28 Launcher blok - přidání bloku, základní funkčnost

V této lekci budeme pokračovat v tvorbě *launcher* bloku, který vystřelí entity při aktivaci redstone signálem. Přidáme si blok do hry a zprovozníme základní funkčnost.

Blok budeme přidávat do módu `custom-block`, který jsme si tvořili v průběhu roku.

Přidání bloku do hry

Z předchozí lekce bychom měli mít vytvořené 2 bloky a uložené ve formátu `.zip`. Pro oba bloky bude postup stejný.

Model a textura

`.json` soubory (modely) přesuneme do složky `src/main/resources/assets/custom-block/models/block`
`.png` soubory (textury) přesuneme do složky `src/main/resources/assets/custom-block/textures/block`

Stav bloku

Ve složce `src/main/resources/assets/custom-block/blockstates` si vytvoříme soubor `launcher.json`, kde definujeme varianty bloku. Zatím bude mít blok jen jednu variantu. Až přidáme i cooldown pro aktivaci, tak si soubor upravíme.

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

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

Stav bloku

Více informací najdete v lekci 14 Přidání bloku do inventáře, textura bloku - Stav bloku

Item bloku

Ve složce `src/main/resources/assets/custom-block/models/item` si vytvoříme soubor `launcher.json`, kde určíme, jaký model se má použít na zobrazení v inventáři.

src/main/resources/assets/custom-block/models/item/launcher.json

```
{
  "parent": "custom-block:block/launcher"
}
```

> **Model itemu**
>
> Více informací najdete v lekci 15 Model itemu, loot table - Model itemu

Registrace bloku v kódu

Protože budeme bloku měnit jeho chování, tak si nejprve vytvoříme novou třídu bloku. Tu vytvoříme ve složce `src/main/java/com/example/block` a pojmenujeme ji `LauncherBlock`. Pokud máte nainstalované rozšíření pro tvorbu módů, tak můžete při tvorbě souboru zvolit možnost *Minecraft Class* a poté vybrat *Block*.

Výsledný soubor by měl vypadat následovně:

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

```
public class LauncherBlock extends Block {
    public LauncherBlock(Settings settings) {
        super(settings);
    }
}
```

Dále musíme blok registrovat ve třídě `ModBlocks`.

Nejprve přidáme klíč (id), pod kterým se registruje blok i item.

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

```
public static final RegistryKey<Block> LAUNCHER_BLOCK_KEY =
RegistryKey.of(
        RegistryKeys.BLOCK,
        Identifier.of(CustomBlock.MOD_ID, "launcher")
);
```

Poté registrujeme samotný blok.

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

```
public static final Block LAUNCHER_BLOCK = register(
        new LauncherBlock(AbstractBlock.Settings.create()),
        LAUNCHER_BLOCK_KEY,
        true
);
```

Nyní bychom měli mít blok ve hře. Zatím jej získáme jen pomocí příkazu `/give`.

Funkčnost bloku

Funkčnost bloku budeme přidávat ve třídě `LauncherBlock`. Využijeme k tomu metodu `neighborUpdate`, která se spouští pokaždé, když se aktualizuje blok v okolí. To můžeme využít na kontrolu, zda je náš blok napájený redstone signálem a poté vystřelit entity.

Metoda vypadá následovně:

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

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

Kontrola redstone signálu

Jako první si zkontrolujeme, zda je aktuální blok napájený redstone signálem. Na to můžeme využít metodu `isReceivingRedstonePower` ze třídy `World`, kde jako parametr uvádíme pozice bloku, který chceme zkontrolovat. V tomto případě to bude aktuální blok a jeho pozice máme uložené v parametru `pos`. Pokud blok není (zápor kontroly se dělá pomocí vykřičníku `!`) napájený redstone signálem, tak dále pokračovat nebudeme.

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;
    }

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

Výběr entit

Dále vezmeme všechny entity, které stojí na bloku a vystřelíme je nahoru.

Na to použijeme metodu výběru entit v určité oblasti. Nejprve si určíme oblast, ze které budeme vybírat entity. Na to si vytvoříme proměnnou typu `Box`. Oblast můžeme určit i pomocí pozice bloku a protože známe pozici aktuálního bloku, tak stačí použít blok o 1 výše, což můžeme udělat pomocí metody `up` ze třídy `BlockPos`.

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

```
1  @Override
2  protected void neighborUpdate(BlockState state, World world, BlockPos pos,
   Block sourceBlock, BlockPos sourcePos, boolean notify) {
3      if (!world.isReceivingRedstonePower(pos)) {
4          return;
5      }
6
7      Box box = new Box(pos.up());
8
9      super.neighborUpdate(state, world, pos, sourceBlock, sourcePos,
   notify);
10 }
```

Dále vybereme všechny entity v této oblasti pomocí metody `getEntitiesByClass` ze třídy `World`. Jako parametry uvádíme třídu, ze které má entita být - pokud chceme všechny entity, tak můžeme použít třídu `Entity`. Pokud chceme jen živé entity, tak můžeme použít třídu `LivingEntity`.

Dále uvádíme oblast - `box`.

A jako poslední uvádíme funkci, kterou filtrujeme entity. My chceme vybrat entity, které stojí na zemi, takže můžeme použít metodu `isOnGround` ze třídy `Entity`. Pro zjednodušený zápis můžeme použít `Entity::isOnGround`. Jinak by zápis vypadal následovně `entity -> entity.isOnGround()`.

Výsledný seznam entit si uložíme do proměnné `entityList`.

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

```
1  @Override
2  protected void neighborUpdate(BlockState state, World world, BlockPos pos,
   Block sourceBlock, BlockPos sourcePos, boolean notify) {
3      if (!world.isReceivingRedstonePower(pos)) {
4          return;
5      }
6
7      Box box = new Box(pos.up());
8      var entityList = world.getEntitiesByClass(Entity.class, box,
   Entity::isOnGround);
```

```
9
10      super.neighborUpdate(state, world, pos, sourceBlock, sourcePos,
    notify);
11  }
```

Přidání rychlosti entitě

Nyní projdeme seznam entit pomocí smyčky `for` a každou entitu vystřelíme nahoru.

Na to abychom entitu vystřelili stačí, abychom jí přidali rychlost ve směru nahoru. To uděláme pomocí metody `addVelocity`, kde jako parametry uvádíme, jakou rychlost chceme přidat v jednotlivých směrech - `x`, `y`, `z`. Nás zajímá směr nahoru - `y`.

Tato metoda ale samostatně nefunguje pro hráče a proto musíme ještě u entity nastavit, že má změněnou rychlost. To uděláme nastavením vlastnosti `velocityModified` na hodnotu `true`.

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

```
1   @Override
2   protected void neighborUpdate(BlockState state, World world, BlockPos pos,
    Block sourceBlock, BlockPos sourcePos, boolean notify) {
3       if (!world.isReceivingRedstonePower(pos)) {
4           return;
5       }
6
7       Box box = new Box(pos.up());
8       var entityList = world.getEntitiesByClass(Entity.class, box,
    Entity::isOnGround);
9       for (var entity : entityList) {
10          entity.addVelocity(0, 1, 0);
11          entity.velocityModified = true;
12      }
13
14      super.neighborUpdate(state, world, pos, sourceBlock, sourcePos,
    notify);
15  }
```

Nyní by nám blok měl fungovat a když se na něj postavíme nebo na něm stojí nějaké entity a spustíme redstone, tak by všechny entity měly vyletět směrem nahoru.

Příště si doděláme cooldown po aktivaci a zároveň si ukážeme, jak měnit model bloku na základě nastavení hodnoty.