Материалы к занятию

Сегодня мы с вами начнем изучать первую библиотеку для создания игр - Arcade.
В Python существуют различные библиотеки для создания 2D игр, arcade одна из самых распространенных, наравне с Pygame.

Pygame: это модуль Python, используемый для разработки видеоигр, позволяющий использовать компьютерную графику и звуковые библиотеки для разработки высококачественных и пользовательских интерактивных игр. Pygame был разработан Питом Шиннерсом. Pygame является переносимым, и его код совместим со всеми операционными системами. С его помощью также можно создавать бесплатные, условно-бесплатные и коммерческие игры с открытым исходным кодом. Код Pygame написан на языке C, а модуль поставляется для Windows, Linux и macOS. Его также можно легко использовать на портативных устройствах.

Arcade: это модуль Python, но работает только для Python 3.6 и выше. Он пытается охватить большинство функций, которые не поддерживались Pygame. Это также использует компьютерную графику и звуковые библиотеки для разработки качественных и пользовательских интерактивных игр. Arcade был разработан Полом Винсентом Крэйвеном. Аркаде нужна поддержка OpenGL 3.3+. Он построен на основе OpenGL и Pyglet и совместим с Windows, Linux и macOS X. С его помощью также можно создавать бесплатные, условно-бесплатные и коммерческие игры с открытым исходным кодом.

В рамках курса мы познакомимся с обеими библиотеками, но начнем с Arcade.
Для начала, нам необходимо ее установить
pip install arcade

Создадим базовое графическое окно нашего приложения

```
import arcade

SCREEN_WIDTH = 1000
SCREEN_HEIGHT = 650
SCREEN_TITLE = "Platformer"


class MyGame(arcade.Window):
  def __init__(self):
    super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)

    arcade.set_background_color(arcade.csscolor.CORNFLOWER_BLUE)

  def setup(self):
```

```
        pass

    def on_draw(self):
        self.clear()


def main():
    window = MyGame()
    window.setup()
    arcade.run()


if __name__ == "__main__":
    main()
```

Как мы видим все наше окно - класс, который наследуется от arcade.Window.
С методом __init__ мы уже знакомы и не будем затрагивать его, а остальные разберем подробнее.
Метод setup - не встроенный метод, поэтому в функции main мы вызываем данный метод в отличие от других. В нем мы описываем, некие стартовые параметры, по типу координат объекта и т.п,а в методе __init__, мы как пример создаем сами объекты, необходимые при старте игры. Также большим плюсом использования двух методов, является возможность добавления в нашу игру в дальнейшем нового функционала, например перезапуска игры.

Метод on_draw отвечает за отрисовку всех объектов в игре, будь то персонаж, фон, либо счет.
Функция main, понятное дело создает экземпляр класса окна и запускает необходимые методы.
Как мы видим класс arcade.Window имеет уже некоторые встроенные методы для отрисовки, как пример, on_draw(). Существуют методы позволяющие обрабатывать нажатие клавиш клавиатуры, мыши или же нажатие кнопок на геймпаде.

Давайте попробуем нарисовать смайлик, метод setup пока трогать не будем.
Так как смайлик состоит из обычных фигур, которые нам требуется отрисовать, то писать мы будем в методе on_draw. Для начала нарисуем желтый круг, с помощью метода draw_circle_filled(залитый цветом круг)

```
def on_draw(self):
    self.clear()
    arcade.draw_circle_filled(300, 300, 200, arcade.color.YELLOW)
```

Данный метод принимает координаты по x и y для нашего круга, его радиус и соответственно цвет. После запуска, в указанных координатах у нас отрисовался наш желтый шар. Добавим два глаза. Глаза это тоже круги, так что ничего нового

```
def on_draw(self):
  self.clear()
  arcade.draw_circle_filled(300, 300, 200, arcade.color.YELLOW)
  arcade.draw_circle_filled(380, 350, 20, arcade.color.BLACK)
  arcade.draw_circle_filled(220, 350, 20, arcade.color.BLACK)
```

И добавим улыбку - дугу, с помощью команды **draw_arc_outline**, которой так же передаем координаты центра, а также ширину и высоту дуги, цвет, углы, под которыми дуга начинается и заканчиваются, и ширину самой линии - для удобства можем создать отдельные переменные, которые использовать в самой команде

```
center_x = 300
center_y = 230
width = 150
height = 80
start_angle = 180
end_angle = 360
line_width = 10
arcade.draw_arc_outline(center_x, center_y, width, height, arcade.color.BLACK, start_angle, end_angle,
                line_width)
```

Отлично у нас получился смайлик.
Теперь давайте попробуем нарисовать небольшой пейзаж
Создадим новый шаблон

```
import arcade

SCREEN_WIDTH = 600
SCREEN_HEIGHT = 400
SCREEN_TITLE = "Platformer"


class MyGame(arcade.Window):
  def __init__(self):
    super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)
    arcade.set_background_color(arcade.csscolor.CORNFLOWER_BLUE)

  def setup(self):
    pass

  def on_draw(self):
```

```
    self.clear()


def main():
  window = MyGame()
  window.setup()
  arcade.run()


if __name__ == "__main__":
  main()
```

Нарисуем траву/землю с помощью команды draw_rectangle_filled, которой также передаем координаты центра по ширине и высоте, ширину и высоту фигуры (получается занимаемое пространство будет половина длины влево-вправо и половина высоты вверх-вниз от центральных координат), и цвет:

```
def on_draw(self):
  self.clear()
  arcade.draw_rectangle_filled(300, 100, 600, 200, arcade.color.GREEN)
```

Нарисуем солнышко - уже знакомый нам круг:

```
arcade.draw_circle_filled(100, 350, 20, arcade.color.YELLOW)
```

Можно нарисовать птичек - для них удобнее сделать отдельную функцию bird, в которой будем отрисовывать две дуги - два крыла. Расположение птицы на окне будем задавать при вызове функции в методе on_draw:

```
def bird(self, x, y):
  arcade.draw_arc_outline(x, y, 20, 20, arcade.color.BLACK, 0, 90)  # левое крыло
  arcade.draw_arc_outline(x + 20, y, 20, 20, arcade.color.BLACK, 90, 180)  # правое крыло
```

```
def on_draw(self):
  …
  self.bird(300, 300)
  self.bird(400, 350)
```

Дерево - тоже отдельной функцией. Рисуем прямоугольник и круг:

```
def tree(self, x, y):
  arcade.draw_rectangle_filled(x, y, 20, 90, arcade.color.DARK_BROWN)
  arcade.draw_circle_filled(x , y + 40, 40, arcade.color.DARK_GREEN)
```

```
self.tree(500, 120)
self.tree(120, 100)
```

Можно еще например нарисовать простой дом - прямоугольники для стены и окошка, а вот для крыши - новая фигура, треугольник, которой нужно задать три пары координат - вершины треугольника:

```
def house(self, x, y):
  arcade.draw_rectangle_filled(x, y, 100, 80, arcade.color.CORN) # стена
  arcade.draw_rectangle_filled(x, y, 30, 30, arcade.color.LIGHT_BLUE) # окно
  arcade.draw_triangle_filled(x1=x, y1=y+80, x2=x-50, y2=y+40, x3=x+50, y3=y+40,
color=arcade.color.RED_BROWN) # крыша
```

```
self.house(330, 125)
```

На следующем занятии мы разберем основные методы класса игрового окна, которые позволяют нам не только, указывать что необходимо отрисовывать или задавать как пример заголовок приложения, но и как пример, как отключить изменение размеров окна, должно ли оно быть на весь экран или же вдруг мы хотим изменить положения окна

Давайте сегодня прежде, чем мы начнем писать хотя бы простейшие игры, попробуем разобрать, какие же методы мы можем использовать при создании игрового окна.
Напишем минимальный код, для создания окна

```
import arcade

class Game(arcade.Window):
  def __init__(self):
    super().__init__()


  def on_draw(self):
    self.clear()


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

При запуске мы видим черное окно размером 800 на 600 и заголовком Arcade Window. Данные значения указаны, как значения по умолчанию и если мы не указываем другие значения, то используются они. Давайте изменим размер окна и заголовок

```
def __init__(self):
  super().__init__()
  self.height = 300
  self.width = 400
  self.set_caption('Моя игра')
```

В данном случае мы изменили значения высоты и ширины окна, а также с помощью метода set_caption(), установили новое значение заголовка. Также эти данные можно передать в напрямую в методе __init__.

```
class Game(arcade.Window):
  def __init__(self):
    super().__init__(width=400, height=300, title='Моя игра')
```

Либо же при создании объекта

```
import arcade

class Game(arcade.Window):
  def __init__(self, width, height, title):
    super().__init__(width=width, height=height, title=title)



  def on_draw(self):
    self.clear()


if __name__ == '__main__':
  game = Game(400, 300, 'Моя игра')
  arcade.run()
```

Давайте оставим как самый оптимальный второй вариант и продолжим рассмотрение на его примере, только размеры, для удобства, сделаем 800x600. Какие еще возможности, нам, предоставляет класс arcade.Window? Давайте рассмотрим. Ну, например, мы можем сделать окно на весь экран, указав параметр *fullscreen=True*

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', fullscreen=True)
```

Если вы хотите запретить изменять размер экрана(растягивать, сжимать), то для данной цели есть параметр *resizable*. Для отключения возможности изменять размер, ему требуется указать значение False.

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', resizable=False)
```

Также мы можем указать будет ли отображаться окно или нет

```
class Game(arcade.Window):
  def __init__(self):
    super().__init__(width=800, height=600, title='Моя игра', resizable=False, visible=False)
```

Дополнительно мы можем использовать также один из этих параметров

- update_rate (float) – Как часто требуется обновлять игровое окно.
- antialiasing (bool) – Использовать ли антиалиасинг
- gl_version (Tuple[int,int]) – Какую версию OpenGL запрашивать.(3, 3) по умолчанию, и значение можно переопределить при использовании более продвинутых функций OpenGL.
- vsync (bool) – Использование вертикальной синхронизации, для более плавной анимации
- gc_mode (bool) – Определяет как объекты будут собираться сборщиком мусора Python(context_gc - по умолчанию, либо же автоматически)
- center_window (bool) – Отобразить игровое окно в центре экрана
- samples (bool) – Качество сглаживания(по умолчанию 4).Можно устанавливать например 2, 4, 8, 16

Отлично с основными параметрами разобрались. Давайте теперь добавим цвет фона для нашего окна с помощью background_color

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', resizable=False)
  self.background_color = arcade.color.ALABAMA_CRIMSON
```

Как мы видим IDE, нам подсказывает какие цвета предлагаются модулем arcade, но также мы можем установить цвет и формате RGB

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', resizable=False)
  self.background_color = (0, 0, 150)
```

Разберем немного подробнее RGB.Мы указываем, сколько красного, зеленого и синего использовать в диапазоне чисел 0-255. Отсутствие света равно нулю. Если мы хотим чтобы цвет был максимально ярким, то он будет иметь значение 255. Нам нужны три цифры, чтобы

указать три цвета, поэтому (0, 0, 0) означает отсутствие красного, зеленого и синего, т.е черный цвет. Вот несколько других примеров:

Красный	Зеленый	Синий	Цвет
0	0	0	Черный
255	255	255	Белый
127	127	127	Серый
255	0	0	Красный
0	255	0	Зеленый
0	0	255	Синий
255	255	0	Желтый

Давайте посмотрим, а для чего же нам метод clear(), который находится в методе on_draw(). clear()- Очищает окно с настроенным цветом фона, установленным через background_color

Понятное дело, все возможности предоставляемые модулем arcade, мы с вами рассмотреть не сможем, но хотя бы немного рассмотрим некоторые. Как пример мы можем вернуть размер окна

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', resizable=False)
  self.background_color = (0, 0, 150)
  print(self.get_size())
```

Расположение окна

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', resizable=False)
  self.background_color = (0, 0, 150)
  print(self.get_location())
```

Установить окно в центр

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', resizable=False)
  self.background_color = (0, 0, 150)
  self.center_window()
```

Для более удобного рассмотрения других методов, давайте рассмотрим метод, который отвечает за нажатие на кнопку мыши. Данный метод on_mouse_press()

```
def on_mouse_press(self, x: int, y: int, button: int, modifiers: int):
  self.set_location(500, 500)
```

Данный метод принимает на координаты мыши(x, y), нажатую кнопку и модификаторы. В примере выше, мы указываем если мы нажали на любую кнопку мыши, то наш экран необходимо переместить в позицию 500, 500. Давайте еще потренируемся и попробуем при щелчке мыши изменить размер окна, заголовок и цвет

```
def on_mouse_press(self, x: int, y: int, button: int, modifiers: int):
  self.set_location(500, 500)
  self.set_size(300, 100)
  self.set_caption('Кнопка мыши нажата')
  self.background_color = (255, 0, 0)
```

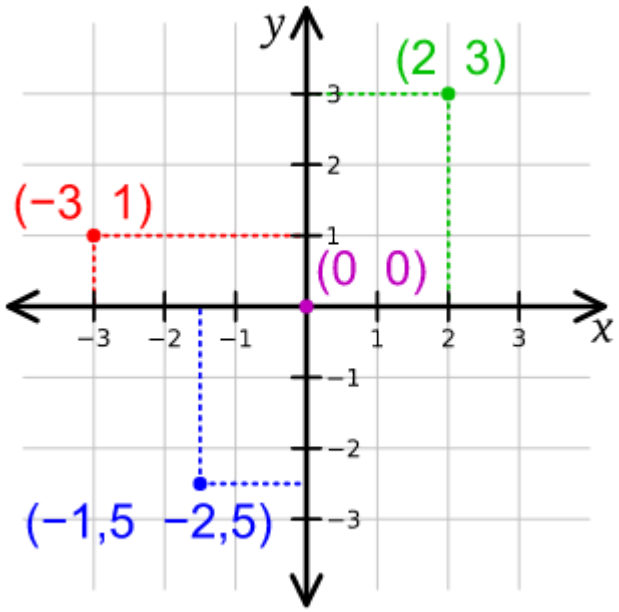
Давайте установим ограничение на минимальный и максимальный размер игрового окна.

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', resizable=True)
  self.background_color = (0, 0, 150)
  self.set_min_size(400, 300)
  self.set_max_size(900, 700)
```

Не всегда курсор мыши в игре нам может быть нужен, поэтому чтобы его сделать невидимым, мы можем воспользоваться set_mouse_visible()

```
def __init__(self):
  super().__init__(width=800, height=600, title='Моя игра', resizable=True)
  self.background_color = (0, 0, 150)
  self.set_mouse_visible(False)
```

Отлично. Мы научились указывать цвет и многое другое, следующее, что нам нужно узнать, - устроена система координат в arcade. Существует декартова система координат, которая выглядит следующим образом:



Наша графика будет отрисована с использованием этой же системы. Но есть вещи, о которых следует знать:

- 0,0 будет находиться в левом нижнем углу экрана, а все отрицательные координаты будут за кадром.
- Каждая “точка” будет представлять собой пиксель. Таким образом, окно шириной 800 пикселей будет иметь координаты x от 0 до 799. (Ноль - это один из пикселей, поэтому 0-799 - это 800 пикселей.)