7.Unity 基于Flux制作Act技能编辑器

2022年11月01日 02:53 1451阅读 · 37喜欢 · 5评论

机智的小草yns
粉丝：1152 文章：17

相关视频：

做了个Act技能编辑器-初版【Unity】

8792 52

02:59 视频 UP 机智的小草yns

萌妹击剑2 Unity-Act练习

3201 24

03:53 视频 UP 机智的小草yns

Flux是一个类似于TimeLine的轨道动画编辑插件，看过Flux的代码，写得真好，拿来用了。然后再凭借之前写的战斗demo经验做出的Act技能编辑器。目前网上关于技能编辑器的资料比较少，花了很久的时间才做得有点模样，自己摸出来的思路可能不算特别好，给大家参考下。（大概半年后会开源）

技能编辑器分3部分:

1.技能预览, Flux在预览部分基本都做好了,用法和TimeLine类似。

2.技能数据保存。需要将Flux中的序列事件信息保存起来，给技能执行器用。

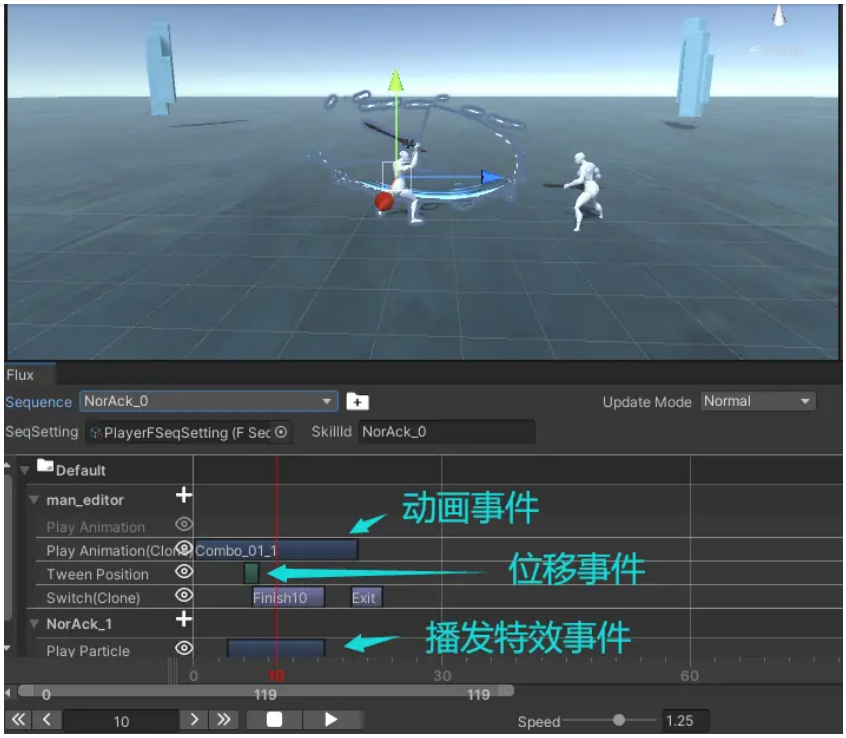
3.技能事件执行器。核心运行逻辑。

总的来说就是，用Flux做技能预览，然后序列化技能数据，保存帧时间序列。比如动作，特效，位移等等事件。最后是模仿Flux的序列(Sequence)做技能事件执行器。

1.技能预览

7.Unity 基于Flux制作Act技能编辑器

机智的小草yns + 关注



Flux的结构是，Sequence→Container→Timeline→Track→Event。一个父容器包含多个子容器。比如Sequence包含多个Container。以此类推，一个Track上放多个Event。

创建流程：创建一个Sequence，在默认容器Container（Default）上， 添加玩家TimeLine，加上动画Track，再加上一段AnimationClip作为动画事件。

同理加上位移事件和播放动画事件，一个简单的技能预览就算做完了，拖动时间轴就可以预览了。

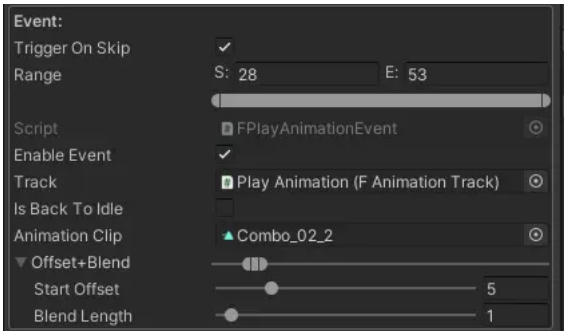
操作起来应该不难，参考我的视频 技能编辑器-初版 的操作。

其次是，想改造这个编辑器得有点编辑器基础。

2.数据保存

Event的主要结构：起始帧Start，结束帧End，以及内容。

如动画事件的内容为AnimationClip，StartOffset（动画偏移） 和 BlendLength（动画混合帧数）



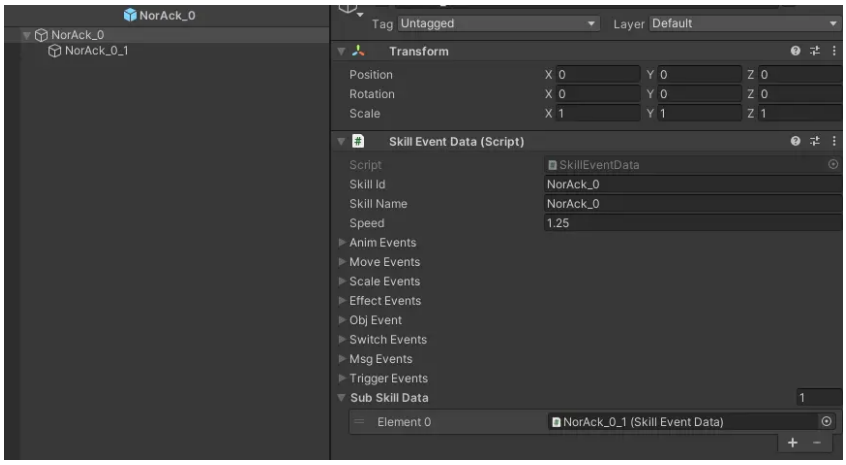
为了序列化数据，AnimationClip则保存Name即可。这样就完成一个事件的保存了。

读取编辑器Sequence数据：

7.Unity 基于Flux制作Act技能编辑器

机智的小草yns + 关注

```
(var _timeline in Sequence.Containers[0].Timelines)
{
   遍历 Timelines-> Tracks ->Evnets
   获取到所有的Evnet,分类保存为一组,
}
```



保存结果如上图所示，一个SkillEventData里有，动画事件，位移缩放事件，特效事件，伤害判定事件，子技能事件等等。

其中, SubSkillData是一个List<SkillEventData>，作为子技能们， 子技能和主技能的结构都是SkillEventData。同样拥有动画，位移等事件。一般用于生成的剑气特效。

按第一张图来说，玩家(man_editor)就是主技能，NorAck_1就是子技能。

（数据保存中名字变为： SkillId+_子技能序号 = “NorAck_0_1”）

NorAck_0 的内容是 动画/位移事件+子技能事件

NorAck_0 _1中的内容则为特效事件+伤害触发事件 +位移事件

这样一个技能的内容就算保存完毕了。

3.技能事件执行器，核心运行逻辑

由于Container一般只用一个。所以运行时的逻辑层简化为 :

Sequence→Timeline→Track→Event。

- Sequence为主技能和子技能的总和，
- Timeline为单个主/子技能，
- Track则对应类型EventList（如AnimEvents），
- Event则是最小的事件单位。

代码中, XCEventsRunner为Timeline ， XCEventOwner 为Track ， XCEvent 为 Event

释放技能则是生成一个 XCEventsRunner 和 它的子XCEventsRunner

技能的运行逻辑是:

所有XCEvent完成 -> 当前XCEventOwner 完成

->所有的XCEventOwner 完成，当前XCEventsRunner完成->

->所有的XCEventsRunner完成 -> 整个技能完成。

7.Unity 基于Flux制作Act技能编辑器

机智的小草yns + 关注

Timeline 层

XCEventsRunner 为 多个 XCEventOwner 的容器, 负责在Update中遍历正在范围内的XCEventOwner。

```
public class XCEventsRunner : MonoBehaviour
    {
        private List<XCEventOwner> ownerUpdateList = new List<XCEventOwner>();
        public List<XCEventsRunner> subRuners = new List<XCEventsRunner>();
        public SkillEventData skillData;
        public string BaseSkillId => skillData.skillId;
        public UnityEvent onFinishEvent = new UnityEvent();

        public bool isMainSkill = false;
        public bool isRun = false;
        public bool isBreak = false;
        public bool isFinish = false;

        public int updateCount;
        public float speed = 1;

        private void Update()
        {
            if (!isRun)
                return;
            updateCount = ownerUpdateList.Count;
            if (updateCount == 0)
            {
                Stop();
                DestroyAll();
                return;
            }
            //下面有RemoveAt,  需要倒序遍历
            for (int i = updateCount - 1; i >= 0; i--)
            {
                ownerUpdateList[i].OwnerUpdate(Time.deltaTime * speed);
                if (isBreak)
                {
                    //如果发送中断, 则未开始的owner 将设置为已完成,不再执行
                    if (!ownerUpdateList[i].IsStarted)
                    {
                        ownerUpdateList[i].HasFinished = true;
                    }
                }

                if (ownerUpdateList[i].HasFinished)
                {
                    ownerUpdateList.RemoveAt(i);
                }
            }
        }
    }
        public void Stop()
        {
            isBreak =true;
            foreach (var item in subRuners)
            {
                item.Stop();
            }
            Finish(true);
        }
        //Finish表示玩家脱离skill状态,而不影响skill的自我运行
        public void Finish(bool isForce = false)
        {
            if (!isFinish)
            {
                isFinish = true;
                onFinishEvent?.Invoke();
            }
```

7.Unity 基于Flux制作Act技能编辑器

机智的小草yns + 关注

Track 层

XCEventOwner 和 XCEventsRunner 类似，每帧Update遍历正在执行的Event，直到所有Event完成。

```
public class XCEventOwner
{
    //事件队列 按时间排序
    public List<XCEvent> _events = new List<XCEvent>();
    public XCAnimEvent curEvent;
    [NonSerialized]
    public XCEventsRunner selfRunner; //当前子技能
    public  Animator animator = null;
    public SkillOwner owner;

    public bool isRun = true;
    public bool HasFinished = false;
    private float _currentTime = -0.1f; //提前1点
    private int _currentFrame = 0;
    private int _currentEvent = 0; //记录队头

    public void Init(List<XCEvent> xcEvents, SkillOwner _owner)
    {
        owner = _owner;
        foreach (var item in xcEvents)
        {
            item.Init(_owner, this);
        }
        _events = xcEvents;
    }

    public void StartEvent()
    {
        isRun = true;
        HasFinished = false;
        IsStarted = false;
    }
    public void StopEvent()
    {
        isRun = false;
    }


    // Update is called once per frame
    public  void OwnerUpdate(float deltaTime)
    {
        if (isRun)
            OnEventUpdate(deltaTime);
    }

    public void SetCurrentTime(float curTime)
    {
        _currentTime = curTime;
    }

    public bool IsStarted = false;

    protected void OnEventUpdate(float deltaTime)
    {
        int limit = _events.Count;

        if (limit == 0)
        {
            HasFinished = true;
            //Debug.Log("yns  limit 0 ");
            return;
        }
```

7.Unity 基于Flux制作Act技能编辑器

机智的小草yns + 关注

```
            //delta不是稳定的
            //当前的1帧,指的的是动画帧,即1/30s,而不是update的一帧
            _currentFrame = Mathf.FloorToInt( _currentTime * XCSetting.FrameRate);
            //Debug.Log("yns  _curFrame " + _currentFrame + "_curTime "+ _currentTime +

            for (int i = _currentEvent; i < limit; i ++)
            {
                //还没开始时
                if (_currentFrame < _events[i].Start)
                {
                    if (_events[i].HasTriggered)
                        _events[i].OnReset();
                }
                else if (_currentFrame >= _events[i].Start && _currentFrame <= _events[i
                {
                    if (!_events[i].HasFinished)
                    {
                        IsStarted = true;
                        _events[i].UpdateEvent(_currentFrame, _currentTime - _events[i].

                    }
                }
                else
                {
                    //当 frame > end ,既已经完成 可以退出了
                    if (!_events[i].HasFinished && _events[i].HasTriggered)
                    {
                        _events[i].OnFinish();
                    }
                    //对于 frame > end 的事件不再检测

                    if(i == limit -1)
                    {
                        //Debug.Log("yns owner HasFinished " + _currentEvent);
                        HasFinished = true;
                    }
                    _currentEvent = Mathf.Clamp(i + 1, 0, limit - 1);
                }
            }
        }
    }
```

Event层

Event的生命周期有Init，Trigger，Update，Finish，Stop。

当技能帧 t ≥Start时，触发一次Init和Trigger。

当End≥ t ≥Start，则持续执行Update(t)

当t≥End，则是执行Finish和Stop。此时Event便被标记为Finished。

Event的核心代码：

```
//XCEvent （简化了部分无关数据）
[Serializable]
public class XCEvent
{
    private bool _hasTriggered = false;
    protected bool _hasFinished = false;
    public int Start
    public int End
    public float Length

          //SkillOwner 为自定义类， 里面放必要的传参，如玩家id，动画机等等
    public virtual void Init(SkillOwner owner, XCEventOwner eventOwner)
    {
        this.skillOwner = owner;
```

7.Unity 基于Flux制作Act技能编辑器

机智的小草yns ＋ 关注

```
    }
    public virtual void OnTrigger(float timeSinceTrigger)
    {
        _hasTriggered = true;
    }
    public void UpdateEvent(int frame, float timeSinceTrigger)
    {
        if (!HasTriggered)
        {
            OnTrigger(timeSinceTrigger);
        }
        //OnUpdateEvent用于子类重写
        OnUpdateEvent(frame, timeSinceTrigger);

        if (range.End <= frame)
        {
            _hasFinished = true;
            OnFinish();
        }
    }
    public virtual void OnFinish() { }
    public virtual void OnUpdateEvent(int frame, float timeSinceTrigger) { }
    public virtual void OnReset()
    {
        _hasFinished = false;
        _hasTriggered = false;
    }
}
```

然后通过继承XCEvent实现不同的事件效果

以动画事件为例：

```
public class XCAnimEvent : XCEvent
{
    private Animator _animator = null;
    private AnimationClip Clip;

    public int clipHash => Animator.StringToHash(eName);

    public float blenderLength = 0;
    public float startOffset = 0;
    public float speed = 1f;
    public bool isBackToIdle;

    //根据名字获取AnimationClip
    void SetClip()
    {
        foreach (var item in _animator.runtimeAnimatorController.animationClips)
        {
            if (item.name == eName)
            {
                Clip = item;
            }
        }
    }
    //初始化
    public override void Init(SkillOwner owner, XCEventOwner eventOwner)
    {
        base.Init(owner, eventOwner);
        //获取当前动画机
        if (_animator == null)
        {
            if (owner.isCustomObject)
            {
                _animator = owner.gameObj.GetComponentInChildren<Animator>();
            }
            else
```

7.Unity 基于Flux制作Act技能编辑器

机智的小草yns + 关注

```
            }
            SetClip();
            _animator.speed = speed;
        }
        //触发
        public override void OnTrigger(float timeSinceTrigger)
        {
            base.OnTrigger(timeSinceTrigger);
            if (Clip == null)
            {
                Debug.LogError("no clip " + eName);
                return;
            }
            //如果主技能被打断则不播放动画
            if(SelfRunner.isMainSkill && SelfRunner.isBreak)
            {
                return;
            }
           //播放动画 通过Clip.length获取动画长度
            _animator.CrossFade(clipHash, blenderLength / Clip.length, 0, startOffset /

            //修正偏差值
            if (timeSinceTrigger > 0)
            {
                _animator.Update(timeSinceTrigger - 0.001f);
            }
        }

        public override void OnFinish()
        {
            base.OnFinish();
        }
    }
```

位移事件为例：

```
[Serializable]
public class XCMoveEvent : XCLineEvent
{
    public CharacterController cc;
    public bool isStartDetal = false;
    public Vector3 startAmgle = Vector3.zero;
    public Vector3 startDetal = Vector3.zero;

    public override void Init(SkillOwner owner, XCEventOwner eventOwner)
    {
        base.Init(owner, eventOwner);
        cc = ownerTF.GetComponent<CharacterController>();
    }

    public override void OnTrigger(float timeSinceTrigger)
    {
        base.OnTrigger(timeSinceTrigger);
        if (isStartDetal)
        {
            ExcuteAddVec(startDetal);
        }
    }
    public override void OnUpdateEvent(int frame, float timeSinceTrigger)
    {
        base.OnUpdateEvent(frame, timeSinceTrigger);
        float t = timeSinceTrigger / LengthTime;

        var move = GetVec3Value(t) - GetVec3Value(lastTime);
        lastTime = t;
        ExcuteAddVec(move);
    }
```

7.Unity 基于Flux制作Act技能编辑器

机智的小草yns + 关注

```
            // DOVirtual.EasedValue(0, 1, t , Ease.OutQuad)
                //moveType表示不同曲线
            return EaseTool.GetVec3Value(startVec, endVec, moveType, t);
        }

        public override void ExcuteAddVec(Vector3 detalMove)
        {
            if (cc != null)
            {
                if (SelfRunner.isMainSkill && SelfRunner.isBreak)
                {
                    _hasFinished = true;
                    return;
                }
                cc.Move(cc.transform.TransformDirection(detalMove));

            }
            else
            {
                ownerTF.TransformDirection(detalMove);
                ownerTF.Translate(detalMove, Space.Self);
            }
        }
    }
```


游戏开发 小草 Unity Act 知识分享 技能编辑器

推荐文章

更多精彩内容 >

AI孙燕姿救不了华语乐坛

原创 格子 X博士"多年以后，孙燕姿老师站在领奖台上，准会想起网友训练出第一个AI孙燕姿模型的那个遥远的下午。" 一位B站…

X博士官方频道 日常 9750 126 26

评论

全部评论 **按时间排序**