29 Launcher blok - cooldown

V této lekci si u launcher bloku přidáme cooldown po aktivaci. Zároveň si ukážeme, jak změnit model bloku nastavením dat v kódu.

Vlastnosti bloků

Cooldown přidáme pomocí vlastnosti bloku. Některé bloky v Minecraftu mají vlastnosti, které se nastavují podle stavu bloku. Například tlačítka, páčka a podobné bloky mají vlastnost `powered`, která označuje, zda blok vydává redstone signál. Dále například pec má vlastnost `lit` (zda hoří) a `facing` (kterým směrem je otočená).

Vlastnosti si můžeme přidávat i vlastní a můžeme do nich ukládat stav našeho bloku. Dále z vlastnosti můžeme hodnotu i číst a na základě toho se rozhodnout, zda se provede nějaká část kódu nebo ne.

Dle hodnoty vlastnosti můžeme poté měnit i model bloku.

Přidání vlastnosti bloku

Vlastnost budeme přidávat ve třídě bloku - `LauncherBlock`.

Nejprve si vytvoříme proměnnou, do které si vlastnost uložíme. Vlastnost bude typu `boolean` - pravdivostní hodnota (`true` - pravda, `false` - nepravda).

src/main/java/com/example/block/LauncherBlock.java

```
1   public static final BooleanProperty COOLDOWN =
    BooleanProperty.of("cooldown");
```

Dále musíme vlastnost registrovat v metodě `appendProperties`:

src/main/java/com/example/block/LauncherBlock.java

```
1   @Override
2   protected void appendProperties(StateManager.Builder<Block, BlockState>
    builder) {
3       builder.add(COOLDOWN);
4       super.appendProperties(builder);
5   }
```

Poté upravíme konstruktor třídy a nastavíme této vlastnosti výchozí hodnotu. Základní hodnota bude `false`. Když bude aktivní cooldown, tak hodnotu nastavíme na `true`.

src/main/java/com/example/block/LauncherBlock.java

```
public LauncherBlock(Settings settings) {
    super(settings);
    setDefaultState(getDefaultState().with(COOLDOWN, false));
}
```

Odpočet času na cooldown

Odpočet budeme počítat v jednotce času `tick` - tak se počítá čas v Minecraftu. `1s = 20 ticků`.

Odpočet času uděláme tak, že si vytvoříme proměnnou, ve které si po aktivaci bloku nastavíme, jak dlouho má být cooldown aktivní např. `20 ticků`. Dále každý tick odečteme ze zbývajícího času `1` a jakmile bude odpočet `0`, tak přepneme blok zpátky do normálního stavu.

Vytvoříme si 2 proměnné.

- `cooldownTicks` - délka cooldownu
- `timer` - odpočet, kolik z cooldownu ještě zbývá

src/main/java/com/example/block/LauncherBlock.java

```
private final int cooldownTicks = 40;
private int timer = 0;
```

Samotný odpočet času uděláme pomocí metody `scheduledTick`. Tu můžeme spouštět z kódu tak, že řekneme, aby se naplánoval `tick` bloku. Následující tick se poté spustí tato metoda. Pokud tedy budeme tuto metodu volat stále dokola, tak se bude spouštět každý tick - tedy 20krát za sekundu.

V základu metoda vypadá následovně:

src/main/java/com/example/block/LauncherBlock.java

```
@Override
protected void scheduledTick(BlockState state, ServerWorld world, BlockPos
pos, Random random) {
    super.scheduledTick(state, world, pos, random);
}
```

Nejprve si zkontrolujeme, zda je hodnota v časovači (proměnná `timer`) větší, než `0`. Pokud ano, tak to znamená, že časovač ještě běží. Snížíme tedy hodnotu časovače o `1`

```
timer--;
```

a naplánujeme další spuštění metody `scheduledTick`. U plánování ticku uvádíme pozice bloku, objekt bloku (`this` je aktuální objekt bloku) a zpoždění, se kterým se má tick spustit.

```
world.scheduleBlockTick(pos, this, 0);
```

src/main/java/com/example/block/LauncherBlock.java

```
@Override
protected void scheduledTick(BlockState state, ServerWorld world, BlockPos
pos, Random random) {
    if (timer > 0) {
        timer--;
        world.scheduleBlockTick(pos, this, 0);
    }

    super.scheduledTick(state, world, pos, random);
}
```

Pokud časovač není větší než `0`, tak to znamená, že doběhnul. Můžeme přidat ještě kontrolu, zda je blok napájený redstonem, aby hráč nemohl mít jen zapnutou páčku a blok se neaktivoval pořád dokola.

Pokud tedy blok není napájený redstone signálem, tak nastavíme vlastnost `COOLDOWN` na hodnotu `false`.

Pomocí slova `else` můžeme zaručit, že pokud není splněná předchozí podmínka `timer > 0`, tak se spustí jiný kód. Toto můžeme dále zkombinovat s další kontrolou `else if` a tato můžeme zkontrolovat více možných situací.

src/main/java/com/example/block/LauncherBlock.java

```
@Override
protected void scheduledTick(BlockState state, ServerWorld world, BlockPos
pos, Random random) {
    if (timer > 0) {
        timer--;
        world.scheduleBlockTick(pos, this, 0);
    } else if (!world.isReceivingRedstonePower(pos)) {
        world.setBlockState(pos, state.with(COOLDOWN, false));
    }

    super.scheduledTick(state, world, pos, random);
}
```

Pokud je ale redstone signál stále aktivní, tak musíme tuto podmínku kontrolovat stále dokola. Proto přidáme ještě poslední možnost `else` a to bude opět naplánování ticku.

src/main/java/com/example/block/LauncherBlock.java

```java
@Override
protected void scheduledTick(BlockState state, ServerWorld world, BlockPos pos, Random random) {
    if (timer > 0) {
        timer--;
        world.scheduleBlockTick(pos, this, 0);
    } else if (!world.isReceivingRedstonePower(pos)) {
        world.setBlockState(pos, state.with(COOLDOWN, false));
    } else {
        world.scheduleBlockTick(pos, this, 0);
    }

    super.scheduledTick(state, world, pos, random);
}
```

Tímto máme odpočet hotový, ale ještě musíme upravit metodu `neighborUpdate` z předchozí lekce, tak aby se blok neaktivoval, když je cooldown aktivní. A když se blok aktivuje, tak aby se spustil cooldown.

Použití cooldown

Metoda `neighborUpdate` aktuálně vypadá následovně:

src/main/java/com/example/block/LauncherBlock.java

```java
@Override
protected void neighborUpdate(BlockState state, World world, BlockPos pos, Block sourceBlock, BlockPos sourcePos, boolean notify) {
    if (!world.isReceivingRedstonePower(pos)) {
        return;
    }

    Box box = new Box(pos.up());
    var entityList = world.getEntitiesByClass(Entity.class, box, Entity::isOnGround);
    for (var entity : entityList) {
        entity.addVelocity(0, 1, 0);
        entity.velocityModified = true;
    }

    super.neighborUpdate(state, world, pos, sourceBlock, sourcePos, notify);
}
```

Poté, co zkontrolujeme, zda je blok napájený redstonem, tak zkontrolujeme ještě, zda je aktivní cooldown. Hodnotu vlastnosti získáme pomocí `state.get(COOLDOWN)`. Pokud je tedy cooldown aktivní, tak se blok neaktivuje.

src/main/java/com/example/block/LauncherBlock.java

```
@Override
protected void neighborUpdate(BlockState state, World world, BlockPos pos,
Block sourceBlock, BlockPos sourcePos, boolean notify) {
    if (!world.isReceivingRedstonePower(pos)) {
        return;
    }

    if (state.get(COOLDOWN)) {
        return;
    }

    Box box = new Box(pos.up());
    var entityList = world.getEntitiesByClass(Entity.class, box,
Entity::isOnGround);
    for (var entity : entityList) {
        entity.addVelocity(0, 1, 0);
        entity.velocityModified = true;
    }

    super.neighborUpdate(state, world, pos, sourceBlock, sourcePos,
notify);
}
```

Pokud ale cooldown aktivní není, tak blok se blok aktivuje a vystřelí entity nahoru. Poté ale musíme spustit cooldown. To uděláme tak, že nastavíme proměnnou `timer` na délku cooldownu (proměnná `cooldownTicks`). Změníme hodnotu vlastnosti cooldown na `true` a naplánujeme spuštění metody `scheduledTick`.

src/main/java/com/example/block/LauncherBlock.java

```
@Override
protected void neighborUpdate(BlockState state, World world, BlockPos pos,
Block sourceBlock, BlockPos sourcePos, boolean notify) {
    if (!world.isReceivingRedstonePower(pos)) {
        return;
    }

    if (state.get(COOLDOWN)) {
        return;
    }

    Box box = new Box(pos.up());
    var entityList = world.getEntitiesByClass(Entity.class, box,
Entity::isOnGround);
    for (var entity : entityList) {
        entity.addVelocity(0, 1, 0);
```

```
            entity.velocityModified = true;
        }

        timer = cooldownTicks;
        world.setBlockState(pos, state.with(COOLDOWN, true));
        world.scheduleBlockTick(pos, this, 0);

        super.neighborUpdate(state, world, pos, sourceBlock, sourcePos,
    notify);
    }
```

Změna modelu podle hodnoty vlastnosti

Na to aby se automaticky změnil model, když se změní hodnota vlastnosti nám stačí změnit soubor s možnými stavy bloku ve složce `src/main/resources/assets/custom-block/blockstates`.

Aktuálně má náš blok jen jedinou variantu a soubor vypadá následovně:

src/main/resources/assets/custom-block/blockstates/launcher.json

```
{
  "variants": {
    "": {
      "model": "custom-block:block/launcher"
    }
  }
}
```

Pokud chceme více variant, tak musíme pro každou variantu určit jaká vlastnost má mít určitou hodnotu a na základě toho se nastaví model. Například tedy pokud je hodnota `cooldown=false`, tak se použije základní model a pokud je hodnota `cooldown=true`, tak se použije model `launcher_cooldown`.

Soubor tedy bude vypadat následovně:

src/main/resources/assets/custom-block/blockstates/launcher.json

```
{
  "variants": {
    "cooldown=false": {
      "model": "custom-block:block/launcher"
    },
    "cooldown=true": {
      "model": "custom-block:block/launcher_cooldown"
    }
  }
```

```
10    }
```

Tímto máme celou funkčnost hotovou a blok by nám měl fungovat. Pokud na něm tedy stojí nějaké entity a aktivujeme redstone, tak by entity měly vyletět nahoru, blok by se měl přepnout do stavu cooldown a po chvíli se změnit zpátky.