CAMP MINECRAFT AI

– Doku 3 –

Kapitel 4 – TestMod und TestModClient

Damit wir unsere Entities, die wir gestern so schön erstellt haben, jetzt auch tatsächlich im Spiel sehen können, müssen wir diese noch im Spiel registrieren. Dafür gehen wir in die TestMod Klasse, in welcher wir bereits den Code für die Entities „porcupine“ und „cloudy“ finden:

```
//Test Custom Mobs und Animals
ModEntities.createCustomEntity( name: "porcupine", PorcupineEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("porcupine"), PorcupineEntity.createModEntityAttributes());
ModEntities.createCustomMob( name: "cloudy", CloudyEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("cloudy"), CloudyEntity.createMobAttributes());
```

Diesen Code können wir uns erneut zur Nutze machen, und nur unsere Namen bzw. Klassen anpassen, die referenziert werden. Dafür kopieren wir die zwei Zeilen vom „porcupine“, fügen diese in einer neuen Zeile unter „cloudy“ ein, und ersetzen folgende Aspekte:

```
ModEntities.createCustomEntity( name: "porcupine", PorcupineEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("porcupine"), PorcupineEntity.createModEntityAttributes());
```

Den Namen mit dem unserer Entität – in meinem Fall also: „superarmadillo“

```
ModEntities.createCustomEntity( name: "porcupine", PorcupineEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("porcupine"), PorcupineEntity.createModEntityAttributes());
```

Die referenzierte Entity-Klasse – in meinem Fall also: SuperarmadilloEntity::new

```
ModEntities.createCustomEntity( name: "porcupine", PorcupineEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("porcupine"), PorcupineEntity.createModEntityAttributes());
```

erneut den Namen der Entität, und zu guter Letzt:

```
ModEntities.createCustomEntity( name: "porcupine", PorcupineEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("porcupine"), PorcupineEntity.createModEntityAttributes());
```

erneut unsere referenzierte Entity-Klasse.

All diese Schritte sind beispielhaft für die Tiere gezeigt, solltet ihr ein Monster erschaffen haben, kopiert ihr einfach die Zeilen Code von „cloudy“!

Als nächstes schauen wir uns die TestModClient-Klasse an, in welcher wir erneut den Code für die beiden oben genannten Entitäten finden:

```
//Zeilen kopieren und Namen ändern vom .get Aufruf - .get("custom_entity_name"), und dem CustomModel::getTexturedModelData
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("porcupine_model"), PorcupineModel::getTexturedModelData);
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("cloudy_model"), CloudyModel::getTexturedModelData);
//Anpassen vom .get("custom_entity_name") und dem CustomEntityRenderer::new
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("porcupine")), PorcupineRenderer::new);
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("cloudy")), CloudyRenderer::new);
```

Hier müssen wir erneut den Code von entweder „porcupine“ für Tiere, oder „cloudy“ für Monster kopieren, im jeweiligen Abschnitt dafür in einer neuen Zeile neu einfügen, und anpassen.

```
//Zeilen kopieren und Namen ändern vom .get Aufruf - .get("custom_entity_name"), und dem CustomModel::getTexturedModelData
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("porcupine_model"), PorcupineModel::getTexturedModelData);
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("                    etTexturedModelData);
//Anpassen vom .get("custom_entity_name") und dem CustomEntityRenderer::new
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("porcupine")), PorcupineRenderer::new);
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("cloudy")), CloudyRenderer::new);
```

Extract Surround

so wird aus „porcupine_model“ → „superarmadillo_model“

```
//Zeilen kopieren und Namen ändern vom .get Aufruf - .get("custom_entity_name"), und dem CustomModel::getTexturedModelData
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("porcupine_model"), PorcupineModel::getTexturedModelData);
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("cloudy_model"), CloudyModel::getTexturedModelData);
//Anpassen vom .get("custom_entity_name") und dem CustomEntityRenderer::new
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("porcupine")), PorcupineRenderer::new);
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("cloudy")), CloudyRenderer::new);
```

die referenzierte Klasse wird bei mir zu: SuperarmadilloModel

```
//Zeilen kopieren und Namen ändern vom .get Aufruf - .get("custom_entity_name"), und dem CustomModel::getTexturedModelData
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("porcupine_model"), PorcupineModel::getTexturedModelData);
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("cloudy_model"), CloudyModel::getTexturedModelData);
//Anpassen vom .get("custom_entity_name") und dem CustomEntityRenderer::new
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("porcupine")), PorcupineRenderer::new);
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("cloudy")), CloudyRenderer::new);
```

der Name wird bei mir erneut zu: „superarmadillo“

```
//Zeilen kopieren und Namen ändern vom .get Aufruf - .get("custom_entity_name"), und dem CustomModel::getTexturedModelData
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("porcupine_model"), PorcupineModel::getTexturedModelData);
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("cloudy_model"), CloudyModel::getTexturedModelData);
//Anpassen vom .get("custom_entity_name") und dem CustomEntityRenderer::new
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("porcupine")), PorcupineRenderer::new);
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("cloudy")), CloudyRenderer::new);
```

und die referenzierte Klasse zu: SuperarmadilloRenderer

Wenn das alles getan ist, sehen die beiden Code Bereiche bei mir dann wie folgt aus:

```
//Test Custom Mobs und Animals
ModEntities.createCustomEntity( name: "porcupine", PorcupineEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("porcupine"), PorcupineEntity.createModEntityAttributes());
ModEntities.createCustomMob( name: "cloudy", CloudyEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("cloudy"), CloudyEntity.createMobAttributes());
ModEntities.createCustomEntity( name: "superarmadillo", SuperarmadilloEntity::new);
FabricDefaultAttributeRegistry.register(ModEntities.ModEntitiesMap.get("superarmadillo"), SuperarmadilloEntity.createModEntityAttributes());
```

```
//Zeilen kopieren und Namen ändern vom .get Aufruf - .get("custom_entity_name"), und dem CustomModel::getTexturedModelData
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("porcupine_model"), PorcupineModel::getTexturedModelData);
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("superarmadillo_model"), SuperarmadilloModel::getTexturedModelData);
EntityModelLayerRegistry.registerModelLayer(EntityModels.get("cloudy_model"), CloudyModel::getTexturedModelData);
//Anpassen vom .get("custom_entity_name") und dem CustomEntityRenderer::new
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("porcupine")), PorcupineRenderer::new);
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("superarmadillo")), SuperarmadilloRenderer::new);
EntityRendererRegistry.register(((FabricEntityType) ModEntities.ModEntitiesMap.get("cloudy")), CloudyRenderer::new);
```

Als nächstes schauen wir uns gemeinsam dann für den restlichen Tag noch die Animationen an, bevor ihr euch um die Verfeinerung der KI, dem Aussehen und den Animationen kümmern könnt.

Kapitel 4.1 – Animationen

Für die Animationen springen wir erneut zurück in Blockbench. Dafür schauen wir uns oben rechts die Funktion „Animate“ an.

Schritt 1: Schlüsselbilder setzen

Schlüsselbilder sind die grundlegenden Elemente einer Animation. Sie definieren die Position, Rotation und Skalierung eines Teils zu einem bestimmten Zeitpunkt. Um ein Schlüsselbild zu setzen, wählt ihr das Teil des Modells aus, das ihr animieren möchtet, und bewegt es in die gewünschte Position. Dann klickt ihr auf das kleine Diamantsymbol in der Zeitleiste, um ein Schlüsselbild zu erstellen.

Schritt 2: Animation erstellen

Um die Animation zu erstellen, setzt ihr mehrere Schlüsselbilder an verschiedenen Stellen in der Zeitleiste. Wenn ihr beispielsweise einen Arm des Modells bewegen möchtet, setzen wir ein Schlüsselbild bei 0 Sekunden, bewegen den Arm ein wenig und setzen ein weiteres Schlüsselbild bei 1 Sekunde. Das wiederholen wir dann, bis die gewünschte Bewegung vollständig ist.

Schritt 3: Interpolation einstellen

Die Interpolation bestimmt, wie die Bewegung zwischen den Schlüsselbildern abläuft. In Blockbench können wir zwischen verschiedenen Interpolationsarten wählen, wie linear, glatt oder sprunghaft. Um die Interpolation zu ändern, klicken wir mit der rechten Maustaste auf die Zeitleiste zwischen zwei Schlüsselbildern und wählen die gewünschte Interpolationsmethode aus.

Nachdem wir die Schlüsselbilder und Interpolationen gesetzt haben, sollten wir die Animation testen, um sicherzustellen, dass sie wie gewünscht aussieht. Dafür klicken wir auf die Wiedergabeschaltfläche in der Zeitleiste, um die Animation abzuspielen. Beobachten wir das Modell und achten darauf, dass alle Bewegungen flüssig und realistisch sind.

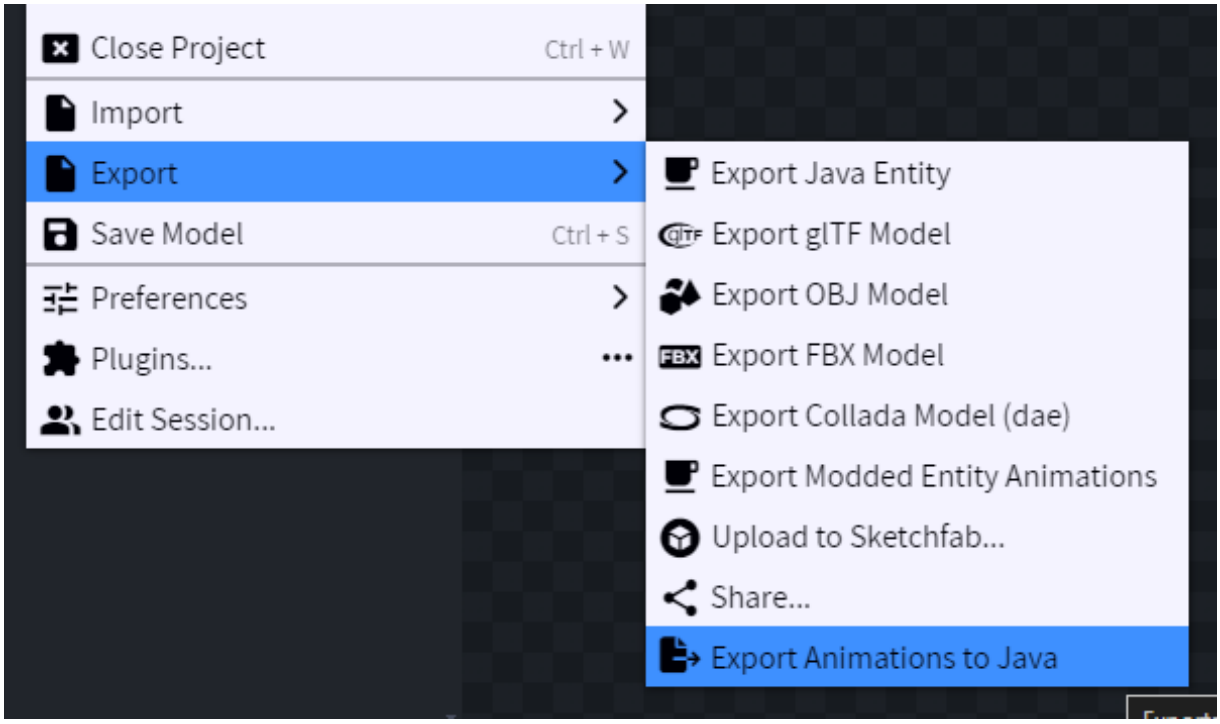
Schritt 4: Animation verfeinern

Falls nötig, können wir die Animation verfeinern, indem wir zusätzliche Schlüsselbilder hinzufügen oder bestehende ändern. Experimentiert mit verschiedenen Bewegungen und Interpolationen, bis ihr mit dem Ergebnis zufrieden sind.

Am besten, ihr erstellt 3 Animationen, eine sogenannte „Idle“-Animation, also eine Animation, die abgespielt wird, wenn die Entität nichts macht, eine Laufanimation, und eine Angriffsanimation.

Kapitel 4.2 – eine fertige Entität

Um die Animationen nun auch in IntelliJ und damit in Minecraft zu kriegen, exportieren wir die Animationen einmal über die Schlatfläche „Datei“ oder „File“, „Export“ → „Export Animations to Java“:



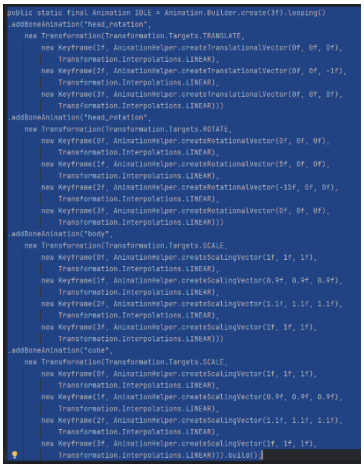
Sollte die Möglichkeit nicht angezeigt werden, müsst ihr einmal unter „Datei“ oder „File“, „Plugins…“ gehen, und dort einmal nach dem Plugin „Animation to Java Converter“ suchen, und dieses Installieren.

Dann wählt ihr bei dem Export – GANZ WICHTIG!! – die „yarn“ Version aus, und exportiert die Datei in den resources Ordner mit dem Namen eurer Entität + „Anim“ dahinter. In meinem Fall also:

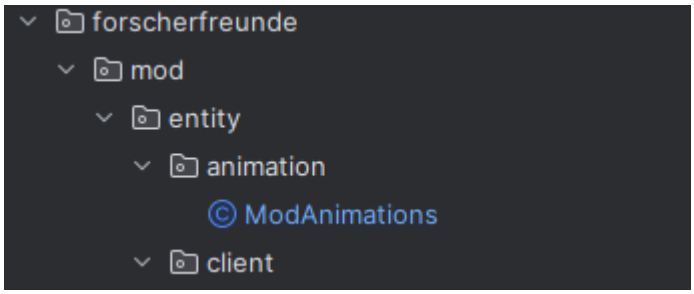
„SuperarmadilloAnim“

Am besten exportiert ihr die Datei erst, wenn ihr alle Animationen fertig habt, damit ihr euch Arbeit spart.

Wenn wir die Datei dann fertig exportiert haben, sollte sie in IntelliJ wie folgt aussehen – nur bei euch dann mit mehr Code und mehr Animationen:



Diesen ganzen Code kopieren wir einfach, und fügen diesen dann in der Klasse „ModAnimations“ unter „entity“ → „animation“, unter der letzten Zeile Code – ABER WICHTIG, VOR DER KLAMMER ZU – ein:



```
463
464     public static final Animation IDLE = Animation.Builder.create(3f).looping()  no usages
465             .addBoneAnimation( name: "head_rotation",
466                     new Transformation(Transformation.Targets.TRANSLATE,
467                             new Keyframe( f: 1f, AnimationHelper.createTranslationalVector( x: 0f,  y: 0f,  z: 0f),
468                                     Transformation.Interpolations.LINEAR),
469                             new Keyframe( f: 2f, AnimationHelper.createTranslationalVector( x: 0f,  y: 0f,  z: -1f),
470                                     Transformation.Interpolations.LINEAR),
471                             new Keyframe( f: 3f, AnimationHelper.createTranslationalVector( x: 0f,  y: 0f,  z: 0f),
472                                     Transformation.Interpolations.LINEAR)))
473             .addBoneAnimation( name: "head_rotation",
474                     new Transformation(Transformation.Targets.ROTATE,
475                             new Keyframe( f: 0f, AnimationHelper.createRotationalVector( x: 0f,  y: 0f,  z: 0f),
476                                     Transformation.Interpolations.LINEAR),
477                             new Keyframe( f: 1f, AnimationHelper.createRotationalVector( x: 5f,  y: 0f,  z: 0f),
478                                     Transformation.Interpolations.LINEAR),
479                             new Keyframe( f: 2f, AnimationHelper.createRotationalVector( x: -15f,  y: 0f,  z: 0f),
480                                     Transformation.Interpolations.LINEAR),
481                             new Keyframe( f: 3f, AnimationHelper.createRotationalVector( x: 0f,  y: 0f,  z: 0f),
482                                     Transformation.Interpolations.LINEAR)))
483             .addBoneAnimation( name: "body",
484                     new Transformation(Transformation.Targets.SCALE,
485                             new Keyframe( f: 0f, AnimationHelper.createScalingVector( x: 1f,  y: 1f,  z: 1f),
486                                     Transformation.Interpolations.LINEAR),
487                             new Keyframe( f: 1f, AnimationHelper.createScalingVector( x: 0.9f,  y: 0.9f,  z: 0.9f),
488                                     Transformation.Interpolations.LINEAR),
489                             new Keyframe( f: 2f, AnimationHelper.createScalingVector( x: 1.1f,  y: 1.1f,  z: 1.1f),
490                                     Transformation.Interpolations.LINEAR),
491                             new Keyframe( f: 3f, AnimationHelper.createScalingVector( x: 1f,  y: 1f,  z: 1f),
492                                     Transformation.Interpolations.LINEAR)))
493             .addBoneAnimation( name: "cube",
494                     new Transformation(Transformation.Targets.SCALE,
495                             new Keyframe( f: 0f, AnimationHelper.createScalingVector( x: 1f,  y: 1f,  z: 1f),
496                                     Transformation.Interpolations.LINEAR),
497                             new Keyframe( f: 1f, AnimationHelper.createScalingVector( x: 0.9f,  y: 0.9f,  z: 0.9f),
498                                     Transformation.Interpolations.LINEAR),
499                             new Keyframe( f: 2f, AnimationHelper.createScalingVector( x: 1.1f,  y: 1.1f,  z: 1.1f),
500                                     Transformation.Interpolations.LINEAR),
501                             new Keyframe( f: 3f, AnimationHelper.createScalingVector( x: 1f,  y: 1f,  z: 1f),
502                                     Transformation.Interpolations.LINEAR))).build();
503
504 }
```

Dann müssen wir die Animationen noch umbenennen, mit dem Namen unserer Entität in Großbuchstaben davor, so dass bei mir dann:

SUPERARMADILLO_IDLE

da steht:

```
Rename usages
public static final Animation SUPERARMADILLO_IDLE = Animation.Builder.create(3f).looping()
        .addBoneAnimation( name: "head_rotation",
                new Transformation(Transformation.Targets.TRANSLATE,
                        new Keyframe( f: 1f, AnimationHelper.createTranslationalVector( x: 0f,  y: 0f,  z: 0f),
                                Transformation.Interpolations.LINEAR),
```

Nun müssen wir die neuen Animationen nur noch in der Model-Klasse einbauen. Dafür gehen wir zunächst in die PorcupineModel-Klasse, und kopieren uns folgenden Code:

```
@Override  no usages  ForscherViking
public void setAngles(PorcupineEntity entity, float limbSwing, float limbSwingAmount, float ageInTicks, float netHeadYaw, float headPitch) {
    this.getPart().traverse().forEach(ModelPart::resetTransform);
    this.setHeadAngles(netHeadYaw, headPitch);

    this.animateMovement(ModAnimations.PORCUPINE_WALK, limbSwing, limbSwingAmount,  limbAngleScale: 2f,  limbDistanceScale: 2.5f);
    this.updateAnimation(entity.idleAnimationState, ModAnimations.PORCUPINE_IDLE, ageInTicks,  speedMultiplier: 1f);
}
```

Diesen Code fügen wir in unserer Model-Klasse dann in der Methode „setAngles()“ ein, und müssen dann ein paar Stellen anpassen, damit wir die Fehler loswerden:

```
@Override no usages
public void setAngles(T entity, float limbAngle, float limbDistance, float animationProgress, float headYaw, float headPitch) {

    this.animateMovement(ModAnimations.PORCUPINE_WALK, limbSwing, limbSwingAmount, limbAngleScale: 2f, limbDistanceScale: 2.5f);
    this.updateAnimation(entity.idleAnimationState, ModAnimations.PORCUPINE_IDLE, ageInTicks, speedMultiplier: 1f);
}
```

Zunächst ändern wir das „T“ zu unserer Entity Klasse ab, bei mir also: SuperarmadilloEntity

```
@Override no usages
public void setAngles(SuperarmadilloEntity entity, float limbAngle, float limbDistance, float animationProgress, float headYaw, float headPitch) {

    this.animateMovement(ModAnimations.PORCUPINE_WALK, limbSwing, limbSwingAmount, limbAngleScale: 2f, limbDistanceScale: 2.5f);
    this.updateAnimation(entity.idleAnimationState, ModAnimations.PORCUPINE_IDLE, ageInTicks, speedMultiplier: 1f);
}
```

Als nächstes ändern wir „limbAngle“ → „limbSwing“

```
@Override no usages
public void setAngles(SuperarmadilloEntity entity, float limbSwing, float limbDistance, float animationProgress, float headYaw, float headPitch) {

    this.animateMovement(ModAnimations.PORCUPINE_WALK, limbSwing, limbSwingAmount, limbAngleScale: 2f, limbDistanceScale: 2.5f);
    this.updateAnimation(entity.idleAnimationState, ModAnimations.PORCUPINE_IDLE, ageInTicks, speedMultiplier: 1f);
}
```

Dann „limbDistance“ → „limbSwingAmount“

```
@Override no usages
public void setAngles(SuperarmadilloEntity entity, float limbSwing, float limbSwingAmount, float animationProgress, float headYaw, float headPitch) {

    this.animateMovement(ModAnimations.PORCUPINE_WALK, limbSwing, limbSwingAmount, limbAngleScale: 2f, limbDistanceScale: 2.5f);
    this.updateAnimation(entity.idleAnimationState, ModAnimations.PORCUPINE_IDLE, ageInTicks, speedMultiplier: 1f);
}
```

Dann machen wir aus „animationProgress“ → „ageInTicks“

```
@Override no usages
public void setAngles(SuperarmadilloEntity entity, float limbSwing, float limbSwingAmount, float ageInTicks, float headYaw, float headPitch) {

    this.animateMovement(ModAnimations.PORCUPINE_WALK, limbSwing, limbSwingAmount, limbAngleScale: 2f, limbDistanceScale: 2.5f);
    this.updateAnimation(entity.idleAnimationState, ModAnimations.PORCUPINE_IDLE, ageInTicks, speedMultiplier: 1f);
}
```

Und zu guter Letzt machen wir noch aus PORCUPINE_WALK → SUPERARMADILLO_WALK

```
@Override no usages
public void setAngles(SuperarmadilloEntity entity, float limbSwing, float limbSwingAmount, float ageInTicks, float headYaw, float headPitch) {

    this.animateMovement(ModAnimations.SUPERARMADILLO_WALK, limbSwing, limbSwingAmount, limbAngleScale: 2f, limbDistanceScale: 2.5f);
    this.updateAnimation(entity.idleAnimationState, ModAnimations.PORCUPINE_IDLE, ageInTicks, speedMultiplier: 1f);
}
```

Und aus PORCUPINE_IDLE → SUPERARMADILLO_IDLE.

Mit den Animationen fertig, gehts dann zurück in Minecraft, um zu schauen, ob wir alles richtig gemacht haben.

Solltet ihr an diesem Punkt unzufrieden mit den Animationen sein, dann müsst ihr nur, nachdem ihr die Animationen angepasst habt, die Dateien neu exportieren, und den Code mit dem neuen Code ersetzen in der „ModAnimations“-Klasse.

Ab jetzt habt ihr alles Wissen und alle Werkzeuge, um einfache Entitäten in Minecraft über unsere Schnittstelle hinzuzufügen. Bedeutet: überarbeitet gerne erst mal eure bisherige Entität, passt nochmal die Textur an, falls ihr damit noch nicht zufrieden seid, passt das Modell final an, und arbeitet die Animationen nochmal durch, bis ihr mit allem davon fertig und zufrieden seid. Falls ihr dann noch genug Zeit habt, könnt ihr euch an eine neue, zweite oder dritte Entität begeben.

Behaltet nur im Kopf – am Ende nehmt ihr nur mit, was ihr auch fertigbekommen habt. Ihr habt dann keine Möglichkeit mehr, im Nachhinein eure Entität anzupassen, wenn ihr zuhause seid.