Материалы к занятию

Модуль arcade позволяет нам рисовать различные фигуры. Фигуры можно разделить на закрашенные цветом, так и на имеющие только обводку. Будем рассматривать их по очереди. Для начала нарисуем дугу, закрашенную каким, либо цветом. Для отрисовки данной дуги воспользуемся функцией draw_arc_filled()

Синтаксис:

arcade.draw_arc_filled(center_x: float, center_y: float, width: float, height: float, color: Union[Tuple[int, int, int], List[int], Tuple[int, int, int, int]], start_angle: float, end_angle: float, tilt_angle: float = 0, num_segments: int = 128)

- center_x – центр фигуры по x
- center_y –центр фигуры по y
- width – ширина фигуры
- height – высота фигуры
- color – цвет фигуры
- start_angle – начальный угол фигуры в градусах
- end_angle – конечный угол фигуры в градусах
- tilt_angle – угол наклона дуги
- num_segments – количество сегментов дуги

```
import arcade


class Game(arcade.Window):
  def __init__(self):
    super().__init__(width=800, height=600, title='Рисование фигур')
    self.background_color = (255, 255, 255)

  def on_draw(self):
    self.clear()
    arcade.draw_arc_filled(
      center_x=400,
      center_y=300,
      width=155,
      height=155,
      color=(0, 0, 0),
      start_angle=0,
      end_angle=90,
      tilt_angle=15,
      num_segments=15
    )


if __name__ == '__main__':
  game = Game()
  arcade.run()
```

Кстати, цвет дуги, может быть не только в формате RGB, но и в RGBA(A - прозрачность). Например, чтобы фигура была полупрозрачной мы можем указать параметр color=(0, 0, 0, 125)

Для отрисовки дуги без заливки воспользуемся функцией draw_arc_outline(). В данной дуге присутствует дополнительный параметр border_width - ширина обводки

```
arcade.draw_arc_outline(
  center_x=400,
  center_y=300,
  width=155,
  height=155,
  color=(0, 0, 0, 125),
  start_angle=0,
  end_angle=90,
  tilt_angle=15,
  num_segments=15,
  border_width=2
)
```

Сделаем небольшой вывод: для фигур залитых цветом используется функции с filled в названии, для обводки - outline.

Теперь давайте отрисуем круг (draw_circle_filled)

```
arcade.draw_circle_filled(
  center_x=400,
  center_y=300,
  radius=100,
  color=(0, 0, 0),
  tilt_angle=0,
  num_segments=-1
)
```

Из новых параметров у нас только радиус круга. Теперь круг с обводкой

```
arcade.draw_circle_outline(
  center_x=400,
  center_y=300,
  radius=100,
  color=(0, 0, 0),
  tilt_angle=0,
  num_segments=-1,
  border_width=2
)
```

Помимо круга, можно также отрисовать например эллипс

```
arcade.draw_ellipse_filled(
  center_x=400,
  center_y=300,
```

```
  color=(0, 0, 0),
  tilt_angle=0,
  num_segments=-1,
  width=100,
  height=150
)
```

Важно: в данных примерах указаны все возможные параметры для данных типов фигур

Теперь эллипс с обводкой

```
arcade.draw_ellipse_outline(
  center_x=400,
  center_y=300,
  color=(0, 0, 0),
  tilt_angle=0,
  num_segments=-1,
  width=100,
  height=150, border_width=2
)
```

Перейдем к линии

```
arcade.draw_line(
  start_x=300,
  start_y=300,
  end_x=500,
  end_y=300,
  color=(0, 0, 0),
  line_width=2
)
```

По параметрам, тут, тоже все достаточно понятно: начало по x, начало по y, конец по x, конец по y, цвет, ширина линии.

Давайте немного отвлечемся и попробуем нарисовать облако

```
import arcade


def create_cloud(x, y):
  arcade.draw_circle_filled(
     center_x=x,
     center_y=y,
     color=(0, 0, 255),
     radius=60
  )
```

```
class Game(arcade.Window):
  def __init__(self):
    super().__init__(width=800, height=600, title='Рисование фигур')
    self.background_color = (255, 255, 255)

  def on_draw(self):
    self.clear()
    for i in range(1, 4):
      create_cloud(100 * i, 400)
    for j in range(1, 3):
      create_cloud(100 * j + 50, 470)


if __name__ == '__main__':
  game = Game()
  arcade.run()
```

Данный вариант нельзя назвать оптимизированным, но для примера нам подойдет. Немного порисовали, теперь перейдем дальше. Нарисуем ломанную линию

```
arcade.draw_line_strip(
  point_list=[(100, 200), (150, 250), (200, 200)],
  color=(0, 0, 0),
  line_width=2
)
```

Из параметром можно выделить point_list - это список кортежей, координат точек по x и y.

Перейдем к прямоугольникам. Для начала разберем функцию, которая принимает координаты углов фигуры(левый угол, правый, вверх и низ)

```
arcade.draw_lrtb_rectangle_filled(
  left=100,
  right=300,
  top=300,
  bottom=100,
  color=(0, 0, 0)
)
```

Аналогично с прямоугольником без заливки

```
arcade.draw_lrtb_rectangle_outline(
  left=100,
  right=300,
  top=300,
  bottom=100,
  color=(0, 0, 0),
  border_width=2
)
```

```

```

Кстати, при желании мы можем отрисовать также и точку

```
arcade.draw_point(
  x=100,
  y=200,
  color=(0, 0, 0),
  size=10
)
```

Ну и перейдем к прямоугольнику, в качестве параметром координат, которому мы передаем центр по x и y

```
arcade.draw_rectangle_filled(
  center_x=300,
  center_y=300,
  width=200,
  height=100,
  color=(0, 0, 0),
  tilt_angle=45
)
```

и без заливки

```
arcade.draw_rectangle_outline(
  center_x=300,
  center_y=300,
  width=200,
  height=100,
  color=(0, 0, 0),
  tilt_angle=45,
  border_width=2
)
```

Ну и напоследок отрисуем треугольник

```
arcade.draw_triangle_filled(
  x1=100,
  y1=100,
  x2=200,
  y2=200,
  x3=300,
  y3=100,
  color=(0, 0, 0)
)
```

В данном случае, мы указываем координаты точек вершин треугольника.

Треугольник без заливки

```
arcade.draw_triangle_outline(
  x1=100,
  y1=100,
  x2=200,
  y2=200,
  x3=300,
  y3=100,
  color=(0, 0, 0),
  border_width=2
)
```

Учитель: На самом деле, мы конечно же можем отрисовывать не просто прямоугольники, а фигуры залитые текстурой. Сейчас мы рассмотрим просто небольшой пример, а более подробнее мы поработаем в дальнейшем.

```
import arcade


class Game(arcade.Window):
  def __init__(self):
    super().__init__(width=800, height=600, title='Рисование фигур')
    self.background_color = (255, 255, 255)
    self.texture = arcade.load_texture(':resources:images/animated_characters/zombie/zombie_jump.png')

  def on_draw(self):
    self.clear()
    arcade.draw_texture_rectangle(
      center_x=100,
      center_y=200,
      width=60,
      height=90,
      texture=self.texture
    )


if __name__ == '__main__':
  game = Game()
  arcade.run()
```

В данном случае в конструкторе класса, мы подгружаем текстуру, а потом отрисовываем ее с помощью соответствующего метода. Текстура берем из встроенных ресурсов arcade.

Для начала создадим класс для минимального приложения с размерами 800 на 600 и заголовком ‘Геометрические фигуры и их анимация’

```
import arcade


class Game(arcade.Window):
  def __init__(self):
    super().__init__(
      width=800, height=600, title="Геометрические фигуры и их анимация"
    )
    self.background_color = (155, 255, 255)

  def on_draw(self):
    self.clear()


if __name__ == "__main__":
  game = Game()
  arcade.run()
```

Отлично.

Мы можем поместить несколько примитивов в список элементов формы, а затем переместить и нарисовать их как один.Данный подход используется, когда мы хотим создать более сложный объект из более простых примитивов. Это также ускоряет рендеринг, поскольку все объекты отрисовываются за одну операцию.

Для этого мы создадим в конструкторе класса список элементов.

```
self.batch = arcade.ShapeElementList()
```

Далее создадим метод setup и там создадим пока, что первый примитив - эллипс

```
def setup(self):
  ellipse1 = arcade.create_ellipse(center_x=100, center_y=100, width=50, height=60,
color=arcade.color.BLUE)

  self.batch.append(ellipse1)
```

В коде выше мы создаем эллипс и добавляем его в наш список элементов.Далее в методе on_draw(), нам необходимо отрисовать наш список

```
def on_draw(self):
  self.clear()
  self.batch.draw()
```

Также эллипс можно сделать незакрашенным

```
ellipse1 = arcade.create_ellipse(
  center_x=300, center_y=300, width=50, height=60, color=arcade.color.BLUE, filled=False
)
```

Давайте теперь отрисуем эллипс, который имеет несколько цветов

```
ellipse2 = arcade.create_ellipse_filled_with_colors(
  center_x=300,
  center_y=300,
  width=200,
  height=250,
  outside_color=arcade.color.RED,
  inside_color=arcade.color.GREEN,
  tilt_angle=45,
)
```

В данном примере мы указываем не один цвет, а два(один внутренний, другой внешний).

Для того чтобы фигура отрисовалась не забываем ее добавить в наш список

```
self.batch.append(ellipse2)
```

Обратите внимание, если отрисовать сначала первый эллипс, а потом второй, то второй перекроет его и мы его не увидим. Таким образом мы можем составлять более сложные фигуры и отрисовывать их, как одно целое. Увы все типы фигур мы рассмотреть не сможем, но их всегда можно найти в официальной документации.

Давайте пока закомментируем все что отрисовали и попробуем, что нибудь отрисовать и попробовать сделать анимацию для наших фигур.

Для начала, для удобства введем переменные со значениями ширины и высоты окна

```
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
```

Изменим конструктор класса игрового окна

```
def __init__(self):
  super().__init__(
     width=SCREEN_WIDTH, height=SCREEN_HEIGHT, title="Геометрические фигуры и их анимация"
  )
```

Теперь создадим отдельный класс для нашей фигуры

```
class Rectangle:
```

```
def __init__(self, x, y, w, h, color, tlit=0, filled=False):
    self.x = x
    self.y = y
    self.w = w
    self.h = h
    self.color = color
    self.tlit = tlit
    self.filled = filled
```

В данном классе мы указали все необходимые параметры для отрисовки данной фигуры, а также не забыли добавить параметр, со значением по умолчанию для отрисовки закрашенного или нет прямоугольника.

Создадим метод draw() для нашего прямоугольника

```
def draw(self):
  if not self.filled:
     arcade.draw_rectangle_outline(self.x, self.y, self.w, self.h, self.color)
  else:
     arcade.draw_rectangle_filled(self.x, self.y, self.w, self.h, self.color)
```

Не забудем создать экземпляр класса в методе __init__ основного класса

```
self.rectangle1 = Rectangle(20, 100, 40, 50, arcade.color.BLUE)
```

И наконец отрисуем его

```
def on_draw(self):
  self.clear()
  self.rectangle1.draw()
```

При запуске мы видим что в левой части экрана отрисовался не залитый прямоугольник. Добавим ему движение. Для этого создадим в классе прямоугольника метод update(), в котором укажем, что его координата x будет увеличиваться на 3 если он не достиг конца экрана

```
def update(self):
  if self.x < SCREEN_WIDTH:
     self.x += 3
```

Для “активации” нашего метода, мы также создадим метод update() в основном классе

```
def update(self, delta_time: float):
  self.rectangle1.update()
```

При запуске мы видим, что наш прямоугольник действительно двигается, но он останавливается чуть дальше экрана. Все потому, что в условии мы используем центр по координате x для нашей фигуры. Ширину фигуры мы знаем, значит от центра до края фигуры можно посчитать, как ширина разделенная на 2. Изменим условие, а также сделаем, чтобы прямоугольник двигался потом в обратную сторону.Создадим переменную отвечающую за значение смещения нашего объекта в __init__ прямоугольника

```
self.change_x = 3
```

А также изменим условие в update

```
def update(self):
  self.x += self.change_x
  if self.x + self.w / 2 > SCREEN_WIDTH or self.x - self.w / 2 < 0:
     self.change_x = -self.change_x
```

Как видим теперь прямоугольник, как только сталкивается с краями экрана, то меняет свое направление. Создадим новый объект и сделаем, чтобы он, например менял свой размер при столкновении со стеной. Прямоугольник сделаем залитым цветом

```
self.rectangle2 = Rectangle(SCREEN_WIDTH - 40, 200, 40, 50, arcade.color.YELLOW, filled=True)
```

Создадим метод для изменения размера в прямоугольнике

```
def change_size(self):
  if self.x + self.w / 2 >= SCREEN_WIDTH or self.x - self.w / 2 <= 0:
     self.w += 2
     self.h += 2
```

Далее добавим необходимые строки в on_draw() и update()

```
def on_draw(self):
  self.clear()
  self.rectangle1.draw()
  self.rectangle2.draw()

def update(self, delta_time: float):
  self.rectangle1.update()
```

```
  self.rectangle2.update()
  self.rectangle2.change_size()
```

Отлично. Давайте добавим для них возможность двигаться по координате y

```
def update(self):
  self.y += self.change_y
  self.x += self.change_x
  if self.x + self.w / 2 > SCREEN_WIDTH or self.x - self.w / 2 < 0:
     self.change_x = -self.change_x
  if self.y + self.h / 2 > SCREEN_HEIGHT or self.y - self.h / 2 < 0:
     self.change_y = -self.change_y
```

Все работает. Но наш второй прямоугольник увеличивается, только в том случае если он сталкивается с левой или правой стороной окна. Исправим

```
def change_size(self):
  if (
     self.x + self.w / 2 >= SCREEN_WIDTH
     or self.x - self.w / 2 <= 0
     or self.y + self.h / 2 >= SCREEN_HEIGHT
     or self.y - self.h / 2 <= 0
  ):
     self.w += 2
     self.h += 2
```