**ДОКУМЕНТАЦИЯ**

**ПО API**

**MINDUSTRY**

автор: Sosiska мой [github](https://github.com/sosiska565)

Оглавление

1 Глава: Установка и описание MindustryJavaModTemplate

1.1……………………………………………………………………………..что такое MindustryJavaModTemplate

1.2…………………………………………………………………………….установка MindustryJavaModTemplate

1.3………………………………………………………………………………………………………..структура мода

1.4……………………………………………………………………………………………….…….основные файлы

2 Глава: Написание мода, разбор главных классов

2.1……………..…………………………………………………………………………………………описание мода

2.2…………………………………………………………………………………………………………главный файл

2.3……………………………………………………………………………………………………..основные классы

2.4….…….…..………………………………………………………………………….создание игрового предмета

2.5……….…………………………………………………………………………………………..свойства предмета

## 3 Глава: Создание блока

3.1…………………………………………………………………………………………………………классы блоков

3.2….…………………………………………………………………………………………………..создание турели

1 Глава — Установка MindustryJavaModTemplate

1.1 Что такое *MindustryJavaModTemplate, это шаблон java мода выложенный создателем игры Anuken на GitHub и предоставляет возможность создавать моды.*

1.2 Как установить?

Все что вам потребуется для установки *MindustryJavaModTemplate это зайти на странцу в гитхабе* [*MindustryJavaModTemplate*](https://github.com/Anuken/MindustryJavaModTemplate) *и либо скачать зип архив, либо же клонировать репозиторий с помощью Git[[1]](#footnote-2).* Если вы скачали зип архив распакуйте его в любую папку.

1.3 Структура мода.

Структура мода будет выглядеть следующим образом:

MindustryJavaModTemplate/

├── assets/

│ ├── bundles/

│ │ └── bundle.properties

│ ├── sprites/

│ │ ├── sprites/

│ │ │ └── example-block.png

│ │ └── mod.hjson

├── gradle/

│ ├── wrapper/

│ │ ├── gradle-wrapper.jar

│ │ └── gradle-wrapper.properties

├── src/

│ └── example/

│ ├── ExampleMod.java

│ └── content/

│ └── ExampleContent.java

├── .gitignore

├── build.gradle

├── gradlew

├── gradlew.bat

├── LICENSE

├── README.md

└── settings.gradle

Примерно так должна выглядеть базовая структура Шаблонного мода.

1.4 В папке sprites мы будем хранить все спрайты и изображения, найдите у себя файл mod.hjson, он отвечает за описание мода, по пути src/example/ хранятся основные java файлы и классы.

Все что я описал выше нам понадобится для создание мода, остальное можно не трогать оно отвечает за сборку мода.

2 Глава — Создание мода.

2.1 Первое что нам нужно сделать так это настроить описание мода, откройте файл с названием mod.hjson это описание нашего мода в игре, вы увидите что-то типа этого:

#the mod name as displayed in-game

displayName: "TestMod"

#the internal name of your mod

name: "TestMod"

#your name

author: "You"

#the fully qualified main class of the mod

main: "example.ExampleJavaMod"

#the mod description as seen in the mod dialog

description: "A Mindustry Java mod template."

#the mod version

version: 1.0

#the minimum game build required to run this mod

minGameVersion: 145

#this is a java mod

java: true

displayName - отвечает за отображаемое имя мода в игре,

name - отвечает за локальное имя мода,

author - отвечает за отображение имени автора в игре,

main - отвечает за главный класс, в нашем случае это ExampleJavaMod.java, учтите что в main не нужно указывать расширение файла. И так же нужно указать полный путь без src, то есть: example.ExampleJavaMod

все что начинается с „ # “ это комментарии.

2.2 Перейдите по пути src/example/ExampleJavaMod.java, это наш главный файл, от сюда мод начинает свою работу.

Листинг 1.0 (ExampleJavaMod.java)

package example;

import arc.\*;

import arc.util.\*;

import mindustry.\*;

import mindustry.mod.\*;

public class ExampleJavaMod extends Mod{

    @Override

    public void loadContent(){

    }

}

я советую вам вставить данный код в свой главный файл.

2.3 Вот основные пакеты с которыми нам представит поработать:

**mindustry.mod**:

Mod: Базовый класс для всех Java-модов. Все моды должны содержать класс, наследующий mindustry.mod.Mod.

Plugin: Класс для серверных плагинов, которые не требуют загрузки на стороне клиента.

**mindustry.content**:

Классы для создания игрового контента: Items, Blocks, Units, Liquids, StatusEffects, и т.д.

Пример: Blocks используется для создания новых блоков, таких как турели или конвейеры.

**mindustry.world**:

Пакет для работы с игровым миром, включая Tile, Building, Block.

Пример: world.blocks.production для создания производственных блоков.

**mindustry.entities**:

Управление юнитами, эффектами (Effect), пулями (BulletType).

Пример: entities.comp.UnitComp для кастомизации поведения юнитов.

**mindustry.ui**:

Для создания пользовательских интерфейсов, таких как меню или информационные панели.

**mindustry.game**:

Работа с игровыми событиями (EventType) и правилами игры (Rules).

**mindustry.graphics**:

Управление визуальными эффектами, текстурами и рендерингом.

2.4 Создание игрового предмета.

Игровой предмет(item) выполняет роль объекта, который можно собирать, хранить, производить и использовать в игре, это как слиток железа в игре Minecraft он выпадает из руды и из него можно сделать другой полезный предмет.

Чтобы создать предмет нам понадобится создать новый Java файл, я вам рекомендую назвать его ItemsRegistration, потому что я рекомендую создать один файл с регистрацией чего либо, будь то BlocksRegistration или UnitsRegistration, поверьте так удобней чем создавать отдельный файл для каждого объекта в вашем моде.

Создадим папку items в подпапке example, в папке items создадим файл с названием ItemsRegistration.java. Первое что нам нужно сделать подключить пакет „example.items“ Далее импортируйте классы „mindustry.type.Item“ и „arc.graphics.Color“ первый нужен чтобы создать объект типа Item, а второй для создания цветов.

В главном классе файла „ItemsRegistration“ нужно создать функцию „public void Load()“ В этой функции мы будем инициализировать все объекты. Перед функцией, нужно создать объект типа „Item” я создал железо: „Item iron“ В функции нам нужно инициализировать объект и делается это следующим образом:

Листинг 1.2 (ItemsRegistration.java)

package example.items;

import arc.graphics.Color;

import mindustry.type.Item;

public class ItemsRegistration {

    public static Item iron;

    public static void Load(){

        iron = new Item("iron", Color.darkGray){{

            hardness = 2;

            cost = 1f;

            flammability = 0f;

            explosiveness = 0f;

            radioactivity = 0f;

            charge = 0f;

            alwaysUnlocked = true;

        }};

    }

}

Первым аргументом в конструктор класса передается имя предмета, а вторым его цвет в моем случает темно серый.

2.5 Далее идут свойства предмета, честно скажу их очень много и запомнить это почти невозможно, а для блока их еще больше. >.<

// Основные свойства предмета

color = Color.valueOf("ffaa5f"); // Цвет предмета, отображается в игре (например, на конвейерах). Принимает объект Color. (Требуется: import arc.graphics.Color; задается через конструктор Item или базовый класс UnlockableContent)

hardness = 0; // Твердость предмета. Определяет, сколько времени требуется для добычи (для руд). Значение от 0 (мгновенно) до 10. (Требуется: доступен напрямую в mindustry.type.Item)

cost = 1f; // Стоимость предмета при использовании в строительстве. Влияет на баланс (например, 1f — стандартная стоимость). (Требуется: доступен напрямую в mindustry.type.Item)

flammability = 0f; // Горючесть предмета (0f–1f). Если > 0, предмет может гореть (например, уголь имеет высокую горючесть). (Требуется: доступен напрямую в mindustry.type.Item)

explosiveness = 0f; // Взрывоопасность предмета (0f–1f). Если > 0, предмет может взрываться при горении (например, пирит). (Требуется: доступен напрямую в mindustry.type.Item)

radioactivity = 0f; // Радиоактивность предмета (0f–1f). Если > 0, предмет излучает радиацию, влияя на окружающую среду. (Требуется: доступен напрямую в mindustry.type.Item)

charge = 0f; // Заряд предмета (0f–1f). Используется для механик, связанных с электричеством (например, в генераторах). (Требуется: доступен напрямую в mindustry.type.Item)

// Свойства для логистики и отображения

hidden = false; // Если true, предмет скрыт из пользовательского интерфейса (например, в меню строительства). По умолчанию false. (Требуется: доступен напрямую в mindustry.type.Item)

buildable = true; // Если false, предмет нельзя использовать для строительства блоков. По умолчанию true. (Требуется: доступен напрямую в mindustry.type.Item)

alwaysUnlocked = false; // Если true, предмет разблокирован с самого начала игры. По умолчанию false (требуется исследование). (Требуется: доступен через базовый класс mindustry.type.UnlockableContent)

// Визуальные и звуковые эффекты

lowPriority = false; // Если true, предмет имеет низкий приоритет в логистике (например, фильтры на конвейерах будут отдавать предпочтение другим предметам). (Требуется: доступен напрямую в mindustry.type.Item)

frames = 0; // Количество кадров анимации для предмета (для анимированных предметов). 0 — без анимации. (Требуется: доступен напрямую в mindustry.type.Item)

transitionFrames = 0; // Количество переходных кадров между анимациями (для плавного переключения). 0 — без перехода. (Требуется: доступен напрямую в mindustry.type.Item)

frameTime = 5f; // Время (в тиках) для смены кадров анимации. Используется, если frames > 0. (Требуется: доступен напрямую в mindustry.type.Item)

// Специфические механики

heatColor = Color.valueOf("ff0000"); // Цвет свечения при нагреве (например, для расплавленных материалов). По умолчанию красный. (Требуется: НЕТ в Item v146; возможно, доступно в более новых версиях или через кастомные моды; проверьте UnlockableContent или обновите API)

coolant = false; // Если true, предмет может использоваться как охладитель (например, вода). По умолчанию false. (Требуется: НЕТ в Item v146; относится к mindustry.type.Liquid)

coolantMultiplier = 1f; // Множитель эффективности охлаждения (если coolant = true). 1f — стандартное значение. (Требуется: НЕТ в Item v146; относится к mindustry.type.Liquid)

liquidDrop = null; // Ассоциированная жидкость, которая "выпадает" из предмета (например, нефть из предмета). По умолчанию null. (Требуется: НЕТ в Item v146; относится к mindustry.world.blocks.environment.OreBlock)

gasColor = Color.clear; // Цвет газа, если предмет превращается в газ (например, при испарении). По умолчанию прозрачный. (Требуется: НЕТ в Item v146; нестандартное свойство, возможно, из кастомных модов)

// Взаимодействие с игровыми механиками

unlockCost = 0; // Стоимость исследования предмета в дереве технологий (в очках исследования). 0 — бесплатно. (Требуется: НЕТ в Item v146; доступно через mindustry.type.UnlockableContent)

researchMultipliers = new ObjectMap<>(); // Множители для стоимости исследования (для кастомизации дерева технологий). По умолчанию пустой. (Требуется: НЕТ в Item v146; доступно через mindustry.type.UnlockableContent; import mindustry.type.ObjectMap)

tags = new String[]{"example"}; // Теги предмета для фильтрации или моддинга (например, "ore" для руд). По умолчанию пустой массив. (Требуется: НЕТ в Item v146; доступно через mindustry.type.UnlockableContent)

Скажу сразу, запоминать их все не стоит, вы можете просто подглядывать сюда либо в моем GitHub будет файл где вы можете найти все значения для предметов блоков и так далее, так же вы можете вернуться в листинг 1.2 и посмотреть какие параметры я поставил. Так же некоторые параметры могут не работать и компилятор их не сможет найти для этого нужно импортировать другие классы, например чтобы воспользоваться параметром coolant нужно импортировать класс mindustry.type.Liquid. Во общем могу ошибаться с дополнительными классами если таких классов у вас нету прощу прощения, но если вы не собираетесь делать очень масштабный мод на 2 и более тысячи строк вам оно и не понадобится, надеюсь…

Осталось добавить текстуру вашему предмету, это делается легко. Перейдите по пути assets/sprites тут вы можете вставить свое изображение, учтите два важных пункта:

1. Изображние должно называться так же как вы назвали свой предмет „new Item("iron", Color.darkGray)“ Как вы видите в первом агрументе конструктора я указал „iron“ значит файл с текстурой должен называться точно так же, учтите если вы добавили frames < 0, игра ожидает от вашего предмета анимации, в этом случае нужно вставить столько изображений сколько вы указали фреймов и правильно их назвать например: iron1.png, iron2.png, iron3.png, просто нужно указать цифру после имени.

2. Сразу вам скажу одна клетка в игре занимает 32x32 пикселя, следовательно нужно рисовать текстуры в таком расширении, например если ваш объект занимает одну клетку следует рисовать 32x32, если же объект занимает в пространстве 2x2 то следует рисовать 64x64, формула тут такая: размер объекта \* 32.

Все готово осталось вызвать метод „Load()“ в главном файле нашего мода(ExampleJavaMod.java), вам следует подключить ваш класс предмета в моем случае это ItemsRegistration

Обновленный листинг 1.0 (ExampleJavaMod.java)

package example;

import arc.\*;

import arc.util.\*;

import mindustry.\*;

import mindustry.mod.\*;

import example.items.ItemsRegistration;

public class ExampleJavaMod extends Mod{

    @Override

    public void loadContent(){

        ItemsRegistration.Load();

    }

}

важный момент, если у вас есть кастомные блоки, следует с начало вызывать „ItemsRegistration.Load()“, а потом вызывать последующие элементы, все звязанно с тем что код выполняется поочередно и может не увидеть объекты которые еще не определенны. Теперь при каждом новом файле нужно будет вызывать его метод „Load()“ чтобы изменения приступили в силу.

3 Глава — создание блока.

Спросите почему я выделил создание блока в отдельную главу? Потому что нам есть о чем поговорит здесь и много о чем. Дело в том что классов для блока очень много, но я расскажу о самых базовых и основных.

Основные параметры для всех блоков:

name // Уникальное имя блока (например, "iron-turret"), задается в конструкторе

size = 1 // Размер блока в клетках (например, 1x1, 2x2); по умолчанию 1

health = -1 // Количество здоровья блока; -1 означает, что используется значение по умолчанию (зависит от размера)

baseExplosivity = 0f // Базовая взрывоопасность блока (влияет на взрыв при разрушении)

itemCapacity = 10 // Сколько предметов может хранить блок (если hasItems = true)

liquidCapacity = 10f // Сколько жидкости может хранить блок (если hasLiquids = true)

powerCapacity = 10f // Сколько энергии может хранить блок (если hasPower = true)

powerProduction = 0f // Сколько энергии производит блок за тик (если producesPower = true)

update = false // Обновляется ли блок каждый тик (например, для анимации или логики)

solid = true // Является ли блок твердым (препятствует движению юнитов)

destructible = false // Может ли блок быть разрушен

breakable = false // Может ли блок быть сломан игроком

rebuildable = true // Может ли блок быть восстановлен после разрушения

placeablePlayer = true // Может ли игрок размещать этот блок

placeableLiquid = false // Можно ли размещать блок на жидкостях

rotate = false // Может ли блок быть повернут при размещении

quickRotate = true // Быстрое вращение при размещении (если rotate = true)

hasItems = false // Может ли блок хранить предметы

hasLiquids = false // Может ли блок хранить жидкости

hasPower = false // Требуется ли блоку энергия

outputsPower = false // Производит ли блок энергию

consumesPower = true // Потребляет ли блок энергию

conductivePower = false // Проводит ли блок энергию (например, как силовые узлы)

hasShadow = true // Отбрасывает ли блок тень

buildVisibility = BuildVisibility.shown // Видимость в меню строительства: shown (виден), hidden (скрыт), debugOnly (только в дебаге), editorOnly (только в редакторе), sandboxOnly (только в песочнице)

category = Category.effect // Категория блока в меню (например, turret, production, defense)

buildCostMultiplier = 1f // Множитель стоимости строительства (1f = стандартная стоимость)

buildTimeMultiplier = 1f // Множитель времени строительства (1f = стандартное время)

buildCost = 0f // Базовая стоимость строительства (обычно переопределяется через requirements)

requirements = new ItemStack[0] // Требования для строительства (массив ItemStack, например, ItemStack.with(Items.copper, 100))

alwaysUnlocked = false // Разблокирован ли блок по умолчанию

group = BlockGroup.none // Группа блока (например, walls, turrets, для логики размещения)

ambientSound = Sounds.none // Фоновый звук блока

ambientSoundVolume = 0f // Громкость фонового звука

sync = false // Синхронизируется ли блок между клиентом и сервером

timers = 0 // Количество таймеров, используемых блоком

timerDump = null // Таймер для сброса предметов/жидкостей (если null, не используется)

configurable = false // Можно ли настроить блок (например, через интерфейс)

saveConfig = false // Сохраняется ли конфигурация блока

copyConfig = true // Копируется ли конфигурация при размещении

clearOnDoubleTap = false // Очищается ли конфигурация при двойном нажатии

drawLiquidLight = true // Отображает ли блок свет от жидкости (если hasLiquids = true)

underBullets = false // Находится ли блок "под пулями" (например, конвейеры)

forceDark = false // Принудительно темный фон для блока

clipSize = -1f // Размер области отсечения для отрисовки (-1 = авто)

fogRadiusMultiplier = 1f // Множитель радиуса видимости в тумане войны

placeOverlapRange = 0f // Радиус, в котором блок перекрывает другие при размещении

placeOverlapMargin = 0f // Дополнительный отступ для перекрытия

disableOverlapCheck = false // Отключает ли проверку перекрытия при размещении

3.1 Классы для блоков:

turret: Боевые блоки, атакующие врагов.

production: Блоки для добычи или производства ресурсов (дрели, насосы).

distribution: Блоки для транспортировки предметов (конвейеры, мосты).

liquid: Блоки для работы с жидкостями (трубы, резервуары).

power: Блоки для генерации и передачи энергии (генераторы, узлы).

defense: Защитные блоки, поглощающие урон (стены, щиты).

craft: Блоки для крафта сложных предметов (фабрики, заводы).

effect: Вспомогательные блоки с эффектами (ускорители, ремонт).

logic: Блоки для логики и автоматизации (процессоры, переключатели).

unit: Блоки для производства юнитов (заводы юнитов, сборщики).

3.2 Создание турели

создадим папку turret в подпапке blocks(src/example/blocks/turret) там создадим файл TurretsRegistration.java

далее нам нужно импортировать „mindustry.world.blocks.defense.turrets.Turret“ для работы с турелями, „mindustry.world.blocks“ отвечает за все блоки, если не хотите постаянно указывать все по отдельности просто укажите „mindustry.world.blocks.\*“.

Далее создайте объект типа „Block“ и после создайте метод „Load()“, впрочем ничего нового.

3.3 Основные параметры для турели

logicControlCooldown = 60 \* 2 // Время (в тиках, 60 \* 2 = 120 тиков или 2 секунды) после управления через логику, после которого турель возобновляет нормальную работу ИИ

timerTarget = timers++ // Идентификатор таймера для проверки цели, увеличивается автоматически (timers++)

targetInterval = 20 // Интервал (в тиках, по умолчанию 20) между попытками найти цель

newTargetInterval = -1f // Интервал (в тиках) для поиска новой цели, если уже есть текущая; -1 означает использование targetInterval

maxAmmo = 30 // Максимальное количество единиц боеприпасов, которые турель может хранить (по умолчанию 30)

ammoPerShot = 1 // Количество единиц боеприпасов, расходуемых за один выстрел (по умолчанию 1)

consumeAmmoOnce = true // Если true, боеприпасы расходуются только один раз за выстрел, независимо от количества пуль (по умолчанию true)

heatRequirement = -1f // Минимальная температура (если используется нагрев), необходимая для выстрела; -1 означает, что нагрев не требуется (по умолчанию -1)

maxHeatEfficiency = 3f // Максимальная эффективность турели при использовании тепла (по умолчанию 3f)

inaccuracy = 0f // Случайное отклонение угла выстрела в градусах (по умолчанию 0f)

velocityRnd = 0f // Доля случайной скорости пули (0f–1f, по умолчанию 0f)

scaleLifetimeOffset = 0f // Доля времени жизни пули, добавляемая к пулям с масштабированием жизни (по умолчанию 0f)

shootCone = 8f // Максимальный угол отклонения (в градусах), при котором турель пытается стрелять (по умолчанию 8f)

shootX = 0f, shootY = Float.NEGATIVE\_INFINITY // Координаты точки выстрела относительно центра турели (по умолчанию 0f и Float.NEGATIVE\_INFINITY)

xRand = 0f // Случайное смещение выстрела по оси X (по умолчанию 0f)

drawMinRange // Если true, рисуется кольцо минимального радиуса действия (по умолчанию false)

trackingRange = 0f // Радиус (в клетках), в котором турель обнаруживает и отслеживает цель, но не стреляет (по умолчанию 0f)

minRange = 0f // Минимальный радиус действия для стрельбы (используется для артиллерии, по умолчанию 0f)

minWarmup = 0f // Минимальный уровень прогрева, необходимый для выстрела (по умолчанию 0f)

accurateDelay = true // Если true, турель точно учитывает движение цели с учетом задержки первого выстрела (по умолчанию true)

moveWhileCharging = true // Если false, турель не может двигаться во время зарядки (по умолчанию true)

reloadWhileCharging = true // Если false, турель не может перезаряжаться во время зарядки (по умолчанию true)

warmupMaintainTime = 0f // Время (в тиках), в течение которого прогрев сохраняется после прекращения стрельбы (по умолчанию 0f)

shoot = new ShootPattern() // Объект паттерна стрельбы, определяющий, как и куда выпускаются пули (по умолчанию новый ShootPattern)

targetAir = true // Если true, турель атакует воздушные цели (по умолчанию true)

targetGround = true // Если true, турель атакует наземные цели и строения (по умолчанию true)

targetBlocks = true // Если true, турель атакует вражеские блоки (по умолчанию true)

targetHealing = false // Если true, турель лечит союзные блоки (по умолчанию false)

playerControllable = true // Если true, турель может управляться игроками (по умолчанию true)

displayAmmoMultiplier = true // Если true, в статистике отображаются множители боеприпасов (по умолчанию true)

targetUnderBlocks = true // Если false, 'under' блоки (например, конвейеры) не атакуются (по умолчанию true)

alwaysShooting = false // Если true, турель стреляет при наличии боеприпасов, независимо от целей (по умолчанию false)

predictTarget = true // Если true, турель предсказывает движение целей (по умолчанию true)

unitermore = UnitSorts.closest // Функция сортировки для выбора цели (по умолчанию UnitSorts.closest — ближайшая цель)

unitFilter = u -> true // Фильтр для выбора типов юнитов для атаки (по умолчанию принимает всех юнитов)

buildingFilter = b -> targetUnderBlocks || !b.block.underBullets // Фильтр для выбора типов зданий для атаки (по умолчанию включает все здания, кроме под конвейерами, если targetUnderBlocks false)

heatColor = Pal.turretHeat // Цвет теплового эффекта, отображаемого на турели (по умолчанию Pal.turretHeat)

shootEffect // Эффект выстрела (по умолчанию null, использует стандартный эффект)

smokeEffect // Эффект дыма от выстрела (по умолчанию null, использует стандартный эффект)

ammoUseEffect = Fx.none // Эффект расхода боеприпасов (по умолчанию Fx.none)

shootSound = Sounds.shoot // Звук выстрела одной пули (по умолчанию Sounds.shoot)

chargeSound = Sounds.none // Звук начала зарядки (по умолчанию Sounds.none)

loopSound = Sounds.none // Циклический звук работы турели (по умолчанию Sounds.none)

loopSoundVolume = 0.5f // Громкость циклического звука (по умолчанию 0.5f)

soundPitchMin = 0.9f, soundPitchMax = 1.1f // Диапазон высоты звука выстрела (по умолчанию 0.9f–1.1f)

ammoEjectBack = 1f // Смещение назад эффекта выброса боеприпасов (по умолчанию 1f)

shootWarmupSpeed = 0.1f // Скорость прогрева турели (по умолчанию 0.1f)

linearWarmup = false // Если true, прогрев турели линейный, а не криволинейный (по умолчанию false)

recoil = 1f // Визуальный откат турели при выстреле (по умолчанию 1f)

recoils = -1 // Количество дополнительных счетчиков для отката (по умолчанию -1)

recoilTime = -1f // Время (в тиках) возвращения турели в исходное положение; -1 использует reloadTime (по умолчанию -1f)

recoilPow = 1.8f // Степень кривой для визуального отката (по умолчанию 1.8f)

cooldownTime = 20f // Время (в тиках) для охлаждения теплового региона (по умолчанию 20f)

elevation = -1f // Визуальная высота тени турели; -1 использует стандартное значение (по умолчанию -1f)

shake = 0f // Сила тряски экрана при выстреле (по умолчанию 0f)

1. . Git — распределённая система управления версиями. [↑](#footnote-ref-2)