DIGITALJUNKY

A journey through information technologies

SHOW MENU

Make a Snake game for Android written in Python – Part 1

FEBRUARY 2, 2015 / 0 COMMENTS

I wanted to code my first android app for a long time but I didn't really know where to start. I wasn't really excited at the idea of having to learn Java and was worried that my current level of C++ wouldn't be enough. So I began to look for a way to code the app in my go-to language : Python. Using Python allows for reuse of the native libraries, which can come in very handy. And it obviously comes with the usual Python magic and agility.

In the following tutorial, I will explain how to code and deploy a simple app on Android : the classic snake game that we all played on the famous and indestructible nokia 3210 (well, my version of it anyway). I personally code on Ubuntu 14.10 with Python 2.7, so this tutorial assumes a similar set-up on your side.

Table of contents

Part 1 – Setting up the environment.

Part 2 – The game engine.

Part 3 – Welcome screen, options and packaging.

The Kivy framework

After a few research on stack overflow, I discovered Kivy, an open source Python library made for cross platform development, just what I needed. The fact that they are a young open source project with a great documentation and community convinced me to give it a go.

As for the performances concerns that might arise at the idea of using Python to build an app, Kivy answers by using Cython so that the most intensive parts of the code are compiled down to the C level. Plus, it uses the GPU as much as possible for graphics-oriented calculations, freeing up CPU resources for other tasks.

There is a great deal of resources available to start with Kivy :

- The official documentation
- Kivy's Wiki, where you'll be able to find pretty much everything that wasn't covered in the doc (examples, code snippets, tutorials, talks, widgets developed by the community etc.)
- You can find an */example* folder on their repository. Try to run them all at least once and you'll see that most situations you will encounter are covered there (I especially like the demo showcase. It became as useful to me as the bootstrap doc is for web dev tasks).

Well that's that. Let's get down to it shall we ?

Setting up the environment

We will see how to set up SublimeText, add the corresponding packages and syntax highlighting so that it'll be tailored to our needs. If you are more comfortable using another editor, just skip this part.

The mains things we will need are :

Sublime Text 3

Package Control : a package manager that will allow us to easily add functionalities to our editor

Anaconda : an awesome Python package adding auto-completion, code linting (super useful to easily format the code so that it follows PEP8 standard) and more.

Everything you need in order to do that is explained in this great tutorial : setting up sublime text 3 for full stack python development.

One more thing : kivy uses its own templating language in order to separate the front-end logic from the back-end. Its syntax is not natively recognized by ST3 so we need to add a custom syntax highlight file.

The procedure is quite simple :

Download to following syntax definition : kivy.tmLanguage

In ST3, click on Preferences/Browse Packages…

There should be a */User* folder in the directory. Copy your .tmLanguage file here, restart ST3 and you'll be all set!

Intalling Kivy

Dependencies

To run properly, kivy needs 3 main dependencies : Cython, pygame and the python-dev package. If you're using Ubuntu, you may also need the gstreamer library which is used to support some of the video capacities of the framework.

```
# install cython package
sudo pip install cython

# install dependencies for pygame
sudo apt-get build-dep python-pygame
sudo apt-get install python-dev build-essential

# install pygame (you need mercurial to use the hg+ command)
sudo pip install hg+http://bitbucket.org/pygame/pygame

# install gstreamer
sudo apt-get install gstreamer1.0-libav
```

Kivy

```
# add kivy's repo to the list iof usable repositories
sudo add-apt-repository ppa:kivy-team/kivy
sudo apt-get update

# install kivy
sudo apt-get install python-kivy
```

Buildozer

We're going to use it to package our app and deploy it on an Android smartphone. Just plug your phone, type the magic words in the console and that's it, you app is installed! Pretty neat.

```
sudo pip install buildozer
```

To quote their readme :

> Buildozer is a tool for creating application packages easily.
>
> The goal is to have one "buildozer.spec" file in your app directory, describing your application requirements and settings such as title, icon, included modules etc. Buildozer will use that spec to create a package for Android, iOS, Windows, OSX and/or Linux.

Note : you will also need a java jdk. If you don't have one installed, openjdk7 should be good. Furthermore if you're on a 64bit system, you'll need the 32bit versions of your dependencies (I got the tip from there).

```
# install java jdk
sudo apt-get install openjdk-7-jdk

# install 32bit dependencies
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libncurses5:i386 libstdc++6:i386 zlib1g:i386
```

Is it working ?

Before we start to make our snake, let's take a few minutes to check if everything is working as it is supposed to. Nothing is more annoying than discovering a new technology, jumping right into the code only to find several hours later that it can't compile because of some damned dependency.

To prevent that from happening, we will build and deploy the illustrious 'Hello World'. I won't explain much since I would only be poorly paraphrasing their wiki, so if you need a more basic understanding I can only direct you there.

First let's initiate buildozer in our directory from the terminal :

```
buildozer init
```

Edit the buildozer .spec file (only the lines written here):

```
# (str) Title of your application
title = Hello World

# (str) Package name
package.name = helloworldapp

# (str) Package domain (needed for android/ios packaging)
package.domain = org.helloworldapp

# comment these two
# (str) Application versioning (method 1)
# version.regex = __version__ = ['"](.*)['"]
# version.filename = %(source.dir)s/main.py

# uncomment that one
version = 1.0.0
```

Our 'Hello World' App :

```
import kivy
kivy.require('1.8.0')  # update with your current version

from kivy.app import App
from kivy.uix.button import Button

class DummyApp(App):

    def build(self):
        return Button(text="Hello World")

```

```
if __name__ == '__main__':
    DummyApp().run()
```

The code is available on the repository. Each successive commit will correspond to a stage of progress in our tutorial.

We're now ready to build the app and deploy it. Back to the terminal :

```
buildozer android debug # will create the apk file in a ./bin folder

buildozer android debug deploy # if you want to install the apk directly on your phone
```

Yay, your first app on a smartphone! How does it feel ?
We're finally ready to start our Snake.

Go to part 2.

("Nay, it didn't work." Feel free to ask about any issue in the comments, but be warned that my knowledge is directly proportional to my ability to phrase questions on stackoverflow ^^).

Note : buildozer and adb sometimes make a great couple…sometimes they don't. Don't ask me why. So if buildozer gets stuck on `# Deploy on adb`, use adb directly : go in the /bin folder of your app where the apk file was compiled and use `adb install` *YourApp*`.apk`. If it still doesn't work (adb doesn't recognize your device, or your device is unauthorized) : `adb kill-server`. Then make sure you restart it as root : `sudo adb start-server`. And if it still doesn't work…tough life eh!

Share this:

Facebook Google Twitter LinkedIn Pinterest Reddit Email

Categories: Programming Tags: android, kivy, Python, tutorial

« And you, what's your organization system ?

Make a Snake game for Android written in Python – Part 2 »

Leave a Reply