

Metroidvania Prototype

Zong J.C. Tian Y.C. Hu Y. Qiu Y.J.

Contents

01 Background & Significance

02 About The Project

03 Systems Explanation

04 Originality & Innovation

05 What We Learned

Background & Significance

01

1 Background & Significance

1.1 What Is Metroidvania Game

Metroidvania is a unique genre of role play game. The term is a portmanteau of the names of the video game series Metroid and Castlevania.

This type of games usually feature in:

- Various **player skills** found in game progression for player to fight with stronger enemies

- Multiple **motion abilities** aiding player with higher agility when battling, and helping player to locate secret areas and shortcuts

- Large **interconnected world** map to explore, parts of which will be inaccessible until player acquire special items or abilities

- Tight integration of **non-linear storyline**



1 Background & Significance

1.2 What Is Game Prototype

A prototype of game is a game demo, which is kind of like a implementation of “elevator pitch” of the game developer

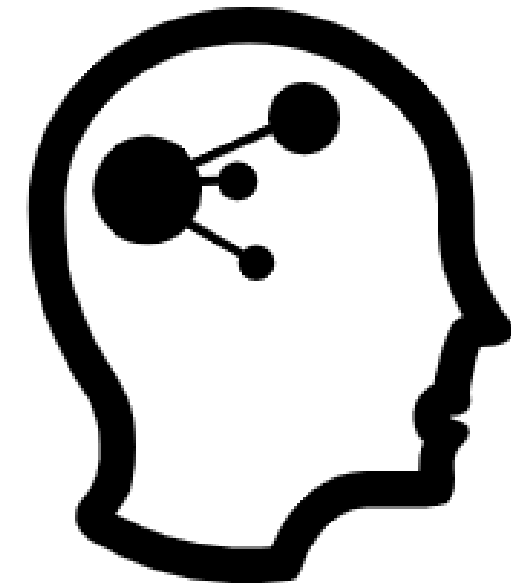
Game prototype focuses on **gameplay fundamentals**, which means art, sound and UI will need to take a back seat at first



1.3 Why This Project

I **appreciate the game genre** and I want to make my own **playful and interesting game** to convey my thoughts and aesthetic choices to the players

We can practice our **object-oriented programming skill** in this attractive topic on which we will be more self-motivated



About The Project

02

2 About The Project

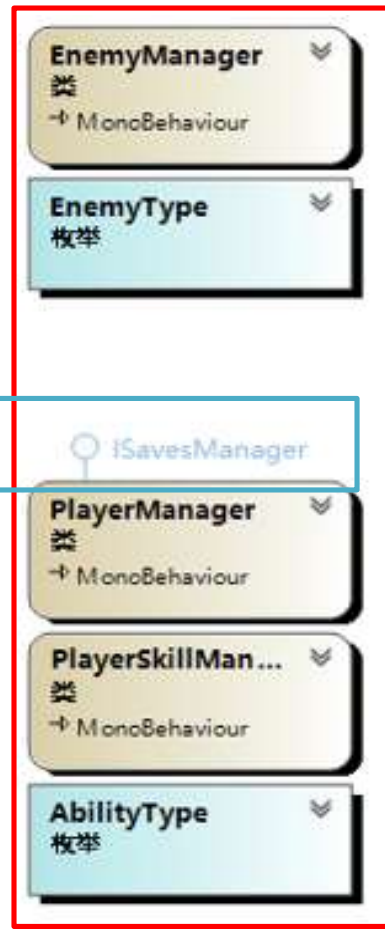
2.1 Game Impression

https://www.bilibili.com/video/BV1fM4m1171d/?share_source=copy_web&vd_source=5ef86699cafaaf10c5dc362759c73a7d

2 About The Project

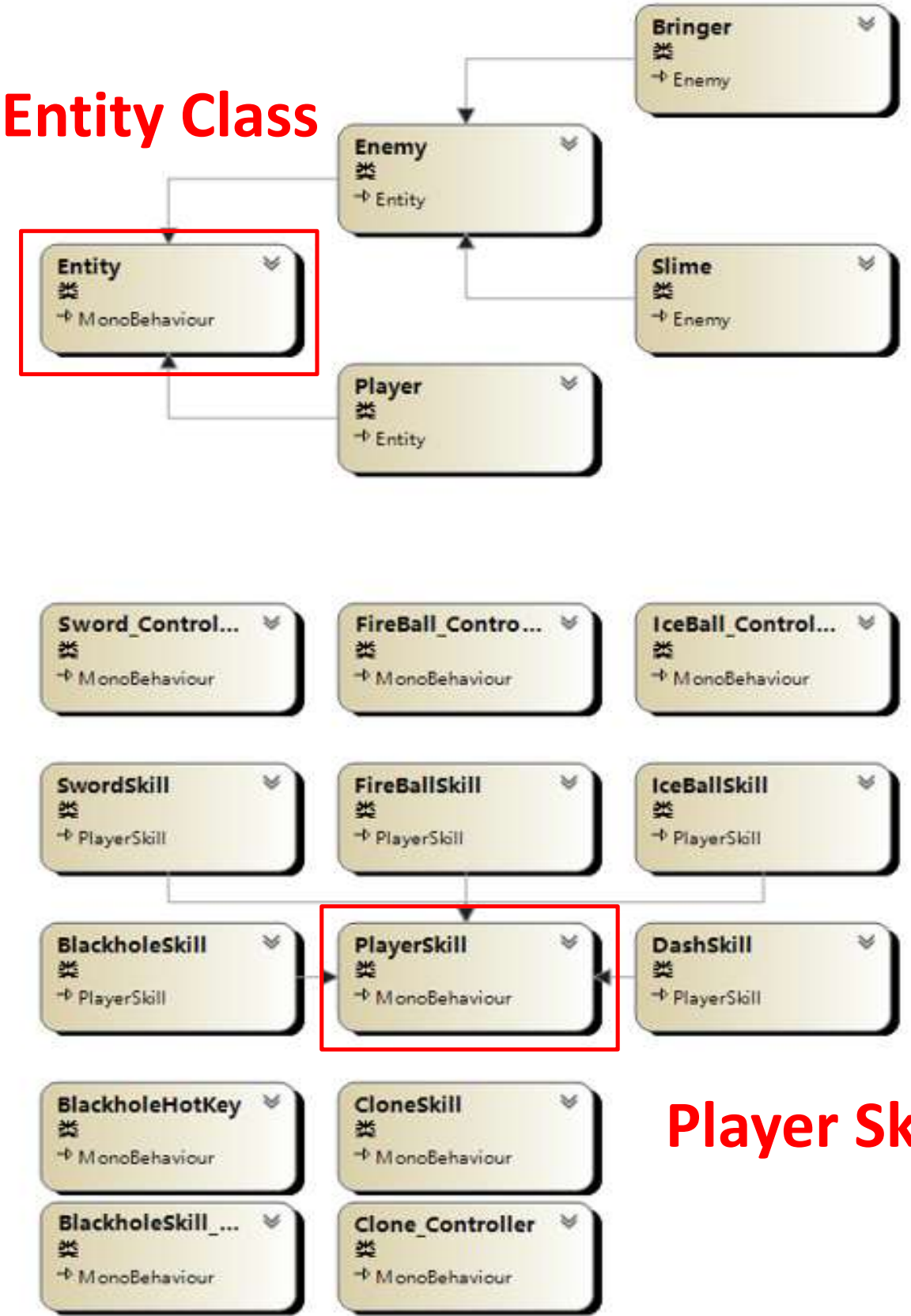
2.2 Class Diagram

Save System Interface

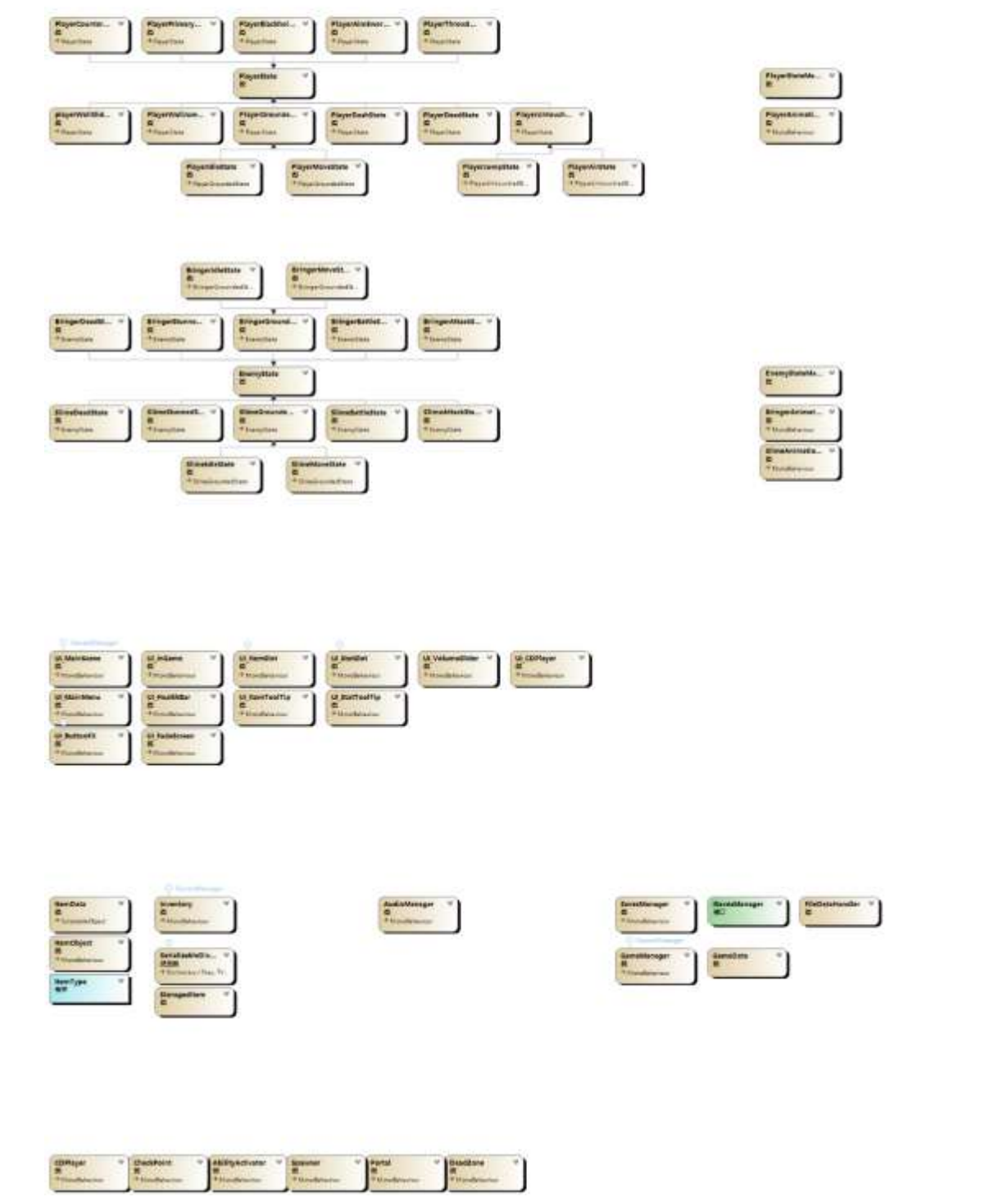
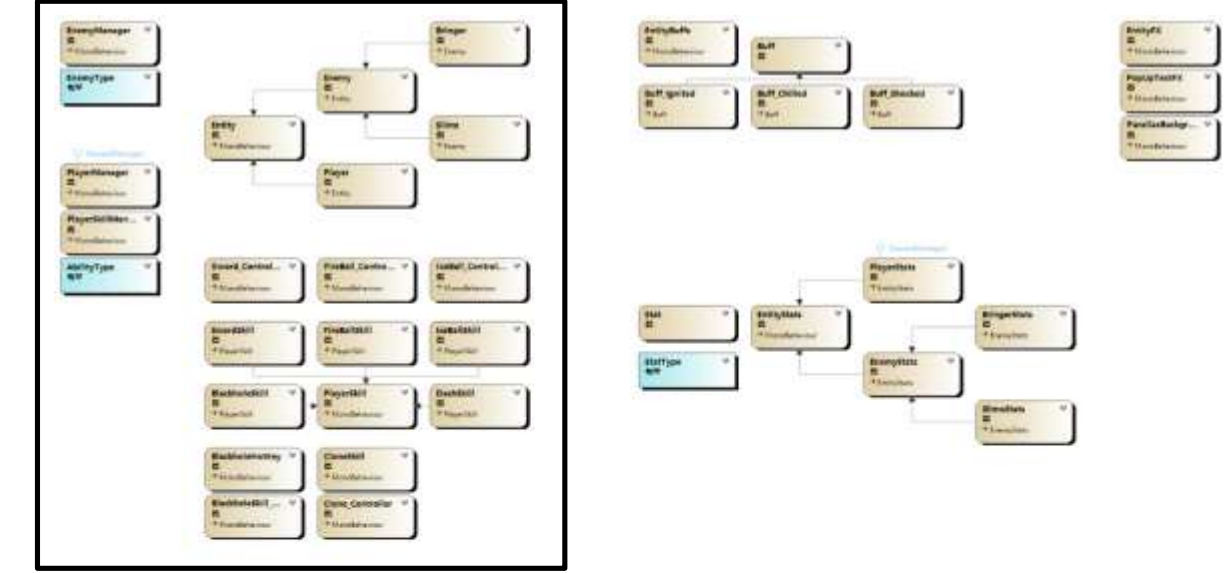


Entity Managers

Entity Class



Player Skill Class



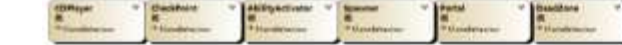
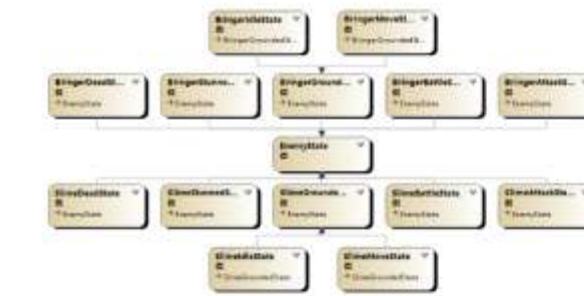
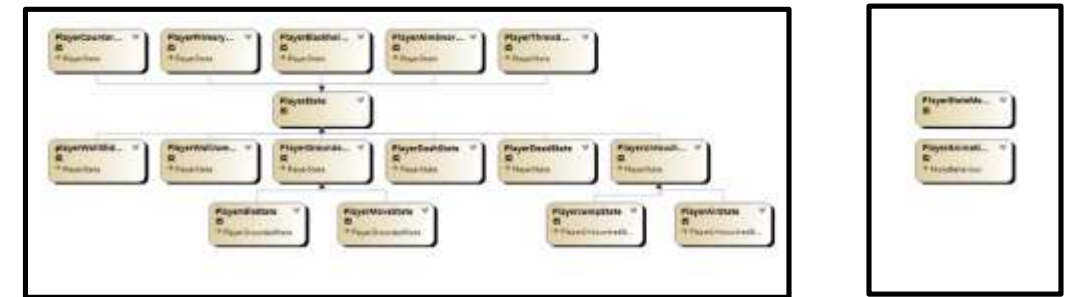
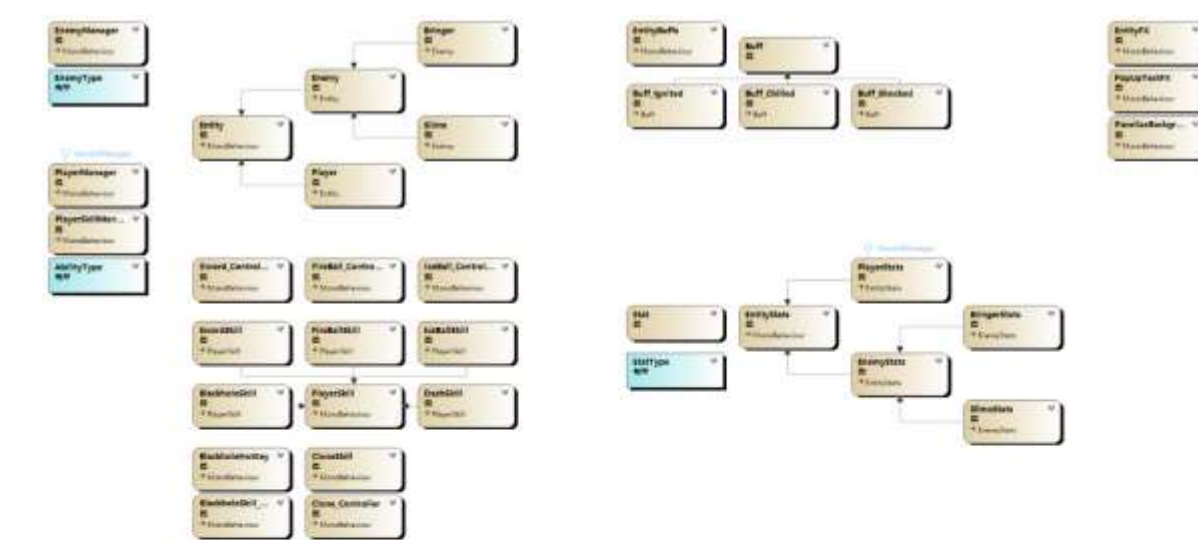
2 About The Project

2.2 Class Diagram

Player Finite State Machine (FSM)



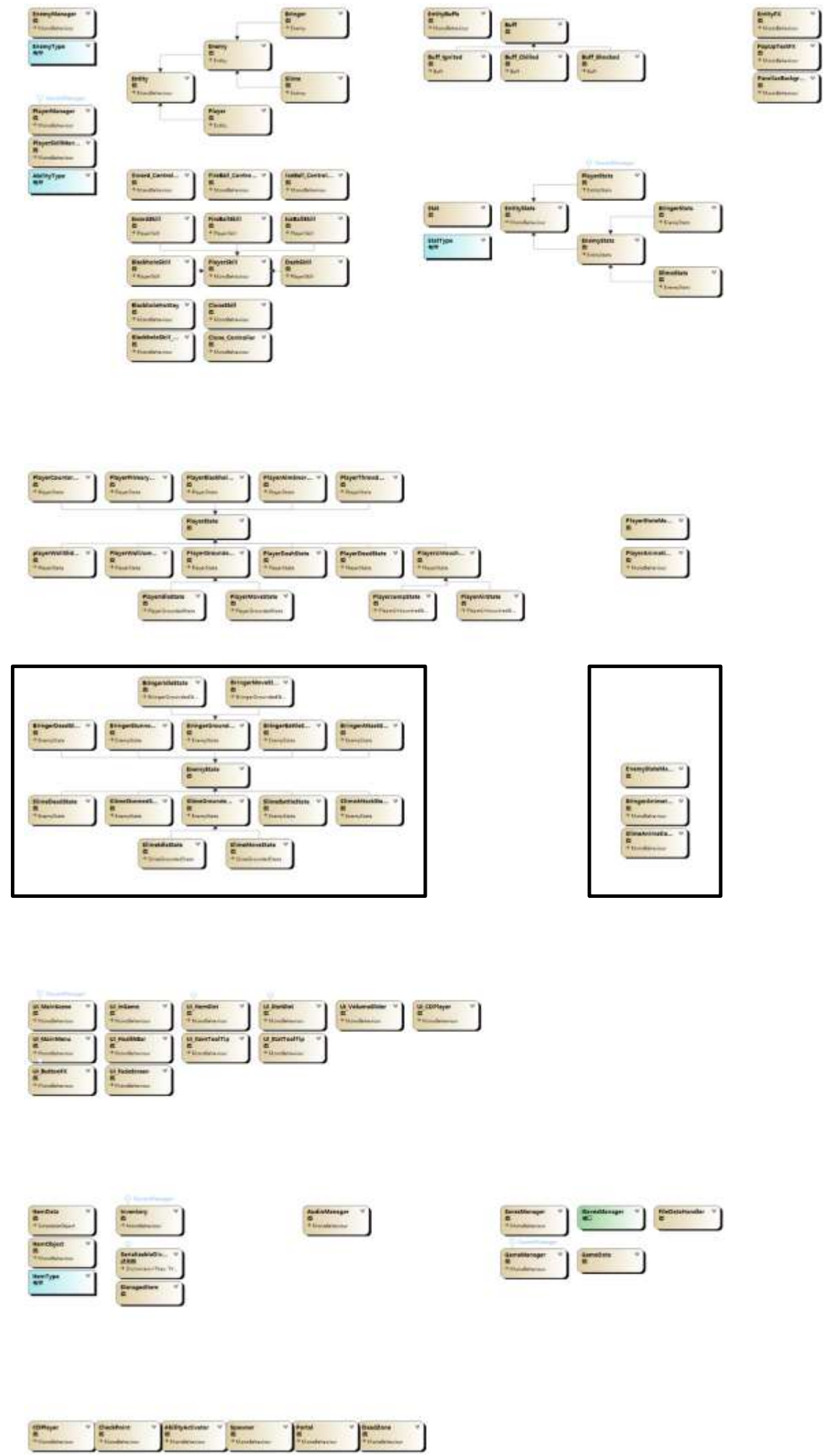
Player State Class



2 About The Project

2.2 Class Diagram

Enemy Finite State Machine (FSM)

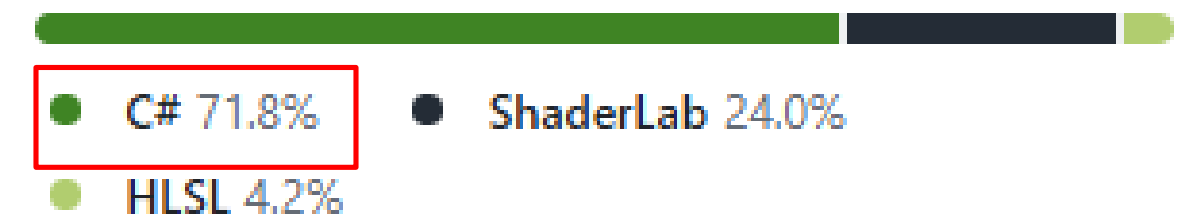


Enemy State Class

2 About The Project

2.3 Codes Work Load

Languages



I have written a large amount of comments on the codes, which causes the huge number of 7379 lines, actually the real amount of executable code lines is only 1675

代码度量值结果

筛选器: 源代码行 最小值: 200

| 层次结构 | 可维护性指数 | 圈复杂度 | 继承深度 | 类耦合度 | 源代码行 | 可执行代码行 |
|-------------------------|--------|-------|------|------|-------|--------|
| Assembly-CSharp (Debug) | 80 | 1,054 | 7 | 163 | 7,379 | 1,675 |
| UI_MainScene | 74 | 48 | 5 | 24 | 269 | 58 |
| Player | 87 | 75 | 6 | 33 | 308 | 62 |
| Inventory | 72 | 29 | 5 | 21 | 211 | 48 |
| EntityStats | 68 | 51 | 5 | 16 | 407 | 117 |
| EntityBufs | 62 | 19 | 5 | 11 | 201 | 56 |
| Entity | 80 | 39 | 5 | 18 | 241 | 45 |
| Enemy | 81 | 25 | 6 | 20 | 234 | 47 |

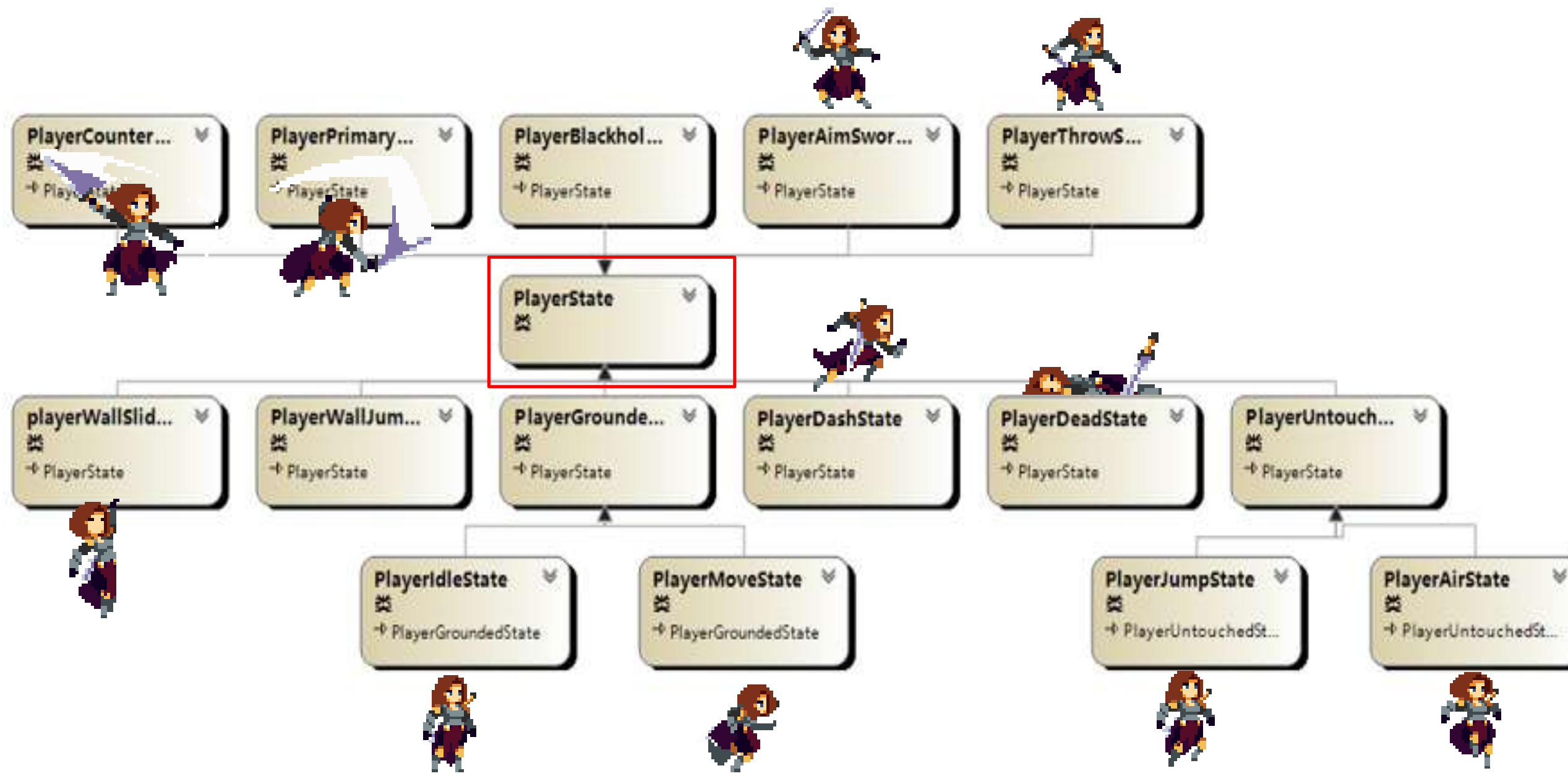
Systems Explanation

03

3 Systems Explanation

3.1 Entity System **3.1.1 Entity Behavior**

We use finite state machine (FSM) to control entity behavior



For States' Changing Logics

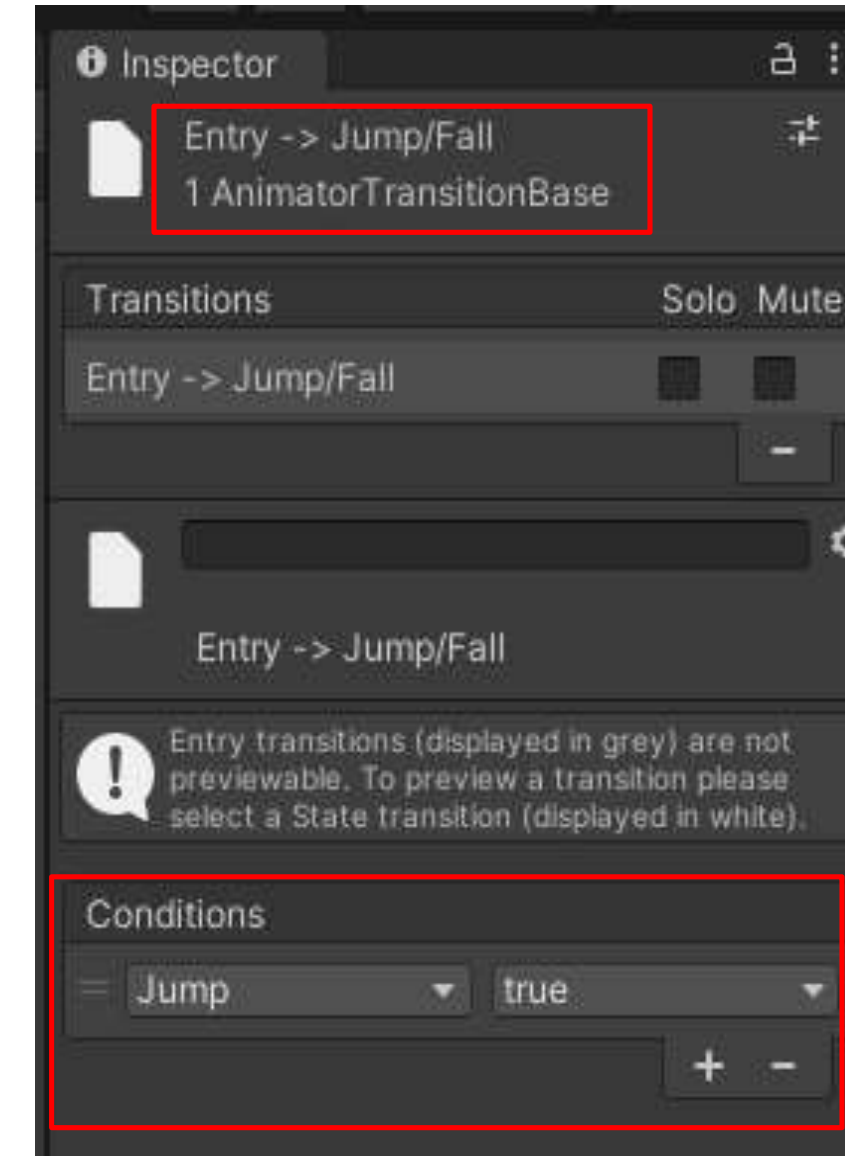
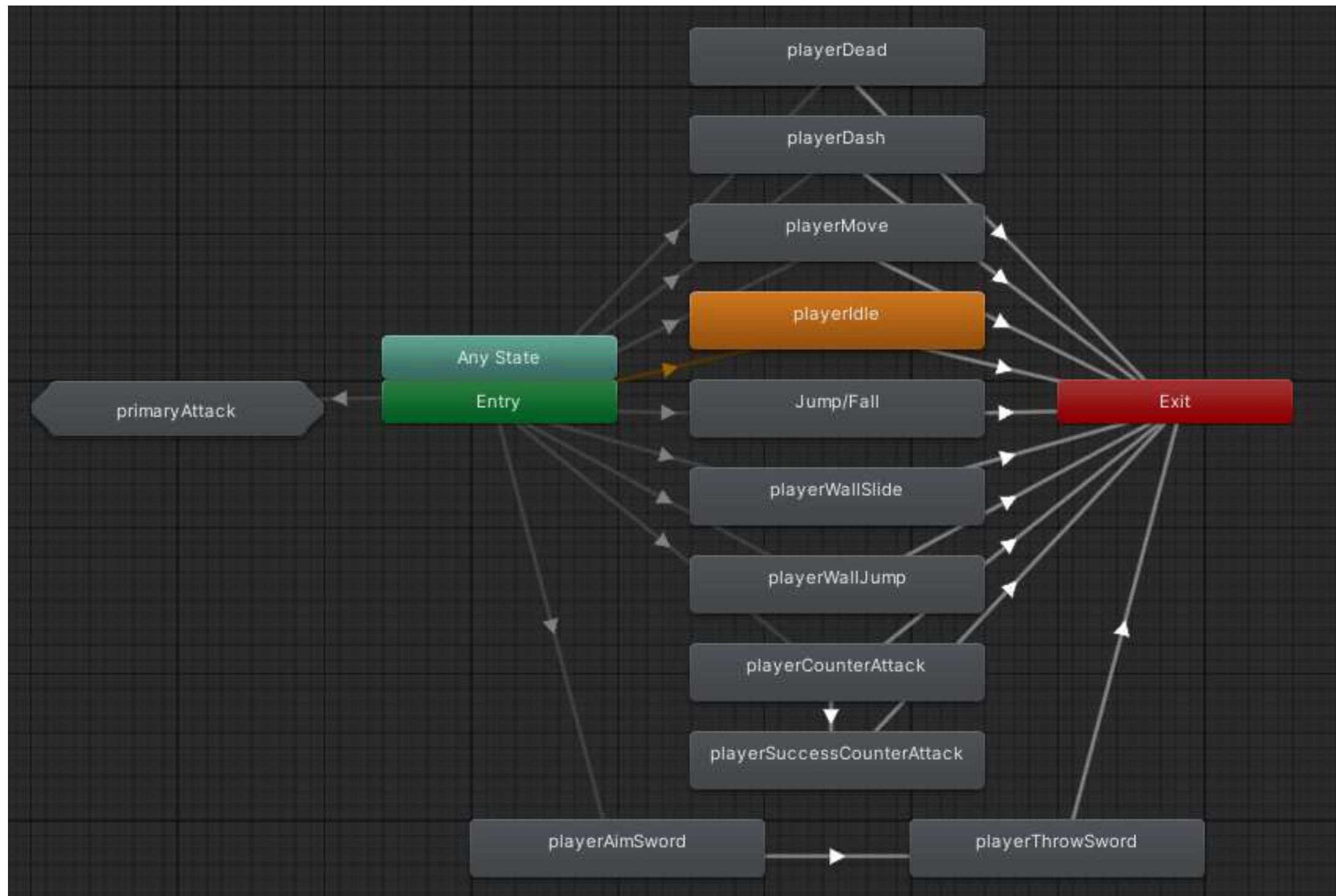


For Animation Logics

3 Systems Explanation

3.1 Entity System 3.1.1 Entity Behavior

The player's and enemy's **animation controller** based on the state machine
Each line is under **parameter** control, which is usually bool or integer



3 Systems Explanation

3.1 Entity System 3.1.1 Entity Behavior

```
public class PlayerState
{
    //受哪个状态机控制
    protected PlayerStateMachine stateMachine;
    //动作的名称, 即这个动作在Animator内相关联的bool值 (状态判断)
    private string animBoolName;
    //动作所属对象
    protected Player player;
    //使得PlayerState子类中调用rb更加快捷 (少写几个代码)
    protected Rigidbody2D rb;
    //水平速度输入
    public float xInput { get; private set; }
    //竖直速度输入
    public float yInput { get; private set; }
    //每个状态都对对应有的一个计时器
    protected float stateTimer;

    public PlayerState(Player _player, PlayerStateMachine _stateMachine,
string _animBoolName)
    //PlayerState类的构造函数, 指明了确定一个动作状态需要三个东西: 动作
    属于谁 (Player), 受哪个状态机控制, 这个动作在Animator内相关联的bool值
    {
        this.player = _player;
        this.stateMachine = _stateMachine;
        this.animBoolName = _animBoolName;
    }
    public virtual void Enter() {.....}
    public virtual void Exit() {.....}
    public virtual void Update() {.....}
}
```



```
public virtual void Enter()
{
    rb = player.rb;
    //赋值这个动作的激活状态为真
    player.anim.SetBool(animBoolName, true);
}

public virtual void Update()
//在Player内使用其Update不断调用此函数
{
    //每1s递减1单位数值
    stateTimer -= Time.deltaTime;
    //持续更新, 将yVelocity参数赋值为当前的竖直速度
    player.anim.SetFloat("yVelocity", rb.velocity.y);
    //将水平速度与AD两个键位绑定, 竖直速度与WS两键位绑定
    xInput = Input.GetAxisRaw("Horizontal");
    yInput = Input.GetAxisRaw("Vertical");
}

public virtual void Exit()
{
    //赋值这个动作的激活状态为假
    player.anim.SetBool(animBoolName, false);
    //每次离开当下状态时, 被离开的状态就成了上一个状态, 记录下来
    player.AssignLastAnimBoolName(animBoolName);
}
```

(太长了, 拖出来展示)

3 Systems Explanation

3.1 Entity System 3.1.1 Entity Behavior



Other State

```
public class PlayerStateMachine
//这个类用于以一定的逻辑操控一个人物的所有动作 (PlayerState) 之间的相互转化; 此处
{
    //存储这个状态机当下展示的动作状态是什么; { get; private set; }表示这个变量对
    25 个引用
    public PlayerState currentState { get; private set; }

    //存储状态机上一个状态是什么
    6 个引用
    public PlayerState formerState { get; private set; }

    1 个引用
    public void Initialize(PlayerState _startState)
    {
        //设定这个状态机的初始状态, 并进入该状态
        this.currentState = _startState;
        currentState.Enter();
    }

    29 个引用
    public void ChangeState(PlayerState _newState)
    {
        //退出上一个状态 (即把其关联的参数设置为false)
        currentState.Exit();
        //在转换状态之前, 记录下是从什么状态转换到了下一个状态
        formerState = currentState;
        //设置当前状态为输入的状态, 然后进入该状态 (即把其关联的参数设置为true)
        currentState = _newState;
        currentState.Enter();
    }
}
```

```
public override void Update()
//这个函数不断地在被Player中的Update函数更新, 所以同样一直在更新
{
    base.Update();

    //对面对着墙壁的情况做单独的判断: 向着墙壁无法转移到Move, 不动则保持静止, 反走则可
    if (player.isWall)
    {
        //若是向着墙壁则无法走到; 特判xInput为零的时候也不动, 否则站着不动出问题
        if(player.facingDir == xInput || xInput == 0)
        {
            //直接停止, 无需判断是否执行最外层if往下的内容
            return;
        }
        //反着墙壁走
        else if(player.facingDir * xInput < 0)
        {
            //不知为何, 总是要帮助人物进行一次手动翻转
            player.Flip();
            player.stateMachine.ChangeState(player.moveState);
        }
    }

    //当在地上且x水平方向有输入的时候才进入移动状态
    if(xInput != 0 && player.isGround)
    {
        //通过自己从PlayerState继承来的成员player (这个player由于被Plaer.cs初始化的时
        player.stateMachine.ChangeState(player.moveState);
    }
}
```

3 Systems Explanation

3.1 Entity System 3.1.2 Entity Polymorphism

Red: generality of base class

Blue: characteristic of child class

```
#region SlowEntity
5 个引用
public virtual void SlowEntityBy(float _slowPercentage, float _slowDuration)
//使实体减速, 传入减速百分比和减速状态持续时长; 注意了, 敌人有move, 而玩家有move、
{
    //Debug.Log(this.name + " BeSlowed");

    //移动速度减速
    this.moveSpeed *= (1 - _slowPercentage);
    //动画的播放速度也需要被减缓
    this.anim.speed *= (1 - _slowPercentage);

    //经历这段时间后恢复原有速度
    Invoke("ReturnDefaultSpeed", _slowDuration);
}
5 个引用
protected virtual void ReturnDefaultSpeed()
{
    //Debug.Log(this.name + " DeSlowd");

    //恢复原有速度
    this.moveSpeed = defaultMoveSpeed;
    this.anim.speed = 1;
}
#endregion
```



```
#region SlowEntityOverride
4 个引用
public override void SlowEntityBy(float _slowPercentage, float _slowDuration)
{
    //特性的减速, 写在前面, 因为Invoke恢复速度的函数在底下的base里面
    jumpForce *= (1 - _slowPercentage);
    dashSpeed *= (1 - _slowPercentage);

    base.SlowEntityBy(_slowPercentage, _slowDuration);
}
3 个引用
protected override void ReturnDefaultSpeed()
{
    base.ReturnDefaultSpeed();

    //恢复速度
    jumpForce = defaultJumpForce;
    dashSpeed = defaultDashSpeed;
}
#endregion
```



```
#region SlowEntityOverride
4 个引用
public override void SlowEntityBy(float _slowPercentage, float _slowDuration)
{
    //特性的减速, 写在前面
    battleSpeedMultiplier *= (1 - _slowPercentage);

    base.SlowEntityBy(_slowPercentage, _slowDuration);
}
4 个引用
protected override void ReturnDefaultSpeed()
{
    base.ReturnDefaultSpeed();

    //恢复速度
    battleSpeedMultiplier = defaultBattleSpeedMultiplier;
}
#endregion
```



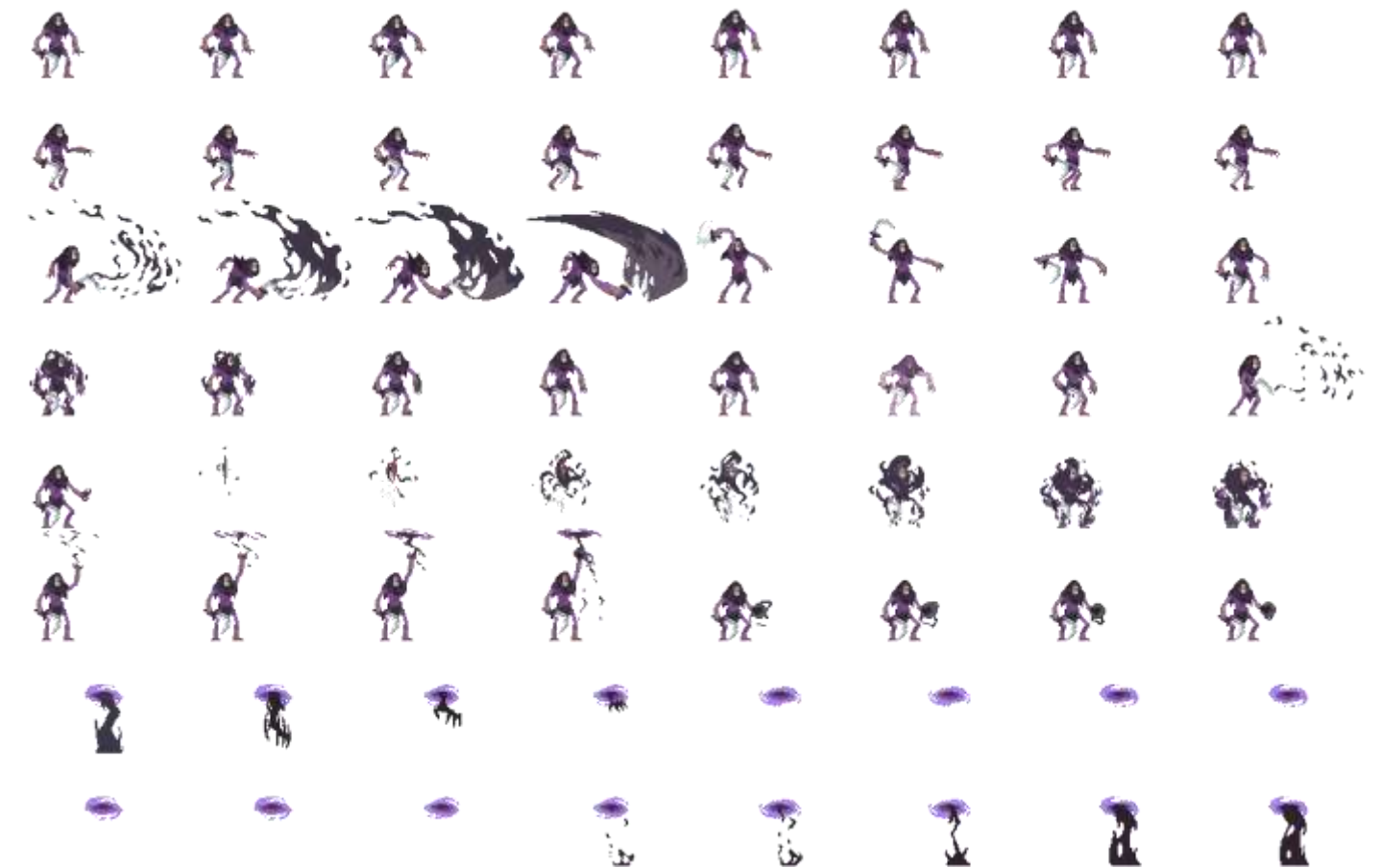
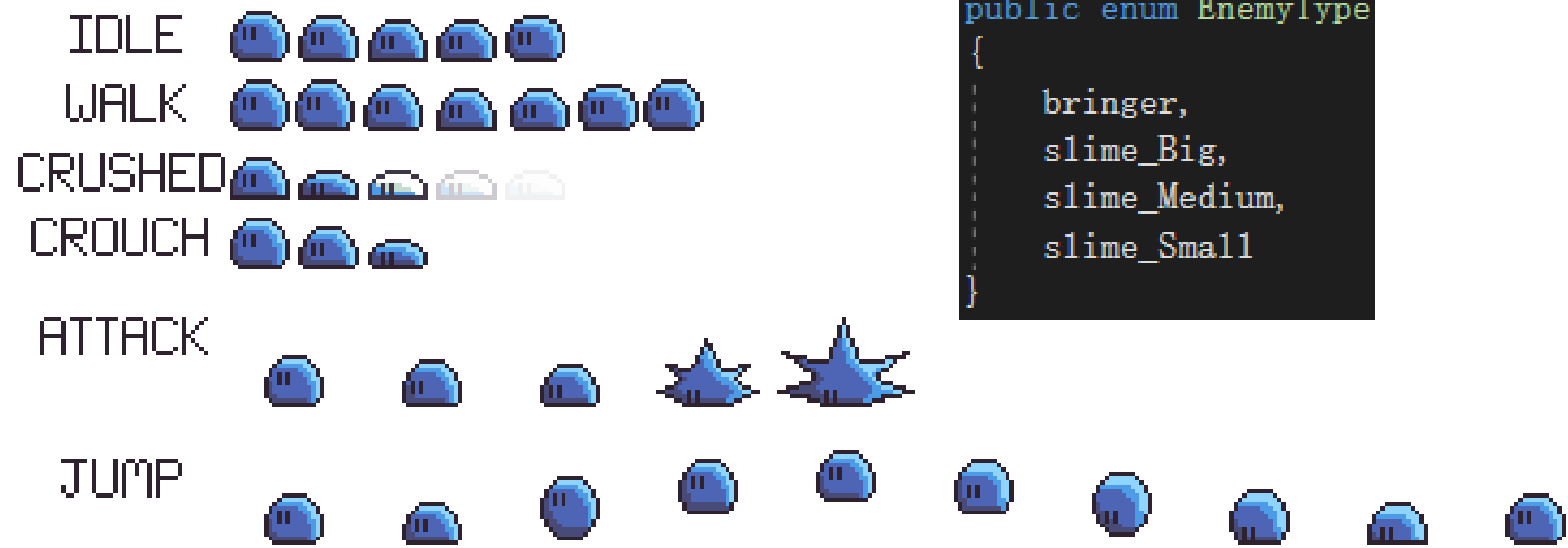
3 Systems Explanation

3.1 Entity System **3.1.2 Entity Polymorphism**

Player can be controlled by keyboard and mouse

Enemy also have several child classes, indicating different behavior logics and animation

Bringer's Die() function is different from Slime's, which will split to smaller slimes



3 Systems Explanation

3.1 Entity System 3.1.3 Entity Statistics



```
public class Stat
//无继承，这是一个单纯表示一种自定义数值的类
{
    //该数值的基础数值
    [SerializeField] private int baseValue = 0;

    public int GetValue()
    //对外提供接口，便可以获取最终输出的数值
    {
        int _finalValue = baseValue;
        return _finalValue;
    }

    public void SetValue(int _value)
    {
        baseValue = _value;
    }
}
```

```
#region Attack
[Header("Attack Stats")]
//暴击率（百分比）
public Stat criticChance;
//暴击伤害倍率（百分比，大于100）
public Stat criticPower;

//实体的基础物理攻击伤害
public Stat primaryPhysicalDamage;
//火焰伤害
public Stat fireAttackDamage;
//冰冻伤害
public Stat iceAttackDamage;
//闪电伤害
public Stat lightningAttackDamage;
#endregion
```

```
#region FinalValues
5 个引用
public virtual int GetNonCritPhysicalDamage()
//得到不进行暴击判定的原始物理伤害
{
    return primaryPhysicalDamage.GetValue() + 10 * strength.GetValue() + 5 * agility.GetValue()
}
4 个引用
public virtual int GetNonCritMagicalDamage()
```

```
public class EntityStats : MonoBehaviour
//这个类负责控制实体的统计数据
{
    Components
    Health
    Attribute
    Attack
    Defence
    Default
    Events
    Unity 消息 | 8 个引用
    protected virtual void Start()
    {
        Components
        //这里的Start函数必须要确保比更新血条UI的Start函数先调用
        //初始时赋予实体其加成过后的最大生命值
        currentHealth = GetFinalMaxHealth();
        //Debug.Log("EntityStats Start() Func Called");
    }
    GetDamaged
    DecreaseDefence
    FinalValues
    ChanceAnalyze
    StatTypeMapping
}
```

3 Systems Explanation

3.1 Entity System 3.1.3 Entity Statistics

```
#region GetDamaged
//整体考虑实体受到的攻击，复合了实体受到的物理和魔法两类伤害
#region TotalDamage
3 个引用
public virtual void GetTotalNormalDmgFrom(EntityStats _attackingEntity, bool _doPhysic, bool _doMagic)
//第二、第三参数位要传入是否只触发单独一类伤害或者两者一起触发的布尔值
{
    Evade&Crit
    AttackedFX

    //若是对方基础伤害为0，则不应进行伤害
    if (_attackingEntity.GetNonCritPhysicalDamage() > 0 && _doPhysic)
    {
        //物理数值伤害的施加
        this.GetPhysicalDamagedBy(_physicDmg);
    }
    if (_attackingEntity.GetNonCritMagicalDamage() > 0 && _doMagic)
    {
        //魔法数值伤害的施加
        this.GetMagicalDamagedBy(_magicDmg);

        //魔法元素相关Buff的施加
        buf.CheckBufsFrom(_attackingEntity);
    }
}
```

Use stat to do damage on other entity

```
private void AttackDamageTrigger()
{
    Collider2D[] collidersInAttackZone = Physics2D.OverlapCircleAll(bringer.attackCheck.position, bringer.attackCheckRadius);

    foreach (var beHitEntity in collidersInAttackZone)
    {
        if (beHitEntity.GetComponent<Player>() != null)
        {
            //攻击减少对方生命值并产生受击效果
            beHitEntity.GetComponent<PlayerStats>().GetTotalNormalDmgFrom(bringer.sts, true, true);
        }
    }
}
```

```
#region Evade&Crit
//记录对方伤害、暴击等属性
int _attackingCritPower = _attackingEntity.GetFinalCriticPower();
int _physicDmg = _attackingEntity.GetNonCritPhysicalDamage();
int _magicDmg = _attackingEntity.GetNonCritMagicalDamage();

//如果触发了闪避，则直接返回，不受伤
if (CanEvade())
{
    return;
}

//如果对方触发了暴击，则读取对方暴击伤害进行受伤的增伤
if (CanCrit(_attackingEntity))
{
    //使用暴击倍率需要除以100变为浮点数形式，但最终还是要返回一个整型数据
    float _criticPowerPercentage = GetFinalCriticPower() * 0.01f;
    //从浮点转化为整型
    _physicDmg = Mathf.RoundToInt(_criticPowerPercentage * _physicDmg);
    _magicDmg = Mathf.RoundToInt(_criticPowerPercentage * _magicDmg);
}
}
#endregion
```

```
#region AttackedFX
//受攻击的音效
AudioManager.instance.PlaySFX(12, null);
//受攻击的粒子效果，在自己（受攻击者）身上
fx.CreateHitFX00(this.transform);
#endregion
```

3 Systems Explanation

3.1 Entity System 3.1.4 Entity Buffs

Buff
类

```
public class Buff
{
    //该Buff的激活状态
    [SerializeField] protected bool status;
    //该Buff的持续时长
    public float duration = 5f;

    6 个引用
    public virtual bool GetStatus() => status;

    9 个引用
    public virtual void SetStatus(bool _bool) => status = _bool;
}
```

```
public class Buff_Chilled : Buff
{
    //处于冰冻状态时, 所有速度减慢
    public float slowPercentage = 0.3f;
}
```

```
public class Buff_Ignited : Buff
{
    //处于灼烧状态时, 每隔多长时间受到一次灼烧伤害
    public float damageCooldown = 1f;
    //每次受到灼烧伤害掉百分之多少血量
    public float burnHealthPercentage = 0.03f;
}
```

```
public class Buff_Shocked : Buff
{
    //眩晕状态下 防御各项属性减少的百分比
    public float defenceDecreasePercentage = 0.2f;
}
```

EntityBufs
类
→ MonoBehaviour

```
#region ApplyBufs
1 个引用
public virtual void CheckBufsFrom(EntityStats _entity)
{
    #region Evaluation
    //存储攻击自己的实体的魔法元素伤害数据
    int _fireDmg = _entity.fireAttackDamage.GetValue();
    int _iceDmg = _entity.iceAttackDamage.GetValue();
    int _lightDmg = _entity.lightningAttackDamage.GetValue();

    //只要有这个类型的魔法伤害, 则施加这个buff
    bool _canApplyIgnite = (_fireDmg > 0);
    bool _canApplyChill = (_iceDmg > 0);
    bool _canApplyShock = (_lightDmg > 0);
    #endregion

    //施加Bufs
    ApplyBufs(_canApplyIgnite, _canApplyChill, _canApplyShock);
}
2 个引用
public virtual void ApplyBufs(bool _ignited, bool _chilled, bool _shocked)
#endregion

#region ClearBufs
1 个引用
public void ClearAllBufs()
{
    //清除所有Bufs
    ignited.SetStatus(false);
    chilled.SetStatus(false);
    shocked.SetStatus(false);
}
#endregion
```

```
public class EntityBufs : MonoBehaviour
//用于控制实体的Buff状态
{
    Components

    #region Bufs
    [Header("Bufs")]
    //燃烧状态, 效果时间内持续掉血
    public Buff_Ignited ignited;
    //冰冻状态, 效果时间内速度减慢
    public Buff_Chilled chilled;
    //眩晕状态, 效果时间内防御降低
    public Buff_Shocked shocked;
    #endregion

    Timers

    Unity 消息 | 0 个引用
    private void Start()
    {
        Components

        //初始时清空所有Bufs
        ClearAllBufs();
    }

    Unity 消息 | 0 个引用
    private void Update()
    {
        //检测各种Buff
        BufsDetector();
    }

    DetectBufs
    ApplyBufs
    ClearBufs
}
```


3 Systems Explanation

3.1 Entity System 3.1.5 Entity Effects



```
public void CreatPopUpText(string _text, Color _color)
//控制弹出这个文字效果的函数，接收需要弹出的内容及其颜色
{
    //调整文字效果相对召唤者的生成位置，在范围内随机
    float _randomX = Random.Range(-1.5f, 1.5f);
    float _randomY = Random.Range(0.5f, 2f);
    Vector3 _positionOffset = new Vector3(_randomX, _randomY, 0);

    //调用预制体
    GameObject _newText = Instantiate(popUpTextPrefab, transform.position + _positionOffset, Quaternion.identity);

    _newText.GetComponent<TextMeshPro>().color = _color;
    _newText.GetComponent<TextMeshPro>().text = _text;
}
```

```
public void CreateHitFX00(Transform _target)
//传入敌人位置，受击效果产生在敌人身上
{
    //随机的位移与旋转，使得效果看起来不一样
    float _xPosition = UnityEngine.Random.Range(-0.5f, 0.5f);
    float _yPosition = UnityEngine.Random.Range(-0.5f, 0.5f);
    //float _zRotation = UnityEngine.Random.Range(-90, 90);

    //生成预制体
    GameObject _newHitFX = Instantiate(hitFX00, _target.position + new Vector3(_xPosition, _yPosition), Quaternion.identity);
    //_newHitFX.transform.Rotate(new Vector3(0, 0, _zRotation));

    //销毁
    Destroy(_newHitFX, 1f);
}
```

```
public class EntityFX : MonoBehaviour
{
    //链接到实体的Animator内的渲染器Component
    private SpriteRenderer sr;

    PopUpText
    AttackFX
    BuffsFX

    Unity 消息 | 0 个引用
    private void Start()
    {
        //链接到实体的Animator内的渲染器Component
        sr = GetComponentInChildren<SpriteRenderer>();

        //记录原始材质
        originMat = sr.material;
    }

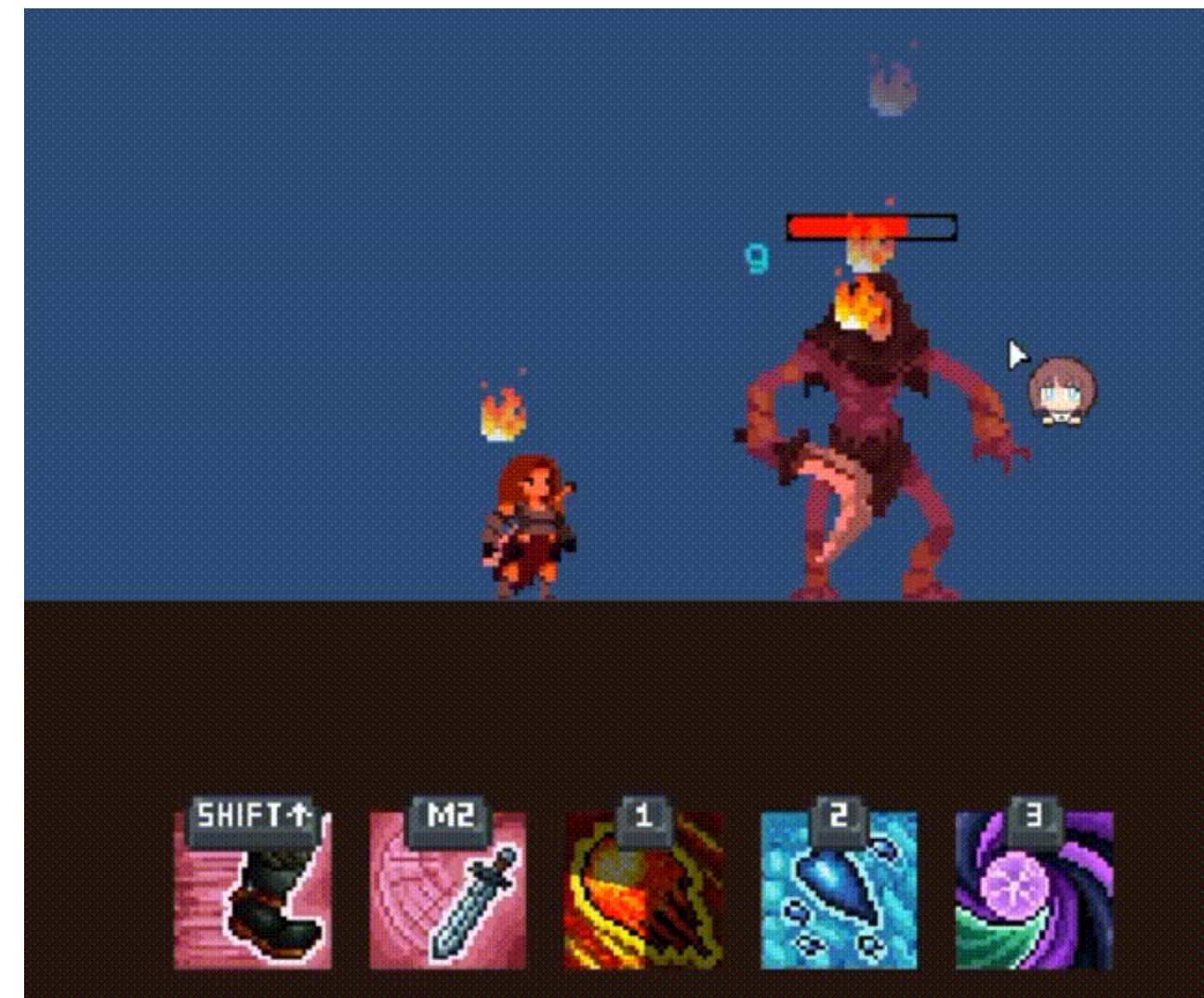
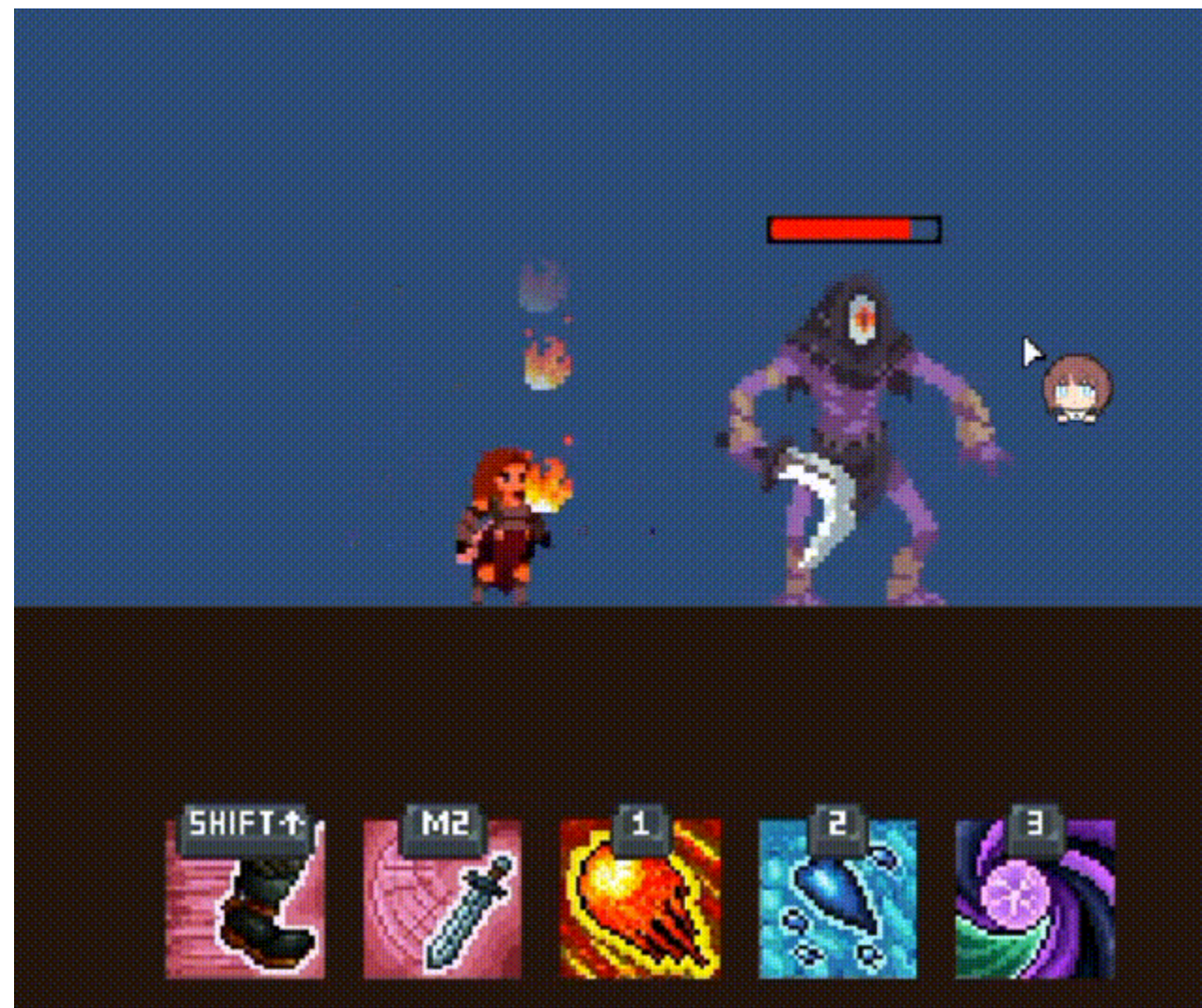
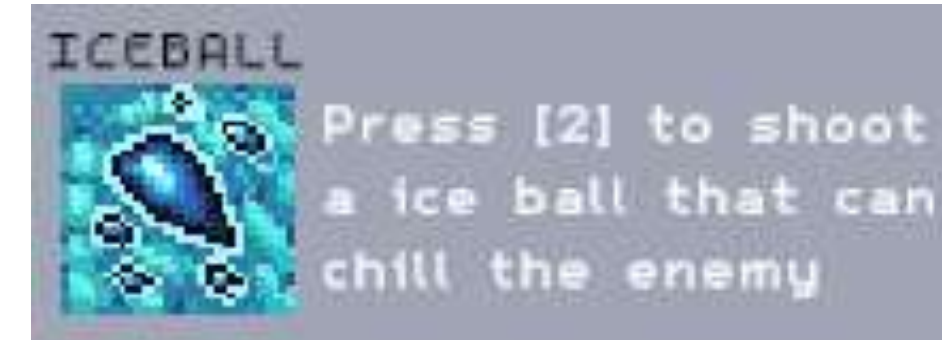
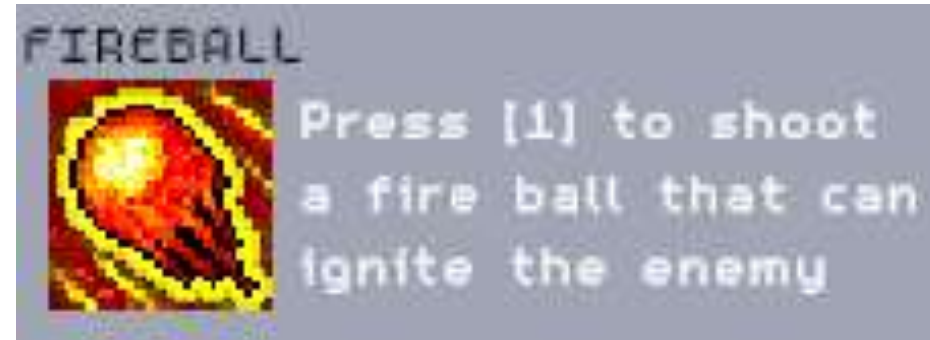
    PopUpText
    Clear
    HitParticleFX
    DamagedMatFX
    BuffsFX
}
```

```
public void InvokeIgnitedFXFor(float _duration)
//调用燃烧效果多长时间
{
    //调用粒子效果
    ignitedFX.gameObject.SetActive(true);
    ignitedFX.Play();

    //调用颜色效果
    sr.color = ignitedColor;
    //经历_duration时间过后结束效果
    Invoke("CancelColorChange", _duration);
}
```

3 Systems Explanation

3.1 Entity System **3.1.6 Player Skills**



3 Systems Explanation

3.1 Entity System 3.1.6 Player Skills

SWORD

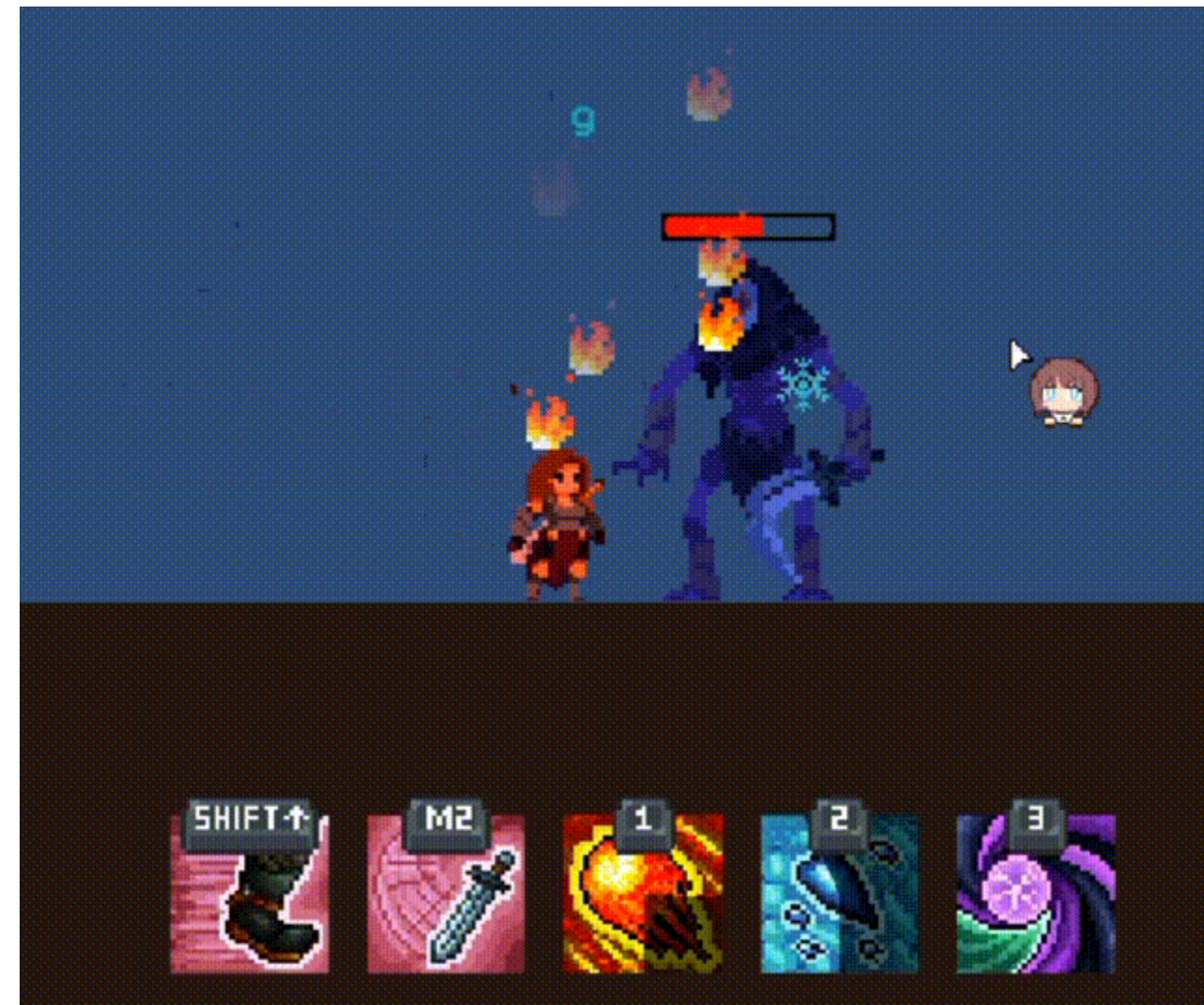
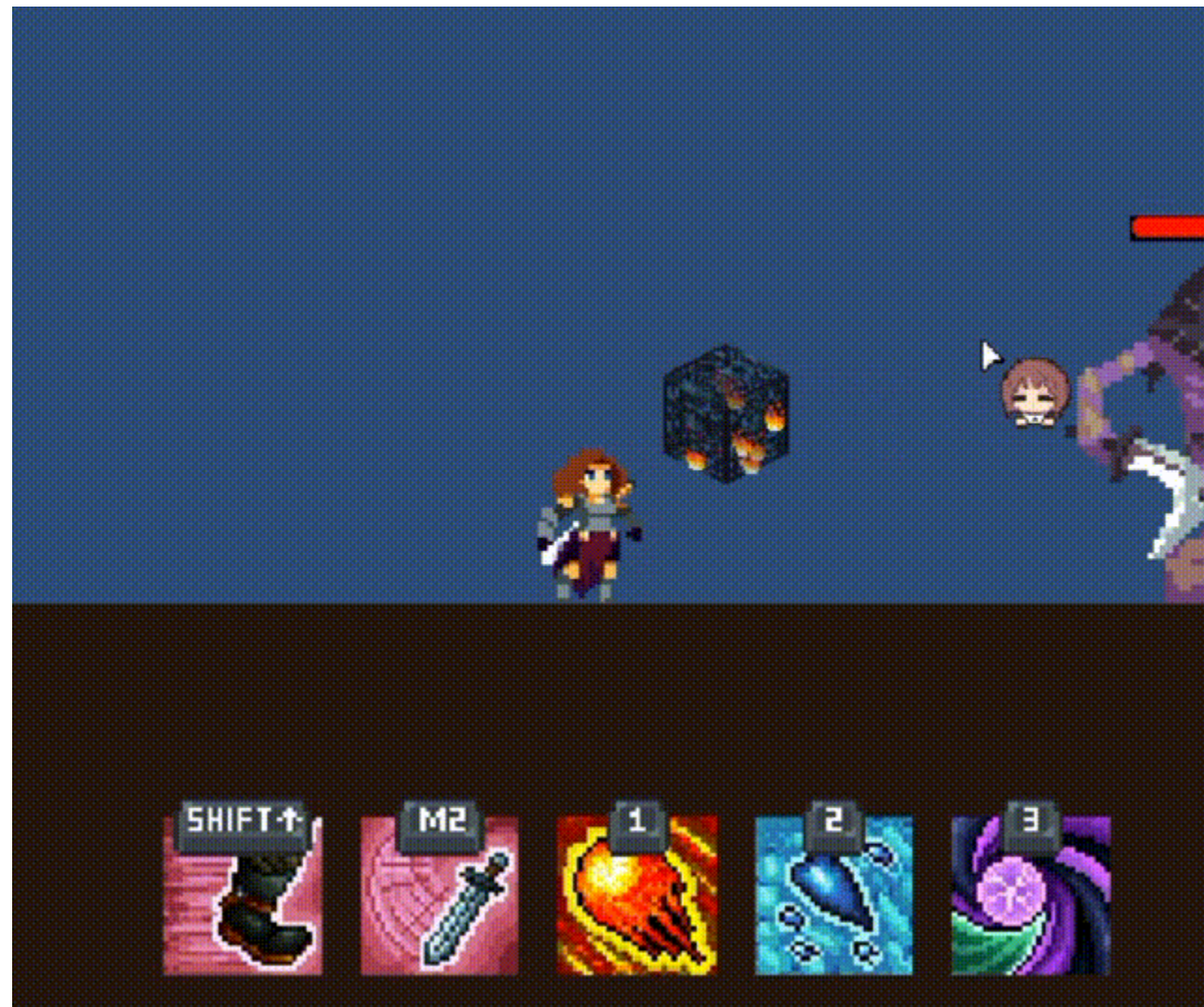


Click [Middle Mouse Button] to shoot a lightning sword

BLACKHOLE



Press [3] to summon a blackhole, press [3] again to leave. If you press the keys appeared above enemies head (such as [Y] [4] [6] [H]) during the skill period, there will be clone warriors attacking the enemies



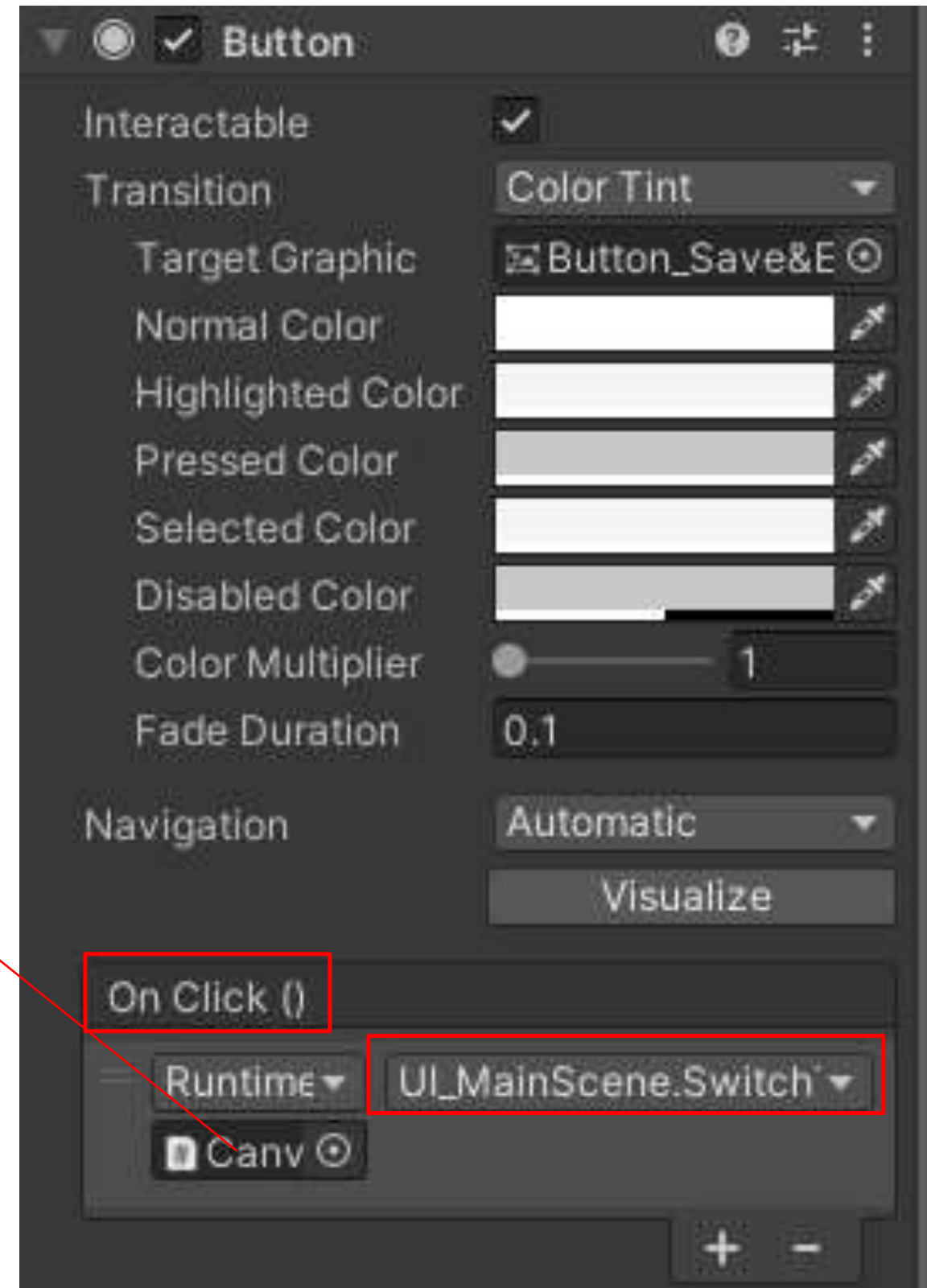
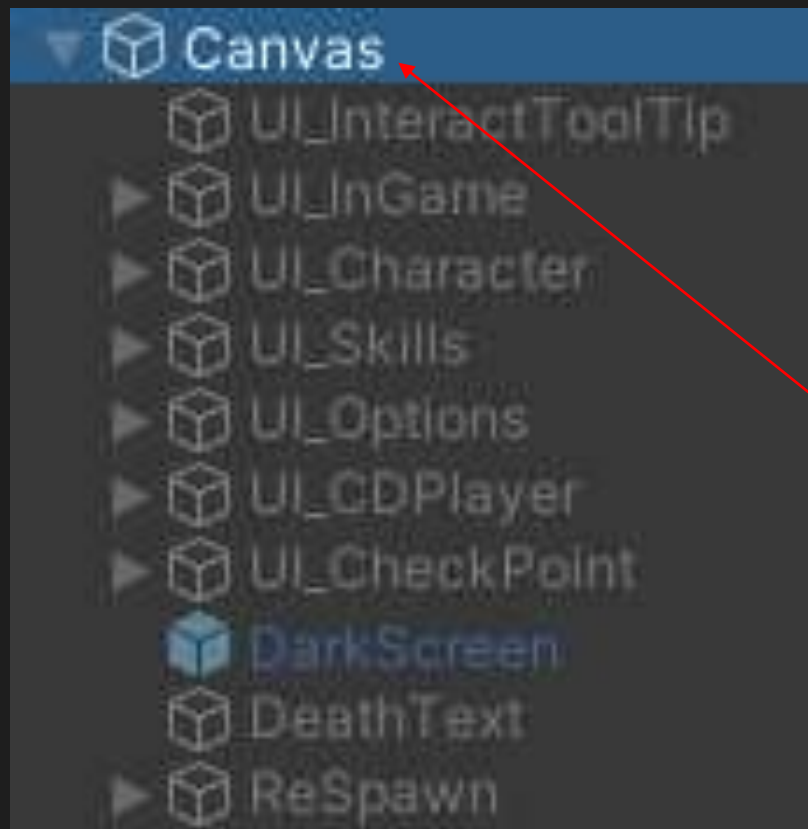
3 Systems Explanation

3.2 UI System

```
public void SwitchToUI(GameObject _menu)
{
    //遍历Canvas的子对象
    for (int i = 0; i < transform.childCount; i++)
    {
        //保证按钮提示UI可以正常显示; 要防止fadeScreen被直接关闭而不能激活屏幕fade的相关动画
        if (transform.GetChild(i).gameObject != interactToolTipUI && transform.GetChild(i).gameObject != fadeScreen)
        {
            //关闭所有子对象
            transform.GetChild(i).gameObject.SetActive(false);
        }
    }

    //开启需要转换到的非空对象
    if(_menu != null)
    {
        _menu.SetActive(true);
        //UI切换的音效
        AudioManager.instance.PlaySFX(8, null);
    }

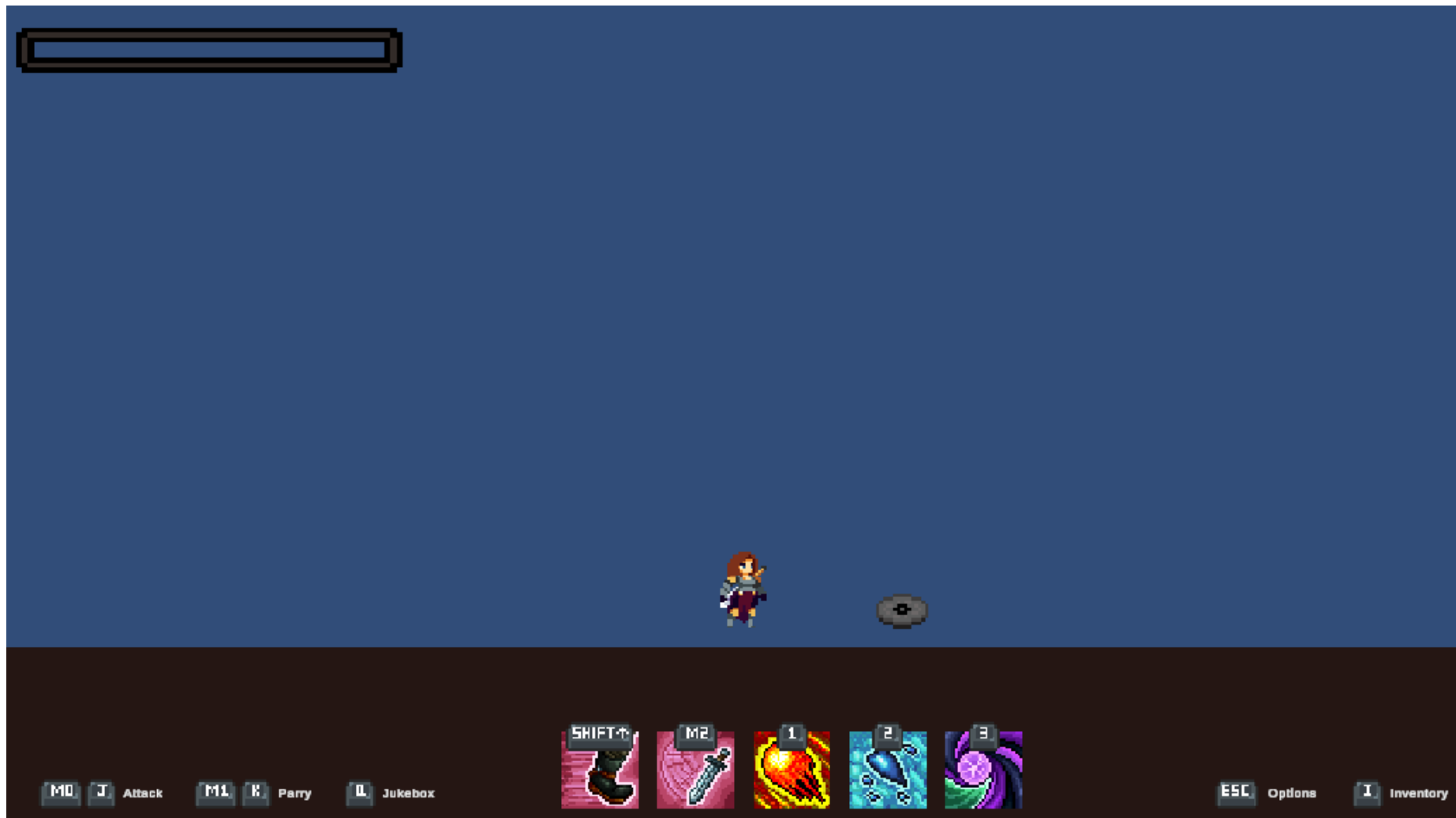
    #region GamePause
    //打开UI时暂停游戏
    if(GameManager.instance != null)
    {
        if (_menu == inGameUI)
            GameManager.instance.PauseGame(false);
        else
            GameManager.instance.PauseGame(true);
    }
    #endregion
}
```



3 Systems Explanation

3.2 UI System

In Game UI



Character UI Menu



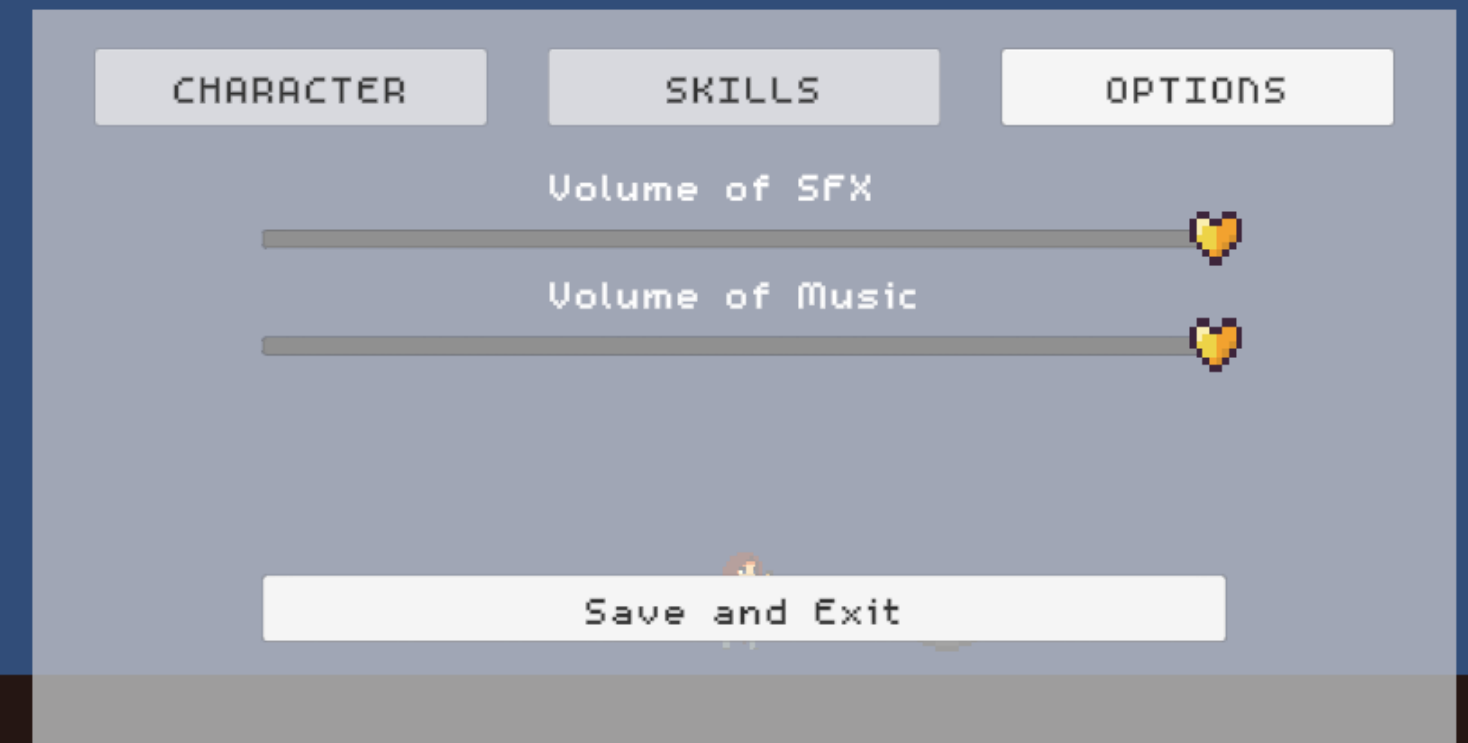
3 Systems Explanation

3.2 UI System

Skill Info UI



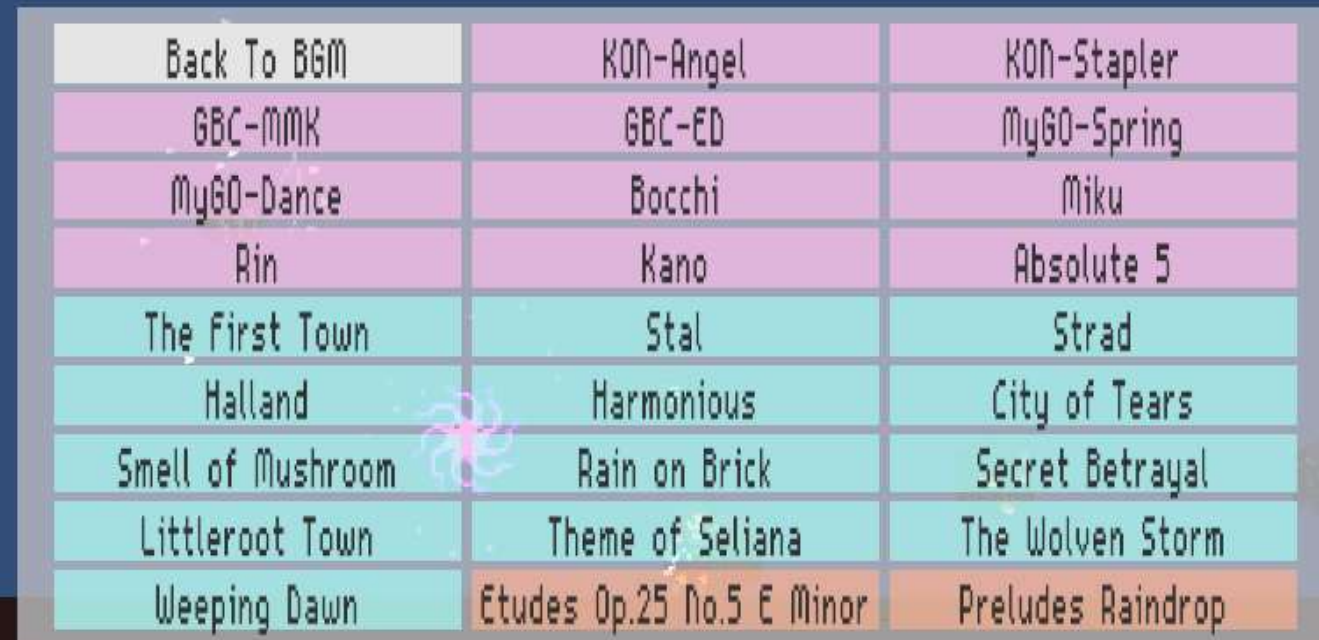
Options UI Menu



3 Systems Explanation

3.2 UI System

Jukebox UI Menu



| | | |
|-------------------|---------------------------|-------------------|
| Back To BGM | KON-Angel | KON-Stapler |
| GBC-MMK | GBC-ED | MyGO-Spring |
| MyGO-Dance | Bocchi | Miku |
| Rin | Kano | Absolute 5 |
| The First Town | Stal | Strad |
| Holland | Harmonious | City of Tears |
| Smell of Mushroom | Rain on Brick | Secret Betrayal |
| Littleroot Town | Theme of Seliana | The Wolven Storm |
| Weeping Dawn | Etudes Op.25 No.5 E Minor | Preludes Raindrop |

Checkpoint UI Menu

If you died, you will respawn near the bonfire at which you last rested

Rest At Bonfire

Refresh your Health, and respawn all the enemies



3 Systems Explanation

3.3 Inventory System

Inventory Save & Load

```
public class ItemData : ScriptableObject
//继承自ScriptableObject这个类，是一种很好用的模板
{
    //物品的名称
    public string itemName;
    //物品的类型
    public ItemType itemType;
    //物品的描述
    [TextArea]
    public string itemDescription;
    //物品的贴图
    public Sprite itemIcon;

    //每个物品特有的id序列，用于存档记录物品栏内物品相关功能
    public string itemID;
```

ItemData
类
→ ScriptableObject

```
public enum ItemType
{
    Weapon,
    Potion,
    CD
}
```

```
[Header("Inventory Items")]
//记录“物品栏物品”（一种类似Stat的自定义数据类型）
public List<StoredItem> inventoryItemsList;
//使用字典来存储ItemData与InventoryItem一一对应
public Dictionary<ItemData, StoredItem> inventoryItemsDict;

[Header("ItemData Base")]
//用于防止build时报错所用到的新代码
public List<ItemData> itemDataBase;
//从存档加载到物品栏的物品列表
public List<StoredItem> loadedItems;
```

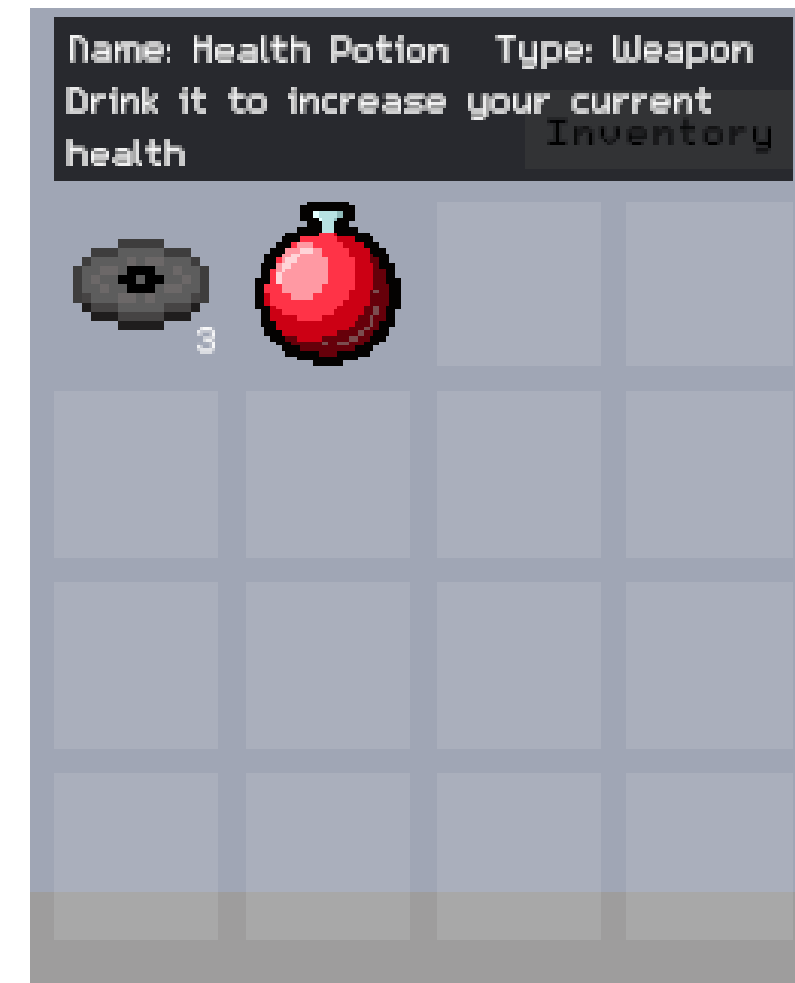
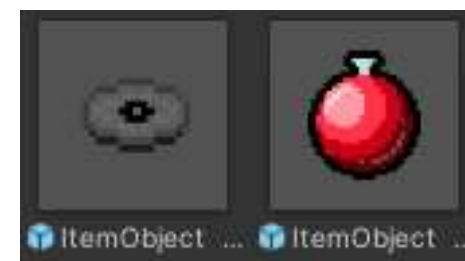
ISavesManager
类
→ MonoBehaviour

Inventory
类
→ MonoBehaviour

```
private void OnTriggerEnter2D(Collider2D collision)
//判断主角是否与物品发生了碰撞，记得保证物品Object有Collider组件
{
    //若主角与物品的碰撞箱碰撞，且背包有余位，则捡起物品
    if(collision.GetComponent<Player>() != null && Inventory.instance.CanAddNewItem())
    {
        //拾取物品音效
        AudioManager.instance.PlaySFX(6, null);

        //通过instance调用物品栏，直接调用Inventory的instance即可
        //不同于PlayerManager的通过instance.player来调用，因为在那里instance代表的是Player
        Inventory.instance.AddItem(itemData);
        //销毁此item
        Destroy(gameObject);
    }
    if(collision.GetComponent<Player>() != null && !Inventory.instance.CanAddNewItem())
    {
        //提示背包没有空间
        PlayerManager.instance.player.fx.CreatPopUpText("No Space", Color.white);
    }
}
```

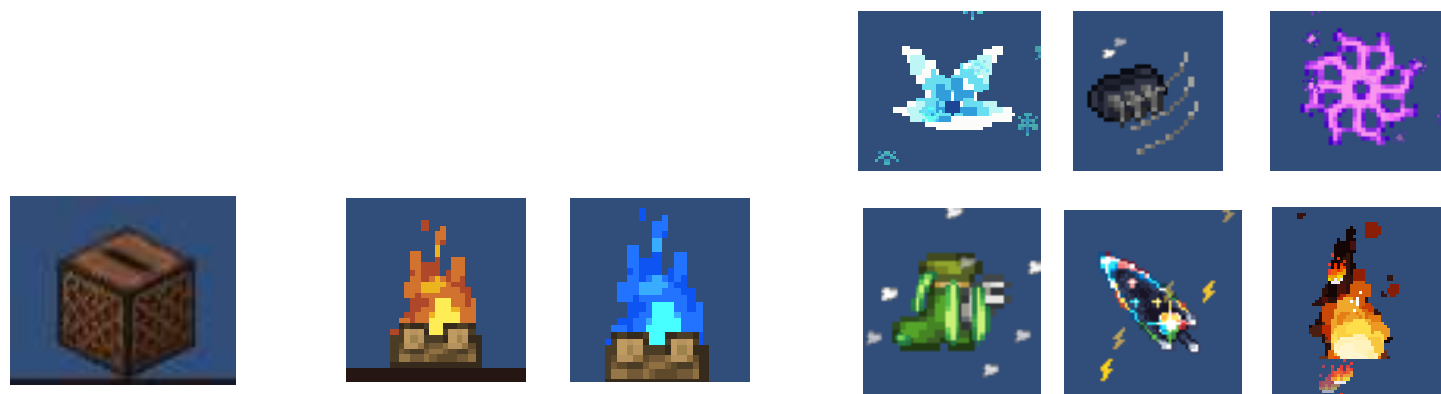
ItemObject
类
→ MonoBehaviour



UI_ItemSlot
类
→ MonoBehaviour

3 Systems Explanation

3.4 Interact System



```
public class CDPlayer : MonoBehaviour
{
    private BoxCollider2D cd;
    private Rigidbody rb;
    private void Awake() {cd = GetComponent<BoxCollider2D>(); rb = GetComponent<Rigidbody>();}
    private void OnTriggerEnter2D(Collider2D collision)
    //当玩家与唱片机的碰撞箱接触时执行的语句
    {
        //必须是玩家，而非别的什么怪物都能触发
        if (collision.GetComponent<Player>() != null)
        {
            //表示人物处于可触发交互界面的区域内，显示按键提示
            UI_MainScene.instance.SetWhetherShowInteractToolTip(true);
            //表示现在接触的可交互物是唱片机
            UI_MainScene.instance.isAtCDPlayer = true;
        }
    }
    private void OnTriggerExit2D(Collider2D collision)
    //当玩家离开唱片机的碰撞箱范围时执行的语句
    {
        if (collision.GetComponent<Player>() != null)
        {
            //关闭按键提示
            UI_MainScene.instance.SetWhetherShowInteractToolTip(false);
            UI_MainScene.instance.isAtCDPlayer = false;

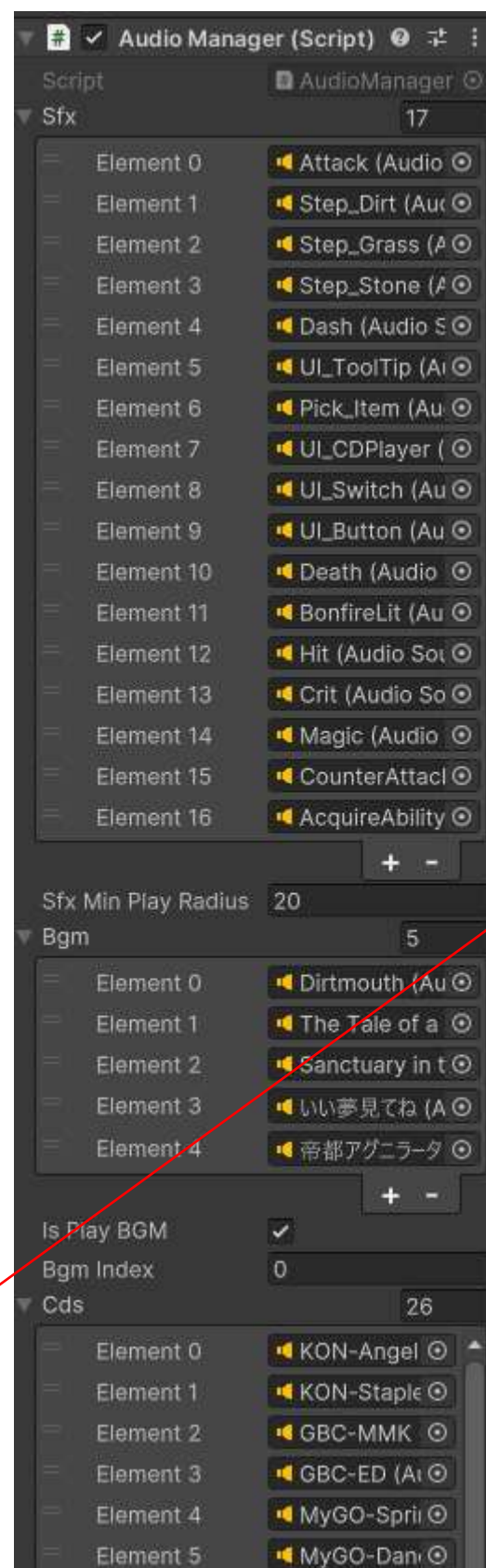
            //若离开时唱片机UI是开启的，则关闭
            //此处有点莫名其妙的bug...?
            if (UI_MainScene.instance.cdPlayerUI != null)
            {
                if (UI_MainScene.instance.cdPlayerUI.activeSelf)
                    UI_MainScene.instance.SwitchToUI(UI_MainScene.instance.inGameUI);
            }
        }
    }
}
```

3 Systems Explanation

3.5 Audio System

```
public class AudioManager : MonoBehaviour
```

```
{  
    public static AudioManager instance;  
    //音效的列表  
    [SerializeField] AudioSource[] sfx;  
    //音效播放的检测半径, 太远的音效不予播放  
    [SerializeField] float sfxMinPlayRadius;  
    private bool canPlaySFX;  
    //背景音乐与唱片相关数据  
    [SerializeField] AudioSource[] bgm;  
    public bool isPlayBGM;  
    public int bgmIndex;  
    [SerializeField] AudioSource[] cds;  
    public bool isPlayCD;  
    private void Awake()  
    {  
        //确保管理器仅有一个  
        if(instance != null)  
            Destroy(instance.gameObject);  
        else  
            instance = this;  
        //在进入场景0.1秒后才允许播放音效  
        Invoke("AllowPlaySFX", 0.1f);  
    }  
    private void Update()  
    {  
        //音效与音乐相关控制, 代码略  
    }  
    //音效与音乐相关控制函数
```



```
public void PlaySFX(int _sfxIndex, Transform _sfxSource)
```

```
{  
    //进入场景时, 0.1秒后才允许播放音效  
    if (!canPlaySFX)  
        return;  
  
    //若存在妄图播放的音效但太过遥远, 则不播放  
    if (_sfxSource != null &&  
        Vector2.Distance(PlayerManager.instance.player.transform.position,  
            _sfxSource.position) >= sfxMinPlayRadius)  
        return;  
  
    //若编号存在于列表内 (编号从0开始哦)  
    if(_sfxIndex < sfx.Length && sfx[_sfxIndex] != null)  
    {  
        //一个小trick, 随机化播放目标音效的音高  
        //sfx[_sfxIndex].pitch = UnityEngine.Random.Range(0.85f, 1.1f);  
  
        //播放音效  
        sfx[_sfxIndex].Play();  
    }  
}
```

```
//停止音效  
public void StopSFX(int _sfxIndex) => sfx[_sfxIndex].Stop();
```

```
//允许播放音效  
public void AllowPlaySFX() => canPlaySFX = true;
```

```
public void PlayBGM(int _index) {.....}  
public void StopAllBGM() {.....}  
public void PlayCD(int _cdIndex) {.....}  
public void StopAllCD() {.....}
```

3 Systems Explanation

3.6 Save&Load System

```
public interface ISavesManager
```

```
//创建一个接口（名字一般以I开头）
```

```
{  
    void LoadData(GameData _data);
```

```
    //注意这里是引用，使得可以改变传入对象  
    void SaveData(ref GameData _data);  
}
```



```
public class PlayerManager : MonoBehaviour, ISavesManager
```

```
{  
    //玩家是否能使用冲刺技能  
    public bool ability_CanDash;
```

```
    //其它代码块
```

```
    public void LoadData(GameData _data)  
    //加载游戏时候执行的操作  
    {  
        //读取能力许可  
        ability_CanDash = _data.canDash;  
    }
```

```
    public void SaveData(ref GameData _data)  
    //存储游戏时候执行的操作  
    {  
        //存储能力许可  
        _data.canDash = ability_CanDash;  
    }  
}
```

```
{  
  "currency": 0,  
  "canWallSlide": false,  
  "canDash": true,  
  "canDoubleJump": true,  
  "canThrowSword": true,  
  "canFireBall": true,  
  "canIceBall": true,  
  "canBlackhole": true,  
  "strength": 0,  
  "agility": 0,  
  "vitality": 0,  
  "intelligence": 0,  
  "originalMaxHealth": 200,  
  "criticPower": 150,  
  "criticChance": 10,  
  "primaryPhysicalDamage": 20,  
  "fireAttackDamage": 0,  
  "iceAttackDamage": 0,  
  "lightningAttackDamage": 5,  
  "swordDamage": 5,  
  "fireballDamage": 25,  
  "iceballDamage": 25,  
  "evasionChance": 5,  
  "physicalArmor": 10,  
  "magicalResistance": 10,  
  "inventory": {  
    "keys": [],  
    "values": []  
  },  
  "checkpointsDict": {  
    "keys": [  
      "763cd3f0-d3b5-4601-ba4a-73e2057316c0",  
      "af18d78d-365e-4827-b361-23583f96e457"  
    ],  
    "values": [  
      false,  
      false  
    ]  
  },  
  "lastRestCPID": "",  
  "volumeSettings": {  
    "keys": [  
      "Volume_SFX",  
      "Volume_Music"  
    ],  
    "values": [  
      1.0,  
      1.0  
    ]  
  }  
}
```

Data File

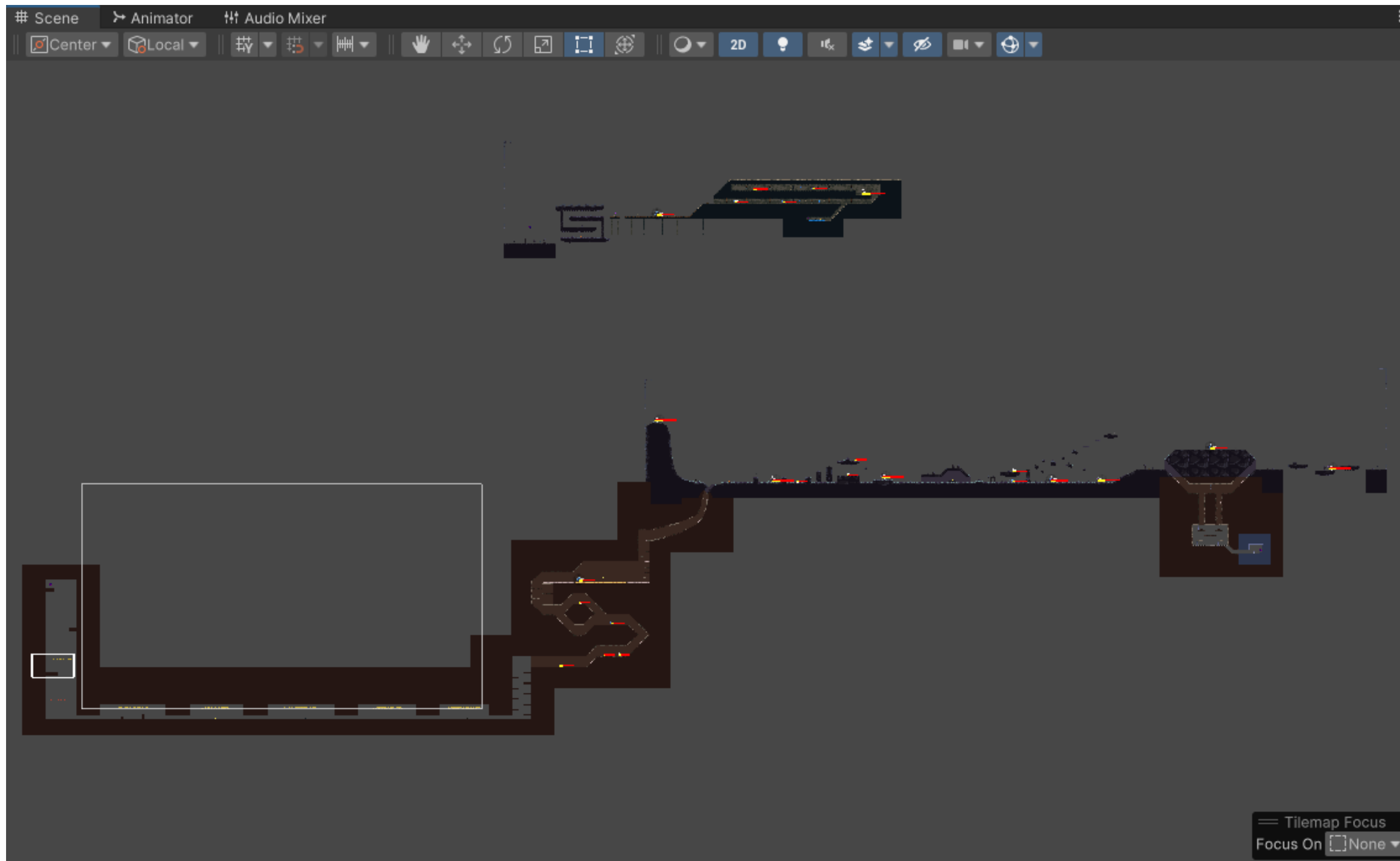
```
public class GameData  
{  
    Currency  
  
    #region Ability  
    public bool canWallSlide;  
    public bool canDash;  
    public bool canDoubleJump;  
    public bool canThrowSword;  
    public bool canFireBall;  
    public bool canIceBall;  
    public bool canBlackhole;  
    #endregion  
  
    Stats  
  
    Inventory  
  
    CheckPoints  
  
    Settings  
  
    1 个引用  
    public GameData()  
    //构造函数  
    {  
        Currency  
  
        Ability  
  
        Stats  
  
        Inventory  
  
        CheckPoints  
  
        Settings  
    }  
}
```

Data Structure

3 Systems Explanation

3.7 Game Scene Build

Building the game scenes will **use all systems we designed** for better gameplay experience



Originality & Innovation

04

4 Originality & Innovation

4.1 Originality In This Project

90%+ codes are our original work, with large amount of comments on the source scripts
Art resources mainly comes from itch.io, we draw some of the sprites using **Aseprite**

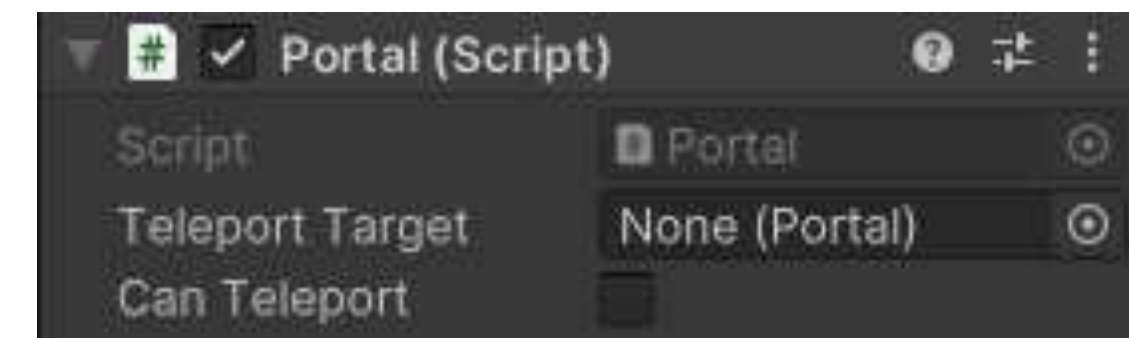
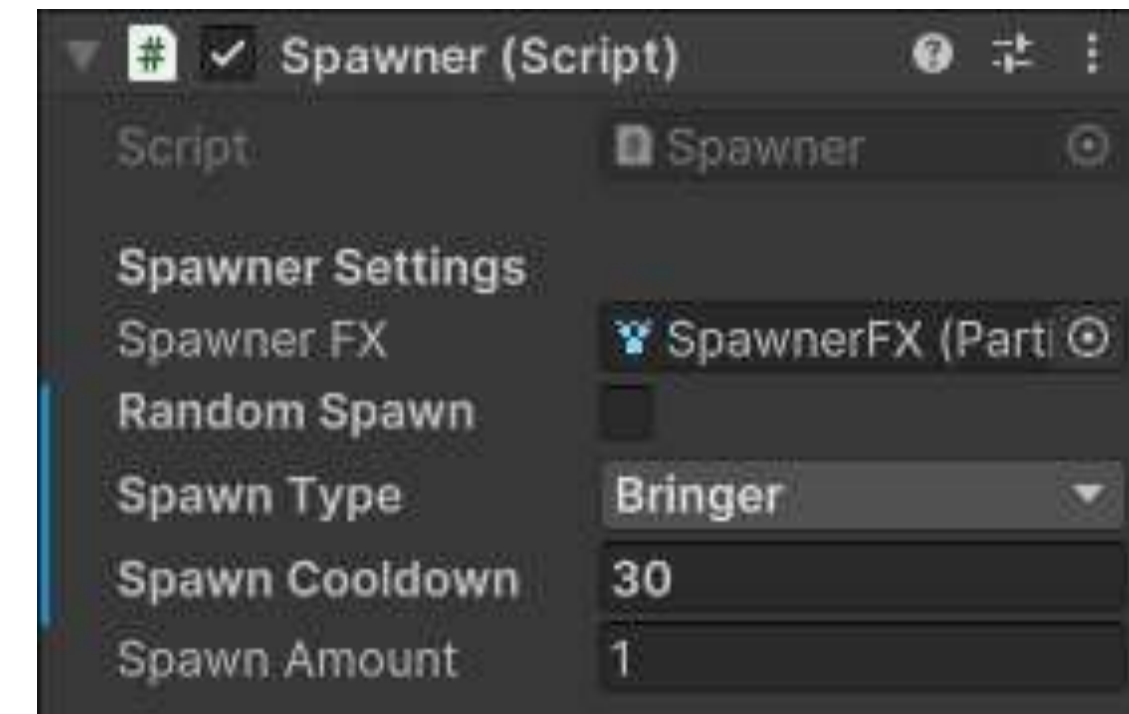


4.2 Innovation In This Project

We **integrate elements** of different game types in our design, for example the idea of jukebox, spawner and portal from sandbox game, and stats and buff system from souls games

We write the project scripts depending on **pure environment** of Unity2D, which makes it clean and uneasy to crash

We implemented various functional **interfaces** and designed the inheritance structure carefully, which builds up various **systems** that is **highly extensible**



Work Allocation

05

5 Work Allocation

5.1 Team Leader

Zong Jichen: Build and Test Systems including Entity Behavior & Skill & Statistics & Buff & Visual Effect, Users Interface, Inventory, Interact, Audio, Save&Load

5.2 Team Members

Hu Yang: Part of Enemies Implementation

Tian Yucheng: Part of Player Skills Implementation

Qiu Yijia: Level Scene Build

What We Learned

06

6 What We Learned

6.1 Control The Versions Under Standards

The project repo link:

<https://github.com/WhythZ/Metroidvania>

I actually have met several fatal crashes and **files loss**, git helps me survive

I use **github** to cooperate remotely with my teammates with their pull request

The screenshot shows the GitHub repository page for 'Metroidvania' by user 'WhythZ'. The repository is public and has 4 stars, 3 forks, and 1 watcher. The main content area displays a file tree for the 'master' branch, listing folders like 'Assets', 'Packages', and 'ProjectSettings', and files like '.gitignore', '.vsconfig', 'LICENSE', 'README.md', and 'ignore.conf'. The 'README' file is selected, showing the repository name 'Metroidvania', the license 'MIT license', and an 'About' section describing it as a 'Metroidvania game prototype developed by Unity2D'. The right sidebar contains sections for 'About', 'Releases', 'Packages', 'Contributors' (listing WhythZ, Elaina-Official, ttyclear, and NSPoison), and 'Languages' (showing C# at 73.6% and ShaderLab at 22.5%).

6 What We Learned

6.2 Keep Optimizing & Rebuilding Codes Structure

Pic1: Remove repetitive members after skill system's establishment

Pic2(a): Optimize damage value calculating magic

Pic2(b): Remove "Ailment" system from "Entity" class script and rebuild them as "Buff" class system

```
public class Player : Entity
{
    States
    Components
    Default
    Movement
    Jump

    #region Dash
    [Header("Dash Info")]
    //冲刺时间默认0.2秒, 即移动速度乘上dashSpeed倍率的持续时长
    public float dashDuration = 0.2f;
    //冲刺速度要比moveSpeed大, 不然不叫冲刺了, 默认为26
    public float dashSpeed = 26;
    //在空中冲刺过一次后, 即使冷却时间到了也不能第二次冲刺
    2 个引用
    public bool canDash { get; private set; } = true;

    //有了PlayerSkillManager管理归属Skill父类的DashSkill, 其内自有相关可调用内容, 故无需下列变量
    //冷却时间长度
    //public float dashCooldown = 0.6f;
    //冷却时间计时器
    //private float dashCooldownTimer;
    #endregion
}
```

Pic1

```
* 2d6f595 6.06 set up the guiding scene
* c4da9c7 6.06 add fire & ice ball skill
* 144e6ef CLEAN-UP: REBUILD ENTITY'S BUFF & ATTACK SYSTEM b
* 2e6731c 6.05 complete shocked ailment fx
* bf659a1 6.05 fix checkpoint logic
* 3ad535a 6.05 create spawner
* a6b6804 6.04 add a arena
* 19e2be6 6.04 add split slimes
* c15e925 6.04 add hit fx and sfx
* 335934a 6.04 fix logic of die function
* 446c8aa 6.03 optimize stat slot and player sprites
* e304c73 6.03 add ailments particle fx
* 7de6948 6.03 optimize damage logic & fix player anim a
* 2e9d3dd 6.02 add player stats to savefile and its interface
* dd324ee 6.02 audio cleanup
```

Pic2

6 What We Learned

6.3 Debug Patiently & Efficiently

```
public class PlayerPrimaryAttack : PlayerState
{
    [ComboSettings]
    1 个引用
    public PlayerPrimaryAttack(Player _player, PlayerStateMachine _stateMachine, string _animBoolName) : base(
    {
    }

    14 个引用
    public override void Enter()
    {
        base.Enter();

        //此音效转移至PlayerAnimationTriggers.cs的AttackDamageTrigger()函数处触发
        //触发攻击音效
        //Audio_Manager.instance.StartPlaySFX(0, null);

        #region ComboCounter
        //如果超出了连招最大个数，则归1（即下一次攻击回归第一招攻击）
        //如果攻击之间间隔太久（超出comboRefreshDuration），则会重新从第一招开始攻击
        if (comboCounter > 3 || (lastTimeAttack + comboRefreshDuration < Time.time))
        {
            comboCounter = 1;
        }
        //Debug.Log(comboCounter);

        //这段代码必须放在下面，保证如果上面归1了这边也能接收到
        //把AttackStack内的comboCounter和Animator内的对应Parameter链接起来
        player.anim.SetInteger("ComboCounter", comboCounter);
        #endregion

    }
    [AttackDetails]
}
```

The image features a white background with decorative blue geometric shapes in the corners. These shapes are composed of overlapping squares and rectangles, creating a modern, abstract pattern. The text "Thanks For Watching!" is centered in a bold, black, sans-serif font.

Thanks For Watching!