МУНИЦИПАЛЬНОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ  
г. Хабаровска

“ВОЛОЧАЕВСКИЙ ЛИЦЕЙ”

ИНДИВИДУАЛЬНЫЙ ИТОГОВЫЙ ПРОЕКТ

**Тема:** “Практическое пособие по созданию игры с помощью Java фреймворка libGDX”

Выполнил обучающийся:  
11.2 класса  
Решетников Сергей Валерьевич

Научный руководитель:  
учитель информатики  
Людмила Викторовна  
Дементьева

г. Хабаровск  
2021-2022 учебный год.

Оглавление

[ВВЕДЕНИЕ 3](#_Toc87410977)

[Плюсы и минусы libGDX 4](#_Toc87410978)

[Глава 1. Настройка проекта 5](#_Toc87410979)

[Изучение Java 5](#_Toc87410980)

[IDE 5](#_Toc87410981)

[Установка Java (рекомендуется) 6](#_Toc87410982)

[Генерация проекта 7](#_Toc87410983)

[Импорт проекта 10](#_Toc87410984)

[Запуск проекта 10](#_Toc87410985)

[Что делать, если не удалось запустить проект? 11](#_Toc87410986)

[Итог 1 главы 11](#_Toc87410987)

[Глава 2. Создание базовой версии игры. 13](#_Toc87410988)

[Жизненный цикл игры 13](#_Toc87410989)

[Вывод текста на экран 16](#_Toc87410990)

[Импорт ассетов 18](#_Toc87410991)

[Игрок 20](#_Toc87410992)

[Логгинг 22](#_Toc87410993)

[Архитектура игры 22](#_Toc87410994)

[Перечисляемые 23](#_Toc87410995)

[Карта 24](#_Toc87410996)

[Создание доски (obsolete) 26](#_Toc87410997)

# ВВЕДЕНИЕ

Данная работа посвящена изучению Java, а конкретнее одного из ее фреймворков под названием “libGDX”. **Тема** данного проекта это и отражает: “Создание игры с помощью Java фреймворка libGDX”.  
Собственно, данный фреймворк и язык являются как и **объектами** исследования, так и **предметами** исследования.  
  
**Проблема** данного проекта заключается в отсутствии достаточного количества самоучителей для этого фреймворка, что останавливает некоторых людей в изучении его, а изучение по профессиональной документации не представляется возможным.  
  
**Гипотеза**: самый легкий способ чему-то научиться, это синхронно повторять действия за кем-то, параллельно понимая для чего они делаются.

**Цель** проекта: научиться прототипировать приложения на libGDX, а также параллельно создать версию игры “Uno”.  
**Задачи** этого проекта:

* Установить необходимые программы для прототипирования.
* Рассмотреть базовые вещи в IDE (todo:)
* Разработать первую игру на libGDX.

**Актуальность:** некоторые люди желают программировать на Java и понимают основы языка, но не обладают достаточными практическими навыками для выбора фреймворка и написания базовых программ.  
Данный проект как раз таки и несет практическую направленность, и предполагает изучение теоретических основ фреймворка, чтобы человеку было легче выбрать.  
  
Этот проект собой подразумевает то, что вы абсолютный новичок в программировании на Java и libGDX. Если Вы владеете хорошими навыками программирования и английского языка, то Вам хватит и англоязычной документации.

Если Вам нужны исходники кода, был создан специальный Git репозиторий, где можно подробно посмотреть воссоздание проекта, по помеченным коммитам. Ссылка на него в приложении.

[Гайд на русском](http://www.libgdx.ru/p/guide.html) [[1]](#footnote-1)| [документация на английском](https://github.com/libgdx/libgdx/wiki/)[[2]](#footnote-2).

# Плюсы и минусы libGDX

Перед началом разработки, стоило бы обозначить плюсы и минусы данного фреймворка, чтобы Вы понимали для чего он нужен. Языки программирования обычно используются как инструменты, и очень важно подобрать правильный для специфичной задачи.  
libGDX – кроссплатформенный фреймворк для создания игр. Он также подойдет также для создания различных приложений, но если Вы хотите создать приложение, которое не работает с графикой (тут используется отрисовка с помощью OpenGL) и использует нативные компоненты системы, я Вам советую не писать на libGDX, несмотря на то, что он написан на том же языке, и искать другие Android фреймворки.

Также, если Вы ищите libGDX, только для котлина, есть также неофициальный фреймворк libKTX.

Плюсы:

* Бесплатность.
* Кроссплатформенность
* Нативность (не надо использовать NDK)
* Низкий порог вхождения
* Open-source, другими словами открытые исходники
* Является больше библиотекой, чем фреймворком, поэтому предполагает строительство своей архитектуры – это плюс для новичков, т.к дает навыки проектирования приложений.  
  Несмотря на это, для некоторых это может оказаться слишком низкоуровневым, так как требуется огромное количество кода для поддержки.

Минусы:

* Долгие выходы обновлений
* Нехватка туториалов, что является причиной создания проекта.  
  Иногда придется ломать голову, что может быть полезно в будущем.
* Слабо развитое 3D, если Вам нужно быстрое 3D, переходите к Unity.

Хотелось бы также сказать, что одним из самых базовых навыков программиста является поиск информации, так что если что-то непонятно или не работает, Вы всегда можете найти что-либо, поискав в поисковике.

# Глава 1. Настройка проекта

## Изучение Java

Несмотря на то, что этот проект был создан для абсолютных новичков, Вам все же необходимы знать базовые элементы языка Java. Я не смогу тут рассмотреть изучение Java как языка, но могу порекомендовать [платформу Metanit](https://metanit.com/java/tutorial/) [[3]](#footnote-3)для изучения абсолютных основ данного языка, т.к иначе Вы рискуете ничего не понять. Из IDE советую использовать **Intellij Idea.**

## IDE

Для начала, вам нужно установить **IDE** (**Integrated Development Environment**). Это что-то вроде редактора для всех ваших файлов, а также контейнера для нужного софта вроде компилятора, Gradle и т.п..

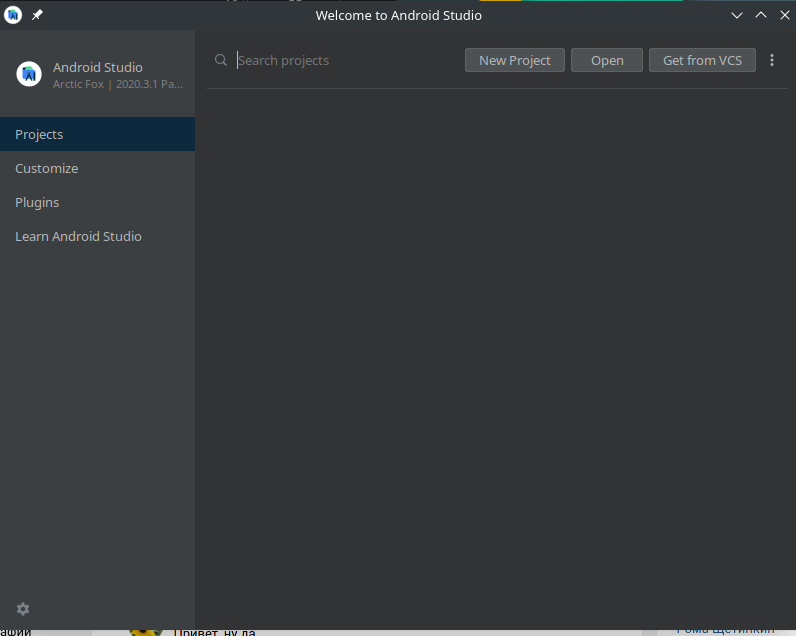
Если IDE у Вас уже установлена, Вы можете смело пропустить [эту часть.](#Java)

Есть множество вариантов, вроде **Android Studio**, **Intellij Idea**, **Eclipse**, или даже можно воспользоваться обычным блокнотом.

Но я Вам рекомендую установить [**Android Studio**](https://developer.android.com/studio)[[4]](#footnote-4),так как:

* Мы будем компилировать приложение на Android, а отлаживать приложение с него удобнее всего.
* Все примеры в данном проекте будут основаны на нём, так что следовать инструкциям будет проще.
* Она бесплатна, и максимально проста для новичков.

После установки, запуска и стартовой настройки программы, у Вас высветится окошко с проектами наподобие этого:



Смело закрывайте окно. Вместо использования **Android Studio** для генерации проекта, мы будем использовать специальную утилиту от Libgdx, так как она упрощает создание проекта.

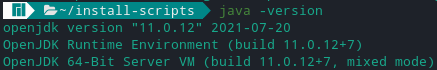
## Установка Java (рекомендуется)

Несмотря на предустановленную JDK (Java Development Kit) в Android Studio, я Вам также рекомендую установить Java 15 т.к она обладает новыми функциями, которые мы собираемся использовать. Вы можете также создать проект с Java 8-14, но тогда могут возникнуть ошибки в синтаксисе, а также проблемы с оптимизацией при точном следовании инструкции. (Java 16 и выше, по какой-то причине не поддерживаются libGDX),

Я Вам рекомендую пользоваться [AdoptOpenJdk](https://adoptopenjdk.net/releases.html)[[5]](#footnote-5), или же [liberica-jdk](https://libericajdk.ru/pages/downloads/)[[6]](#footnote-6) от Bellsoft.

После установки рекомендую проверить открыть командную строку/консоль, и проверить ее версию. (в windows: Win + R – cmd.exe открывает командную строку)

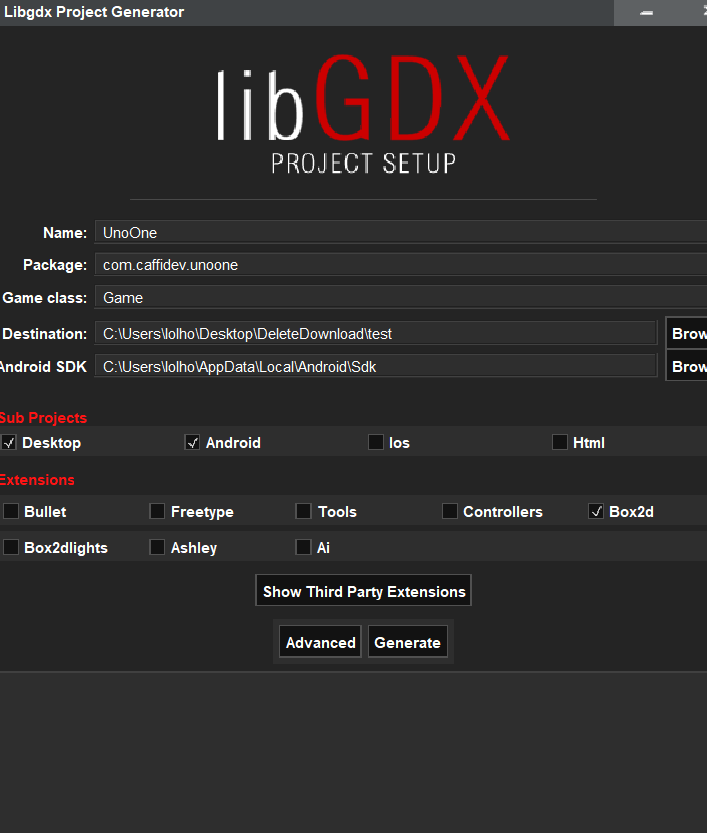
Проверить версию можно командой: java -version

  
*у меня jdk 11 как основная, не бойтесь если показана другая версия, вы в IDE всегда можете ее сменить*

Если вылезла ошибка о том, что такой команды нет, то либо Вы неверно ее установили, либо у Вас не прописана она в [переменной среды](https://java-lessons.ru/common-errors/javac-is-not-recognized).[[7]](#footnote-7)

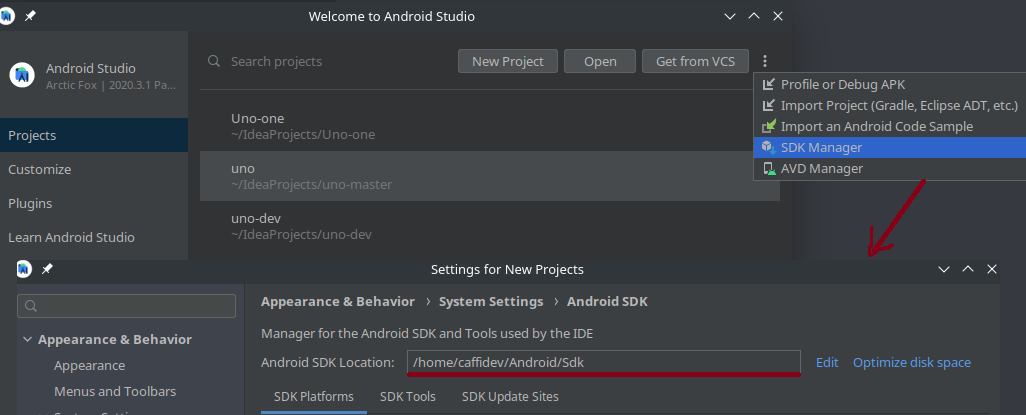
## Генерация проекта

Дальше Вам надо скачать специальную [утилиту](https://libgdx.com/assets/downloads/legacy_setup/gdx-setup_latest.jar)[[8]](#footnote-8) для генерации проектов libGDX.  
После ее скачивания, откройте консоль в папке, куда вы поместили файл и введите команду `java -jar [имя вашего файла]`.



Должен открыться следующий GUI (графический интерфейс)

У Вас спрашивают следующие параметры:

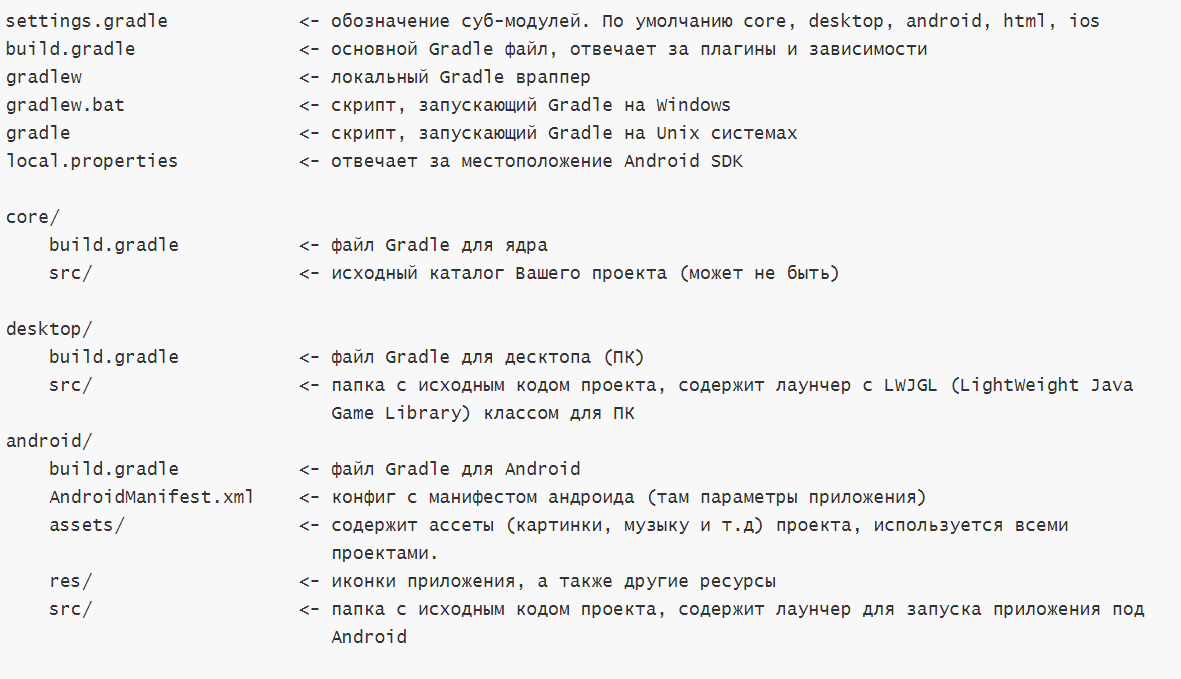
* Name: название Вашего приложения (лучше использовать kebab-case) (unoone)
* Package: Java пакет, в котором будет находится ваш код (com.{username}.unoone)
* Game Class: название основного класса (Game)
* Destination: папка, где код Вашего приложения будет создан.
* Android SDK: местоположение Android SDK.  
  Вы можете его найти его так:

Sub Projects: libGDX кроссплатформен и сам по себе построен на C/C++, а Java – скриптовый язык в нем. Тут мы можем настроить платформы, на которые мы будем компилировать код. Я выбрал Android и Desktop, но по желанию Вы можете выбрать и другие.

Extensions: расширения. В этом гайде они не понадобятся, за исключением Box2D. Подробнее Вы о них можете узнать тут в [документации](https://github.com/libgdx/libgdx/wiki/)[[9]](#footnote-9).

Чтобы сгенерировать проект, нажмите кнопку Generate.

Это сгенерирует директорию с данной расстановкой:



## Импорт проекта

1. Откройте файл build.gradle с помощью Android Studio в директории, в которой Вы сгенерировали проект.
2. Есть вероятность, что некоторые зависимости еще не скачались, поэтому возможно Вам придется перезапустить проект.  
   Чтобы это сделать, нажмите кнопку “Reimport all Gradle projects” в вкладке Gradle, которую можно открыть с помощью “View” -> “Tool Windows” -> Gradle.
3. В этой же вкладке, с версии Android Studio 4.2, задачи Gradle по умолчанию больше не видны. Чтобы их включить, зайдите в “Settings” -> “Experimental” и отключите галочку “Do not build Gradle task during Gradle sync”, после чего синхронизируйте проект с помощью “File” -> “Sync Project with Gradle Files”.
4. Я Вам также советую обновить плагин “Android Gradle Plugin” для наилучшей работоспособности.
5. На Android 12, при компиляции вылезает ошибка в манифесте, в которой сказано что в манифест нужно добавить параметр ‘android:exported’.
   1. Чтобы это сделать, зайдите в манифест AndroidManifest.xml
   2. Добавьте после android:name=”” android:exported=”true”
   3. Этот параметр отвечает за доступность компонента приложения другим приложениям.
6. Смените версию Java, если у Вас установлено несколько экземпляров.
7. Я Вам также рекомендую найти телефон, и подключить его к компьютеру, включив режим отладки в настройках разработчика. Это позволит запускать версию приложения под Андроид нативно, не прибегая к эмулятору.
8. При джаве выше 8 при запущенном проекте могут выскакивать предупреждения такого характера: “Illegal reflective access”.  
   Можно смело их игнорировать, это жалуется движок lwjgl на java >8 по неизвестной мне причине

## Запуск проекта

Проект можно запускать несполькими способами:

1. Меню Gradle:
   1. Откройте вкладку Gradle сбоку от Вашего окна.
   2. Расширьте задачи Вашего проекта, и выберите: desktop/android -> other -> run.
2. Консольная команда:
   1. Откройте Terminal снизу (если его нет, Вы можете добавить его с помощью View).
   2. Введите команду: ./gradlew desktop:run/android:run
3. Меню конфигурации:
   1. Кликните по DesktopLauncher правой кнопкой мыши.
   2. Выберите “Run DesktopLauncher.main()”;
   3. Откройте меню конфигурации сверху.
   4. Убедитесь что в Working Directory стоит ссылка на папку android/assets.
   5. Аналогично можно сделать и с AndroidLauncher, благодаря чему меняя конфигурацию можно будет менять проект двумя нажатиями мыши.

### Что делать, если не удалось запустить проект?

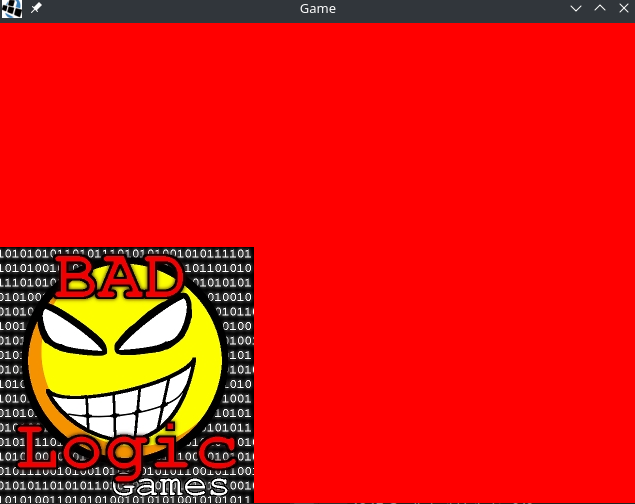
Если ошибка в коде, то исправьте ее по сообщению компилятора, но нежели если ошибка вылезает при билде Gradle, то тогда запустите команду:

./gradlew {платформа}:run , добавив параметр –debug, т.е ./gradlew {плафторма}:run –debug.

Это при повторной компиляции выдаст Вам более подробную ошибку, дав Вам больше информации для решения проблемы.

## Итог 1 главы

Если у Вас все в итоге получилось, Вы должны были увидеть вот эту счастливую мордашку:



В следующей главе мы:

* Рассмотрим жизненный цикл игры
* Продумаем архитектуру игры
* Создадим начальную версию игры.

# Глава 2. Создание базовой версии игры.

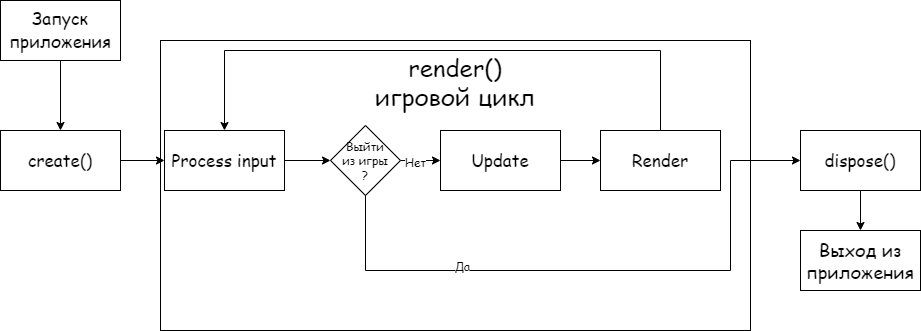
## Жизненный цикл игры

У каждой игры есть свой жизненный цикл. Структура этого цикла зависит от движка, и разработчики должны знать как он проходит. Разумеется, он есть и у libGDX:

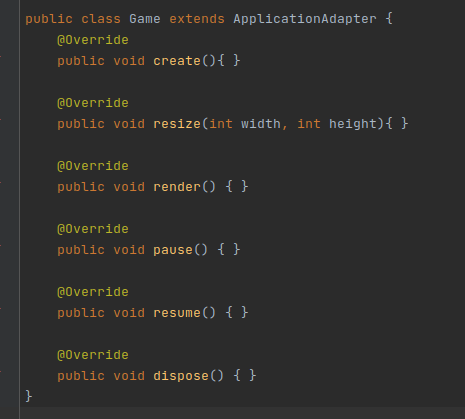
В новой версии libGDX наш основной класс в директории core/ расширяет класс ApplicationAdapter, который является стартовой точкой нашего приложения. Он отвечает за инициализацию нашего приложения, обновление игрового процесса, отображение изображений, установка паузы игры, сохранение состояния приложения и выгрузку ресурсов при выходе из приложения.



Для реализации приложения мы реализуем методы в приложении, которые работают согласно этому жизненному циклу:



Все эти методы содержатся в классе ApplicationAdapter, которые с помощью @override мы можем переопределить, зайдя в наш главный класс под в директории core/ и реализовав данные функции:



Благодаря этим методам мы можем реализовать структуру нашего приложения согласно данной схеме:

* create(): вызывается однократно после запуска приложения, инициализирует нужные классы, переменные, объекты для игрового процесса.
* render() - вызывается каждый раз, когда происходит отрисовка экрана. Вызывается непрерывно, пока мы не решим, что игра закончена. В показанной выше схеме видим, что функция универсальна, и то, что помимо отрисовки экрана мы можем в нее засунуть ввод данных и обновление переменных.  
  На самом деле, разработчик может включить в нее что угодно: музыку, логгинг, получение данных с сервера, и что вообще ему нужно. В этом libGDX и удобен, что в отличие от Unity в нем можно полностью продумать каркас своей программы, несмотря на то, что это дольше и сложнее.
* dispose() - вызывается при закрытии приложения (после метода pause()). Здесь можно подчистить за собой мусор, который не очистил за вас GC (Garbage Collector – сборщик мусора).

Но за что же отвечают остальные элементы?

На самом деле жизненный цикл не настолько прост, т.к помимо самого приложения у нас есть внешняя среда – операционная система, которая им управляет. Если на ПК это Windows/Linux/macOS, где наша программа является процессом, который можно закрыть/открыть/расширить и т.п, то  
в Android вместо процессов используются activity, которые работают по вот этой схеме:



Для таких случаев и нужны следующие методы:

* resize(int width, int height)- вызывается каждый раз, когда приложение меняет размер и оно не находится в состоянии паузы. Так же данный метод вызывается однажды, сразу после вызова метода create(). Передаваемые параметры ширины и высоты меняются в зависимости от размера окна так же каждый раз при вызове метода.
* pause() - вызывается прежде чем приложение будет закрыто. В Android это происходит, когда мы сворачиваем приложение. На пк метод вызывается перед вызовом метода dispose() при закрытии приложения. Обычно в этом методе сохраняют состояние приложение, потому что не факт, что к нему ещё вернуться.
* resume() - вызывается при получении фокуса приложением.

## Вывод текста на экран

Для начала стоило бы рассмотреть способ выводить текст на экран. Их существует много в libGDX, но мы рассмотрим batch.

В libGDX существует класс SpriteBatch. Он позволяет рисовать различные спрайты при помощи кода. Несмотря на то, что его обычно используют для отрисовки картинок, он вполне может использоваться для рендера текстового спрайта.

Чтобы показать на примере, давайте создадим класс, показывающий значение FPS в игре т.е кадров в секунду.

Кликните по папке проекта, и создайте класс под названием Framerate.  
Напишите следующий код:

|  |
| --- |
| **package** com.caffidev.unoone;  **import** **com.badlogic.gdx.Gdx**;  **import** **com.badlogic.gdx.graphics.OrthographicCamera**;  **import** **com.badlogic.gdx.graphics.g2d.BitmapFont**;  **import** **com.badlogic.gdx.graphics.g2d.SpriteBatch**;  **import** **com.badlogic.gdx.utils.Disposable**;  **import** **com.badlogic.gdx.utils.TimeUtils**;  **public** **class** **Framerate** **implements** Disposable {  **long** lastTimeCounted;  **private** **float** sinceChange;  **private** **float** framerate;  **private** BitmapFont font;  **private** SpriteBatch batch;  **private** OrthographicCamera cam;  **public** **Framerate**(){  lastTimeCounted = TimeUtils.millis();  sinceChange = **0**;  framerate = Gdx.graphics.getFramesPerSecond();  font = **new** BitmapFont();  batch = **new** SpriteBatch();  cam = **new** OrthographicCamera(Gdx.graphics.getWidth(), Gdx.graphics.getHeight());  }  **public** **void** **resize**(**int** screenWidth, **int** screenHeight){  cam = **new** OrthographicCamera(screenWidth, screenHeight);  cam.translate(screenWidth /**2**, screenHeight /**2** );  cam.update();  batch.setProjectionMatrix(cam.combined);  }    **public** **void** **renderWithUpdate**(){  update();  render();  }    **public** **void** **update**() {  **long** delta = TimeUtils.timeSinceMillis(lastTimeCounted);  lastTimeCounted = TimeUtils.millis();  sinceChange += delta;  **if**(sinceChange >= **1000**) {  sinceChange = **0**;  framerate = Gdx.graphics.getFramesPerSecond();  }  }  **public** **void** **render**() {  batch.begin();  font.draw(batch, "Debug:\n"+(**int**)framerate + " fps", **3**, Gdx.graphics.getHeight() - **3**);  batch.end();  }  **public** **void** **dispose**() {  font.dispose();  batch.dispose();  }  } |

|  |
| --- |
| **package** com.caffidev.unoone;  **import** **com.badlogic.gdx.Application**;  **import** **com.badlogic.gdx.ApplicationAdapter**;  **import** **com.badlogic.gdx.Gdx**;  **import** **com.badlogic.gdx.Input**;  **import** **com.badlogic.gdx.graphics.GL20**;  **import** **com.badlogic.gdx.graphics.Texture**;  **import** **com.badlogic.gdx.graphics.g2d.SpriteBatch**;  **import** **com.badlogic.gdx.graphics.g2d.TextureRegion**;  **import** **com.badlogic.gdx.scenes.scene2d.Stage**;  **import** **com.badlogic.gdx.utils.Logger**;  **import** **com.badlogic.gdx.utils.ScreenUtils**;  **import** **org.w3c.dom.Text**;  **public** **class** **Game** **extends** ApplicationAdapter {  **protected** Framerate framerate;  **@Override**  **public** **void** **create**(){  framerate = **new** Framerate();  }    **@Override**  **public** **void** **resize**(**int** width, **int** height){  framerate.resize(width, height);  }    **@Override**  **public** **void** **render**() {  // Clean screen  Gdx.gl.glClearColor(**0**, **0**, **0**, **1**);  Gdx.gl.glClear(GL20.GL\_COLOR\_BUFFER\_BIT);    framerate.renderWithUpdate();  **if**(Gdx.input.isKeyPressed(Input.Keys.ESCAPE)){ Gdx.app.exit();}  }    **@Override**  **public** **void** **pause**() { }    **@Override**  **public** **void** **resume**() { }    **@Override**  **public** **void** **dispose**() {  framerate.dispose();  }  } |

Каким чудом засунуть это

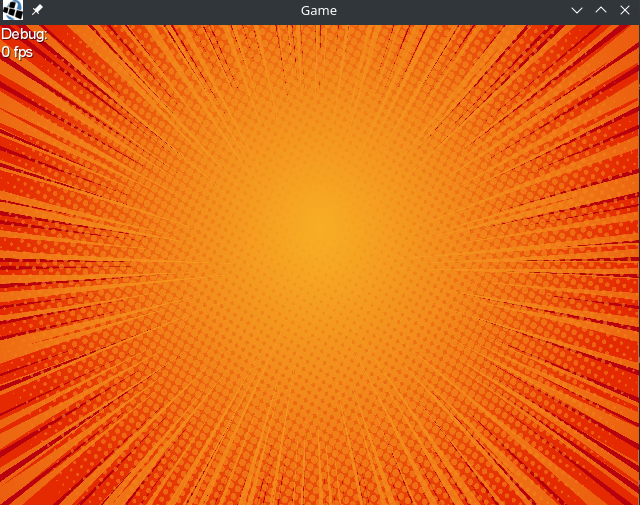
**if**(Gdx.input.isKeyPressed(Input.Keys.ESCAPE)){ Gdx.app.exit();}  
<..> - пропуск кода, для удобства

## Импорт ассетов

Давайте попробуем теперь с помощью SpriteBatch отрисовать картинку. Для этого нам нужно импортировать в проект какое-нибудь изображение.

Из-за условностей Android (каких?) , папка в android/assets является главной, и все ассеты должны храниться там.  
todo: показать как все это импортировать  
todo: как отрисовать

|  |
| --- |
| **package** com.caffidev.unoone;  **import** **com.badlogic.gdx.Application**;  **import** **com.badlogic.gdx.ApplicationAdapter**;  **import** **com.badlogic.gdx.Gdx**;  **import** **com.badlogic.gdx.Input**;  **import** **com.badlogic.gdx.graphics.GL20**;  **import** **com.badlogic.gdx.graphics.Texture**;  **import** **com.badlogic.gdx.graphics.g2d.SpriteBatch**;  **import** **com.badlogic.gdx.graphics.g2d.TextureRegion**;  **import** **com.badlogic.gdx.scenes.scene2d.Stage**;  **import** **com.badlogic.gdx.utils.Logger**;  **import** **com.badlogic.gdx.utils.ScreenUtils**;  **import** **org.w3c.dom.Text**;  **public** **class** **Game** **extends** ApplicationAdapter {  **protected** Framerate framerate;  **protected** SpriteBatch batch;  **protected** Texture background;    **@Override**  **public** **void** **create**(){  batch = **new** SpriteBatch();  framerate = **new** Framerate();  background = **new** Texture("background.jpg");  }    **@Override**  **public** **void** **resize**(**int** width, **int** height){  framerate.resize(width, height);  }    **@Override**  **public** **void** **render**() {  // Clean screen  Gdx.gl.glClearColor(**0**, **0**, **0**, **1**);  Gdx.gl.glClear(GL20.GL\_COLOR\_BUFFER\_BIT);  batch.begin();  batch.draw(background, **0**, **0**,Gdx.graphics.getWidth(), Gdx.graphics.getHeight());  batch.end();  framerate.renderWithUpdate();  **if**(Gdx.input.isKeyPressed(Input.Keys.ESCAPE)){ Gdx.app.exit();}  }    **@Override**  **public** **void** **pause**() { }    **@Override**  **public** **void** **resume**() { }    **@Override**  **public** **void** **dispose**() {  framerate.dispose();  }  } |

  
todo: конец, предложить импортировать ассеты карт

Вам необязательно использовать эти ассеты. Вы можете нарисовать как свои, как и придумать имена объектов.  
Скачать: <https://drive.google.com/drive/folders/110AmAVddNwU7BJfi1SkgUJI1nZuJYgnA?usp=sharing>

## Игрок

Пора потиихоньку переходить к созданию игры.  
Нам нужно добавить игрока – главного объекта данной игры, который будет управлять картами – наш игрок должен обладать четырьмя параметрами:

* Иметь свою стопку карт, которой он может управлять
* Иметь собственное имя
* Иметь собственный uuid т.е уникальный идентификатор, отличающий его от других
* Иметь способность взаимодействовать с колодой игры, а также с другими игроками.

К первому и четвертому мы перейдем позже, когда создадим колоду карт, а 2 и 3, хоть это и необязательно, мы можем сделать сейчас.

Создадим класс Player:

|  |
| --- |
| **package** com.caffidev.unoone;  **public** **class** **Player** {  **private** **final** String name;    **public** **Player**(String name) {  **this**.name = name;  }  } |

Теперь наш игрок может иметь имя, но как сделать UUID?  
К счастью, Java имеет для этого встроенный модуль.  
Мы могли поместить все нужные функции в класс игрока, но уникальные id может иметь не только игрок, но и что-либо другое todo  
Поэтому было бы удобнее создать абстрактный класс, который бы брал на себя эту логику, а игрок бы его расширял.

Создадим пакет abstracts (папка), а в нем абстрактный класс Entity

|  |
| --- |
| **package** com.caffidev.unoone.abstracts;  **import** **java.util.UUID**;  **public** **abstract** **class** **Entity** {  **private** UUID uuid;  **public** **Entity**() {**this** (UUID.randomUUID());}  **public** **Entity**(UUID uuid){  uuid = uuid;  }    **public** UUID **getUuid**() {  **return** uuid;  }  } |

Теперь мы можем добавить в основной класс:

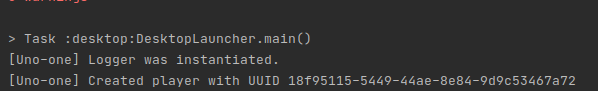
Player player = **new** Player("Кеша");

,наследовать класс Entity в классе Player, и запустить. Все запускается без ошибок!  
Но на самом деле это не так. Мы допустили критическую ошибку, т.к в конструкторе мы присваиваем переменную саму на себя. Впоследствии, когда мы захотим получить UUID, он будет равен null, и придется отслеживать по stacktrace ошибку.  
Для того, чтобы это исправить, достаточно добавить this перед uuid, которому мы присваиваем значение, а также хорошо было бы добавить модификатор final к UUID, чтобы объявление переменной было обязательным, а сама переменная не могла быть переопределена.

## Логгинг

Искать ошибку в коде с помощью точек остановки очень долго и непрактично. Также мы можем упустить некоторые ошибки, как было показано в предыдущей главе. Именно поэтому, стоило бы добавить систему логирования важных событий в консоль, чтобы нам, как разработчикам, было проще понять в чем заключается ошибка.  
К счастью, для этого нам не надо создавать специальных классов, или обращаться к стандартным классам Java – libGDX имеет внутренний логгер, который мы можем настроить под свои нужды.  
Добавим в основной класс эти строчки:

|  |
| --- |
| **<..>**  **public** **class** **Game** **extends** ApplicationAdapter {  **public** **static** **final** Logger logger = **new** Logger("Uno-one");  **protected** Framerate framerate;  **protected** SpriteBatch batch;  **protected** Texture background;  **public** **void** **create**(){  batch = **new** SpriteBatch();  framerate = **new** Framerate();  background = **new** Texture("background.jpg");    Gdx.app.setLogLevel(Application.LOG\_DEBUG);  logger.setLevel(Logger.DEBUG);  logger.debug("Logger was instantiated.");    Player player = **new** Player("Кеша");  *logger.debug("Created player with UUID "+ player.getUuid().toString());*  }    **<..>** |

получилось  


## Архитектура игры

Сейчас мы изучали только азы, чтобы ознакомиться с элементарным функционалом данного движка. Настало самое время продумать архитектуру игры. Существуют много разработчиков, которые начинают кодить, прежде чем продумывать идею проекта. Я считаю, что это неправильный подход. Программировать на самом деле очень легко, а вот подобрать нужный инструмент, нужную архитектуру под конкретную задачу куда сложнее.  
  
Есть много вариаций архитектур, паттернов для приложений, которые позволяют нам развиваться, и нам, как разработчикам, очень важно подобрать нужную под наши задачи.  
В данном проекте я планирую сделать игру Uno, которая будет доступна для игры на одном компьютере (hotseat) от 2 до 4 человек (можно сделать больше, если у Вас есть желание и идеи) , и которая является прототипом, который я не планирую расширять, или же масштабировать.  
Тогда нам не нужно разбивать все на микросервисы, и простой ООП монолитной архитектуры без MVC и прочих шаблонов, с Builder’ом игровой сессии будет достаточно.  
Про другие архитектуры, вы можете todo:

У нас уже есть класс игрока, и таких классов планируется несколько, т.к каждый игрок представляет собой объект, играющий в игру (CardGame) с другими игроками. У каждого игрока есть своя стопка карт (HandCard), которая ему нужна для выполнения различных действий (Card – как общий объект). Игрок может взаимодействовать с другими игроками, может менять правила игры, управляя директором игроков – т.е заставлять игрока пропускать ход, или меняя направление хода (то есть в игре должен быть показатель движения - GameDirection).  
Игрок способен также взять из колоды (CardDeck) карту в нужный момент, а также знать некоторую, но ограниченную информацию о других игроках.  
Главный класс Game при этом всем будет визуализатором сервиса, хостящего всю игру), отрисовывая объекты и разрешая производить определенные действия со стороны игрока.

Карты могут отличаться по цвету (CardColor), типу (CardType). а также иметь цифру (от -1 до 9, где – 1 – карта действия).

После вот такого продумывания, приходит большое количество идей о том, как можно сделать проект, как надо его строить, что очень сильно облегчает разработку т.к ты уже знаешь с чего можно начать. Также не стоит забывать что Java ООП язык, и чрезмерная зависимость от ООП иногда очень сильно вредит проектам, создавая циркулярные зависимости, некоторые проекты пишутся гораздо проще на нем, чем при использовании функционального программирования.

## Перечисляемые

Для начала, нам стоит создать нужные нам типы, чтобы за счет них потом строить объекты, которыми будут управлять пользователи.

Создадим сначала отдельный пакет под названием enums, где мы и будем хранить наши перечисления.  
Создадим класс CardColor, где будут перечислены все возможные цвета в Uno. Их четыре, если не считать бесцветных карт, значение цвета которых мы будем брать за null;

|  |
| --- |
| **package** com.caffidev.unoone.enums;  /\*\* Usual cards can only have 4 colors + wild, but we don't need it. \*/  **public** **enum** CardColor {  RED,  GREEN,  BLUE,  YELLOW,  } |

Дальше нам однозначно пригодятся перечисляемые направления и типа карты:

|  |
| --- |
| **package** com.caffidev.unoone.enums;  ///The direction, where players are going to go.  **public** **enum** GameDirection {  CLOCKWISE,  COUNTER\_CLOCKWISE  } |

|  |
| --- |
| **package** com.caffidev.unoone.enums;  /\*\* A card type enumeration \*/  **public** **enum** CardType {  NUMBER, //Usual card type  SKIP,  REVERSE,  PLUS\_TWO,  WILD\_PLUS\_FOUR, //Also it's a wildcard (can be played anywhere)  WILD\_COLOR  } |

## Карта

Благодаря всем этим параметрам, мы можем создать абстрактный объект карты, который будет делится на 3 под типа: карта с номером, карта действия и карта неожиданности.  
Поэтому давайте создадим в папке abstracts еще один абстрактный класс:

|  |
| --- |
| **package** com.caffidev.unoone.abstracts;  **import** **com.badlogic.gdx.Gdx**;  **import** **com.badlogic.gdx.graphics.Texture**;  **import** **com.badlogic.gdx.graphics.g2d.TextureRegion**;  **import** **com.badlogic.gdx.scenes.scene2d.ui.ImageButton**;  **import** **com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable**;  **import** **com.caffidev.unoone.Game**;  **import** **com.caffidev.unoone.enums.CardColor**;  **import** **com.caffidev.unoone.enums.CardType**;  /\*\* A card model, that has its own Image \*/  **public** **abstract** **class** **Card** **extends** ImageButton {  **private** **final** CardType cardType;  **private** **final** CardColor cardColor;  **private** **final** Integer cardNumber; //if it's unique, has -1 number;  **public** **Card**(CardType type, CardColor color, Integer number){  **super**(**new** TextureRegionDrawable());  cardType = type;  cardColor = color;  cardNumber = number;  }  **public** CardType **getCardType**() {**return** cardType;}  **public** CardColor **getCardColor**() {**return** cardColor;}  **public** Integer **getCardNumber**() {**return** cardNumber;}    } |

Todo: объяснения

Эта карта наследует класс …  
Мы уже импортировали эти текстурки в прошлых главах, поэтому мы можем написать статический метод, который позволит импортировать нужную текстурку в зависимости от цвета, номера и типа:

|  |
| --- |
| **private** **static** String **getTexturePath**(CardType type, CardColor color, Integer number) {  // Wildcards  **if** (type == CardType.WILD\_COLOR) **return** "black\_wildcard.png";  **else** **if** (type == CardType.WILD\_PLUS\_FOUR) **return** "black\_+4.png";  String colorStr = color.toString().toLowerCase();  **switch** (type) {  **case** **PLUS\_TWO:**  **return** colorStr + "\_+2.png";  **case** **REVERSE:**  **return** colorStr + "\_reverse.png";  **case** **SKIP:**  **return** colorStr + "\_skip.png";  **case** **NUMBER:**  **return** colorStr + "\_" + number + ".png";  **default**:  Game.logger.error("Could not load assets. App will exit now.");  Gdx.app.exit();  **return** **null**; |

Осталось дело за малым: сделать так, чтобы при инициализации карта использовала эти текстурки.  
Это сделать очень просто: достаточно вызвать конструктор ImageButton, и передать ему текстурку.

|  |
| --- |
| **public** **Card**(CardType type, CardColor color, Integer number){  **super**(**new** TextureRegionDrawable(**new** TextureRegion(**new** Texture(getTexturePath(type,color,number)))));  getStyle().imageDisabled = **new** TextureRegionDrawable(**new** TextureRegion(**new** Texture("back.png")));  cardType = type;  cardColor = color;  cardNumber = number;  } |

Все! Теперь при создании класса Card будет автоматически подбирать нужную текстурку с ассетов.

### Вариации карт

Теперь пришло время разделить эту карту на подгруппы. Создадим отдельный пакет cards, в котором будут лежать:

1. Карта с номером (NumberCard):

|  |
| --- |
| **package** com.caffidev.unoone.cards;  **import** **com.caffidev.unoone.abstracts.Card**;  **import** **com.caffidev.unoone.enums.CardColor**;  **import** **com.caffidev.unoone.enums.CardType**;  **public** **class** **NumberCard** **extends** Card {  **public** **NumberCard**(**int** number, CardColor color){  **super**(CardType.NUMBER, color, number);  }  **public** String **toString**(){  **return** **this**.getCardColor().toString().toLowerCase()  + " number " + **this**.getCardNumber()  + " card";  }  } |

2. Карта с действием (ActionCard):

|  |
| --- |
| **package** com.caffidev.unoone.cards;  **import** **com.caffidev.unoone.abstracts.Card**;  **import** **com.caffidev.unoone.enums.CardColor**;  **import** **com.caffidev.unoone.enums.CardType**;  **public** **class** **ActionCard** **extends** Card {  **public** **ActionCard**(CardType type, CardColor color){  **super**(type,color,-**1**);  }  **public** String **toString**(){  **return** **this**.getCardColor().toString().toLowerCase()  + " " + **this**.getCardType().toString().toLowerCase()  + " card";  }  } |

3. Карта неожиданности (WildCard):

|  |
| --- |
| **package** com.caffidev.unoone.cards;  **import** **com.caffidev.unoone.abstracts.Card**;  **import** **com.caffidev.unoone.enums.CardType**;  **public** **class** **WildCard** **extends** Card {  **public** **WildCard**(CardType type){  **super**(type, **null**, -**1**);  }  **public** String **toString**() {  **return** **this**.getCardType().toString().toLowerCase() + " card";  }  } |

Todo: объяснить в чем прикол

## Создание доски (obsolete)

Сменить с доски, на другие объекты, доступные там.

Java – это объектно-ориентированный язык, и несмотря на то, что мне некоторые парадигмы этого языка не подходят для создания массивных продуктов, я считаю что для подобных игр ООП подходит в самый раз.

Для начала, нам нужно создать доску, на которой будет происходить игра.  
Мы будем создавать игру, которая работает по механике HotSeat, т.е когда несколько игроков играет на одном компьютере. Если Вы хотите, Вы можете по желанию дополнить ее одиночной игрой с ботами, или по интернету, но у меня такой возможности нет.

Выделим объект доски в отдельный класс. Кликните по папке проекта, и создадите класс под названием Board.  
Напишите следующий код:

|  |
| --- |
| **package** com.caffi.training;  **import** **com.badlogic.gdx.Gdx**;  **import** **java.util.ArrayList**;  **public** **class** **Board** {  Program main;  **protected** Player currentPlayer;  **public** **Board**(Program main){  **this**.main = main;  players.add(**new** Player(main));  players.add(**new** Player(main));  players.add(**new** Player(main));  players.add(**new** Player(main));  setCurrentPlayer(players.get(**0**));  }  **public** **void** **putOnBoard**(Card card){  cards.add(card);  **float** offsetX = (**float**) ((Math.random() \* offsetRange \* **2**) - offsetRange);  **float** offsetY = (**float**) ((Math.random() \* offsetRange \* **2**) - offsetRange);  card.setPosition(Gdx.graphics.getWidth() /**2**f - card.getPrefWidth() /**2** + offsetX, Gdx.graphics.getHeight() /**2**f - card.getPrefHeight() /**2** + offsetY);  main.stage.addActor(card);  }  } |

Статья про типы поддерживаемых карт

Illegal reflective access; fix problem

Все ассеты взяты с <https://commons.wikimedia.org/wiki/File:UNO_cards_deck.svg?useLang=ru>

Card deck, implementions of card interface?

Card matches, read architectures?Services?  
Что можно сделать:

Мультиплеер

Главное меню

Смена фона

SpriteBatch = framerate?

Game -> GameService (отвечает за игру, помогает масштабироваться, так и написать) -> GameBuilder, помогает строить игру (также поможет при создании главного меню) -> при создании игры, создает колоду, перемешивает карты  
создает ее с помощью переменных  
-> CardGame – экземпляр игры

Основные ссылки:

Глава 1:

<https://libgdx.com/dev/setup> - подробная инструкция по установке libgdx (на английском)

<https://github.com/caffidevjava/uno-school> - GIT репозиторий с файлами

<https://metanit.com/java/tutorial/> - Подробное изучение Java  
<https://developer.android.com/studio> - Android Studio.

<https://adoptopenjdk.net/releases.html> - AdoptOpenJdk от Adoptium

<https://libericajdk.ru/pages/downloads/> - Liberica JDK от Bellsoft

<https://java-lessons.ru/common-errors/javac-is-not-recognized> - переменная среды  
<http://www.libgdx.ru/p/guide.html> - Гайд по Libgdx на русском  
<https://github.com/libgdx/libgdx/wiki/> - Документация по Libgdx на английском  
<https://suvitruf.ru/2012/10/20/2274/libgdx-chast-1-zhiznennyj-cikl-igry/> - немного отсюда

Косяк в гит репозитории: не наследуется Entity

1. <http://www.libgdx.ru/p/guide.html> [↑](#footnote-ref-1)
2. <https://github.com/libgdx/libgdx/wiki/> [↑](#footnote-ref-2)
3. <https://metanit.com/java/tutorial/> [↑](#footnote-ref-3)
4. <https://developer.android.com/studio> [↑](#footnote-ref-4)
5. <https://adoptopenjdk.net/releases.html> [↑](#footnote-ref-5)
6. <https://libericajdk.ru/pages/downloads/> [↑](#footnote-ref-6)
7. <https://java-lessons.ru/common-errors/javac-is-not-recognized> [↑](#footnote-ref-7)
8. <https://libgdx.com/assets/downloads/legacy_setup/gdx-setup_latest.jar> [↑](#footnote-ref-8)
9. <https://github.com/libgdx/libgdx/wiki/> [↑](#footnote-ref-9)