A modular adventure game

Gaëtan Blaise-Cazalet - Head of programming section

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()
```

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

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()
```

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.)

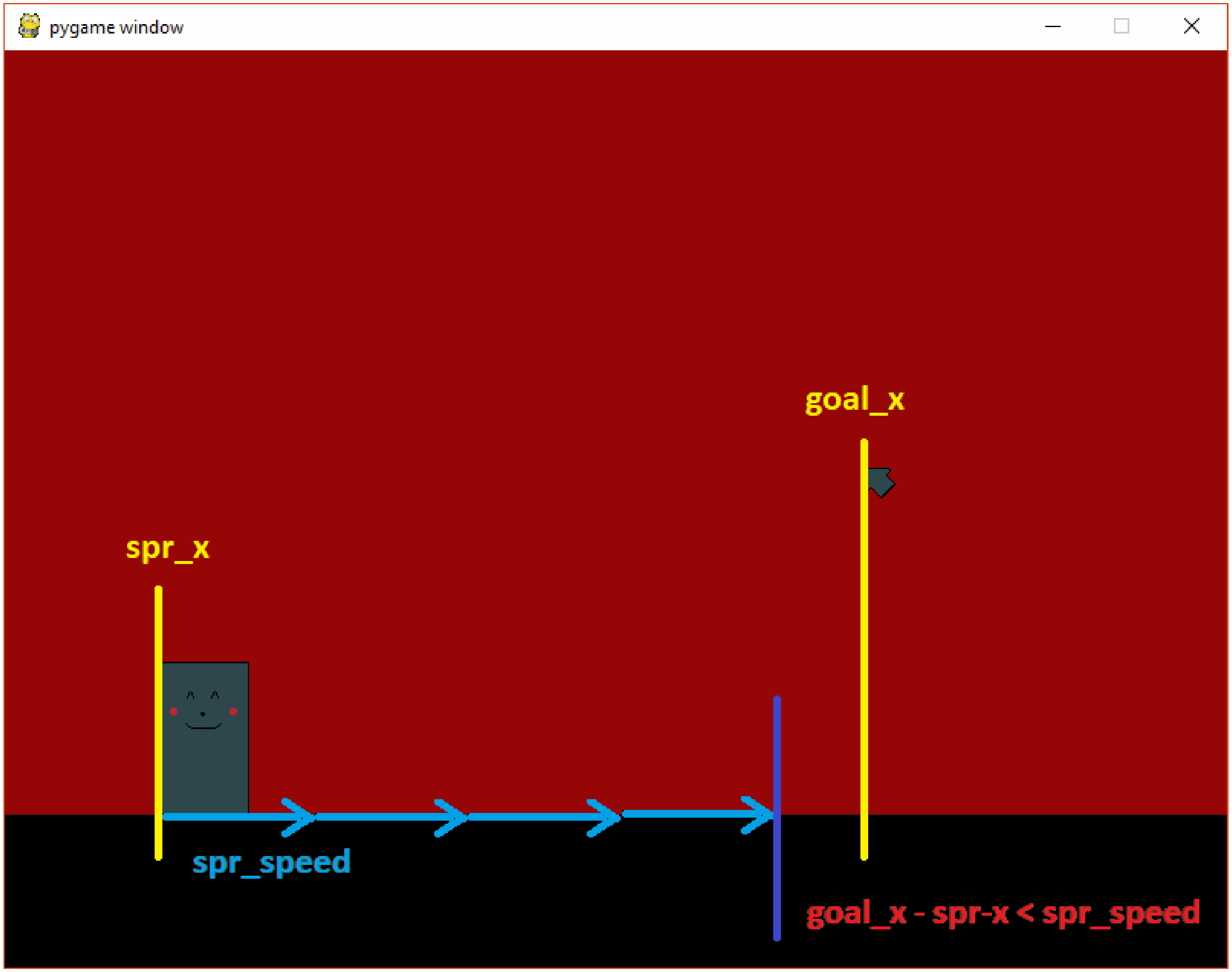
Progressive move

- 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
```

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)
    ....
```

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 INSTANCIATE 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 omited 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

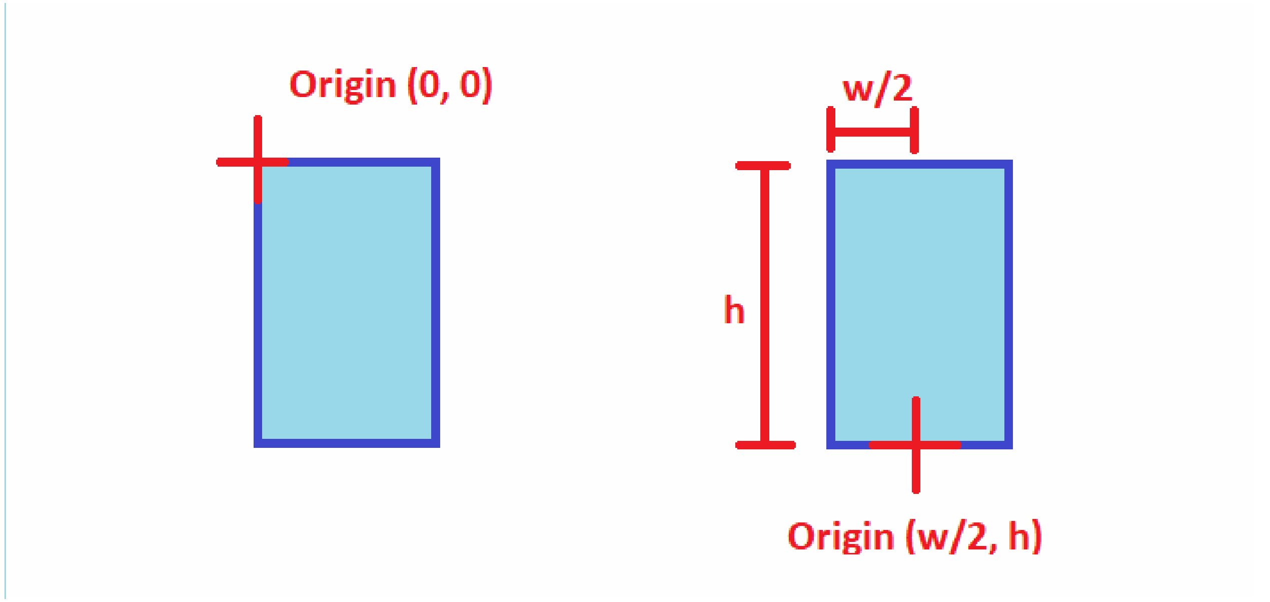
```
from sprite_controlled import SpriteControlled

    # 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                                        # New constructor argument. Update main !
    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))
```

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 continuated

- 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)
```

Function is passed as an argument

- scene.py

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

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

- When we will want to create a menu, we will use subclass. But for now, let's try data

- We would like to create our scene with this kind of file :

level00.lvl

```
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 thant 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)
```