Developer Guidelines

Overview:

1. Installation

1.1.1 Cloning down repositories

First you clone down the repository named CatchAndroid to some location on your hard drive. Navigate to CatchAndroid/jni and in there you clone down the CatchLib repository (resulting in CatchAndroid/jni/CatchLib).

1.1.2 Build native code with Android NDK

If you don't already have Android NDK you should go to developer.android.com and download it. Then you navigate to CatchAndroid and run ndk-build. This should compile the native code of the project (all C/C++ files inside the jni folder).

1.1.3 Build java code with Eclipse

Now it's time to build and run the project with Eclipse (you must have the ADT plugin for Eclipse to develop for Android).

1.1.4 Run on device

The application has to run on a real device or an emulator that is at least api version 15 in order for the rendering to work.

1.2.1 Cloning down repositories

First you need to clone down the repository named CatchiOS to some location on your harddrive. Inside the project folder (CatchiOS) you clone down the CatchiOS repository (resulting in CatchiOS/CatchLib).

1.2.2 Build and run on device or in iOS simulator

Now you just open the project in XCode and compile and run on either a real iOS device or in the iOS simulator.

2. System overview

CatchLib

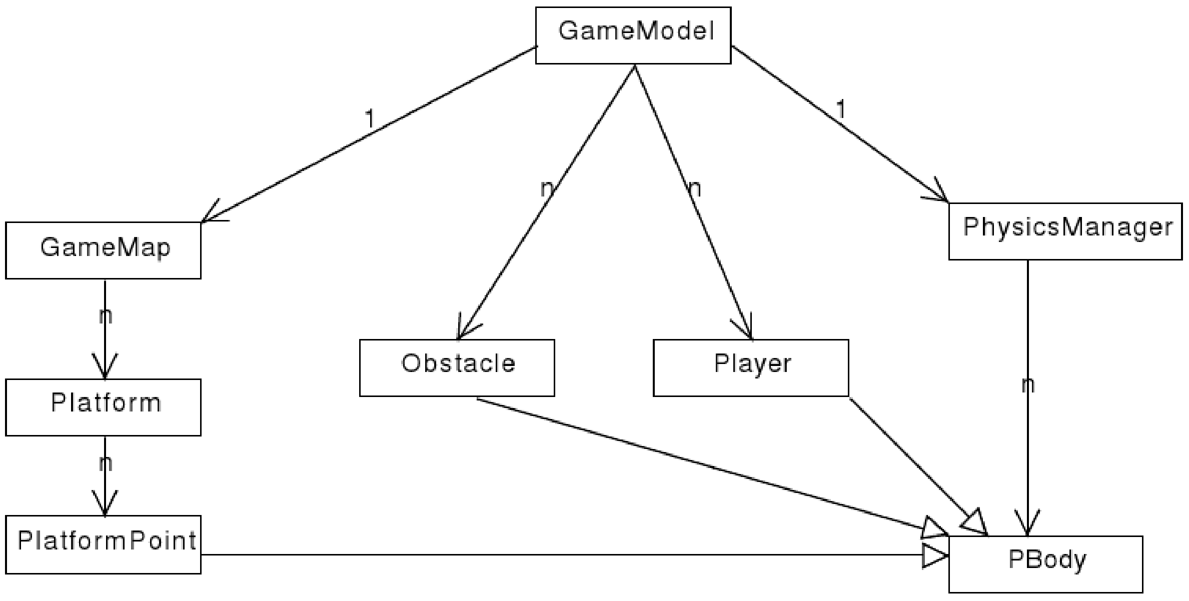
The game is based on a C/C++ core which is all located in the CatchLib repository. This is where all platform independent code should be. There are some exceptions to this at this point (in the GLRenderer class) since Android and iOS have some very minor differences in their implementation of OpenGL. This is worked around by letting the preprocessor checking if __APPLE__ is defined, if so use the iOS specific code, otherwise the Android specific counterpart. This could potentially become a problem if one were to port this application for other operating systems such as Windows Phone, Mac, PC, Linux etc. This will surely be reviewed.

CatchAndroid and CatchiOS

This is where all the system specific code is located. At this point that code is initialization code for OpenGL, handling touch-events and passing those on to the C++ core and creating an instance of the C++ core. It will also be responsible for telling the core where resources (images, sound etc.) are located as well as where it can save data (saving current gamestate and highscores).

3. Design model

A rough idea of what the design model will look like. The highest abstraction on this model is the GameModel which is “owned” by the GameController. It communicates with the rest of the system via an EventBus which will be explained in more detail later.



4. Event bus

The event bus is used to bind the Model, View and Controller together. It’s a singleton with global access that anyone can register to listen to. The Event bus will represent the application neural network which binds all modules together.

A more exact model of this will be included at a later stage.

5. Rendering engine

Since this is a fairly simple game the rendering will be based on rendering the animations frame by frame (as opposed to a skeletal animation system for instance). All frames for all animations should be stored in the same image file (called spritesheet) and data regarding what frame of which animation is located where should if possible be stored in some external file rather than hardcoded. The rendering engine will work with “actors” which all represent an object from the model and are responsible for the visual representation of that object. The communication between GameModel and the rendering engine will all go through the EventBus.

6. Physics

The physics system is built on one physical body, which is used to describe a body in space. It has some properties such as if it’s affected by gravity, if it’s stationary and movement vectors. Its position is represented by vectors (points) which correspond to every corner of the body and they are all affected by the movement vector when the body’s state is updated . This allows for collision detection with the single axis theorem which in turn gives easy support for non rectangular shapes and rotation.

7. Release

To create a release of LumberJack Jack you’ll need:
- GNU Make installed
- Apache Ant (or simply Ant) installed

The first time you’ll need to run the following command:

```
$ android update project --path .
```

Then all you have to do is run these commands every time you want to build a version of the application:

```
$ make debug
```

The runnable APK will be located in the bin/ directory with the name *GameActivity-debug.apk*