

A modular adventure game



Start code

- main.py

```
import pygame, sys

def main():

    # Load
    pygame.init()
    screen = pygame.display.set_mode((800, 600))
    font = pygame.font.Font(None, 24)
    quit = False

    while not(quit):
        # Inputs
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    quit = True

        # Update

        # Draw
        screen.fill((0, 0, 0))
        pygame.display.update()

if __name__ == "__main__":
    main()
```

Sprites

Display background

- main.py

```
import pygame, sys

# Load
...
path = 'D:\\Code\\..\\adventure\\'
background = pygame.image.load(path+'background.png').convert()
...
while not(quit):
    ...
    # Draw
    screen.fill((0, 0, 0))
    screen.blit(background, (0, 0))
    pygame.display.update()
```

- `pygame.image.load(...)` has one argument, which is the path to the image on your hard drive
- `path + "background.png"` concatenate the content of the `path` variable and `"background.png"`
- `convert()` function change the pixel format of the image, to give the display surface format

Now, display a sprite

- main.py

?

(you can do it, you have all necessary code)

Now, display a sprite

- main.py

```
import pygame, sys

# Load
...
spr_surface = pygame.image.load(path+'sprite.png').convert()
spr_pos = spr_x, spr_y = 100, 400
...

# Draw
screen.fill((0, 0, 0))
screen.blit(background, (0, 0))
screen.blit(spr_surface, spr_pos)
pygame.display.update()
```

- You can use convert_alpha() if your sprite has transparency
- We draw image with the painter algorithm : first paint the background, then elements of foreground from back to front
- Python's short way to define multiple variables : spr_pos = spr_x, spr_y = 100, 400

Mouse cursor

- You get cursor position with : `cursor_pos = pygame.mouse.get_pos()`
- This code should go in the update part, because we update the `cursor_pos` variable
- Hide OS default cursor with : `pygame.mouse.set_visible(False)` (in the Load part)
- You have what you need

Mouse cursor

· main.py

```
cursor = pygame.image.load(path+'cursor.png').convert_alpha()
pygame.mouse.set_visible(False)
...
while not(quit):
    ...
    # Update
    cursor_pos = pygame.mouse.get_pos()

    # Draw
    screen.fill((0, 0, 0))
    screen.blit(background, (0, 0))
    screen.blit(spr_surface, spr_pos)
    screen.blit(cursor, cursor_pos) → # Cursor above all
    pygame.display.update()
```

Move

Teleport sprite to cursor

- You can know if there is a mouse click event with :

```
if event.type == pygame.MOUSEBUTTONDOWN:
```

- Get the mouse position at this moment and move the sprite to that position
- For clarity, it is better to manage inputs in the Inputs part, and moving in the Update part

Teleport sprite to cursor

• main.py

```
# Inputs
for event in pygame.event.get():
    ...
    if event.type == pygame.MOUSEBUTTONDOWN:
        mouse_click = pygame.mouse.get_pos()
        spr_is_moving = True

# Update
...
if(spr_is_moving):
    spr_pos = mouse_click
    spr_is_moving = False
```



Avoid setting position every frame

Stay on the ground

- Add a ground image in front of the background
- Set the sprite so that its bottom hits the ground
- Now, when we move the sprite, we want it to stay on the ground

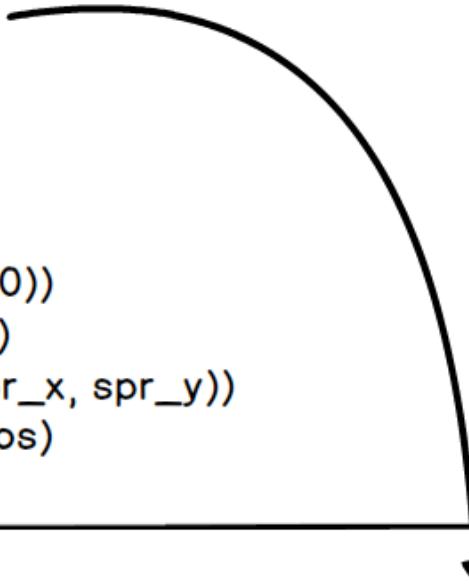
Stay on the ground

· main.py

```
# Load
ground = pygame.image.load(path+'ground.png').convert()
spr_x, spr_y = 100, 400

# Update
...
if(spr_is_moving):
    spr_x = mouse_click[0]
    spr_is_moving = False

# Draw
screen.fill((0, 0, 0))
screen.blit(background, (0, 0))
screen.blit(ground, (0, 500))
screen.blit(spr_surface, (spr_x, spr_y))
screen.blit(cursor, cursor_pos)
pygame.display.update()
```

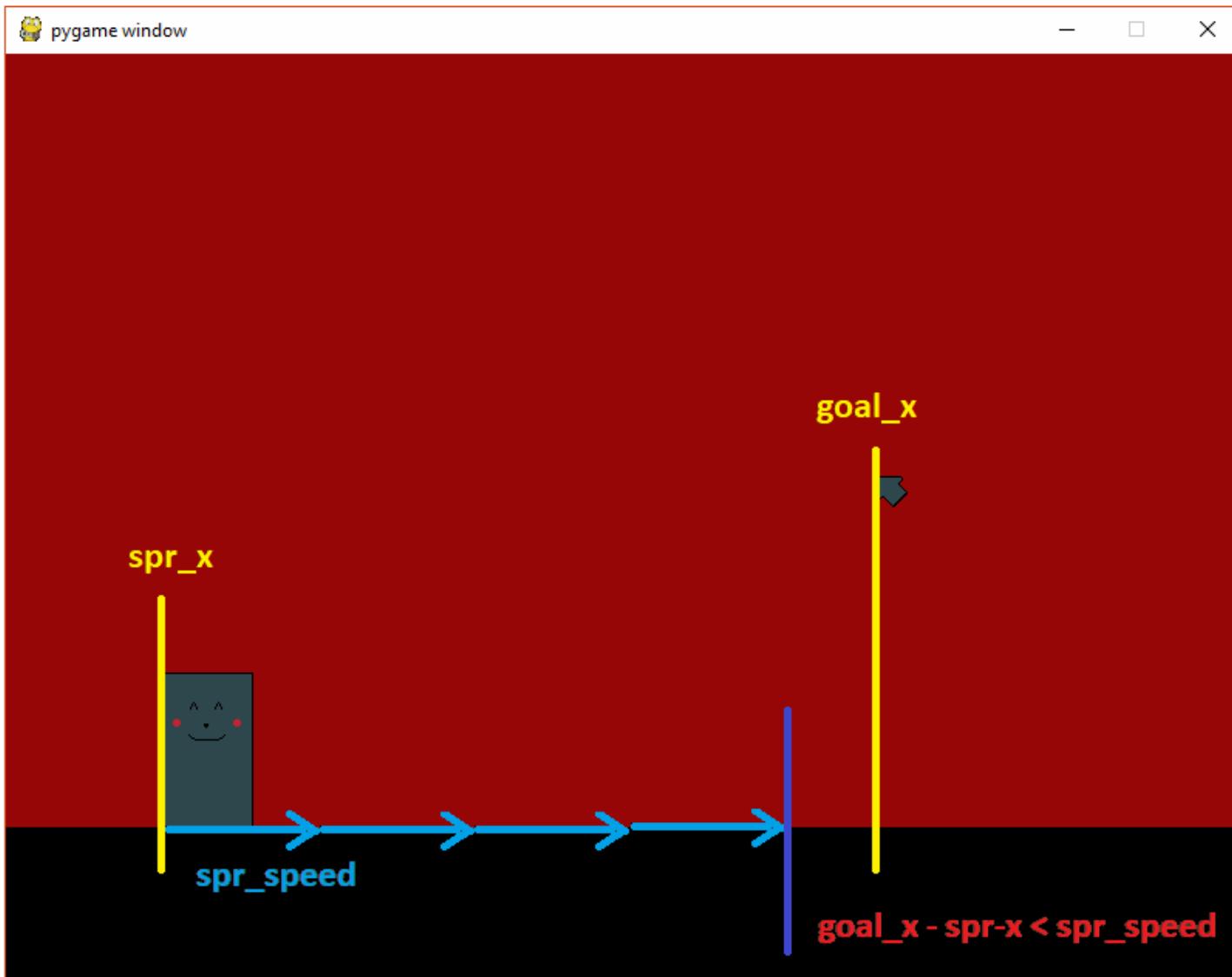


First element of the mouse_click Tuple

(In programming field we count from zero.)

Progressive move

- You need :
a variable `goal_x` to set the goal of the moving sprite,
a boolean flag to tell the sprite is moving,
a speed variable to move the sprite gradually each frame
- You stop the move when the difference between sprite position and sprite goal is inferior to speed
- May this diagram be with you :



- import math and use `math.fabs(goal_x - spr_x)` to check the difference on any side (it computes absolute value)

Progressive move

· main.py

```
import math
...
# Load
...
spr_is_moving = False
spr_speed = 2
goal_x = 0
...
# Inputs
for event in pygame.event.get():
    ...
    if event.type == pygame.MOUSEBUTTONDOWN:
        mouse_click = pygame.mouse.get_pos()
        goal_x = mouse_click[0]
        spr_is_moving = True
# Update
...
if(spr_is_moving):
    if(spr_x < goal_x):
        spr_x = spr_x + spr_speed
    if(spr_x > goal_x):
        spr_x = spr_x - spr_speed
    if(math.fabs(goal_x - spr_x) < spr_speed):
        spr_is_moving = False
```

Collisions

Add a friend

- Add an other sprite, which will be an interactive element

Friend

- main.py

```
# Load
...
copain_x, copain_y = 500, 400
copain_surface = pygame.image.load(path+'copain.png').convert()
...

# Draw
...
screen.blit(copain_surface, (copain_x, copain_y))
...
```

Display text on collision

- Display a text above your player's sprite when it hits its friend
- Collision condition (w is width and h is height) :

```
if(not(x1 + w1 < x2 or x2 + w2 < x1 or y1 + h1 < y2 or y2 + h2 < y1)):
```

Friend collision

· main.py

```
# Load
...
collision_text = font.render("Oops, sorry.", False, (0, 0, 0))
...

# Draw
...
screen.blit(spr_surface, (spr_x, spr_y))

x1, y1, w1, h1 = spr_x, spr_y, spr_surface.get_width(), spr_surface.get_height()

x2, y2, w2, h2 = copain_x, copain_y, copain_surface.get_width(), copain_surface.get_height()

if(not(x1 + w1 < x2 or x2 + w2 < x1 or y1 + h1 < y2 or y2 + h2 < y1)):
    screen.blit(collision_text, (spr_x, spr_y - 100))

screen.blit(cursor, cursor_pos)
...
```

Objects oriented programming

Classes

Hey, an other friend who collides!

- No, it's a joke. But imagine we would want to add an other friend.
- We would have to create new variable, new coordinates, new collisions test, between every couples of friends. Long. Boring.
- Let's get more abstract to find a solution. A Sprite CLASS which would implement the behaviour of any colliding sprite
- We would only have to INSTANCE as many sprite we want and let them manage their collisions

Sprite class

· sprite.py

```
import pygame

class Sprite:

    path = 'D:\\Code\\...\\adventure\\' → # variable shared between instances

    def __init__(self, x, y, filename): → # constructor : special method that initialise instances
        self.x = x → # self : member specific to this instance
        self.y = y
        self.surface = pygame.image.load(Sprite.path + filename).convert_alpha()

    def set_position(self, position): → # method : class function.
        self.x = position[0] → # The first parameter is omitted when called
        self.y = position[1]

    def intersects(self, sprite):
        x1, y1, w1, h1 = self.x, self.y, self.surface.get_width(), self.surface.get_height()
        x2, y2, w2, h2 = sprite.x, sprite.y, sprite.surface.get_width(), sprite.surface.get_height()
        return not(x1 + w1 < x2 or x2 + w2 < x1 or y1 + h1 < y2 or y2 + h2 < y1)

    def draw(self, screen):
        screen.blit(self.surface, (self.x, self.y))
```

Using the Sprite class

• main.py

```
...
from sprite import Sprite

# Load
...
player = Sprite(100, 400, 'sprite.png')
copain = Sprite(500, 400, 'copain.png')

spr_is_moving = False
spr_speed = 2
goal_x = 0
...

# Draw
...
copain.draw(screen)
player.draw(screen)
if(player.intersects(copain)):
    screen.blit(collision_text, (player.x, player.y - 100))

cursor.draw(screen)
pygame.display.update()
```

Some perfectionism

- Our code is lighter, but we would like to get rid of all that player sprite move management variables
- We will create a `SpriteControlled` class, subclass of `Sprite`
- It will share all `Sprite` behaviour, plus manage movement
- What is a subclass ? Here is an analogy. If `Animal` is a class, `Dog` is a subclass. A subclass is a specialisation of a class.

Sprite class

- sprite_controlled.py

```
import math
from sprite import Sprite

class SpriteControlled(Sprite):

    def __init__(self, x, y, filename, speed):
        Sprite.__init__(self, x, y, filename)
        self.speed = speed
        self.goal_x = x
        self.is_moving = False

    def move_to(self, x):
        self.goal_x = x
        self.is_moving = True

    def update(self):
        if(self.is_moving):
            if(self.x < self.goal_x):
                self.x = self.x + self.speed
            if(self.x > self.goal_x):
                self.x = self.x - self.speed
            if(math.fabs(self.goal_x - self.x) < self.speed):
                self.is_moving = False
```

Sprite class

· main.py

```
from sprite_controlled import SpriteControlled

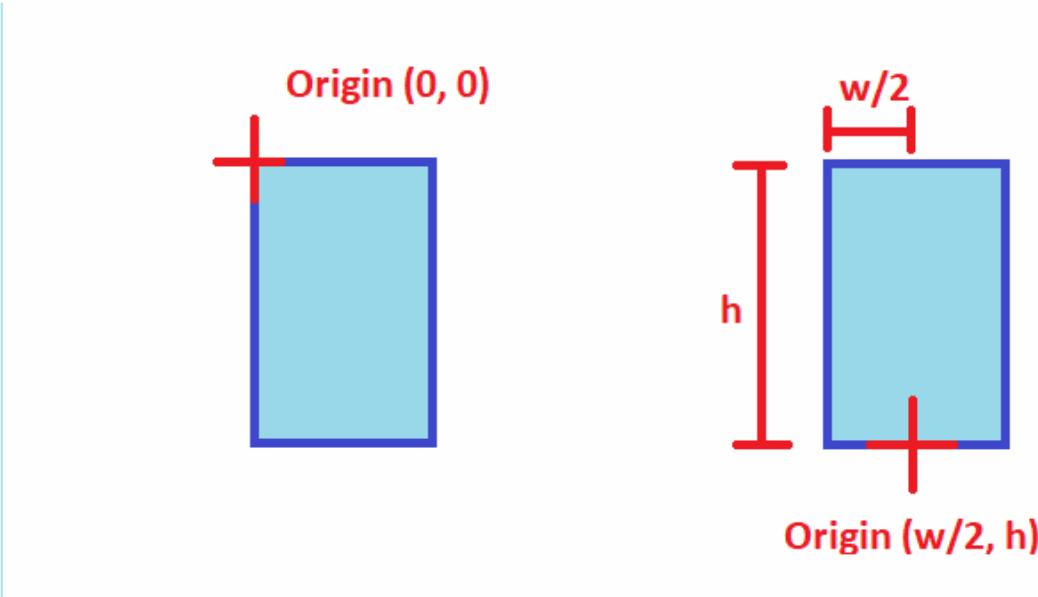
# Load
...
player = SpriteControlled(100, 400, 'sprite.png', 2)
...

# Inputs
for event in pygame.event.get():
    ...
    if event.type == pygame.MOUSEBUTTONDOWN:
        mouse_click = pygame.mouse.get_pos()
        player.move_to(mouse_click[0])

# Update
...
player.update()
```

Center sprite - Origin

- For now our sprite is displayed with the upper left corner as a point of reference (or "origin")
- We want some sprite to have an other origin point : the center bottom
- For that, we use two member variables, ox and oy , as offset from the upper left point



- Use `self.surface.get_width()` and `self.surface.get_height()` to get the sprite size values
- Some sprites still need the upper left point as origin (e.g.: cursor). So we will set a boolean in the constructor to choose which origin the Sprite uses.
- Draw the Sprite from this point
- Don't forget to update the intersect method code !

Sprite class

· sprite.py

```
def __init__(self, x, y, filename, centered):
    ...
    self.ox = 0
    self.oy = 0
    if(centered):
        self.ox = -self.surface.get_width() / 2
        self.oy = -self.surface.get_height()

    def intersects(self, sprite):
        x1, y1, w1, h1 = self.x + self.ox, self.y + self.oy, self.surface.get_width(), self.surface.get_height()
        x2, y2, w2, h2 = sprite.x + sprite.ox, sprite.y + sprite.oy, sprite.surface.get_width(), sprite.surface.get_height()
        return not(x1 + w1 < x2 or x2 + w2 < x1 or y1 + h1 < y2 or y2 + h2 < y1)

    def draw(self, screen):
        screen.blit(self.surface, (self.x + self.ox, self.y + self.oy))
```



New constructor argument. Update main !

Scenes

Scene

- For now we populate our game by adding elements in the main function. But imagine we want different levels, or different screens (menu, gameplay, credits...)
- Once more, we will use the power of abstraction. We will create a Scene class.
- The scene will handle the logic of a game : inputs, update, draw
- It will contains the elements that were in the main function

Scene class

· scene.py

```
import pygame
from sprite_controlled import SpriteControlled
from sprite import Sprite

class Scene:

    def __init__(self, name, background_file, ground_file):
        self.name = name
        self.background = Sprite(0, 0, background_file, False)
        self.ground = Sprite(0, 0, ground_file, False)
        screen_w, screen_h = pygame.display.get_surface().get_size()
        ground_height = screen_h - self.ground.surface.get_height()
        self.ground.y = ground_height

        self.player = SpriteControlled(10, ground_height, 'sprite.png', True, 2)
        self.cursor = Sprite(0, 0, 'cursor.png', False)

    def load(self):
        pass

    def inputs(self, events):
        for event in events:
            if event.type == pygame.MOUSEBUTTONDOWN:
                mouse_click = pygame.mouse.get_pos()
                self.player.move_to(mouse_click[0])

    def update(self):
        self.cursor.set_position(pygame.mouse.get_pos())
        self.player.update()

    def draw(self, screen):
        self.background.draw(screen)
        self.ground.draw(screen)
        self.player.draw(screen)
        self.cursor.draw(screen)
```

Scene class

· main.py

```
import pygame, time, sys, math
from sprite import Sprite
from sprite_controlled import SpriteControlled
from scene import Scene

def main():

    # Load
    pygame.init()
    screen = pygame.display.set_mode((800, 600))
    pygame.mouse.set_visible(False)

    level00 = Scene("level00", "background.png", "ground.png")
    current_scene = level00

    quit = False

    while not(quit):
        # Inputs
        events = pygame.event.get()
        for event in events:
            if event.type == pygame.QUIT:
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    quit = True
        current_scene.inputs(events)

        # Update
        current_scene.update()

        # Draw
        screen.fill((0, 0, 0))
        current_scene.draw(screen)
        pygame.display.update()
```

Scene variety

- Now let's transition between scenes
- We need a data structure to store scenes with an identifier (e.g. : "level00", "level01")
- Conveniently, computer science people created the map structure (also called dictionary)
- A map is an unsorted list of key / value pairs. In python you create a map with :

```
map = {}
```

- ... and you populate it with :

```
map[key] = value
```

- So create two scenes and populate a scenes map with your scenes and their identifiers

A scene map

- main.py

```
def main():

    # Load
    ...
    level00 = Scene("background.png", "ground.png")
    level01 = Scene("background.png", "ground1.png")
    scenes = {}
    scenes["level00"] = level00
    scenes["level01"] = level01
    current_scene = level00
```

Scene variety continued

- We will create a Sprite subclass that will change the scene when intersected (a Warp)
- This Warp needs a member to store the key of the new scene
- Conveniently, computer science people created the map structure (also called dictionary)
- Add the collision logic to your scene code

Scene transitioning

- warp.py

```
import pygame
from sprite import Sprite

class Warp(Sprite):
    def __init__(self, x, y, filename, centered, to_scene):
        Sprite.__init__(self, x, y, filename, centered)
        self.to_scene = to_scene
```

- scene.py

```
def __init__(self, background_file, ground_file):
    ...
    self.warp = Warp(500, 0, 'warp.png', False, "level01")
    self.warp.y = ground_height - self.warp.surface.get_height() / 2

def update(self):
    self.cursor.set_position(pygame.mouse.get_pos())
    self.player.update()
    if(self.player.intersects(self.warp)):
        ???
```

- The problem is we need to change scene WHILE we are in a scene
- So the logic of scene changing should come from outside the current scene
- We will use a function argument

Pass a function to transition scene

• main.py

```
# Load
...
current_scene = level00

def change_scene(name):
    nonlocal current_scene
    current_scene = scenes[name]

while not(quit):
    ...
    # Update
    current_scene.update(change_scene)
```

• scene.py

```
def update(self, change_scene):
    ...
    if(self.player.intersects(self.warp)):
        change_scene(self.warp.to_scene)
```



Function is passed as an argument

Scene data

- If we want to create different scenes, we have two choices :
 - use a scene subclass,
 - or provide different data to the scene we already have
- The most easy solution is creating a scene subclass, that we will call LevelXX (XX is the number).
- Each LevelXX is a Scene. So we can use in main.py :

```
def main():

    # Load
    ...
    level00 = Level00("background.png", "ground.png")
    level01 = Level01("background.png", "ground1.png")
    scenes = {}
    scenes["level00"] = level00
    scenes["level01"] = level01
    current_scene = level00
```

Scene transitioning

- scene.py

```
class Scene:  
  
    def __init__(self, background_file, ground_file):  
        self.background = Sprite(0, 0, background_file, False)  
        self.ground = Sprite(0, 0, ground_file, False)  
  
    def load(self):  
        pass  
  
    def inputs(self, events):  
        pass  
  
    def update(self, change_scene):  
        pass  
  
    def draw(self, screen):  
        pass
```

- level_00.py

```
def __init__(self, background_file, ground_file):  
    Scene.__init__(self, background_file, ground_file)  
    , screenh = pygame.display.get_surface().get_size()  
    ground_height = screen_h - self.ground.surface.get_height()  
    self.ground.y = ground_height  
    self.player = SpriteControlled(10, ground_height, 'sprite.png', True, 2)  
    self.cursor = Sprite(0, 0, 'cursor.png', False)  
    self.warp = Warp(500, 0, 'warp.png', False, "level01")  
    self.warp.y = ground_height - self.warp.surface.get_height() / 2
```

- We have a problem: when the player go back to level 00, he teleports immediately to level 01
- That is because his position is kept in memory.

Last problem

- We are teleported on the warp, so teleported back
- Solution : rather than teleport to a scene, teleport to a scene AND a x location
- level_00.py

```
class Level00(Scene):  
  
    def __init__(self, background_file, ground_file):  
        ...  
        self.warp = Warp(680, 0, 'warp.png', False, "level01", 0)
```

Last problem

- warp.py

```
def __init__(self, x, y, filename, centered, to_scene, to_scene_x):  
    Sprite.__init__(self, x, y, filename, centered)  
    self.to_scene = to_scene  
    self.to_scene_x = to_scene_x
```

- main.py

```
def change_scene(name, x):  
    nonlocal current_scene  
    current_scene = scenes[name]  
    current_scene.player.x = x  
    current_scene.player.is_moving = False
```

- level_00.py

```
def update(self, change_scene):  
    ...  
    if self.player.intersects(self.warp):  
        change_scene(self.warp.to_scene, self.warp.to_scene_x)
```

Data (optional)

This part of the lesson is OPTIONAL. Step in if you are comfortable.

Normal lesson continues at "UI"

Scene data

- In this optional part of the lesson, we will create our level design with data, and not with a subclass
- We would like to create our scene with this kind of file :

level00lvl

```
background;background  
ground;ground  
player	sprite;150;ground;2  
sprite	copain;600;ground  
warp	warp;700;ground;level01
```

- This file will be loaded with a load function, each line will be read, and we will instanciate objects in function of the line data

How to process data

- scene.py

```
def __init__(self, filename):
    self.filename = filename
    self.load(filename)

def load(self, filename):
    file = open(Scene.path + filename)
    data = file.read().splitlines()

    ground_height = 0
    self.cursor = Sprite(0, 0, 'cursor.png', False)
    self.sprites = []
    self.warps = []

    for line in data:
        cell = line.split(";")
        # Ground
        if(cell[0] == "ground"):
            self.ground = Sprite(0, 0, cell[1]+".png", False)
            _, screen_h = pygame.display.get_surface().get_size()
            ground_height = screen_h - self.ground.surface.get_height()
            self.ground.y = ground_height
```

How to process data (continued)

· scene.py

```
# Background
elif(cell[0] == "background"):
    self.background = Sprite(0, 0, cell[1]+".png", False)
# Player
elif(cell[0] == "player"):
    height = 0
    if cell[3] == "ground":
        height = -1
    self.player = SpriteControlled(int(cell[2]), height, cell[1]+".png", True, int(cell[4]))
# Sprites
elif(cell[0] == "sprite"):
    height = 0
    if cell[3] == "ground":
        height = -1
    sprite = Sprite(int(cell[2]), height, cell[1]+".png", True)
    self.sprites.append(sprite)
# Warps
elif(cell[0] == "warp"):
    height = 0
    if cell[3] == "ground":
        height = -1
    warp = Warp(int(cell[2]), height, cell[1]+".png", False, cell[4])
    self.warps.append(warp)

# Set heights
if(self.player.y == -1):
    self.player.y = ground_height
for s in self.sprites:
    if(s.y == -1):
        s.y = ground_height
for w in self.warps:
    if(w.y == -1):
        w.y = ground_height - w.surface.get_height() / 2
```

Last problem

- We are teleported on the warp, so teleported back
- Solution : rather than teleport to a scene, teleport to a scene AND a x location

level00.lvl

```
...
warp,warp;700;ground;("level01",50)
```

Solve problem

- main.py

```
def change_scene(name, x):
    nonlocal current_scene
    current_scene = scenes[name]
    current_scene.player.x = x
    current_scene.player.is_moving = False
```

- scene.py

```
def update(self, change_scene):
    ...
    for w in self.warps:
        if(self.player.intersects(w)):
            change_scene(w.to_scene, w.to_scene_x)
```

- warp.py

```
def __init__(self, x, y, filename, centered, to_scene):
    Sprite.__init__(self, x, y, filename, centered)
    self.to_scene = to_scene[0]
    self.to_scene_x = to_scene[1]
```

UI

UI Panel

- Let's create a panel
- ui_panel.py

```
import pygame

class UiPanel:
    def __init__(self, x, y, w, h):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.visible = True
        self.color = (255, 255, 255)

    def set_visible(self, value):
        self.visible = value

    def update(self):
        pass

    def draw(self, screen):
        pygame.draw.rect(screen, self.color, pygame.Rect(self.x, self.y, self.w, self.h))
```

- scene.py

```
def load(self, filename):
    ...
    self.panel = UiPanel(0, 0, 800, 100)

def update(self, change_scene):
    ...
    self.panel.update()

def draw(self, screen):
    ...
    self.panel.draw(screen)
```

Reactive panel

- Now let's make it reactive : the panel will change color when mouse is above it (hover)
- When we put the mouse on the panel, we want the panel to trigger a "hover_in" event
- Functions triggered by events are called "callbacks"
- ui_panel.py

```
class UiPanel:  
    def __init__(self, x, y, w, h):  
        ...  
        self.events = {} → # a map of event names / callbacks  
        self.is_hover = False  
        self.set_event("hover_in", self.on_hover_in) → # we use a helper function to set event  
        self.set_event("hover_out", self.on_hover_out) → # names and callback  
  
    def set_event(self, event_type, function): → # this is the helper function  
        self.events[event_type] = function  
  
    def on_hover_in(self): → # this is a callback  
        self.change_color((175, 175, 175))  
  
    def on_hover_out(self):  
        self.change_color((255, 255, 255))  
  
    def change_color(self, color):  
        self.color = color  
  
    def update(self): → # the update function checks mouse position an  
        mouse_x, mouse_y = pygame.mouse.get_pos() → # and trigger events  
        if(  
            not(self.is_hover)  
            and (mouse_x > self.x and mouse_x < self.x + self.w and mouse_y > self.y and mouse_y < self.y + self.h)  
        ):  
            self.is_hover = True  
            self.events["hover_in"]() → # select the callback in the map and call it  
        if(  
            self.is_hover  
            and not(mouse_x > self.x and mouse_x < self.x + self.w and mouse_y > self.y and mouse_y < self.y + self.h)  
        ):  
            self.is_hover = False  
            self.events["hover_out"]()
```

UI Elements

- **Ui elements are available as groups in games.** For instance, on a panel, you have multiple ui buttons and texts.
- We want to create a base class for all ui elements. This base class contains the ui position and size, the events, the visible flag
- Please code it :)

UI Elements

- ui_element.py

```
class UiElement(object):  
    def __init__(self, x, y, w, h):  
        self.x = x  
        self.y = y  
        self.w = w  
        self.h = h  
        self.visible = True  
        self.events = {}  
  
    def set_visible(self, value):  
        self.visible = value  
  
    def set_event(self, event_type, function):  
        self.events[event_type] = function
```

- Now make UiPanel a child of this class.

New UI Panel

- ui_panel.py

```
import pygame
from ui_element import UiElement

class UiPanel(UiElement):
    def __init__(self, x, y, w, h):
        UiElement.__init__(self, x, y, w, h)
        self.color = (255, 255, 255)
        UiElement.set_event(self, "hover_in", self.on_hover_in)
        UiElement.set_event(self, "hover_out", self.on_hover_out)
        self.is_hover = False

    ... former code ...
```

Ui Group

- Let's finish the refactoring by creating the `UiGroup` class
- It will only contain an array of `UiElements` and apply the usual functions on them
- We a way to easily add an element in the group
- `ui_group.py`

```
class UiGroup(object):
    def __init__(self):
        self.elements = []

    def add_element(self, element):
        self.elements.append(element)

    def set_visible(self, value):
        for e in self.elements:
            e.set_visible(value)

    def inputs(self, events):
        for e in self.elements:
            e.inputs(events)

    def update(self):
        for e in self.elements:
            e.update()

    def draw(self, screen):
        for e in self.elements:
            e.draw(screen)
```

Ui Group - Scene

· scene.py

```
def load(self, filename):
    ...
    self.ui_top = UiGroup()
    panel = UIPanel(0, 0, 800, 100)
    self.ui_top.add_element(panel)

    def inputs(self, events):
        for event in events:
            ...
            self.ui_top.inputs(events)

    def update(self, change_scene):
        ...
        self.ui_top.update()
```

Ui Button

- The UiButton has an hover_in and hover_out events, but also a click event and a release event
- Mouse will be managed in the inputs() function
- ui_button.py

```
import pygame
from ui_element import UiElement

class UiButton(UiElement):
    def __init__(self, x, y, w, h):
        UiElement.__init__(self, x, y, w, h)
        self.color = (0, 0, 255)
        UiElement.set_event(self, "hover_in", self.on_hover_in)
        UiElement.set_event(self, "hover_out", self.on_hover_out)
        UiElement.set_event(self, "click", self.on_click)
        UiElement.set_event(self, "release", self.on_release)
        self.is_hover = False
        self.is_clicked = False

    def on_hover_in(self):
        self.change_color((175, 175, 175))

    def on_hover_out(self):
        self.change_color((0, 0, 255))

    def on_click(self):
        self.change_color((255, 255, 0))

    def on_release(self):
        if(self.is_hover):
            self.change_color((175, 175, 175))
        else:
            self.change_color((0, 0, 255))

    def change_color(self, color):
        self.color = color

    def inputs(self, events):
        for event in events:
            if event.type == pygame.MOUSEBUTTONDOWN:
                mouse_x, mouse_y = pygame.mouse.get_pos()
                if(mouse_x > self.x and mouse_x < self.x + self.w and mouse_y > self.y and mouse_y < self.y + self.h):
                    self.is_clicked = True
                    self.events["click"]()
            if event.type == pygame.MOUSEBUTTONUP:
                self.is_clicked = False
                self.events["release"]()

    def update(self):
        mouse_x, mouse_y = pygame.mouse.get_pos()
        if(
            not(self.is_hover)
            and (mouse_x > self.x and mouse_x < self.x + self.w and mouse_y > self.y and mouse_y < self.y + self.h)
        ):
            self.is_hover = True
            self.events["hover_in"]()
        if(
            self.is_hover
            and not(mouse_x > self.x and mouse_x < self.x + self.w and mouse_y > self.y and mouse_y < self.y + self.h)
        ):
            self.is_hover = False
            self.events["hover_out"]()

    def draw(self, screen):
        pygame.draw.rect(screen, self.color, pygame.Rect(self.x, self.y, self.w, self.h))
```

Ui Button Fix

- We have a little bug : when we click on the button, the player moves
- We fix this bug by not triggering the click event for the player when the mouse is over the panel
- This is quite dirty, add a comment to highlight the trick
- scene.py

```
def inputs(self, events):
    for event in events:
        if event.type == pygame.MOUSEBUTTONDOWN:
            mouse_click = pygame.mouse.get_pos()
            if(mouse_click[1] > self.ui_top.elements[0].h): # Dirty but effective
                self.player.move_to(mouse_click[0])
...
...
```

Ui Button Images

- Now we would like to use sprites instead of colors on the button
- We just have to load an idle sprite, a hover sprite and a clicked sprite, and define a current sprite
- The events management code is the same

Ui Button Images

· ui_button.py

```
class UiButton(UiElement):
    def __init__(self, x, y, w, h, filename):
        UiElement.__init__(self, x, y, w, h)
        UiElement.set_event(self, "hover_in", self.on_hover_in)
        UiElement.set_event(self, "hover_out", self.on_hover_out)
        UiElement.set_event(self, "click", self.on_click)
        UiElement.set_event(self, "release", self.on_release)
        self.is_hover = False
        self.is_clicked = False
        self.sprite_idle = Sprite(x, y, filename+"_idle.png", False)
        self.sprite_hover = Sprite(x, y, filename+"_hover.png", False)
        self.sprite_click = Sprite(x, y, filename+"_click.png", False)
        self.current_sprite = self.sprite_idle

    def on_hover_in(self):
        self.current_sprite = self.sprite_hover

    def on_hover_out(self):
        self.current_sprite = self.sprite_idle

    def on_click(self):
        self.current_sprite = self.sprite_click

    def on_release(self):
        if(self.is_hover):
            self.current_sprite = self.sprite_hover
        else:
            self.current_sprite = self.sprite_idle
```

Tidying

- Our folder is a mess
- Create a folder for images, create an other folder for data, if you did the optional part
- Update your absolute paths

UI messaging

Ingame effect when we click a button

- We would like a sprite to change state (e.g. its image) when we click a button
- First, let's make a sprite subclass with a state
- Those SpriteStateful will have an array of states, an array of filenames (sort in the same order) and a change_state function

Stateful Sprite

- sprite_stateful.py

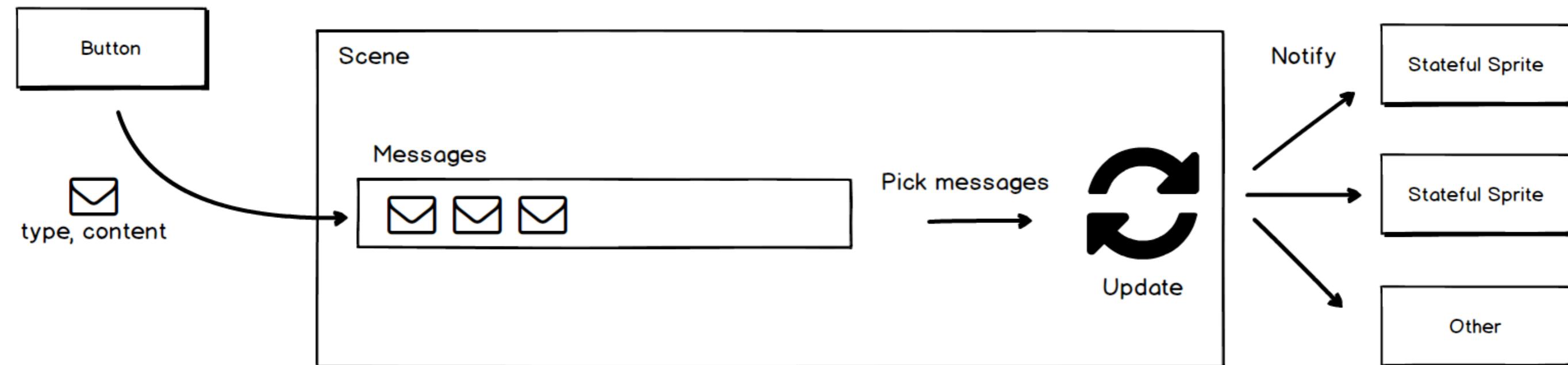
```
import pygame
from sprite import Sprite

class SpriteStateful(Sprite):
    def __init__(self, x, y, filenames, centered, states, base_state):
        self.states = states
        self.filenames = filenames
        Sprite.__init__(self, x, y, filenames[0], centered)
        self.change_state(base_state)

    def change_state(self, new_state):
        index = self.states.index(new_state)
        self.surface = pygame.image.load(Sprite.path + self.filenames[index]).convert_alpha()
        self.current_state = new_state
```

Taking time to think

- We want to transmit to our stateful sprite the information that the button has been clicked
- We could have several sprite interested by this information
- We could have other classes (e.g.: some class that would count the button clicks)
- So we would have:
 - * An array of messages in the scene
 - * An array of observers that will get the messages
 - * The buttons will send the messages
 - * In update, we will check if there are messages. If so, we will give the messages to the observers
 - * The observers will execute a method if the message fit them



Messages

· message.py

```
class Message:  
    def __init__(self, type, content):  
        self.type = type  
        self.content = content
```

· scene.py

```
def load(self, filename):  
    ...  
    self.messages = []  
    self.observers = []  
    ...  
    self.observers.append(sprite) → # when we create stateful sprites  
    ...  
  
def update(self, change_scene):  
    ...  
    for message in self.messages:  
        for observer in self.observers:  
            observer.notify(message)  
    self.messages.clear() → # empty messages after they are broadcast  
  
def send_message(self, message):  
    self.messages.append(message)
```

· sprite_stateful.py

```
class SpriteStateful(Sprite):  
    def __init__(self, x, y, filenames, centered, states, base_state):  
        ...  
        self.change_state(base_state)  
    ...  
    def notify(self, message):  
        if message.type == "change_state":  
            self.change_state(message.content)
```

Message creation

- Now button just have to create messages
- ui_button.py

```
class UiButton(UiElement):  
    def __init__(self, x, y, w, h, filename, send_message):  
        ...  
        self.send_message = send_message  
    ...  
    def on_click(self):  
        self.current_sprite = self.sprite_click  
        self.send_message(Message("change_state", "happy"))
```

- The method we use for broadcasting messages is a variation of the "Observer pattern"

Inventory (optional)

*This part of the lesson is OPTIONAL. Step in if you are comfortable.
Normal lesson continues at "Animation"*

Pickable sprite

- We want to be able to pick a sprite : colliding it give it a pick state
- When state is picked, the item won't be displayed anymore

(We can use `Sprite_Stateful`, though the code presented here won't.)

- We want an inventory that will be kept when we change scene : it will be defined in main

Pickables sprites in scene

· main.py

```
def main():
    # Load
    ...
    inventory = []
    level00 = Scene("level00.lvl", inventory)
    level01 = Scene("level01.lvl", inventory)
```

· scene.py

```
def __init__(self, filename, inventory):
    self.inventory = inventory
    ...
    self.pickables = []
    ...
    # Code to add the pickables

def update(self, change_scene):
    ...
    for p in self.pickables:
        if(self.player.intersects(p)):
            self.add_to_inventory(p.pick())
    ...

def add_to_inventory(self, item):
    self.inventory.append(item)
```

· sprite_pickable.py

```
import math
from sprite import Sprite

class SpritePickable(Sprite):

    def __init__(self, x, y, filename, centered, name):
        Sprite.__init__(self, x, y, filename, centered)
        self.name = name
        self.is_picked = False

    def pick(self):
        self.is_picked = True
        return self.name

    def draw(self, screen):
        if(not(self.is_picked)):
            Sprite.draw(self, screen)
```

Inventory

- The inventory is simply a list of item name
- We want, for each item in the inventory, to add a button in the top ui
- Clicking on this button will represent using the object

Add buttons in UI

- scene.py

```
def add_to_inventory(self, item):
    x = self.inventory.__len__() * 95 + 10
    y = 10
    w = 90
    h = 90
    self.ui_top.add_element(UiButton(x, y, w, h, item))
    self.inventory.append(item)
```

- This will create a bug: the buttons will be added in the UI even if it has already been picked (while we stand on the pickable)
- So the game will slow
- Furthermore: the UI will not display the item when we change scene, because UI is created with scene
- We have to update UI only once when we pick an item and when we change scene

Inventory fix

· scene.py

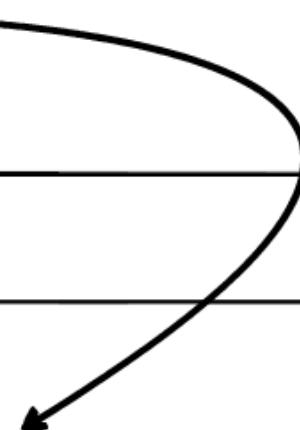
```
def add_to_inventory(self, item):
    self.inventory.append(item)
    self.update_inventory_ui()

def update_inventory_ui(self):
    i = 0
    for item in self.inventory:
        x = i * 95 + 10
        y = 10
        w = 90
        h = 90
        self.ui_top.add_element(UiButton(x, y, w, h, item))
        i = i + 1

def after_change(self):
    self.update_inventory_ui()
```

· main.py

```
def change_scene(name, x):
    ...
    current_scene.after_change()
```



Animation

Animation

- We will start with a SpriteController child class : we want our animated sprite to move
- Our SpriteAnimated will HAVE an animation (composition over inheritance)
- An animation is simply a frame shown a certain amount of time, then replace by the next frame
- An Animation will have a file with all frames
- The SpriteAnimated will manage animation selection, the Animation will manage frame update

Animation code

Let's start with the Animation class :

- animation.py

```
import pygame

class Animation:
    path = 'D:\Code\ArtFx\Python\python-training\01.adventure\28.animation\images\'

    def __init__(self, frame_number, frame_speed, image):
        self.frame_number = frame_number
        self.current_frame = 0
        self.frame_speed = frame_speed
        self.counter = 0
        self.surface = pygame.image.load(Animation.path + image).convert_alpha()
        self.width, self.height = self.surface.get_width() / frame_number, self.surface.get_height()

    def update(self):
        self.counter = self.counter + 1
        if self.counter > self.frame_speed:
            self.counter = 0
            self.current_frame = (self.current_frame + 1) % self.frame_number

    def get_surface(self):
        return self.surface.subsurface(pygame.Rect(self.current_frame * self.width, 0, self.width, self.height))
```

Next frame or starting frame if we are on the last frame

Sprite Animated code

The SpriteAnimated will contain animations in a map :

- sprite_animated.py

```
import pygame
from sprite_controlled import SpriteControlled
from animation import Animation

class SpriteAnimated(SpriteControlled):
    def __init__(self, x, y, centered, speed, base_animation):
        self.animations = {}
        self.animations["idle"] = Animation(1, 50, "sprite_idle.png")
        self.animations["walking"] = Animation(8, 10, "sprite_walking.png")
        SpriteControlled.__init__(self, x, y, "sprite_idle.png", centered, speed)
        self.current_animation = base_animation

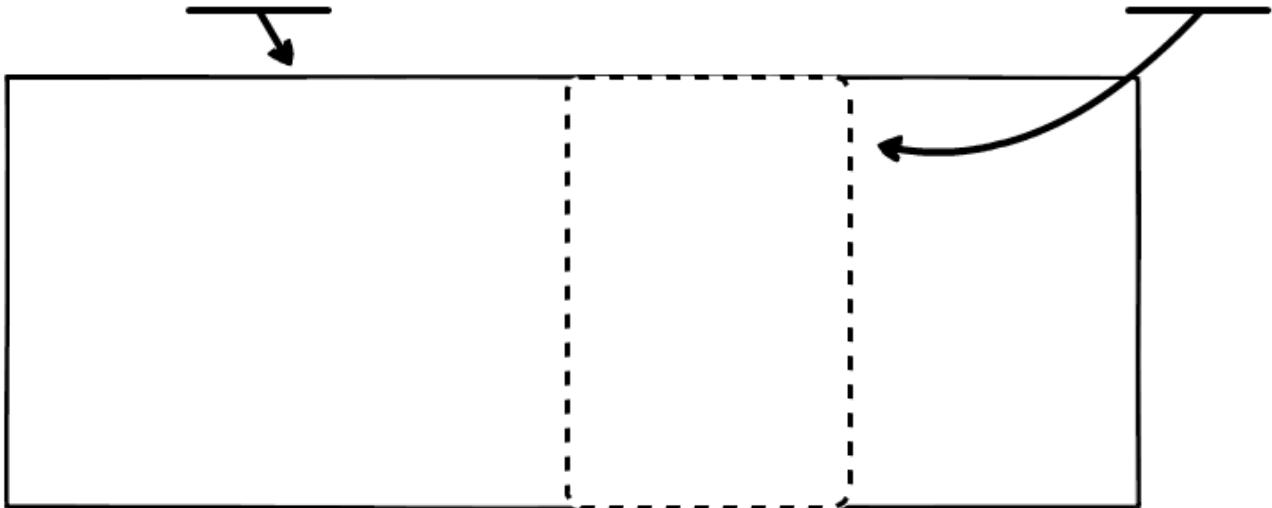
    def update(self):
        self.animations[self.current_animation].update() → # update current animation
        SpriteControlled.update(self) → # update sprite move
        if self.is_moving:
            self.current_animation = "walking"
        else:
            self.current_animation = "idle"

    def draw(self, screen):
        screen.blit(self.animations[self.current_animation].get_surface(), (self.x + self.ox, self.y + self.oy))
```

You just have to use you SpriteAnimated class in your scene. If you want multiple animated sprites, use inheritance.

Animation

- The animation code create some collision bugs. The reason is simple.
- The Sprite class we inherited from use the image size to check the collision. But it should use only the frame size.



- Let's update our code to use the right size.
- We also have to deal with ox and oy

Update sprite code

We create a width member in the Sprite class :

· sprite.py

```
def __init__(self, x, y, filename, centered):
    ...
    self.width = self.surface.get_width()
    self.height = self.surface.get_height()
    self.centered = centered
    if(centered):
        self.ox = -self.width / 2
        self.oy = -self.height

    def intersects(self, sprite):
        x1, y1, w1, h1 = self.x + self.ox, self.y + self.oy, self.width, self.height
        x2, y2, w2, h2 = sprite.x + sprite.ox, sprite.y + sprite.oy, sprite.width, sprite.height
        return not(x1 + w1 < x2 or x2 + w2 < x1 or y1 + h1 < y2 or y2 + h2 < y1)
```

SpriteAnimated change this width member when the animation changes:

· sprite_animated.py

```
def update(self):
    self.animations[self.current_animation].update()
    SpriteControlled.update(self)
    if self.is_moving:
        self.current_animation = "walking"
        self.update_width_height() ←
    else:
        self.current_animation = "idle"
        self.update_width_height() ←

    def update_width_height(self):
        self.width = self.animations[self.current_animation].get_surface().get_width()
        self.height = self.animations[self.current_animation].get_surface().get_height()
        if(self.centered):
            self.ox = -self.width / 2
            self.oy = -self.height
```

