Материалы к занятию

```
import arcade

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600


class Game(arcade.Window):
  def __init__(self):
    super().__init__(
      width=SCREEN_WIDTH,
      height=SCREEN_HEIGHT,
      title="Работа со спрайтами",
    )

  def setup(self):
    pass

  def on_draw(self):
    self.clear()

  def update(self, delta_time: float):
    pass


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

Спрайты-это простой способ создать двухмерный растровый объект в Arcade. В Arcade есть методы, которые облегчают рисование, перемещение и анимацию спрайтов. Мы также можем легко использовать спрайты для обнаружения столкновений между объектами.

Создать экземпляр класса Sprite в Arcade из изображения очень легко. Нам нужно только имя файла изображения, чтобы спрайт был создан, и необязательное число, для масштабирования изображения. Например:

```
SPRITE_SCALING_COIN = 0,2
self.coin = arcade.Sprite("coin_01.png", SPRITE_SCALING_COIN)
```

Этот код создаст спрайт, используя изображение, хранящееся под именем coin_01.png. Изображение будет уменьшено до 20% от его первоначальной высоты и ширины.
У каждого объекта, как мы знаем есть координаты, задать их мы можем с помощь значений center_x и center_y соответственно.

```
self.coin.center_x = 64
self.coin.center_y = 120
```

Также мы можем использовать спрайты из стандартного набора arcade, для этого необходимо указать :resourse: и путь до файла. Перейдем к практике. Давайте добавим персонажа на наше окно.
Создадим константы с размером

```
CHARACTER_SCALING = 1
```

В методе __init__ создадим переменную нашего игрока со значением None

```
self.player_sprite = None
```

В методе setup загрузим текстуру и укажем координаты

```
def setup(self):
  image_source = ":resources:images/animated_characters/female_adventurer/femaleAdventurer_idle.png"
  self.player_sprite = arcade.Sprite(image_source, CHARACTER_SCALING)
  self.player_sprite.center_x = 64
  self.player_sprite.center_y = 128
```

После этого в on_draw мы можем отрисовать нашего игрока

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

Также хорошей практикой является создания списка спрайтов и добавления их туда. В дальнейшем список спрайтов поможет реализовать нам анимацию. Отрисовывать в такой случае мы должны именно его.

```
class MyGame(arcade.Window):
  def __init__(self):
    super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)
    arcade.set_background_color(arcade.csscolor.CORNFLOWER_BLUE)
    self.player_sprite = None
    self.player_list = None

  def setup(self):
    self.player_list = arcade.SpriteList()

    image_source =
":resources:images/animated_characters/female_adventurer/femaleAdventurer_idle.png"
```

```
    self.player_sprite = arcade.Sprite(image_source, CHARACTER_SCALING)
    self.player_sprite.center_x = 64
    self.player_sprite.center_y = 128
    self.player_list.append(self.player_sprite)

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

При запуске, мы видим все тот же результат. Давайте разберемся в синтаксисе данных классов
class arcade.Sprite(filename=None, scale=1, image_x = 0, image_y =0, image_width = 0, image_height = 0, center_x= 0, center_y = 0, flipped_horizontally = False, flipped_vertically = False, flipped_diagonally = False, hit_box_algorithm = 'Simple', hit_box_detail = 4.5, texture = None, angle = 0)

Параметры

- filename (str) – имя изображения.
- scale (float) – масштаб изображения. По умолчанию 1.0
- image_x (float) – смещение по x к спрайту внутри списка спрайтов.
- image_y (float) – смещение по x к спрайту внутри списка спрайтов
- image_width (float) – ширина изображения
- image_height (float) – высота изображения
- center_x (float) – расположение по координате x
- center_y (float) – расположение по координате y
- flipped_horizontally (bool) – отразить зеркально по горизонтали
- flipped_vertically (bool) – отразить зеркально по вертикали
- flipped_diagonally (bool) – отразить зеркально по диагонали
- hit_box_algorithm (str) – алгоритм определения хитбокса. По умолчанию Simple. Существуют значения None, Simple, Detailed/
- texture (Texture) – текстура
- angle (float) –угол наклона спрайта в градусах

Давайте таким же образом создадим землю. Для начала создадим константу

```
TILE_SCALING = 0.5
```

В методе __init__ создадим переменную

```
self.ground_list = None
```

В методе setup укажем список спрайтов. Так как спрайт земли у нас будет не один, то через цикл for мы расставим землю на нашей карте.

```
self.ground_list = arcade.SpriteList(use_spatial_hash=True)
for x in range(0, 1250, 64):
  ground = arcade.Sprite(":resources:images/tiles/grassMid.png", TILE_SCALING)
  ground.center_x = x
  ground.center_y = 32
  self.ground_list.append(ground)
```

Отлично теперь давайте попробуем расставить ящики аналогичным способом. Так как спрайт у нас для земли подгружается в цикле, то дополнительным циклом мы можем из этой же переменной сделать ящики

```
self.box_list = None
```

```
self.box_list = arcade.SpriteList(use_spatial_hash=True)
```

```
coordinate_list = [[512, 96], [256, 96], [768, 96]]

for coordinate in coordinate_list:
  box = arcade.Sprite(
     ":resources:images/tiles/boxCrate_double.png", TILE_SCALING
  )
 box.position = coordinate
  self.box_list.append(box)
```

Теперь можем отрисовать их в on_draw()

```
self.box_list.draw()
```

Далее мы добавим сцену в нашу игру. Сцена — это инструмент для управления несколькими различными списками SpriteLists, присваивая каждому из них имя и поддерживая порядок рисования.
SpriteLists можно рисовать напрямую, но Scene может быть полезен для обработки множества различных списков одновременно и возможности рисовать их все одним вызовом к сцене.
Для начала мы удалим наши списки спрайтов из функции и заменим их объектом сцены.

```
def __init__(self):
```

```
  super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)

  self.scene = None
  self.player_sprite = None
```

Далее мы инициализируем объект scene в функции, а затем добавим в него SpriteList вместо непосредственного создания новых объектов SpriteList в setup.
Затем, вместо того, чтобы добавлять Спрайты непосредственно в SpriteLists, мы можем добавить их в Scene и указать по имени, в какой SpriteList мы хотим, чтобы они были добавлены.

```
def setup(self):
  self.scene = arcade.Scene()
  self.scene.add_sprite_list("Player")
  self.scene.add_sprite_list("Walls", use_spatial_hash=True)

  image_source = ":resources:images/animated_characters/female_adventurer/femaleAdventurer_idle.png"
  self.player_sprite = arcade.Sprite(image_source, CHARACTER_SCALING)
  self.player_sprite.center_x = 64
  self.player_sprite.center_y = 128
  self.scene.add_sprite("Player", self.player_sprite)

  for x in range(0, 1250, 64):
    ground = arcade.Sprite(":resources:images/tiles/grassMid.png", TILE_SCALING)
    ground.center_x = x
    ground.center_y = 32
    self.scene.add_sprite("Walls", ground)

  coordinate_list = [[512, 96], [256, 96], [768, 96]]

  for coordinate in coordinate_list:
    # Add a crate on the ground
    box = arcade.Sprite(
      ":resources:images/tiles/boxCrate_double.png", TILE_SCALING
    )
    box.position = coordinate
    self.scene.add_sprite("Walls", box)
```

Конечно же отрисовываем в данном случае сцену

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

Таким образом мы научились базовой работе со спрайтами. В дальнейших занятиях, мы с вами разберем, также как создавать анимацию из спрайтов и как создать управление для игровых объектов. В примере выше мы построили наше игровое окружение с помощью

цикла, но при разработке игр мы также можем пользоваться сторонним ПО для создания карты.

Давайте для начала добавим возможность двигаться нашему персонажу.
Создадим константу со скоростью персонажа

```
PLAYER_MOVEMENT_SPEED = 5
```

Каждый спрайт имеет атрибуты. Их изменение приведет к изменению местоположения спрайта. (Существуют также атрибуты для верхнего, нижнего, левого, правого угла, которые будут перемещать спрайт)

Каждый спрайт имеет и переменные. Они могут быть использованы для удержания скорости, с которой движется спрайт. Мы будем корректировать их в зависимости от того, какую клавишу пользователь нажимает. Если пользователь нажимает клавишу со стрелкой вправо, мы хотим получить положительное значение для смещения. Если значение равно 5, оно будет перемещать 5 пикселей на кадр.
В этом случае, когда пользователь нажимает клавишу, мы изменяем спрайты, меняем x и y.
Создадим метод, который срабатывает в момент нажатия на клавишу управления.

```
def on_key_press(self, key, modifiers):
  if key == arcade.key.UP or key == arcade.key.W:
    self.player_sprite.change_y = PLAYER_MOVEMENT_SPEED
  elif key == arcade.key.DOWN or key == arcade.key.S:
    self.player_sprite.change_y = -PLAYER_MOVEMENT_SPEED
  elif key == arcade.key.LEFT or key == arcade.key.A:
    self.player_sprite.change_x = -PLAYER_MOVEMENT_SPEED
  elif key == arcade.key.RIGHT or key == arcade.key.D:
    self.player_sprite.change_x = PLAYER_MOVEMENT_SPEED
```

Метод on_key_press() принимает нажатую клавишу(key) и модификаторы(например SHIFT+D)
В данном методе мы проверяем если нажали какую либо клавишу, то наш персонаж будет двигаться по оси x, либо y . У объекта класса arcade.Sprite() есть поля chenge_x и change_y, которые будут отвечать за смещение по данным осям.

Пока наш объект не двигается. Для того, чтобы исправить данную проблему, нам необходимо вызвать метод update(), для нашего объекта класса arcade.Sprite()

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

Результатом мы видим, что наш объект двигается, но после того как мы отпустили клавишу, он продолжает движение в последнем направлении, ну и конечно же он может проходить сквозь объекты. Обработку коллизий мы рассмотрим в следующем занятии, а на этом мы сделаем имитацию гравитации и обработки коллизий.

Добавим метод, который срабатывает, когда пользователь отпускает клавишу(не хотим же мы, чтобы персонаж двигался бесконечно). Данный метод проверяет, какую нажатую клавишу мы отпустили.

```
def on_key_release(self, key, modifiers):
  if key == arcade.key.UP or key == arcade.key.W:
     self.player_sprite.change_y = 0
  elif key == arcade.key.DOWN or key == arcade.key.S:
     self.player_sprite.change_y = 0
  elif key == arcade.key.LEFT or key == arcade.key.A:
     self.player_sprite.change_x = 0
  elif key == arcade.key.RIGHT or key == arcade.key.D:
     self.player_sprite.change_x = 0
```

Как только мы нажимаем на клавишу управления срабатывает метод on_key_press, а как только отпускаем клавишу on_key_release. В методе update мы будем прописывать ту логику, которая должна обновляться постоянно в течении игры.

Единственная проблема - персонаж как бы повисает в воздухе. Давайте реализуем простейшую гравитацию для нашего персонажа и прыжок(продвинутую физику мы рассмотрим позднее)
Создадим две переменные - сила прыжка и гравитация.

```
GRAVITY = 2
PLAYER_JUMP_SPEED = 20
```

Давайте теперь для удобства создадим отдельный класс для нашего персонажа и перенесем все необходимые данные туда

```
class Player(arcade.Sprite):
  def __init__(self):
     super().__init__(
        ":resources:images/animated_characters/female_adventurer/femaleAdventurer_idle.png",
        CHARACTER_SCALING,
     )
     self.center_x = 64
     self.center_y = 128
```

```

```

Также добавим метод update() и укажем что персонаж постоянно должен падать

```
def update(self):
  self.center_x += self.change_x
  self.center_y += self.change_y
  self.center_y -= GRAVITY
```

Персонаж сразу же проваливается сквозь землю. Давайте укажем, что он не может находится ниже земли.

```
def update(self):
  self.center_x += self.change_x
  self.center_y += self.change_y
  self.center_y -= GRAVITY
  if self.center_y <= 128:
     self.center_y = 128
```

Отлично теперь сделаем имитацию прыжка и уберем возможность двигаться персонажем вниз. Для этого изменим метод on_key_press. Укажем если мы нажимаем клавишу вверх, то наш персонаж изменит свое положение на значение силы прыжка.

```
def on_key_press(self, key, modifiers):

  if key == arcade.key.UP or key == arcade.key.W:
     self.player_sprite.change_y = PLAYER_JUMP_SPEED
  elif key == arcade.key.LEFT or key == arcade.key.A:
     self.player_sprite.change_x = -PLAYER_MOVEMENT_SPEED
  elif key == arcade.key.RIGHT or key == arcade.key.D:
     self.player_sprite.change_x = PLAYER_MOVEMENT_SPEED
```

Также изменим метод on_key_release

```
def on_key_release(self, key, modifiers):
  if key == arcade.key.LEFT or key == arcade.key.A:
     self.player_sprite.change_x = 0
  elif key == arcade.key.RIGHT or key == arcade.key.D:
     self.player_sprite.change_x = 0
  elif  key == arcade.key.W or key == arcade.key.UP:
     self.player_sprite.change_y = 0
```

Наша “псевдогравитация” работает. Единственная проблема - персонаж может использовать двойной прыжок. Давайте исправим данную ситуацию. Для этого введем переменную в основном классе, которая будет отвечать за состояние(на земле или нет)

```
self.isGround = True
```

Также изменим условие, что клавиша вверх будет срабатывать, только если значение переменной True, а также будем переключать ее в False в момент прыжка

```
if self.isGround:
  if key == arcade.key.UP or key == arcade.key.W:
    self.player_sprite.change_y = PLAYER_JUMP_SPEED
    self.isGround = False
```

Ну и соответственно если мы на земле, то меняем значение на True в update класса персонажа.

```
if self.center_y <= 128:
  self.center_y = 128
  game.isGround = True
```

Как мы видим проблема двойного прыжка исправлена.

Давайте теперь сделаем, чтобы ящик двигался вверх вниз как платформа. Для начала вынесем необходимое в класс для ящика

```
class Box(arcade.Sprite):
  def __init__(self):
    super().__init__(":resources:images/tiles/boxCrate_double.png", TILE_SCALING)
```

Также внесем изменения в setup основного класса и оставим один ящик

```
self.box = Box()
self.box.position = [512, 96]
self.scene.add_sprite("Walls", self.box)
```

Запускаем. Результат как и прежде. Отлично. Давайте теперь займемся движением ящика. Для начала определимся, что ящики будут двигаться по координате у и создадим метод update

```
class Box(arcade.Sprite):
  def __init__(self):
    super().__init__(":resources:images/tiles/boxCrate_double.png", TILE_SCALING)
    self.change_y = 2

  def update(self):
```

```
    self.center_y += self.change_y
```

И запустим метод update()

```
def update(self, delta_time: float):
  self.player_sprite.update()
  self.box.update()
```

Ящик двигается и улетает вверх. Добавим условие в update класса ящика, чтобы он двигался попеременно вверх и вниз по достижении отдельных координат.

```
def update(self):
  self.center_y += self.change_y
  if self.top >= 500:
    self.change_y = -self.change_y
  elif self.bottom <= 100:
    self.change_y = -self.change_y
```

В данном случае мы используем не center_y, а top. У спрайтов мы можем определять края с помощью top, right, left и bottom соответственно. Таким образом, мы проверяем не достиг ли вверх спрайта координаты 500
Запускаем и видим, что наш ящик как будто застрял и не может двинуться вверх. Все дело в том, что мы не знаем начальных координат нижнего края спрайта. Для этого мы можем вывести значение в консоль

```
print(self.bottom)
```

Как мы видим положение края 64. Изменим наше условие, исходя из новых данных

```
def update(self):
  self.center_y += self.change_y
  print(self.bottom)
  if self.top >= 500:
    self.change_y = -self.change_y
  elif self.bottom <= 64:
    self.change_y = -self.change_y
```

Отлично. Давайте последнее, что добавим это задержку в одну секунду когда ящик достигает своих крайних координат и немного оптимизируем условие

```
def update(self):
```

```
self.center_y += self.change_y
if self.top >= 500 or self.bottom <= 64:
    time.sleep(1)
    self.change_y = -self.change_y
```