Day 4

•••

The start of KivyBird!

This time, we’re gonna start the flappy bird app!

Remember to add this code to the bottom of your python file

main.py

```
if __name__ == "__main__":
    KivyBirdApp().run()
```

We will first start with animating the background

Then we start on the rest of the sprites, and see how far we get!

A general convention is to put all your import statements at the top of the file for readability, so remember to do that throughout the code!

The basic widget

This is the basic widget.
Last time we talked about how classes can inherit from each other.
That is why we create a base class that gives all our other classes the opportunity to grab our textures.

main.py

```
+from kivy.core.image import Image
+from kivy.uix.widget import Widget

+class BaseWidget(Widget):
+    def load_tileable(self, name):
+        t = Image('images/{}.png'.format(name)).texture
+        t.wrap = 'repeat'
+        setattr(self, 'tx_{}'.format(name), t)
```

We also make it so that the texture can be called from other things as an attribute called tx_* - for our background, that would be *tx_background*

The background

This is the class for everything that has to do with the background. First we creat the class and prepare an *ObjectProperty* that is to contain the actual texture.

Further more, we create a helpermethod called set_background_size that helps us set the texture to the right size for the screen. We also create the method *on_size* in order to change the texture size every time the screen changes.

main.py

```
+from kivy.properties import ObjectProperty
+class Background(BaseWidget):
+    tx_background = ObjectProperty(None)
+    def __init__(self, **kwargs):
+        super(Background, self).__init__(**kwargs)
+        self.load_tileable('background')
+    def set_background_size(self, tx):
+        tx.uvsize = (self.width / tx.width, -1)
+    def on_size(self, *args):
+        self.set_background_size(self.tx_background)
```

The app class

Within the app class, we create our own *on_start* method that sets the background, and tells it to update itself roughly 60 times a second.

self.roots.ids.background refers to the *kv* file that we will create momentarily.

main.py

```
+from kivy.app import App

+from kivy.clock import Clock

+class KivyBirdApp(App):

+    def on_start(self):

+        self.background = self.root.ids.background

+        Clock.schedule_interval(self.update, 1.0/60.0)

+    def update(self, nap):

+        self.background.update(nap)
```

The KV file!

Here, we create a FloatLayout as the *root-widget.* This means that it is the first thing to be added to the screen is this layout, and everything that goes with it.

Background is defined in *main.py* and here we *instantiate* it - and give it an id.

kivybird.kv

```
+FloatLayout:
+    Background:
+        id: background
+        canvas:
+            Rectangle:
+                pos: self.pos
+                size: (self.width, self.height)
+                texture: self.tx_background
```

canvas is the canvas in the background - and something we can draw on and give textures. In this case, *tx_background.*

The Update method!

Here is the update method that we call 60 times a minute. The update method calls a new method, *set_background_uv* with the name of the texture, and the value it is going to use in the new position.

main.py

```
class Background(BaseWidget):
...
+   def update(self, nap):
+       self.set_background_uv('tx_background', 2 * nap)

+    def set_background_uv(self, name, val):
+        t = getattr(self, name)
+        t.uvpos = ((t.uvpos[0] + val) % self.width, t.uvpos[1])
+        self.property(name).dispatch(self)
```

The rest of the imports!

For simplicity's sake, we will just import the rest of the classes we are using today

Once you are done, it will look like this:

main.py

```
from kivy.app import App
from kivy.clock import Clock
from kivy.core.image import Image
from kivy.uix.widget import Widget
-from kivy.properties import ObjectProperty
+from kivy.properties import ObjectProperty, ListProperty, NumericPropert
+from kivy.properties import AliasProperty
+from kivy.core.window import Window, Keyboard
+from kivy.uix.image import Image as ImageWidget

+import random
```

Pipes! - in the KV file

kivybird.kv

```
<Pipe>:
    canvas:
        Rectangle:
            pos: (self.x + 4, self.FLOOR)
            size: (56, self.lower_len)
            texture: self.tx_pipe
            tex_coords: self.lower_coords

        Rectangle:
            pos: (self.x, self.FLOOR + self.lower_len)
            size: (64, self.PTOP_HEIGHT)
            texture: self.tx_ptop
```

```
      Rectangle:
            pos: (self.x +4, self.upper_y)
            size: (56, self.upper_len)
            texture: self.tx_pipe
            tex_coords: self.upper_coords

        Rectangle:
            pos: (self.x, self.upper_y - self.
PTOP_HEIGHT)
            size: (64, self.PTOP_HEIGHT)
            texture: self.tx_ptop
    size_hint: (None, 1)
    width: 64
```

Pipes! Attributes

FLOOR refers to the “ground level” of the pipes
PTOP_HEIGHT is the size of the “top” of the pipes
PIPE_GAP is the space between the pipes

main.py

```
+class Pipe(BaseWidget):
+    FLOOR = 96
+    PTOP_HEIGHT = 26
+    PIPE_GAP = 150
+    tx_pipe = ObjectProperty(None)
+    tx_ptop = ObjectProperty(None)
+    ratio = NumericProperty(0.5)
+    lower_len = NumericProperty(0)
+    lower_coords = ListProperty((0, 0, 1, 0, 1, 1, 0, 1))
+    upper_len = NumericProperty(0)
+    upper_coords = ListProperty((0, 0, 1, 0, 1, 1, 0, 1))
+    upper_y = AliasProperty(
+            lambda self: self.height - self.upper_len,
+            None, bind=['height', 'upper_len'])
```

Pipes! - the textures

Here we add the same init method as in the background, in order to load the textures for our pip and its top.

main.py

```
class Pipe(BaseWidget):

...

+def __init__(self, **kwargs):

+         super(Pipe, self).__init__(**kwargs)


+         for name in ('pipe', 'ptop'):

+             self.load_tileable(name)
```

Pipes - Coordinates (difficult subject!)

Here we set the coordinates for the pipes. This is a complex subject, and we will skip going into detail here.

If wanted, I can prepare something for next time that will explain this.

main.py

```
class Pipe(BaseWidget):
...
+   def set_coords(self, coords, len):
+       len /= 16
+       coords[5:] = (len, 0, len)
+    def on_size(self, *args):
+       pipes_length = self.height - (
+               Pipe.FLOOR + Pipe.PIPE_GAP + 2 * Pipe.PTOP_HEIGHT)
+       self.lower_len = self.ratio * pipes_length
+       self.upper_len = pipes_length - self.lower_len
+       self.set_coords(self.lower_coords, self.lower_len)
+       self.set_coords(self.upper_coords, self.upper_len)
+       self.bind(ratio=self.on_size)
```

Spawning pipes!

Here we add the pipes to the main class!

Here we create a method to store the pipes we put out there, so we can remove them when they go out of bounds.

main.py

```
class KivyBirdApp(App):
+    pipes = []
def on_start(self):
+           self.spacing = 0.5 * self.root.width
...
def spawn_pipes(self):
        for p in self.pipes:
            self.root.remove_widget(p)
        self.pipes = []
        for i in range(4):
            p = Pipe(x=self.root.width + (self.spacing * i))
            p.ratio = random.uniform(0.25, 0.75)
            self.root.add_widget(p)
            self.pipes.append(p)
```

Moving the pipes!

Here, we update the *update* method so that it also updates and fetches pipes.

main.py

```
class KivyBirdApp(App):
...
    def update(self, nap):
            self.background.update(nap)

+           for p in self.pipes:
+               p.x -= 96 * nap
+               if p.x <= -64:
+                   p.x += 4 * self.spacing
+                   p.ratio = random.uniform(0.25, 0.75)
```

“The Bird”!

Just like with the background, we bring in the bird.

Here, the bird is only going to be at a set point at the beginning.

Since we don’t *tile* the bird, we just bring it in as a regular image..

kivybird.kv

```
FloatLayout:
...

+    Bird:
+            id: bird
+            pos_hint: {'center_x': 0.3333, 'center_y': 0.6}
+            size: (50, 50)
+            size_hint: (None, None)
+            source: 'images/flappynormal.png'
```

The Bird!

In order to bring in the bird, all you need to do is this!

main.py

```
+class Bird(ImageWidget):
+    pass
```

User input!

Here, we bind the touch input and a method called *on_key_down* - this calls self.user_action if you press space.
What happens is that you start the game, for now.

```
class KivyBirdApp(App):
+   playing = False
def on_start(self):
...
+        Window.bind(on_key_down=self.on_key_down)
+        self.background.on_touch_down = self.user_action
+   def on_key_down(self, window, key, *args):
+        if key == Keyboard.keycodes['spacebar']:
+            self.user_action()
+    def user_action(self, *args):
+        if not self.playing:
+            self.spawn_pipes()
+            self.playing = True
```

Flying is only delaying a fall!

Here, we add “gravity” to our bird.

Note that *ACCEL_FALL*can be changed for increased difficulty!

```
class Bird(ImageWidget):
-     pass
+     ACCEL_FALL = 0.25
+     speed = NumericProperty(0)
+     def gravity_on(self, height):
+         self.pos_hint.pop('center_y', None)
+         self.center_y = 0.6 * height
+     def update(self, nap):
+         self.speed -= Bird.ACCEL_FALL
+         self.y += self.speed
```

Updating the main class

Here we add that the bird is supposed to start falling once you press the screen.

Also adds update to the bird itself.

```
class KivyBirdApp(App):
...
def user_action(self, *args):
         if not self.playing:
+                self.bird.gravity_on(self.root.height)
             self.spawn_pipes()
             self.playing = True
def update(self, nap):
        self.background.update(nap)
        if not self.playing:
            return
+        self.bird.update(nap)
```

Flying!

Here we add how quickly the bird ascends through *ACCEL_JUMP*

And add the method we call for it!

```
class Bird(ImageWidget):
...
+   ACCEL_JUMP = 5
...
+  def bump(self):
+         self.speed = Bird.ACCEL_JUMP
```

Rotate the bird depending on speed! (Difficult subject!)

This is a property to calculate the angle of the burde in relation to its speed, together with what we'll add in the KV file

```
class Bird(ImageWidget):
...
+    angle = AliasProperty(
+                lambda self: 5 * self.speed,
+                None, bind=['speed'])
...
```

Rotate the bird, KV part

Again, this is a difficult subject so I won't go into full detail here.

```
+<Bird>:
+    canvas.before:
+        PushMatrix
+        Rotate:
+            angle: root.angle
+            axis: (0, 0, 1)
+            origin: root.center

+    canvas.after:
+        PopMatrix
```

Collision!

Here we check if the bird has reached the roof or floor

We also check if if has collided with the pipe, through a default kivy function called *collide_widget*

```
class KivyBirdApp(App):
...
+   def test_game_over(self):
+        screen_height = self.root.height
+        if self.bird.y < 90 or self.bird.y > screen_height - 50:
+            return True
+        for p in self.pipes:
+            if not p.collide_widget(self.bird):
+                continue
+            if (self.bird.y < p.lower_len + 116 or
+                    self.bird.y > screen_height - (p.upper_len + 75)):
+                return True
+        return False
```

Check if game over

Here we check if game over equals True

If it is, set playing to false.

```
   def update(self, nap):
...
+        if self.test_game_over():
+            self.playing = False
```