B0 - HUB

B-INN-000

Mod Minecraft

Créer son premier mod Minecraft

Mod Minecraft

language: java
build tool: gradle

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

Workshop Mod Minecraft

Pré-requis

Si vous n'avez pas déjà installé les outils nécessaires à ce workshop suivez les instructions situées ici :

`https://github.com/alwyn974/workshop-mc-mod/blob/master/REQUIREMENTS.md`

Configuration du workspace

On va commencer par modifier le `mcmod.info` pour y mettre les informations nécessaires.
Il faut modifier le `modid` pour mettre le vôtre (uniquement des lettres minuscules, sans espace).
Le reste à vous de modifier en fonction:

- modid : le modid de votre mod, remplacez donc examplemod par votre modid
- name : le nom du mod, à nouveau ce que vous avez mis dans la classe principale de votre mod
- description : une courte description du mod, vous pouvez ne rien mettre en laissant juste " ".
- version : la version du mod, ne changez rien ici ${version} est automatiquement remplacé par la valeur se trouvant dans le build.gradle
- mcversion : la version de minecraft, comme pour la version, le remplacement est automatique
- url : le lien vers la présentation de votre mod
- updateUrl : l'url pour télécharger le mod
- authorList : la liste des auteurs. ["auteur1", "auteur2", "auteur3"] pour plusieurs auteurs, ["auteur"] pour un seul auteur
- credits : créditez ici les éventuels contributeurs.
- logoFile : le chemin à partir du début de l'archive vers le logo du mod. Si votre logo se trouve directement dans src/main/resouces et s'appelle logo.png il faudra mettre logo.png

- screenshots : une liste de screenshots, fonctionne comme pour le logo (chemin à partir du début de l'archive)
- dependencies : les dépendances, je conseille plutôt d'utiliser le String dépendance de l'annotation **@**Mod.

En suite, on va rename le package de base créer par le MDK, renommé donc `com.example.examplemod` par votre package.
Renommez aussi la classe `ExampleMod` par votre `Modid`Mod

Pensez aussi à faire les modifications dans le fichier ExampleMod.java

Préparation pour créer un item

On va devoir déclarer chaque item dans une classe spéciale, pour cela on va créer `ModidItems.java` dans le package `item` qui contiendra :

- Une ArrayList en publique, statique et finale
- Chaque Item qu'on va devoir créer sera déclaré dans cette classe en statique et final

Vous allez devoir ajouter ces deux méthodes à cette classe, elles seront utiles pour enregistrer les Items plus facilement.

```
public static void setItemName(Item item, String name) {
    item.setRegistryName(ModidMod.MODID, name).setUnlocalizedName(ModidMod.MODID
        + "." + name);
    ITEMS.add(item);
}

public static void setItemBlockName(Item item, Block block) {
    item.setRegistryName(block.getRegistryName());
    ITEMS.add(item);
}
```

Bien sûr, il faut remplacer `ModidMod` par votre classe.

Ensuite on va devoir ajouter une méthode qui s'occupera d'enregistrer les models de nos items. Pour cela on va devoir utiliser un Event de Forge.
Vu que l'on va enregistrer le rendu de nos items, il faudra le faire uniquement coté client et pour cela on devra utiliser une **annotation** pour indiquer à forge d'exécuter le code uniquement coté client.

```
@SideOnly(Side.CLIENT) //Forge annotation for Side managing
@SubscribeEvent //Forge annotation to subscribe to an event
protected static void registerItemModels(ModelRegistryEvent e) {
    ITEMS.forEach(item -> registerModel(item, 0)); //A simple forEach to call the
        method below
}

@SideOnly(Side.CLIENT)
protected static void registerModel(Item item, int metadata) {
    ModelLoader.setCustomModelResourceLocation(item, metadata, new
        ModelResourceLocation(item.getRegistryName(), "inventory"));
}
```

Actuellement notre Event ne fonctionnera pas, c'est parce qu'il manque une annotation qui permet à forge de trouver notre classe. Je vous invite donc à chercher sur la documentation comment ajouter celle-ci.
Plus précisément vous devez trouver comment enregistrer un Event statique (vu que notre méthode est statique).
Je vous conseille de CTRL+Clique gauche sur l'annotation pour voir quel type de paramètre elle prend.

Concepts de forge: Les sides

Plus d'informations sur les Events : Documentation Forge

Maintenant, il nous manque l'event pour enregistrer les items.

- Créez un package `register`
- Créez une classe `RegistryHandler`
- Enregistrez le en tant qu'EventSubscriber, sans modifier les sides
- Déclarez une méthode `registerItems` qui prendra en paramètre `RegistryEvent.Register<Item> event` (ne pas oublier l'annotation `@SubscribeEvent` sinon l'event ne sera pas prit en compte)
- Et pour chaque Item dans la liste `ModidItems.ITEMS` appeler la méthode `getRegistry().register` de l'event

Création d'un item simple

Creation de la classe BasicItem

Créez une classe `BasicItem` dans le package item :

- Elle devra hériter d'`Item` qui provient du package `net.minecraft.item`
- Elle aura un constructeur qui prend une `String name` en paramètre
- Et dans son constructeur, on devra appeler notre méthode `ModidItems.setItemName()`

Ensuite dans votre classe `ModidItems` vous aller déclarer une nouvelle variable publique, statique, finale, de type `Item` et l'initialiser avec `BasicItem`

Et voilà on a créé un item simple, mais il n'est pas encore fonctionnel.

Création du CreativeTabs

Pour organiser un peu mieux nos items, on va créer un CreativeTabs. Pour cela on va créer une classe `ModidCreativeTab` qui héritera de `CreativeTabs`.

- Il faudra un constructeur vide, qui appellera le constructeur parent avec `ModidMod.MODID + "_tab"`
- Et surcharger la méthode `getTabIconItem` pour qu'elle renvoie un `ItemStack` de notre Item qu'on vient de créer

Maintenant on peut déclarer dans notre `ModidMod` une variable publique, statique, finale, de type `CreativeTabs` et initialisé avec notre propre CreativeTabs.
On va modifier notre classe `BasicItem` pour ajouter dans le constructeur un appel à la méthode `setCreativeTab` comme ça chacun de nos items seront trié dans ce CreativeTabs

Création du model

Si vous avez lancé votre jeu précédemment, vous avez vu que notre item n'a pas de texture et n'a pas de nom aussi. On va commencer par lui ajouter une texture.

Pour cela il va falloir créer plusieurs sous dossiers dans le dossier `src/main/resources` :
Créer un dossier `assets/modid/` remplacer modid par le votre
Dans ce dossier créer un dossier :

- lang/
- textures/
- textures/items
- textures/blocks/
- textures/models/
- models/
- models/item
- models/block
- blockstates/

Dans le dossier `assets/modid/models/item/` créer un fichier en fonction du nom que vous avez donné à votre item (pour moi, ça sera `basic_item.json`)
Dans ce json vous aller devoir mettre:

```
{
  "parent": "item/generated",
  "textures": {
    "layer0": "modid:items/votre_texture"
  }
}
```

La texture devra être dans le dossier `assets/modid/textures/items/`

Si vous avez fait tout correctement en lançant votre jeu vous aurez une texture. Si vous n'avez pas de texture, vous pouvez utiliser celle fourni sur le github du workshop `basic_item.png`

Creation du fichier lang

Comme vous avez pu remarquer notre Item et CreativeTab ont des noms inconnus. Pour remédier à cela on va devoir créer un fichier `en_us.lang` dans le dossier lang (crée précédemment).
Dans ce fichier, il faudra mettre chaque nom non traduit.

Exemple:

```
itemGroup.modid_tab=Votre nom de CreativeTab
item.modid.votre_item.name=Votre nom d'item
```

N'oubliez pas de remplacer modid par le votre
Maintenant quand vous lancerez votre jeu, il y aura un nom pour le creative tab et un nom pour l'item

Préparation à la création d'un block

Comme pour les items, on va créer une classe qui contient tout nos Block.
Créer une classe `ModidBlocks` dans le package `block`

- Une ArrayList en publique, statique et finale
- Chaque Block qu'on va devoir créer sera déclaré dans cette classe en statique et final

Vous allez devoir créer comme pour les Items, une méthode pour mettre le nom du block et ajouter à la liste de blocks.
Son prototype sera `public static void setBlockName(Block block, String name)`

On va devoir modifier la classe `RegistryHandler` du package register, pour ajouter l'event d'enregistrement des blocks.
Vous allez devoir y ajouter une méthode :

- Déclarer une méthode `registerBlocks` et prendra en paramètre `RegistryEvent.Register<Block> event`
- Pour chaque block dans la liste `ModidBlocks.BLOCKS` appeler la méthode `getRegistry().register` de l'event

Ne pas oublier l'annotation pour recevoir un event : `@SubscribeEvent`

Création d'un block basique

Création de l'ItemBlock

Pour avoir notre block en jeu, on va d'abord devoir créer un ItemBlock.

Créez une classe `BasicItemBlock` dans le package item :

- Elle devra hériter d'ItemBlock qui provient du package `net.minecraft.item`
- Elle aura un constructeur qui prend une `Block block` en paramètre
- Et dans son constructeur, on devra appeler notre méthode `ModidItems.setItemBlockName()`
- Et appeler la méthode `setCreativeTab` avec le notre

Création du Block

Créer une classe `BasicBlock` dans le package `block`:

- Elle devra hériter de `Block` qui provient du package `net.minecraft.block`
- Elle aura un constructeur qui prend une `String name, Material material` en paramètre
- Dans son constructeur, on devra appeler notre méthode `ModidBlocks.setBlockName`
- On va aussi mettre le `harvestLevel` du block, cela déterminera avec quel outil et de quel niveau notre block sera cassable
- Mettre un harvest level de niveau 2 et pour une pioche

  Ensuite dans votre classe `ModidBlocks` vous aller déclarer une nouvelle variable publique, statique, finale de type `Block` et l'initialiser avec `BasicBlock`.
  Maintenant qu'on a déclaré notre block, on a besoin de créer son item qui lui ait lié (aka `ItemBlock`).
  Dans la classe `ModidItems` ajouter une nouvelle variable de type `Item` et l'initialiser avec `BasicItemBlock`

Ne pas oublier d'assigner le block au `CreativeTab` qu'on a créé

Pour le material du block, mettez juste `Material.ROCK`

Création du model

Comme pour notre item, on va devoir créer un json pour notre block, plus précisément 3 jsons par block.
Le premier json devra être dans le dossier `assets/modid/blockstates/`, le second dans `assets/modid/models/block/` et le dernier dans `assets/modid/models/item`.
Les 3 fichiers devront avoir le même nom, plus précisément le nom que vous avez donné lors de l'instanciation du block (pour moi, ça sera `basic_block.json`)
Dans ces 3 json vous allez devoir mettre :

- Celui dans blockstates

```
{
  "variants": {
    "normal": {
      "model": "modid:votre_block"
    }
  }
}
```

- Celui dans item

```
{
  "parent": "modid:block/votre_block"
}
```

- Celui dans block

```
{
  "parent": "block/cube_all",
  "textures": {
    "all": "modid:blocks/votre_texture"
  }
}
```

La texture devra être dans le dossier `assets/modid/textures/blocks/`

Si vous avez fait tout correctement en lançant votre jeu vous aurez une texture. Si vous n'avez pas de texture, vous pouvez utiliser celle fourni sur le github du workshop `basic_block.png`.
Comme pour les items il faudra ajouter une valeur à notre fichier `en_us.lang` pour chaque block
Exemple : `tile.modid.votre_block.name=Votre nom de block`
N'oubliez pas de remplacer modid par le votre

AJOUTER UN CRAFT

Ajoutons un craft pour notre block, en utilisant notre `BasicItem`, pour cela il faudra dans le dossier `assets/modid/recipes/` ajouter un fichier json contenant le craft.
Le nom du json déprendra du craft, par exemple: `basic_block_from_basic_item.json`
Dans ce json il faudra mettre :

```
{
  "type": "minecraft:crafting_shaped",
  "group": "basic_block",
  "pattern": [
    "###",
    "###",
    "###"
  ],
  "key": {
    "#": {
      "item": "modid:item_to_use"
    }
  },
  "result": {
    "item": "modid:item_to_get"
  }
}
```

Le paterne est libre à vous, il faudra juste ne pas dépasser un 3x3