Demo – Project – Flappy Bird

We will create a new file, ground.py in the entities folder. Using the ground image, we will create the moving picture of the ground. The size of the ground is 37 X 128. We can do it by using a loop.

As the width of the ground is 37, ground.png should be 37 width apart from each other. The image is above the ground, so we first need to make the ground.

Using the `for` loop we will be printing, **`screen.blit(self.image, ground_rect)`**. We need multiple images of the ground with some distance apart. We will create a function, **`self.images = get_ground_image()`** which will be an array.

Back to the for loop, we will give the position of the ground image using the **`current_x`** and **`ground_rect.centerx`**. we have already set the position of the ground, before the for loop so that the following for loop will run the image in a loop. We are printing all the images of the loop at the same time.

main.py | ground.py | ground.png | pipe.png | background.py | constants.py

entities > ground.py > Ground > display

```python
import pathlib
import os
import pygame

class Ground:
    def __init__(self):
        self.width = 37
        self.height = 128

        current_path = pathlib.Path(__file__).parent.parent.absolute()
        path =  os.path.join(current_path, "assets", "images", "ground.png")

        self.images = [pygame.image.load(path) for _ in range(10)]


    def display(self, screen):
        ground_rect = self.images[0].get_rect()
        ground_rect.center = (100, 800)

        current_x = self.width
        for i in range(10):
            current_x += self.width
            ground_rect.centerx += current_x
            screen.blit(self.images[i], ground_rect)
```

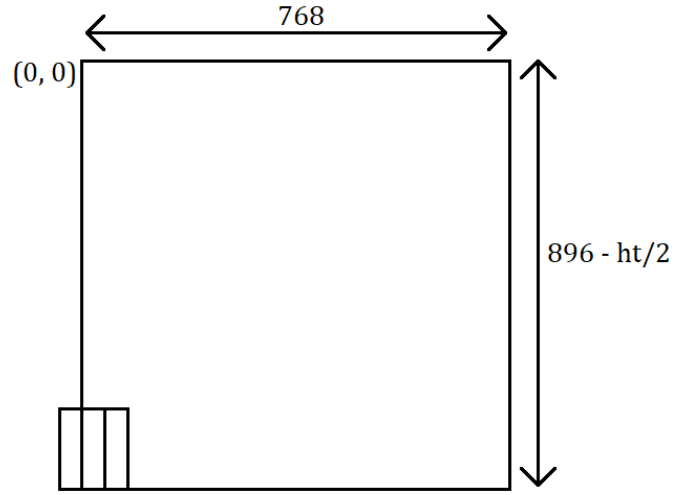
We have got four images; we need to reduce the gap and club-them-up.

The first image, the center should be **`self.width//2`**. This will put the image.png at the left end of the screen. We have to repeat the same image again and again until it fills the whole screen.

To repeat the second image, if we use **`self.width//2 + self.width`**, we will have two image.pngs' side-by-side.

```
main.py   ground.py ×   ground.png   pipe.png   background.py   constants.py
entities > ground.py > Ground > display
import pathlib
import os
import pygame

class Ground:
    def __init__(self):
        self.width = 37
        self.height = 128

        current_path = pathlib.Path(__file__).parent.parent.absolute()
        path =  os.path.join(current_path, "assets", "images", "ground.png")

        self.images = [pygame.image.load(path) for _ in range(10)]


    def display(self, screen):
        ground_rect1 = self.images[0].get_rect()
        ground_rect1.center = (self.width/2, 800)
        screen.blit(self.images[0], ground_rect1)

        ground_rect2 = self.images[0].get_rect()
        ground_rect2.center = (self.width/2 + self.width, 800)
        screen.blit(self.images[0], ground_rect2)

```

The size of the game window is 768 X 896 and the size of the ground.png is 37 X 128. The (0, 0) is at the top left of the screen.



So, if we place the image at the point of (0, y), then half of the ground will be outside the window and the remaining half will be visible to us on the screen. We then repeat the same for the next gound.png file. If 'ht' is the height of the tile, then 896-ht/2 will the position of y, so the image is placed at (width/2, 896-ht/2).

All we need to do is repeat the logic using for loop. Every time we are adding, we need a variable ‘dx’ which is initially ‘0’ and it will be incremented by **`self.width`**.

```python
    def __init__(self):
        self.width = 37
        self.height = 128

        current_path = pathlib.Path(__file__).parent.parent.absolute()
        path =  os.path.join(current_path, "assets", "images", "ground.png")

        self.images = [pygame.image.load(path) for _ in range(10)]


    def display(self, screen):
        dx = 0
        for i in range(10):
            ground_rect = self.images[i].get_rect()
            ground_rect.center = (self.width/2 + dx, Constants.height - self.height / 2)
            screen.blit(self.images[i], ground_rect)
            dx += self.width
```

We need (768 / 37) = 24 tiles.

```python
import pygame


class Ground:
    def __init__(self):
        self.width = 37
        self.height = 128

        current_path = pathlib.Path(__file__).parent.parent.absolute()
        path =  os.path.join(current_path, "assets", "images", "ground.png")

        self.images = [pygame.image.load(path) for _ in range(21)]


    def display(self, screen):
        dx = 0
        for i in range(21):
            ground_rect = self.images[i].get_rect()
            ground_rect.center = (self.width/2 + dx, Constants.height - self.height / 2)
            screen.blit(self.images[i], ground_rect)
            dx += self.width
```

Let us have a function which gives us position of the array (**`get_positions():`**). We will use this function for images of the ground and in the **`display()`** function we are using the image positions to update the center.

```
    def get_positions(self):
        positions = list()
        dx = 0
        for _ in range(21 * 2):
            positions.append((self.width/2 + dx + self.tick, Constants.height - self.hei
            dx += self.width
        self.tick += 1
        return positions

    def display(self, screen):
        image_positions = self.get_positions()
        for i in range(21 * 2):
            ground_rect = self.images[i].get_rect()
            ground_rect.center = image_positions[i]
            screen.blit(self.images[i], ground_rect)

```

Now, we will set our condition that if the platform moves towards the negative then then platform comes back to its original position.

```
    def get_positions(self):
        positions = list()
        dx = 0
        for _ in range(21 * 2):
            x = self.width/2 + dx + self.tick
            y = Constants.height - self.height / 2
            if x < -1 * Constants.width:
                x = Constants.width
            positions.append((x, y))
            dx += self.width
        self.tick += 1
        return positions

    def display(self, screen):
        image_positions = self.get_positions()
        for i in range(21 * 2):
            ground_rect = self.images[i].get_rect()
            ground_rect.center = image_positions[i]
            screen.blit(self.images[i], ground_rect)
```

We need to optimize it as we are using lot of images.

We are using a **`draw_platform`** function place the platform at a particular position, on the game screen and **`update_platform`** function is used to move the platform.

```python
    def draw_platform(self):
        dx = 0
        for _ in range(21 * 2):
            x = self.width/2 + dx
            y = Constants.height - self.height / 2
            self.positions.append((x, y))
            dx += self.width


    def update_platform(self):
        for idx in range(len(self.positions)):
            x, y = self.positions[idx]
            if x + self.tick < -1 * Constants.width:
                self.positions[idx] = (Constants.width, y)
                self.tick = 0
            else:
                self.positions[idx] = (x + self.tick, y)
            self.tick -= 1
```

We can see the platform running for infinite times but with gap and at a high speed. The issue is we made a function to draw the platform and then in the display we have updated platform. In this position array, it is updating the position of each and every block in the array, and then it gets displayed.

```python
class Ground:
    def __init__(self):
        self.width = 37
        self.height = 128

        current_path = pathlib.Path(__file__).parent.parent.absolute()
        path =  os.path.join(current_path, "assets", "images", "ground.png")

        self.images = [pygame.image.load(path) for _ in range(21 * 2)]
        self.positions = list()
        self.draw_platform()

    def draw_platform(self):
        dx = 0
        for _ in range(21 * 2):
            x = self.width/2 + dx
            y = Constants.height - self.height / 2
            self.positions.append((x, y))
            dx += self.width
```

```python


    def update_platform(self):
        speed = 1
        for idx in range(len(self.positions)):
            x, y = self.positions[idx]
            if x - speed <= -1 * Constants.width:
                self.positions[idx] = (Constants.width + self.width / 2 - speed, y)
            else:
                self.positions[idx] = (x - speed, y)

    def display(self, screen):
        self.update_platform()
        print("current positions", self.positions)
        for i in range(21 * 2):
            ground_rect = self.images[i].get_rect()
            ground_rect.center = self.positions[i]
            screen.blit(self.images[i], ground_rect)

```

We can create a class Sprite in sprite.py. Using inheritance, we will put all display function into sprite.

Let's try to jump the bird. Bird is an object and we will define a method **`jump()`.** If we are jumping the image, the value of 'y' will increase. There will be gravity acting on the bird to pull it down. If we are not using the Up-key then the gravity should pull it down.

When we are jumping, we update the position. In the game x-coordinate will not change when we jump, only the y-coordinate would change. We can give a jump value to the function. Then we will make another function for gravity.

We will finish this tomorrow.

https://github.com/joshi95/flappy-bird