022/03~2022/05（3か月）
・開発エンジン、言語：Unity（2020.3.0f1）、C#
・解像度：1920*1080
・2022ゲーム大賞アマチュア部門一次選考通過

ゲームコンセント：

＊この作品は2022ゲーム大賞 アマチュア部門の応募作品です。

感触（ゲーム大賞テーマ）

ゲーム概要：

このゲームはスライムを操る2Dアクションゲームです。
スライムをはじいて障害物を乗り越えたり、スライムをちぎって小さくなることで、狭い通路を通ることが出来るようになったりとスライムの弾力や重さ、そしてコントローラーやスティックの触感を意識して攻略していくゲームに仕上がりました。
スライムやコントローラーの『感触』を楽しんでください。

チーム作品Ｉ－概要

・担当箇所

・プログラム部分：
　・ギミック作成
　（レーザー、ボタン、ドアー、トランポリン）
　・オープニング演出
　・エンディング演出
・イラスト部分：
　・スライム（アイドル、移動、落下、はじく、ちぎる動画作成）
　・チュートリアル動画（移動、はじく、ちぎる）
　・タイトルロゴ、ロゴ動画
　・ギミック動画
　（レーザー、ボタン、ドアー、トランポリン、ベルトコンペア、台ばかり）
　・アイテム（饅頭）
　・ステージ背景（影部分のみ）

・技術アピールポイント：
　・ギミックは各自の番号を持ち、同じ番号を持っているギミックに
　影響を与える（ボタンとドアーの関係）
　・ギミックは管理オブジェクトに登録して、管理オブジェクトから
　提供されたメソッドから対応番号のギミックを起動する
　・オープニングとエンディング演出はTimelineを利用して
　動画の管理を行います

・操作方法

・ゲームパッドのみ（振動あり）
　左右移動：Lスティック
　はじく：Lスティックを押し込む->倒す->離す
　　（倒す方向と同じ方向にはじく）
　ちぎる：
　　1.LT,RTを押し込む
　　2.Lスティックを左へ、Rスティックを右へ倒す

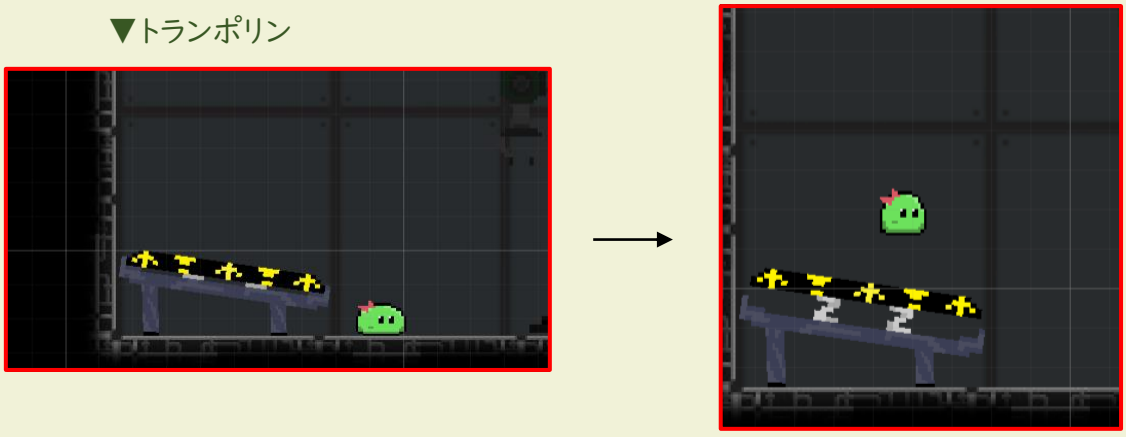
チーム作品Ｉ－説明

・ステージギミック

▼レーザー

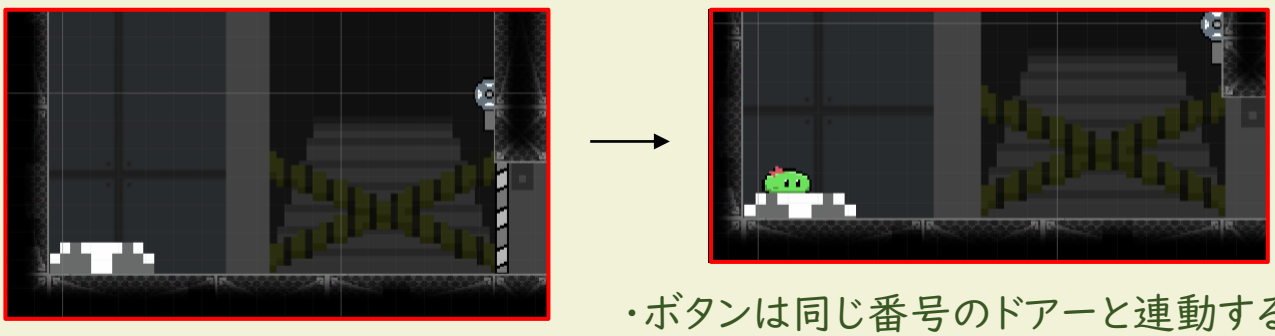
・レーザーは瓦礫を分解する

▼トランポリン



・アイテムはトランポリンの上方向に飛びます

▼ボタンとドアー

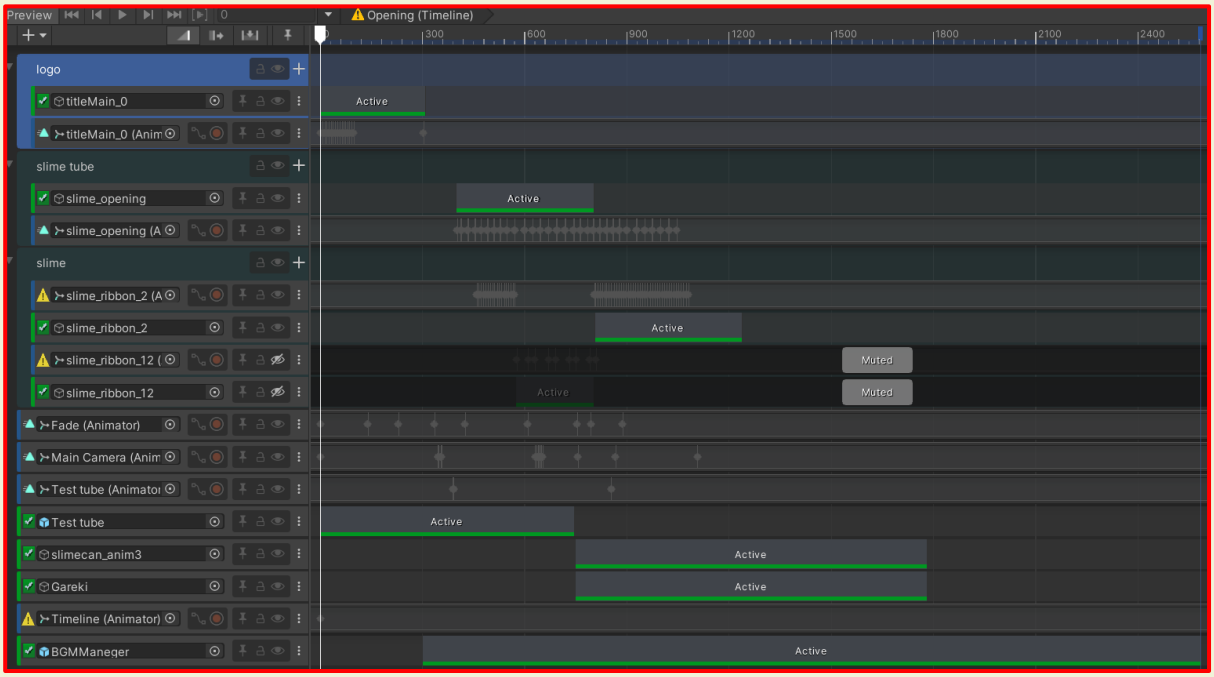


・ボタンは同じ番号のドアーと連動する

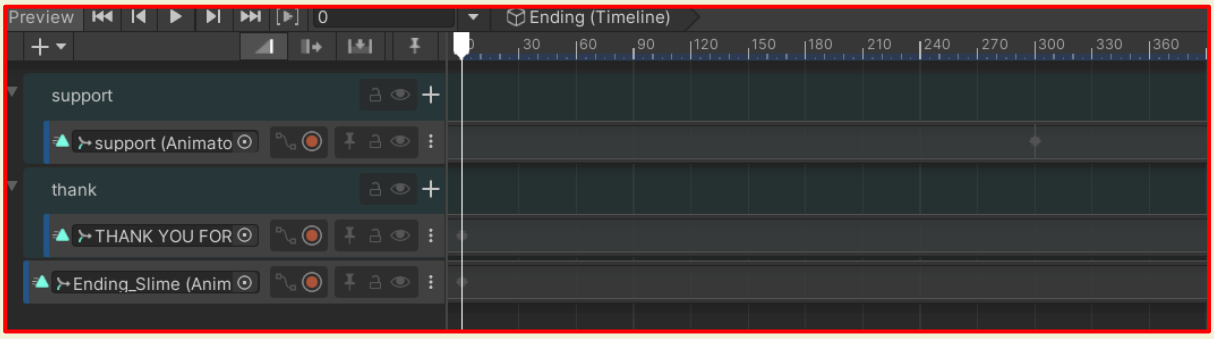
チーム作品Ｉ－説明

・オープニング、エンディング演出

▼オープニングタイムライン



▼エンディングタイムライン

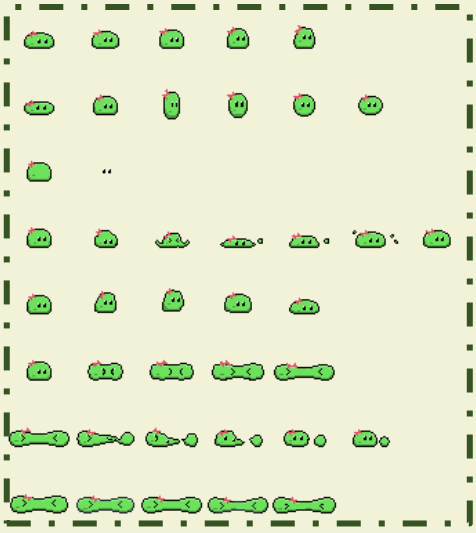


・タイムラインコンポーネントを利用して、動画演出を作成しました。
・音声はアニメションイベントで呼び出します。

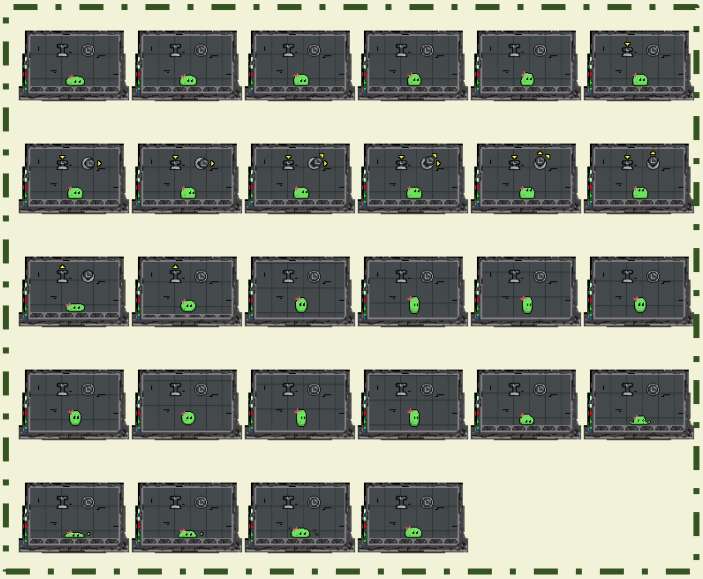
チーム作品Ⅰ－説明

・イラスト部分

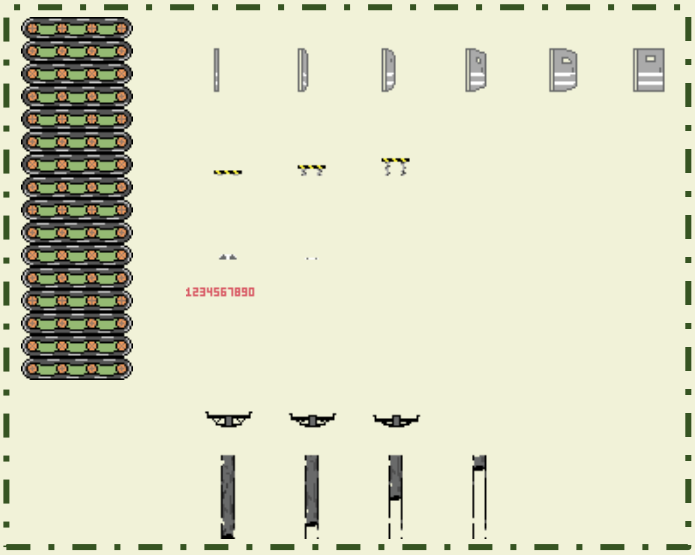
▼スライム



▼チュートリアル

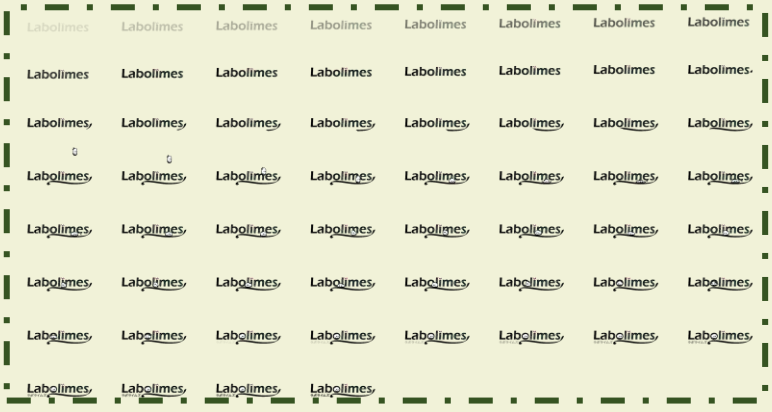


▶ステージギミック

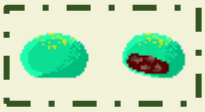


▼ボタン

▼ロゴ



▼饅頭



チーム作品Ｉ－説明

・制作感想

チームメンバー全員プログラマー志望していますが、ゲームを完成するため、プログラミング以外の仕事もやらなければならない。ですので、私ともう一人はデザイナーの仕事を受け取りました。プログラミングの部分はほぼ触っていないが、ゲームの進行、キャラのアニメション、ギミックの提案など、プログラマーの担当者と話をしました。特に素材のサイズ、アニメションの設定などの問題解決、もしくは表現方式の更新すること。スライムの色を決まるだけでもかなりの時間をがかかりました。確か不慣れの仕事について困難点がたくさんありますが、それは他のメンバーも同じですので、考え方を変えて、チームメンバーと一緒に乗り越えることを学びました。

・備考欄

・ソースコードはAsset>jyo>Stage>Scriptsのファイル内です。

チーム作品ⅡⅠ概要

github:Mekabob00/MaxiKnight (github.com)

開発概要：

マキナイト
Maximum Knight

・タイトル：マキナイト（MaxiKnight）
・メンバー数：10人
（プライナー：4人、プログラマー：3人、デザイナー：3人）
・開発期間：2021/12~2022/02（3か月）
・開発エンジン、言語：Unity（2020.3.0f1）、C#
・解像度：1920*1080
・校内進級展示会出展、プライナーの卒業展示会出展

ゲームコンセント：

邀撃ディフェンスゲーム

ゲーム概要：

ロボットを操り、拠点を防衛するアクションゲームです。
敵を倒して手に入れた資源でロボットと拠点を強化できます。

チーム作品ロー概要

・担当箇所

・最終ボス（動画6:00から出現した赤色ロボット）
・ショップ制御、数値管理（資源、強化）
・UI（HPバー、資源表示）
・拠点
・ゲームオーバー演出

・技術アピールポイント：
・ボスは列挙型で各状態の行動を制御して、流れ図と同じ動きを実現しました。
・データ管理用のオブジェクトを作り、資源を管理します。

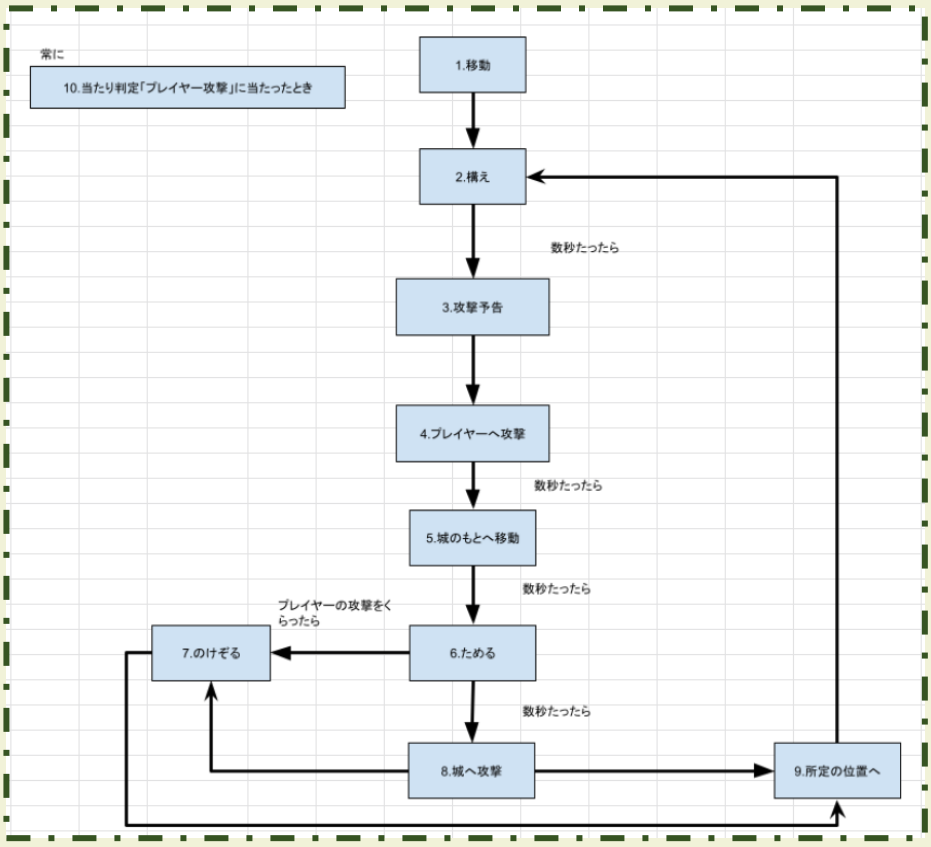
・操作方法

・キーボート+マウス
左右矢印キー：キャラ移動
上下矢印キー：レーン切り替え
スペースキー：チュートリアルをスキップ
Vキー：会話を進む
Cキー：回避
Zキー：攻撃
マウス：ショップ画面操作

チーム作品ⅡⅠ説明

・最終ボス

▼最終ボスの行動流れ(仕様)



・ボスの行動は列挙型変数でコントローラされている

▼最終ボスの状態

```
01. public enum STATE {
02.     MOVE,           //移動
03.     ENTER,          //構え
04.     STANDBY,        //攻撃予告
05.     ATTACKPLAYER,   //プレイヤーへ攻撃
06.     CHARGE,         //ためる
07.     ATTACKCASTLE,   //城へ攻撃
08.     RETURN,         //元の位置に戻る
09.     DAMAGE,         //攻撃を受ける
10.     DOWN,           //チャージ中断
11.     STANDUP,        //中断後
12.     DEAD            //死亡
13. };
```

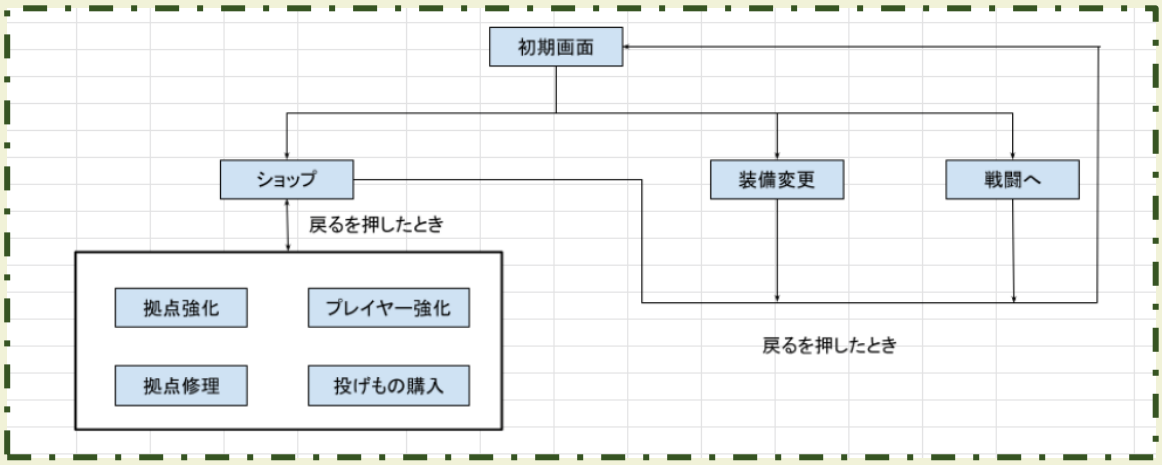
ボスの状態による更新▶

```
01. public class BossBehavior : MonoBehaviour, IPlayerDamege
02. {
03. //.......
04.     void SetState()
05.     {
06.         switch (m_state)
07.         {
08.             case STATE.MOVE:
09.                 StateUpdate_Move();
10.                 break;
11.             case STATE.ENTER:
12.                 StateUpdate_Enter();
13.                 break;
14.             case STATE.STANDBY:
15.                 StateUpdate_StandBy();
16.                 break;
17.             case STATE.ATTACKPLAYER:
18.                 StateUpdate_AttackPlayer();
19.                 break;
20.             case STATE.CHARGE:
21.                 StateUpdate_Charge();
22.                 break;
23.             case STATE.ATTACKCASTLE:
24.                 StateUpdate_AttackCastle();
25.                 break;
26.             case STATE.RETURN:
27.                 StateUpdate_Return();
28.                 break;
29.             case STATE.DAMAGE:
30.                 StateUpdate_Damage();
31.                 break;
32.             case STATE.DOWN:
33.                 StateUpdate_Down();
34.                 break;
35.             case STATE.STANDUP:
36.                 StateUpdate_StandUp();
37.                 break;
38.             case STATE.DEAD:
39.                 StateUpdate_Dead();
40.                 break;
41.         }
42. //.......
43.     }
44. }
```

チーム作品ⅡⅠ説明

・ショップ

▼ショップ構成(仕様)



・ショップはUnityのUIButtonを利用している
・新ゲームのデータはリセットするので、セーブ機能はないです

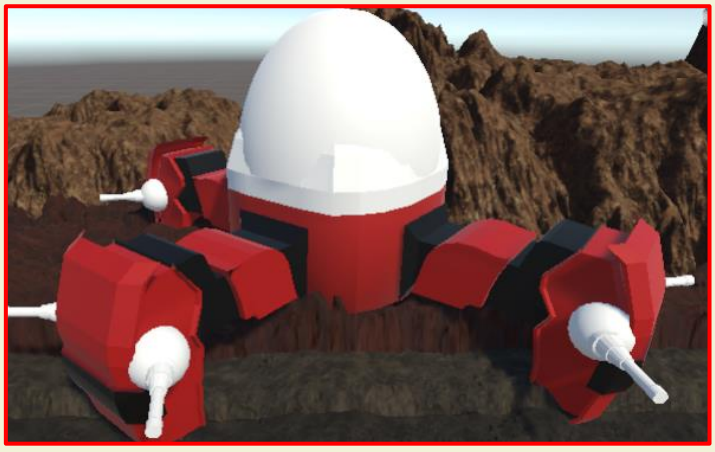
▼強化画面

▼データ管理用クラス

チーム作品ⅡⅠ説明

・拠点

▼拠点



▼拠点の攻撃

・拠点は一番近い敵に攻撃する

・UI（拠点HPバー、資源表示）

▼ステージ中

▼ショップ

チーム作品ⅡⅠ説明

・制作感想

大人数で、配れた作業を責任を持って進行する事は、今回の制作から学んだことです。仕様書通りに進行し、不明のところはプライナーと相談し、実現方法を考えながら機能を作ります。そして、デザイナーの方からもらった素材をうまく利用して、ゲームの質感を上がるために調整します。ですが、進捗やゲームの表現は先生方から色んな問題点を示し、期間内で完成できるかどうか誰も答えないです。ですので、プライナーはゲームの企画を修正して、本来キャラはフィールド内で自在に移動可能ですが、今は制限されたラインで直線移動だけになりました。城の動画や演出などもキャンセルされました。作品は完成しましたが、仕様通りに完成出来なかったことで自分の能力が足りない部分を感じました。

・備考欄

・ソースコード位置：

Assets>Boss>Scripts
Assets>Castle>Scripts
Assets>UI>Scripts
Assets>UpgradeGamen>Scripts