PYGAME.

ЦЕЛЬ: **Разработать алгоритм проверки условия пересечения спрайтов.**

ПРИМЕЧАНИЕ:

ПЛАНИРОВАНИЕ

1. Этап

Как уже, я думаю, все поняли, положение каждого спрайта определяется его координатами. Давайте более детально рассмотрим, как же определяются эти координаты.

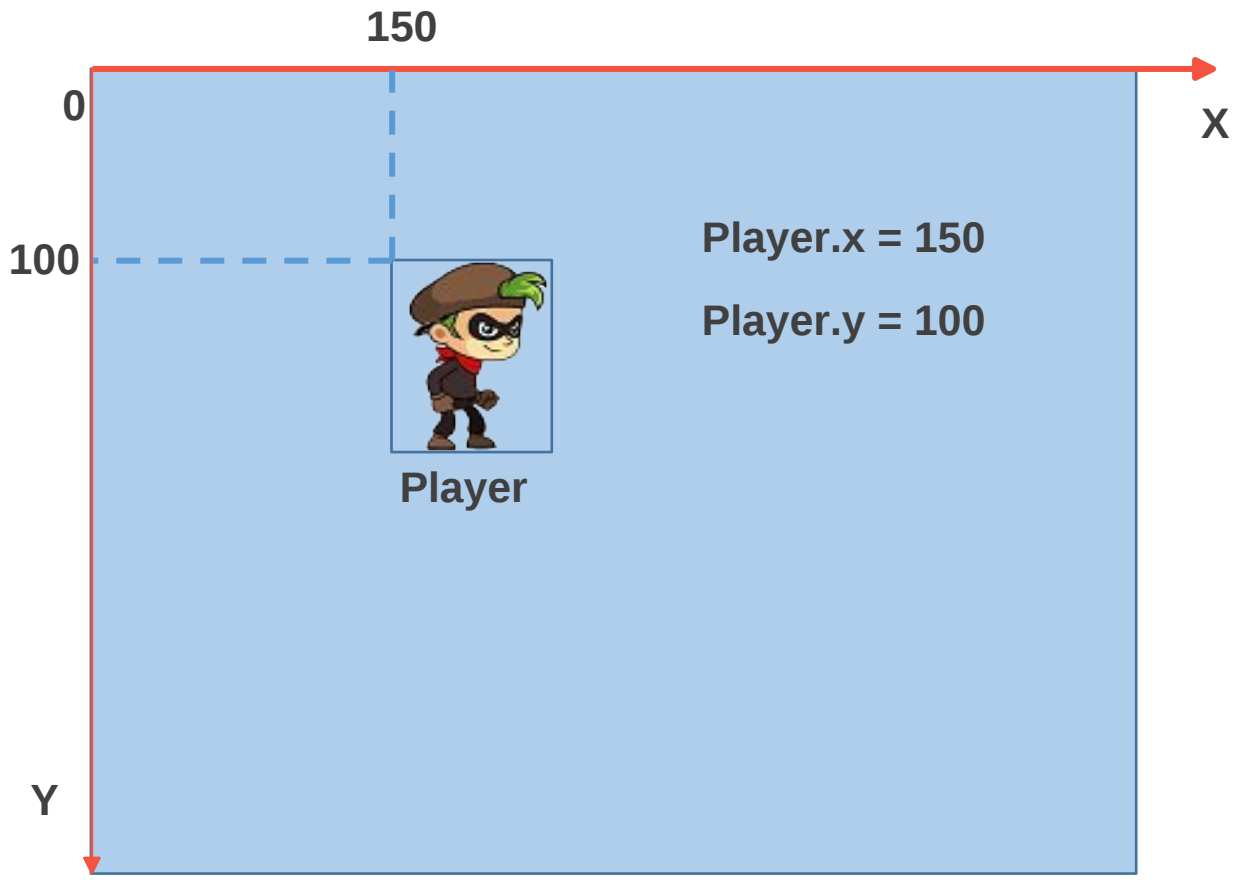
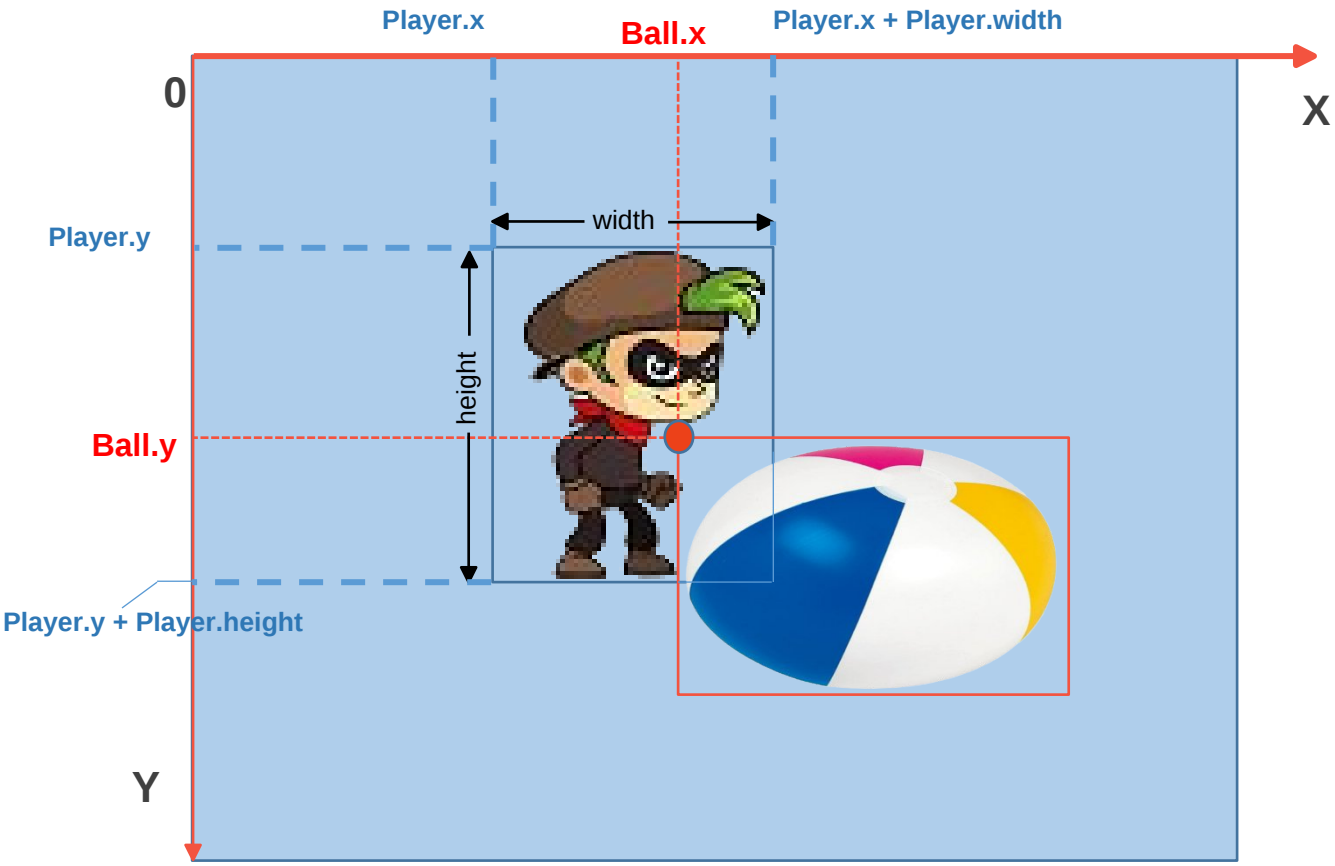


Рисунок 1

На рисунке 1 представлена схема, как располагается система координат в pygame и как определяются координаты спрайта в этой системе. Начало системы координат, как видно расположено в левом верхнем углу окна (поверхности), а точка, по которой определяются координаты спрайта, расположена в его левом

верхнем углу. На схеме спрайт обведен прямоугольником, чтобы показать что для pygame изображение спрайта – это изображение которое хранится в файле, а изображение в файле является прямоугольной областью с разноцветными пикселами.

А теперь посмотрим как на такой схеме будет выглядеть событие, когда спрайты пересеклись (Рисунок 2).

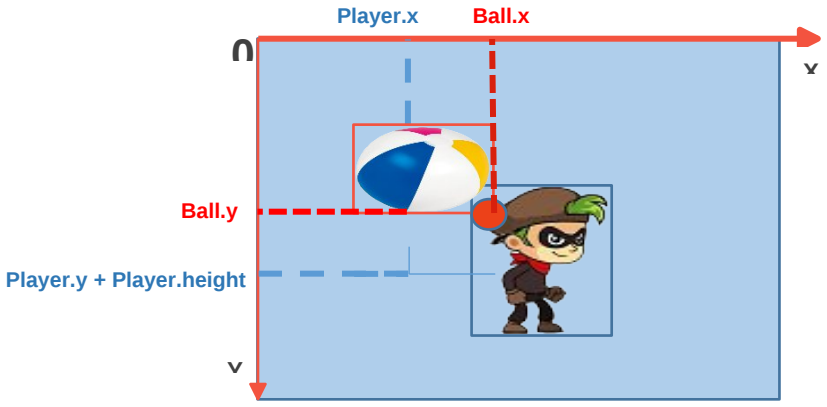


Математически условие пересечения спрайтов можно записать так:

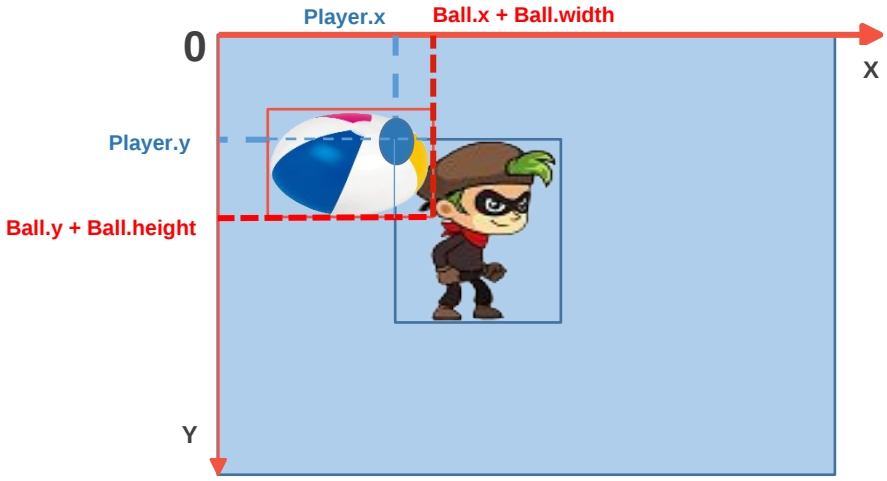
Player.x **<** Ball.x **<** Player.x + Player.width **AND** Player.y **<** Ball.y **<** Player.y + Player.height

Можно сказать, что спрайт Ball пересек спрайт Player своим левым верхним углом, значит есть еще три варианта пересечения:

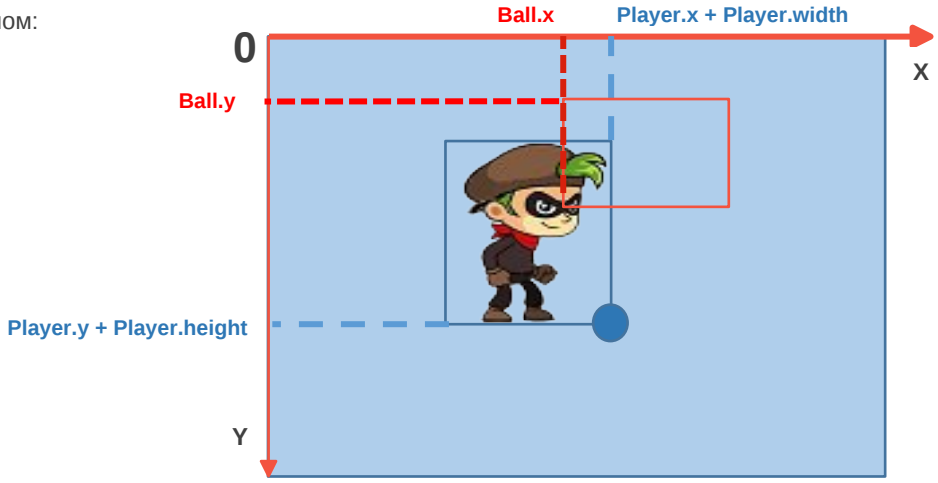
- Правым нижним углом:



- И другой спрайт левым верхним углом:



И правым нижним углом:



Получаем, что должно выполняться одно из четырех условий:

Player.x < Ball.x < Player.x + Player.width **AND** Player.y < Ball.y < Player.y + Player.height

Player.x<Ball.x +Ball.widht < Player.x + Player.width **AND** Player.y < Ball.y+Ball.Height < Player.y+Player.height

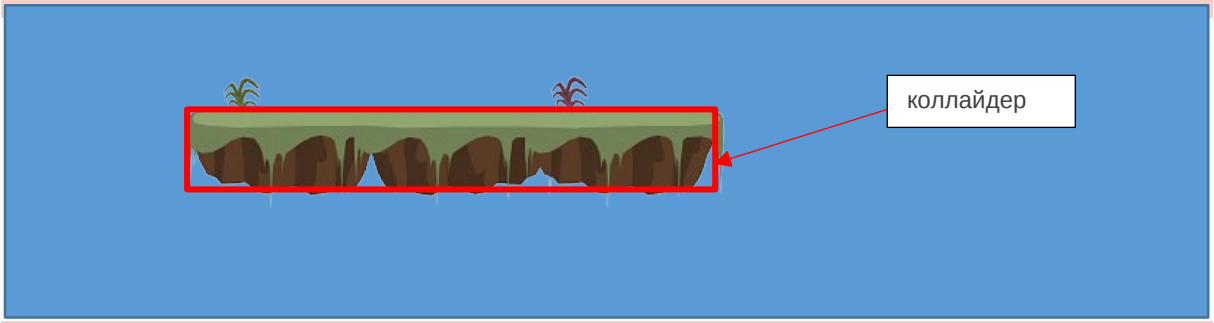
Ball.x < Player.x < Ball.x + Ball.width **AND** Ball.y < Player.y + Player.height < Ball.y + Ball.height

Ball.x < Player.x + Player.width < Ball.x + Ball.width **AND** Ball.y < Player.y < Ball.y + Ball.height

Теперь создадим в нашей программе функцию, которая будет проверять пересечение двух спрайтов.

```
def collide (Sprite1, Sprite2):
    if ((Sprite1.x <=  Sprite2.x  <= Sprite1.x + Sprite1.width
        AND   Sprite1.y <= Sprite2.y <= Sprite1.y + Sprite1.height    )
        OR   (Sprite1.x<=Sprite2.x +Sprite2.widht  <= Sprite1.x + Sprite1.width
        AND   Sprite1.y <= Sprite2.y+Sprite2.Height <= Sprite1.y+Sprite1.height)
        OR   (Sprite2.x <= Sprite1.x <= Sprite2.x + Sprite2.width
        AND   Sprite2.y <= Sprite1.y+ Sprite1.height <= Sprite2.y + Sprite2.height  )
        OR   (Sprite2.x <= Sprite1.x + Sprite1.width <= Sprite2.x + Sprite2.width
        AND    Sprite2.y <= Sprite1.y <= Sprite2.y + Sprite2.height)) :
            return  True
```

Важно: В создании игр есть такое понятие – коллайдер. Коллайдер – это воображаемая область спрайта показывающая границы, пересечение которых считается касанием спрайта:



! Проблемная задача: запрограммировать функцию. Создать второй спрайт. При пересечении спрайтов изменить изображение спрайта.

Решение:

```
import pygame

# вспомогательная функция, показывает коллайдер спрайто

def ShowCollide(Spr, w):

    pygame.draw.rect(w, (0, 200, 64), Spr, 3)


def collide(Sprite1, Sprite2):

    if ((    Sprite1.x <=  Sprite2.x  <= Sprite1.x + Sprite1.width

        and   Sprite1.y <= Sprite2.y <= Sprite1.y + Sprite1.height)

        or   (Sprite1.x <= Sprite2.x + Sprite2.width  <= Sprite1.x + Sprite1.width

        and   Sprite1.y <= Sprite2.y + Sprite2.Height <= Sprite1.y + Sprite1.height)

        or   (Sprite2.x <= Sprite1.x + Sprite1.width  <= Sprite2.x + Sprite2.width

        and   Sprite2.y <= Sprite1.y <= Sprite2.y + Sprite2.height)

        or   (Sprite2.x <= Sprite1.x <= Sprite2.x + Sprite2.width

        and   Sprite2.y <= Sprite1.y + Sprite1.height <= Sprite2.y + Sprite2.height)):
```

```
            return  True
    else:
        return False


class Sprite():
    def __init__(self, x, y, speed, img):
        self.speed = speed
        self.image = pygame.image.load(img)
        self.rect = self.image.get_rect(topleft = (x, y))
    def setimage(self, img):
        self.image = pygame.image.load(img)



Ncadr = 0
w = pygame.display.set_mode ((1279, 700))
Player = Sprite(100, 100, 1, 'Images/Player.png')

# animation for go to right
ImgPlayerGoR = [pygame.image.load('Images/GoAnim/b1.png'),
                pygame.image.load('Images/GoAnim/b2.png'),
                pygame.image.load('Images/GoAnim/b3.png'),
                pygame.image.load('Images/GoAnim/b4.png'),
                pygame.image.load('Images/GoAnim/b5.png')]

# animation for go to left
ImgPlayerGoL = []
for img in ImgPlayerGoR:
    ImgPlayerGoL.append(pygame.transform.flip(img, True, False))

# тестовый спрайт - Дерево
Tree = Sprite(400, 100, 0, 'Images/Tree.png')

# Игрвой цикл
game = True
while game:
    for ev in pygame.event.get ():
        if ev.type == pygame.QUIT:
            game = False
    # test for collide
```

```
    if collide(Player.rect, Tree.rect):
        Tree.setimage('Images/Player.png')
    else:
        keys = pygame.key.get_pressed()
        if keys[pygame.K_RIGHT]:
            Player.rect.x += Player.speed
            Player.image = ImgPlayerGoR[Ncadr % 5]
        elif keys[pygame.K_LEFT]:
            Player.rect.x -= Player.speed
            Player.image = ImgPlayerGoL[Ncadr % 5]

    w.fill((0, 0, 0))
    w.blit(Player.image, Player.rect)
    w.blit(Tree.image, Tree.rect)
    #ShowCollide(Player.rect, w)
    #ShowCollide(Tree.rect, w)
    pygame.display.update()
    Ncadr += 1

pygame.quit ()
```

2. Класс Rect

Так как все поверхности в **pygame** являются прямоугольниками, и вся физика спрайтов рассчитывается исходя из их координат и размера – ширины и высоты, то получается что спрайт по сути задается прямоугольником с 4-мя параметрами: **x, y, height, width.** И в pygame для такого удобства есть класс – Rect. Это класс, который описывает прямоугольник с 4-мя параметрами: : **x, y, height, width.**

Такой командой можно создать объект – прямоугольник:

rect = pygame.Rect(x, y, height, width)

можно и в таком формате:

rect = pygame.Rect((x, y), (height, width))

И еще кое-что новое про метод – **blit():**

Мы его использовали так:

w.blit(Player.image, (Player.x, Player.y))

Но вместо координат, оказывается в него можно передавать просто объект – Rect

w.blit(Player.image, rect)

Пример:

rect = pygame.Rect((300, 400), (20, 40))

w.blit(Player.image, rect)

Спрайт отобразится в координатах: 300, 400

И еще одна фишка это метод – **get_rect()** у объекта **Surface.** Он возвращает объект Rect, который соответствует размерам изображения.

Значит мы можем в конструкторе спрайта написать так:

self.rect = self.image.get_rect()

Но у объекта **rect** есть еще две координаты **x** и **y.** Чтобы их задать можно написать:

self.rect = self.image.get_rect(topleft = (x, y))

вместо topleft, можно написать и другой атрибут (topright, bottomleft, center)

Теперь давайте модифицируем программу, уберем из класса Sprite свойства **x, y, height, width** и создадим одно новое – **rect**

! Проблемная задача. Модифицировать программу используя у спрайта свойство – rect

Создать свою функцию ShowCollide(), которая отображает коллайдер спрайта.

Решение:

```
def ShowCollide(Spr, w):
    pygame.draw.rect(w, (0, 200, 64), Spr, 3)
```

3. Рефлексия

- Сегодня мы научились программировать проверку пересечения двух спрайтов. Узнали понятие - коллайдер

- Познакомились с классом Rect

- модифицировали программу и создали функцию , которая отображает коллайдеры спрайтов.

-