Урок PG. Физика

# Столкновения и другие взаимодействия



## Аннотация

*В уроке рассмотрим разные виды взаимодействия спрайтов друг с другом: пересечение по прямоугольнику, по окружности, по маске.*

## 1. Введение

Когда речь заходит об играх, то обязательно всплывает тема о физических процессах, которые лежат в основе поведения игровых объектов. Движение объектов в поле других объектов, столкновения объектов и другая функциональность — все это требует физического и математического моделирования.

Например, определение столкновений — это определение пересечений геометрических фигур.

Перед тем, как мы поговорим о возможностях модуля `sprite`, давайте решим две первые классные задачи. Их нужно решить **не пользуясь** библиотекой Pygame.

В Pygame не реализована поддержка физических процессов, но в модуле `sprite` существуют удобные функции для определения столкновений объектов.

## 2. Пересечение по прямоугольнику и окружности

Какие же возможности дает библиотека Pygame программисту?

Во-первых, Pygame позволяет проверить спрайт на столкновение с другим спрайтом. Причем проверить это можно двумя способами:

1. По ограничивающему прямоугольнику (метод `collide_rect()`)

2. По ограничивающей окружности (метод `collide_circle()`)

Для «рядовых ситуаций» этого достаточно.

А во-вторых, Pygame может проверить спрайт на столкновение с группой других спрайтов.

Попробуем написать простейший бильярд: кружочки в окне, отскакивающие от стенок. Вероятно, вы уже сталкивались с этой задачей, и, вполне возможно, решали ее. Здесь мы ударим из пушки по воробьям и решим задачу таким образом, чтобы изучить всю мощь детекторов столкновений.

Вам уже не составит труда создать класс шарика. Он может выглядеть примерно так:

```
class Ball(pygame.sprite.Sprite):
    def __init__(self, radius, x, y):
        super().__init__(all_sprites)
        self.radius = radius
        self.image = pygame.Surface((2 * radius, 2 * radius),
                                    pygame.SRCALPHA, 32)
        pygame.draw.circle(self.image, pygame.Color("red"),
                           (radius, radius), radius)
        self.rect = pygame.Rect(x, y, 2 * radius, 2 * radius)
        self.vx = random.randint(-5, 5)
        self.vy = random.randrange(-5, 5)

    def update(self):
        self.rect = self.rect.move(self.vx, self.vy)
```

Определим рамку, которая ограничивает поле экрана в виде спрайта. Тогда мы сможем определять отталкивание шариков от стенок как пересечение спрайта шарика со спрайтами рамки (поскольку рамка состоит из четырех стенок).

Стенки будут либо строго горизонтальными, либо строго вертикальными. Для нас играет роль именно принадлежность стенки к той или иной группе спрайтов: при столкновении с вертикальными нужно поменять x-координату, иначе — y-координату. Действительно, например, ударившись о нижнюю или верхнюю стенку, шарик должен изменить своё направление по вертикали, а направление по горизонтали у него останется прежним.

Заведем две дополнительные группы. Мощь модуля `sprite` библиотеки Pygame состоит в том, что можно вызовом одной команды проверять столкновения с целой группой спрайтов и даже получать список тех из них, с которыми имеется пересечение.

```
horizontal_borders = pygame.sprite.Group()
vertical_borders = pygame.sprite.Group()
```

Сам класс `Border` можно оформить так:

```
class Border(pygame.sprite.Sprite):
    # строго вертикальный или строго горизонтальный отрезок
    def __init__(self, x1, y1, x2, y2):
        super().__init__(all_sprites)
        if x1 == x2:  # вертикальная стенка
            self.add(vertical_borders)
```

```
            self.image = pygame.Surface([1, y2 - y1])
            self.rect = pygame.Rect(x1, y1, 1, y2 - y1)
        else:  # горизонтальная стенка
            self.add(horizontal_borders)
            self.image = pygame.Surface([x2 - x1, 1])
            self.rect = pygame.Rect(x1, y1, x2 - x1, 1)
```

Добавим в программу четыре стенки и несколько шариков:

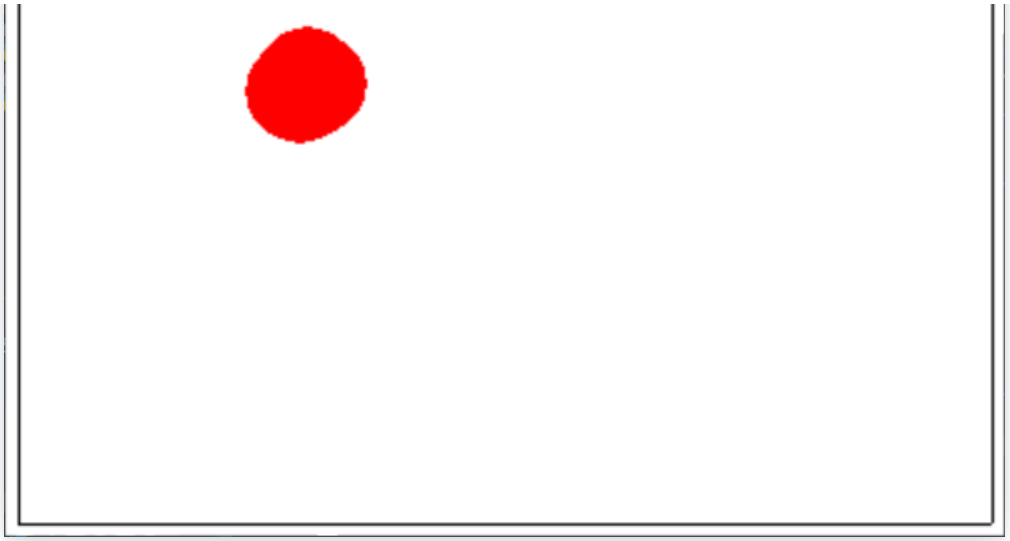
```
Border(5, 5, width - 5, 5)
Border(5, height - 5, width - 5, height - 5)
Border(5, 5, 5, height - 5)
Border(width - 5, 5, width - 5, height - 5)

for i in range(10):
    Ball(20, 100, 100)
```

Все шарики создаются в одном месте, но потом разлетаются. Естественно, не обращая внимания на стенки.


И вот тут начинается самое интересное. Добавим в `update()` шариков элементарную проверку на столкновение с группами стенок:

```
# движение с проверкой столкновение шара со стенками
def update(self):
    self.rect = self.rect.move(self.vx, self.vy)
    if pygame.sprite.spritecollideany(self, horizontal_borders):
        self.vy = -self.vy
    if pygame.sprite.spritecollideany(self, vertical_borders):
        self.vx = -self.vx
```

Теперь шарики двигаются как надо, отскакивая от стенок.



Именно так просто, в две строки. Функция `spritecollideany()` возвращает спрайт из группы, с которым произошло столкновение или None, если столкновение не обнаружено.

Другая функция, `spritecollide()`, принимает в качестве аргументов так же спрайт и группу — возвращает *список* спрайтов из группы, с которыми произошло пересечение. Третьим параметром можно передать логическое значение True, и тогда все спрайты, с которыми есть пересечение, будут уничтожены и убраны из группы.

Организовать столкновения шариков между собой немного сложнее. Вернее, организовать столкновение строго по законам физики действительно сложно. Но вполне можно пойти на упрощение: например, при столкновении менять скорости шариков на противоположные или обменивать их скорости.

Попробуйте ради практики реализовать такую задачу. Мы же будем двигаться дальше.

## 3. Пересечение по маске

Не все объекты имеют простую форму, поэтому их обработку не всегда можно свести к проверке столкновений ограничивающих объекты прямоугольников и окружностей.

Для таких сложных объектов возможно использовать функцию `pygame.sprite.collide_mask()`, которая ориентируется именно на *пересечение картинок*. Чтобы ее использовать, стоит заранее вычислить маску в проверяемых на столкновение спрайтах.

```
sprite.mask = pygame.mask.from_surface(sprite.image)
```

Или, если в конструкторе:

```
self.mask = pygame.mask.from_surface(self.image)
```

Напишем простой пример: будем высаживать десант на горы.

Горы и парашютик оформим в виде спрайтов. Картинка гор должна иметь прозрачный фон.

(Если он изначально непрозрачный, но одноцветный, то можно сделать его прозрачным, если передать в функцию `load_image()` параметр `colorkey=-1`).

В конструкторе сразу вычислим маски, чтобы потом сравнение на столкновение проходило быстрее.

Гору располагаем внизу окна, парашютики будем генерировать по щелчку мыши.

```
class Mountain(pygame.sprite.Sprite):
    image = load_image("mountains.png")

    def __init__(self):
        super().__init__(all_sprites)
        self.image = Mountain.image
        self.rect = self.image.get_rect()
        # вычисляем маску для эффективного сравнения
        self.mask = pygame.mask.from_surface(self.image)
        # располагаем горы внизу
        self.rect.bottom = height
```

Создаем гору:

```
mountain = Mountain()
```

С переменной `mountain` будет работать класс-парашютик:

```
class Landing(pygame.sprite.Sprite):
    image = load_image("pt.png")

    def __init__(self, pos):
        super().__init__(all_sprites)
        self.image = Landing.image
        self.rect = self.image.get_rect()
        # вычисляем маску для эффективного сравнения
        self.mask = pygame.mask.from_surface(self.image)
        self.rect.x = pos[0]
        self.rect.y = pos[1]

    def update(self):
        self.rect = self.rect.move(0, 1)
```

В игровом цикле добавляем парашютики:

```
    # в основном игровом цикле
    ...
    for event in pygame.event.get():
```

```
        ...
        if event.type == pygame.MOUSEBUTTONDOWN:
            Landing(event.pos)
    ...
```

Теперь парашютики летят вниз, до конца экрана:

Но если добавить в функцию `update()` класса `Landing` проверку на пересечение по маске,

```
def update(self):
    # если ещё в небе
    if not pygame.sprite.collide_mask(self, mountain):
        self.rect = self.rect.move(0, 1)
```

то ситуация изменится, и парашютики будут останавливаться в точке касания с горой.


Сравнение происходит по пересечению непрозрачных пикселей изображения (это верх горы). Видно, что один из парашютиков остановился «не долетев», потому что произошло пересечение областей не с квадратиком-грузом, а со стропами.

Справка