Northeastern University

INFO 6105
Data Sci Eng Tools & Mthds
Lecture 9 Client-side UIs

20 March 2019

UI Development

UIs

- This class is *data science*, so it's all on the server side
- Let's take a break and spend on a class on the client side and see how we can leverage what we learned with python to develop UIs with python
- Specifically, let's use a framework called `kivy` that lets you build UIs on Android, so you can have a UI on your phone

Editors

- On Windows:
  - Notepad++
    - https://notepad-plus-plus.org/download/v7.6.4.html
- On the Mac:
  - Sublime Text
    - https://www.sublimetext.com/3
- On both Windows and the Mac:
  - Install Visual Studio Code
    - https://code.visualstudio.com/Download
  - Install python extension:
    - https://marketplace.visualstudio.com/items?itemName=ms-python.python
  - Read and try this out:
    - https://code.visualstudio.com/docs/python/python-tutorial
  - Install `kivy` extension:
    - https://marketplace.visualstudio.com/items?itemName=BattleBas.kivy-vscode

Do *not*:

- Use `Notepad` or any other rich text editor for development

Install new python runtimes

- Python 3.7
- Python 2.7 (NO NEED!)
- From https://www.python.org/downloads/
- Yes, that's right, install *both*, and:
  - Do *not* make them your default python engine
  - *Reject* the suggested installation folder and install them on your root data drive:
    - `Python37_64`
    - `Python27_32`
- *On Windows*: Create environment variables to switch from one to the other
  - See professor's
- Mac: See next slide
- Create virtual environments folders at the root of your data drive for *each* python runtime
  - E,g, `C:/python-virtual-environments, C:/python-2.7-virtual-environments`

Do not use Anaconda3 as your base env!

- **Anaconda3 is a server-side, data science environment**
  - **It won't run on a cell phone!**

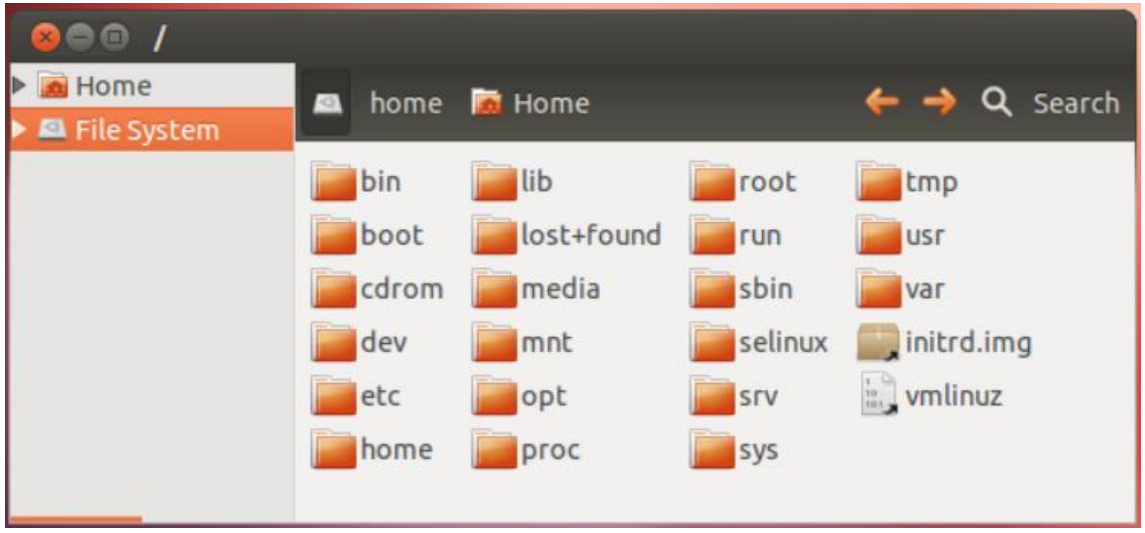
On the Mac

- Replace `python` with `python3` for labs today
- Replace `pip` with `pip3` (if you have `pip3` installed, otherwise continue using `pip`)
- Use `python` when referring to python 2.7, and `python3` for python 3.x

About folders

- Recommend not to run code, nor save Lecture files inside OS folders, like `C:\Users`, `MyDocuments`, `Desktop`, `Downloads`, etc.
  - These are OS-controlled folders. That means your OS plays with them, and may one day, decide to delete their contents
- So, on Windows, create your own `C:\usr` folder, and save all your lectures and code therein
- On the Mac, learn your linux folder structure
  - Use `Command + Shift + G` to go to your linux folder structure
  - http://osxdaily.com/2011/08/31/go-to-folder-useful-mac-os-x-keyboard-shortcut/

Linux folder structure



Linux folders

/ – The Root Directory
Everything on your Linux system is located under the / directory, known as the root directory. You can think of the / directory as being similar to the C:\ directory on Windows – but this isn't strictly true, as Linux doesn't have drive letters. While another partition would be located at D:\ on Windows, this other partition would appear in another folder under / on Linux.

/bin – Essential User Binaries
The /bin directory contains the essential user binaries (programs) that must be present when the system is mounted in single-user mode.

/boot – Static Boot Files
The /boot directory contains the files needed to boot the system

/dev – Device Files
Linux exposes devices as files, and the /dev directory contains a number of special files that represent devices

/etc – Configuration Files

/home – Home Folders
The /home directory contains a home folder for each user. For example, if your user name is bob, you have a home folder located at /home/bob. This home folder contains the user's data files and user-specific configuration files. Each user only has write access to their own home folder and must obtain elevated permissions (become the root user) to modify other files on the system.

/lib – Essential Shared Libraries
The /lib directory contains libraries needed by the essential binaries in the /bin and /sbin folder. Libraries needed by the binaries in the /usr/bin folder are located in /usr/lib.

Linux folders (continued)

/opt – Optional Packages
The /opt directory contains subdirectories for optional software packages. It's commonly used by proprietary software that doesn't obey the standard file system hierarchy – for example, a proprietary program might dump its files in /opt/application when you install it.

/proc – Kernel & Process Files
The /proc directory similar to the /dev directory because it doesn't contain standard files. It contains special files that represent system and process information.

/root – Root Home Directory
The /root directory is the home directory of the root user. Instead of being located at /home/root, it's located at /root. This is distinct from /, which is the system root directory.

/run – Application State Files
The /run directory is fairly new, and gives applications a standard place to store transient files they require like sockets and process IDs. These files can't be stored in /tmp because files in /tmp may be deleted.

/sbin – System Administration Binaries
The /sbin directory is similar to the /bin directory. It contains essential binaries that are generally intended to be run by the root user for system administration.

Linux folders (continued)

/srv – Service Data
The /srv directory contains “data for services provided by the system.” If you were using the Apache HTTP server to serve a website, you’d likely store your website’s files in a directory inside the /srv directory.

/tmp – Temporary Files
Applications store temporary files in the /tmp directory. These files are generally deleted whenever your system is restarted and may be deleted at any time by utilities such as tmpwatch.

/usr – User Binaries & Read-Only Data
The /usr directory contains applications and files used by users, as opposed to applications and files used by the system.

The /usr/local directory is where locally compiled applications install to by default – this prevents them from mucking up the rest of the system.

/var – Variable Data Files
The /var directory is the writable counterpart to the /usr directory, which must be read-only in normal operation. Log files and everything else that would normally be written to /usr during normal operation are written to the /var directory.

Using an `Info6105` folder, right?

- **With all your Web Tools lectures**
  - **Not in `C:/Users` nor `C:/Desktop`**
  - **Mac:**
    - **Lectures: CTRL ALT g → `C:\home\dino\info6105`**
    - **Apps: CTRL ALT g → `C:\usr\local\kivy\pong`**
- **Recommend create a folder for *each one* of your NU classes, and download *all* your lectures therein**
  - **With all material for each lecture in the right folder**
  - **For example, for Lecture 9, there should be a folder called `labs`, and within it a folder called `kivy`**

Create a new virtual environment (python 3)

- Type `python --version`
  - Make sure it's `python 3.x`, otherwise, install python 3.x at home and follow game development with your neighbor, or go ahead with python 2.7 and *maybe* it works
  - On the MAC: Use `python3` instead of `python` on cmds. below
- Upgrade pip
  - `python -m pip install -U pip`
- Install virtualenv using
  - `pip install virtualenv`
- Go to your root drive, and create a folder called
  - `python-virtual-environments`
- ***In that folder***, create a virtualenv called `kivy`:
  - `python -m venv kivy`
  - Windows: `cd kivy\Scripts`
  - `activate`
  - Mac: `cd kivy\bin`
  - `source activate`

To leave the environment and go back to the base environment

- **Type:**
  - `deactivate`

References

- https://docs.python.org/3/library/venv.html
- https://www.geeksforgeeks.org/creating-python-virtual-environment-windows-linux/

Create solution folder & install `kivy`

- Now, *leave* that virtual environment folder!
  - That is where your binary packages will install. Go somewhere *entirely different* and create a folder for your source code
  - You can create a folder wherever you wish, and when you install packages, all package installs will go to your `kivy` environment folder, not your current source code folder, and you are completely *independent* from any other installations for other environments or your base environment
- Install `kivy`, Windows:
- `python -m pip install docutils pygments kivy.deps.sdl2 kivy.deps.glew`
- `python -m pip install kivy.deps.gstreamer`
- `python –m pip install kivy`
- Install `kivy`, Mac:
- `brew install pkg-config sdl2 sdl2_image sdl2_ttf sdl2_mixer gstreamer`
- `pip install Cython==0.26.1`
- `pip install kivy`

Reference

- **Problems during install?**
  - https://kivy.org/#download
  - https://riptutorial.com/kivy

Hello World in Kivy

- **Create new file `helloword.py`:**

```
from kivy.app import App
from kivy.uix.label import Label

class FirstKivy(App):
    def build(self):
        return Label(text="Hello World!")

FirstKivy().run()
```

- **On the command line, run:**
  - **`python helloworld.py`**

Kivy button

- **Create new file `button.py`:**

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

class FirstKivy(App):
    def build(self):
        return Button(text="Welcome to games!")

FirstKivy().run()
```

- **On the command line, run:**
  - `python button.py`
- **Click on the text**

Kivy button & change background

- **Create new file `button_b.py`:**

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

class KivyButton(App):
    def build(self):
        return Button(text="Welcome to pink!", background_color=(155,0,51,53))

KivyButton().run()
```

- **On the command line, run:**
  - **`python button_b.py`**
- **Click on the text**

Disable a button after key press

- **Create new file `button_d.py`:**

```
from kivy.uix.button import Button
from kivy.app import App
from functools import partial

class KivyButton(App):
    def disable(self, instance, *args):
        instance.disabled = True

    def update(self, instance, *args):
        instance.text = "I am Disabled!"

    def build(self):
        mybtn = Button(text="Click me to disable")
        mybtn.bind(on_press=partial(self.disable, mybtn))
        mybtn.bind(on_press=partial(self.update, mybtn))
        return mybtn

KivyButton().run()
```

- **`update()` updates the text of our button after clicking on it**
- **`build()` runs when a button is created**
- **The return value from `disable()` is bound to the `on_press()` function of our button**

Size and position

- **Create new file `button_s.py`:**

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

class KivyButton(App):
    def build(self):
        return Button(text="Welcome to games!", pos=(300,350), size_hint = (.25, .18))

KivyButton().run()
```

Button that looks like your professor

- **Create new file `button_dino.py`:**
  - **Add `dino.jpg` to your folder**

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.lang import Builder
Builder.load_string("""
<KivyButton>:
    Button:
        text: "Hello Dino!"
        size_hint: .12, .12
        Image:
            source: 'dino.jpg'
            center_x: self.parent.center_x
            center_y: self.parent.center_y
""")

class KivyButton(App, BoxLayout):
    def build(self):
        return self

KivyButton().run()
```

Scrolling

- **Create new file `scroll.py`:**

```
from kivy.app import App
from kivy.uix.recycleview import RecycleView
from kivy.lang import Builder

Builder.load_string('''
<ExampleRV>:
    viewclass: 'Button'
    RecycleBoxLayout:
        size_hint_y: None
        height: self.minimum_height
        orientation: 'vertical'
''')

class ExampleRV(RecycleView):
    def __init__(self, **kwargs):
        super(ExampleRV, self).__init__(**kwargs)
        self.data = [{'text': str(x)} for x in range(20)]

class ScrollApp(App):
    def build(self):
        return ExampleRV()

ScrollApp().run()
```

Canvas app

- **Create new file `canvas.py`:**

```
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.boxlayout import BoxLayout
kvWidget = """
MyWidget:
    orientation: 'vertical'
    canvas:
        Rectangle:
            size: self.size
            pos: self.pos
            source: 'dino.jpg'
"""
class MyWidget(BoxLayout):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

class CanvasApp(App):
    def build(self):
        return Builder.load_string(kvWidget)

CanvasApp().run()
```

Pong #1

CANVAS

(SKIP THIS SECTION AND GO STRAIGHT TO BREAKOUT FOR ONE-PLAYER APP)

Use Visual Studio Code

- Create new folder
- Add two empty files:
  - `main.py`
  - `pong.kv`
- Start Visual Studio Code (VSC)
  - Open folder (containing the two files above)
  - Install VSC extension for Python (marketplace)
  - Install VSC extension for kivy (marketplace)
- Edit files `main.py` and `pong.kv` from within VSC
- Run `python main.py` from terminal within VSC
  - Or use VSC ”>” button with a python3 option
- Add debugging breakpoint

main.py

```
from kivy.app import App
from kivy.uix.widget import Widget

class PongGame(Widget):
    pass

class PongApp(App):
    def build(self):
        return PongGame()

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

pong.kv

```
#:kivy 1.0.9

<PongGame>:
    canvas:
        Rectangle:
            pos: self.center_x - 5, 0
            size: 10, self.height

    Label:
        font_size: 70
        center_x: root.width / 4
        top: root.top - 50
        text: "0"

    Label:
        font_size: 70
        center_x: root.width * 3 / 4
        top: root.top - 50
        text: "0"
```

Pong #2

ADDING A BALL

main.py

```
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import NumericProperty, ReferenceListProperty
from kivy.vector import Vector

class PongBall(Widget):
    velocity_x = NumericProperty(0)
    velocity_y = NumericProperty(0)
    velocity = ReferenceListProperty(velocity_x, velocity_y)

    def move(self):
        self.pos = Vector(*self.velocity) + self.pos

class PongGame(Widget):
    pass

class PongApp(App):
    def build(self):
        return PongGame()

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

pong.kv

```
#:kivy 1.0.9

<PongBall>:
    size: 50, 50
    canvas:
        Ellipse:
            pos: self.pos
            size: self.size

<PongGame>:
    canvas:
        Rectangle:
            pos: self.center_x-5, 0
            size: 10, self.height

    Label:
        font_size: 70
        center_x: root.width / 4
        top: root.top - 50
        text: "0"

    Label:
        font_size: 70
        center_x: root.width * 3 / 4
        top: root.top - 50
        text: "0"

    PongBall:
        center: self.parent.center
```

Pong #3

ADDING ANIMATION

Making the ball move

- The ball has a move function… but it's not moving yet
- We need the move method of our ball to be called regularly
- Kivy lets you schedule any function you want using the Clock and specifying the interval:
  - `# update function of game object called once every 60th s`
    `Clock.schedule_interval(game.update, 1.0/60.0)`
- Need an update method for PongGame class

```
class PongGame(Widget):

    def update(self, dt):
        # call ball.move and other stuff
        pass

class PongApp(App):

    def build(self):
        game = PongGame()
        Clock.schedule_interval(game.update, 1.0/60.0)
        return game
```

Hooking up in the kv file

- **By giving the child widget an id and setting the PongGame's ball ObjectProperty to that id:**

```
<PongGame>:
    ball: pong_ball

    # ... (canvas and Labels)

    PongBall:
        id: pong_ball
        center: self.parent.center
```

Move the ball randomly

- everything is hooked up for the ball to bounce around
- Why is the ball not moving?
  - The ball's velocity is set to 0 on both x and y
  - Add a `serve_ball` method to the `PongGame` class and call it in the app's build method
  - Set a random x and y velocity for the ball, and also reset the position, so you can use it later to reset the ball when a player has scored a point

```
def serve_ball(self):
    self.ball.center = self.center
    self.ball.velocity = Vector(4, 0).rotate(randint(0, 360))
```

```
class PongApp(App):
    def build(self):
        game = PongGame()
        game.serve_ball()
        Clock.schedule_interval(game.update, 1.0 / 60.0)
        return game
```

Pong #3

CONNECT INPUT EVENTS

Adding Players and reacting to touch input

- Need movable player rackets and keep track of score
- How to move the Player widgets in response to user input?
- A widget can react to input by implementing the `on_touch_down`, the `on_touch_move` and the `on_touch_up` methods
  - By default, the Widget class implements these methods by just calling the corresponding method on all its child widgets to pass on the event until one of the children returns `True`
- Rackets just need to move up and down
  - Implement the `on_touch_move` function for the `PongGame` class and have it set the position of the left or right player based on whether the touch occurred on the left or right side of the screen

main.py

```
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import NumericProperty, ReferenceListProperty,\
   ObjectProperty
from kivy.vector import Vector
from kivy.clock import Clock

class PongPaddle(Widget):
   score = NumericProperty(0)

   def bounce_ball(self, ball):
      if self.collide_widget(ball):
         vx, vy = ball.velocity
         offset = (ball.center_y - self.center_y) / (self.height / 2)
         bounced = Vector(-1 * vx, vy)
         vel = bounced * 1.1
         ball.velocity = vel.x, vel.y + offset

class PongBall(Widget):
   velocity_x = NumericProperty(0)
   velocity_y = NumericProperty(0)
   velocity = ReferenceListProperty(velocity_x, velocity_y)

   def move(self):
      self.pos = Vector(*self.velocity) + self.pos

class PongGame(Widget):
   ball = ObjectProperty(None)
   player1 = ObjectProperty(None)
   player2 = ObjectProperty(None)

   def serve_ball(self, vel=(4, 0)):
      self.ball.center = self.center
      self.ball.velocity = vel

   def update(self, dt):
      self.ball.move()
```

main.py (continued)

```
        # bounce of paddles
        self.player1.bounce_ball(self.ball)
        self.player2.bounce_ball(self.ball)

        # bounce ball off bottom or top
        if (self.ball.y < self.y) or (self.ball.top > self.top):
            self.ball.velocity_y *= -1

        # went of to a side to score point?
        if self.ball.x < self.x:
            self.player2.score += 1
            self.serve_ball(vel=(4, 0))
        if self.ball.x > self.width:
            self.player1.score += 1
            self.serve_ball(vel=(-4, 0))

    def on_touch_move(self, touch):
        if touch.x < self.width / 3:
            self.player1.center_y = touch.y
        if touch.x > self.width - self.width / 3:
            self.player2.center_y = touch.y

class PongApp(App):
    def build(self):
        game = PongGame()
        game.serve_ball()
        Clock.schedule_interval(game.update, 1.0 / 60.0)
        return game

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

pong.kv

```
#:kivy 1.0.9

<PongBall>:
    size: 50, 50
    canvas:
        Ellipse:
            pos: self.pos
            size: self.size

<PongPaddle>:
    size: 25, 200
    canvas:
        Rectangle:
            pos:self.pos
            size:self.size

<PongGame>:
    ball: pong_ball
    player1: player_left
    player2: player_right

    canvas:
        Rectangle:
            pos: self.center_x-5, 0
            size: 10, self.height

    Label:
        font_size: 70
        center_x: root.width / 4
        top: root.top - 50
        text: str(root.player1.score)
```

pong.kv (continued)

```
Label:
    font_size: 70
    center_x: root.width * 3 / 4
    top: root.top - 50
    text: str(root.player2.score)

PongBall:
    id: pong_ball
    center: self.parent.center

PongPaddle:
    id: player_left
    x: root.x
    center_y: root.center_y

PongPaddle:
    id: player_right
    x: root.width-self.width
    center_y: root.center_y
```

Run!

- **Possible improvements:**
  - **Make the game end after a certain score**
    - **Maybe once a player has 10 points, you can display a large "PLAYER 1 WINS" label and/or add a main menu to start, pause and reset the game**
  - **Make it a 4 player Pong Game, left, right, up down. Most tablets have Multi-Touch support, so wouldn't it be cool to have a player on each side and have four people play at the same time?**
  - **Fix simplistic collision check so hitting the ball with an end of the paddle results in a more realistic bounce**

Breakout game

BREAKOUT

main.py

```
import kivy
kivy.require('1.1.3')

from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import NumericProperty, ReferenceListProperty, ObjectProperty, StringProperty,
ListProperty
from kivy.vector import Vector
from kivy.clock import Clock
from kivy.logger import Logger
from kivy.uix.floatlayout import FloatLayout
from kivy.core.window import Window

class Breaker(FloatLayout):
    pass


class DebugPanel(Widget):
    fps = StringProperty(None)
    windowHeight = StringProperty(None)
    windowWidth = StringProperty(None)

    def __init__(self, **kwargs):
        super(DebugPanel, self).__init__(**kwargs)
        Clock.schedule_interval(self.update_fps, 0.1)

    def update_fps(self, dt):
        self.fps = str(int(Clock.get_fps()))
        self.windowWidth = str(Window.width)
        self.windowHeight = str(Window.height)
```

main.py (continued)

```
class BreakerPaddle(Widget):
    score = NumericProperty(0)
    max_velocity = 10

    def bounce_ball(self, ball):
        if self.collide_widget(ball):
            vx, vy = ball.velocity
            offset = (ball.center_x - self.center_x) / (self.width / 10)
            bounced = Vector(vx, -1 * vy)

            if (abs(bounced.x) < self.max_velocity) and (abs(bounced.y) < self.max_velocity):
                bounced *= 1.1

            ball.velocity = bounced.x + offset, bounced.y


class BreakerBall(Widget):
    velocity_x = NumericProperty(0)
    velocity_y = NumericProperty(0)
    velocity = ReferenceListProperty(velocity_x, velocity_y)

    def move(self):
        self.pos = Vector(*self.velocity) + self.pos
```

main.py (continued)

```
class BreakerBrick(Widget):
    particle = ObjectProperty(None)
    game = ObjectProperty(None)

    def collide(self, game, ball):
        if self.collide_widget(ball):
            # Bounce
            vx, vy = ball.velocity
            offset = (ball.center_x - self.center_x) / (self.width / 10)
            bounced = Vector(vx, -1 * vy)
            ball.velocity = bounced.x + offset, bounced.y

            # Remove brick
            game.bricks.remove(self)
            self.opacity = 0


class BreakerGame(Widget):
    ball = ObjectProperty(None)
    player = ObjectProperty(None)
    brick = ObjectProperty(None)
    brick2 = ObjectProperty(None)
    brick3 = ObjectProperty(None)
    bricks = ListProperty(None)

    def ini_bricks(self):
        self.bricks.append(self.brick)
        self.bricks.append(self.brick2)
        self.bricks.append(self.brick3)

    def serve_ball(self, velocity=(0, 4)):
        self.ball.center_x = self.player.center_x
        self.ball.center_y = self.player.height + self.ball.height
        self.ball.velocity = velocity
```

main.py (continued)

```
def update(self, dt):
    self.ball.move()

    # Bounce of paddles
    self.player.bounce_ball(self.ball)

    # Brick collision
    for brickItem in self.bricks:
        brickItem.collide(self, self.ball)


    # Bounce ball off top
    if self.ball.top > self.top:
        self.ball.velocity_y *= -1

    # Bounce ball off left or right
    if (self.ball.x < self.x) or (self.ball.right > self.right):
        self.ball.velocity_x *= -1

    # Ball lost
    if self.ball.y < self.y:
        self.serve_ball()

def on_touch_move(self, touch):
    if touch.y < (self.height / 2):
        if (touch.x + self.player.width / 2) > self.width:
            self.player.center_x = self.width - (self.player.width / 2)
        else:
            if (touch.x - self.player.width / 2) < 0:
                self.player.center_x = self.player.width / 2
            else:
                self.player.center_x = touch.x
```

main.py (continued)

```
class BreakerApp(App):
    def build(self):
        game = BreakerGame()
        game.ini_bricks()
        game.serve_ball()
        Clock.schedule_interval(game.update, 1.0 / 60.0)

        return game

if __name__ in ('__main__', '__android__'):
    BreakerApp().run()
```

breaker.kv

```
#:kivy 1.0.9

<BreakerBall>:
    size: 40, 40
    canvas:
        Ellipse:
            pos: self.pos
            size: self.size

<BreakerBrick>
    size: 150, 30
    canvas:
        Rectangle:
            pos: self.pos
            size: self.size

<BreakerPaddle>:
    size: 200, 25
    canvas:
        Rectangle:
            pos: self.pos
            size: self.size

<DebugPanel>:
    fps: root.fps
    windowWidth: root.windowWidth
    windowHeight: root.windowHeight

    Label:
        pos: root.pos
        size: 15, 15
        font_size: 15
        text: 'FPS: ' + root.fps if root.fps != None else 'FPS:'
```

breaker.kv (continued)


```
    Label:
        x: root.x + 100
        y: root.y
        size: 15, 15
        font_size: 15
        text: 'Height: ' + root.windowHeight if root.windowHeight != None else 'Height:'

    Label:
        x: root.x + 200
        y: root.y
        size: 15, 15
        font_size: 15
        text: 'Width: ' + root.windowWidth if root.windowWidth != None else 'Width:'

<BreakerGame>:
    ball: breaker_ball
    player: breaker_player
    brick: breaker_brick_test
    brick2: breaker_brick_test2
    brick3: breaker_brick_test3

    BreakerBall:
        id: breaker_ball
        center: self.parent.center

    BreakerPaddle:
        id: breaker_player
        center_x: root.center_x
        center_y: self.height
```

breaker.kv (continued)

```
BreakerBrick:
    id: breaker_brick_test
    center_x: root.center_x
    center_y: root.height - 50

BreakerBrick:
    id: breaker_brick_test2
    center_x: 100
    center_y: root.height - 50

BreakerBrick:
    id: breaker_brick_test3
    center_x: root.width - 100
    center_y: root.height - 50

DebugPanel:
    id: debug_fps
    x: 25
    y: root.height - 20
```

Run

- **Add more bricks!**

Publishing

DEPLOY TO ANDROID/IOS

Deploy & play on your phone

- Go to `github.com/bulldozer`, download & extract package
  - Or, install `git`, then run:
    `git clone git://github.com/kivy/bulldozer`
- Then go to top folder of your download:
  - `cd bulldozer`
  - `sudo python/python3 setup.py`
  - `install`
- Then go back to your app folder and run:
  - `bulldozer init`
    - This creates a `bulldozer.spec` file with info to create your Android APK file
  - `bulldozer android debug`
    - Will download the Android SDK and NDK (hundreds of MBs)
    - Compile your APK in a newly created `bin` folder
    - Now, email APK file to your phone
    - Enable `application installation from unknown sources` on your phone, or digitally sign the APK so it can be uploaded to the Play store

Reference

- https://kivy.org/doc/stable/guide/packaging-android.html

Putting your APK on the Play Store #1

Be very careful with carriage returns!

- **Build and sign a release APK:**
  - `keytool -genkey -v -keystore test-release-key.keystore -alias test-alias -keyalg RSA -keysize 2048 -validity 10000`
- **Compile app in release mode for android:**
  - `bulldozer android release`
- **Compile app in release mode for iOS (?):**
  - `bulldozer ios release`
- **Sign the APK with your new key:**
  - `jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore ./test-release-key.keystore ./bin/BreakoutApp-0.1-release-unsigned.apk test-alias`
- **Align the APK file:**
  - `~/.bulldozer/android/platform/android-sdk-21/tools/zipalign -v 4 ./bin/BreakoutApp-0.1-release-unsigned.apk ./bin/BreakoutApp-0.1-release.apk`

Putting your APK on the Play Store #2

- **Sign up as a Google developer:**
  - **https://play.google.com/apps/publish/signup**
  - **You'll need to pay a one-off $25 charge, but then you can upload as many apps as you want**
  - **Up to you!**

Putting your APK on the Play Store #3

- **Upload app to your app store**
  - **Click `Add new application`**
  - **Click `Publish`**
  - **Few hours for your app to go live**

Explore github

- https://github.com/kivy/kivy/wiki/List-of-Kivy-Projects
- https://play.google.com/store/apps/details?id=au.com.quadropoly.quadropoly


CLASS
DISMISSED
iStockphoto