Проект PyGame. Изображения. Спрайты

План урока

Аннотация

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

1. Изображения

До этого момента мы с вами 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.

При этом в **Pygame** нет специального класса **Image**, а изображения представлены объектами класса **Surface**, с которым мы познакомились ещё на самом первом занятии. Чтобы создать простую картинку и вывести её 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**. Напишем функцию, которая загрузит изображение по имени файла, в котором оно хранится.

Пусть её сигнатура будет следующей:

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

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, то нужно вывести сообщение об ошибке.

В результате фрагмент загрузки изображения будет выглядеть следующим образом:

```
def load_image(name, color_key=None):
    fullname = os.path.join('data', name)
    try:
        image = pygame.image.load(fullname).convert()
    except pygame.error as message:
        print('Cannot load image:', name)
        raise SystemExit(message)
 ...
```

Обычно картинки должны быть с прозрачным фоном. Чтобы человечек выглядел именно человечком, а не чёрным 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(color_key)**, и тогда переданный ей цвет станет прозрачным.

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

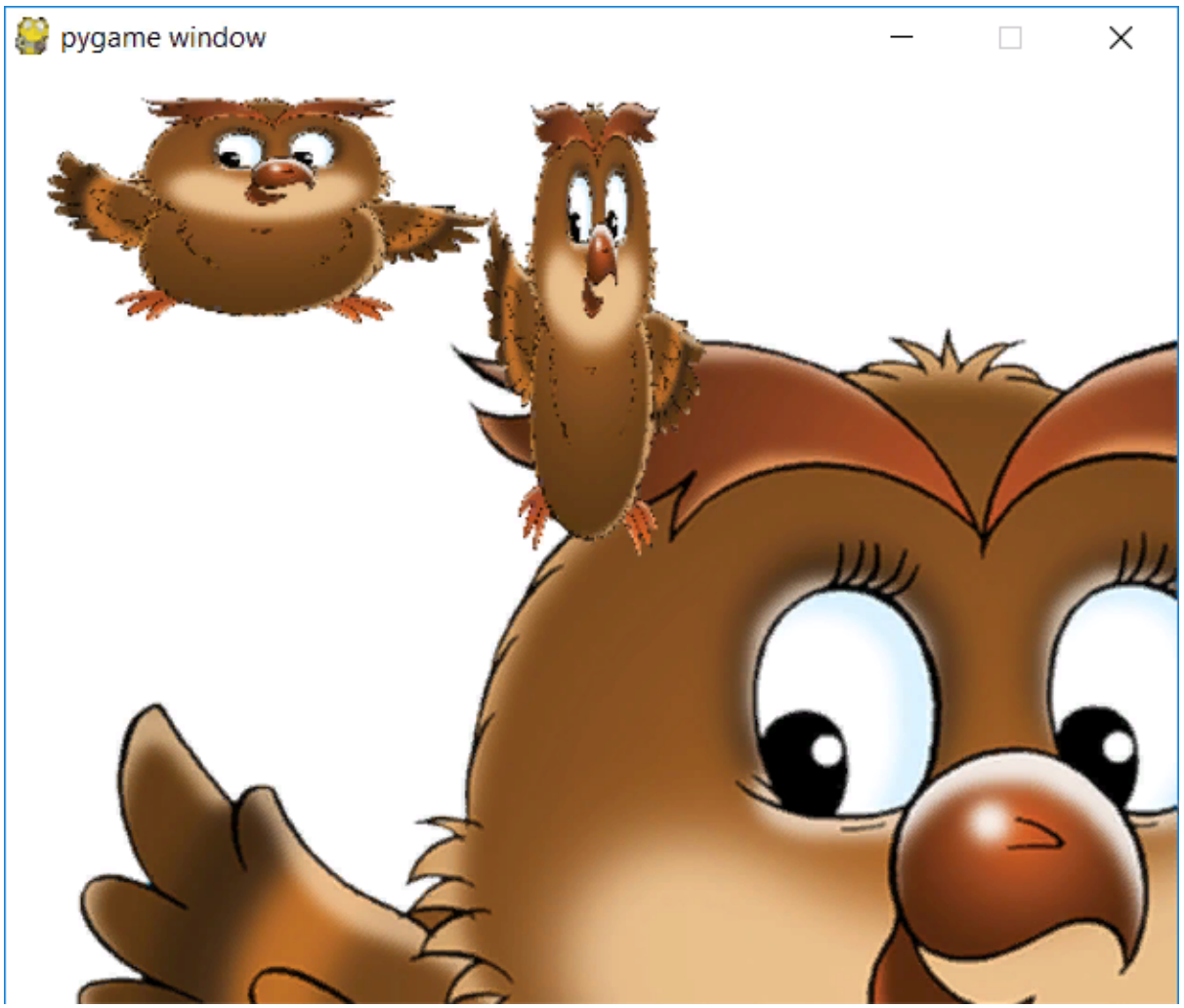
```
    if color_key is not None:
        if color_key == -1:
            color_key = image.get_at((0, 0))
        image.set_colorkey(color_key)
    else:
        image = image.convert_alpha()
    return image
```

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 изменённых изображений появляются артефакты. Именно поэтому 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 походит к лестнице и т.д.). В этом нам помогут **спрайты**.

Назовём **спрайтом** произвольный игровой графический объект, который может перемещаться по игровому полю. Этим объектом может управлять или 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 создаёт.

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

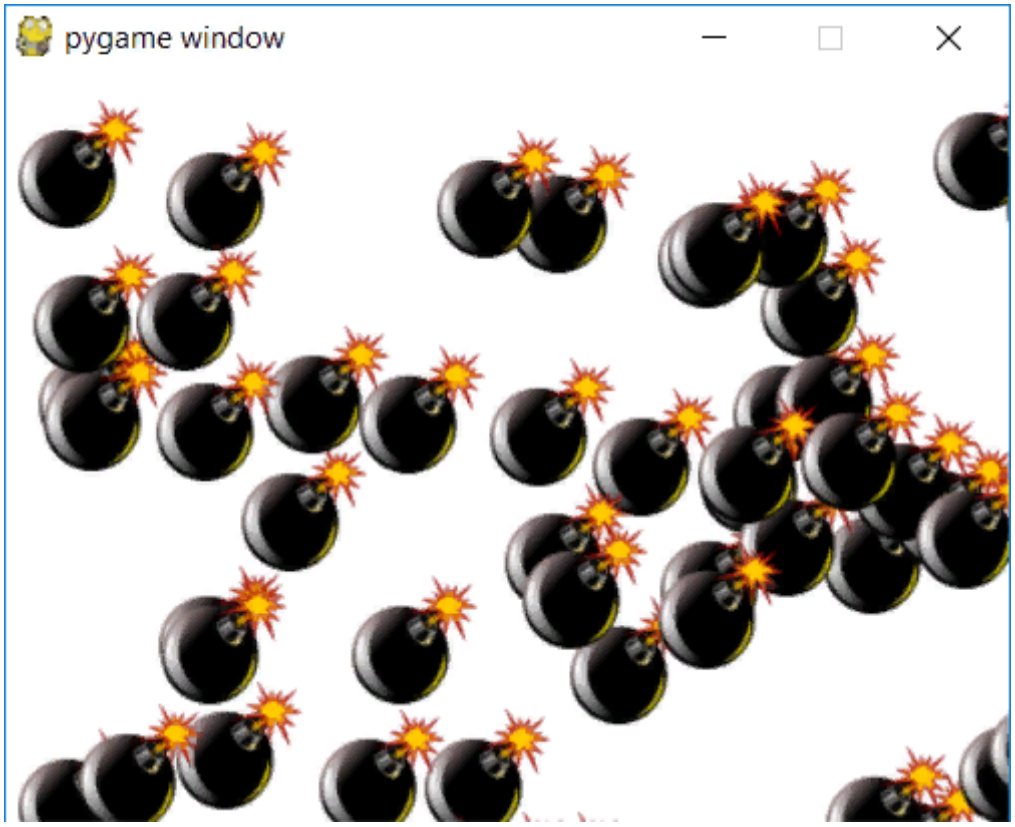
Соответственно, если мы хотим разбросать пятьдесят бомбочек по экрану, нам поможет такой код:

```
# 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.Surface.

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, произойдёт вызов **update()** для каждого спрайта, который входит в неё.

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)
```

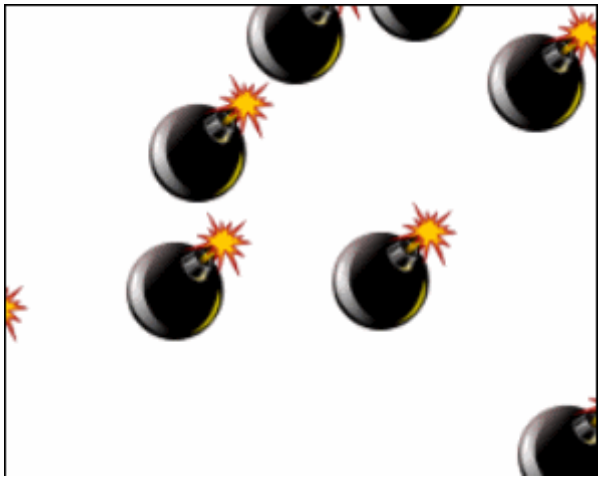
Тогда код создания упростится:

```
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.

Достаточно добавить им функцию **get_event()**, куда передавать события, полученные в главном игровом цикле — и мир станет интерактивным. Например, вот 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)

    # Поручим бомбочке получать событие и взрываться (поменяем картинку)
    def get_event(self, event):
        if self.rect.collidepoint(event.pos):
            self.image = self.image_boom
```

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

Станет ещё 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.