```java
package net.mcreator.direwolfs;

import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.LogManager;

import net.minecraftforge.network.simple.SimpleChannel;
import net.minecraftforge.network.NetworkRegistry;
import net.minecraftforge.network.NetworkEvent;
import net.minecraftforge.fml.util.thread.SidedThreadGroups;
import net.minecraftforge.fml.javafmlmod.FMLJavaModLoadingContext;
import net.minecraftforge.fml.common.Mod;
import net.minecraftforge.eventbus.api.SubscribeEvent;
import net.minecraftforge.eventbus.api.IEventBus;
import net.minecraftforge.event.TickEvent;
import net.minecraftforge.common.MinecraftForge;

import net.minecraft.resources.ResourceLocation;
import net.minecraft.network.FriendlyByteBuf;

import net.mcreator.direwolfs.init.DirewolfsModTabs;
import net.mcreator.direwolfs.init.DirewolfsModItems;
import net.mcreator.direwolfs.init.DirewolfsModEntities;

import java.util.function.Supplier;
import java.util.function.Function;
import java.util.function.BiConsumer;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.List;
import java.util.Collection;
import java.util.ArrayList;
import java.util.AbstractMap;

@Mod("direwolfs")
public class DirewolfsMod {
    public static final Logger LOGGER = LogManager.getLogger(DirewolfsMod.class);
    public static final String MODID = "direwolfs";

    public DirewolfsMod() {
        // Start of user code block mod constructor
```

```
        // End of user code block mod constructor
        MinecraftForge.EVENT_BUS.register(this);
        IEventBus bus = FMLJavaModLoadingContext.get().getModEventBus();

        DirewolfsModItems.REGISTRY.register(bus);
        DirewolfsModEntities.REGISTRY.register(bus);

        DirewolfsModTabs.REGISTRY.register(bus);

        // Start of user code block mod init
        // End of user code block mod init
    }

    // Start of user code block mod methods
    // End of user code block mod methods
    private static final String PROTOCOL_VERSION = "1";
    public static final SimpleChannel PACKET_HANDLER =
NetworkRegistry.newSimpleChannel(new ResourceLocation(MODID, MODID), () ->
PROTOCOL_VERSION, PROTOCOL_VERSION::equals, PROTOCOL_VERSION::equals);
    private static int messageID = 0;

    public static <T> void addNetworkMessage(Class<T> messageType, BiConsumer<T,
FriendlyByteBuf> encoder, Function<FriendlyByteBuf, T> decoder, BiConsumer<T,
Supplier<NetworkEvent.Context>> messageConsumer) {
        PACKET_HANDLER.registerMessage(messageID, messageType, encoder, decoder,
messageConsumer);
        messageID++;
    }

    private static final Collection<AbstractMap.SimpleEntry<Runnable, Integer>>
workQueue = new ConcurrentLinkedQueue<>();

    public static void queueServerWork(int tick, Runnable action) {
        if (Thread.currentThread().getThreadGroup() == SidedThreadGroups.SERVER)
            workQueue.add(new AbstractMap.SimpleEntry<>(action, tick));
    }

    @SubscribeEvent
    public void tick(TickEvent.ServerTickEvent event) {
```

```
        if (event.phase == TickEvent.Phase.END) {
            List<AbstractMap.SimpleEntry<Runnable, Integer>> actions = new
ArrayList<>();
            workQueue.forEach(work -> {
                work.setValue(work.getValue() - 1);
                if (work.getValue() == 0)
                    actions.add(work);
            });
            actions.forEach(e -> e.getKey().run());
            workQueue.removeAll(actions);
        }
    }
}
```

// Made with Blockbench 4.11.2
// Exported for Minecraft version 1.17 or later with Mojang mappings
// Paste this class into your mod and generate all required imports

public class BadDog<T extends Entity> extends EntityModel<T> {
	// This layer location should be baked with EntityRendererProvider.Context in the entity renderer and passed into this model's constructor
	public static final ModelLayerLocation LAYER_LOCATION = new ModelLayerLocation(new ResourceLocation("modid", "baddog"), "main");
	private final ModelPart head;
	private final ModelPart body;
	private final ModelPart upperBody;
	private final ModelPart leg0;
	private final ModelPart leg1;
	private final ModelPart leg2;
	private final ModelPart leg3;
	private final ModelPart tail;

	public BadDog(ModelPart root) {

```
        this.head = root.getChild("head");
        this.body = root.getChild("body");
        this.upperBody = root.getChild("upperBody");
        this.leg0 = root.getChild("leg0");
        this.leg1 = root.getChild("leg1");
        this.leg2 = root.getChild("leg2");
        this.leg3 = root.getChild("leg3");
        this.tail = root.getChild("tail");
        }

        public static LayerDefinition createBodyLayer() {
        MeshDefinition meshdefinition = new MeshDefinition();
        PartDefinition partdefinition = meshdefinition.getRoot();

        PartDefinition head = partdefinition.addOrReplaceChild("head",
CubeListBuilder.create().texOffs(24, 16).addBox(-3.0F, -3.0F, -2.0F, 6.0F, 6.0F, 4.0F, new
CubeDeformation(0.0F))
        .texOffs(30, 0).addBox(-3.0F, -5.0F, 0.0F, 2.0F, 2.0F, 1.0F, new CubeDeformation(0.0F))
        .texOffs(24, 26).addBox(-1.5F, -0.0156F, -5.0F, 3.0F, 3.0F, 4.0F, new
CubeDeformation(0.0F)), PartPose.offset(-1.0F, 13.5F, -7.0F));

        PartDefinition head_r1 = head.addOrReplaceChild("head_r1",
CubeListBuilder.create().texOffs(30, 0).addBox(-1.0F, -1.0F, -0.5F, 2.0F, 2.0F, 1.0F, new
CubeDeformation(0.0F)), PartPose.offsetAndRotation(2.0F, -4.0F, 0.5F, 0.6109F, -0.9163F,
0.0F));

        PartDefinition body = partdefinition.addOrReplaceChild("body",
CubeListBuilder.create().texOffs(0, 16).addBox(-4.0F, 1.0F, -3.0F, 6.0F, 6.0F, 6.0F, new
CubeDeformation(0.0F)), PartPose.offsetAndRotation(0.0F, 14.0F, 2.0F, 1.5708F, 0.0F, 0.0F));

        PartDefinition upperBody = partdefinition.addOrReplaceChild("upperBody",
CubeListBuilder.create().texOffs(0, 0).addBox(-4.0F, -8.0F, -3.0F, 8.0F, 9.0F, 7.0F, new
CubeDeformation(0.0F)), PartPose.offsetAndRotation(-1.0F, 14.0F, 2.0F, 1.5708F, 0.0F, 0.0F));

        PartDefinition leg0 = partdefinition.addOrReplaceChild("leg0",
CubeListBuilder.create().texOffs(0, 28).addBox(-1.0F, 0.0F, -1.0F, 2.0F, 8.0F, 2.0F, new
CubeDeformation(0.0F)), PartPose.offset(-2.5F, 16.0F, 7.0F));

        PartDefinition leg1 = partdefinition.addOrReplaceChild("leg1",
CubeListBuilder.create().texOffs(0, 28).addBox(-1.0F, 0.0F, -1.0F, 2.0F, 8.0F, 2.0F, new
CubeDeformation(0.0F)), PartPose.offset(0.5F, 16.0F, 7.0F));
```

```
        PartDefinition leg2 = partdefinition.addOrReplaceChild("leg2",
CubeListBuilder.create().texOffs(0, 28).addBox(-1.0F, 0.0F, -1.0F, 2.0F, 8.0F, 2.0F, new
CubeDeformation(0.0F)), PartPose.offset(-2.5F, 16.0F, -4.0F));

        PartDefinition leg3 = partdefinition.addOrReplaceChild("leg3",
CubeListBuilder.create().texOffs(0, 28).addBox(-1.0F, 0.0F, -1.0F, 2.0F, 8.0F, 2.0F, new
CubeDeformation(0.0F)), PartPose.offset(0.5F, 16.0F, -4.0F));

        PartDefinition tail = partdefinition.addOrReplaceChild("tail",
CubeListBuilder.create().texOffs(8, 28).addBox(-1.0F, 0.0F, -1.0F, 2.0F, 8.0F, 2.0F, new
CubeDeformation(0.0F))
        .texOffs(16, 28).addBox(-1.0F, 8.0F, -1.0F, 2.0F, 3.0F, 2.0F, new
CubeDeformation(0.0F)), PartPose.offsetAndRotation(-1.0F, 12.0F, 8.0F, 0.9599F, 0.0F, 0.0F));

        return LayerDefinition.create(meshdefinition, 64, 64);
        }

        @Override
        public void setupAnim(Entity entity, float limbSwing, float limbSwingAmount, float
ageInTicks, float netHeadYaw, float headPitch) {

        }

        @Override
        public void renderToBuffer(PoseStack poseStack, VertexConsumer vertexConsumer, int
packedLight, int packedOverlay, float red, float green, float blue, float alpha) {
        head.render(poseStack, vertexConsumer, packedLight, packedOverlay, red, green, blue,
alpha);
        body.render(poseStack, vertexConsumer, packedLight, packedOverlay, red, green, blue,
alpha);
        upperBody.render(poseStack, vertexConsumer, packedLight, packedOverlay, red, green,
blue, alpha);
        leg0.render(poseStack, vertexConsumer, packedLight, packedOverlay, red, green, blue,
alpha);
        leg1.render(poseStack, vertexConsumer, packedLight, packedOverlay, red, green, blue,
alpha);
        leg2.render(poseStack, vertexConsumer, packedLight, packedOverlay, red, green, blue,
alpha);
        leg3.render(poseStack, vertexConsumer, packedLight, packedOverlay, red, green, blue,
alpha);
        tail.render(poseStack, vertexConsumer, packedLight, packedOverlay, red, green, blue,
alpha);
        }
}
```