09 Úprava chování hůlky

Aktuálně nám hůlka sice vytváří blesky, ale nechová se úplně tak, jak bychom si představovali. Blesk se sice vytvoří, ale vždy jen ve směru nějaké světové strany (sever, jih,...) a blesk se nevytvoří na povrchu, ale může se vytvořit i ve vzduchu. Toto chování si nyní upravíme.

Toto řešení bude o něco složitější, než naše aktuální řešení. Bude zahrnovat více matematiky, kterou nebudu vysvětlovat do detailu.

Budeme upravovat kód z minulé hodiny a vždy budou zvýrazněné změněné/přidané řádky.

Získání směru

Nejprve si opravíme získání směru, kam se hráč dívá. Bohužel není jednoduchá metoda, kterou můžeme zavolat a vrátí nám směr, kterým se hráč dívá. Můžeme to ale zjistit z jiných dostupných údajů.

V Minecraftu se otočení hlavy hráče udává pomocí dvou čísel `pitch` (směr nahoru a dolů) a `yaw` (směr doleva a doprava). Tyto údaje získáme ze třídy `PlayerEntity` pomocí metod `getPitch` a `getYaw`. Z těchto dvou údajů si poté pomocí metody `fromPolar` ze třídy `Vec3d` můžeme získat směr, kterým se hráč dívá.

```
1   Vec3d.fromPolar(user.getPitch(), user.getYaw());
```

Rovnou si můžeme získat hodnotu, o kterou později posuneme pozici hráče a tím zjistíme, kam máme umístit blesk. Tuto hodnotu si můžete představit jako šipku, která jde od hráče směrem, kterým se dívá a konec šipky určuje, kam se umístí blesk. Této šipce se říká *vektor*.

Nejprve musíme šipku (vektor) upravit tak, aby měla délku 1, abychom tuto vzdálenost mohli jednoduše vynásobit nějakým číslem a do této vzdálenosti se blesk položí. To uděláme pomocí metody `normalize`. Ta upraví vektor (šipku) tak, aby měl délku 1. Poté jej jen vynásobíme pomocí metody `multiply`, kde jako parametr uvedeme vzdálenost, do které chceme blesk umístit.

> **Typ proměnné** `var`
>
> V předchozích lekcích jsme si říkali, že vždy musíme uvádět typ hodnoty proměnné. To si ale můžeme zjednodušit a místo typu uvádět slovo `var`. Typ proměnné se pak nastaví podle první hodnoty, kterou do ní uložíme.

```
public TypedActionResult<ItemStack> use(World world, PlayerEntity user, Hand hand) {
    if (world.isClient) {
        return TypedActionResult.pass(user.getStackInHand(hand));
    }

    var offset = Vec3d.fromPolar(user.getPitch(), user.getYaw())
            .normalize().multiply(10);

    BlockPos blockPos = user.getBlockPos().offset(user.getHorizontalFacing(), 10);

    LightningEntity lightning = new LightningEntity(EntityType.LIGHTNING_BOLT, world);

    lightning.setPosition(blockPos.toCenterPos());
    world.spawnEntity(lightning);

    return TypedActionResult.success(user.getStackInHand(hand));
}
```

Získání pozice bloku

Dále si potřebujeme získat blok, který je v námi určené vzdálenosti od hráče a ideálně bychom chtěl, aby blesk udeřil do země a ne do vzduchu.

Pozici, kam má udeřit blesk získáme tak, že nejprve si zjistíme pozici očí hráče (metoda `getEyePos`) a posuneme ji o danou vzdálenost z předchozího kroku (metoda `add`).

```
var pos = user.getEyePos().add(offset);
```

Abychom získali pozici na povrchu, tak nejprve musíme získanou pozici převést na pozici bloku ve světě. Na to můžeme použít metodu `ofFloored` ze třídy `BlockPos`. Tato metoda zaokrouhlí souřadnice, které jí předáme a vrátí nám pozici bloku, ve kterém se tyto souřadnice nachází.

```
BlockPos blockPos = BlockPos.ofFloored(pos);
```

Zjištění nejvyššího bloku

Jako poslední krok musíme zjistit, jaký nejvyšší blok se na těchto souřadnicích nachází. Na to máme ve třídě `World` metodu `getTopPosition`.

V Minecraftu je pro celý svět vytvořená mapa výšky (*heightmap*), kde jsou uložené informace o tom, jaký nejvyšší blok se na každém místě nachází. Existuje více druhů těchto map. Pro nás jsou důležité následující možnosti:

- `MOTION_BLOCKING` - jakýkoliv pevný blok, přes který hráč nemůže projít nebo tekutina (voda, láva)
- `MOTION_BLOCKING_NO_LEAVES` - stejné, jako předchozí jen nezahrnuje listy

```
blockPos = world.getTopPosition(Heightmap.Type.MOTION_BLOCKING, blockPos);
```

```
public TypedActionResult<ItemStack> use(World world, PlayerEntity user, Hand hand) {
    if (world.isClient) {
        return TypedActionResult.pass(user.getStackInHand(hand));
    }

    var offset = Vec3d.fromPolar(user.getPitch(), user.getYaw())
            .normalize().multiply(10);

    var pos = user.getEyePos().add(offset);

    BlockPos blockPos = BlockPos.ofFloored(pos);
    blockPos = world.getTopPosition(Heightmap.Type.MOTION_BLOCKING, blockPos);

    LightningEntity lightning = new LightningEntity(EntityType.LIGHTNING_BOLT, world);

    lightning.setPosition(blockPos.toCenterPos());
    world.spawnEntity(lightning);

    return TypedActionResult.success(user.getStackInHand(hand));
}
```

Poznámka

Původní řádek

```
BlockPos blockPos = user.getBlockPos().offset(user.getHorizontalFacing(), 10);
```

jsme změnili na

```
BlockPos blockPos = BlockPos.ofFloored(pos);
```

Nyní by nám kód měl fungovat a blesk by se měl umístit 10 bloků před hráče na pevný blok na povrchu.

Vytvoření exploze

Jako poslední věc si k blesku přidáme ještě explozi. To není nijak složité a ve třídě `World` na to existuje metoda `createExplosion`. Tato metoda má spoustu parametrů, ale zjednodušuje nám tvorbu exploze.

> **Umístění kódu**
>
> Kód umístíme na konec pod řádek, kde umisťujeme blesk: `world.spawnEntity(lightning);`

Jako první parametr uvádíme entitu, která je původem exploze. V našem případě to může být blesk - proměnná `lightning`.

Dále uvádíme pozice, kde se má exploze vytvořit. Ty můžeme vzít z proměnné `blockPos`, protože tyto souřadnice používáme pro umístění blesku. Souřadnice musíme uvést jednotlivě jako pozice `x`, `y` a `z` (v tomto pořadí). Ty získáme pomocí metod `getX`, `getY` a `getZ`.

Další parametr je už zajímavější a to je síla exploze. To se uvádí jako desetinné číslo proto za číslo napíšeme písmeno `f`. Pro porovnání - TNT má sílu exploze `4f` a charged creeper `6f`.

> **Pozor**
>
> Pokud zadáte moc vysoké číslo, tak se může stát, že vám hra spadne a daný svět už nepůjde načíst. Tato metoda není dělaná na vysoké hodnoty a proto exploze s vyšší sílou mohou ničit bloky v nepředvídatelných tvarech (díra po explozi nebude vypadat hezky). **Zkoušejte proto na vlasní riziko.**

Předposlední parametr určuje, zda exploze vytvoří i oheň. `true` - ano, `false` - ne.

A poslední parametr je typ exploze. Tím se upravuje, jestli se exploze chová jako výbuch moba nebo bloku. Například ovlivňuje, zda vypadnou všechny bloky, které se zničí. To lze v Minecraftu nastavit pomocí *gamerule* a nastavuje se to zvlášť pro výbuchy mobů a bloků. Vybereme jednu z hodnot:

- `World.ExplosionSourceType.MOB`
- `World.ExplosionSourceType.BLOCK`

Celá metoda vytvoření exploze bude vypadat následovně:

```
1   world.createExplosion(lightning, blockPos.getX(), blockPos.getY(),
    blockPos.getZ(), 1f, true, World.ExplosionSourceType.MOB);
```

Celá metoda použití hůlky tedy bude vypadat následovně:

```
1  public TypedActionResult<ItemStack> use(World world, PlayerEntity user,
   Hand hand) {
2      if (world.isClient) {
3          return TypedActionResult.pass(user.getStackInHand(hand));
4      }
5
6      var offset = Vec3d.fromPolar(user.getPitch(), user.getYaw())
7              .normalize().multiply(10);
8
9      var pos = user.getEyePos().add(offset);
10
11     BlockPos blockPos = BlockPos.ofFloored(pos);
12     blockPos = world.getTopPosition(Heightmap.Type.MOTION_BLOCKING,
   blockPos);
13
14     LightningEntity lightning = new
   LightningEntity(EntityType.LIGHTNING_BOLT, world);
15
16     lightning.setPosition(blockPos.toCenterPos());
17     world.spawnEntity(lightning);
18     world.createExplosion(lightning, blockPos.getX(), blockPos.getY(),
   blockPos.getZ(), 4f, true, World.ExplosionSourceType.MOB);
19
20     return TypedActionResult.success(user.getStackInHand(hand));
21 }
```