

- **Installation de Kivy**

Nous avons installé Kivy sur nos ordinateurs Mac. Pour cela, il faut :

1. Télécharger Cython et Pygame. Sur le terminal, nous lançons ***pip install cython*** et ***pip install kivy***
2. Télécharger Kivy sur son site officiel suivant son système d'exploitation [1]

Parfois, certaines commandes ne fonctionnent pas sur le terminal, nous avons alors ajouté "sudo" au début de la commande pour avoir les droits administratifs.

Pour utiliser Kivy, nous avons soit un fichier .py, soit un fichier .py et un ou plusieurs fichiers.kv. Le code .py est le code qu'il faut exécuter (fonctions, ...). Il faut dans le code Python (fichier .py), ajouter au début du programme : ***from kivy.app import App*** . Le fichier .kv est le code qui permet de créer le design de l'application. Dans ce fichier, certains mots sont entre < > et correspondent à un nom d'une classe. Soit cette classe est déclarée dans le code .py, soit cette classe existe déjà dans Kivy. Lorsque la classe existe sur Kivy (voir documentation Kivy [2]) il n'y a pas besoin de l'importer car elle est incorporée dans Kivy et elle comporte des paramètres par défaut. Lorsque nous implémentons une classe de Kivy dans le code .kv, cela nous permet de modifier ses paramètres pour personnaliser l'application. Par exemple, pour la classe ***Label*** [3] l'écriture par défaut est en noire et de taille 15sp mais nous pouvons décider qu'elle soit blanche et de taille 18dp en écrivant dans le .kv :

***<Label>***

***Color : FFFFFFFF***

***font\_size: 18dp***

- **Comment exécuter un programme Kivy sur ordinateur ?**

Lorsque nous voulons exécuter un programme Kivy sur l'ordinateur, nous pouvons soit ouvrir le fichier ***main.py*** avec Kivy, soit nous tapons la commande suivante sur le terminal (en se plaçant au préalable dans le dossier contenant le fichier ***main.py*** et le ou les fichiers .kv) : ***python main.py*** .

Il y a une spécificité sur Windows : Le système Windows ne reconnaît pas les accents dans les fichiers Kivy, donc il affiche mal le texte. Dans ce cas là, il faut aller dans le dossier "lang" à l'adresse [voir Figure III.2.b - 1] (chaque ordinateur peut avoir une adresse différente, il faut chercher dans le dossier kivy) :

> Users > lenovo > AppData > Local > Programs > Python > Python37 > Lib > site-packages > kivy > lang >

Figure III.2.b - 1 : Capture de l'adresse du dossier "lang"

Dans ce dossier, nous pouvons trouver le fichier ***builder.py*** : [voir Figure III.2.b - 2]




 <code>_init_</code>	2019/5/16 16:43	Python File
 <code>builder</code>	2019/6/20 12:08	Python File
 <code>parser</code>	2019/5/16 16:43	Python File

Figure III.2.b - 2 : Contenu du dossier “lang”

Nous avons ensuite modifié la ligne 288 du fichier **builder.py** en ajoutant **encoding='utf-8'** [voir Figure III.2.b - 3] :

```
trace('Lang: load file %s' % filename)
with open(filename, 'r', encoding='utf-8') as fd:
    bwaes['filename'] = filename
```

Figure III.2.b - 3 : Code du fichier builder.py

- **Comment créer un package pour iOS ?**

Pour créer un package iOS d'une application avec Kivy, il faut télécharger le logiciel Xcode [4]. Il est conseillé d'installer Homebrew [5] afin d'avoir les prérequis nécessaires. Une fois installé, il faut taper sur un terminal les commandes suivantes [6] :

**brew install autoconf automake libtool pkg-config**

**brew link libtool** → Cette commande permet de vérifier si l'ordinateur est bien connecté à libtool

**sudo easy\_install pip** → Nous pouvons négliger cette commande, mais il faut vérifier avant la version de python (pip) en utilisant la commande : **which pip**

**sudo pip install Cython**

Une fois que ces prérequis sont installés, il faut installer Kivy-iOS en tapant sur le terminal la commande suivante : **git clone git://github.com/kivy/kivy-ios** . Après l'installation de Xcode et Kivy-iOS, il faut entrer dans le dossier Kivy-iOS : **cd kivy-ios**. Pour la première utilisation, il faut taper : **./toolchain.py build python3 kivy** . Attention, il faut d'abord vérifier la version de Python en tapant **python --v** et si la version de Python est 2 il faut écrire “2” à la place de “3”. [6] S'il y a encore des erreurs qui s'affichent, il faut vérifier que le fichier requirements.txt est bien installé [voir Figure III.2.b - 4] : **pip install requirements.txt**

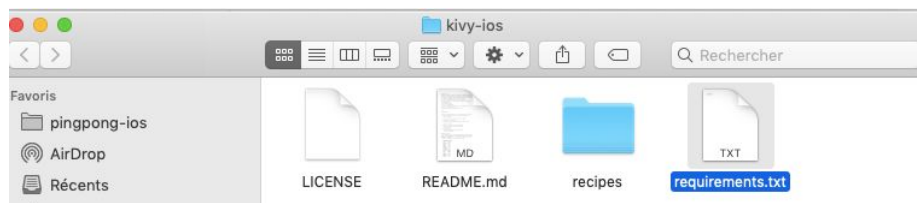


Figure III.2.b - 4 : Fichier requirement.txt dans le dossier kivy-ios

Pour créer la première application, il faut taper [6] :

**./toolchain.py <nom de fichier> <adresse de fichier>**

Par exemple : **./toolchain.py create came /Users/mac/Desktop/KIVY/camera**

Dans ce cas là, un projet Xcode est créé [voir Figure III.2.b - 5].

```

--
Project directory : came-ios
XCode project      : came-ios/came.xcodeproj
MacBook-Air-de-mac:kivy-ios mac$ open came-ios/came.xcodeproj

```

Figure III.2.b - 5 : Capture d'écran du terminal affichant que le projet Xcode est créé

Nous pouvons lire les informations sur le dossier came-ios [voir Figure III.2.b - 6]. Puis, il faut changer les permissions en mettant **“Lecture et écriture”** pour tout le monde. Si nous ne faisons pas cela, occasionnellement nous ne pouvons pas construire le package iOS avec Xcode. [7]

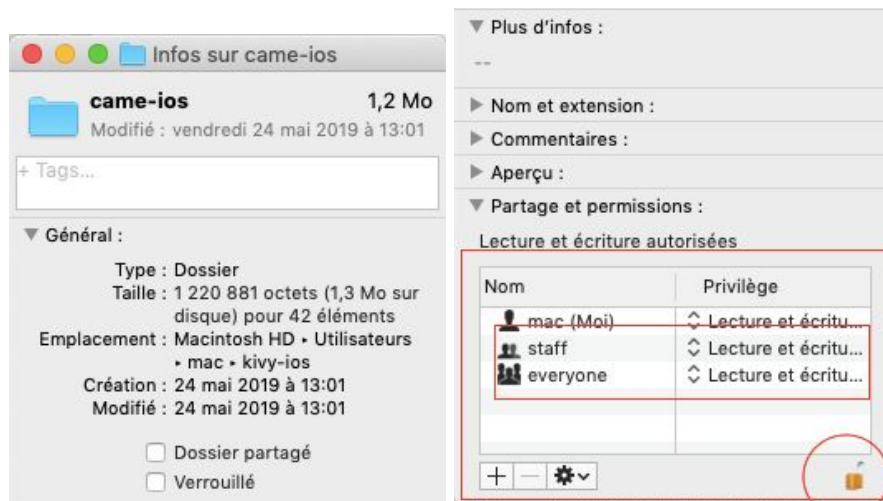


Figure III.2.b - 6 : Informations sur le projet came-ios

Puis, il faut taper sur le terminal : **open came-ios/came.xcodeproj** pour que le projet Xcode s'ouvre. [6] Il faut alors choisir grâce au menu déroulant le simulateur Iphone [voir Figure III.2.b - 7].

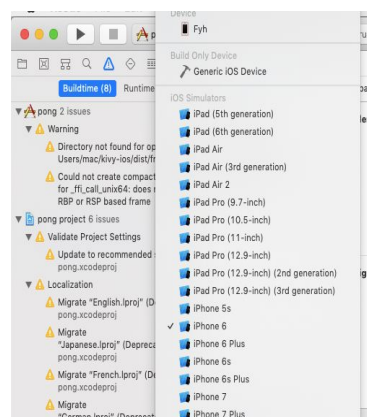


Figure III.2.b - 7 : Choix du simulateur

Pour finir, il faut cliquer sur le bouton en haut à gauche (encadré en rouge ci-dessous) [voir Figure III.2.b - 8] pour lancer le programme. S'il a fonctionné, le simulateur du portable s'affiche.

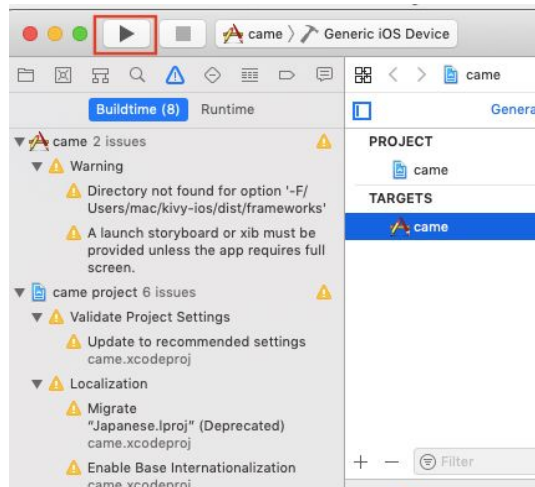


Figure III.2.b - 8 : Lancer un projet Xcode

- **Comment générer un apk pour Android ?**

Pour créer un package apk pour Android d'une application avec Kivy, il faut installer Buildozer. Buildozer prend en charge les packages pour Android via **python-for-android** et il prend aussi en charge les packages pour iOS via **kivy-ios**. Nous n'avons pas exploité Buildozer pour iOS. En effet, comme nous l'avons vu précédemment, nous sommes directement passées par Xcode. Pour Android, nous avons installé Buildozer tout en respectant la version de Python en tapant les commandes suivantes [8] :

***sudo pip install buildozer***

***sudo pip install https://github.com/kivy/buildozer/archive/master.zip***

***git clone https://github.com/kivy/buildozer***

***cd buildozer***

***python setup.py build***

***sudo pip install -e .***

Une fois Buildozer installé, il faut aller sur le terminal dans le dossier contenant notre **main.py** et **pong.kv** (dossier source) et lancer la commande **buildozer init** .[8] Cette commande permet la création d'un fichier qui se nomme **buildozer.spec**. Une fois ce fichier créé, comme nous avons Python 3 sur nos ordinateurs, il faut télécharger et extraire Crystax NDK [9] et le mettre dans le répertoire **.buildozer** . Il faut s'assurer que dans le fichier **buildozer.spec** [8] :

***requirements = python3crystax,kivy et android.ndk\_path = .buildozer/crystax-ndk-10.3.2*** (Pointe sur le dossier où nous avons extrait Crystax NDK [10]). Puis, nous tapons la commande **buildozer android debug play run** .[8]

Cependant, nous avons eu une erreur lorsque nous avons essayé de créer le package pour l'application Pong Game de Kivy (voir partie suivante).

Nous avons dû installer ou réinstaller un JDK [11] ainsi que Android Studio [12] pour pouvoir installer et gérer l'installation du SDK et API. De même pour installer les différents modules comme Android SDK Tools et Android SDK Platform-tools [voir Figure III.2.b - 9]. Pour cela, sur Android Studio, il faut cliquer sur "Configure" et puis après avoir cliqué sur "SDK manager" une page s'ouvre pour qu'on puisse installer les différents SDK et outils :

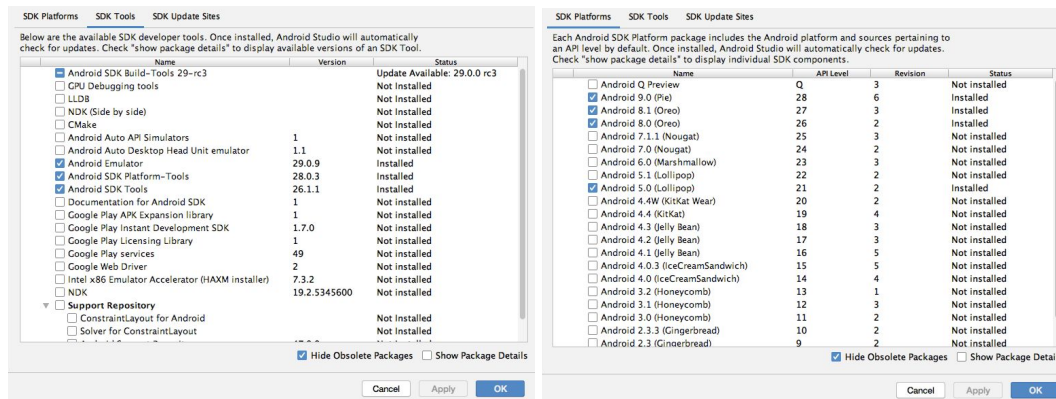


Figure III.2.b - 9 : SDK Tools sur Android Studio et SDK Platforms sur Android Studio

Nous avons essayé de refaire le package Android grâce à la commande **buildozer android debug play run** [8] Cependant, nous avons rencontré une autre erreur mais nous n'avons pas réussi à la résoudre [voir Figure III.2.b - 10] :

```
# PATH = '/Users/Catherine/.buildozer/android/platform/apache-ant-1.9.4/bin:/anaconda3/bin:/anaconda3/condabin:/anaconda/bin:/Library/Frameworks/Python.framework/Versions/3.4/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin'
# CONDA_PREFIX = '/anaconda3'
# PHD = '/Users/Catherine/Desktop/App'
# LANG = 'fr_FR.UTF-8'
# XPC_FLAGS = '0x0'
# _CE_M = ''
# XPC_SERVICE_NAME = '0'
# SHLVL = '1'
# HOME = '/Users/Catherine'
# CONDA_PYTHON_EXE = '/anaconda3/bin/python'
# LOGNAME = 'Emilie'
# CONDA_DEFAULT_ENV = 'base'
# = '/anaconda3/bin/buildozer'
# PACKAGES_PATH = '/Users/Catherine/.buildozer/android/packages'
# ANDROIDSDK = '/Users/Catherine/.buildozer/android/platform/android-sdk'
# ANDROIDNDK = '/Users/Catherine/.buildozer/android/platform/android-ndk-r17c'
# ANDROIDAPI = '27'
# ANDROIDMINAPI = '21'
#
# Buildozer failed to execute the last command
# The error might be hidden in the log above this error
# Please read the full log, and search for it before
# raising an issue with buildozer itself.
# In case of a bug report, please add a full log with log_level = 2
(base) MacBook-Air-de-Catherine:App Emilie$
```

Figure III.2.b - 10 : Les erreurs lors du lancement de Buildozer sur le terminal

Ainsi, nous avons décidé d'installer une machine virtuelle VirtualBox sur un ordinateur Windows [13] [14]. Ensuite, nous avons téléchargé Ubuntu en étant dans la machine virtuelle [15]. Puis, on lance un terminal dans lequel on tape les commandes vues précédemment [8] :

```
sudo pip install buildozer
sudo pip install https://github.com/kivy/buildozer/archive/master.zip
git clone https://github.com/kivy/buildozer
cd buildozer
python setup.py build
sudo pip install -e .
```

Nous n'avons pas installé Crystax sur la machine virtuelle parce que lorsque nous créons le fichier .apk grâce à la commande **buildozer android debug play run** [8] la machine virtuelle va télécharger automatiquement Apache ANT, Android SDK et Android NDK manquants. Il faut alors, sur le terminal, aller dans le dossier contenant notre **main.py** et **pong.kv** (dossier source) et lancer la commande **buildozer init** [8]. Cette commande permet la création d'un fichier qui se nomme **buildozer.spec** et dans lequel nous n'avons rien modifié car ce fichier est défini par défaut pour pouvoir créer une simple application (package .apk). Nous pouvons par la suite ajouter des permissions, changer le nom et



l'icône de l'application, spécifier le type de fichiers et les dossiers que nous incluons et que nous excluons dans le package .apk.

Une fois, ce fichier créé, nous lançons la commande ***buildozer android debug play run*** [Figure III.2.b - 11] [8] pour créer le package Android (.apk) qui se trouvera alors dans le dossier ***bin*** lui-même dans le dossier source.

```
kivyekivyvm:~/Desktop$ buildozer init
File buildozer.spec created, ready to customize!
kivyekivyvm:~/Desktop$ buildozer android debug deploy run
# Check configuration tokens
# Ensure build layout
# Check configuration tokens
# Preparing build
# Check requirements for android
# Install platform
# Apache ANT found at /home/kivy/.buildozer/android/platform/apache-ant-1.9.4
# Android SDK found at /home/kivy/.buildozer/android/platform/android-sdk-26
# Android NDK found at /home/kivy/.buildozer/android/platform/android-ndk-r9c
# Check application requirements
# Check garden requirements
# Compile platform
# Distribution already compiled, pass.
# Build the application #11
# Package the application
# Android packaging done!
# APK MyApplication-0.1-debug.apk available in the bin directory
# Application pushed.
# Application started.
kivyekivyvm:~/Desktop$
```

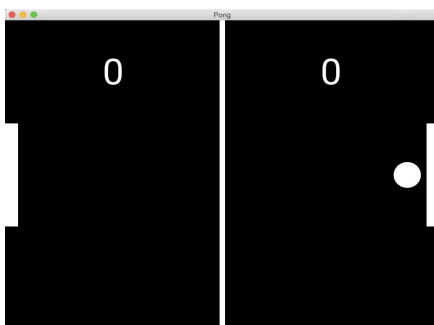
Figure III.2.b - 11: Création du package Android sur la machine virtuelle

- **Exemples de programme Kivy**  
--> **PONG GAME&HELLO WORLD**

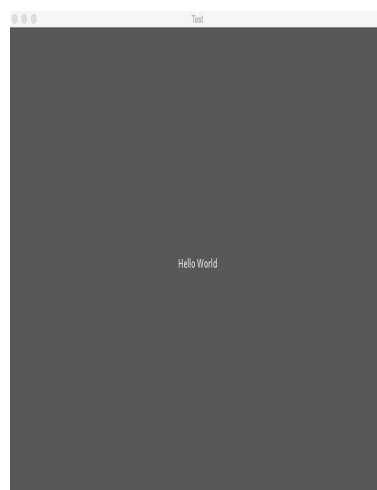
Nous avons voulu tout d'abord tester des codes déjà prêts pour pouvoir prendre en main le logiciel Kivy. Pour cela, nous avons téléchargé le Pong Game et Hello World sur le site officiel de Kivy [16], Pour Hello World, le programme ***main.py*** qui va permettre d'afficher "Hello World" et de créer un bouton. et le Pong game est constitué de deux fichiers : ***main.py*** et ***pong.kv***.

Voici les résultats que nous avons obtenus sur les différentes plateformes [voir Figure III.2.b - 13, Figure III.2.b - 14 et Figure III.2.b - 15] :

Pong Game



Affiche "Hello World"



Quand nous cliquons sur le bouton

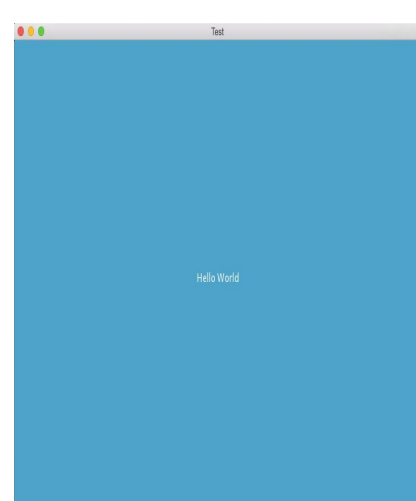


Figure III.2.b - 13 : Capture d'Écran du Pong Game et Hello World sur l'ordinateur MacOS X

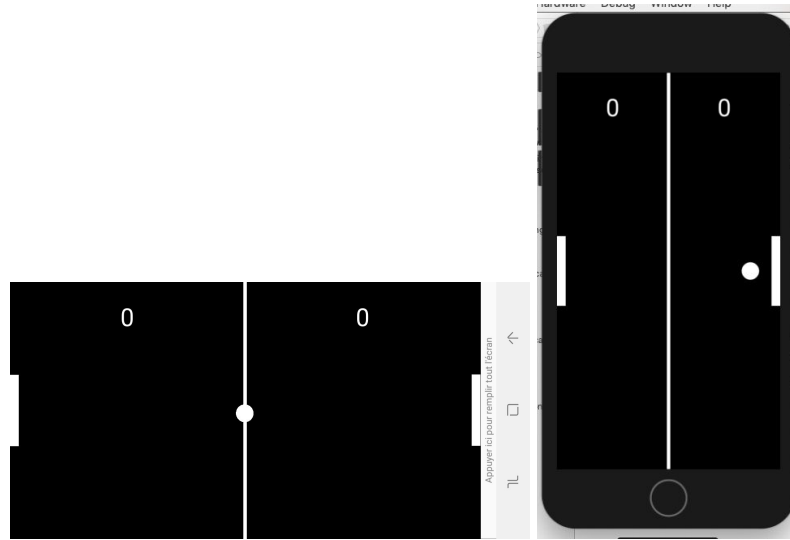


Figure III.2.b - 14 : Capture d'Écran du Pong Game sur Android (Version 8) et simulateur IOS

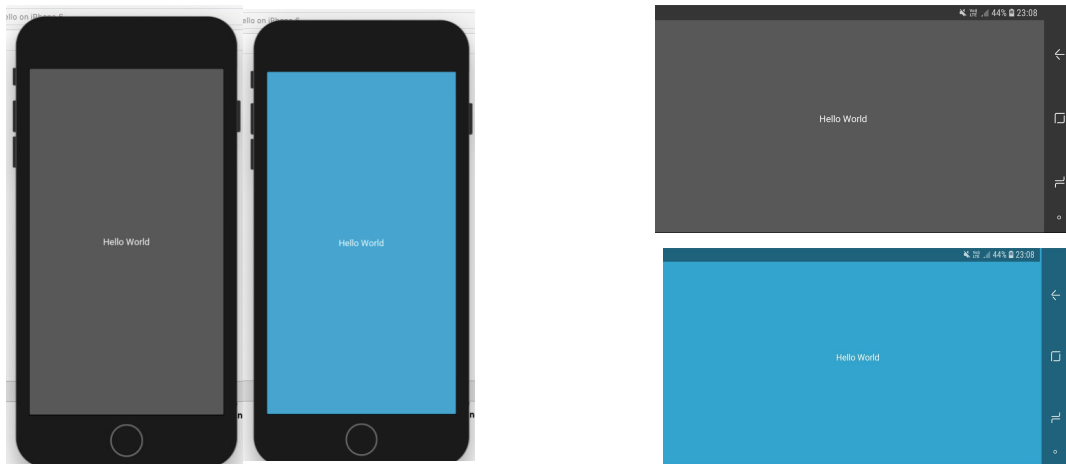


Figure III.2.b - 15 : Capture d'Écran de Hello World sur simulateur iOS et Android

- [1] : <https://kivy.org/#download>
- [2] : <https://kivy.org/doc/stable/api-kivy.html>
- [3] : <https://kivy.org/doc/stable/api-kivy.uix.label.html>
- [4] : <https://apps.apple.com/fr/app/xcode/id497799835?mt=12>
- [5] : <https://brew.sh/>
- [6] : <https://kivy.org/doc/stable/guide/packaging-ios.html>
- [7] : <https://www.youtube.com/watch?v=UAi3PG-qN2k>
- [8] : <https://github.com/kivy/buildozer>
- [9] : <https://www.crystax.net/en/download>
- [10] : <https://github.com/kivy/buildozer/issues/648>
- [11] : <https://www.oracle.com/technetwork/java/javase/downloads/index.html>
- [12] : <https://developer.android.com/studio>
- [13] : <https://www.virtualbox.org/wiki/Downloads>
- [14] : <https://stackoverflow.com/questions/29188304/kivy-buildozer-vm>
- [15] : <https://www.ubuntu.com/download/desktop>
- [16] : <https://kivy.org/doc/stable/tutorials/pong.html>
- [17] : <https://kivy.org/#home>