Урок PG. Спрайты

Изображения. Спрайты

Аннотация

В этом уроке мы начнем разговор об изображениях и спрайтах — анимационных элементах игры.

1. Изображения

До этого момента мы только рисовали на холсте. Однако игры, в которых не используются созданные профессиональными художниками изображения, встречаются очень редко.

Поэтому в Pygame есть модуль `image` для работы с изображениями разных форматов. Документация на него находится **здесь**.

При этом в Pygame нет специального класса `Image` (как, например, в PIL или в PyQT), а изображения представлены объектами класса `Surface`, с которым мы познакомились еще на самом первом занятии. Чтобы создать простую картинку и вывести ее на экран, можно поступить так:

```
# загруженные изображения – это, фактически, обычный Surface
image = pygame.Surface([100, 100])
image.fill(pygame.Color("red"))
...
screen.blit(image, (10, 10))
```

Но, конечно, для нас интерес представляет именно *загрузка* готовых изображений. Это делается с помощью функции `load()` модуля `image`. Давайте напишем к ней собственную функцию-обертку, которая будет загружать изображение по имени файла, в котором оно хранится.

Пусть ее сигнатура будет следующей:

```
def load_image(name, colorkey=None)
```

Помимо имени изображения у функции будет еще один необязательный параметр — `colorkey`, который мы будем использовать чуть позже, чтобы обрезать фон изображения.

Игровые ресурсы (картинки, звуковые файлы, видеофрагменты) принято хранить в специальной папке отдельно от кода программы.

Условимся, что мы **храним все ресурсы в папке data**.

Кроме того, нужно учесть, что если файла с указанным именем не существует, нужно вывести сообщение об ошибке.

В результате фрагмент загрузки изображения будет выглядеть следующим образом:

```
import os
import sys

import pygame

# Изображение не получится загрузить
# без предварительной инициализации pygame
pygame.init()
size = width, height = 500, 500
screen = pygame.display.set_mode(size)


def load_image(name, colorkey=None):
    fullname = os.path.join('data', name)
    # если файл не существует, то выходим
    if not os.path.isfile(fullname):
        print(f"Файл с изображением '{fullname}' не найден")
        sys.exit()
    image = pygame.image.load(fullname)
    return image
```

Обычно картинки должны быть с прозрачным фоном. Чтобы человечек выглядел именно человечком, а не черным или белым квадратиком, на котором нарисован человечек.

Если при этом изображение уже прозрачно (это обычно бывает у картинок форматов png и gif), то после загрузки вызываем функцию `convert_alpha()`, и загруженное изображение сохранит прозрачность.

Если изображение было непрозрачным, то используем функцию `Surface set_colorkey(colorkey)`, и тогда переданный ей цвет станет прозрачным.

Сделаем функцию удобной: если мы передадим в качестве `colorkey` специальное значение (скажем, –1), функция сама возьмет прозрачным цветом левый верхний угол изображения (обычно это будет цвет фона, который хочется сделать прозрачным).

```
if colorkey is not None:
    image = image.convert()
```

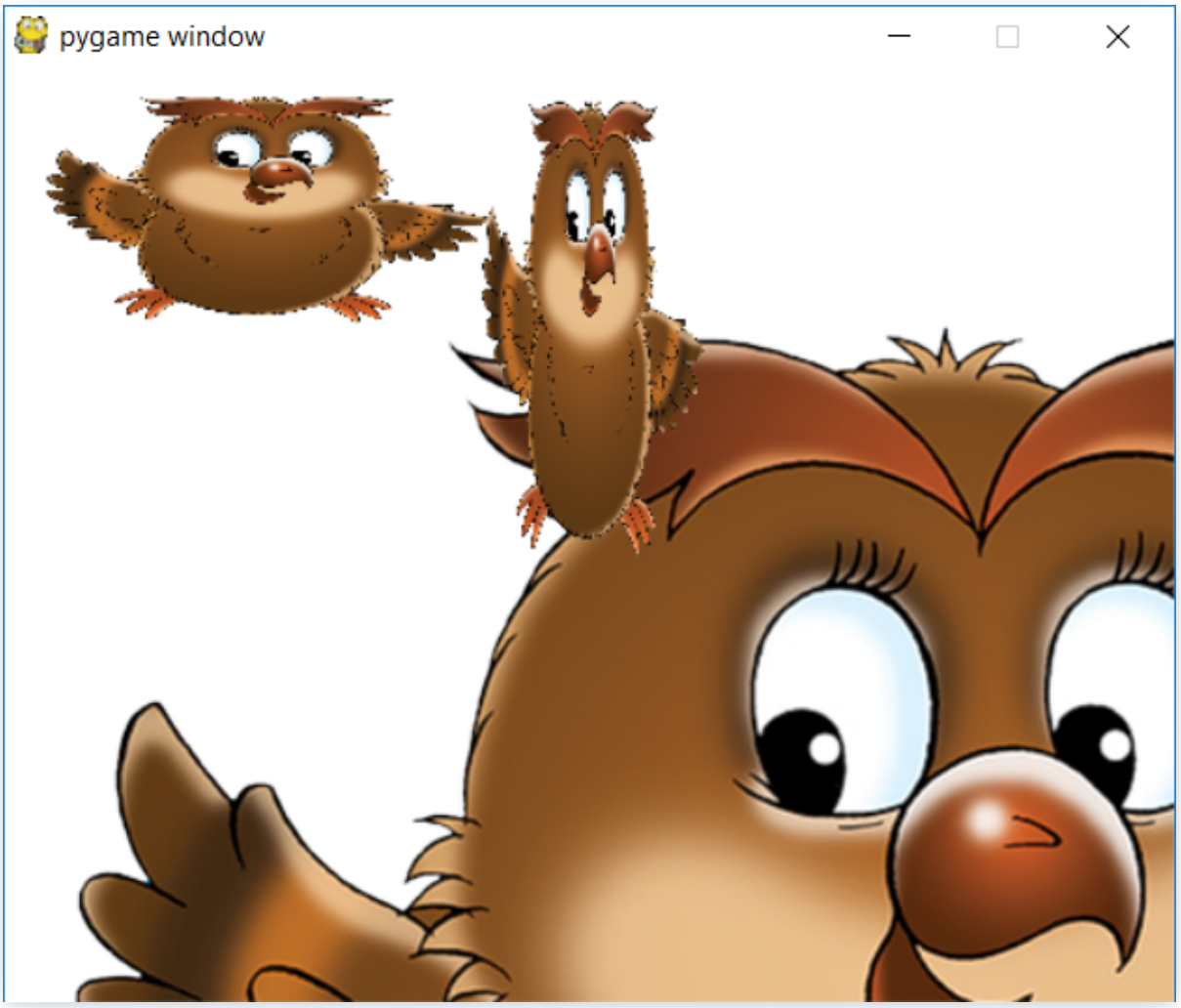
```
    if colorkey == -1:
        colorkey = image.get_at((0, 0))
    image.set_colorkey(colorkey)
else:
    image = image.convert_alpha()
return image
```

Наша функция `load_image()` возвращает `Surface`, на котором расположено изображение «в натуральную величину».

Лучше сразу подготавливать правильный размер изображения в графическом редакторе, но при необходимости можно изменить его с помощью функции `scale()` модуля `transform`:

```
image = load_image("owls.png")
image1 = pygame.transform.scale(image, (200, 100))
image2 = pygame.transform.scale(image, (100, 200))
```

Созданные изображения будут выглядеть так:



Если присмотреться, то видно, что у измененных изображений появляются артефакты. Именно поэтому предварительная подгонка изображений под нужные размеры является очень важным этапом создания игры.

На странице **документации модуля transform** есть много других функций по преобразованию изображений.

2. Спрайты

До этого момента мы работали с «клеточными» играми, в которых каждый объект занимал свою клетку на игровом поле.

Но во многих играх объекты имеют произвольные размеры (не привязаны к клеткам), и при этом важно управлять движением и взаимоотношениями нескольких объектов (например, столкновениями: выстрел попадает в танк, человечек подходит к лестнице и т.д.). В этом нам помогут **спрайты**.

Назовем **спрайтом** произвольный игровой графический объект. Этот объект может перемещаться по игровому полю или быть неподвижным, им может управлять игрок, или же его поведение контролируется непосредственно алгоритмом игры.

У спрайтов нет функции draw(). Для того чтобы работать со спрайтами, их объединяют в группы. Потом достаточно отрисовать группу, и все спрайты, принадлежащие группе, будут отрисованы (мы уже сталкивались с подобным принципом). Спрайт может и обычно принадлежит нескольким группам одновременно.

При создании спрайта нужно **не забыть** задать его вид (`image`) и размер (`rect`). Обычно для определения размеров берут прямоугольник, ограничивающий загруженное изображение.

Для работы со спрайтами в Pygame есть специальный модуль — **sprite**. По ссылке есть много интересной информации и дополнительных примеров.

В простейшем случае спрайт можно создать так:

```
# создадим группу, содержащую все спрайты
all_sprites = pygame.sprite.Group()

# создадим спрайт
sprite = pygame.sprite.Sprite()
# определим его вид
sprite.image = load_image("bomb.png")
# и размеры
sprite.rect = sprite.image.get_rect()
# добавим спрайт в группу
all_sprites.add(sprite)
```

Для размещения спрайта на холсте надо задать координаты его левого верхнего угла:

```
sprite.rect.x = 5
sprite.rect.y = 20
```

При этом размеры спрайта нигде не указываются, а определяются размерами картинки, которая спрайт создает.

Как уже говорилось, для изменения размеров самой картинки можно применять функцию `scale()` модуля `transform`, но такой способ не рекомендуется.

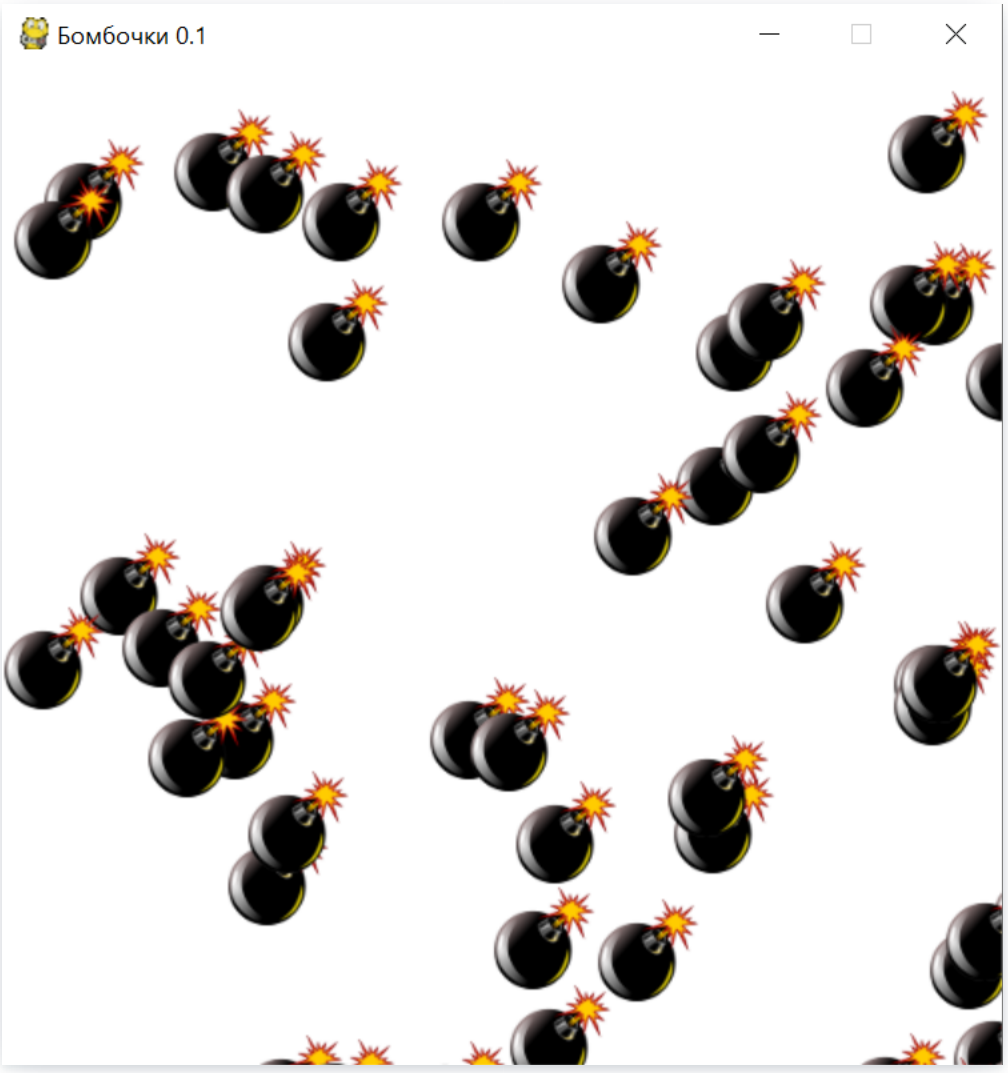
Соответственно, если мы хотим разбросать пятьдесят бомбочек по экрану, нам поможет такой код:

```
# изображение должно лежать в папке data
bomb_image = load_image("bomb.png")

for i in range(50):
    # можно сразу создавать спрайты с указанием группы
    bomb = pygame.sprite.Sprite(all_sprites)
    bomb.image = bomb_image
    bomb.rect = bomb.image.get_rect()

    # задаём случайное местоположение бомбочке
    bomb.rect.x = random.randrange(width)
    bomb.rect.y = random.randrange(height)
```

В главном игровом цикле достаточно отрисовать группу одной командой:

```
# в главном игровом цикле
all_sprites.draw(screen)
```

Попробуйте собрать все «кусочки» программы и получить следующий результат:



Можно заметить, что некоторые бомбочки не только залезают за край экрана, но и накладываются друг на друга. Пока закроем на это глаза. На следующем занятии мы подробно поговорим о пересечениях спрайтов. Это очень просто решит проблему наложения.

3. Наследование спрайтов

Код работы со спрайтами выглядит достаточно громоздким. Например, после создания спрайта его нужно «донастраивать» отдельными командами.

Работа со спрайтами станет значительно проще, если использовать «объектный» подход и унаследовать новый класс-спрайт от `pygame.sprite.Sprite`.

При наследовании важно не забыть вызвать конструктор базового класса. Иначе мы получим ошибку:

```
AttributeError: 'mysprite' instance has no attribute '_Sprite__g'
```

В производном классе можно добавить свои функции, а также переопределить функцию `update()`. Тогда при вызове этой функции для группы, произойдет вызов `update()` для каждого спрайта, который входит в нее.

Наши бомбочки в виде класса оформляются так:

```
class Bomb(pygame.sprite.Sprite):
    image = load_image("bomb.png")

    def __init__(self, *group):
        # НЕОБХОДИМО вызвать конструктор родительского класса Sprite.
        # Это очень важно!!!
        super().__init__(*group)
        self.image = Bomb.image
        self.rect = self.image.get_rect()
        self.rect.x = random.randrange(width)
        self.rect.y = random.randrange(height)

    def update(self):
        self.rect = self.rect.move(random.randrange(3) - 1,
                                   random.randrange(3) - 1)
```

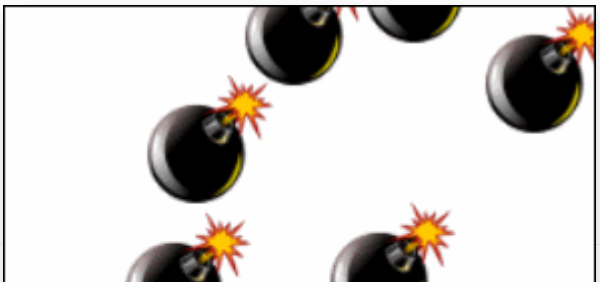
Тогда код создания упростится:

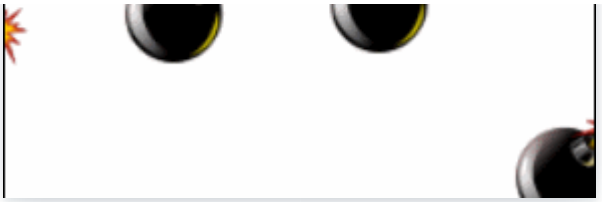
```
for _ in range(50):
    Bomb(all_sprites)
```

И при обновлении группы в главном игровом цикле

```
# в главном игровом цикле
all_sprites.draw(screen)
all_sprites.update()
```

бомбочки начинают дрожать.





Весь игровой мир можно реализовать на спрайтах.

Чтобы мир стал интерактивным, спрайтам можно добавлять свои функции, в которые передавать события из главного игрового цикла.

Но это не очень удобно: для вызова метода у всех спрайтов в группе придется перебирать все элементы в цикле (допустим, мы назвали наш метод `get_event()`):

```
if event.type == pygame.MOUSEBUTTONDOWN:
    for bomb in all_sprites:
        bomb.get_event(event)
```

Гораздо удобнее использовать для этого уже упомянутый метод `update`, который может принимать и дополнительные аргументы:

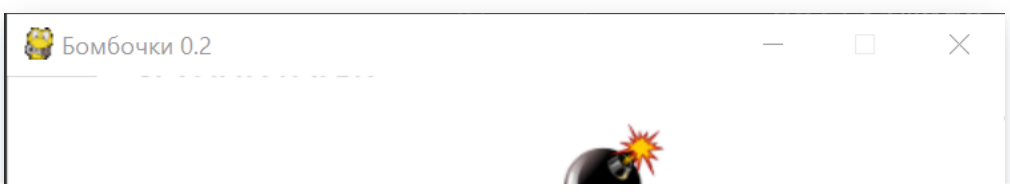
```
class Bomb(pygame.sprite.Sprite):
    image = load_image("bomb.png")
    image_boom = load_image("boom.png")

    def __init__(self, group):
        # НЕОБХОДИМО вызвать конструктор родительского класса Sprite.
        # Это очень важно !!!
        super().__init__(group)
        self.image = Bomb.image
        self.rect = self.image.get_rect()
        self.rect.x = random.randrange(width)
        self.rect.y = random.randrange(height)

    def update(self, *args):
        self.rect = self.rect.move(random.randrange(3) - 1,
                                   random.randrange(3) - 1)
        if args and args[0].type == pygame.MOUSEBUTTONDOWN and \
                self.rect.collidepoint(args[0].pos):
            self.image = self.image_boom
```

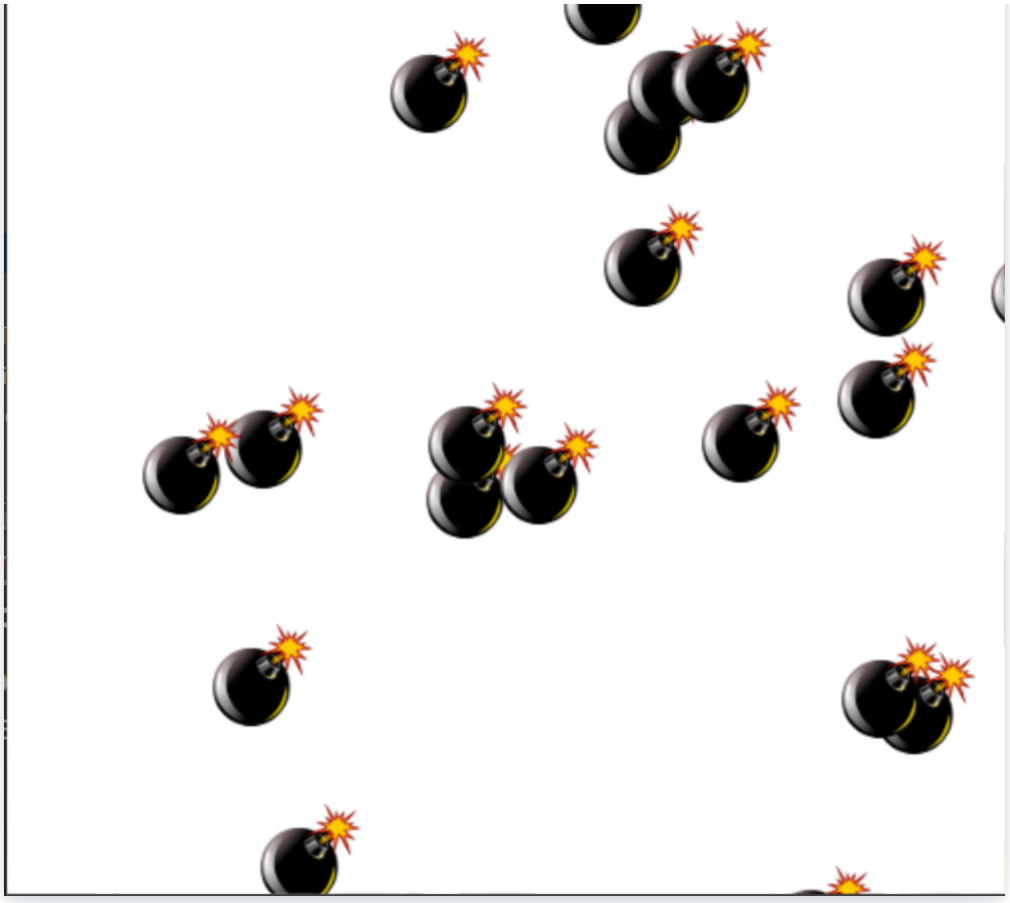
Функция `rect.collidepoint(pos)` проверяет, находится ли точка с координатами `pos` внутри прямоугольника.

Тогда код вызова проверки, стоит ли взрывать бомбочки, приобретет уже знакомый вид:

```
all_sprites.update(event)
```

Переданный `event` попадет в функции `update` всех элементов группы `all_sprites`.





Станет еще удобнее, если получится сделать класс группы спрайтов (своего рода «продвинутый» контейнер), принимающий события, и уже эта группа будет раздавать события своим элементам.

Программирование игр очень редко обходится без готовых изображений. Мы полагаем, что и в вашем проекте обязательно должны использоваться спрайты.