Pygame Zero Documentation

Vydanie 1.2

Daniel Pope

25. máj 2019

Obsah

Pygame Zero je knižnica na rýchlu tvorbu hier.

Knižnica je určená na použitie vo vzdelávaní, aby učitelia mohli vyučovať základy programovania bez nutnosti vysvetľovať Pygame API alebo písať hernú slučku.

KAPITOLA 1

Návody

1.1 Úvod do Pygame Zero

1.1.1 Vytvorenie okna

Najprv vytvorte prázdny súbor s názvom `intro.py`.

Uistite sa, že sa po spustení nasledujúceho príkazu vytvorí prázdne okno

```
pgzrun intro.py
```

Všetko v knižnici Pygame Zero je voliteľné; prázdny súbor je valídnym skriptom Pygame Zero!

Hru môžete ukončiť kliknutím na ikonu pre zatvorenie okna alebo stlačením kláves `Ctrl-Q` (`-Q` na Mac-u). Ak z nejakého dôvodu prestane hra reagovať, môžete ju ukončiť stlačením `Ctrl-C` v okne vášho terminálu.

1.1.2 Vykreslenie pozadia

Teraz pridáme funkciu *`draw()`* a nastavíme rozmery okna. Pygame Zero bude túto funkciu volať zakaždým, keď bude potrebovať kresliť na obrazovku.

Do súboru `intro.py` pridajte tieto riadky:

```
1 WIDTH = 300
2 HEIGHT = 300
3
4 def draw():
5     screen.fill((128, 0, 0))
```

Znovu spustite príkaz `pgzrun intro.py` čím sa obrazovka zmení na červený štvorec!

Ako tento kód pracuje?

Premenné `WIDTH` a `HEIGHT` nastavujú šírku a výšku okna. Tento kód teda nastaví šírku aj výšku okna na 300 pixelov.

`screen` je zabudovaný objekt, ktorý reprezentuje plochu okna. Obsahuje *množstvo metód pre vykreslovanie sprite-ov a tvarov*. Zavolaním metódy `screen.fill()` dôjde k vyfarbeniu obrazovky jednou farbou, ktorá je definovaná ako trojica `(červená, zelená, modrá)`. `(128, 0, 0)` bude stredne tmavočervená. Vyskúšajte si meniť hodnoty týchto čísiel v rozsahu od 0 do 255 a sledujte, aké farby dokážete vytvoriť.

Poďme nastaviť sprite, ktorý môžeme animovať.

1.1.3 Kreslíme sprite

Ešte predtým, ako čokoľvek nakreslíme, potrebujeme stiahnuť sprite s mimozemšťanom. Kliknite na neho pravým tlačidlom a uložte ho (Vyberte „Uložit obrázok ako…“ alebo niečo podobné).

(Tento sprite je priesvitný (má „alpha“ kanál), čo je pre hry skvelé! Je ale navrhnutý pre tmavé pozadie, takže pokiaľ sa v hre nezobrazí, nemusíte vidieť mimozemšťanovu prilbu).

Tip: Množstvo voľných spritov, vrátane tohto, môžete nájsť na stránke kenney.nl. Tento sprite pochádza z balíka Platformer Art Deluxe pack.

Súbor potrebujete uložiť na správne miesto, aby ho knižnica Pygame Zero vedela nájsť. Vytvorte priečinok s názvom `images` a obrázok do neho ulože pod názvom `alien.png`. Názvy oboch musia byť malými písmenami. V opačnom prípade sa bude knižnica sťažovať, aby vás upozornila na potenciálny problém v multiplatformnej kompatibilite.

Ak ste to urobili, váš projekt by mal vyzerať takto:

```
.
├── images/
│   └── alien.png
└── intro.py
```

`images/` je štandardný priečinok, ktorý bude Pygame Zero používať na hľadanie vašich obrázkov.

V knižnici sa nachádza zabudovaná trieda s názvom `Actor`, ktorú môžete použiť na reprezentovanie grafiky, ktorá sa má vykresliť na obrazovku.

Poďme jednu vytvoriť. Upravte súbor `intro.py` takto:

```
1 alien = Actor('alien')
2 alien.pos = 100, 56
3
4 WIDTH = 500
5 HEIGHT = alien.height + 20
6
7 def draw():
8     screen.clear()
9     alien.draw()
```

Váš mimozemšťan by sa mal teraz zobraziť na obrazovke! Odovzdaním reťazca `'alien'` do konštruktora triedy `Actor` sa automaticky nahrá sprite a má k dispozícii atribúty ako pozícia a rozmer. To nám umožní nastaviť `HEIGHT` okna na základe výšky mimozemšťana.

Metóda `alien.draw()` vykreslí sprite na obrazovku na jeho aktuálnu pozíciu.

1.1.4 Hýbeme s mimozemšťanom

Let's set the alien off-screen; zmeňte riadok obsahujúci `alien.pos` na:

```
alien.topright = 0, 10
```

Všimnite si, ako sa priradením hodnoty do `topright` zmení poloha mimozemšťana vzhľadom na jeho pravý horný roh. If the right-hand edge of the alien is at `0`, the alien is just offscreen to the left. Teraz ho poďme rozpohybovať. Pridajte tieto riadky na koniec súboru:

```
def update():
    alien.left += 2
    if alien.left > WIDTH:
        alien.right = 0
```

Pygame Zero zavolá vašu funkciu *`update()`* raz za každý snímok. Posunutím mimozemšťana o niekoľko pixelov počas každého snímku spôsobí, že sa bude po obrazovke presúvať. Keď dosiahne pravý okraj obrazovky, znovu ho spustíme od ľavého okraja.

Vaše funkcie `draw()` a `update()` pracujú veľmi podobne, ale sú navrhnuté pre dva rozličné účely. Funkcia `draw()` zobrazí mimozemšťana na jeho aktuálnu pozíciu, zatiaľ čo funkcia `update()` sa použije na aktualizovanie pohybu mimozemšťana po obrazovke.

1.1.5 Ošetrenie kliknutí

Teraz poďme zabezpečiť, aby hra urobila niečo po kliknutí na mimozemšťana. Aby sme to mohli spraviť, potrebujeme zadefinovať funkciu s názvom *`on_mouse_down()`*. Pridajte do kódu tieto riadky:

```
1 def on_mouse_down(pos):
2     if alien.collidepoint(pos):
3         print("Au!")
4     else:
5         print("Netrafil si!")
```

Spustite hru a skúste na mimozemšťana klikať.

Knižnica Pygame Zero je pomerne chytrá čo sa týka volania vašich funkcií. Ak v definícii vašej funkcie nepoužijete parameter `pos`, Pygame Zero ju bude volať bez pozície. Funkcia `on_mouse_down` taktiež obsahuje parameter `button`. Takže ju môžete napísať takto:

```
def on_mouse_down():
    print("Klikol si!")
```

alebo takto:

```
def on_mouse_down(pos, button):
    if button == mouse.LEFT and alien.collidepoint(pos):
        print("Au!")
```

1.1.6 Zvuky a obrázky

Teraz poďme zabezpečiť, aby mimozemšťan vyzeral zranene. Uložte si tieto súbory:

- alien_hurt.png - súbor uložte ako `alien_hurt.png` do priečinku `images`.
- eep.wav - vytvorte priečinok s názvom `sounds` a uložte do neho tento súbor ako `eep.wav`.

Váš projekt by mal teraz vyzerať takto:

```
.
├── images/
│   └── alien.png
│   └── alien_hurt.png
├── sounds/
│   └── eep.wav
└── intro.py
```

`sounds/` je štandardný priečinok, v ktorom bude knižnica Pygame Zero hľadať vaše zvukové súbory.

Teraz zmeňme funkciu `on_mouse_down` tak, aby použila tieto nové zdroje:

```
def on_mouse_down(pos):
    if alien.collidepoint(pos):
        alien.image = 'alien_hurt'
        sounds.eep.play()
```

Ak teraz kliknete na mimozemšťana, mali by ste počuť zvuk a sprite sa zmení na nešťastného mimozemšťana.

V hre sa však nachádza chyba; mimozemšťan sa už nikdy nezmení späť na šťastného (zvuk sa však vždy po kliknutí prehrá). Túto chybu preto hneď opravíme.

1.1.7 Clock

Ak poznáte Python mimo programovania počítačových hier, určite poznáte metódu `time.sleep()`, ktorá pozastaví vykonávanie programu. To vás môže zvádzať k napísaniu takéhoto programu:

```
1 def on_mouse_down(pos):
2     if alien.collidepoint(pos):
3         alien.image = 'alien_hurt'
4         sounds.eep.play()
5         time.sleep(1)
6         alien.image = 'alien'
```

Žiaľ, tento prístup nie je vôbec vhodný pre použitie v počítačových hrách. Metóda `time.sleep()` blokuje všetky činnosti a my chceme, aby hra pokračovala v behu a veci sa hýbali. Vlastne sa len potrebujeme vrátiť z funkcie `on_mouse_down` a nechať hru rozhodnúť, kedy resetuje mimozemšťana ako súčasť jej bežnej práce počas vykonávania vašich metód `draw()` a `update()`.

To je však nie je zložité vyriešiť s knižnicou Pygame Zero, pretože obsahuje zabudovanú triedu class:*Clock*, ktorá umožňuje naplánovať spúšťanie funkcií na neskôr.

Najprv „refaktorujme“ (to znamená reorganizujme) kód. Môžeme vytvoriť funkcie, ktoré nastavia mimozemšťana na zraneného a rovnako ho vrátia naspäť do normálu:

```
1 def on_mouse_down(pos):
2     if alien.collidepoint(pos):
3         set_alien_hurt()
4
5
6 def set_alien_hurt():
7     alien.image = 'alien_hurt'
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
    sounds.eep.play()


def set_alien_normal():
    alien.image = 'alien'
```

Zatiaľ však k žiadnej zmene nedôjde, pretože funkcia `set_alien_normal()` sa nikde nevolá. Ale zmeňme funkciu `set_alien_hurt()` tak, aby používala objekt `clock`, takže funkcia `set_alien_normal()` sa bude volať o chvíľu neskôr.:

```
def set_alien_hurt():
    alien.image = 'alien_hurt'
    sounds.eep.play()
    clock.schedule_unique(set_alien_normal, 0.5)
```

Metóda `clock.schedule_unique()` spôsobí, že funkcia `set_alien_normal()` sa zavolá po uplynutí `0.5` sekundy. Metóda `clock.schedule_unique()` taktiež zabráni tomu, aby rovnaká funkcia bola naplánovaná viackrát, ako napríklad keď budete klikať veľmi rýchlo.

Vyskúšajte to a uvidíte, že sa mimozemšťan vráti do normálu po uplynutí 0.5 sekundy. Skúste rýchlo klikať a overte, že sa mimozemšťan nevráti po 0.5 sekunde od posledného kliknutia. rapidly and verify that the alien doesn't revert until 0.5 second after the last click.

`clock.schedule_unique()` akceptuje pre zadanie časového intervalu ako celé tak aj desatinné čísla. V tomto návode sme použili desatinné čísla, aby to bolo vidieť, ale neváhajte použiť obe, aby ste videli rozdiel a dopad, ktorý majú rozličné hodnoty.

1.1.8 Zhrnutie

Ukázali sme si, ako nahrať a zobraziť sprity, prehrávať zvuky, ošetriť vstupné udalosti a ako sa používajú zabudované hodiny.

Môžete hru rozšíriť o skóre alebo môžete upraviť mimozemšťana tak, aby sa pohyboval viac náhodne.

Knižnica Pygame Zero obsahuje množstvo ďalších zabudovaných vlastností, ktoré robia prácu s ňou jednoduchou. Ak sa chcete naučiť aj zvyšné API, navštívte stránku o *zabudovaných objektoch*.

1.2 Prechod z nástroja Scratch

Tento návod porovná implementáciu hry Flappy Bird napísanú v Scratch-i s implementáciou v Pygame Zero. The Scratch and Pygame Zero programs are similar to a remarkable extent.

The Pygame Zero version can be found in Pygame Zero repository.

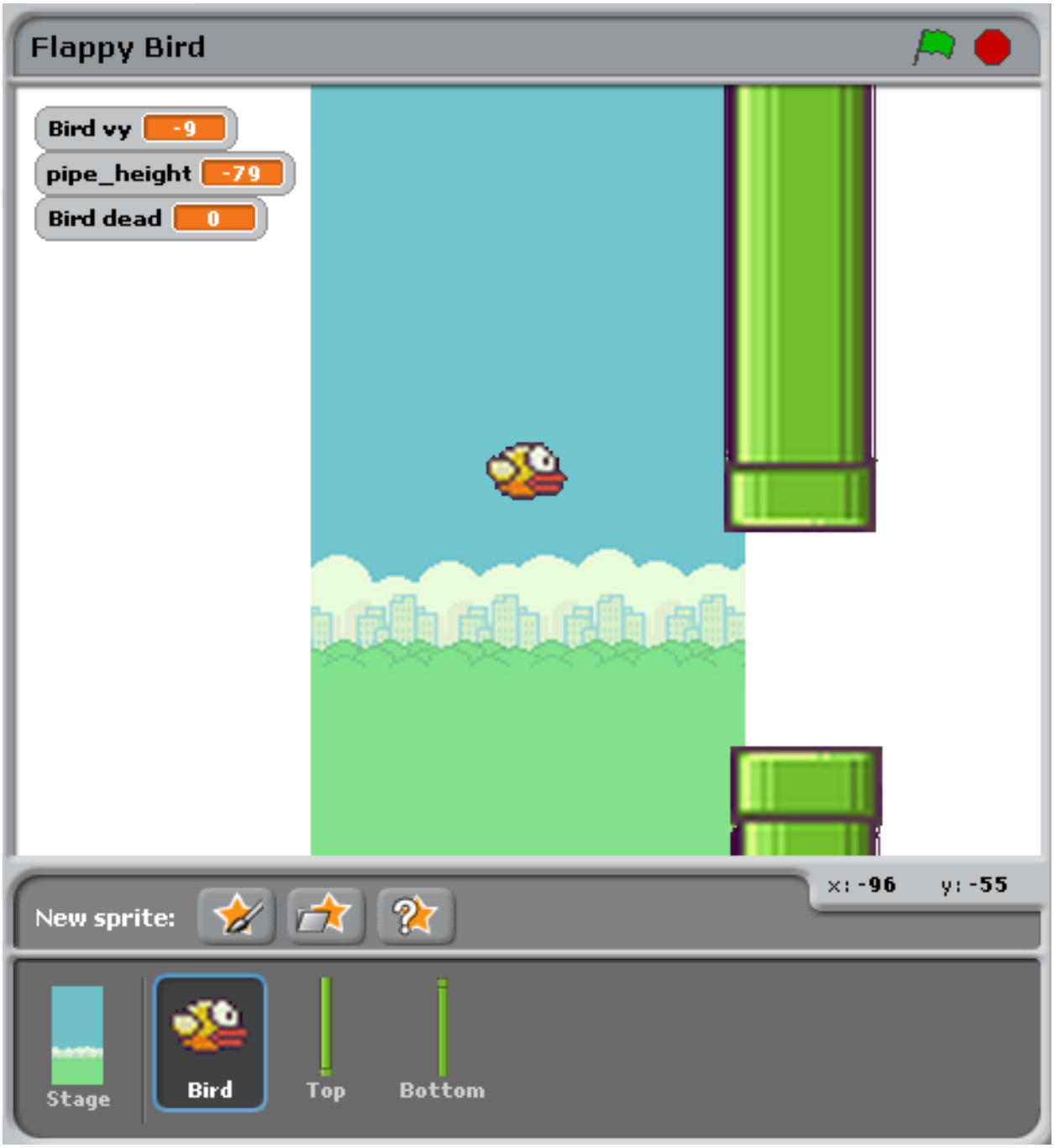
You can also download the Scratch version from the repository.

The Pygame Zero version includes scoring logic, which is omitted in the code examples on this page to make it a closer comparison.

The Python code shown below is re-arranged for clarity within the examples.

1.2.1 The stage

Here's how the stage is laid out in our Scratch program:



There are just three objects, aside from the background: the bird, and the top and bottom pipes.

This corresponds to the Pygame Zero code setting these objects up as `Actors`:

```
bird = Actor('bird1', (75, 200))
pipe_top = Actor('top', anchor=('left', 'bottom'))
pipe_bottom = Actor('bottom', anchor=('left', 'top'))
```

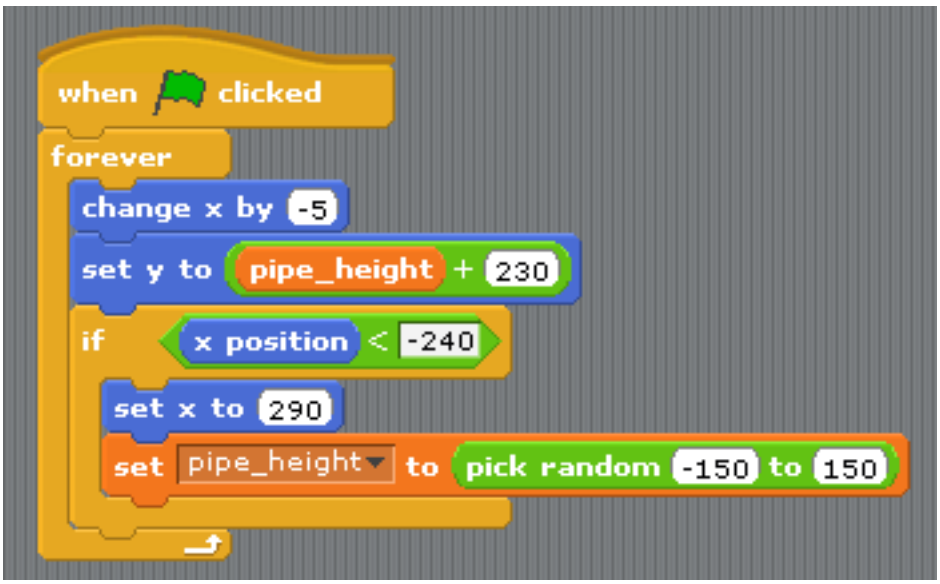
V Pygame Zero je ešte potrebné zabezpečiť vykreslenie týchto objektov. To v princípe poskytuje viac flexibility v tom, ako vykresliť scénu:

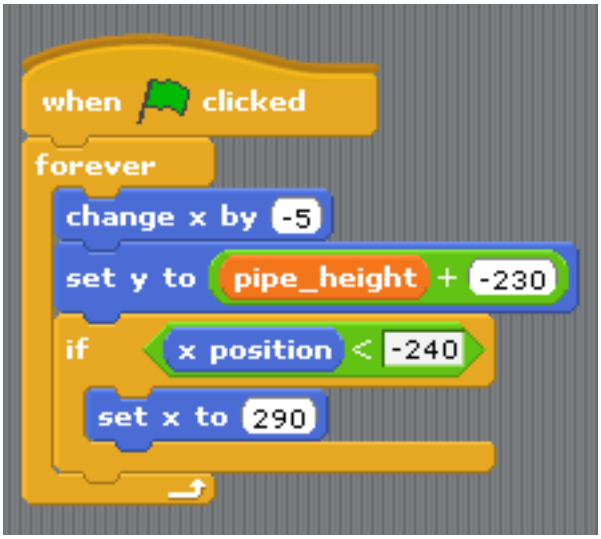
```
def draw():
    screen.blit('background', (0, 0))
    pipe_top.draw()
    pipe_bottom.draw()
    bird.draw()
```

1.2.2 Pohyb potrubia

Potrubie sa pohybuje konštantnou rýchlosťou bez ohľadu na vtáka. Keď zmizne z obrazovky na ľavej strane, znovu sa objaví na pravej strane, pričom jeho vertikálna pozícia bude náhodná.

V Scratch-i je to možné zabezpečiť dvoma rozličnými skriptami pre vrchné a dolné potrubie.





To summarise what's happening here:

- The condition `x position < -240` is true when a pipe is off the left-hand side of the screen, and this is the trigger to reset the pipes.

- The `pipe_height` variable is used to coordinate the two pipes. Because the gap between them should remain the same, we can't pick *both* heights randomly. Therefore one of the scripts has this logic and the other doesn't.
- The `set y position to pipe height +/- 230` sets one pipe to be above `pipe_height` and the other pipe below `pipe_height`.

This code becomes much simpler in Pygame Zero. We could write a single function that updates both pipes. In fact I split it a different way to make it clear that the reset actions go together:

```
import random

WIDTH = 400
HEIGHT = 708
GAP = 130
SPEED = 3

def reset_pipes():
    pipe_gap_y = random.randint(200, HEIGHT - 200)
    pipe_top.pos = (WIDTH, pipe_gap_y - GAP // 2)
    pipe_bottom.pos = (WIDTH, pipe_gap_y + GAP // 2)

def update_pipes():
    pipe_top.left -= SPEED
    pipe_bottom.left -= SPEED
    if pipe_top.right < 0:
        reset_pipes()
```

A small difference here is that I can extract values that I want to re-use as „constants", spelled in UPPERCASE. This lets me change them in one place when I want to tune the game. For example, in the code above, I could widen or narrow the gap between the two pipes simply by changing `GAP`.

The biggest thing that differs is that there is no `forever` loop in Python code. This is the big difference between Scratch and most text-based programming languages: you must update the game by one animation step and then return. Returning gives Pygame Zero a chance to do things like processing input or redrawing the screen. Loop forever and the game would just sit there, so any loops need to finish quickly.

Pygame Zero calls an `update()` function when it wants you to update the animation by one step, so we just need to a call to `update_walls()`:

```
def update():
   update_walls()
```

1.2.3 Vták

The patterns described above for how Scratch logic translates to Python code also apply for the bird logic. Let's look at the Python code first this time.

The code to update the bird is organised into a function called `update_bird()`. The first thing this function contains is some code to move the bird according to gravity:

```
GRAVITY = 0.3

# Initial state of the bird
bird.dead = False
bird.vy = 0

def update_bird():
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
    uy = bird.vy
    bird.vy += GRAVITY
    bird.y += bird.vy
    bird.x = 75
```

This is a simple gravity formula:

- Gravity means constant **acceleration downwards**.
- Acceleration is change in **velocity**.
- Velocity is change in **position**.

To represent this we need to track a variable `bird.vy`, which is the bird's velocity in the `y` direction. This is a new variable that we are defining, not something that Pygame Zero provides for us.

- Gravity means constant acceleration downwards: `GRAVITY` is greater than 0.
- Acceleration is change in velocity: `GRAVITY` gets added to `bird.vy`
- Velocity is change in position: `bird.vy` gets added to `bird.y`

Note that the bird does not move horizontally! Its `x` position stays at 75 through the whole game. We simulate movement by moving the pipes towards it. This looks as though it's a moving camera following the bird. So there's no need for a `vx` variable in this game.

The next section makes the bird flap its wings:

```
if not bird.dead:
    if bird.vy < -3:
        bird.image = 'bird2'
    else:
        bird.image = 'bird1'
```

This checks if the bird is moving upwards or downwards. We show the `bird2` image if it is moving upwards fast and the `bird1` image otherwise. (-3 was picked by trial and error to make this look convincing).

The next section checks if the bird has collided with a wall:

```
if bird.colliderect(pipe_top) or bird.colliderect(pipe_bottom):
    bird.dead = True
    bird.image = 'birddead'
```

If so we set `bird.dead` to `True`. This is a **boolean value** meaning it is either `True` or `False`. We can use this to easily check if the bird is alive. If it isn't alive it won't respond to player input.

And the final section checks if the bird has fallen off the bottom (or the top) of the game screen. If so it resets the bird:

```
if not 0 < bird.y < 720:
    bird.y = 200
    bird.dead = False
    bird.vy = 0
    reset_pipes()
```

What's `reset_pipes()` doing there? Because I'd organised my pipes code to be a separate function, I can just call it whenever I want to reset my walls. In this case it makes it a better game because it gives the player a chance to react when the bird moves back to its start position.

Again, this needs to be called every frame, so we add it to `update()`:

```
def update():
    update_walls()
    update_bird()
```

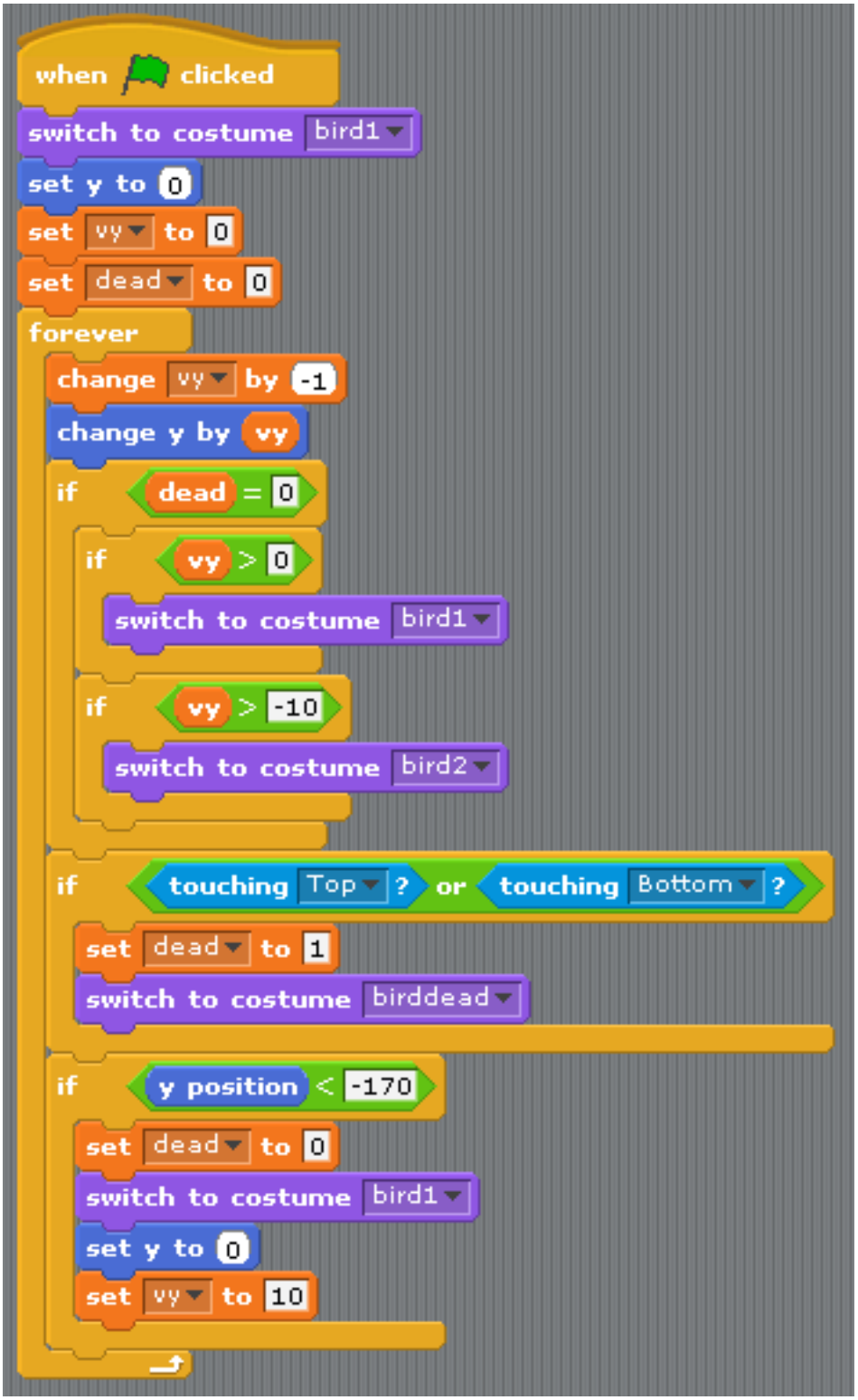
The final part of the bird logic is that it has to respond to player control. When we press a key, the bird flaps upwards. Pygame Zero will call an `on_key_down()` function - if you've defined one - whenever a key is pressed:

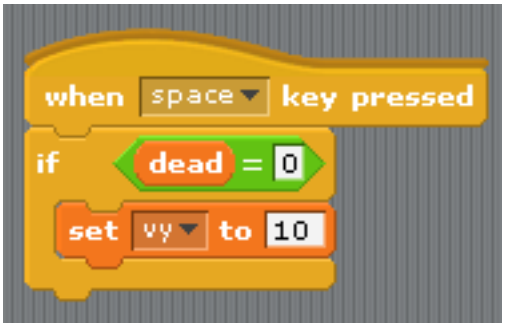
```
FLAP_VELOCITY = -6.5

def on_key_down():
    if not bird.dead:
        bird.vy = FLAP_VELOCITY
```

Here, if the bird is not dead, we set its `vy` to a negative number: in Pygame Zero this means it starts moving upwards.

Mali by ste byť schopní nájsť mnoho paralel medzi kódom v jazyku Python a týmto kódom v Scratch-i:


when clicked
switch to costume bird1
set y to 0
set vy to 0
set dead to 0
forever
change vy by -1
change y by vy
if dead = 0
if vy > 0
switch to costume bird1
if vy > -10
switch to costume bird2
if touching Top ? or touching Bottom ?
set dead to 1
switch to costume birddead
if y position < -170
set dead to 0
switch to costume bird1
set y to 0
set vy to 10



Najväčšie rozdiely medzi prostredím Scratch a Pygame Zero sú tieto:

- You cannot loop forever in Pygame Zero - just update for one frame and then return.
- The coordinates are different. In Pygame Zero, the top left of the screen is `x = 0, y = 0`. The `x` direction goes from left to right as before, but `y` goes down the screen! This is why `GRAVITY` is a positive number and `FLAP_VELOCITY` is a negative number in Python.
- `bird.dead` is a bool, so I can write code like `if not bird.dead` instead of `dead = 0` as in Scratch.

1.2.4 Zhrnutie

Množstvo konceptov dostupných v nástroji Scratch môže byť preložených priamo do knižnice Pygame Zero.

Tu je niekoľko porovnaní:

V nástroji Scratch	V knižnici Pygame Zero
`change y by 1` (up)	`bird.y -= 1`
`change y by -1` (down)	`bird.y += 1`
`set costume to <name>`	`bird.image = 'name'`
`if dead = 0`	`if not bird.dead:`
`set dead to 0`	`bird.dead = False`
`if touching Top?`	`if bird.colliderect(pipe_top)`
`When Flag clicked... forever`	Put code into the `update()` function.
`When [any] key pressed`	`def on_key_down():`
`pick random a to b`	`import random` to load the `random` module, then `random.randint(a, b)`
(0, 0) is the centre of the stage	(0, 0) je ľavý horný roh okna

V niektorých prípadoch je kód jednoduchší v jazyku Python, pretože môže byť organizovaný spôsobom, ktorý zvýši jeho čitateľnosť.

The power of Pygame Zero's actors also makes the coordinate manipulation easier. We used the `anchor` position to position the pipes, and we were able to see if a pipe was off-screen by checking `pipe_top.right < 0` rather than `if x position < -240`.

KAPITOLA 2

Reference

2.1 Event Hooks

Pygame Zero will automatically pick up and call event hooks that you define. This approach saves you from having to implement the event loop machinery yourself.

2.1.1 Game Loop Hooks

Typická herná slučka vyzerá takto:

```
while game_has_not_ended():
    process_input()
    update()
    draw()
```

Spracovanie vstupu je trochu komplikovanejšie, ale Pygame Zero umožňuje jednoducho definovať funkcie `update()` a `draw()` v module vašej hry.

draw()

Funkcia je volaná knižnicou Pygame Zero, keď potrebuje prekresliť okno vašej hry.

`draw()` nesmie mať žiadne argumenty.

Pygame Zero attempts to work out when the game screen needs to be redrawn to avoid redrawing if nothing has changed. On each step of the game loop it will draw the screen in the following situations:

- If you have defined an `update()` function (see below).
- If a clock event fires.
- If an input event has been triggered.

One way this can catch you out is if you attempt to modify or animate something within the draw function. For example, this code is wrong: the alien is not guaranteed to continue moving across the screen:

```
def draw():
    alien.left += 1
    alien.draw()
```

The correct code uses `update()` to modify or animate things and draw simply to paint the screen:

```
def draw():
    alien.draw()

def update():
    alien.left += 1
```

update(*) alebo update(dt)*

Funkcia je volaná knižnicou Pygame Zero na vykonanie jedného kroku hernej logiky vašej hry. Bude volaná opakovane 60 krát za sekundu.

Pri písaní tejto funkcie je možné použiť dva rozličné prístupy.

V jednoduchých hrách môžete predpokladať, že medzi každým volaním funkcie `update()` ubehol krátky časový úsek (zlomok sekundy). Jeho veľkosť vás možno vôbec nebude zaujímať: len budete posúvať objekty o pevný počet pixelov každý snímok (alebo ich budete zrýchlovať o pevnú konštantu, atď.)

Pokročilejší prístup znamená založiť váš pohyb a fyzické výpočty na skutočnom množstve času, ktorý uplynul medzi jednotlivými volaniami. To umožní, aby boli animácie plynulejšie, ale potrebné výpočty na ich dosiahnutie môžu byť náročnejšie a musíte dávať väčší pozor, aby ste sa vyhli nepredvídateľnému správaniu, keď sa začnú časové úseky zväčšovať.

Pre použitie prístupu založeného na čase je potrebné upraviť funkciu tak, aby obsahovala jeden parameter. V tom prípade ho Pygame Zero odovzdá funkcii vo forme uplynutého času v sekundách. To môžete využiť na správne nastavenie výpočtov vašich pohybov.

2.1.2 Event Handling Hooks

Similar to the game loop hooks, your Pygame Zero program can respond to input events by defining functions with specific names.

Somewhat like in the case of `update()`, Pygame Zero will inspect your event handler functions to determine how to call them. So you don't need to make your handler functions take arguments. For example, Pygame Zero will be happy to call any of these variations of an `on_mouse_down` function:

```
def on_mouse_down():
    print("Mouse button clicked")

def on_mouse_down(pos):
    print("Mouse button clicked at", pos)

def on_mouse_down(button):
    print("Mouse button", button, "clicked")

def on_mouse_down(pos, button):
    print("Mouse button", button, "clicked at", pos)
```

It does this by looking at the names of the parameters, so they must be spelled exactly as above. Each event hook has a different set of parameters that you can use, as described below.

on_mouse_down(*[pos][, button]*)

Called when a mouse button is depressed.

Parametre

- **pos** – A tuple (x, y) that gives the location of the mouse pointer when the button was pressed.
- **button** – A *mouse* enum value indicating the button that was pressed.

on_mouse_up([*pos*][, *button*])
Called when a mouse button is released.

Parametre

- **pos** – A tuple (x, y) that gives the location of the mouse pointer when the button was released.
- **button** – A *mouse* enum value indicating the button that was released.

on_mouse_move([*pos*][, *rel*][, *buttons*])
Called when the mouse is moved.

Parametre

- **pos** – A tuple (x, y) that gives the location that the mouse pointer moved to.
- **rel** – A tuple (delta_x, delta_y) that represent the change in the mouse pointer's position.
- **buttons** – A set of *mouse* enum values indicating the buttons that were depressed during the move.

To handle mouse drags, use code such as the following:

```
def on_mouse_move(rel, buttons):
    if mouse.LEFT in buttons:
        # the mouse was dragged, do something with `rel`
        ...
```

on_key_down([*key*][, *mod*][, *unicode*])
Called when a key is depressed.

Parametre

- **key** – An integer indicating the key that was pressed (see *below*).
- **unicode** – Where relevant, the character that was typed. Not all keys will result in printable characters - many may be control characters. In the event that a key doesn't correspond to a Unicode character, this will be the empty string.
- **mod** – A bitmask of modifier keys that were depressed.

on_key_up([*key*][, *mod*])
Called when a key is released.

Parametre

- **key** – An integer indicating the key that was released (see *below*).
- **mod** – A bitmask of modifier keys that were depressed.

on_music_end()
Called when a *music track* finishes.

Note that this will not be called if the track is configured to loop.

Tlačidlá a klávesy

Built-in objects `mouse` and `keys` can be used to determine which buttons or keys were pressed in the above events.

Note that mouse scrollwheel events appear as button presses with the below `WHEEL_UP`/`WHEEL_DOWN` button constants.

class mouse

A built-in enumeration of buttons that can be received by the `on_mouse_*` handlers.

LEFT

MIDDLE

RIGHT

WHEEL_UP

WHEEL_DOWN

class keys

A built-in enumeration of keys that can be received by the `on_key_*` handlers.

BACKSPACE

TAB

CLEAR

RETURN

PAUSE

ESCAPE

SPACE

EXCLAIM

QUOTEDBL

HASH

DOLLAR

AMPERSAND

QUOTE

LEFTPAREN

RIGHTPAREN

ASTERISK

PLUS

COMMA

MINUS

PERIOD

SLASH

K_0

K_1

K_2

K_3

K_4

K_5

K_6

K_7

K_8

K_9

COLON

SEMICOLON

LESS

EQUALS

GREATER

QUESTION

AT

LEFTBRACKET

BACKSLASH

RIGHTBRACKET

CARET

UNDERSCORE

BACKQUOTE

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

DELETE

KP0

KP1

KP2

KP3

KP4

KP5

KP6

KP7

KP8

KP9

KP_PERIOD

KP_DIVIDE

KP_MULTIPLY

KP_MINUS

KP_PLUS

KP_ENTER

KP_EQUALS

UP

DOWN

RIGHT

LEFT

INSERT

HOME

END

PAGEUP

PAGEDOWN

F1

F2

F3

F4

F5

F6

F7

F8

F9

F10

F11

F12

F13

F14

F15

NUMLOCK

CAPSLOCK

SCROLLOCK

RSHIFT

LSHIFT

RCTRL

LCTRL

RALT

LALT

RMETA

LMETA

LSUPER

RSUPER

MODE

HELP

PRINT

SYSREQ

BREAK

MENU

POWER

EURO

LAST

Additionally you can access a set of constants that represent modifier keys:

class keymods

Constants representing modifier keys that may have been depressed during an `on_key_up`/`on_key_down` event.

LSHIFT

RSHIFT

SHIFT

LCTRL

RCTRL

CTRL

LALT

RALT

ALT

LMETA

RMETA

META

NUM

CAPS

MODE

2.2 Zabudované objekty

Pygame Zero poskytuje užitočné zabudované objekty, ktoré vám pomôžu vytvárať vaše hry jednoduchšie.

2.2.1 Screen

Text Formatting

The *Screen*'s `draw.text()` method has a very rich set of methods for position and formatting of text. Some examples:

```
screen.draw.text("Text color", (50, 30), color="orange")
screen.draw.text("Font name and size", (20, 100), fontname="Boogaloo", fontsize=60)
screen.draw.text("Positioned text", topright=(840, 20))
screen.draw.text("Allow me to demonstrate wrapped text.", (90, 210), width=180,␣
↪lineheight=1.5)
screen.draw.text("Outlined text", (400, 70), owidth=1.5, ocolor=(255,255,0), color=(0,
↪0,0))
screen.draw.text("Drop shadow", (640, 110), shadow=(2,2), scolor="#202020")
screen.draw.text("Color gradient", (540, 170), color="red", gcolor="purple")
```

(pokračuje na ďalšej strane)

(pokračovanie z predošlej strany)

```
screen.draw.text("Transparency", (700, 240), alpha=0.1)
screen.draw.text("Vertical text", midleft=(40, 440), angle=90)
screen.draw.text("All together now:\nCombining the above options",
    midbottom=(427,460), width=360, fontname="Boogaloo", fontsize=48,
    color="#AAFF00", gcolor="#66AA00", owidth=1.5, ocolor="black", alpha=0.8)
```

In its simplest usage, `screen.draw.text` requires the string you want to draw, and the position. You can either do this by passing coordinates as the second argument (which is the top left of where the text will appear), or use the positioning keyword arguments (described later):

```
screen.draw.text("hello world", (20, 100))
```

`screen.draw.text` takes many optional keyword arguments, described below.

Font name and size

Fonts are loaded from a directory named `fonts`, in a similar way to the handling of images and sounds. Fonts must be in `.ttf` format. For example:

```
screen.draw.text("hello world", (100, 100), fontname="Viga", fontsize=32)
```

Keyword arguments:

- `fontname`: filename of the font to draw. By default, use the system font.
- `fontsize`: size of the font to use, in pixels. Defaults to `24`.
- `antialias`: whether to render with antialiasing. Defaults to `True`.

Color and background color

```
screen.draw.text("hello world", (100, 100), color=(200, 200, 200), background="gray")
```

Keyword arguments:

- `color`: foreground color to use. Defaults to `white`.
- `background`: background color to use. Defaults to `None`.

`color` (as well as `background`, `ocolor`, `scolor`, and `gcolor`) can be an (r, g, b) sequence such as `(255,127,0)`, a `pygame.Color` object, a color name such as `"orange"`, an HTML hex color string such as `"#FF7F00"`, or a string representing a hex color number such as `"0xFF7F00"`.

`background` can also be `None`, in which case the background is transparent. Unlike `pygame.font.Font.render`, it's generally not more efficient to set a background color when calling `screen.draw.text`. So only specify a background color if you actually want one.

Colors with alpha transparency are not supported (except for the special case of invisible text with outlines or drop shadows - see below). See the `alpha` keyword argument for transparency.

Positioning

```
screen.draw.text("hello world", centery=50, right=300)
screen.draw.text("hello world", midtop=(400, 0))
```

Keyword arguments:

```
top left bottom right
topleft bottomleft topright bottomright
midtop midleft midbottom midright
center centerx centery
```

Positioning keyword arguments behave like the corresponding properties of `pygame.Rect`. Either specify two arguments, corresponding to the horizontal and vertical positions of the box, or a single argument that specifies both.

If the position is overspecified (e.g. both `left` and `right` are given), then extra specifications will be (arbitrarily but deterministically) discarded. For constrained text, see the section on `screen.draw.textbox` below.

Word wrap

```
screen.draw.text("splitting\nlines", (100, 100))
screen.draw.text("splitting lines", (100, 100), width=60)
```

Keyword arguments:

- `width`: maximum width of the text to draw, in pixels. Defaults to `None`.
- `widthem`: maximum width of the text to draw, in font-based em units. Defaults to `None`.
- `lineheight`: vertical spacing between lines, in units of the font's default line height. Defaults to `1.0`.

`screen.draw.text` will always wrap lines at newline (`\n`) characters. If `width` or `widthem` is set, it will also try to wrap lines in order to keep each line shorter than the given width. The text is not guaranteed to be within the given width, because wrapping only occurs at space characters, so if a single word is too long to fit on a line, it will not be broken up. Outline and drop shadow are also not accounted for, so they may extend beyond the given width.

You can prevent wrapping on a particular space with non-breaking space characters (`\u00A0`).

Text alignment

```
screen.draw.text("hello\nworld", bottomright=(500, 400), align="left")
```

Keyword argument:

- `align`: horizontal positioning of lines with respect to each other. Defaults to `None`.

`align` determines how lines are positioned horizontally with respect to each other, when more than one line is drawn. Valid values for `align` are the strings `"left"`, `"center"`, or `"right"`, a numerical value between `0.0` (for left alignment) and `1.0` (for right alignment), or `None`.

If `align` is `None`, the alignment is determined based on other arguments, in a way that should be what you want most of the time. It depends on any positioning arguments (`topleft`, `centerx`, etc.), `anchor`, and finally defaults to `"left"`. I suggest you generally trust the default alignment, and only specify `align` if something doesn't look right.

Outline

```
screen.draw.text("hello world", (100, 100), owidth=1, ocolor="blue")
```

Keyword arguments:

- owidth: outline thickness, in outline units. Defaults to None.
- ocolor: outline color. Defaults to "black".

The text will be outlined if owidth is specified. The outlining is a crude manual method, and will probably look bad at large sizes. The units of owidth are chosen so that 1.0 is a good typical value for outlines. Specifically, they're the font size divided by 24.

As a special case, setting color to a transparent value (e.g. (0,0,0,0)) while using outilnes will cause the text to be invisible, giving a hollow outline. (This feature is not compatible with gcolor.)

Valid values for ocolor are the same as for color.

Drop shadow

```
screen.draw.text("hello world", (100, 100), shadow=(1.0,1.0), scolor="blue")
```

Keyword arguments:

- shadow: (x,y) values representing the drop shadow offset, in shadow units. Defaults to None.
- scolor: drop shadow color. Defaults to "black".

The text will have a drop shadow if shadow is specified. It must be set to a 2-element sequence representing the x and y offsets of the drop shadow, which can be positive, negative, or 0. For example, shadow=(1.0,1.0) corresponds to a shadow down and to the right of the text. shadow=(0,-1.2) corresponds to a shadow higher than the text.

The units of shadow are chosen so that 1.0 is a good typical value for the offset. Specifically, they're the font size divided by 18.

As a special case, setting color to a transparent value (e.g. (0,0,0,0)) while using drop shadow will cause the text to be invisible, giving a hollow shadow. (This feature is not compatible with gcolor.)

Valid values for scolor are the same as for color.

Gradient color

```
screen.draw.text("hello world", (100, 100), color="black", gcolor="green")
```

Keyword argument:

- gcolor: Lower gradient stop color. Defaults to None.

Specify gcolor to color the text with a vertical color gradient. The text's color will be color at the top and gcolor at the bottom. Positioning of the gradient stops and orientation of the gradient are hard coded and cannot be specified.

Requries pygame.surfarray module, which uses numpy or Numeric library.

Alpha transparency

```
screen.draw.text("hello world", (100, 100), alpha=0.5)
```

Keyword argument:

- alpha: alpha transparency value, between 0 and 1. Defaults to 1.0.

In order to maximize reuse of cached transparent surfaces, the value of alpha is rounded.

Requires pygame.surfarray module, which uses numpy or Numeric library.

Anchored positioning

```
screen.draw.text("hello world", (100, 100), anchor=(0.3,0.7))
```

Keyword argument:

- `anchor`: a length-2 sequence of horizontal and vertical anchor fractions. Defaults to `(0.0, 0.0)`.

`anchor` specifies how the text is anchored to the given position, when no positioning keyword arguments are passed. The two values in `anchor` can take arbitrary values between `0.0` and `1.0`. An `anchor` value of `(0,0)`, the default, means that the given position is the top left of the text. A value of `(1,1)` means the given position is the bottom right of the text.

Rotation

```
screen.draw.text("hello world", (100, 100), angle=10)
```

Keyword argument:

- `angle`: counterclockwise rotation angle in degrees. Defaults to `0`.

Positioning of rotated surfaces is tricky. When drawing rotated text, the anchor point, the position you actually specify, remains fixed, and the text rotates around it. For instance, if you specify the top left of the text to be at `(100, 100)` with an angle of `90`, then the Surface will actually be drawn so that its bottom left is at `(100, 100)`.

If you find that confusing, try specifying the center. If you anchor the text at the center, then the center will remain fixed, no matter how you rotate it.

In order to maximize reuse of cached rotated surfaces, the value of `angle` is rounded to the nearest multiple of 3 degrees.

Constrained text

```
screen.draw.textbox("hello world", (100, 100, 200, 50))
```

`screen.draw.textbox` requires two arguments: the text to be drawn, and a `pygame.Rect` or a `Rect`-like object to stay within. The font size will be chosen to be as large as possible while staying within the box. Other than `fontsize` and positional arguments, you can pass all the same keyword arguments to `screen.draw.textbox` as to `screen.draw.text`.

Objekt `screen` reprezentuje vašu hernú obrazovku.

It is a thin wrapper around a Pygame surface that allows you to easily draw images to the screen („blit“ them).

class Screen

surface
Referencia na Pygame surface, ktorý reprezentuje buffer obrazovky. Môžete ho použiť na pokročilé grafické úlohy.

clear()
Resetuje obrazovku na čierno.

fill(*(red, green, blue)*)
Vyplní obrazovku farbou.

blit (*image*, *(left*, *top)*)
Vykreslí obrázok na obrazovku na zadanú pozíciu.

`blit()` akceptuje buď `Surface` alebo reťazec ako jej parameter `image`. Ak je `image` reťazec (`str`), tak sa nahrá obrázok s týmto názvom z priečinku `images/`.

`draw.`**line** (*start*, *end*, *(r*, *g*, *b)*)
Nakreslí čiaru od počiatočného bodu (`start`) po koncový (`end`).

`draw.`**circle** (*pos*, *radius*, *(r*, *g*, *b)*)
Nakreslí kružnicu.

`draw.`**filled_circle** (*pos*, *radius*, *(r*, *g*, *b)*)
Nakreslí kruh.

`draw.`**rect** (*rect*, *(r*, *g*, *b)*)
Nakreslí obrys obdĺžnika.

Očakáva *Rect*.

`draw.`**filled_rect** (*rect*, *(r*, *g*, *b)*)
Nakreslí vyplnený obdĺžnik.

`draw.`**text** (*text*[, *pos*], ***kwargs*)
Nakreslí text.

K dispozícii je bohaté API pre umiestňovanie a formátovanie textu; pre zobrazenie kompletnej dokumentácie navštívte *Text Formatting*.

`draw.`**textbox** (*text*, *rect*, ***kwargs*)
Nakreslí text, ktorý vyplní daný *Rect*.

K dispozícii je bohaté API pre formátovanie textu; pre zobrazenie kompletnej dokumentácie navštívte *Text Formatting*.

2.2.2 Rect

Trieda Pygame Rect je dostupná ako zabudovaná. Je možné ju použiť mnohými spôsobmi, ako napríklad na detekciu kliknutí v oblasti ohraničenej práve obdĺžnikom:

Obdĺžnik nakreslíte napríklad takto:

```
RED = 200, 0, 0
BOX = Rect((20, 20), (100, 100))

def draw():
    screen.draw.rect(BOX, RED)
```

2.2.3 Nahrávanie zdrojov

The `images` and `sounds` objects can be used to load images and sounds from files stored in the `images` and `sounds` subdirectories respectively. Pygame Zero will handle loading of these resources on demand and will cache them to avoid reloading them.

You generally need to ensure that your images are named with lowercase letters, numbers and underscores only. They also have to start with a letter.

File names like these will work well with the resource loader:

```
alien.png
alien_hurt.png
alien_run_7.png
```

These will not work:

```
3.png
3degrees.png
my-cat.png
sam's dog.png
```

Obrázky

Pygame Zero can load images in `.png`, `.gif`, and `.jpg` formats. PNG is recommended: it will allow high quality images with transparency.

We need to ensure an images directory is set up. If your project contains the following files:

```
space_game.py
images/alien.png
```

Then `space_game.py` could draw the ‚alien‘ sprite to the screen with this code:

```
def draw():
    screen.clear()
    screen.blit('alien', (10, 10))
```

The name passed to `blit()` is the name of the image file within the images directory, without the file extension.

Or using the *Aktéri* API,

```
alien = Actor('alien')

def draw():
    alien.draw()
```

There are some restrictions on the file names in both cases: they may only contain lowercase latin letters, numbers and underscores. This is to prevent compatibility problems when your game is played on a different operating system that has different case sensitivity.

Image Surfaces

You can also load images from the `images` directory using the `images` object. This allows you to work with the image data itself, query its dimensions and so on:

```
forest = []
for i in range(5):
    forest.append(
        Actor('tree', topleft=(images.tree.width * i, 0))
    )
```

Each loaded image is a Pygame `Surface`. You will typically use `screen.blit(...)` to draw this to the screen. It also provides handy methods to query the size of the image in pixels:

class Surface

get_width()
Returns the width of the image in pixels.

get_height()
Returns the height of the image in pixels.

get_size()
Returns a tuple (width, height) indicating the size in pixels of the surface.

get_rect()
Get a `Rect` that is pre-populated with the bounds of the image if the image was located at the origin.

Effectively this is equivalent to:

```
Rect((0, 0), image.get_size())
```

Zvuky

Pygame Zero can load sounds in `.wav` and `.ogg` formats. WAV is great for small sound effects, while OGG is a compressed format that is more suited to music. You can find free .ogg and .wav files online that can be used in your game.

We need to ensure a sounds directory is set up. If your project contains the following files:

```
drum_kit.py
sounds/drum.wav
```

Then `drum_kit.py` could play the drum sound whenever the mouse is clicked with this code:

```
def on_mouse_down():
    sounds.drum.play()
```

Each loaded sound is a Pygame `Sound`, and has various methods to play and stop the sound as well as query its length in seconds:

class Sound

play()
Play the sound.

play(*loops*)
Play the sound, but loop it a number of times.

Parametre loops – The number of times to loop. If you pass `-1` as the number of times to loop, the sound will loop forever (or until you call *`Sound.stop()`*

stop()
Stop playing the sound.

get_length()
Get the duration of the sound in seconds.

You should avoid using the `sounds` object to play longer pieces of music. Because the sounds sytem will fully load the music into memory before playing it, this can use a lot of memory, as well as introducing a delay while the music is loaded.

2.2.4 Hudba

Nové vo verzii 1.1.

Varovanie: The music API is experimental and may be subject to cross-platform portability issues.

In particular:

- MP3 may not be available on some Linux distributions.
- Some OGG Vorbis files seem to hang Pygame with 100% CPU.

In the case of the latter issue, the problem may be fixed by re-encoding (possibly with a different encoder).

A built-in object called `music` provides access to play music from within a `music/` directory (alongside your `images/` and `sounds/` directories, if you have them). The music system will load the track a little bit at a time while the music plays, avoiding the problems with using `sounds` to play longer tracks.

Another difference to the sounds system is that only one music track can be playing at a time. If you play a different track, the previously playing track will be stopped.

`music.`**`play`**(*name*)
Play a music track from the given file. The track will loop indefinitely.

This replaces the currently playing track and cancels any tracks previously queued with `queue()`.

You do not need to include the extension in the track name; for example, to play the file `handel.mp3` on a loop:

```
music.play('handel')
```

`music.`**`play_once`**(*name*)
Similar to `play()`, but the music will stop after playing through once.

`music.`**`queue`**(*name*)
Similar to `play_once()`, but instead of stopping the current music, the track will be queued to play after the current track finishes (or after any other previously queued tracks).

`music.`**`stop`**()
Stop the music.

`music.`**`pause`**()
Pause the music temporarily. It can be resumed by calling `unpause()`.

`music.`**`unpause`**()
Unpause the music.

`music.`**`is_playing`**()
Returns True if the music is playing (and is not paused), False otherwise.

`music.`**`fadeout`**(*duration*)
Fade out and eventually stop the current music playback.

Parametre `duration` – The duration in seconds over which the sound will be faded out. For example, to fade out over half a second, call `music.fadeout(0.5)`.

`music.`**`set_volume`**(*volume*)
Set the volume of the music system.

This takes a number between 0 (meaning silent) and 1 (meaning full volume).

`music.`**`get_volume`**`()`
Get the current volume of the music system.

If you have started a music track playing using *`music.play_once()`*, you can use the *`on_music_end() hook`* to do something when the music ends - for example, to pick another track at random.

2.2.5 Clock

Often when writing a game, you will want to schedule some game event to occur at a later time. For example, we may want a big boss alien to appear after 60 seconds. Or perhaps a power-up will appear every 20 seconds.

More subtle are the situations when you want to delay some action for a shorter period. For example you might have a laser weapon that takes 1 second to charge up.

We can use the `clock` object to schedule a function to happen in the future.

Let's start by defining a function `fire_laser` that we want to run in the future:

```
def fire_laser():
    lasers.append(player.pos)
```

Then when the fire button is pressed, we will ask the `clock` to call it for us after exactly 1 second:

```
def on_mouse_down():
    clock.schedule(fire_laser, 1.0)
```

Note that `fire_laser` is the function itself; without parentheses, it is not being called here! The clock will call it for us.

(It is a good habit to write out times in seconds with a decimal point, like `1.0`. This makes it more obvious when you are reading it back, that you are referring to a time value and not a count of things.)

`clock` provides the following useful methods:

`class Clock`

`schedule`(*callback*, *delay*)
Schedule *callback* to be called after the given delay.

Repeated calls will schedule the callback repeatedly.

Parametre

- **`callback`** – A callable that takes no arguments.
- **`delay`** – The delay, in seconds, before the function should be called.

`schedule_unique`(*callback*, *delay*)
Schedule *callback* to be called once after the given delay.

If *callback* was already scheduled, cancel and reschedule it. This applies also if it was scheduled multiple times: after calling `schedule_unique`, it will be scheduled exactly once.

Parametre

- **`callback`** – A callable that takes no arguments.
- **`delay`** – The delay, in seconds, before the function should be called.

`schedule_interval`(*callback*, *interval*)
Schedule *callback* to be called repeatedly.

Parametre

- **callback** – A callable that takes no arguments.
- **interval** – The interval in seconds between calls to *callback*.

unschedule(*callback*)
Unschedule callback if it has been previously scheduled (either because it has been scheduled with `schedule()` and has not yet been called, or because it has been scheduled to repeat with `schedule_interval()`.

Note that the Pygame Zero clock only holds weak references to each callback you give it. It will not fire scheduled events if the objects and methods are not referenced elsewhere. This can help prevent the clock keeping objects alive and continuing to fire unexpectedly after they are otherwise dead.

The downside to the weak references is that you won't be able to schedule lambdas or any other object that has been created purely to be scheduled. You will have to keep a reference to the object.

2.2.6 Aktéri

Keď sa vám začne v hre pohybovať veľa obrázkov, bolo by praktické mať k dispozícii niečo, čo drží na jednom mieste obrázok spolu s pozíciou, kde sa má vykresliť. Každý pohybujúci sa obrázok na obrazovke budeme volať aktér a bude reprezentovaný triedou `Actor`. Pre vytvorenie aktéra vám bude stačiť zadať názov obrázku (z priečinku s obrázkami, ako bolo uvedené vyššie). Vykresliť mimozemšťana, o ktorom sme hovorili vyššie, môžete takto:

```
alien = Actor('alien', (50, 50))

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

Pohybovať s aktérom môžete zmenou jeho atribútu `pos` vo funkcii `update()`:

```
def update():
    if keyboard.left:
        alien.x -= 1
    elif keyboard.right:
        alien.x += 1
```

Zmeniť jeho obrázok, ktorý reprezentuje aktéra, môžete nastavením jeho atribútu `image` na nejaký nový názov obrázku:

```
alien.image = 'alien_hurt'
```

Aktéri majú všetky metódy a atribúty rovnaké ako *Rect*, vrátane metód ako .colliderect(), ktorá sa zistí, či dvaja aktéri nie sú v kolízii.

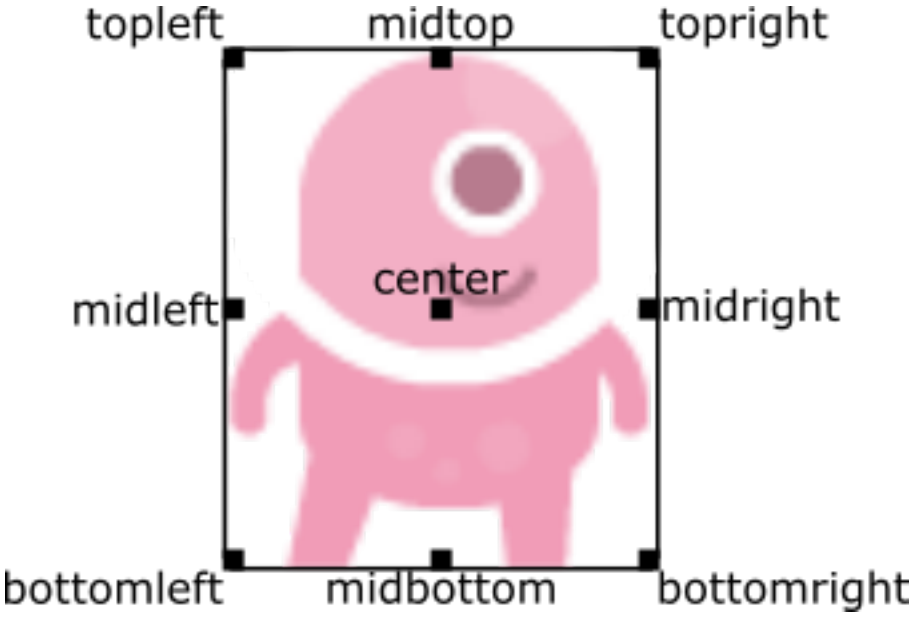
Umiestňovanie aktérov

Ak do niektorého pozičného atribútu priradíte novú hodnotu, aktér bude presunutý. Napríklad:

```
alien.right = WIDTH
```

presunie mimozemšťana tak, že jeho pravý okraj bude nastavený na `WIDTH`.

Podobne môžete nastaviť počiatočnú polohu aktéra odovzdaním do konštruktora jedného z týchto kľúčových slov ako keyword argument: `pos`, `topleft`, `topright`, `bottomleft`, `bottomright`, `midtop`, `midleft`, `midright`, `midbottom` alebo `center`:

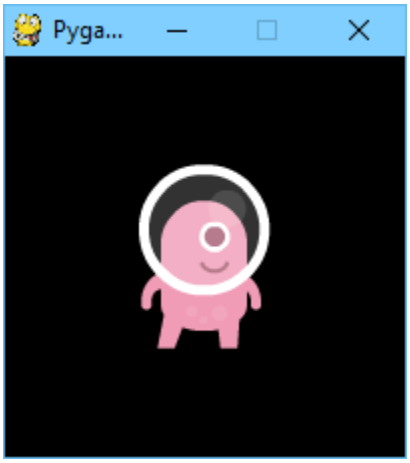


Toto môže byť zabezpečené počas vytvárania alebo alebo priradením páru súradníc x, y. Napríklad:

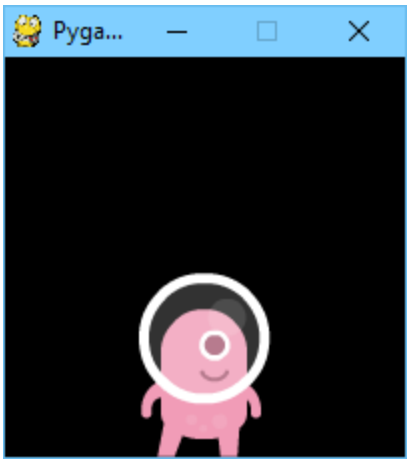
```
WIDTH = 200
HEIGHT = 200

alien = Actor('alien', center=(100,100))

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



Zmenou `center=(100, 100)` na `midbottom=(100, 200)` dostanete:



Ak nezadáte počiatočnú pozíciu, aktér bude umiestnený v ľavom hornom rohu (ekvivalent zápisu `topleft=(0, 0)`).

Anchor point

Aktéri majú „anchor position“, čo je praktický spôsob, ako umiestniť aktéra do scény. Predvolene je tento bod v strede, takže atribút `.pos` referuje na stred aktéra (rovnako aj súradnice `x` a `y`). Je bežné chcieť zmeniť anchor point na inú časť sprite-u (možno na nohu, aby bolo jednoduché nastaviť aktéra tak, že na niečom „stojí“):

```
alien = Actor('alien', anchor=('center', 'bottom'))
spaceship = Actor('spaceship', anchor=(10, 50))
```

`anchor` je špecifikovaná ako entica `(xanchor, yanchor)`, kde hodnoty môžu byť desatinné čísla alebo reťazce `left`, `center`/`middle`, `right`, `top` alebo `bottom` podľa potreby.

Otáčanie

Nové vo verzii 1.2.

Atribút aktéra `.angle` riadi otáčanie sprite-u v stupňoch proti smeru hodinových ručičiek.

Stredom otáčania je aktérov *anchor point*.

Pozor však na to, že dôjde k zmene atribútov `width` a `height` aktéra.

Ako príklad môže byť pomalé otáčanie sprite-u asteroidu vo vesmíre proti smeru hodinových ručičiek:

```
asteroid = Actor('asteroid', center=(300, 300))

def update():
    asteroid.angle += 1
```

Aby sa začal otáčať v smere hodinových ručičiek, zmeníme funkciu `update()` takto:

```
def update():
    asteroid.angle -= 1
```

Ako ďalší príklad urobíme aktéra `ship`, ktorý sa vždy otočí smerom ku kurzoru myši.

Keďže *`angle_to()`* vráti hodnotu 0 pre „pravú stranu“, sprite, ktorý použijeme pre „ship“ by mal byť natočený doprava:

```
ship = Actor('ship')

def on_mouse_move(pos):
    ship.angle = ship.angle_to(pos)
```

Nezabudnite, že uhly loop round, takže 0 stupňov == 360 stupňov == 720 stupňov. Taktiež -180 stupňov == 180 stupňov.

Vzdialenosť a uhol medzi aktérmi

Aktéri majú prktické metódy pre výpočet ich vzdialenosti alebo uhlu k ďalšiemu aktérovi alebo voči páru súradníc `(x, y)`.

`Actor.`**`distance_to`**`(`*target*`)`
: Vráti vzdialenosť od pozície tohto aktéra k cieľu v pixeloch.

`Actor.`**`angle_to`**`(`*target*`)`
: Vráti uhol medzi pozíciou tohto aktéra a cieľom v stupňoch.

  Metóda vráti číslo v rozsahu od -180 do 180 stupňov. Vpravo je 0 stupňov a uhol rastie proti smeru hodinových ručičiek. This will return a number between -180 and 180 degrees. Right is 0 degrees

  Takže:

  - Vľavo je 180 stupňov.
  - Hore je 90 stupňov.
  - Dolu je -90 stupňov.

2.2.7 Klávesnica

Určite ste si všimli, že sme v kódoch vyššie používali `keyboard`. Ak by ste chceli vedieť, aké klávesy boli stlačené na klávesnici, môžete sa na ne dopytovať pomocou atribútov zabudovanej triedy `keyboard`. Ak je napríklad podržaný kláves šípka vľavo, tak `keyboard.left` bude `True`, v opačnom prípade bude `False`.

Atribút existuje pre každý kláves; niekoľko príkladov:

```
keyboard.a  # kláves 'A'
keyboard.left  # kláves ľavej šípky
keyboard.rshift  # kláves pravej šípky
keyboard.kp0  # kláves '0' na numerickej klávesnici
keyboard.k_0  # kláves '0' na hlavnej klávesnici
```

Kompletný zoznam konštánt jednotlivých kláves nájdete v dokumentácii Tlačidlá a klávesy, ale atribúty sú malými písmenami, pretože sa jedná o premenné a nie konštanty.

Zastarané od verzie 1.1: Názvy atribútov s veľkými písmenami alebo s prefixom (napr. `keyboard.LEFT` alebo `keyboard.K_a`) sú odteraz zastarané; používajte miesto nich názvy atribútov malými písmenami.

Nové vo verzii 1.1: Odteraz sa taktiež môžete dopytovať na stav kláves pomocou ich vlastných klávesových konštánt:

```
keyboard[keys.A]  # True ak bol stlačený kláves 'A'
keyboard[keys.SPACE]  # True ak bol stlačený medzerník
```

2.2.8 Animácie

V knižnici pygame môžete animovať väčšinu vecí pomocou zabudovanej funkcie `animate()`. Napríklad, ak chcete presunúť *Actor* z jedho aktuálnej polohy na obrazovke na pozíciu `(100, 100)`:

```
animate(alien, pos=(100, 100))
```

animate(*object*, *tween='linear'*, *duration=1*, *on_finished=None*, ***targets*)
Animate the attributes on object from their current value to that specified in the targets keywords.

Parametre

- **tween** – The type of *tweening* to use.
- **duration** – Trvanie animácie v sekundách.
- **on_finished** – Funkcia, ktorá má byť zavolaná po skončení animácie.
- **targets** – Cieľové hodnoty atribútov pre animáciu.

Hodnota `tween` argumentu môže byť jedna z nasledujúcich:

‚linear‘	Animácia konštantnou rýchlosťou zo začiatku do cieľa
‚accelerate‘	Začne pomaly a zrýchľuje až do konca
‚decelerate‘	Začne rýchlo a spomaľuje až do cieľa
‚accel_decel‘	Zrýchľuje do stredu a spomaľuje do konca
‚in_elastic‘	Give a little wobble at the end
‚out_elastic‘	Have a little wobble at the start
‚in_out_elastic‘	Have a wobble at both ends
‚bounce_end‘	Accelerate to the finish and bounce there
‚bounce_start‘	Bounce at the start
‚bounce_start_end‘	Bounce at both ends

Funkcia `animate()` vrátiu inštanciu triedy `Animation`:

class Animation

stop(*complete=False*)
Zastaví animáciu, voliteľne dokončí prechod do cieľových hodnôt vlastností.

Parametre complete – Nastaví atribút animácie na jeho konečnú hodnotu.

running
Bude mať hodnotu `True`, ak je animácia spustená. A hodnotu `False`, ak uplynula alebo bola pred uplynutím zavolaná metóda `stop()`.

on_finished
Tento atribút môžete nastaviť na funkciu, ktorá má byť zavolaná, keď sa ukončí prehrávanie animácie. Nezavolá sa, ak bola zavolaná metóda `stop()`. Táto funkcia nemá žiadne argumenty. You may set this attribute to a function which will be called when the animation duration runs out. The `on_finished` argument to `animate()` also sets this attribute. It is not called when `stop()` is called. This function takes no arguments.

2.2.9 Generátor tónov

Nové vo verzii 1.2.

Pygame Zero can play tones using a built-in synthesizer.

tone.**play**(*pitch*, *duration*)
Play a note at the given pitch for the given duration.

Duration is in seconds.

The *pitch* can be specified as a number in which case it is the frequency of the note in hertz.

Alternatively, the pitch can be specified as a string representing a note name and octave. For example:

- `'E4'` would be E in octave 4.
- `'A#5'` would be A-sharp in octave 5.
- `'Bb3'` would be B-flat in octave 3.

Creating notes, particularly long notes, takes time - up to several milliseconds. You can create your notes ahead of time so that this doesn't slow your game down while it is running:

tone.**create**(*pitch*, *duration*)
Create and return a Sound object.

The arguments are as for play(), above.

This could be used in a Pygame Zero program like this:

```
beep = tone.create('A3', 0.5)

def on_mouse_down():
    beep.play()
```

KAPITOLA 3

Používateľská príručka

3.1 Inštalácia knižnice Pygame Zero

3.1.1 Dodávané s Mu

Mu IDE, ktoré sa zameriava na začiatočníkov, je dodávané s knižnicou Pygame Zero.

Ak budete chcieť knižnicu použiť, budete musieť zmeniť režim na režim Pygame Zero. Potom napíšte svoj program a použitím tlačidla Play ho spustíte s knižnicou Pygame Zero.

Poznámka: Verzia dodávanej knižnice Pygame Zero v editore Mu však nemusí byť najnovšia! Verziu knižnice môžete zistiť spustením tohto kódu v editore Mu:

```
import pgzero
print(pgzero.__version__)
```

3.1.2 Samostatná inštalácia

Najprv potrebujete mať nainštalovaný **Python 3**! Ten je obyčajne predinštalovaný, ak používate **Linux** alebo **Raspberry Pi**. Pre iné systémy si ho môžete stiahnuť zo stránky *python.org <https://www.python.org/>*.

Windows

Pre nainštalovanie knižnice Pygame Zero použite **pip**. Do príkazového riadku zadajte

```
pip install pgzero
```

Mac

Do okna terminálu zadajte

```
pip install pgzero
```

Všimnite si, že zatiaľ nie sú k dispozícii žiadne Wheels pre knižnicu Pygame s podporou Python 3.4 pre Mac, takže budete musieť aktualizovať Python na verziu >=3.6 (alebo použite Python 2.7), aby ste mohli knižnicu Pygame nainštalovať. Pre získanie zoznamu dostupných Wheels navštívte stránku *pyPI_*

Linux

Do okna terminálu zadajte

```
sudo pip install pgzero
```

Niektoré linuxové systémy však používajú `pip3`; ak predchádzajúci príkaz vypíše niečo ako `sudo: pip: príkaz nenájdený` potom zadajte:

```
sudo pip3 install pgzero
```

Občas pip nie je nainštalovaný, takže ho treba doinštalovať. Ak je to váš prípad, zadajte najprv nasledujúci príkaz pred opätovným spustením predchádzajúcich:

```
sudo python3 -m ensurepip
```

3.1.3 Inštalácia REPL

Pygame Zero's REPL is an optional feature. This can be enabled when installing with `pip` by adding `pgzero[repl]` to the pip command line:

```
pip install pgzero[repl]
```

If you aren't sure if you have the REPL installed, you can still run this command (it won't break anything if it is installed!).

3.2 Using the REPL (Read-Evaluate-Print Loop)

The REPL allows you to interact with a running Pygame Zero game using Python commands. As you type it will offer suggestions based on variables that exist in your program. This can be useful for debugging your game or tuning how difficult it is.

REPL is short for a Read-Evaluate-Print Loop; it means:

1. **Read** a line of code typed by you
2. **Evaluate** the code
3. **Print** the result
4. **Loop** back to step 1!

This is an *optional feature* that may need to *be installed* if it was not originally installed with Pygame Zero. If you try using the REPL, Pygame Zero will let you know if it is not installed.

3.2.1 Running a Pygame Zero program with the REPL

If you normally run your Pygame Zero program using the terminal, add `--repl` to the command line when running `pgzrun`. For example, if your game is in a file called `mygame.py`, run:

```
pgzrun --repl mygame.py
```

3.2.2 Using the REPL

Python code that you type at the REPL is evaluated as if you had typed it into your game file.

For example, if your game file contains the code

```
alien = Actor('alien', pos=(54, 60))

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

Then at the REPL you could type `alien` to see the alien object:

```
>>> alien
<Actor 'alien' pos=(54, 60)>
```

You can set attributes on the `alien` object and see it move:

```
>>> alien.x = 90
```

3.3 Running Pygame Zero in IDLE and other IDEs

Nové vo verzii 1.2.

Pygame Zero is usually run using a command such as:

```
pgzrun my_program.py
```

Certain programs, such as integrated development environments like IDLE and Edublocks, will only run `python`, not `pgzrun`.

Pygame Zero includes a way of writing a full Python program that can be run using `python`. To do it, put

```
import pgzrun
```

as the very first line of the Pygame Zero program, and put

```
pgzrun.go()
```

as the very last line.

3.3.1 Example

Here is a Pygame Zero program that draws a circle. You can run this by pasting it into IDLE:

```
import pgzrun


WIDTH = 800
HEIGHT = 600

def draw():
    screen.clear()
    screen.draw.circle((400, 300), 30, 'white')


pgzrun.go()
```

3.4 Other libraries like Pygame Zero

Pygame Zero started a trend for Python „zero“ libraries. Our friends have created these great libraries. Some of these can be combined with Pygame Zero!

3.4.1 Network Zero

Network Zero makes it simpler to have several machines or several processes on one machine discovering each other and talking across a network.

Upozornenie: If you want to use Network Zero with Pygame Zero, make sure you don't let it **block** (stop everything while waiting for messages). This will interrupt Pygame Zero so that it stops animating the screen or even responding to input. Always set the `wait_for_s` or `wait_for_reply_s` options to `0` seconds.

3.4.2 GUI Zero

GUI Zero is a library for creating Graphical User Interfaces (GUIs) with windows, buttons, sliders, textboxes and so on.

Because GUI Zero and Pygame Zero both provide different approaches for drawing to the screen, they are not usable together.

3.4.3 GPIO Zero

GPIO Zero is a library for controlling devices connected to the General Purpose Input/Output (GPIO) pins on a Raspberry Pi.

GPIO Zero generally runs in its own thread, meaning that it will usually work very well with Pygame Zero.

Upozornenie: When copying GPIO Zero examples, do not copy the `time.sleep()` function calls or `while True:` loops, as these will stop Pygame Zero animating the screen or responding to input. Use *Clock* functions instead to call functions periodically, or the *`update()`* function to check a value every frame.

3.4.4 Adventurelib

Adventurelib is a library for creating making text-based games easier to write (and which doesn't do everything for you!).

Writing text-based games requires a very different set of skills to writing graphical games. Adventurelib is pitched at a slightly more advanced level of Python programmer than Pygame Zero.

Adventurelib cannot currently be combined with Pygame Zero.

3.4.5 Blue Dot

Blue Dot allows you to control your Raspberry Pi projects wirelessly using an Android device as a Bluetooth remote.

Blue Dot generally runs in its own thread, meaning that it will usually work very well with Pygame Zero.

Upozornenie: Avoid `time.sleep()` function calls, `while True:` loops and Blue Dot's blocking `wait_for_press` and `wait_for_release` methods, as these will stop Pygame Zero animating the screen or responding to input. Use *Clock* functions instead to call functions periodically, or the *`update()`* function to check a value every frame.

Tip: Know of another library that belongs here?

Open an issue on the issue tracker to let us know!

3.5 Changelog

3.5.1 1.2 - 2018-02-24

- New: *Actors can be rotated* by assigning to `actor.angle`
- New: Actors now have *`angle_to()`* and *`distance_to()`* methods.
- New: Actors are no longer subclasses of Rect, though they provide the same methods/properties. However they are now provided with floating point precision.
- New: `tone.play()` function to allow playing musical notes.
- New: `pgzrun.go()` to allow running Pygame Zero from an IDE (see *Running Pygame Zero in IDLE and other IDEs*).
- New: show joypad icon by default
- Examples: add Asteroids example game (thanks to Ian Salmons)
- Examples: add Flappy Bird example game
- Examples: add Tetra example game (thanks to David Bern)
- Docs: Add a logo, fonts and colours to the documentation.
- Docs: Documentation for the *anchor point system for Actors*
- Docs: Add *Prechod z nástroja Scratch* documentation
- Fix: `on_mouse_move()` did not correctly handle the `buttons` parameter.
- Fix: Error message when resource not found incorrectly named last extension searched.
- Fix: Drawing wrapped text would cause crashes.
- Fix: *`animate()`* now replaces animations of the same property, rather than creating two animations which fight.
- Updated ptext to a revision as of 2016-11-17.
- Removed: removed undocumented British English `centrex`, `centrey`, `centre` attribute aliases on ZRect (because they are not Rect-compatible).

3.5.2 1.1 - 2015-08-03

- Added a spell checker that will point out hook or parameter names that have been misspelled when the program starts.
- New ZRect built-in class, API compatible with Rect, but which accepts coordinates with floating point precision.
- Refactor of built-in `keyboard` object to fix attribute case consistency. This also allows querying key state by `keys` constants, eg. `keyboard[keys.LEFT]`.
- Provide much better information when sound files are in an unsupported format.

- `screen.blit()` now accepts an image name string as well as a Surface object, for consistency with Actor.
- Fixed a bug with non-focusable windows and other event bugs when running in a virtualenv on Mac OS X.
- Actor can now be positioned by any of its border points (eg. `topleft`, `midright`) directly in the constructor.
- Added additional example games in the `examples/` directory.

3.5.3 1.0.2 - 2015-06-04

- Fix: ensure compatibility with Python 3.2

3.5.4 1.0.1 - 2015-05-31

This is a bugfix release.

- Fix: Actor is now positioned to the top left of the window if `pos` is unspecified, rather than appearing partially off-screen.
- Fix: repeating clock events can now unschedule/reschedule themselves

  Previously a callback that tried to unschedule itself would have had no effect, because after the callback returns it was rescheduled by the clock.

  This applies also to `schedule_unique`.
- Fix: runner now correctly displays tracebacks from user code
- New: Eliminate redraws when nothing has changed

  Redraws will now happen only if:

  - The screen has not yet been drawn
  - You have defined an update() function
  - An input event has been fired
  - The clock has dispatched an event

3.5.5 1.0 - 2015-05-29

- New: Added `anchor` parameter to Actor, offering control over where its `pos` attribute refers to. By default it now refers to the center.
- New: Added Ctrl-Q/-Q as a hard-coded keyboard shortcut to exit a game.
- New: `on_mouse_*` and `on_key_*` receive `IntEnum` values as `button` and `key` parameters, respectively. This simplifies debugging and enables usage like:

```
if button is button.LEFT:
```

3.5.6 1.0beta1 - 2015-05-19

Initial public (preview) release.

KAPITOLA 4

Nálepky od

Sticker Mule have graciously offered to provide Pygame Zero laptop stickers for learners.

4.1 Laptop Stickers

Sticker Mule have graciously offered to provide a number of stickers for Pygame Zero users for free.

Laptop stickers are a great way to encourage students to continue tinkering and learning, as well as spreading the word about Pygame Zero.

The stickers should look a little like this (not to scale):

4.1.1 For learners

Due to the costs of distribution, and because Pygame Zero is a free community library, **we don't have a way of getting stickers directly to you yet.**

It may be possible to obtain stickers at conferences and meet-ups. Please check back soon.

4.1.2 For educators/meet-ups

We would like to distribute stickers primarily via educators and educational meet-ups. At this time it is not known how many stickers we will be able to distribute in this way (and it may be prohibitively expensive to ship them outside the UK).

Please fill out our Google Form to express your interest.

4.1.3 For developers

Free stickers are primarily intended for learners. However, if a pull request you make to Pygame Zero or a translation is accepted, we would be very happy to give you a free laptop sticker if the opportunity arises.

Please request a sticker in your Pull Request comments (or make yourself known at a conference/meet-up).

If you attend educational events or Python events regularly, and you would be willing to distribute stickers, this could also be useful. Please let us know.

KAPITOLA 5

Vylepšovanie knižnice Pygame Zero

5.1 Roadmap

Pygame Zero is an open source project, and as with any such project, the development roadmap is subject to change.

This document just lays out some goals for future releases, but there is **no guarantee** that these targets will be hit.

5.1.1 Translations

Pygame Zero is aimed at young users, whose English skills might not be good enough to read the documentation if it isn't in their own language.

Adding translations of the documentation would help to bring Pygame Zero to new users. This is something that needs contributors to help with. My own language skills aren't good enough!

Please see *the translating guide* if you think you can help.

5.1.2 Gamepad Support

Github Issue: #70

SNES-style gamepads are now extremely cheap. For example, they are sold for a few pounds from the Pi Hut, in packs of 2 at Amazon, and even in some Raspberry Pi bundles.

Gamepad support should not be limited to these specific models; rather, we should treat this as a lowest-common-denominator across modern gamepads, as nearly all more modern gamepads have at least as many buttons and axes.

This feature needs to be added in a way that will not **require** a gamepad to play any Pygame Zero game, in order to follow the principle of *Make it accessible*.

5.1.3 Surface juggling

Github Issue: #71

Pygame experts make lots of use of off-screen surfaces to create interesting effects.

Pygame Zero chose to consider only the screen surface, which we wrap with a richer `Screen` API for drawing, etc.

The problem is that there is no easy path to using additional surfaces - Pygame Zero immediately becomes dead weight as you start to look past that curtain.

We should look to smooth out this path to make Pygame Zero Actors and Screen work better with custom surfaces.

5.1.4 Storage

Github Issue: #33

It would be useful for users to be able to save and load data.

The obvious application is save games, but saving and loading whole games can be pretty hard to get right. The simpler application would just be saving settings, customisations, high scores, or the highest level reached.

Python of course has APIs for reading and writing files, but this has additional complexity that teachers might not want to teach immediately.

5.2 Principles of Pygame Zero

Please read the following carefully before contributing.

Because Pygame Zero is aimed at beginners we must take extra care to avoid introducting hurdles for programmers who have not yet learned to deal with them.

5.2.1 Make it accessible

The main aim of Pygame Zero is to be accessible to beginner programmers. The design of the API is, of course, influenced by this.

This also applies to things like hardware requirements: Pygame Zero chose originally to support only keyboard and mouse input, in order to be accessible to any user.

5.2.2 Be conservative

Early in the development of Pygame Zero, Richard and I (Daniel) went backwards and forwards over various features. We put them in, tried them and took them out again.

Features should be rejected if they are too experimental, or if they might cause confusion.

This also applies to things like OS support: we disallow filenames that are not likely to be compatible across operating systems.

5.2.3 Just Work

Pygame Zero wraps Pygame almost completely - but we don't expose all the features. We expose only the features that work really well without extra fuss, and hide some of the other features that work less well or need extra steps.

5.2.4 Minimise runtime cost

At the end of the day, Pygame Zero is a games framework and performance is an issue.

Doing expensive checking every frame to catch a potential pitfall is not really acceptable. Instead, we might check at start up time, or check only when an exception is raised to diagnose it and report more information.

5.2.5 Error clearly

When exceptions are thrown by Pygame Zero, they should have clear error messages that explain the problem.

5.2.6 Document well

Like all projects, Pygame Zero needs good documentation. Pull requests are more likely to be accepted if they include the necessary documentation.

Try to avoid complicated sentences and technical terms in the documentation, so that it is more easily readable by inexperienced programmers.

5.2.7 Minimise breaking changes

In educational environments, users don't always have control of the version of a library they use. They don't know how to install or upgrade to the latest version.

It is more important to get the features right first time than in many other projects.

5.3 Contributing to Pygame Zero

The Pygame Zero project is hosted on GitHub:

> https://github.com/lordmauve/pgzero

5.3.1 Reporting an bug or request

You can report bugs, or request features that you think should be in Pygame Zero, using the Github issue tracker.

Here are some things to bear in mind before you do this:

- It might not just be you! You should check if someone has already reported the issue by searching through the existing issues - both open and closed.
- The developers need to know what version of Pygame Zero you are using, and what operating system you are running (Windows, Mac, Linux etc) and version (Windows 10, Ubuntu 16.04, etc).

5.3.2 How to do a pull request

You can make changes to Pygame Zero by creating a pull request.

It's a good idea to *report an issue* first, so that we can discuss whether your change makes sense.

Github has help on how to create a pull request, but here's the quick version:

1. Make sure you are logged into Github.

2. Go to the Github page for Pygame Zero.
3. Click „Fork“ to create your own fork of the repository.
4. Clone this fork to your own computer:

```
git clone git@github.com:yourusername/pgzero.git
```

   Remember to change `yourusername` to your Github username.

5. Create a branch in which to do your changes. Pick a branch name that describes the change you want to make.

```
git checkout -b my-new-branch master
```

6. Make the changes you want.
7. Add the files that you want to commit:

```
git add pgzero
```

8. Commit the files with a clear commit message:

```
git commit -m "Fixed issue #42 by renaming parameters"
```

   You can do steps 6 to 8 as many times as you want until you’re happy.

9. Push the commit back to your fork.

```
git push --set-upstream origin my-new-branch
```

10. Go to the Github page for your fork, and click on the „Create pull request“ button.

5.3.3 Development installation

It’s possible to create a locally-editable install using pip. From the root directory of the checked out source, run:

```
pip3 install --editable .
```

The installed version will now reflect any local changes you make.

Alternatively, if you don’t want to install it at all, it may be run with:

    python3 -m pgzero <name of pgzero script>

For example:

    python3 -m pgzero examples/basic/demo1.py

5.3.4 How to run tests

The tests can be run with

    python3 setup.py test

5.3.5 Helping to translate the documentation

Pygame Zero's APIs will always be English, but we can bring Pygame Zero to more users around the world if the documentation is available in their language.

If you are fluent in another language, please consider contributing by translating all or part of the documentation.

The documentation is written in reStructuredText, which is a text-based markup language for technical documentation. As much as possible, the existing formatting should be preserved. reStructuredText isn't too difficult once you get used to it.

Creating a translation is done by creating a separate repository on Github with a copy of the documentation, rewritten (at least in part) into the language you would like to support. One advantage of this is that you can work on translations at your own pace, without having to submit pull requests back to the `pgzero` project itself. Please see the translation guide on Read The Docs for details.

If this sounds like something you could tackle, here's how you might go about it:

1. First, open an issue on the pgzero issue tracker. You should search for an existing issue covering the translation you want to do, before opening a new one. This will help ensure that you don't do translation work that has already been done by someone else (perhaps you can collaborate instead).
2. Create a new Github repository under your user, called pgzero-*language*, eg. `pgzero-spanish` if you're going to translate into Spanish.
3. Clone the repository to your own computer.
4. Download the Pygame Zero `doc/` directory and commit it in your project. You can do this by extracting them from repository ZIP file. You only need the `doc/` directory from the ZIP file. You can delete the other files.
5. Now, work through the .rst files in the docs directory, translating, using your preferred editor. You should commit regularly, and push your commits to Github.
6. Post a link to your repository as a comment in the Github issue you created in step 1. You can do this as soon as you have some progress to show; this will help people collaborate with you on the translation if they are interested.
7. Set up the documentation to build on Read The Docs. Again, post a comment on the Github issue when you have this working.
8. We can then link up the new, translated documentation with the Pygame Zero documentation.

Note that Pygame Zero will have updates, and the documentation will be changed accordingly. Using Git it is possible to see a diff of what changed in the English documentation, so that you can make corresponding changes in the translated documentation.

Index

Q

R

S

T

U

V

W

X

Y

Z