1 LazyJamGDX Manual

Hello! This Document is meant to describe Version 1 of the LazyJam GDX. LazyJam is designed to be a library for Game-Jams.

2 Introduction

LazyJam started out as a library for Slick2D, but later was ported to libgdx instead, because it offers more and nice functionality and didnt have the same limits as Slick2D.
What LazyJam can do for you:

- An easy to use entity component system with little reading effort and compatibility to most existing software
- An Abstraction for Graphical Assets allowing you to only use text assets until you are ready to exchange assets, allowing to code before any assets are done
- A TiledMap to GameState/SceneGraph conversion, that is quite powerful, if you happen to have a lot of game designers and little coeders

The Drawbacks are

- It uses Java 8, therefore the android (Maybe also WebGL) compatibility is lost
- Perfomance was no issue whatsover, it can take a buller hell shooter, but very complex things, might run into perfomance problems
- currently, only 2D without any Z Sorting is avaiable

3 Starters Guide

First off, you should extend the *LazyJamApplicationAdaptor* class. The initAsset method you will need to implement is the place, where you can use the libgdx assetmaanager and get the assets. In the Create Method you should have some kind of *GameState* inited, and either use the *TiledMapProvider* or add everything Manually. The rest is Done by the Components

3.1 Context

What i wanted to try out with this Software was the aspect of "some method in some object, but the parameters are any and avaiable". The Context Objects are objects that hold references to any *Type* of Object you need. Most of the Annotations found in LazyJam will result in that method called with a Context from where all used Parameters are gotten.

3.2 GameObjects and Components

The GameObjects are when instantiated registered at the GameState, the GameObjects are nothing but a container for position and components. The Position is a regular *Vector2* and the components are a List of Objects. Any Object you add to the GameObject can have methods with several Annotations.

- *@Update* This marks methods to be called for logics every tick (default are 60 ticks per second can be changed in the *AGameState*
- *@Render* This marks drawing methods (also tick based)
- *@Collide* needs a ExtraSimpleCollisionComponent (Or something more sophisticated at both game Objects, will be called on Collision
- *@Destrory* gets called when the gameobject is destructed

So only thing left is, how do i get the Data i need for my Update method? just add it as a parameter, if it is known in the Context.

3.3 Services

A "Service" is what i prefer to call a singleton, that is not a singleton anymore. Service Classes, can be registered at the *ServiceManager* manually or by adding a @Service Annoation to the type. It will be instantiated on start then. It is avaiable in the Global-Context. Services can have other Services as Dependency, add *@InjectedService* to the Field, then the serviceManager will set the Reference. Be warned, the ServiceManager is not able to remove Services. The GameStateService can be deleted and instantiated again, but you can never remove a single Service!

4 Ending

Keep in mind that you have to adjust the *ReflectionUtil* Class before deploying a fat jar. Thanks to the Guys working on IGGJ7_Laika and B. Koppelmann, who added more things to LazyJam. If you do anything to this library please let me know, and if you have wished, leave them in my github account. For Reference you can take a lkook at Laika and see how lazyjam is used there.