CAMP MINECRAFT AI

– Doku 4 –

Kapitel 5 – komplexe KI

Um euch einen kleinen Einblick in komplexere KI und deren Implementierung zu geben, schauen wir uns ganz schnell einmal den Code von einem Schwein in Minecraft an – zumindest die Bereiche di uns interessieren.

```
protected void initGoals() {
    this.goalSelector.add( priority: 0, new SwimGoal( mob: this));
    this.goalSelector.add( priority: 1, new EscapeDangerGoal( mob: this, (double) 1.25F));
    this.goalSelector.add( priority: 3, new AnimalMateGoal( animal: this, (double) 1.0F));
    this.goalSelector.add( priority: 4, new TemptGoal( entity: this,  speed: 1.2, Ingredient.ofItems(new ItemConvertible[]{Items.CARROT_ON_A_STICK}),  canBeScared: false));
    this.goalSelector.add( priority: 4, new TemptGoal( entity: this,  speed: 1.2, BREEDING_INGREDIENT,  canBeScared: false));
    this.goalSelector.add( priority: 5, new FollowParentGoal( animal: this,  speed: 1.1));
    this.goalSelector.add( priority: 6, new WanderAroundFarGoal( pathAwareEntity: this, (double) 1.0F));
    this.goalSelector.add( priority: 7, new LookAtEntityGoal( mob: this, PlayerEntity.class,  range: 6.0F));
    this.goalSelector.add( priority: 8, new LookAroundGoal( mob: this));
}
```

Diese Zeilen Code sollten euch recht bekannt vorkommen – es handelt sich um die KI bzw. das Verhalten von den Schweinen in Minecraft. Die Ziele sind hier so geschrieben, wie sie im Code eigentlich zu finden sind – anders als bei uns, denn ich habe die wichtigsten der 198 Ziele für uns auf Deutsch – und deutlich vereinfacht für euch bereitgestellt. Wenn ihr die Ziele in dieser Form implementieren wolltet – was möglich ist – müsstet ihr deutlich mehr Code schreiben, und vor allem verstehen was ihr als Parameter bei den Zielen angeben müsstet, damit sie auch so funktionieren wie ihr euch das vorstellt.

Die folgenden Ausschnitte werden alle benötigt, um dem Schwein überhaupt die Möglichkeit zu geben, von einem Spieler geritten zu werden. Anhand dieses Beispiels wird euch eventuell dann auch bewusst, warum wir diese Möglichkeiten nicht in diesem Camp zur Verfügung stellen und alles einzeln erklären können, da es sich um hunderte Zeilen Code handelt, die es zu verstehen und dann auch noch richtig zu übernehmen gilt.

Diese Zeilen sind sogar nicht mal direkt dafür da, dass der Spieler das Schwein reiten kann, sondern sind nur sogenannte Hilfsmethoden, um all die Daten zu verwalten, die benötigt werden, damit der Spieler das Schwein reiten könnte.

```java
@Nullable
public LivingEntity getControllingPassenger() {
    if (this.isSaddled()) {
        Entity var2 = this.getFirstPassenger();
        if (var2 instanceof PlayerEntity) {
            PlayerEntity playerEntity = (PlayerEntity) var2;
            if (playerEntity.isHolding(Items.CARROT_ON_A_STICK)) {
                return playerEntity;
            }
        }
    }

    return super.getControllingPassenger();
}

public void onTrackedDataSet(TrackedData<?> data) {
    if (BOOST_TIME.equals(data) && this.getWorld().isClient) {
        this.saddledComponent.boost();
    }

    super.onTrackedDataSet(data);
}

protected void initDataTracker() {
    super.initDataTracker();
    this.dataTracker.startTracking(SADDLED, initialValue: false);
    this.dataTracker.startTracking(BOOST_TIME, initialValue: 0);
}

public void writeCustomDataToNbt(NbtCompound nbt) {
    super.writeCustomDataToNbt(nbt);
    this.saddledComponent.writeNbt(nbt);
}

public void readCustomDataFromNbt(NbtCompound nbt) {
    super.readCustomDataFromNbt(nbt);
    this.saddledComponent.readNbt(nbt);
}
```

```java
public ActionResult interactMob(PlayerEntity player, Hand hand) {
    boolean bl = this.isBreedingItem(player.getStackInHand(hand));
    if (!bl && this.isSaddled() && !this.hasPassengers() && !player.shouldCancelInteraction()) {
        if (!this.getWorld().isClient) {
            player.startRiding( entity: this);
        }

        return ActionResult.success(this.getWorld().isClient);
    } else {
        ActionResult actionResult = super.interactMob(player, hand);
        if (!actionResult.isAccepted()) {
            ItemStack itemStack = player.getStackInHand(hand);
            return itemStack.isOf(Items.SADDLE) ? itemStack.useOnEntity(player,  entity: this, hand) : ActionResult.PASS;
        } else {
            return actionResult;
        }
    }
}

public boolean canBeSaddled() { return this.isAlive() && !this.isBaby(); }

protected void dropInventory() {
    super.dropInventory();
    if (this.isSaddled()) {
        this.dropItem(Items.SADDLE);
    }

}

public boolean isSaddled() { return this.saddledComponent.isSaddled(); }

public void saddle(@Nullable SoundCategory sound) {
    this.saddledComponent.setSaddled(true);
    if (sound != null) {
        this.getWorld().playSoundFromEntity((PlayerEntity) null,  entity: this, SoundEvents.ENTITY_PIG_SADDLE, sound,  volume: 0.5F,  pitch: 1.0F);
    }

}
```

Dieser Code sorgt jetzt tatsächlich dafür, dass ein Spieler das Schwein reiten kann…

```java
public Vec3d updatePassengerForDismount(LivingEntity passenger) {
    Direction direction = this.getMovementDirection();
    if (direction.getAxis() == Axis.Y) {
        return super.updatePassengerForDismount(passenger);
    } else {
        int[][] is = Dismounting.getDismountOffsets(direction);
        BlockPos blockPos = this.getBlockPos();
        BlockPos.Mutable mutable = new BlockPos.Mutable();
        UnmodifiableIterator var6 = passenger.getPoses().iterator();

        while (var6.hasNext()) {
            EntityPose entityPose = (EntityPose) var6.next();
            Box box = passenger.getBoundingBox(entityPose);

            for (int[] js : is) {
                mutable.set( x: blockPos.getX() + js[0], blockPos.getY(),  z: blockPos.getZ() + js[1]);
                double d = this.getWorld().getDismountHeight(mutable);
                if (Dismounting.canDismountInBlock(d)) {
                    Vec3d vec3d = Vec3d.ofCenter(mutable, d);
                    if (Dismounting.canPlaceEntityAt(this.getWorld(), passenger, box.offset(vec3d))) {
                        passenger.setPose(entityPose);
                        return vec3d;
                    }
                }
            }
        }

        return super.updatePassengerForDismount(passenger);
    }
}
```

Dieser Code ist alleine notwendig dafür, dass der Spieler das Schwein auch wieder ordentlich verlassen kann…

```java
protected void tickControlled(PlayerEntity controllingPlayer, Vec3d movementInput) {
    super.tickControlled(controllingPlayer, movementInput);
    this.setRotation(controllingPlayer.getYaw(),  pitch: controllingPlayer.getPitch() * 0.5F);
    this.prevYaw = this.bodyYaw = this.headYaw = this.getYaw();
    this.saddledComponent.tickBoost();
}

protected Vec3d getControlledMovementInput(PlayerEntity controllingPlayer, Vec3d movementInput) {
    return new Vec3d((double) 0.0F, (double) 0.0F, (double) 1.0F);
}

protected float getSaddledSpeed(PlayerEntity controllingPlayer) {
    return (float) (this.getAttributeValue(EntityAttributes.GENERIC_MOVEMENT_SPEED) * 0.225 * (double) this.saddledComponent.getMovementSpeedMultiplier());
}

public boolean consumeOnAStickItem() { return this.saddledComponent.boost(this.getRandom()); }

@Nullable
public PigEntity createChild(ServerWorld serverWorld, PassiveEntity passiveEntity) {
    return (PigEntity) EntityType.PIG.create(serverWorld);
}

public boolean isBreedingItem(ItemStack stack) { return BREEDING_INGREDIENT.test(stack); }

public Vec3d getLeashOffset() {
    return new Vec3d((double) 0.0F, (double) (0.6F * this.getStandingEyeHeight()), (double) (this.getWidth() * 0.4F));
}

protected Vector3f getPassengerAttachmentPos(Entity passenger, EntityDimensions dimensions, float scaleFactor) {
    return new Vector3f( x: 0.0F,  y: dimensions.height - 0.03125F * scaleFactor,  z: 0.0F);
}

static {
    SADDLED = DataTracker.registerData(PigEntity.class, TrackedDataHandlerRegistry.BOOLEAN);
    BOOST_TIME = DataTracker.registerData(PigEntity.class, TrackedDataHandlerRegistry.INTEGER);
    BREEDING_INGREDIENT = Ingredient.ofItems(new ItemConvertible[]{Items.CARROT, Items.POTATO, Items.BEETROOT});
}
```

und zu guter Letzt sind diese Zeilen für die Bewegung des Schweins – durch die Inputs des Spielers gesteuert – verantwortlich, sowie einzelne weitere Hilfsmethoden.

Natürlich dürft ihr euch auch an solchen Code heranwagen, wenn ihr das wollt, ihr könnt eure Entitäten auch Sattelbar machen, zu schwimmenden oder fliegenden Entitäten machen, oder gar zu einer Kreatur wie dem Enderdrachen. Allerdings ist der Code dafür, damit das alles ordentlich funktioniert, nicht einfach zu verstehen, und teilweise auch gar nicht so einfach einzubinden in eine neue Kreatur. So könnt ihr zwar eine Kreatur erschaffen, welche grundlegend ein ähnliches Verhalten und KI wie der Enderdrache aufweist, jedoch werdet ihr sie nicht im Ende spawnen können, bzw. sie wird nicht das Spiel beenden, wenn ihr sie besiegt.

Im Folgenden gebe ich euch die Möglichkeit an die Hand, etwas durch die bestehenden Entity-Klassen durchzuschauen und etwas herumzuprobieren, falls ihr das wollt.

Kapitel 5.1 – Hierarchie und Minecraft-Entity

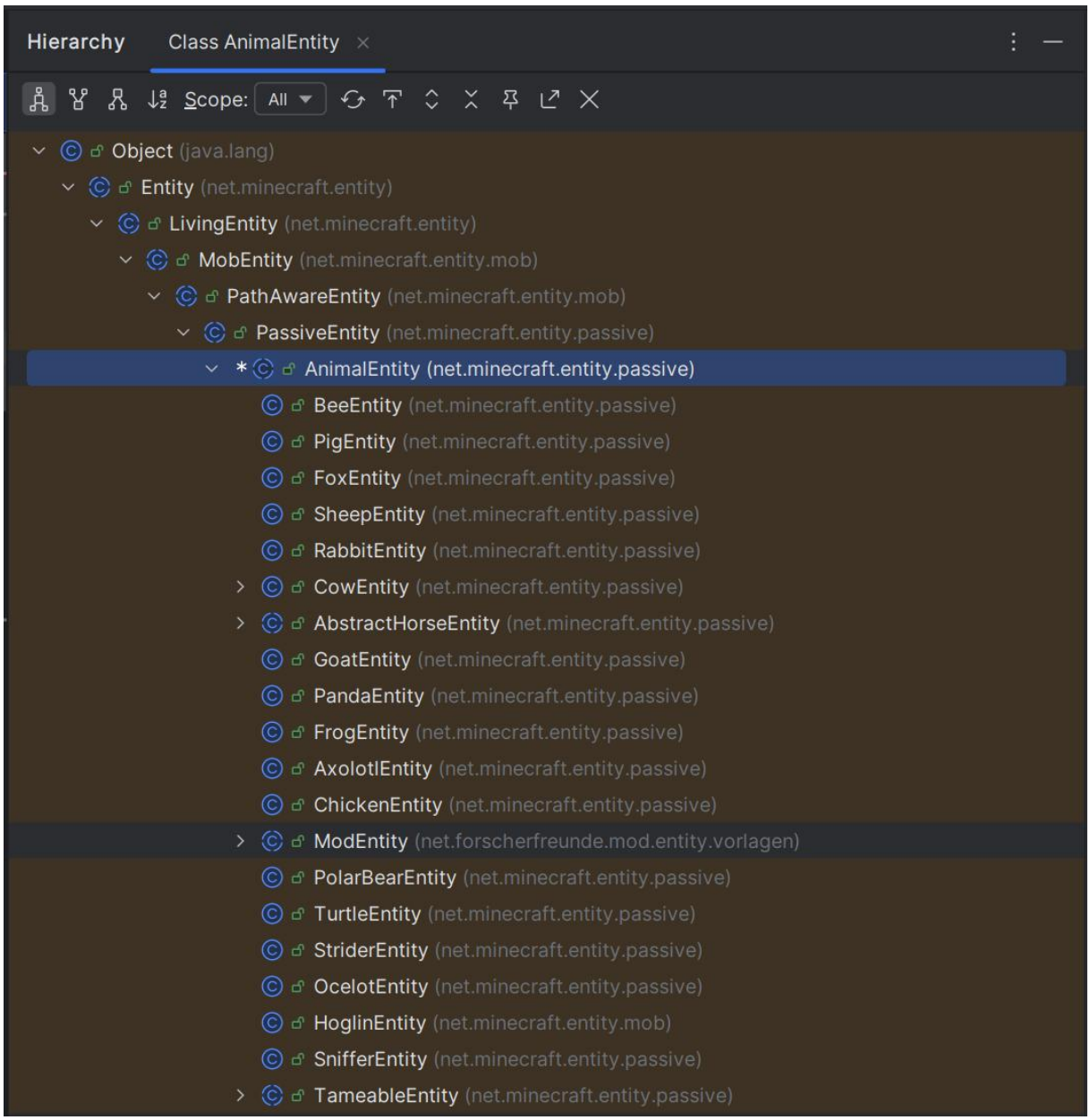
Wie bereits erwähnt, dürft ihr euch gerne von den bereits vorhandenen Klassen inspirieren lassen und diese als Vorlagen für eure eigene Entitäten verwenden. Dafür könnt ihr einmal mit der Maus einen Rechtsklick in eurer Entitäten-Klasse auf „ModEntity“ machen:

```java
public class SuperarmadilloEntity extends ModEntity {  11 usages  ForscherViking

    public SuperarmadilloEntity(EntityType<? extends AnimalEntity> entityType, World world) {  1 usage  ForscherViking
        super(entityType, world);
    }
```

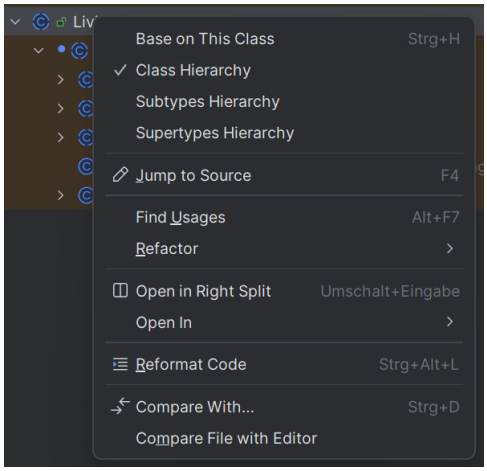
Dann sollte sich die Klasse „ModEntity öffnen. In dieser macht ihr einen Rechtsklick auf „Animal Entity“,

und in dieser Klasse angekommen drückt ihr auf der Tastatur strg + H (bzw. ctrl + H), um die Hierarchie zu öffnen:



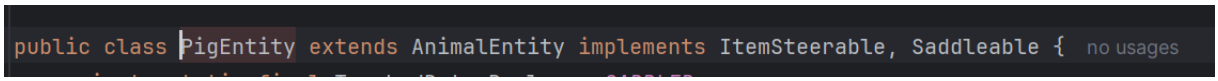
In dieser angekommen, könnt ihr euch umschauen, und findet dort alle Tier-Entitäten in Minecraft.

Natürlich gibt es auch noch andere Entitäten, die ihr euch anschauen könnt – vor allem die Monster. Dafür könnt ihr in der Hierarchie einfach einen Rechtsklick auf „MobEntity“, und wählt dort „base on this class“ aus:



Nun könnt ihr die Klassen anschauen, indem ihr die einzelnen Unterklassen aufklappt in der Hierarchie, und diese per Doppelklick öffnen und euch den Code davon anschauen.

Falls ihr eure eigene Entität auf einer der bestehenden aufbauen wollt, dann könnt ihr einfach schauen, welche Funktionen die Entität verwendet – Beispiel von dem Schwein: verwendet die Funktionen „ItemSteerable“ und „Saddleable“:



Dann könnt ihr eure Entität mit denselben Funktionen versehen – indem ihr

implements

gefolgt von den Funktionsinterfaces dahinter schreibt, und dann im Anschluss schaut, welche Methoden ihr aus der jeweiligen Klasse wohl benötigen werdet, um die Funktionen auch ordentlich einzubauen. Dann kopiert ihr euch die Zeilen Code, und versucht etwas herum, passt den Code entsprechend an, und schaut, ob ihr es schafft die Funktionalitäten einzubauen.