Презентация: Разработка модификаций для Minecraft с использованием Fabric

В этой презентации мы рассмотрим основы разработки модификаций для Minecraft с использованием Fabric, популярного инструмента для создания модов.

Введение

Популярность Minecraft

Minecraft – популярная игра, поддерживающая модификации.

Преимущества Fabric

Fabric – один из наиболее удобных инструментов для создания модов, предлагающий гибкость и высокую производительность.

Основы Fabric

1

API и Loader

Fabric включает в себя API и Loader, которые позволяют разрабатывать и загружать моды.

2

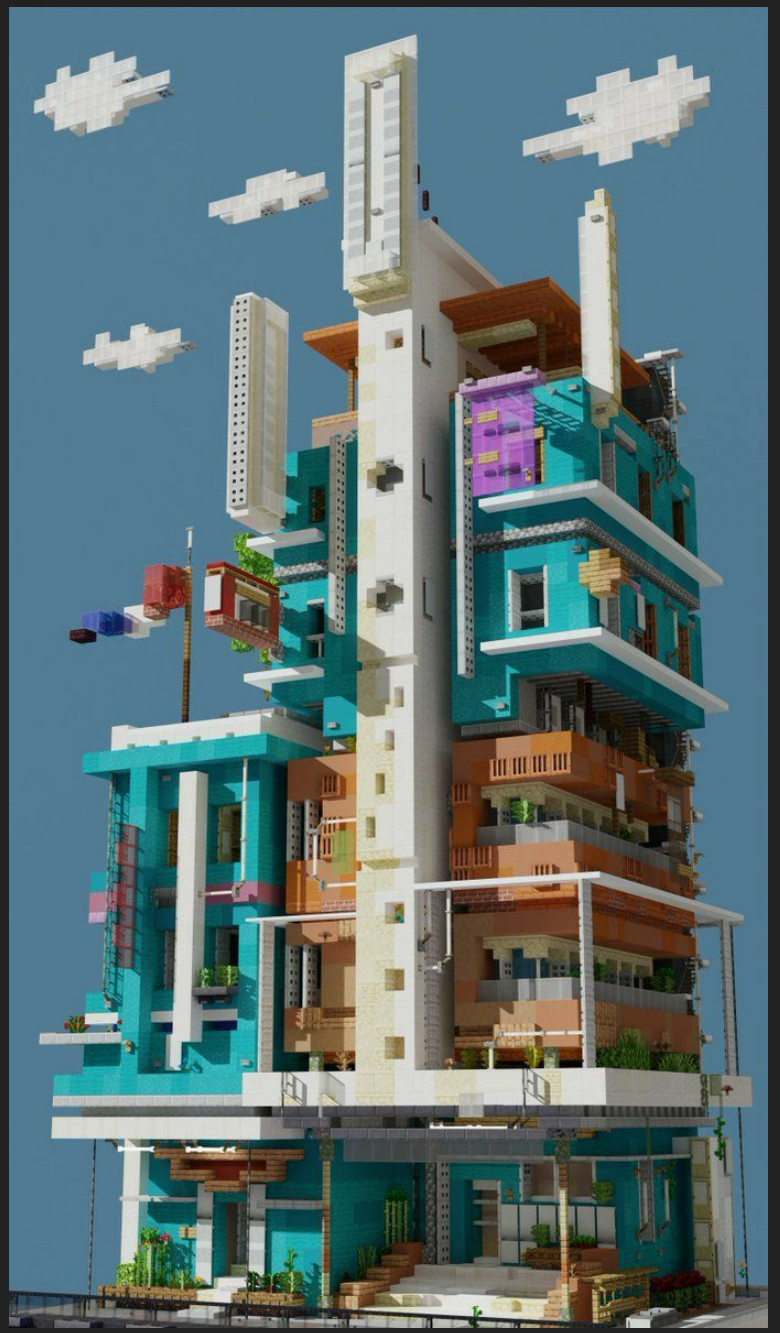
Точка входа

Основной точкой входа в мод является метод onInitialize().

3

Инструменты

Fabric API предоставляет удобные инструменты для взаимодействия с механиками игры.



ООП в разработке модов

Инкапсуляция

Разделение данных и логики.

Наследование

Создание новых объектов на основе существующих.

Полиморфизм

Универсальные интерфейсы для разных классов.

```
    ServerPlayConnectionEvents.JOIN.register(( ServerPlayNetworkHandler handler,  PacketSender sender,  MinecraftServer server) -> {
        ServerPlayerEntity player = handler.getPlayer();
        GiveOfClassesForPlayers giveOfClassesForPlayers = new GiveOfClassesForPlayers();
        if (player instanceof CustomClassData customData) {
            customData.setCustomClass(giveOfClassesForPlayers.getClassToUser(player));
        }
    });
    CommandRegistrationCallback.EVENT.register(( CommandDispatcher<ServerCommandSource> dispatcher,  CommandRegistryAccess registryAccess,
        CustomCommands.register(dispatcher, registryAccess);
    });
    ServerLivingEntityEvents.ALLOW_DAMAGE.register(( LivingEntity entity,  DamageSource source,  float amount) -> {
        if (source.getAttacker() instanceof PlayerEntity attacker && entity instanceof PlayerEntity target) {
            if (attacker instanceof CustomClassData customAttacker && target instanceof CustomClassData customTarget) {
                if (customAttacker.getParty().equals(customTarget.getParty())) {
                    attacker.sendMessage(Text.of( string: "You can't bite your teammate! It's terrible!!"));
                    return false;
                }
            }
        }
        return true;
    });
    LOGGER.info("Hello Fabric world!");
}
```

Регистрация команд

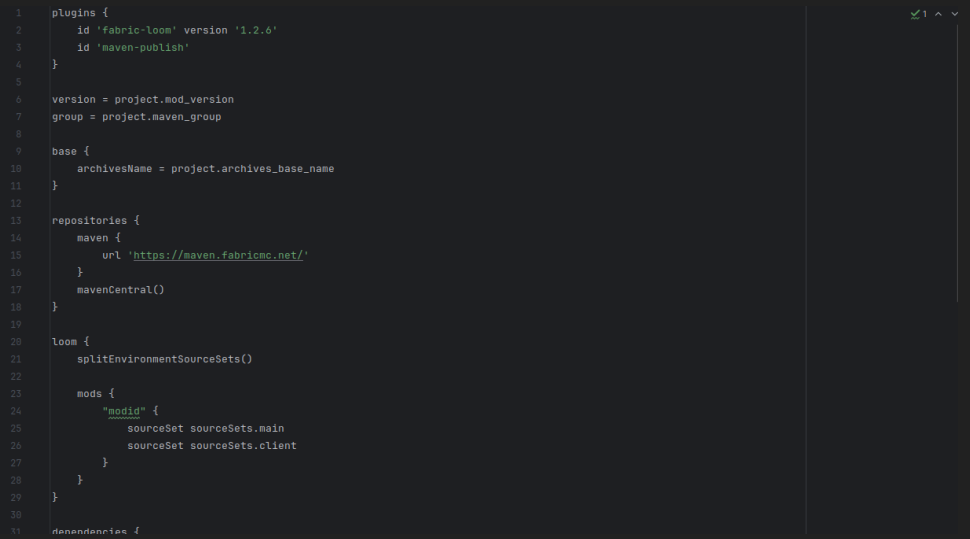
Взаимодействие

Команды позволяют игрокам взаимодействовать с механиками мода.

Пример

Пример создания команды:

```
CommandManager.literal("createteam") .executes(context -> { PlayerEntity player = context.getSource().getPlayer(); players.put(0, player); playersInParty.put(player.getName().getString(), players); player.sendMessage(Text.of("Команда создана!")); return Command.SINGLE_SUCCESS; });
```



```
plugins {
    id 'fabric-loom' version '1.2.6'
    id 'maven-publish'
}

version = project.mod_version
group = project.maven_group

base {
    archivesName = project.archives_base_name
}

repositories {
    maven {
        url 'https://maven.fabricmc.net/'
    }
    mavenCentral()
}

loom {
    splitEnvironmentSourceSets()

    mods {
        "modid" {
            sourceSet sourceSets.main
            sourceSet sourceSets.client
        }
    }
}

dependencies {
```

Работа с билд-системой

1. **Gradle**

   Fabric использует Gradle для сборки модов.

2. **build.gradle**

   Описывает зависимости и настройки сборки.

3. **fabric.mod.json**

   Конфигурация мода.

```
    ServerPlayConnectionEvents.JOIN.register(( ServerPlayNetworkHandler handler, PacketSender sender, MinecraftServer server) -> {
        ServerPlayerEntity player = handler.getPlayer();
        GiveOfClassesForPlayers giveOfClassesForPlayers = new GiveOfClassesForPlayers();
        if (player instanceof CustomClassData customData) {
            customData.setCustomClass(giveOfClassesForPlayers.getClassToUser(player));
        }
    });
    CommandRegistrationCallback.EVENT.register(( CommandDispatcher<ServerCommandSource> dispatcher, CommandRegistryAccess registryAccess,
        CustomCommands.register(dispatcher, registryAccess);
    });
    ServerLivingEntityEvents.ALLOW_DAMAGE.register(( LivingEntity entity, DamageSource source, float amount) -> {
        if (source.getAttacker() instanceof PlayerEntity attacker && entity instanceof PlayerEntity target) {
            if (attacker instanceof CustomClassData customAttacker && target instanceof CustomClassData customTarget) {
                if (customAttacker.getParty().equals(customTarget.getParty())) {
                    attacker.sendMessage(Text.of( string: "You can't bite your teammate! It's terrible!!"));
                    return false;
                }
            }
        }
        return true;
    });
    LOGGER.info("Hello Fabric world!");
}
```

Оптимизация и тестирование

1

Юнит-тестирование

Проверка отдельных компонентов мода.

2

Интеграционное тестирование

Тестирование работы мода с другими модификациями.

3

Игровые тесты

Проверка механик внутри игры.

Итоги и перспективы

API

Fabric предоставляет мощные инструменты для создания модификаций.

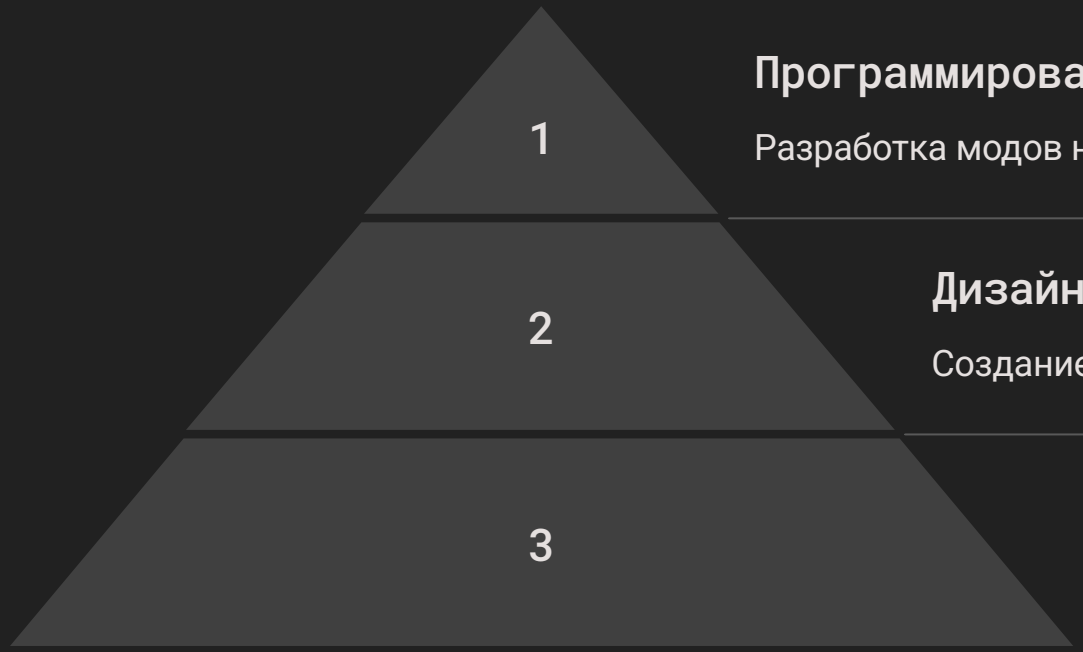
Механики

Добавление предметов, эффектов и механик.

Развитие

Добавление более сложных классов и фракций.

Заключение



Программирование

Разработка модов на Fabric – увлекательный процесс.

Дизайн

Создание уникальных модов, расширяющих возможности Minecraft.

Сообщество

Работа с сообществом.

Спасибо за внимание!