DIGITALJUNKY

A journey through information technologies

SHOW MENU ≡

Make a Snake game for Android written in Python – Part 3

FEBRUARY 5, 2015 / 2 COMMENTS

I hope you enjoyed the first two sections of the tutorial. If you got through it, rest assured that the hardest part is behind us. The game engine is at 90% done at that point, and handling screens is very straightforward in kivy. We're just going to make a few arrangements here before packaging the app. It's a good front-end exercise because this time we'll rely a lot more on the kivy language.

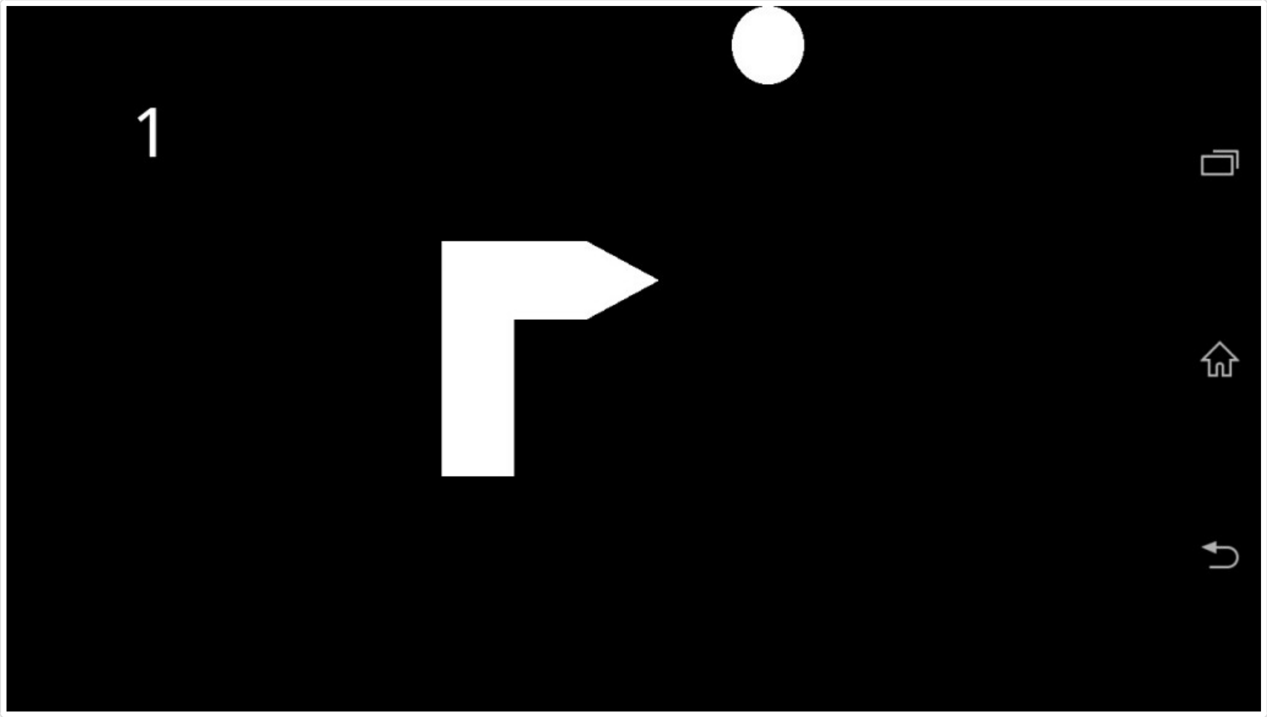
Creating the screens

Our application requires two screens : one for the welcome screen and one for the playground. I don't count the widget where we're going to display the options as a screen because it will be a popup placed on the welcome screen. First we will specify the layout of our widgets in the .kv file, and only then write the corresponding classes in Python.

Front-end :

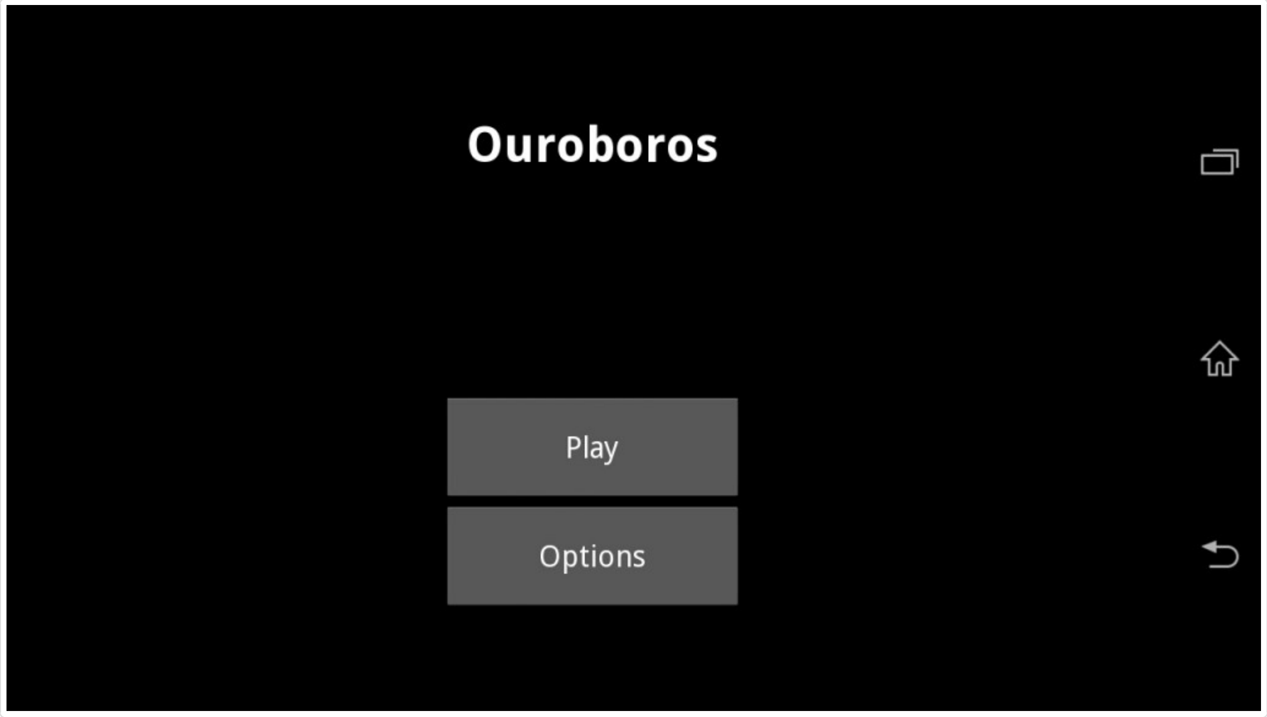
The PlaygroundScreen is the easiest : it only contains the playground !

```
<PlaygroundScreen>:
    game_engine: playground_widget_id

    Playground:
        id: playground_widget_id
```



The main screen will be composed of several internal widgets that will help us organize the three main elements : the title of the App (I chose Ouroboros but feel free to call it what you want), a Play button that triggers the entry of the PlaygroundScreen and an Option button calling the Option popup. Layouts will help us organize these elements in term of dimensions and positioning.

```
<WelcomeScreen>
    AnchorLayout:
        anchor_x: "center"

        BoxLayout:
            orientation: "vertical"
            size_hint: (0.5, 1)
            spacing: 10

            Label:
                size_hint_y: .4
                text: "Ouroboros"
                valign: "bottom"
                bold: True
                font_size: 50
                padding: 0, 0

            AnchorLayout:
                anchor_x: "center"
                size_hint_y: .6

                BoxLayout:
                    size_hint: .5, .5
                    orientation: "vertical"
                    spacing: 10

                    Button:
                        halign: "center"
                        valign: "middle"
                        text: "Play"

                    Button:
                        halign: "center"
                        valign: "middle"
                        text: "Options"
```



The option popup will occupy 3/4 of the welcome screen. It will contain the widgets needed to interact on the parameters and a Save button. We'll add the options *per se* afterward. For now we're just preparing the layout.

```
<OptionsPopup>
    title: "Options"
    size_hint: .75, .75

    BoxLayout:
        orientation: "vertical"
        spacing: 20

        GridLayout:
            size_hint_y: .8
            cols: 2

        AnchorLayout:
            anchor_x: "center"
            size_hint: 1, .25

            Button:
                size_hint_x: 0.5
                text: "Save changes"
                on_press: root.dismiss()
```

Back-end :

Let's add the Python counterparts of the classes we defined in the kivy language, and the behavior we want to give them. Both screens will inherit from Screen (what a surprise) and the OptionsPopup from Popup (again, how baffling).

The Welcome screen requires only one method : *show_popup()* that will be called when the Options button is pressed on the main screen. We don't have to define anything else for the Play button because it will use the ability of its parent to access the screen manager. Yay Kivy ! Since we'll act on the OptionsPopup, its instance will be stored as a property.

```
class WelcomeScreen(Screen):
    options_popup = ObjectProperty(None)

    def show_popup(self):
        # instanciate the popup and display it
        self.options_popup = OptionsPopup()
        self.options_popup.open()
```

The GameScreen contains the Playground. Recall how in the last part we started the game on App start. We don't want that anymore : the Welcome screen should be shown on App start, and the game should start only when the Playground comes into view. Thus you can delete the `on_start()` method from the main class and transfer the afferent logic here :

```
class PlaygroundScreen(Screen):
    game_engine = ObjectProperty(None)

    def on_enter(self):
        # we screen comes into view, start the game
        self.game_engine.start()
```

We might as well prepare the OptionsPopup class right now. We have to anyway otherwise the bindings we're going to make in a few steps wont work properly.

```
class OptionsPopup(Popup):
    pass
```

Our screens are ready. Good. What we want now is to add a ScreenManager to the App and to register the two screens. When we wanted to interact with an object after its instanciation, we usually declared it as an instance property of the object holding it. Here we're going to declare the screen manager at the class level because we'll need to call it without any direct reference to the parent holding it (we'll be able to call the parent class, but not the object directly).

```
class SnakeApp(App):
    screen_manager = ObjectProperty(None)

    def build(self):
        # declare the ScreenManager as a class property
        SnakeApp.screen_manager = ScreenManager()

        # instanciate the screens
        ws = WelcomeScreen(name="welcome_screen")
        ps = PlaygroundScreen(name="playground_screen")

        # register the screens in the screen manager
        self.screen_manager.add_widget(ws)
        self.screen_manager.add_widget(ps)

        return self.screen_manager
```

Give it a run : the welcome screen appears but hey, nothing happens if we press the buttons ! Maybe we should think of adding some bindings. Back to snake.kv. We're going to specify a behavior for the on_press() event of the buttons. Play will tell the screen manager to switch to the playground and Options will trigger the show_popup() method we just defined. Don't forget the button of the options popup : it will simply dismiss its parent widget.

```
<WelcomeScreen>
...
                    Button:
...
                        on_press: root.manager.current = "playground_screen"

                    Button:
...
                        on_press: root.show_popup()

<OptionsPopup>
...
            Button:
...
                on_press: root.dismiss()
```

One more thing : how do we get back to the main menu when we're in game ? Indeed, at that point we're stuck on the playground as soon as we reach it. In order to keep things simple, we'll just call a change of screen if the game is lost.

```
class Playground(Widget):
...
    def update(self, *args):
...
        # check for defeat
        # if it happens to be the case, reset the game and switch back to
        # the welcome screen
        if self.is_defeated():
            self.reset()
            SnakeApp.screen_manager.current = "welcome_screen"
            return
...
```

Full code.

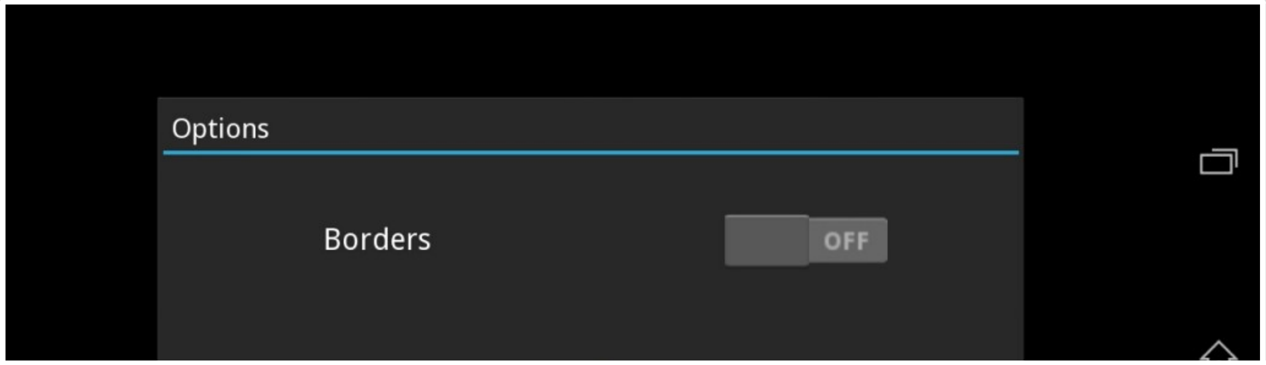
Try your new build : it's starting to look like something isn't it ? One more step and we'll be ready to package !

Adding the options

We're going to add two options :

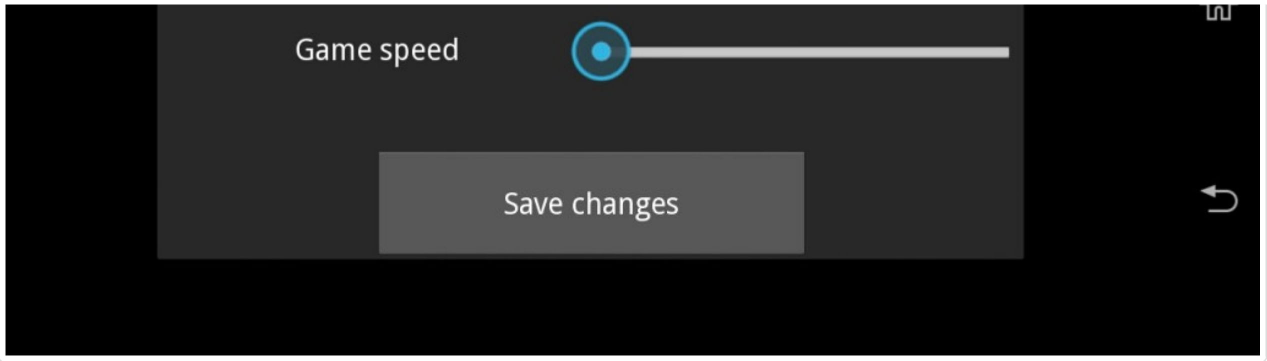
- Borders : at the moment the game is lost is the snake goes outbound, but in the snake I remember you could also choose to disable the borders (it even was the default setting wasn't it?). In that case, the snake would reappear on the other side. Let's implement that.
- Speed : we set the duration of each turn to 1 second. We're going to allow the user to choose from a predefined range of values its starting speed. Furthermore, we will add a mechanism so that the speed increases each time a fruit is eaten.

It will look like that :



Options

Borders OFF



Adding the necessary widgets to our options popup :

```
<OptionsPopup>
    border_option_widget: border_option_widget_id
    speed_option_widget: speed_option_widget_id

    title: "Options"
    size_hint: .75, .75

    BoxLayout:
        orientation: "vertical"
        spacing: 20

        GridLayout:
            size_hint_y: .8
            cols: 2

            Label:
                text: "Borders"
                halign: "center"

            Switch:
                id: border_option_widget_id

            Label:
                text: "Game speed"
                halign: "center"

            Slider:
                id: speed_option_widget_id
                max: 10
                min: 1
                step: 1
                value: 1
```

Before modifying the OptionPopup class so that it can dispatch its options, we have to prepare our game engine to receive them. What changes do we need to make ? First, add properties so that we can receive the option’s values. Then modify start() : if the border option is on, we’ll draw a line around the Playground to symbolize it. We’ll also compute the refresh rate based on the speed option. What else ? reset() will require a tweak too. Oh, and we have to add a new method to handle the snake re-positioning if the border option is off and it exits the screen, in which case it must reaper on the opposite side.

```
class Playground(Widget):
...
    # user options
    start_speed = NumericProperty(1)
    border_option = BooleanProperty(False)
...
    # game variables
...
    start_time_coeff = NumericProperty(1)
    running_time_coeff = NumericProperty(1)
...
    def start(self):
        # if border_option is active, draw rectangle around the game area
        if self.border_option:
            with self.canvas.before:
                Line(width=3.,
                     rectangle=(self.x, self.y, self.width, self.height))

        # compute time coeff used as refresh rate for the game using the
        # options provided (default 1.1, max 2)
        # we store the value twice in order to keep a reference in case of
        # reset (indeed the running_time_coeff will be incremented in game if
        # a fruit is eaten)
        self.start_time_coeff += (self.start_speed / 10)
        self.running_time_coeff = self.start_time_coeff
```

```
...
    def reset(self):
        # reset game variables
...
        self.running_time_coeff = self.start_time_coeff
...
    def is_defeated(self):
...
        # if the snake it out of the board and border option is on : defeat
        if self.border_option:
            if snake_position[0] > self.col_number \
                    or snake_position[0] < 1 \
                    or snake_position[1] > self.row_number \
                    or snake_position[1] < 1:
                return True

        return False

    def handle_outbound(self):
        """
        Used to replace the snake on the opposite side if it goes outbound
        (only called if the border option is set to False)
        """
        position = self.snake.get_position()
        direction = self.snake.get_direction()

        if position[0] == 1 and direction == "Left":
            # add the current head position as a tail block
            # otherwise one block would be missed by the normal routine
            self.snake.tail.add_block(list(position))
            self.snake.set_position([self.col_number + 1, position[1]])
        elif position[0] == self.col_number and direction == "Right":
            self.snake.tail.add_block(list(position))
            self.snake.set_position([0, position[1]])
        elif position[1] == 1 and direction == "Down":
            self.snake.tail.add_block(list(position))
            self.snake.set_position([position[0], self.row_number + 1])
        elif position[1] == self.row_number and direction == "Up":
            self.snake.tail.add_block(list(position))
            self.snake.set_position([position[0], 0])

    def update(self, *args):
...
        # registering the fruit poping sequence in the event scheduler
        if self.turn_counter == 0:
...
            Clock.schedule_interval(
                self.fruit.remove, self.fruit_rythme / self.running_time_coeff)
        elif self.turn_counter == self.fruit.interval:
...
            Clock.schedule_interval(
                self.pop_fruit, self.fruit_rythme / self.running_time_coeff)

        # if game with no borders, check if snake is about to leave the screen
        # if so, replace to corresponding opposite border
        if not self.border_option:
            self.handle_outbound()
...
        # check if the fruit is being eaten
        if self.fruit.is_on_board():
            # if so, remove the fruit, increment score, tail size and
            # refresh rate by 5%
            if self.snake.get_position() == self.fruit.pos:
...
                self.running_time_coeff *= 1.05
...
        # schedule next update event in one turn
        Clock.schedule_once(self.update, 1 / self.running_time_coeff)
```

Complete the OptionsPopup so that it passes on its values when we dismiss it :

```
class OptionsPopup(Popup):
    border_option_widget = ObjectProperty(None)
    speed_option_widget = ObjectProperty(None)

    def on_dismiss(self):
        Playground.start_speed = self.speed_option_widget.value
        Playground.border_option = self.border_option_widget.active
```

Full code.

My friends, I think we're ready to install our app on our favorite Android smartphone ! Mine is a Sony xperia z3 compact if ever you were wondering. I really that this size of screen. Anyhow, I'm digressing.

Packaging

If you read part 1, you know the drill.

Update the buidlozer.spec file with the title, name and domain of your app.
Set the version.
Fire up a console, navigate to your folder :

```
buildozer android debug

# I haven't had much luck with the 'deploy' option these past few days so I used adb directly for that
cd bin/
adb install adb install Ouroboros-1.0.0-debug.apk
```

Final code

Share this:

Facebook Google Twitter LinkedIn Pinterest Reddit Email

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

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

Make an anti procrastination box – with Arduino »

2 Comments

Netchose
FEBRUARY 6, 2015 AT 6:06 PM

Salut Alexis,
congratulations for your tutorial !
I found it very informative and the code is well built .
In the part 1 I think you should specify the version of Python

"this solution is not very elegant. But it works" =>I often tell me that , when I use Kivy 🙂

REPLY

alexis.matelin (Post author)
FEBRUARY 7, 2015 AT 12:01 AM

Merci beaucoup.
You're very right about the version. I'm going to correct that right now.

Since you're a kivy user, I have question : someone commented on reddit "I recommend avoiding Kivy like the plague on Android" because of its inefficiency compared to a native solution. Is that true ? Am I wasting my time learning it ?

REPLY

Leave a Reply