

# UNITY TIMELINE

Preview 58

Attack Sequence

The image shows a timeline for an "Attack Sequence" from 0 to 150 seconds. A vertical white line marks the current time at approximately 58 seconds. The tracks and their events are as follows:

Track	Event	Start Time (s)	End Time (s)
Attacker1	Idle	0	30
Attacker1	Crouch	30	40
Attacker1	Idle	40	50
Attacker1	Attack	50	70
Attacker1	Jump	70	80
Attacker1	Die	80	150
None (Audio M)	Crawl	30	40
None (Audio M)	Attack-Taunt	50	70
None (Audio M)	Grunt	70	80
None (Audio M)	Die Grunt	80	150
LookAt Playable	LookAt	0	150
Main Camera	(No event)	0	150
Master (NewAu)	Ambient	0	150
Control Track	Explosion	0	90
Squad Car	(No event)	0	150
Outpost Light	Active	0	65
Outpost Light	Active	125	150
Beam	Active	0	10
Beam	Active	20	30
Beam	Active	40	50
Beam	Active	70	150
Beam	Active	130	150

Зачем оно  
надо и что  
это дает ?

- Создание интерактивных катсцен
- Возможность редактирования без участия программиста
- Простота изменения готовых катсцен
- Возможность переиспользования
- Результат виден сразу без необходимости запуска приложения
- Быстрая итерация и более короткий цикл

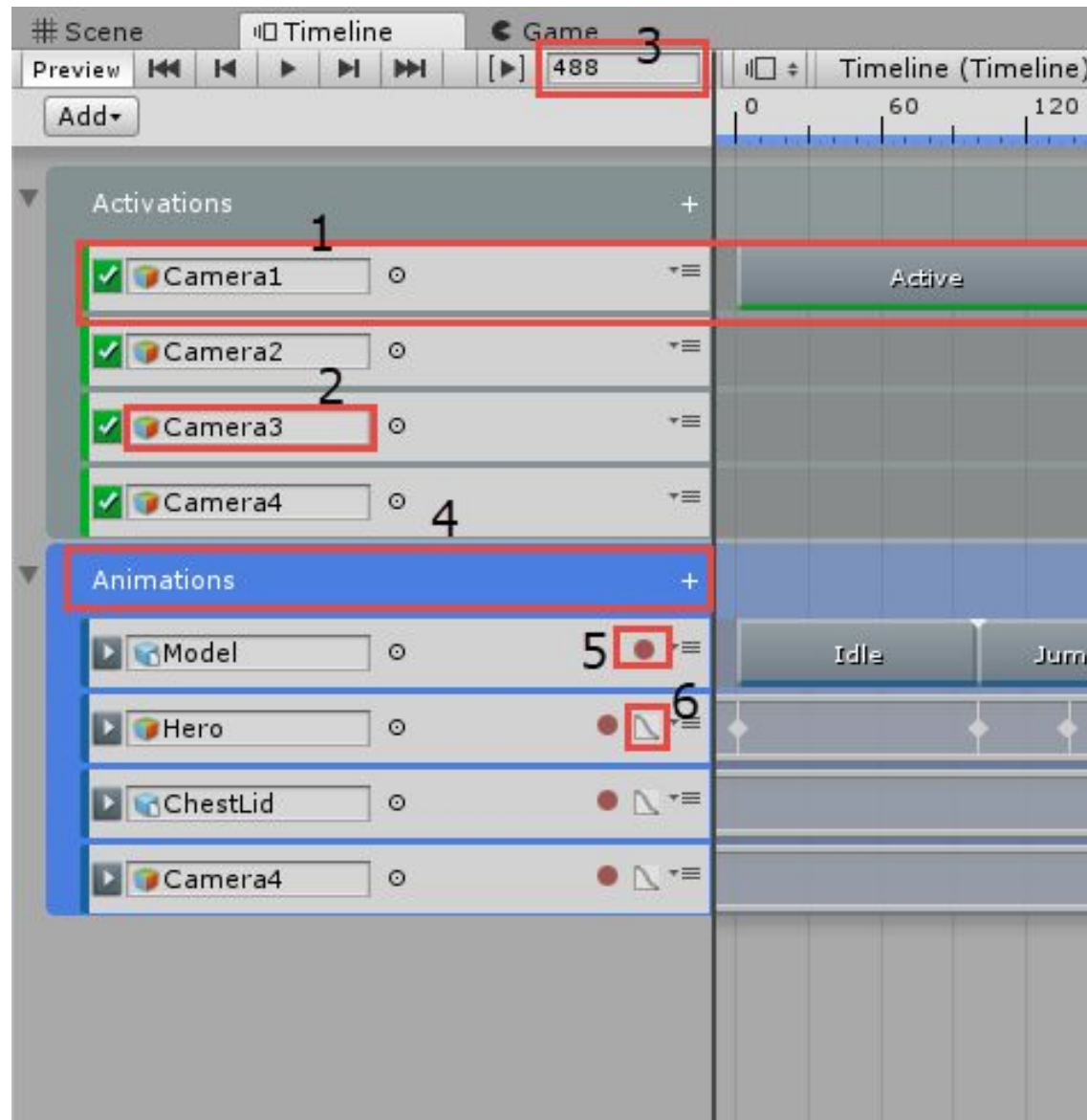
- Tweens
- Coroutines
- Scripting

# Альтернативы

Полностью готовые  
катсцены созданные  
усилиями художников  
мультипликаторов

# Из чего состоит визуальный редактор

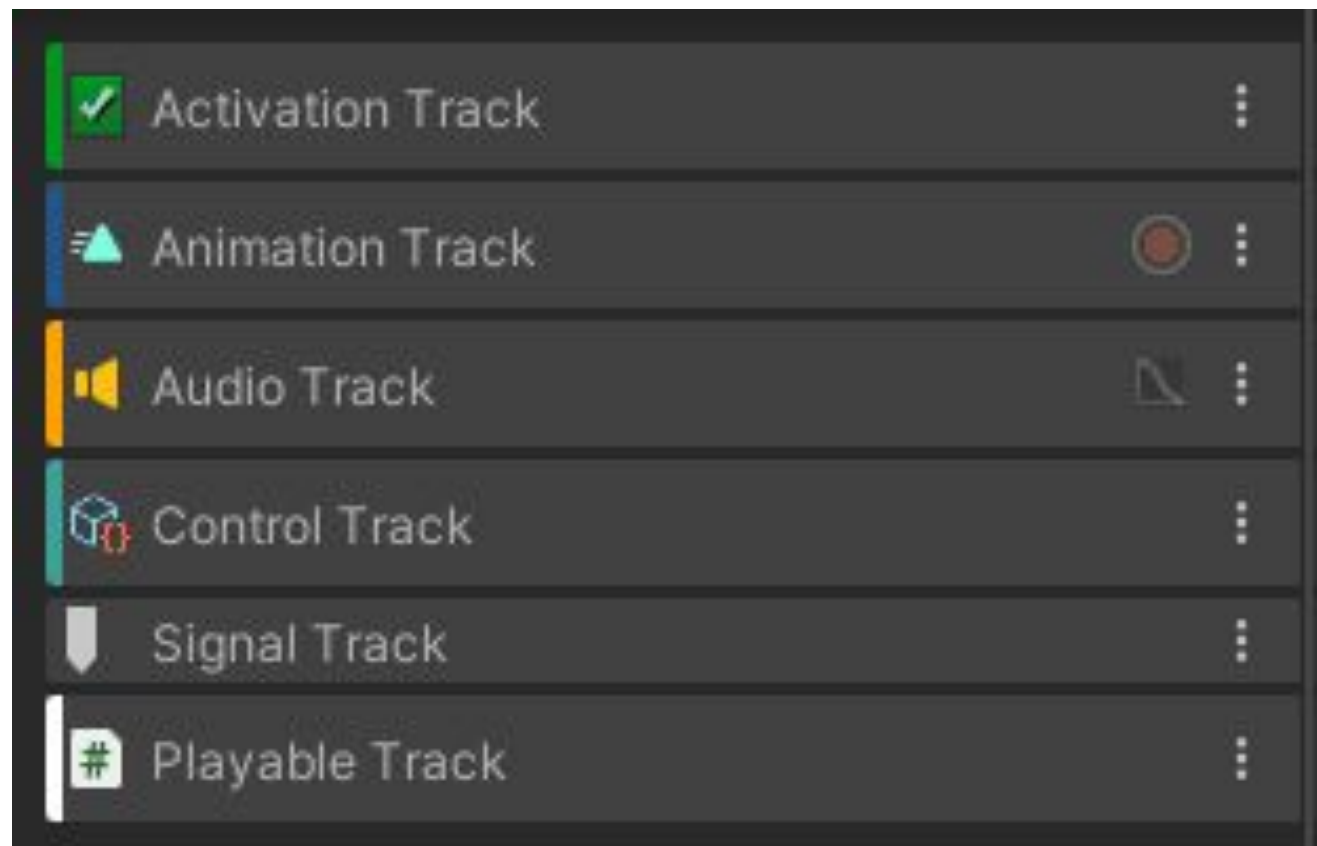
1. *Timeline Asset*
2. *Associated GameObject*
3. *Frame*
4. *Track Group*
5. *Record Button*
6. *Curves Icon*



# Playable director и Exposed Reference

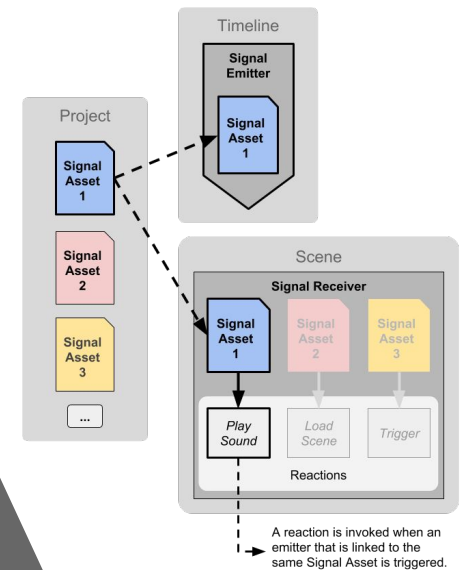
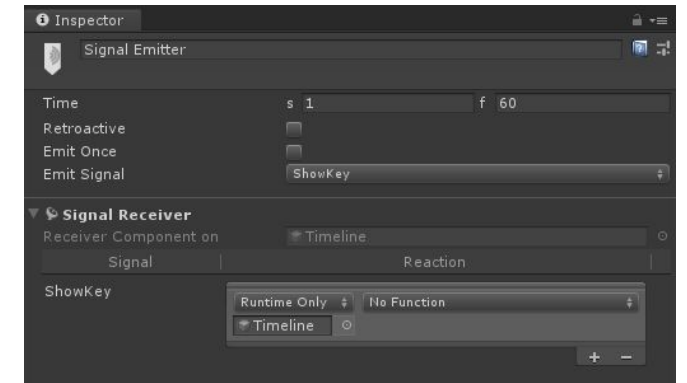
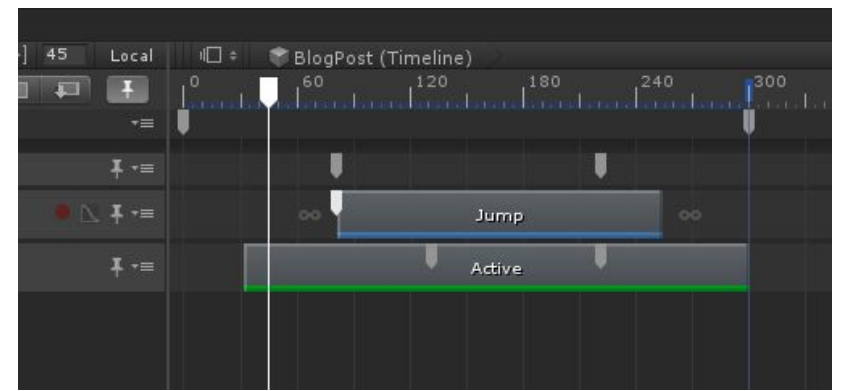
Timeline (Playable asset) это ассет и не может содержать прямые ссылки на объекты сцены. Поэтому в ассете создаются Exposed reference, в которые при исполнении с помощью Resolver (Playable director) будут проставлены необходимые объекты.

# Стандартны е треки Unity



# Signals

- Signal Asset - связующее звено между Emitter и Receiver. Может быть переиспользован в различных таймлайнах
- Signal Emitter - содержит ссылку на Signal Asset. На таймлайне визуализируется как маркер
- Signal Receiver - компонент со списком обработчик. В каждом обработчике есть ссылка на Signal Asset





# Custom Tracks

- Behaviour
- Clip
- Track
- MixerBehaviour

# Behaviour

---

Кастомная логика. Вообще должен должен реализовывать

```
public override void ProcessFrame(Playable  
playable, FrameData info, object playerData)
```

Но на скрине его нет, так как ее реализует mixer

```
2 4 usages  Maksim Sharov  1 exposing API  
public class UIScaleBehaviour : PlayableBehaviour {  
    public AnimationCurve scaleCurve = AnimationCurve.Linear( timeStart: 0f, valueStart: 0f, timeEnd: 1f, valueEnd: 1f);  Serializable  
    public Vector3 scaleStart;  Serializable  
    public Vector3 scaleEnd;  Serializable  
  
    1 usage  Maksim Sharov  
    public float EvaluateCurve(AnimationCurve curve, float time) {  
        if (curve.IsNormalized()) {  
            return curve.Evaluate(time);  
        }  
  
        Debug.LogError( message: "Curve is not normalised. Curve must start at 0,0 and end at 1,1.");  
        return 0f;  
    }  
}
```

# Clip

---

Содержит данные и сериализуется в Timeline Asset

Должна реализовывать метод

```
public override Playable  
CreatePlayable(PlayableGraph graph,  
GameObject owner)
```

```
⌕ No asset usages 1 usage 2 Maksim Sharov  
public class UIScaleClip : PlayableAsset, ITimelineClipAsset {  
    public UIScaleBehaviour template = new UIScaleBehaviour(); ⌕ Serializable  
  
    ⌕ Maksim Sharov  
    public ClipCaps clipCaps => ClipCaps.Blending;  
  
    ⌕ Maksim Sharov  
    public override Playable CreatePlayable(PlayableGraph graph, GameObject owner) {  
        var playable = ScriptPlayable<UIScaleBehaviour>.Create(graph, template);  
        return playable;  
    }  
}
```

# Track

Может хранить в себе ссылку на GameObject, Component или Asset.

- **TrackClipType** обозначает с каким типом PlayableAsset будет работать данный трек.
- **TrackBindingType** указывает на то, с чем можно будет в дальнейшем связать данный трек.

```
[TrackColor(r: 0.855f, g: 0.8623f, b: 0.870f)]
[TrackClipType(typeof(UIScaleClip))]
[TrackBindingType(typeof(RectTransform))]
No asset usages Maksim Sharov
public class UIScaleTrack : TrackAsset {
    Maksim Sharov
    public override Playable CreateTrackMixer(PlayableGraph graph, GameObject go, int inputCount) {
        return ScriptPlayable<UIScaleMixerBehaviour>.Create(graph, inputCount);
    }
    Maksim Sharov
    public override void GatherProperties(PlayableDirector director, IPropertyCollector driver) {
#if UNITY_EDITOR
        var comp = director.GetGenericBinding(key: this) as RectTransform;
        if (comp == null) {
            return;
        }
        var so = new SerializedObject(comp);
        var iter :SerializedProperty = so.GetIterator();
        while (iter.NextVisible(enterChildren: true)) {
            if (iter.hasVisibleChildren) {
                continue;
            }
            driver.AddFromName<RectTransform>(comp.gameObject, iter.propertyPath);
        }
#endif
        base.GatherProperties(director, driver);
    }
}
```

# MixerBehaviour

Позволяет делать перекрытие треков на timeline. Переопределяет поведение Behaviour.

```
public class UIScaleMixerBehaviour : PlayableBehaviour {
    ↳ Maksim Sharov
    public override void ProcessFrame(Playable playable, FrameData info, object playerData) {
        var trackBinding = playerData as Transform;

        if (trackBinding == null) {
            return;
        }

        ProcessScale(trackBinding, playable);
    }

    1 usage ↳ Maksim Sharov
    private void ProcessScale(Transform transform, Playable playable) {
        var inputCount :int = playable.GetInputCount();
        var totalWeight = 0f;

        var blendedScale :Vector3 = Vector3.zero;
        var defaultScale :Vector3 = transform.localScale;

        for (var i = 0; i < inputCount; i++) {
            var playableInput = (ScriptPlayable<UIScaleBehaviour>)playable.GetInput(i);
            var input :UIScaleBehaviour = playableInput.GetBehaviour();

            var inputWeight :float = playable.GetInputWeight(i);
            totalWeight += inputWeight;
            var normalisedTime = (float)(playableInput.GetTime() / playableInput.GetDuration());
            var value :float = input.EvaluateCurve(input.scaleCurve, normalisedTime);
            blendedScale += (input.scaleStart + (input.scaleEnd - input.scaleStart) * value) * inputWeight;
        }

        blendedScale += defaultScale * (1f - totalWeight);
        transform.localScale = blendedScale;
    }
}
```