GeekSchool

Программирование на Python

Урок 7

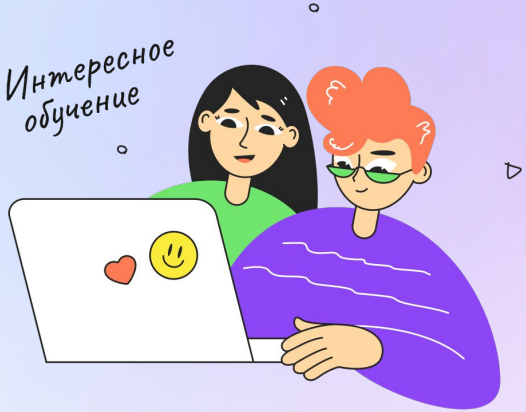
Взаимодействие с игроком Pygame

Немного повторим прошлый урок

Что будет на уроке сегодня?

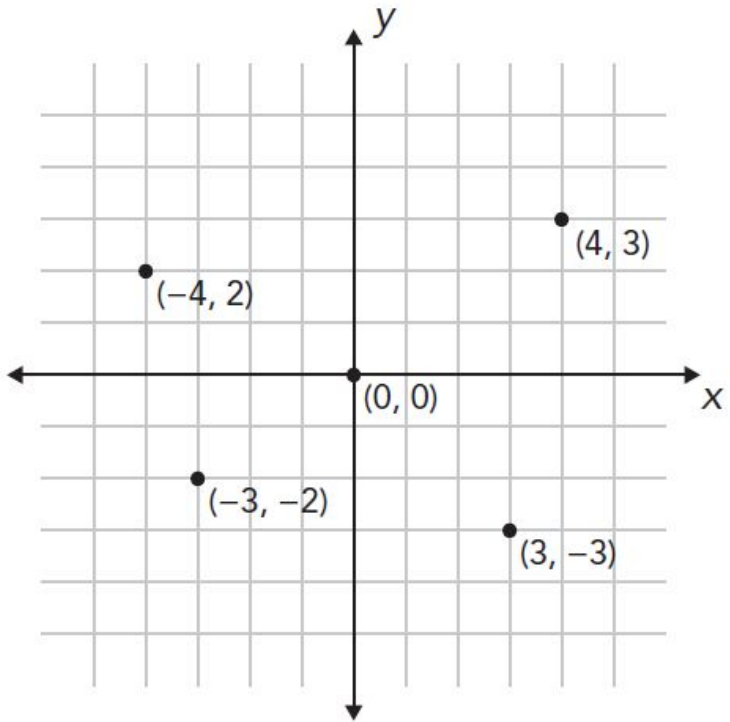
- Знакомство с координатами игрового окна
- Логика движения и анимаций
- Игровой объект, которым можно управлять
- Коллизии
- Условия взаимодействия игровых объектов

Знакомство с координатами



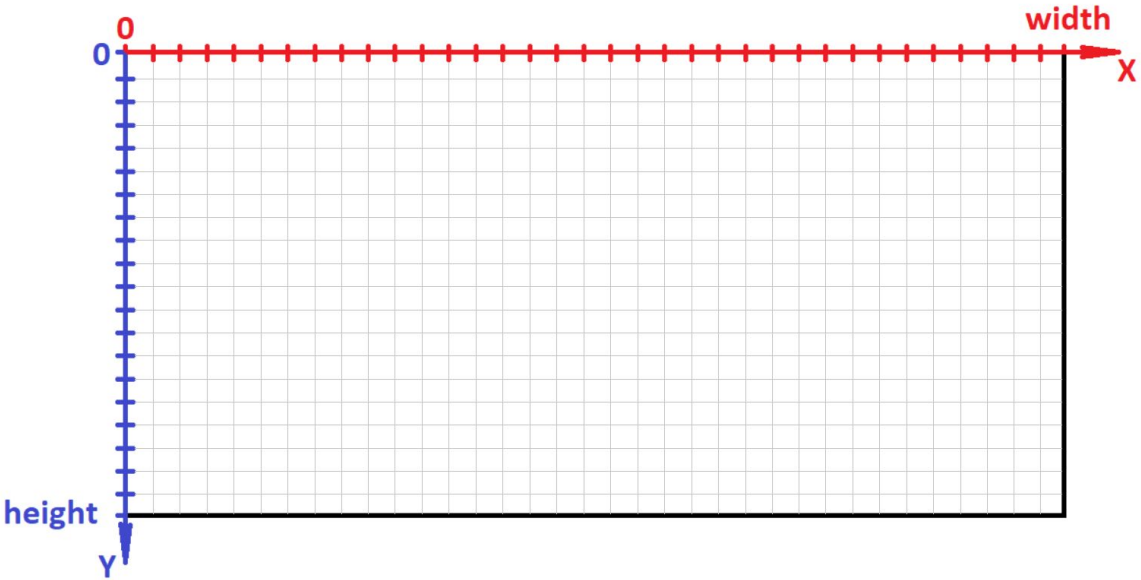
Координаты в пространстве

Для того, чтобы удобно было определять положение объектов в игре и управлять ими, существуют две оси: Горизонтальная X и вертикальная Y:



Координаты игрового окна

Однако в игре положение осей немного другое:

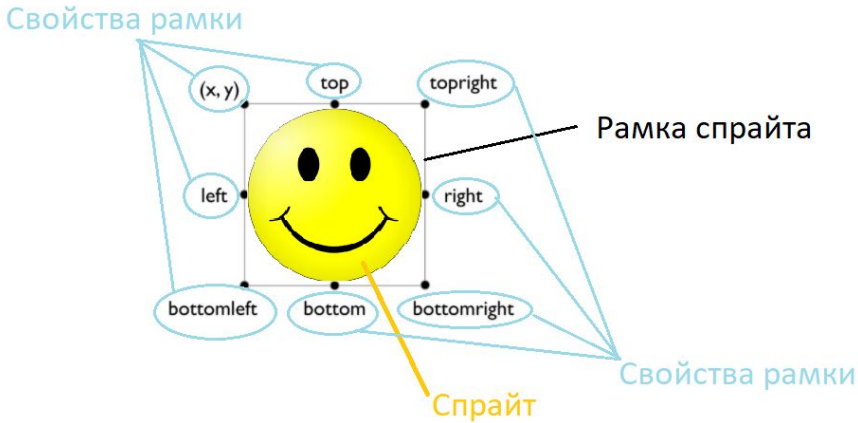


Перемещение спрайтов

Точки управления спрайтом

Взаимодействие со спрайтами происходит с помощью его рамки. Рамка спрайта обладает очень удобными и полезными свойствами, которые мы можем использовать для того, чтобы его перемещать по нашему игровому пространству:

Удобнее всего использовать точки X и Y рамки для перемещения. Хотя другие точки тоже могут быть очень полезны. Более подробно про координаты и точки X и Y мы поговорим чуть позже.



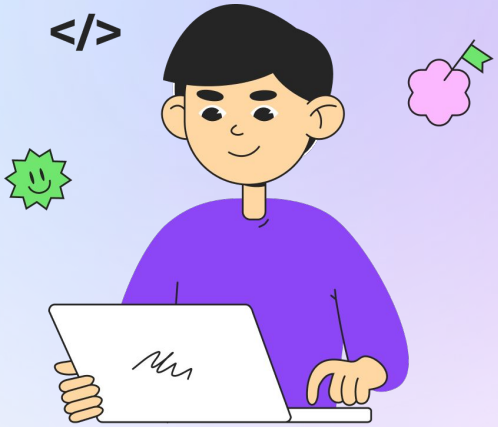
Задаем скорость перемещения

Создадим переменные, которые будут хранить скорость перемещения нашего спрайта по горизонтали и по вертикали. Сделать это нужно до игрового цикла:

```
speedx = 10                                    # Скорость движения по горизонтали
speedy = 10                                    # Скорость движения по вертикали

run = True
while run:                                     # Начинаем бесконечный цикл
```

Достижение краев экрана



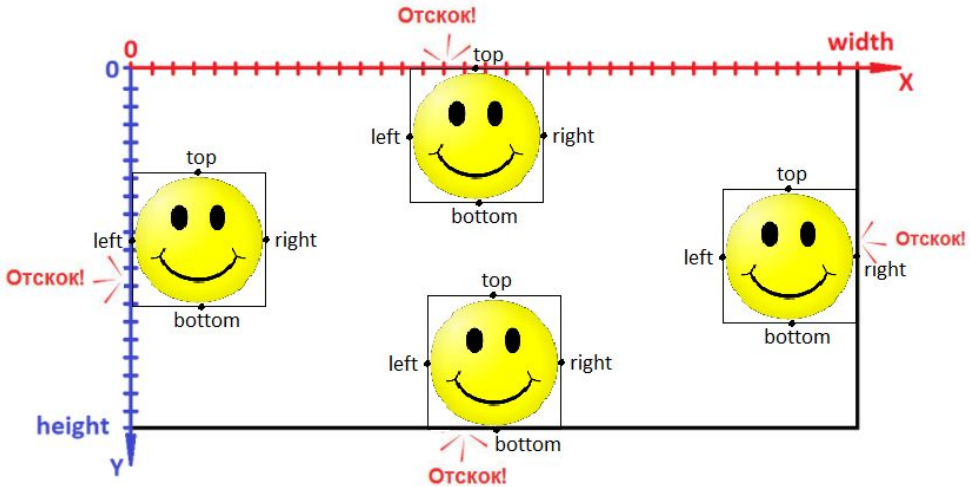
Перемещаем спрайт

Теперь, для того, чтобы положение нашего спрайта постоянно изменялось, необходимо увеличивать координаты X и Y нашей рамки на это значение:

```
    screen.blit(pic, pic_rect)          # Отрисовываем спрайт с рамкой
    pic_rect.x += speedx                # Увеличиваем координату X спрайта
    pic_rect.y += speedy                # Увеличиваем координату Y спрайта
    pygame.display.update()             # Обновляем экран
```

Отскок от краев экрана

Сейчас наш спрайт пропадает за границей экрана. Это не очень хорошо, нам хотелось бы, чтобы он вел себя более естественно. Для этого нам важно понять что происходит в момент, когда спрайт достигает краев экрана. Давайте для большего удобства посмотрим на рисунок:



Условия для отскоков:

Мы помним, что сделали ширину нашего экрана равной тому значению, что записано в переменной width.

1. Если наш спрайт двигается **вправо** (его координата X **увеличивается**) значит, как только точка **right** достигнет **правой границы** экрана (это значение **width**), то это и будет наше условие для отскока.

1. Если наш спрайт двигается **влево** (его координата X **уменьшается**) и как только точка **left** достигнет **левой границы** экрана (это число **0**), то мы снова должны сделать отскок.

1. Если наш спрайт двигается **вниз** (его координата Y **увеличивается**) и как только точка **bottom** достигнет **нижней границы** экрана (это значение **height**), то мы снова должны сделать отскок.

1. Если наш спрайт двигается **вверх** (его координата Y **уменьшается**) и как только точка **top** достигнет **верхней границы** экрана (это число **0**), то мы снова должны сделать отскок.

Сам отскок:

Чтобы сделать отскок, нам всего лишь нужно изменить скорость движения спрайта на противоположную.
Для этого мы просто должны в переменную скорости присвоить то же самое значение, но со знаком минус:

```
    speedx = - speedx                    # Меняем скорость по горизонтали
    speedy = - speedy                    # Меняем скорость по вертикали
```

Но делать это нужно только при условии, что спрайт достиг границ экрана!
Причем важно менять правильную скорость при достижении определенного края.

- **Для верхней и нижней границы** — это будет **вертикальная** скорость.
- **Для левой и правой границы** — это будет **горизонтальная** скорость.

Программируем отскоки:

Отскоки настраиваем уже в игровом цикле. Так как важно, чтобы игра постоянно проверяла, не зашел ли спрайт за границы экрана:

```
if pic_rect.bottom > height:    # Если достигли нижней границы экрана
    speedy = -speedy
if pic_rect.top < 0:            # Если достигли верхней границы экрана
    speedy = -speedy
if pic_rect.left < 0:           # Если достигли левой границы экрана
    speedx = -speedx
if pic_rect.right > width:      # Если достигли правой границы экрана
    speedx = -speedx
```

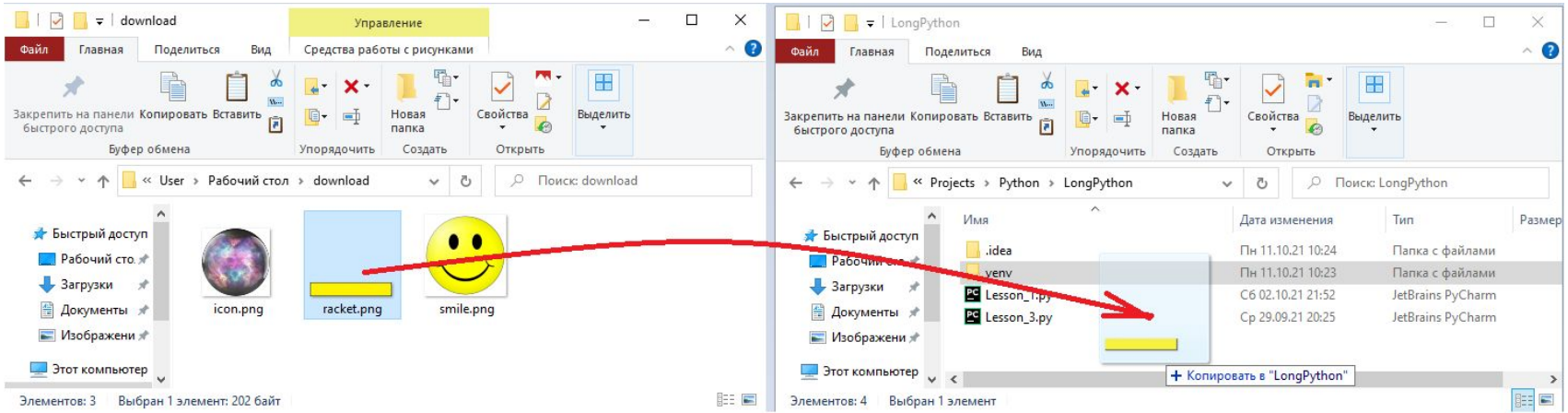
Перерыв

10 мин

Второй спрайт ракетки

Загружаем спрайт в проект и добавляем в код

Мы уже добавляли спрайты в нашу игру. Спрайт игрока добавляется точно таким же образом. Копируем картинку в ту же папку, что и файл с кодом.



Загружаем спрайт в проект и добавляем в код

Затем **до игрового цикла** создаем еще одну переменную, в которую загружаем спрайт, а также определяем рамку:

```
racket = pygame.image.load('racket.png')        # Загружаем спрайт игрока
racket_rect = racket.get_rect()                 # Получаем рамку спрайта игрока
```

И не забудем добавить отрисовку спрайта игрока **внутри игрового цикла:**

```
screen.fill(CYAN)                       # Заливка заднего фона
screen.blit(pic, pic_rect)              # Отрисовываем смайлик
screen.blit(racket, racket_rect)        # Отрисовываем ракетку
```

Правильное стартовое положение ракетки

Как видите, ракетка появляется немного не в том месте, где она обычно бывает в играх — арканоидах. А чтобы поместить в нужную позицию, необходимо изменить координаты рамки спрайта!

Так как стартовая **позиция спрайта должна устанавливаться всего один раз,** то удобно это сделать сразу как только загрузили спрайт и создали его рамку:

? Чтобы установить положение по горизонтали, мы должны поставить центр рамки спрайта в центре по-горизонтали:

```
racket_rect.x = width / 2
```

? Но, так как сама точка X рамки спрайта находится не по-середине, а в левом углу, то нам необходимо еще добавить вычитание ширины спрайта, разделенное на 2. Выглядит это следующим образом:

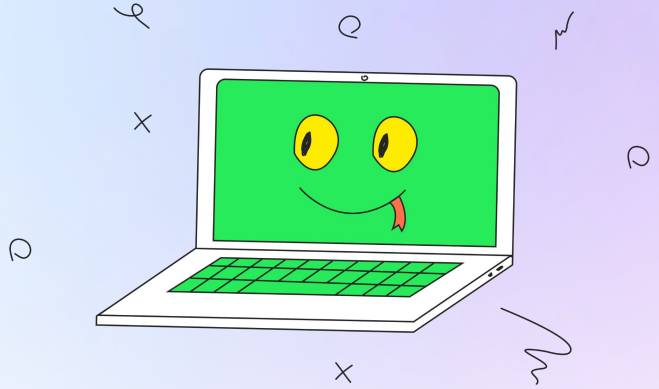
```
racket_rect.x = width / 2 - racket.get_width()/2
```

Правильное стартовое положение ракетки

? С позицией по вертикали всё проще — просто берем нижнюю и поднимаемся чуть выше:

```
racket_rect.x = width / 2 - racket.get_width()/2
racket_rect.y = height - 50
```

Управление спрайтом



События нажатия кнопок клавиатуры

Мы уже с вами изучали как следить за событием выхода из игры. Но в случае с действиями, которые должны выполняться постоянно, необходимо использовать другой подход. Перейдем внутрь игрового цикла и создадим переменную в которую будут попадать все нажатия на клавиши клавиатуры:

```
while run:                              # Начинаем бесконечный цикл
    timer.tick(fps)                     # Контроль времени (обновление игры)
    for event in pygame.event.get():    # Обработка ввода (события)
        if event.type == pygame.QUIT:   # Проверить закрытие окна
            run = False                 # Завершаем игровой цикл
    key = pygame.key.get_pressed()      # Считываем нажатия на клавиши
```

Реагируем на клавиши

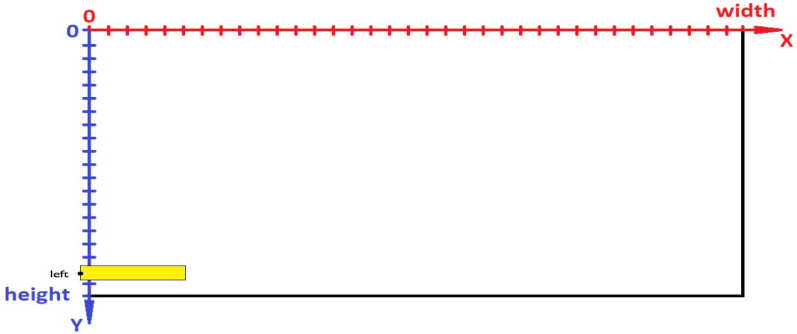
Теперь, какую кнопку мы бы не нажали, в переменной key у нас будет храниться информация о ней. Осталось только проверить а какая именно кнопка была нажата. И если нажата клавиша стрелка влево, то просто изменить координату X ракетки нужным образом

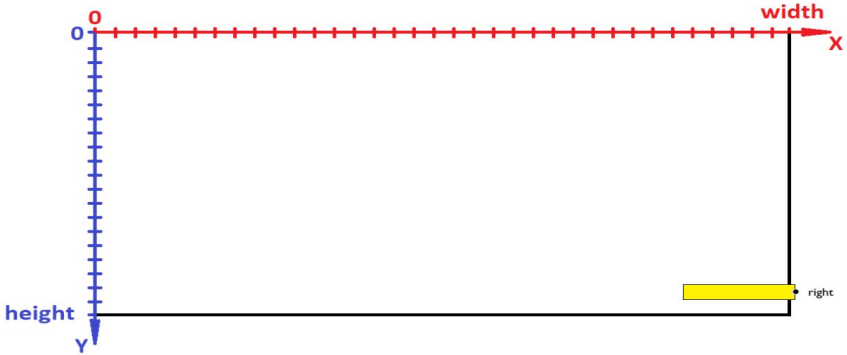
```
if key[pygame.K_LEFT]:    # Движение влево
    racket_rect.x -=10
if key[pygame.K_RIGHT]:  # Движение вправо
    racket_rect.x +=10
```

Ограничиваем движение

Теперь наша ракетка имеет ту же проблему, что и смайлик в самом начале.
Она выходит за края экрана, если мы слишком близко приблизимся к нему!

Ограничить такое поведение можно очень просто, если мы будем двигать ракетку не только проверяя какая клавиша нажата, но еще и проверяя, не зашла ли ракетка за пределы экрана! У нас будет две ситуации, которые нам надо предусмотреть:





Ограничиваем движение

Мы с вами уже делали что-то подобное, когда реализовывали отскоки смайлика от краев экрана. Здесь же мы просто добавим еще одно условие к проверке нажатия клавиш. То есть ракетка будет продолжать движение, **если** кнопка нажата **И** точка ракетки не вылезла за край экрана.

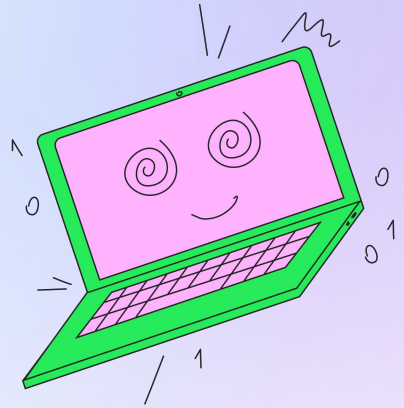
? Для левого края это будет так:

```
if key[pygame.K_LEFT] and racket_rect.left > 0:    # Движение влево
```

? Для правого края это будет так:

```
if key[pygame.K_RIGHT] and racket_rect.right < width:  # Движение вправо
```

Столкновения спрайтов



Коллизии

Столкновение — вещь неизбежная и постоянно случается в играх. А на каждое столкновение есть своя реакция. Как минимум, предметы должны отскакивать друг от друга с разной силой. В играх столкновение называется **коллизиями**.

Коллизии в pygame определяются столкновением не изображением спрайтов, а их рамок. У каждой рамки спрайта есть функция `colliderect()` в скобках которой мы должны указать рамку второго спрайта с которым хотим настроить столкновения. В нашем случае это будет так:

```
if pic_rect.colliderect(racket_rect): # Столкновение рамок смайлика и ракетки
    speedy = -speedy                  # Изменяем скорость движения смайлика
```

Полный код программы

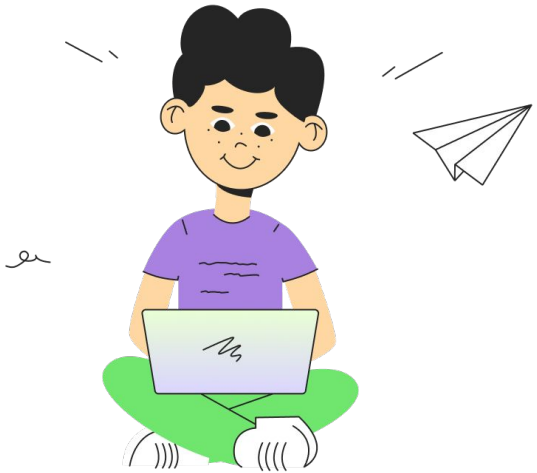
Сюда не влазит, поэтому посмотреть можно по **ссылке** :)

Итоги

- Познакомились с принципами движения спрайтов
- Научились составлять условия для ограничения движений
- Добавили второй спрайт с ракеткой
- Настроили управление для ракетки
- Узнали про коллизии и реализовали их в своей игре

На следующем занятии:

- Добавим события проигрыша
- Снабдим мячик жизнями и выведем их на экран
- Реализуем логику потери жизней
- Добавим звуки и музыку
- Добавим задний фон и анимируем его



Немного повторим

Когда спрайт движется вправо, какая координата изменяется и каким образом?

**Если спрайт заходит за нижнюю границу экрана,
за какой точкой на рамке мы должны следить?**

Что нужно сделать, чтобы спрайт появлялся сразу в нужной части экрана?

Что такое коллизии?

Ваши вопросы

Спасибо
за внимание

Домашнее задание

Заполни, пожалуйста,
форму обратной связи по уроку

Напоминание для преподавателя

- **Проверить заполнение Журнала**
- **Заполнить форму Т22**