

Design your first game with PyGameZero!

Steps to install pygamezero (works for Windows, Mac, or Linux!)

Launch a terminal/command prompt. Google how to do this if you can't remember.

Make sure you have Python running:

Windows: > python --version

Mac/Linux: \$ python --version

Make sure you have pip (a Python package installer):

```
$ pip --version
```

If you don't have Python or pip, find instructions online for your system type for installing Python via Miniconda.

Now use pip to install pygame zero

```
$ pip install pgzero
```

Write your first pygamezero program

Create a directory in a place that you can find easily and call it pythongames (or whatever you want). Within this directory, create a subdirectory called zelda-classic (or whatever you want to call it).

Use your preferred method to create an empty file called zelda-classic.py. Make sure to save it in the zelda-classic directory. (Options: IDLE, Sublime)

Your directory structure should look like this:

```
pythongames/  
├── zelda-classic/  
│   └── zelda-classic.py
```

Use the command line to navigate to your zelda-classic directory, and run this blank file:

```
$ pgzrun zelda-classic.py
```

Congratulations! You've just built a game that does absolutely nothing! Let's change that.

Draw a blank screen and play some music:

Open up `zelda-classic.py` and add the following `pygamezero` code, exactly as you see it here:

```
WIDTH = 800
HEIGHT = 500

def draw():
    screen.clear()
```

This will draw a blank screen that is 800 pixels wide and 500 pixels tall. Black is the default background color.

Run the file with `pygame zero`:

```
$ pgzrun zelda-classic.py
```

Well that's the most boring game ever!

Let's make it more interesting by adding some music. Create a `music` subdirectory within the main directory:

```
pythongames/
├── zelda-classic/
│   ├── zelda-classic.py
│   └── music/
```

You should have been given a file called `overworld.ogg`. Put that file inside of the `music` subdirectory.

```
pythongames/
├── zelda-classic/
│   ├── zelda-classic.py
│   └── music/
│       └── overworld.ogg
```

One line of code is enough for `pygamezero` to play the music for you on infinite repeat. Add this line somewhere in your code:

```
music.play("overworld.ogg")
```

Run your program again with `$ pgzrun zelda-classic.py` to see what happens.

Put an actor on the screen

You should have been given a file called `littledude.png`. Create an `images` subdirectory in your app (next to `music`) and put `littledude` in that directory:

```
pythongames/
├── zelda-classic/
│   ├── images/
│   │   └── littledude.png
│   ├── zelda-classic.py
│   └── music/
│       └── overworld.ogg
```

Pygamezero automatically knows that you keep images in the `images` directory. This is convenient.

To put our little guy on the screen in our game, we create an `Actor` object that uses his image and call it something unique. We'll call him `'littleguy'` for now. Add this code somewhere above the `draw()` function definition.

```
littleguy = Actor('littledude')
```

Below this line, we can tell the pygamezero engine where we want our little guy to start on the screen by specifying his position. Add this code just below the instantiation (first creation) of `littleguy`:

```
littleguy.pos = WIDTH/2, HEIGHT/2
```

Halfway across the width and halfway down the height, hmm, where do you think `littleguy` will be on the screen?

To make sure that `littleguy` is rendered to the screen, add the following code inside the `draw()` function definition:

```
littleguy.draw()
```

Run your code with `pgzrun` and see what happens!

Change littleguy's name

Anyone who has played Zelda knows that littleguy is actually our favorite hero Link, so let's change his name to Link everywhere. Be careful about one thing: pygamezero only likes its names to be lowercase.

Change the file structure so that it looks like this:

```
pythongames/
├── zelda-classic/
│   ├── images/
│   │   └── link.png
│   ├── zelda-classic.py
│   └── music/
│       └── overworld.ogg
```

and change every reference in zelda-classic.py from littledude to link. Your code should look something like this now:

```
music.play("overworld.ogg")
```

```
WIDTH = 800
HEIGHT = 600
```

```
link = Actor('link')
link.pos = WIDTH/2, HEIGHT/2
```

```
def draw():
    screen.clear()
    link.draw()
```

Run the code to make sure everything is still okay...

Add keyboard inputs to move link left and right

Pygamezero makes it pretty easy to build a "game loop", the underlying logic of a computer game. Basically the program draws to the screen, accepts input from the user (for example, a button pushed), makes decisions based on the input, and then draws to the screen again.

Pygamezero's game loop does whatever you tell it to inside the `update()` function.

Here's how we make it possible to move link around on the screen using keyboard buttons:

```
def update():
    if keyboard.left:
        animate(link, pos=(link.x - 100, link.y))
```

If we add this to the bottom of our `zelda-classic.py` code, when we run the program, link's image will be animated to move 100 pixels to the left from its original position.

Try it! Notice that you still can't move link to the right or up and down because we haven't written that code yet!

Now we can add the ability for link to move right:

```
def update():
    if keyboard.left:
        animate(link, pos=(link.x - 100, link.y))
    elif keyboard.right:
        animate(link, pos=(link.x + 100, link.y))
```

Try it!

This is awfully slow motion, isn't it?

Let's change the duration of the animation by defining a `duration` variable which we will pass as a keyword argument to the `animate()` function.

Somewhere up near the top of your code, add this line:

```
duration = 0.05
```

Then change the `update()` function to include this `duration` (measured in seconds) inside the `animate()` method:

```
def update():
    if keyboard.left:
        animate(link, duration = duration, pos=(link.x - 100, link.y))
    elif keyboard.right:
        animate(link, duration = duration, pos=(link.x + 100, link.y))
```

Run the code with `$ pgzrun zelda-classic.py`. What happens if the `duration` is longer or shorter?

Customize how far Lnk moves with each keystroke

100 pixels might be too far for our actor to move on the screen with just one keystroke. We also want to make it easy to modify this distance, so just like we did with the duration variable, we can create a distance variable that is used to calculate the new position of our actor after a keystroke.

Just above or below duration, create a distance variable using the following code:

```
distance = 100
```

Now modify the `animate()` function calls within `update()` by replacing '100' with `distance`:

```
def update():
    if keyboard.left:
        animate(link, duration = duration, pos=(link.x - distance, link.y))
    elif keyboard.right:
        animate(link, duration = duration, pos=(link.x + distance, link.y))
```

Run the code. Play with `distance` until you find movement that you like.

Add up and down functionality

Add `elif` (else if) statements to the `update` function to allow the actor to move up and down. You'll want to keep the `x` position constant and only modify the `y` position with the `distance` parameter. Obviously, you want the actor to move up after you press the up arrow, and down after you press the down arrow!

Add a background

You should have been given an image file called `overworld.png`. Put this in the `images` folder.

One line of code is enough to draw this background image in your game!

The `screen` object has a method called `blit()` that allows you to draw layered images on the screen. Blit stands for *bit block transfer*, a method used for combining two or more bitmaps into one.

Add this function call:

```
screen.blit('overworld', (0, 0))
```

to the `draw()` function definition to give you:

```
def draw():
    screen.clear()
    screen.blit('overworld', (0, 0))
    link.draw()
```

Run the program with `$ pgzrun zelda-classic.py`.

What improvements can you see that need to be made? (there are many... but this should get you started!)