

Obsidian API Tutorial #6 – Animation End Events – v0.3.0+

<< Previous (<http://www.dabigjoe.com/obsidian-suite/tutorials/5-entity-ai/>) All tutorials (<http://www.dabigjoe.com/obsidian-suite/tutorials/>) Next >> (<http://www.dabigjoe.com/obsidian-suite/tutorials/6a-more-animation-events>)

Let's start using animation events

The Obsidian API provides a few handy tricks to deal with common animation events. In this tutorial, we're going to look at how we setup an animation event listener and use animation end events.

Prerequisites

At least one working entity and animation.

Render vs State

Before we really get going, I want to cover some background stuff regarding how the API is built and how I think about using it.

There are basically two things to consider when using the API. One is state – the internal workings and what the entity is doing. The other is rendering – what is actually drawn on the screen and seen by the user. The API is designed so that the rendering always represents the state. It should always be seen as a one way street: state feeds into rendering and not the other way around.

However... It can sometimes be useful to have things happen at certain stages of an animation, for example when it starts, ends or somewhere in the middle. That's what we're going to look at now.

What goes in the state?

Well, the state contains all the information about an entity that we need to determine what it is doing. So far, we've looked at stuff like entity AI and walking, but what about something that doesn't come into those categories? An example I've come up with for the Saiga is a 'call', like a cow's moo or something. This doesn't have any real effect, other than maybe playing a sound. It's basically just a little extra idle animation to make the entity seem more real.

Adding to the Entity State

You may have noticed methods in Entity classes such as `isSleeping` or `isSneaking` – these are all representations of the state of the entity. For the saiga, I'm going to add a calling state. It requires four parts: the variable, the getter and setter (`setCalling` and `isCalling`), and some logic.

```
private boolean calling = false;
```

```
public void setCalling(boolean calling) {
    this.calling = calling;
}
```

```
public boolean isCalling() {
    return calling;
}
```

The logic happens in the `onEntityUpdate` method. This is a great place to put any entity logic as it is called every game tick. In my example, there's a random chance that the Saiga will start calling if it isn't already.

```
@Override
public void onEntityUpdate() {
    super.onEntityUpdate();
    if(!calling && this.rand.nextFloat() < 0.01)
        calling = true;
}
```

What happens now?

Let's try adding the Saiga call animation. In the `CommonProxy`, just register an animation as normal. The Saiga call has slightly higher priority than the idle animation, but all the other animations are more important. For its `isActiveFunction`, we just need to check our new state:

```
IsActiveFunction isSaigaCalling = (entity) -> {
    return entity instanceof EntitySaiga ? ((EntitySaiga) entity).isCalling() : false;
};
```

```
AnimationRegistry.registerAnimation("saiga", "call", saigaCall, 40, false, isSaigaCalling);
```

Let's try this out! Uh-oh, it doesn't work as we want... The Saigas seem fine until they start, then they won't stop! This is why we need the end event.

Animation Event Handler

The API creates animation events when certain things happen, like animations starting or ending. We need to create a new class that listens for these events. I've called mine `AnimationEventHandler`.

To listen to an `AnimationEvent`, we mark a method with an annotation:

```
@AnimationEventListener(type = AnimationEventType.END, entityName="Saiga", animationName="Call")
```

```
public void onSaigaCallEnd(AnimationEvent event) {
```

The type parameter defines what we're listening for – start, end etc.. The `entityName` parameter restricts what the method will listen for – if we set it to Saiga the method will only be fired for Saiga animation events. If left blank, it will respond to animation events for all entities. Similarly, the `animationName` event can restrict the method to animations with a certain name.

In my example, I only want the method `onSaigaCallEnd` to be fired when the Saiga has finished the call animation, therefore the type is `end`, the `entityName` is `Saiga` and the `animationName` is `call`.

The method has one parameter – the event that was fired. From this, we can get the entity that it was fired for, allowing us to change its state. In my case, I want to stop the entity from calling, so I set its calling variable to false:

```
public void onSaigaCallEnd(AnimationEvent event) {
    ((EntitySaiga) event.entity).setCalling(false);
}
```

One last thing

We need to register our new `AnimationEventHandler` with the ObsidianAPI event bus, so it knows to actually send events to it. This is done in the main mod class, in the `load FMLInitializationEvent` method:

```
ObsidianAPI.EVENT_BUS.register(new AnimationEventHandler());
```

That should've fixed it

So now, when a Saiga's animation finishes, it updates its own state so that it is no longer calling, thus the animation won't play again until the `onEntityUpdate` method gets lucky and sets calling to true again.

That's a quick overview of how to get an animation event handler setup, next time we'll look at doing more with it!

<< Previous (<http://www.dabigjoe.com/obsidian-suite/tutorials/5-entity-ai/>) All tutorials (<http://www.dabigjoe.com/obsidian-suite/tutorials/>) Next >> (<http://www.dabigjoe.com/obsidian-suite/tutorials/6a-more-animation-events>)

Edited files

EntitySaiga

```
public class EntitySaiga extends EntityCreature implements IEntityAnimated
{

    private boolean calling = false;

    public EntitySaiga(World world)
    {
        super(world);
        this.tasks.taskEntries.clear();
        this.tasks.addTask(0, new EntityAIWander(this, 1.0D));
        this.tasks.addTask(1, new EntityAIEat(this));
    }

    @Override
    protected boolean isAIEnabled()
    {
        return true;
    }

    @Override
    protected void applyEntityAttributes()
    {
        super.applyEntityAttributes();
        this.getEntityAttribute(SharedMonsterAttributes.maxHealth).setBaseValue(10.0D);
        this.getEntityAttribute(SharedMonsterAttributes.movementSpeed).setBaseValue(0.18D);
    }

    @Override
    public boolean isMoving() {
        return limbSwingAmount > 0.02F;
    }

    @Override
    public void onEntityUpdate() {
        super.onEntityUpdate();
        if(!calling && this.rand.nextFloat() < 0.01)
            calling = true;
    }

    public void setCalling(boolean calling) {
        this.calling = calling;
    }
}
```

```

    }

    public boolean isCalling() {
        return calling;
    }
}

AnimationEventHandler

public class CommonProxy {

    private ResourceLocation saigaWalk = new ResourceLocation("mod_api_tutorial:animations/Walk.oba");
    private ResourceLocation saigaIdle = new ResourceLocation("mod_api_tutorial:animations/SaigaIdle.oba");
    private ResourceLocation saigaEat = new ResourceLocation("mod_api_tutorial:animations/SaigaEat.oba");
    private ResourceLocation saigaCall = new ResourceLocation("mod_api_tutorial:animations/SaigaCall.oba");

    public void init()
    {
        ModEntities.registerEntities();
        registerRendering();
        registerAnimations();
    }

    public void registerRendering() {}

    public void registerAnimations() {
        AnimationRegistry.init();

        IsActiveFunction isWalking = (entity) -> {
            return ObsidianAPIUtil.isEntityMoving(entity);
        };

        IsActiveFunction returnTrue = (entity) -> {
            return true;
        };

        IsActiveFunction isSaigaCalling = (entity) -> {
            return entity instanceof EntitySaiga ? ((EntitySaiga) entity).isCalling() : false;
        };

        AnimationRegistry.registerEntity(EntitySaiga.class, "saiga");
        AnimationRegistry.registerAnimation("saiga", "walk", saigaWalk, 0, true, isWalking);
        AnimationRegistry.registerAnimation("saiga", EntityAIEat.name, new AIAnimationWrapper(EntityAIEat.name, saigaEat, 10, true));
        AnimationRegistry.registerAnimation("saiga", "call", saigaCall, 40, false, isSaigaCalling);
        AnimationRegistry.registerAnimation("saiga", "idle", saigaIdle, 50, true, returnTrue);
    }
}

AnimationEventHandler

public class AnimationEventHandler {

    @AnimationEventListener(type = AnimationEventType.END, entityName="Saiga", animationName="Call")
    public void onSaigaCallEnd(AnimationEvent event) {
        ((EntitySaiga) event.entity).setCalling(false);
    }
}

Main mod class

@Mod(modid = "ApiTutorial")
public class ModAPITutorial {

    @Mod.Instance("ApiTutorial")
    public static ModAPITutorial instance;

    @SidedProxy(serverSide = "apiTutorial.CommonProxy", clientSide = "apiTutorial.ClientProxy")
    public static CommonProxy proxy;


    @Mod.EventHandler
    public void init(FMLInitializationEvent event)
    {
        instance = this;
    }
}

```

```
proxy.init();

AnimationNetworkHandler.init();
}

@EventHandler
public void load(FMLInitializationEvent event) {
    MinecraftForge.EVENT_BUS.register(new ObsidianEventHandler());
    ObsidianAPI.EVENT_BUS.register(new AnimationEventHandler());
}
}
```

 Edit (<http://www.dabigjoe.com/wp-admin/post.php?post=769&action=edit>)

LEAVE A REPLY

Logged in as dabigjoe (<http://www.dabigjoe.com/wp-admin/profile.php>). Log out? (http://www.dabigjoe.com/wp-login.php?action=logout&redirect_to=http%3A%2F%2Fwww.dabigjoe.com%2Fobsidian-suite%2Ftutorials%2F6-animation-end-events%2F&wnonce=6e8ffaa668)

Comment