Libgdx中文指南

Huang YunKun

目錄

Libgdx 中文指南

本文档大部分内容源于官方Wiki，其他内容源于相关博客资料。

贡献者：

- AyoCrazy http://www.ayogame.cn/
- 纯洁的坏蛋 http://www.mwplay.net/

阅读和下载

目前本书托管在gitbook.io上，所有更改会自动同步。gitbook的访问有时候比较慢，后期如果有需要我会另外准备一个服务器。

- 在线阅读 https://www.gitbook.com/read/book/htynkn/libgdx
- PDF下载 https://www.gitbook.com/download/pdf/book/htynkn/libgdx
- MOBI下载 https://www.gitbook.com/download/mobi/book/htynkn/libgdx
- EPUB下载 https://www.gitbook.com/download/epub/book/htynkn/libgdx

贡献

如果你对该项目有兴趣，欢迎你为本项目贡献。主要有两方面的工作：

- 翻译
- 纠错

如果你选择翻译，请先在issue中提出你要翻译的文件，翻译完成后提出pull request请求即可。

如果你选择纠错，那么直接在issue中提出即可。

本地预览

出于各种原因，你需要在本地就行预览。

请参考gitbook工具文档，安装并启动本地服务器。

```
npm install -g gitbook
gitbook serve libgdx-book
```

构建检查

Passed

Libgdx是一个跨平台的游戏开发框架。目前Libgx支持的平台包括Windows, Linux, Mac OS X, Android, iOS 和 HTML5。

Libgdx可以帮助你在不需要其他额外操作的情况下将同一套代码部署到不同的平台上。你不在需要等待你最新的修改部署到测试设备或者编译成HTML5，你可以 轻松享受桌面环境开发的快捷与便利。同时你可以使用Java生态圈的所有工具来帮助你提升开发效率。

Libgdx可以让你近距离的控制你的代码，你可以直接访问文件系统，输入设备，音频设备，还可以通过OpenGL ES 2.0 和 3.0的接口操作OpenGL。

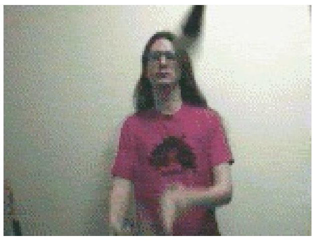
在低层次的抽象之上，Libgdx还提供了开发游戏所常用的封装。比如绘制纹理，构建用户界面，播放背景音效，线性代数和三角计算，解析Json和Xml等等。

在必要时，Libgdx会使用本地代码来提升效率，你不需要担心这些本地代码的兼容性问题。Libgdx是一个使用广泛的开源项目，在成千上万的使用者的帮助下，Libgdx的跨平台性能稳定而高效。

Libgdx致力于成为一款出色的框架而不是引擎，因为很难有一个万能通用的解决方案。相反，Libgdx提供了便利的抽象和封装让你自由选择如何去组织你的游戏或者应用。

站在巨人肩上

Libgdx使用了很多成熟的第三方库来实现相关功能：

| 

| |

|

LWJGL

OpenAL

KissFFT

Libgdx是一款Java游戏开发框架，它提供了统一的API，能够在所有的它所支持的平台上（Windows, Linux, Mac OS X, Android, BlackBerry, iOS, HTML/WebGL without applets）工作。

这套框架提供了一种可以快速开发和迭代的环境。不需要在每次修改代码后将修改部署到Android或者iOS上，你可以直接在桌面环境开发和调试你的项目。桌面版JVM的一些特写，比如热部署，可以极大加快你的开发。

Libgdx并没有希望成为一个万能解决方案。 它不强求你使用任何一种特定的设计模式。 你可以选择你需要的部分构建你自己的游戏或者应用，甚至构建最适合你自己的游戏开发引擎。

特性

Libgdx是一个快速发展的项目，它的所有支持的功能和特性你都可以从以下网址找到：
http://libgdx.badlogicgames.com/features.html

Libgdx是一个由世界各地的开发者参与的开源项目。这样的团队成员不能保证通过邮件或者其他及时通讯手段给予你一对一的帮助和支持。相反，我们希望你关注论坛和IRC频道以获得帮助，同时以帮助他人。

论坛地址: http://badlogicgames.com/forum

*IRC*频道: irc.freenode.net, #libgdx

请不要直接给我们发送邮件。我们会经常出现在IRC频道，但是我们不能保证及时回复你。最好的获取帮助的渠道就是论坛。当然Google也是你的朋友。

想了解最新的开发情况，请访问blog 或者关注 Twitter。

为Libgdx项目做出贡献的渠道有很多：

- 在Github上Fork项目
- 了解如何运行demo和测试
- 编写代码并在Github上申请Pull Request

API变动

如果你修改的代码涉及了公开的API或者你增加了新的API，你确保你添加相关信息到CHANGES文件中。 除了这个CHANGES文件，相关的变动也会发布在blog和Twitter上，以便整个社区知晓。

如果你希望和其他开发人员交流，你可以选择申请Pull Request来分享你的代码。也可以在论坛中this sub-forum发起一个新的话题。在论坛中你可能需要特别的授权才能在这个版块发帖，请给contact@badlogicgames.com发送一封邮件并附上你的论坛ID。在论坛页面的底部有一个订阅按钮，你最好通过邮件订阅该板块的信息。当然你也可以访问IRC (irc.freenode.org, #libgdx)，这里常常有核心开发者在线。

贡献者许可协议

Libgdx基于Apache 2.0 license协议发行。在我们接受你的代码贡献之前，你需要签署贡献者协议。将它打印出来，然后填写相关信息并发送到邮箱即可。

签署这份协议将允许我们使用并分发你的代码。这是一个非独占许可，所以你保留所有你的代码的权利。这是一个安全防范，特别是某人贡献了重要的代码，随后又希望将其收回。

Eclipse格式

如果你在编写libgdx源代码，我们要求你使用Eclipse formatter。

Eclipse formatter可以保证所有开发者使用的常用格式设置的一致性。

如果你是用IntelliJ IDEA，请参考这篇文章：this article

代码风格

Libgdx没有官方的编码规范。我们遵循常见的Java风格，同时我们也希望你也遵守。

我们不希望看到以下情况：

- 在任何类型的标识符的下划线
- 匈牙利标记法
- 带前缀的字段或者参数
- 大括号换行

如果你修改了任何以后的代码，请遵循这些代码风格。如果不影响可读性，部分大括号将被移除。

如果你添加了新文件，确保它们包含Apache文件头部说明。

如果你创建了一个新的类，请至少添加文档来说明类的使用方法和范围。当然，你可以省略一些简单的、不言自明的方法。

如果你的类是显式线程安全的，请在Javadoc中提及这点。默认的假设是类不是线程安全的，锁的使用时昂贵的，我们希望减少它们在代码库的数量。

跨平台兼容性

GWT的实现不能支持)所有的java特性。在编写代码的时候，请注意一些常见的局限性：

- 格式化。 String.format()是不支持的，使用StringBuilder或者直接拼接字符串。
- 正则表达式。 一个有限功能的Pattern和[Matcher(https://github .com/libgdx/libgdx/blob/master/backends/gdx-backends-gwt/src/com/badlogic/gdx/backends/gwt/emu/java/util/regex/Matcher.java)是可用的。
- 反射. 使用Libgdx提供的反射工具com.badlogic.gdx.utils.reflect。
- 多线程. 只支持Timers。

如果你添加了新的类，请判断它是否与GWT兼容，并添加信息到GWT module中。

一些类（比如Matrix4或者BufferUtils）为了保证GWT的兼容性在本地代码中做了特殊处理。如果你修改了这些类，请确保你的修改同样在这些地方体现。

性能方面的考虑

Libgdx可以运行在桌面和移动环境，包括浏览器（Javascript）。但是需要注意的是桌面环境的性能远远超过其他环境。桌面环境的HotSpot VM可以提供一些不必要的资源来运行程序，但是Dalvik等其他环境不能。

一些常见指导准则:

- 尽可能避免临时对象分配
- 不要拷贝副本
- 避免锁，Libgdx中的类都是默认非线程安全的，除非明确说明
- 使用Libgdx提供的集合类com.badlogic.gdx.utils package
- 不要进行参数检查，有时候这样会导致大量的调用
- 尽可能使用池，如果可能也不要暴露池给开发人员， 它们是复杂的API。

Git

Libgdx的大部分成员是Git新手。为了避免一些问题，请保持你的Pull Request尽可能小。如果一个提交修改了超过过多文件，很可能它们不会被合并。

对于较大的API修改，我们会新建一个分支。如果你的提交和它们相关，请确保你的Pull Request指定了正确的分支。

提交到master分支的Pull Request会被多个核心开发人员检查。如果我们认为它们还不适合合并，我们将会拒绝你的合并请求。 希望你能够理解这种情况。Libgdx被大量项目使用，我们必须确保它健壮而可用。

基于Libgdx构建游戏有很多，在官方主页上你还能看到专门的版块来展示它们。

如果你的游戏或者应用基于Libgdx构建，欢迎你提交它们到展示专栏中。

Libgdx使用 Gradle来管理依赖，编译，构建和IDE集成。

这样你就可以在不同的开发环境开发你的游戏或者应用了。整个项目组中的成员可以任意选择自己喜欢的开发环境。不要将IDE相关的文件纳入版本控制中。 Git中的 `.gitignore` 可以帮助你排除不需要纳入版本控制的文件。如果你并不熟悉如何编写这个文件，你可以在github找到不同类型的项目常用的 `.gitignore` 例子。

不同的开发环境需要的基础环境稍有区别，这里我们简单提及，之后的章节会详细涉及怎么在不同的开发环境中开发Libgdx。

Eclipse环境

使用Eclipse开发Libgdx项目，你需要以下软件和工具包：

- Java Development Kit 7+ (JDK) (6 will not work!)
- Eclipse, "Eclipse IDE for Java Developers" 是较为合适的选择。
- Android SDK, 你只需要下载SDK，不需要下载ADT。通过SDK Manager来下载对应的目标平台。
- Android Development Tools for Eclipse, 插件的安装升级地址: https://dl-ssl.google.com/android/eclipse/
- Eclipse Integration Gradle, 插件的安装升级地址e: http://dist.springsource.com/snapshot/TOOLS/gradle/nightly (for Eclipse 4.4) or http://dist.springsource.com/release/TOOLS/gradle。该插件要求Eclipse版本在1.4以上。

如果你的目标平台包含iOS

- 一个Mac，iOS的开发不能在Windows/Linux进行，谢谢苹果。
- 最新的XCode，你可以从Mac OS X App Store免费获取。
- RoboVM, 安装它的Eclipse插件即可。

我个人既不喜欢Mac，如果你有其他顺手的笔记本电脑，可以考虑购买一个Mac mini作为编译和少量开发使用。

Intellij IDEA环境

使用Intellij IDEA开发Libgdx项目，你需要以下软件和工具包：

- Java Development Kit 7+ (JDK) (6 will not work!)
- Intellij IDEA 14.+, 社区版本即可。
- Android SDK, 你只需要下载SDK，不需要下载ADT。通过SDK Manager来下载对应的目标平台。

对于iOS的环境需求，和Eclipse一样。

NetBeans环境

使用NetBeans开发Libgdx项目，你需要以下软件和工具包：

- Java Development Kit 7+ (JDK) (6 will not work!)
- NetBeans 7.3+, "Java SE"版本即可
- Android SDK, 你只需要下载SDK，不需要下载ADT。通过SDK Manager来下载对应的目标平台。
- NBAndroid, 安装升级地址: http://nbandroid.org/updates/updates.xml
- Gradle Support for NetBeans, 使用NetBeans IDE升级中心。

对于iOS的环境需求，和Eclipse一样。

一旦你的环境准备就绪，你可以开始下一章。

新建**libgdx**项目

Libgdx提供了一个初始化工具 `gdx-setup.jar` ，包含了一个可执行的UI和命令行模式。直接执行该文件你就可以看到该工具界面。

如果你希望从命令行启动，使用命令: `java -jar gdx-setup.jar`

下载gdx-setup.jar

指定你的应用名称，Java包名以及启动类名称，输出目录和Android SDK目录。

然后选择你希望支持的平台。 提示: 一旦项目生成，你只能手动自行添加新的平台支持**!**。

Libgdx提供了一些扩展，有部分扩展不能兼容所有平台，对于这些扩展，一旦你选择以后软件会给出你提示。

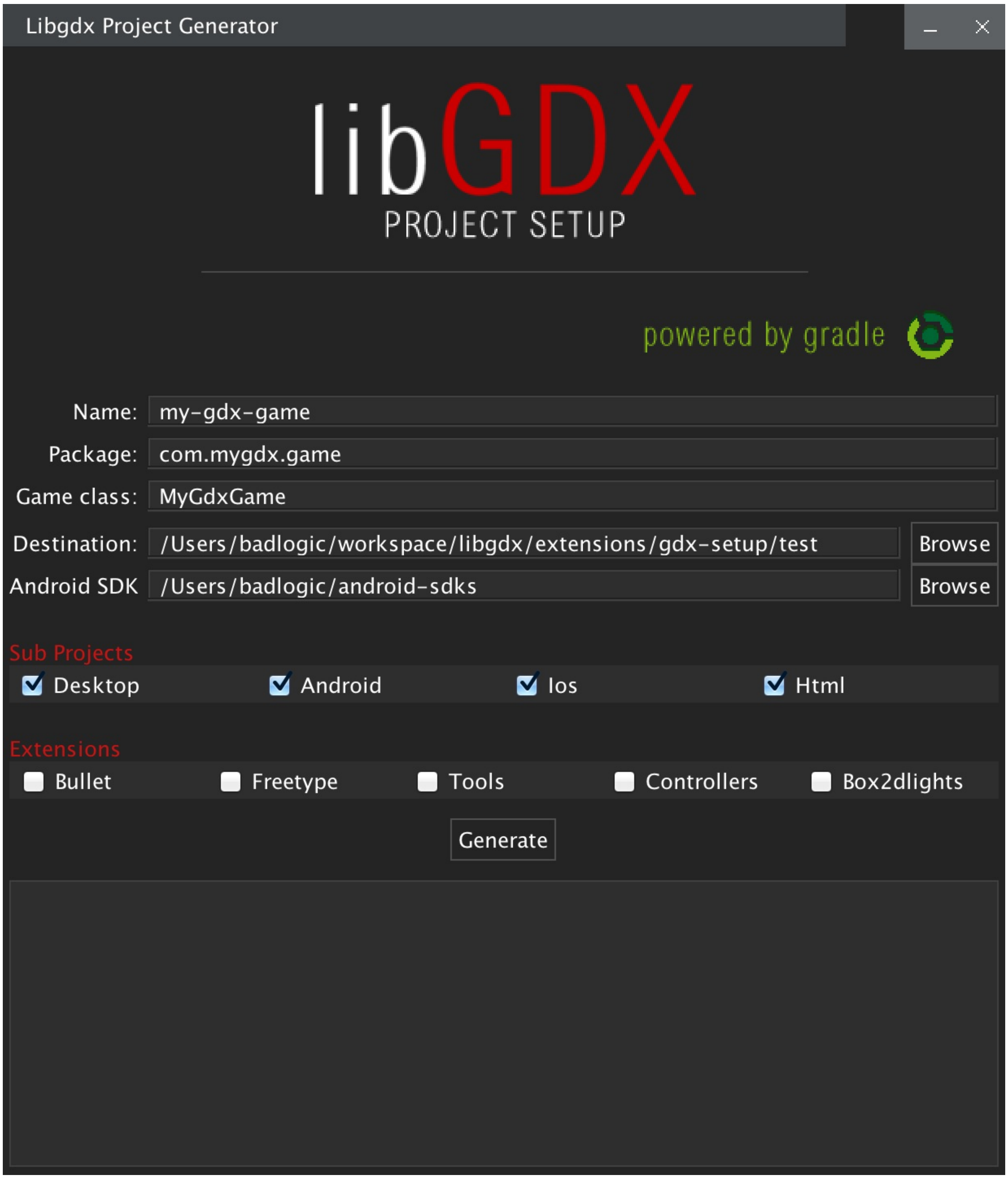
一切完成后点击"Generate"。

项目文件生成后，你就可以导入到**IDE**并运行和调试了。

- Eclipse
- Intellij-Idea
- NetBeans
- Commandline

点击"Advanced"按钮可以设置是否生成Eclipse或者IDEA项目文件，也可以设置一个依赖仓库的镜像去下载依赖包。

国内访问Maven中央仓库有时候很慢，可以考虑使用oschina提供的镜像服务。


Libgdx Project Generator

Name: my-gdx-game

Package: com.mygdx.game

Game class: MyGdxGame

Destination: /Users/badlogic/workspace/libgdx/extensions/gdx-setup/test

Android SDK /Users/badlogic/android-sdks

Sub Projects

[✓ Desktop] [✓ Android] [✓ Ios] [✓ Html]

Extensions

Bullet Freetype Tools Controllers Box2dlights

使用命令行创建Libgdx项目

如果你不喜欢使用UI界面，或者不具备使用UI界面的条件，你可以使用命令行直接调用创建工具。

命令行参数有以下几个：

- **dir**: 项目地址，绝对路径和相对路径都可以
- **name**: 项目的名字
- **package**: Java包名，比如com.badlogic.mygame
- **mainClass**: 启动类的名称，比如MyGame
- **sdkLocation**: Android SDK地址

提供以上参数即可创建项目，比如:

`java -jar gdx-setup.jar --dir mygame --name mygame --package com.badlogic.mygame --mainClass MyGame --sdkLocation mySdkLocation`

项目结构

创建好的项目结构如下：

```
settings.gradle            <- 定义子模块，比如core, desktop, android, html, ios
build.gradle               <- Gradle的主要配置文件，声明了依赖和插件
gradlew                    <- Gradle在Unix 环境的执行脚本
gradlew.bat                <- Gradle在Windows 环境的执行脚本
gradle                     <- 本地包装器
local.properties           <- Intellij专有配置文件，定义android sdk位置

core/
    build.gradle           <- Gradle针对core的配置文件
    src/                   <- 项目代码，包含游戏的主要代码（平台无关）

desktop/
    build.gradle           <- Gradle针对desktop的配置文件
    src/                   <- 桌面项目的启动器和其他平台相关代码

android/
    build.gradle           <- Gradle针对android的配置文件
    AndroidManifest.xml    <- Android配置
    assets/                <- 游戏所需的所有资源文件，包括音乐，图片等等
    res/                   <- app的图标和其他资源
    src/                   <- Android项目的启动器和平台相关代码

html/
    build.gradle           <- Gradle针对html的配置文件
    src/                   <- Html项目的启动器和平台相关代码
    webapp/                <- War包模板，包含了启动页面和web.xml配置

ios/
    build.gradle           <- Gradle针对iOS的配置文件
    src/                   <- iOS项目的启动器和平台相关代码
```

生成的项目中包含了所有打包本地文件和针对不同的平台的打包分发。请小心修改这些文件，如果你熟悉Gradle和Libgdx项目，那么你可以手动添加或者修改一些任务。

你之前生成了Libgdx项目, 现在该是大胆的用Eclipse开发的时候了!

在导入项目到**Eclipse**之前**,**确保你已经配置好开发环境**!**

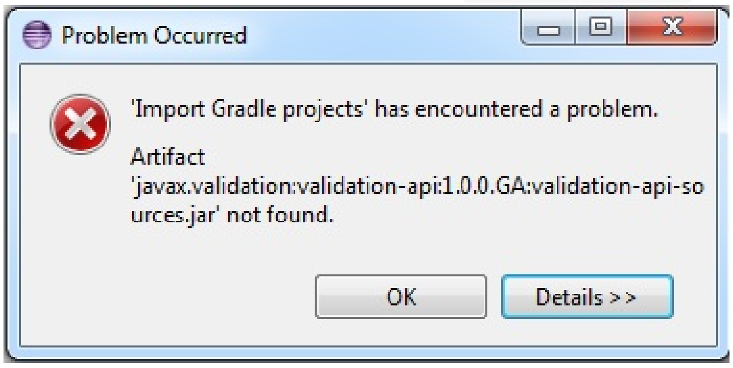
导入项目

到 `File -> Import -> Gradle -> Gradle Project` , 选中你的项目根目录, 然后点击 `Build Model` . 等待一会儿, 你将会看到一个根项目和几个子项目 (android, core, desktop, html, ios). 选中所有的项目,然后点击 `Finish` . 注意第一次操作这个过程可能会花费一两分钟,因为后台会下载Gradle和一些必要的依赖文件.

常见问题

你的项目根文件夹和Eclipse工作区(workspace)文件夹一定不能相同(详情见issue)

假如你在运行的时候由于丢失 validation-api:1.0.0.GA 出现问题, 请删除Maven缓存, 一般在 `C:\Users\username\.m2` or `/Users/username/.m2` or `/home/username/.m2` .


Problem Occurred

'Import Gradle projects' has encountered a problem.

Artifact 'javax.validation:validation-api:1.0.0.GA:validation-api-sources.jar' not found.

当你在第一次导入项目的时候遇到如下问题: `com/github/jtakakura/gradle/plugins/robovm/RoboVMPlugin : Unsupported major.minor version 51.0` 请确保你的java运行环境版本在jdk 7以上

运行项目

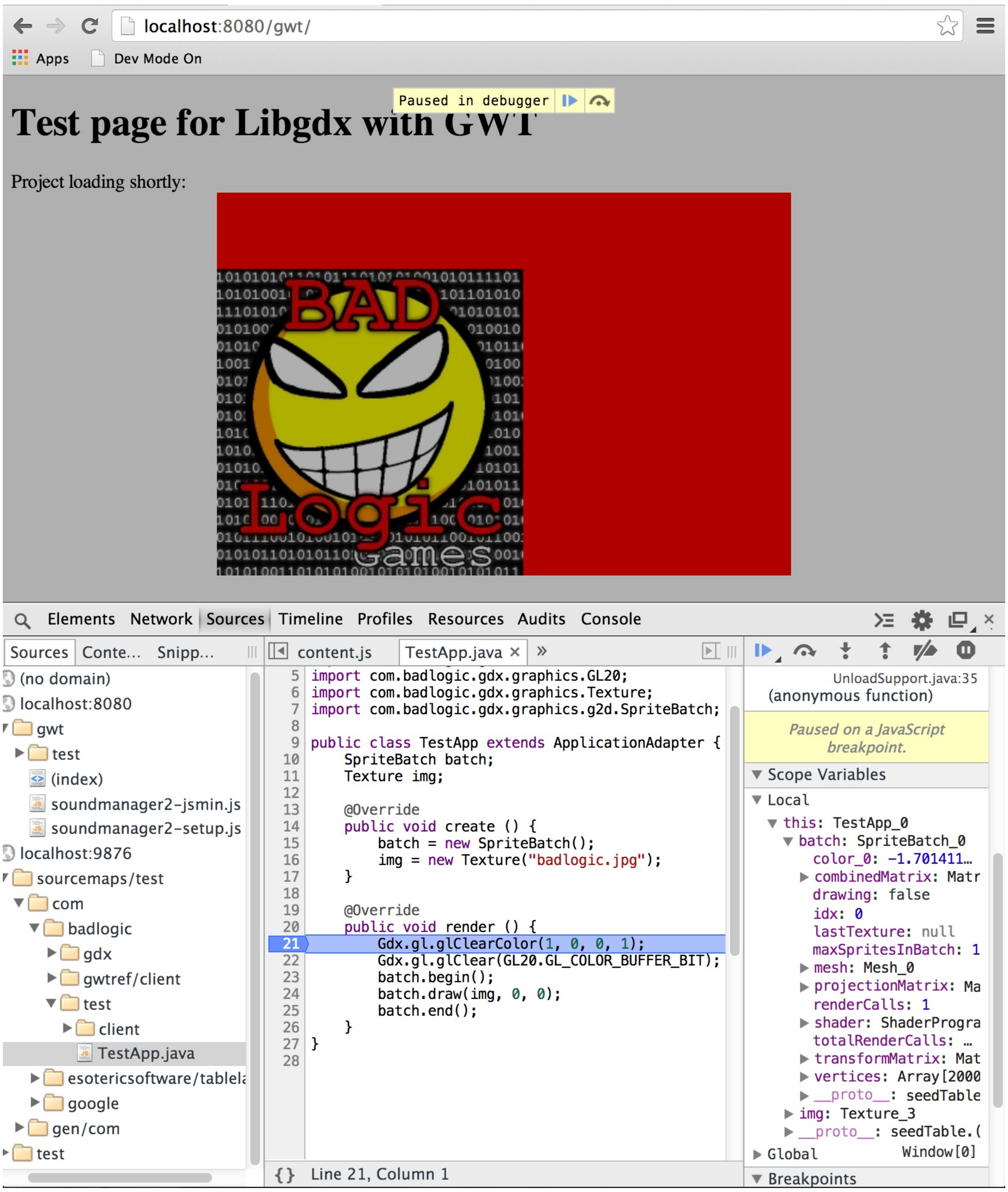
- **Desktop**: 在你的桌面项目上点击鼠标右键, `Run As -> Java Application` . 选中你的桌面启动类 (例如DesktopLauncher.java).
- **Android**: 确保在DDMS里面已经显示了一个连接成功的android设备 (见 Android Developer Guide). 在Android项目上点击右键, `Run As -> Android Application` .
- **iOS RoboVM**: 在robovm项目上点击鼠标右键, `Run As -> iOS Device App` 在连接成功的设备上运行, 或者 `Run As -> iOS Simulator App` 在IOS模拟器上运行. 假如需要在真机上运行, 你需要 provision 它能让你把app部署到真机里面运行!
- **HTML5**: 在html项目上点击鼠标右键, `Run As -> External Tools Configuration` . 在左侧边栏上双击'Program',创建一个新的配置(configuration). 给新配置设置一个名字, 例如 GWT SuperDev. 给'gradlew.vat'(Windows), 或者'gradlew' (Linux, Mac)文件设置一个本地的环境变量. 把working directory设置成项目的根目录. 指定 `html:superDev` 作为参数. 点击 'Apply', 然后点击 'Run'. 等到命令行视图(console view)里面显示'The code server is ready.', 然后打开链接 http://localhost:8080/html. 你可以让服务器一直保持运行. 假如修改了代码或者资源文件, 你只需要在浏览器里面点击按钮'SuperDev Refresh', 然后编译器就会重新编译app,重新载入网站

由于这个 bug in the Gradle tooling API 被修复后, 我们只需要简单运行HTML5通过使用Gradle集成, 这个Gradle进程会一直运行直到手动取消.

调试项目

只需要把正常运行项目步骤中通过'Run as'运行程序改为使用'Debug as'即可进入调试程序的过程. 最新的libgdx版本已经支持调试ios项目 如果想调试html项目,可以在浏览器中按照如下步骤进行调试:

和之前一样运行'superDev'配置项. 在http://localhost:8080/html点击 'SuperDev Refresh'按钮. 在Chrome浏览器中,按'F12' 唤起开发工具箱, 在sources标签中找到你需要调试的java文件. 设置断点, 单步调试 然后使用source maps检查变量。 当改变了代码或者资源文件的时候,点击'SuperDev Refresh'按钮(保持服务器一直运行!).



```
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

public class TestApp extends ApplicationAdapter {
    SpriteBatch batch;
    Texture img;

    @Override
    public void create () {
        batch = new SpriteBatch();
        img = new Texture("badlogic.jpg");
    }

    @Override
    public void render () {
        Gdx.gl.glClearColor(1, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
        batch.begin();
        batch.draw(img, 0, 0);
        batch.end();
    }
}
```

```
UnloadSupport.java:35
(anonymous function)
Paused on a JavaScript
breakpoint.
▼ Scope Variables
▼ Local
  ▼ this: TestApp_0
    ▼ batch: SpriteBatch_0
        color_0: -1.701411…
      ▶ combinedMatrix: Matr
        drawing: false
        idx: 0
        lastTexture: null
        maxSpritesInBatch: 1
      ▶ mesh: Mesh_0
      ▶ projectionMatrix: Ma
        renderCalls: 1
      ▶ shader: ShaderProgra
        totalRenderCalls: …
      ▶ transformMatrix: Mat
      ▶ vertices: Array[2000
      ▶ __proto__: seedTable
    ▶ img: Texture_3
    ▶ __proto__: seedTable.(
▶ Global                Window[0]
▼ Breakpoints
```

程序打包

在命令行里面打包你的程序非常容易, 如果你偏好图形界面， 可以在Eclipse中使用Gradle tasks打包. 如果想查看Gradle tasks的用法, 请在命令行里面输入Gradle -help或者查看Gradle的文档.

你已经生成你的libgdx项目,现在是时候开始使用IntelliJ IDEA了。 你您可以将项目导入到IntelliJ IDEA之前,请确保你设置好了你的开发环境。

导入项目

点击 `Import Project` ,定位到你的项目文件夹,然后选择 `build.gradle` 文件， 点击 `OK` 。 在下一个对话框中,请将所有设置并再次单击 `OK` 。 IntelliJ IDEA现在开始导入你的项目。

第一次导入可能要花一段时间,因为它会下载gradle wrapper和一些依赖。

常见问题

如果您在使用过程中遇到缺少validation-api:1.0.0.GA的问题,删除Maven缓存， 它们通常位于 `C:\Users\username\.m2` 或者 `/home/username/.m2` 。 如果在Mac OS X上出现“Unsupported major.minor version 51.0”错误且这是你使用IntelliJ IDEA的第一个项目,请确保正确设置了JDK。

请参阅帮助页面),或者从欢迎屏幕转到Configure -> Project Defaults -> Project Structure then add your JDK in Platform Settings -> SDKs。

如果你遇到“Error:org.gradle.tooling.GradleConnectionException：Could not execute build using Gradle installation“错误， 检查项目结构(Ctrl+Alt+Shift+S)并添加Java JDK设置。

运行项目

桌面

`Run ->Edit Configurations...` ,单击加号(+)按钮,然后选择 `Application` 。 运行桌面程序之前， 选择使用桌面的模块的classpath,然后选择desktoplauncher类作为 `Main Class` 。 将工作目录设置为 `android/assets/` (或core/assets/)文件夹。 点击 `Apply` ,然后单击OK。 现在,您已经创建了一个运行的桌面项目配置。 现在,您可以选择配置并运行它。

Android

导入项目时会自动创建一个Android项目的配置。 因此,您只需要选择配置,然后运行它即可。

iOS

`Run ->Edit Configurations...` ,单击加号(+)按钮,然后选择 `Gradle` 。 在gradle列表中选择launchiphonesimulator任务。你还可以选择launchIPadSimulator或者launchIOSDevicefor任务。 点击 `Apply` ,然后单击OK。 现在,你已经创建了一个运行iOS项目的配置。

第一次运行会花费较长的时间，因为robovm会编译整个JDK。 随后的运行编译速度将要快得多!

HTML

选择 `View -> Tool Window -> Terminal` ， 确保此时你位于项目根目录。 执行 `gradlew.bat html:superDev` (Windows) 或者 `./gradlew html:superDev` (Linux, Mac OS X)。

这将需要一些时间,因为你的Java代码被编译为JavaScript。 一旦你看到了代码服务器已准备就绪的消息,启动你的浏览器并转到http://localhost:8080/html。 就可以看到你的应用程序在浏览器中运行。

当你更改任何Java代码或assert中的资源文件时时,只需单击“refresh”按钮， 服务将重新编译你的代码并重新加载该页面。 终止该进程,只需简单地在终端窗口中按下Ctrl+C。

提示，端口9876会被GWT开发过程中会被占用。 此外,以任何其他方式尝试使用端口8080访问游戏都不会自动启动你的项目。

该进程将一直运行gradle直到被取消。

调试项目

请按照以下步骤操作,运行该项目,唯一区别在于不要通过运行按钮与西宁，而是启动调试按钮。 请注意,RoboVM当前不支持调试。 调试的HTML项目可以直接在浏览器操作，参考以下步骤：

- 运行superdev gradle任务。
- 跳转到http://localhost:8080/html，点击刷新按钮。
- 在Chrome浏览器中,按下F12开启开发人员工具,请转至“Source”选项卡,然后找到你要调试的Java文件。
- 你可以设置断点、单步跟进和检查变量。

打包项目

最简便的方法是从命令行,或使用IntelliJ IDEA内的gradle任务打包应用程序。 相关细节查看gradle有关任务或者看到gradle命令行的文档。

You just [[generated your libgdx project|Project Setup Gradle]], now it's time to start developing its guts in NetBeans! Before you can import your project into NetBeans, make sure you [[setup your development environment|Setting up your Development Environment (Eclipse, Intellij IDEA, NetBeans)]]!

Importing Your Project

Go to `File -> Open Project...`, navigate to the root folder of your project, then click `Open Project`. NetBeans will now import your project. This can take a while on the first time, as it downloads the Gradle wrapper and some dependencies. In the project view, expand the project, then right-click the `Subprojects` node and select `Open Subprojects`.

Common Problems

If you run into problems due to a missing validation-api:1.0.0.GA artifact, delete your Maven cache at `C:\Users\username\.m2` or `/Users/username/.m2` or `/home/username/.m2`.

Running Your Project

In the project view:

- **Desktop**: Right click the desktop project, `Run`. You can run it.
- **Android**: make sure you have a device connected. Right click your Android project, `Tasks -> installDebug`.
- **iOS RoboVM**: Right click the robovm project, `Tasks -> launchIPhoneSimulator` (alternatives are `launchIPadSimulator` and `launchIOSDevice` for provisioned devices) The first run will take a bit longer as RoboVM has to compile the entire JDK for iOS. Subsequent runs will compile considerably faster!
- **HTML5**: Right click the html project, `Tasks -> superDev`. You can now follow the build process in the console. This will take a while, as your Java code is compiled to Javascript. Once you see the message `The code server is ready`, fire up your browser and go to http://localhost:8080/gwt. This is your app running in the browser! When you change any of your Java code or assets, just click the `SuperDev Refresh` button while you are on the site and the server will recompile your code and reload the page!

Debugging Your Project

Follow the steps for running the project, but instead of launching via `Run`, launch your configuration via `Debug`. Note that RoboVM currently does not support debugging. Debugging of the html build can be done in the browser as follows:

Run the superDev Gradle task as before. Go to http://localhost:8080/gwt, click on the `SuperDev Refresh` button and hit `Compile`. In Chrome, press `F12` to bring up the developer tools, go to the sources tab and find the Java file you want to debug. Set breakpoints, step and inspect variables using the power of source maps!

[[images/Screen%20Shot%202014-03-23%20at%2019.11.27-BkaIpjttPQ.png]]

Packaging Your Project

It's easiest to package your application from the command line, or using the Gradle tasks from within NetBeans. To see the relevant Gradle tasks, check the [[Gradle command line documentation|Gradle on the Commandline]].

这一章节将向你展示如何在命令行中运行你的程序，并且打包到不同的平台。

配置ANDROID_HOME

在你做任何命令行操作之前，ANDROID_HOME环境变量必须指向一个有效的Android SDK。

Windows: `set ANDROID_HOME=C:/Path/To/Your/Android/Sdk`

Linux, Mac OS X: `export ANDROID_HOME=/Path/To/Your/Android/Sdk`

或者你也可以创建一个命名为"local.properties"的文件，内容如下： `sdk.dir /Path/To/Your/Android/Sdk`

运行项目

Gradle让你很方便地在命令行中运行项目。只需要使用gradlew命令行指定你的目标平台，以及使用该平台的运行命令。

运行desktop项目

`gradlew desktop:run`

这条命令编译你的core项目和desktop项目，并且运行desktop启动器。工作路径是android项目的assets文件夹。

如果你运行时遇到了找不到文件的错误，请确保项目路径是正确的。

运行Android项目

`gradlew android:installDebug android:run`

这条命令将为你的应用创建一个debug APK(调试安装包)，把它安装在第一次连接的虚拟机或者真机，并且启动main activity。 进程被分到两个任务中，因为Android Gradle插件可以让你为你的应用创建多个flavors。

你可以在Android Gradle Plugin site找到更多信息。

运行 iOS 项目

`gradlew ios:launchIPhoneSimulator`

`gradlew ios:launchIPadSimulator`

`gradlew ios:launchIOSDevice`

前两条命令将在iPhone或ipad模拟器中启动你的程序，最后一条命令将在一个已连接的真机上运行你的iOS项目，如果已经配置好的话。请参阅Apple的文档关于如何配置真机。请注意，当你第一次运行iOS项目，编译需要很长的时间。编译时间将会在以后的运行中显著降低！

运行HTML项目

`gradlew html:superDev`

这条命令将以GWT Super Dev模式启动你的应用，此模式将你的Java代码编译成Javascript，并允许你在你的浏览器中直接调试Java代码。 如果你在命令行窗口看到这条信息 `Next, visit: http://localhost:9876` ，打开你的浏览器并跳转到这个地址。拖拽"Dev Mode On"标签到你浏览器的书签栏。 接下来，打开http://localhost:8080/html。这里就是你运行在浏览器上的应用！如果你改变了core项目中的任何JAVA代码，只需要点一下书签，然后点击“Compile”。改变的代码几秒钟后就会产生效果。如果你修改了你的资源，你必须用上面的命令重启服务。

打包项目

每个平台都有不同的发布方式。在这一节我们来看看如何通过Gradle来打包发布。

desktop项目打包

`gradlew desktop:dist`

这条命令将在 `desktop/build/libs/` 目录下创建一个可执行的JAR文件。它包含了所有的必需代码，也包含了所有你在android/assets目录下的资源，并且可以通过双击它或者使用命令行 `java -jar jar-file-name.jar` 来运行。 你的用户必需已经为此安装了一个JVM才能运行。JAR文件可以运行在Windows，Linux和Mac OS X!

**如果你想要打包成一个带有JVM的JAR来发布，你可以使用我们的 packr tool!。用这种方式你的用户就不需要安装JVM，那将花费大约每个平台23-30mb的下载量。

android项目打包

`gradlew android:assembleRelease`

这条命令将在 `android/build/outputs/apk` 目录下创建一个未签名的APK。在你安装或者发布这个APK之前，你必需对它进行签名 sign it。通过上面的命，APK就以release模式创建完成，你只需要遵循Keytool和Jarsigner步骤。 你可以安装这个APK文件到任何允许未知源安装的Android设备 installation from unknown sources。

ios项目打包

`gradlew ios:createIPA`

这条命令将在 `ios/build/robovm` 目录下创建一个你将发布到Apple App Store的IPA。你可以按照Apple的引导来做 app store distribution。

web项目打包

`gradlew html:dist`

这条命令会将你的应用编译成为Javascript，并且将编译好的Javascript，Html和资源文件放到 `html/build/dist/` 目录下。这个目录下的内容必须部署到一个Web服务器上，例如Apache或者Nginx。就像你对待其他的静态HTML/Javascript站点一样。这里不再涉及到Java或者Java Applets。 如果你已经安装了Python，你可以通过在 `html/build/dist` 目录下执行以下命令来测试你发布的应用。

`python -m SimpleHTTPServer`

然后你打开一个浏览器跳转到 http://localhost:8000就能看到你的项目在运行。

调试与常见问题

Gradle任务失败

如果当你调用gradle时，构建或者刷新未能获得更多信息时(即失败，译者注)，请重新运行同样的命令，并增加 --debug 参数到命令中。例如： `./gradlew tasks --debug` 这将为你提供堆栈跟踪，并让你更好了解为什么Gradle失败。

常见问题

(已确认) AVG - 当运行 gradlew desktop:dist 时，杀软会导致失败。 请增加一个例外到杀软来允许运行。

调试项目

调整

你可能会和我一样，希望从dist任务获得一个jar。Gradle的目录规则是目录加上版本号，像desktop-1.0。 你可能也希望让每次构建都有一个以某种方式命名的独特版本号或构建日期，可以这样做：

在项目根目录的build.gradle文件，增加：

```
def getDate() {
    def date = new Date()
    def formattedDate = date.format('yyyyMMddHHmmss')
    return formattedDate
}
```

在allprojects配置快中新增'version = "0.1-build-" + getDate()'

在桌面项目中的dist任务中新增

'baseName = "myproject"'

可以肯定还有其他更好的处理方法，你可以花些时间来研究。 现在在你desktop/libs目录下有一个命名类似'project-0.1-build-20150120033412.jar'的jar文件，这将使发布更加容易，更易追踪，更少冲突等等。

Libgdx版本管理

Libgdx是一个活跃的开源项目，使用较新的版本是一个不错的选择，你可以通过以下网址获取相关信息 http://libgdx.badlogicgames.com/versions.html

Libgdx使用Gradle管理依赖，所以切换Libgdx版本是一件很轻松的事情。

Libgdx和大多数开源项目一样，有两种版本：

- Release版本: 稳定版本。你可以在Maven Central查看版本号。
- Nightly版本: 俗称快照版本。每个提交到代码仓库的修改都会触发快照版本的生成。快照版本一般为版本号加上SNAPSHOT，比如 1.0.1-SNAPSHOT. 你可以在 这里查看快照版本。

基于Gradle的项目更换版本是一件非常轻松的事情。打开 `build.gradle` 文件，然后找到如下声明:

```
gdxVersion = "1.5.2"
```

你现在看到的版本可能已经高于1.5.2了。在你需要的时候，你只需要确定版本字段的内容，你就可以简单地升级Libgdx版本（或者降级）。同样的，你还可以升级在Gradle管理下的依赖。编辑完成后保存 `build.gradle` 文件。

接下来的步骤和你的IDE有关:

- **Eclipse**: 在package explore中选择所有项目，右键选择 `Gradle -> Refresh All`。 Eclipse会自动下载依赖然后将下载的jar文件配置到项目中。
- **Intellij IDEA**: 一旦你的build.gradle文件修改，IDE会自动显示一个刷新按钮供。点击以后，IDE会自动下载依赖。你也可以到Gradle视图中找到刷新按钮或者'builddependents'任务并点击。
- **Netbeans**: 在"Projects"视图中, 选中主项目右键选择"Reload Project"。 所有子项目都会自动重新加载。
- **Command Line**: 调用任何命令都会自动下载新版本依赖并运用。

使用Gradle管理版本就是这么简单。不需要手动下载jar，so文件或者其他任何东西了。只需要修改版本号，然后在IDE或者命令行中执行相应的刷新就行了。

Useful links

Dependency management with Gradle is easy to understand, and has many different approaches. If you are familiar with Maven or Ivy, Gradle is fully compatible with both approaches, as well as being able to support custom approaches. If you aren't familiar with Gradle, there are great resources on their site to learn, it is recommended you give them a read to get comfortable with Gradle.

- Gradle's User Guide
- Gradle's Depedency Management Guide
- [Declare your dependencies](http://www.gradle.org/docs/current/userguide/dependency_management.html#sec:how_to_declare_your_dependencies)

Guide to build.gradle

Gradle projects are managed by `build.gradle` files in their root directory. If you have used the gdx-setup.jar to build your libgdx project you will notice the structure: Structure Example

The root directory, and each sub directory contains a `build.gradle` file, for clarity we will define the dependencies in the root directory's `build.gradle` file. (Note it can be done in each of the `build.gradle` scripts in the sub directories, it is just cleaner and easier to follow when it is handled all in one place)

Here is a small section of the *default* buildscript that is generated from the setup:

Full script you will see will differ slightly depending on what other modules you have, see here

```
buildscript {
    //Defines the repositories that are required by this script, e.g. android plugin
    repositories {
        //maven central repository, needed for the android plugin
        mavenCentral()
        //repository for libgdx artifacts
        maven { url "https://oss.sonatype.org/content/repositories/snapshots/" }
    }
    //
    dependencies {
        //Adds the android gradle plugin as a dependency of this buildscript
        classpath 'com.android.tools.build:gradle:0.9+'
    }
}

allprojects {
    apply plugin: "eclipse"
    apply plugin: "idea"

    version = "1.0"
    ext {
        appName = "%APP_NAME%"
        gdxVersion = "1.0-SNAPSHOT"
        roboVMVersion = "0.0.10"
    }

    repositories {
        //Defines all repositories needed for all projects
        mavenLocal();
        mavenCentral()
        maven { url "https://oss.sonatype.org/content/repositories/snapshots/" }
    }
}

project(":core") {
    apply plugin: "java"

    dependencies {
        //Defines dependencies for the :core project, in this example the gdx depdendency
        compile "com.badlogicgames.gdx:gdx:$gdxVersion"
    }
}

project(":desktop") {
    apply plugin: "java"

    dependencies {
    //Defines dependencies for the :desktop project, adds dependency on the :core project as well as
    //The gdx lwjgl backend and native dependencies
        compile project(":core")
        compile "com.badlogicgames.gdx:gdx-backend-lwjgl:$gdxVersion"
        compile "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-desktop"
    }
}

project(":android") {
    apply plugin: "android"

    configurations { natives }

    dependencies {
        //defines dependencies for the :android project, depends on the :core project,
        //and also the android backends and all platform natives. Note the 'natives' classifier
        //in this project
        compile project(":core")
        compile "com.badlogicgames.gdx:gdx-backend-android:$gdxVersion"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-x86"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi-v7a"
    }
}
```

Libgdx Dependencies

Dependencies are configured in the **root** `build.gradle` file as shown in the build.gradle guide above. In order to add an external dependency to a project, you must declare the dependency correctly under the correct part of the build.script.

(Some) Libgdx extensions are mavenized and pushed to the maven repo, which means we can very easily pull them into our projects from the `build.gradle` file. You can see in the list below of the format that these dependencies take. If you are familiar with maven, notice the format:

```
compile '<groupId>:<artifactId>:<version>:<classifier>'
```

Let's take a quick example to see how this works with the root `build.gradle` file.

As mentioned earlier, you do not need to modify the individual `build.gradle` files in each of the different platform-specific folders (e.g., -desktop, -ios, -core). You only need to modify the root `build.gradle` file.

Here we see the dependencies for the FreeType Extension, say we want our Android project to have this dependency. We locate our `project(":android")` stub in the root directory's `build.gradle` :

```
project(":android") {
    apply plugin: "android"

    configurations { natives }

    dependencies {
        compile project(":core")
        compile "com.badlogicgames.gdx:gdx-backend-android:$gdxVersion"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-x86"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi-v7a"
    }
}
```

We know our FreeType extension has declarations:

```
compile "com.badlogicgames.gdx:gdx-freetype:$gdxVersion"
natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-armeabi"
natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-armeabi-v7a"
natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-x86"
```

So all we need to do is whack it in the dependencies stub

```
project(":android") {
    apply plugin: "android"

    configurations { natives }

    dependencies {
        compile project(":core")
        compile "com.badlogicgames.gdx:gdx-backend-android:$gdxVersion"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-x86"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi-v7a"

        compile "com.badlogicgames.gdx:gdx-freetype:$gdxVersion"
        natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-armeabi"
        natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-armeabi-v7a"
        natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-x86"
    }
}
```

And we are done, our android project now has the freetype dependency. After this you will need to refresh your dependencies. Easy eh.

Libgdx Extensions

Mavenized libgdx extensions ready to import from the `build.gradle` script include:

- [Box2D] (#box2d-gradle)
- [Bullet] (#bullet-gradle)
- [FreeTypeFont] (#freetypefont-gradle)
- Controllers
- [Tools] (#tools-gradle)
- [Box2DLights] (#box2dlights-gradle)
- [Ai] (#ai-gradle)

Box2D Gradle

Core Dependency:

```
compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"
```

Desktop Dependency:

```
compile "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-desktop"
```

Android Dependency:

```
compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"
natives "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-armeabi"
natives "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-armeabi-v7a"
natives "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-x86"
```

iOS Dependency:

```
compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"
natives "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-ios"
```

HTML Dependency:

```
compile "com.badlogicgames.gdx:gdx-box2d-gwt:$gdxVersion:sources"
compile "com.badlogicgames.gdx:gdx-box2d:$gdxVersion:sources"
```

Bullet Gradle

Core Dependency:

```
compile "com.badlogicgames.gdx:gdx-bullet:$gdxVersion"
```

Desktop Dependency:

```
compile "com.badlogicgames.gdx:gdx-bullet-platform:$gdxVersion:natives-desktop"
```

Android Dependency:

```
compile "com.badlogicgames.gdx:gdx-bullet:$gdxVersion"
natives "com.badlogicgames.gdx:gdx-bullet-platform:$gdxVersion:natives-armeabi"
natives "com.badlogicgames.gdx:gdx-bullet-platform:$gdxVersion:natives-armeabi-v7a"
natives "com.badlogicgames.gdx:gdx-bullet-platform:$gdxVersion:natives-x86"
```

iOS Dependency:

```
compile "com.badlogicgames.gdx:gdx-bullet:$gdxVersion"
natives "com.badlogicgames.gdx:gdx-bullet-platform:$gdxVersion:natives-ios"
```

HTML Dependency: Not compatible!

FreeTypeFont Gradle

Core Dependency:

```
compile "com.badlogicgames.gdx:gdx-freetype:$gdxVersion"
```

Desktop Dependency:

```
compile "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-desktop"
```

Android Dependency:

```
compile "com.badlogicgames.gdx:gdx-freetype:$gdxVersion"
natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-armeabi"
natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-armeabi-v7a"
natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-x86"
```

iOS Dependency:

```
compile "com.badlogicgames.gdx:gdx-freetype:$gdxVersion"
natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-ios"
```

HTML Dependency: Not compatible!

Controllers Gradle

Core Dependency:

```
compile "com.badlogicgames.gdx:gdx-controllers:$gdxVersion"
```

Desktop Dependency:

```
compile "com.badlogicgames.gdx:gdx-controllers-desktop:$gdxVersion"
compile "com.badlogicgames.gdx:gdx-controllers-platform:$gdxVersion:natives-desktop"
```

Android Dependency:

```
compile "com.badlogicgames.gdx:gdx-controllers:$gdxVersion"
compile "com.badlogicgames.gdx:gdx-controllers-android:$gdxVersion"
```

iOS Dependency: Not supported, but you can still compile and run your iOS app. Controllers just won't be available

HTML Dependency:

```
compile "com.badlogicgames.gdx:gdx-controllers:$gdxVersion:sources"
compile "com.badlogicgames.gdx:gdx-controllers-gwt:$gdxVersion"
compile "com.badlogicgames.gdx:gdx-controllers-gwt:$gdxVersion:sources"
```

Tools Gradle

Core Dependency:

```
Dont put me in core!
```

Desktop Dependency:

```
compile "com.badlogicgames.gdx:gdx-tools:$gdxVersion"
```

Android Dependency: Not compatible!

iOS Dependency: Not compatible!

HTML Dependency: Not compatible!

Box2DLights Gradle

- **Note:** this extension also requires the Box2D extension

Core Dependency:

```
compile "com.badlogicgames.box2dlights:box2dlights:1.2"
```

Android Dependency:

```
compile "com.badlogicgames.box2dlights:box2dlights:1.2"
```

HTML Dependency:

```
compile "com.badlogicgames.box2dlights:box2dlights:1.2:sources"
```

Ai Gradle

Core Dependency:

```
compile "com.badlogicgames.gdx:gdx-ai:$gdxVersion"
```

Android Dependency:

```
compile "com.badlogicgames.gdx:gdx-ai:$gdxVersion"
```

HTML Dependency:

```
compile "com.badlogicgames.gdx:gdx-ai:$gdxVersion:sources"
```

and in `./html/src/yourgamedomain/GdxDefinition*.gwt.xml` add `<inherits name="com.badlogic.gdx.ai"/>`

External Dependencies

Adding external repositories

Gradle finds files defined as dependencies by looking through all the repositories defined in the buildscript. Gradle understands several repository formats, which include Maven and Ivy.

Under the `allprojects` stub, you can see how repositories are defined. Here is an example:

```
allprojects {
    repositories {
        // Remote Maven repo
        maven { url "https://oss.sonatype.org/content/repositories/snapshots/" }
        // Maven Central Repo
        mavenCentral()
        // Local Maven repo
        mavenLocal()
        // Remote Ivy dir
        ivy { url "http://some.ivy.com/repo" }
        // Local Ivy dir
        ivy { url "../local-repo" }
    }
}
```

Adding Dependencies

External dependencies are identified by their group, name, version and sometimes classifier attributes.

```
dependencies {
    compile group: 'com.badlogicgames.gdx', name: 'gdx', version: '1.0-SNAPSHOT', classifier: 'natives-desktop'
}
```

Gradle allows you to use shortcuts when defining external dependencies, the above configuration is the same as:

```
dependencies {
    compile 'com.badlogicgames.gdx:gdx:1.0-SNAPSHOT:natives-desktop'
}
```

Mavenizing Local Dependencies

If you would prefer to use maven repositories to manage local .jar files, these two commands will take any local .jar file and install them (and their source) to your local maven repository.

```
mvn install:install-file -Dfile=<path-to-file> -DgroupId=<group-id> -DartifactId=<artifact-id> -Dversion=<version> -Dpa
```

```
mvn install:install-file -Dfile=<path-to-source-file> -DgroupId=<group-id> -DartifactId=<artifact-id> -Dversion=<versio
```

To then set up gradle to include your new dependency, edit your build.gradle file in the root project directory and edit the core project entry:

```
project(":core") {
   ...

    dependencies {
        ...
        compile "<group-id>:<artifact-id>:<version>"
        compile "<group-id>:<artifact-id>:<version>:sources"
    }
}
```

After this you will need to refresh your dependencies for your IDE to see, so run: Command line - `$ ./gradlew --refresh-dependencies` Eclipse - `$ ./gradlew eclipse` IntelliJ - `$ ./gradlew idea`

Also, don't forget that any dependencies added this way also need to be included in the [GWT inheritance file] (#gwt-inheritance).

File Dependencies

If you have a dependency that is not mavenized, you can still depend on them!

To do this, in your project stub in the root `build.gradle` file, locate the dependencies { } section as always, and add the following:

```
dependencies {
    compile fileTree(dir: 'libs', include: '*.jar')
}
```

This will include all the .jar files in the libs directory as dependencies.

NOTE: "dir" is relative to the project root, if you add the dependencies to your android project, 'libs' would need to be in the android/ directory. If you added the dependencies in the core project, 'libs' would need to be in the core/ directory.

An example with a more *complete* script:

```
project(":android") {
    apply plugin: "android"

    configurations { natives }

    dependencies {
        compile project(":core")
        compile "com.badlogicgames.gdx:gdx-backend-android:$gdxVersion"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-x86"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi"
        natives "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-armeabi-v7a"

        compile "com.badlogicgames.gdx:gdx-freetype:$gdxVersion"
        natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-armeabi"
        natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-armeabi-v7a"
        natives "com.badlogicgames.gdx:gdx-freetype-platform:$gdxVersion:natives-x86"
        compile fileTree(dir: 'libs', include: '*.jar')
    }
}
```

It is worth nothing that these file dependencies are not included in the published dependency descriptor for your project, but they are included in transitive project dependencies within the same build.

Android Pitfall

When adding `flat file` dependencies to a project, for example the core project, you would need to duplicate the dependency declaration for the android project. This is because the Android Gradle plugin can't handle transitive `flat file` dependencies.

For example, if you were to add the all the jars in your `libs` directory as dependencies for your project, you would need to do the following.

```
project(":core") {
   ...
   compile fileTree(dir: '../libs', include: '*.jar')
   ...
}

// And also

project(":android") {
   ...
   compile fileTree(dir: '../libs', include: '*.jar')
   ...
}
```

This is only required for the android project, all other projects inherit `flat file` dependencies OK.

External Dependencies Examples

Universal-Tween-Engine by jar

1. Download the jars
2. Place jars in the directory core/libs (You can change this if you wish)
3. Alter your **build.gradle** script in the root directory as follows:

    - Locate the :core stub where the core project's dependencies are declared
    - Add the line in dependencies

      ```
      compile fileTree(dir: 'libs', include: '*.jar')
      ```

    - Your script should now look a little like this:

      ```
      project(":core") {
      ...

      dependencies {
         ...
         compile fileTree(dir: 'libs', include: '*.jar')
      }
      }
      ```

Finally Refresh your Gradle project, either on command line or using your IDE plugin.

Universal-Tween-Engine by locally mavenizing maven

First, download and extract the tween-engine-api from its repository (https://code.google.com/p/java-universal-tween-engine/). To install this dependency and its source files into your local maven repo, use these commands:

```
mvn install:install-file -Dfile=tween-engine-api.jar -DgroupId=aurelienribon -DartifactId=tweenengine -Dversion=6.3.3 -

mvn install:install-file -Dfile=tween-engine-api-sources.jar -DgroupId=aurelienribon -DartifactId=tweenengine -Dversion
```

With the tween engine jars in your local maven repo, add a dependency to them in your build.gradle in the root project file.

```
project(":core") {
   ...

    dependencies {
        ...
        compile "aurelienribon:tweenengine:6.3.3"
        compile "aurelienribon:tweenengine:6.3.3:sources"
    }
}
```

Add the inheritance to GdxDefinition.gwt.xml and GdxDefinitionSuperdev.gwt.xml

```
<inherits name='aurelienribon.tweenengine'/>
```

Then just refresh your dependencies.

```
$ ./gradlew --refresh-dependencies
```

Gwt Inheritance

Gwt is special, so in order to let the GWT compiler know what modules the project depends on, and *inherits* from, you need to let it know.

This is done in the `gwt.xml` files in the gwt sub directory. You will need to make the changes both to the `GdxDefinition.gwt.xml` and also the `GdxDefinitionSuperdev.gwt.xml`.

The *default* gwt.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit trunk//EN" "http://google-web-toolkit.googlecode.com/sv
<module rename-to="html">
    <inherits name='com.badlogic.gdx.backends.gdx_backends_gwt' />
    <inherits name='com.badlogic.mygame.MyGame' />
    <entry-point class='com.badlogic.mygame.client.GwtLauncher' />

    <set-configuration-property name="gdx.assetpath" value="../android/assets" />
</module>
```

We depend on the libgdx gwt backend, as well as the core project, so we have them defined in a tag. So when you add your dependency via methods above, you need to add it here too!

Libgdx Extension Inherits

These are the libgdx extensions that are supported in gwt

- Libgdx Core - `<inherits name='com.badlogic.gdx.backends.gdx_backends_gwt' />`
- Box2d - `<inherits name='com.badlogic.gdx.physics.box2d.box2d-gwt' />`
- Box2dLights - `<inherits name='Box2DLights' />`
- Controllers - `<inherits name='com.badlogic.gdx.controllers.controllers-gwt' />`

An example: The Universal Tween Engine

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit trunk//EN" "http://google-web-toolkit.googlecode.com/sv
<module rename-to="html">
    <inherits name='com.badlogic.gdx.backends.gdx_backends_gwt' />
    <inherits name='com.badlogic.mygame.MyGame' />
    //Let's inherit tween
    <inherits name='aurelienribon.tweenengine'/>
    <entry-point class='com.badlogic.mygame.client.GwtLauncher' />

    <set-configuration-property name="gdx.assetpath" value="../android/assets" />
</module>
```

Step by step Universal Tween Engine guide

This guide uses everything that is explained in this article to add the Universal Tween Engine to your project as a dependency.

Click me

简介

Libgdx是非常强大的，一旦你习惯并能熟练使用它，那么它将是一个强力的工具，但与你的IDE集成可能导致一些工作流程问题。具体的情况每个项目都不同，每个人也不同。接下来是一个例子：

Libgdx工程一般是一个多项目工程，你可能需要经常运行桌面构建来检查你的最新变化。

由于Gradle的工作原理（它允许访问和更改的任何部分项目的任何部分构建的灵活性），它必须在执行任何任务之前配置的多项目工程中的所有项目。

当你只有一个桌面项目，这通常是非常迅速的，但是当你在Android项目中或者HTML项目做出更改时，启动时间将大大增加。这在IntelliJ IDEA中尤其明显。

加速Gradle

你可以尝试一些事情让Gradle的运行更高效：

daemon模式

Daemon模式的目的是降低执行任务的时间，尤其执行频繁任务是所消耗的启动时间。

您可以通过添加'--daemon`标志来运行的daemon模式。

您还可以添加这个选项标志到项目的根目录下的 `gradle.properties` 文件中，这样所有所有任务运行都会自动采用这个配置。添加：`org.gradle.daemon = TRUE`

扩展阅读： *Daemon http://www.gradle.org/docs/current/userguide/gradle_daemon.html*

Demand配置

This setting helps to solve the issue of Gradle configuring the world before execution. Using this setting, the root project is always configured, the project in the directory where the build is executed is configured, and only dependent projects are configured, resulting in much faster build times.

To use this setting add the following to the `gradle.properties` file in the root directory of your project: Add the line; `org.gradle.configureondemand=true`

How to remove Gradle IDE integration from your project

Note: This only applies for Eclipse and Intellij IDEA users

With the project that is generated from gdx-setup.jar, the Gradle IDEA, and Gradle Eclipse plugins are applied to all projects. These plugins allow you to generate IDE specific files that you can then work from.

Doing this allows you to keep Gradle out of your IDE, so that when you want to launch any task, it wont configure your whole project and will be as quick as if you had a regular IDE generated project. This does mean that you need to sort out support for the sub modules in the IDE yourself, android support, robovm ect. (It is still recommended that you do all GWT/html stuff from the command line, as well as using the great packaging tasks that are supplied)

It also allows you to keep the power of Gradle at your disposal, to handle all your dependencies, packaging tasks, signing tasks, deployment and anything else you may want to implement by running from the command line.

创建IDEA项目

From the command line, run the following command from your project root directory: (If on a UNIX based OS, use ./gradlew to invoke gradle)

```
gradlew idea
```

In IDEA: File > OPEN > Locate the .ipr that the task above generates

创建Eclipse项目

From the command line, run the following command from your project root directory: (If on a UNIX based OS, use ./gradlew to invoke gradle)

```
gradlew eclipse
```

In Eclipse: File > Import > Existing project into workspace > Locate the .project file and import

Libgdx without Gradle

You are **never** forced to use Gradle, it is just recommended. If you want clarification on why it is recommended, check [here] (#why-does-libgdx-recommend-gradle)

If you don't want to use Gradle, you have the following choices.

- You can use the setup to create your project still and never touch Gradle again. Follow the guide from [here] (#how-to-remove-gradle-ide-integration-from-your-project) and never touch Gradle again.
- You can setup your project manually.
- You can maintain the old setup and use that to generate your projects.

Doing any of the above will have the following consequences:

- You no longer have access to packaging tasks, you will have to package and deploy your projects yourself
- You will lose support for IDE's other than IDEA and Eclipse
- You install all the plugins required for your IDE to interpret each of the sub-projects
- You manually update your dependencies from now on, don't forget the platform specific natives too!

Why does LibGDX recommend Gradle

Support

- Using Gradle, LibGDX is supported regardless of what IDE you use, even if you don't use one. Your development environment doesn't support Android/RoboVM/HTML/Java? Don't worry, Gradle can compile/build/run/package your project no matter what you use to code with.
- Having one unified system of managing your project makes it easy to get support from other users. The more people that are using the same approach, the larger the support user base grows.

Dependency Management

One of the biggest pros of any build/dependency management system.

- Easy to understand
- Insanely quick and simple to stay up to date with the latest LibGDX, update one line-refresh-done. No more jar hunting.
- Transitive dependencies (Inherit dependencies from projects you depend on)
- Custom dependency definitions, depend on anything!
- Full compatibility with all repository types (Maven, Ivy, etc)

Flexibility

Gradle is built with Ant and Maven very much in mind, and aims to combine the two to make a very powerful and flexible build automation tool.

- Structure your build however you want, add custom tasks, use the rich API to completely customize your build to suit your needs
- Want to run your project on a CI server? Just throw it up there install the jenkins Gradle plugin and you're good to go
- The Gradle Wrapper allows you to be free of installing Gradle!

Maintainability

As LibGDX grows, extensions are created from scratch or by removing them from core where appropriate, making LibGDX more modular to avoid bloat when you don't need those extensions. (This happened with Box2D recently). This means **more** dependencies for you to add and therefore more jars to add. If you are using a minimalist project with no extensions, if you were to manually manage your dependencies that would mean 1 core dependency, 1 desktop dependency, 4 android dependencies, 4 ios dependencies and 3 gwt dependencies.

That is 13 jars you need to track down every time you want to update to the latest LibGDX with all the greatest new features. If you use Box2D, Box2D lights, FreeType, you are looking at a lot of dependencies which will quickly become very annoying to update and you will probably put off doing it. The setup has been built with this in mind, and makes it very quick to add these to your project and painless to update by changing one line with the power of Gradle.

The setup itself is very simple, very transparent and maintainable by developers that didn't create it. This lowers the bus factor for LibGDX and reduces strain when the setup requires an update/alteration.

As previously mentioned, the Gradle project that is created by **gdx-setup.jar** includes tasks to run and package/distribute each project. This allows LibGDX to have a unified way of packaging your project, rather than x amount of ways for y amount of IDE's. This saves a lot of time in terms of writing documentation, following what every IDE is doing, and providing support on the forums; this gives developers more time to work on great features for LibGDX to help make your project awesome.

It revolutionizes everything

Look down. Now look back up. Look down again. You're sitting on a chair. Look back up. You aren't using Gradle, your life is OK but it could be better. You read this Article and you decide you want to use Gradle. You use the **gdx-setup.jar** to generate your project and you get tinkering with it. You use the power of Gradle to create 500 flappy clones from the same source code. You look down again, your chair is now made of gold and there are an infinite amount of monkeys with keyboards to code for you. Your life is now better and you can now concentrate on this things that matter.