Материалы к занятию

Для начала создадим простой шаблон со спрайтами из прошлого занятия

```
import arcade


SCREEN_WIDTH = 1000
SCREEN_HEIGHT = 650
SCREEN_TITLE = "Простой платформер"

CHARACTER_SCALING = 1

TILE_SCALING = 0.5


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

    self.wall_list = None
    self.player_list = None
    self.player_sprite = None



  def setup(self):
    self.player_list = arcade.SpriteList()
    self.wall_list = arcade.SpriteList(use_spatial_hash=True)

    image_source = ":resources:images/animated_characters/male_adventurer/maleAdventurer_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)

    for x in range(0, 1250, 64):
      wall = arcade.Sprite(":resources:images/tiles/dirtMid.png", TILE_SCALING)
      wall.center_x = x
      wall.center_y = 32
      self.wall_list.append(wall)

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

    for coordinate in coordinate_list:

      wall = arcade.Sprite(
        ":resources:images/tiles/boxCrate_double.png", TILE_SCALING
      )

      wall.position = coordinate
      self.wall_list.append(wall)
```

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


def main():
  game = Game()
  game.setup()
  arcade.run()


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

Все строчки кода из примера выше нам уже знакомы. Далее мы добавим сцену. Со сценой мы уже тоже знакомы поэтому каких то сложностей возникнуть не должно. Также не забудем удалить наши списки спрайтов

```
class Game(arcade.Window):
  def __init__(self):
    super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)
    arcade.set_background_color(arcade.csscolor.AQUA)

    self.player_sprite = None
    self.scene = None

  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/male_adventurer/maleAdventurer_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):
      wall = arcade.Sprite(":resources:images/tiles/dirtMid.png", TILE_SCALING)
      wall.center_x = x
      wall.center_y = 32
      self.scene.add_sprite("Walls", wall)

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

    for coordinate in coordinate_list:

      wall = arcade.Sprite(
        ":resources:images/tiles/boxCrate_double.png", TILE_SCALING
```

```
        )

        wall.position = coordinate
        self.scene.add_sprite("Walls", wall)

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

Теперь приступим к управлению, но оно у нас будет реализовано немного по другому. Мы будем использовать простой физический движок поставляемый с arcade.

Добавим переменную со скоростью персонажа

```
PLAYER_MOVEMENT_SPEED = 5
```

Далее, в конце метода setup создадим физический движок, который будет перемещать нашего игрока и не позволит ему проходить сквозь объекты. Класс PhysicsEngineSimple принимает два параметра: движущийся спрайт и список спрайтов, по которым движущийся спрайт не может перемещаться.

```
self.physics_engine = arcade.PhysicsEngineSimple(
    self.player_sprite, self.scene.get_sprite_list("Walls")
)
```

Не забудем создать переменную в __init__

```
self.physics_engine = None
```

Добавим управление для нашего персонажа

```
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

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
```

```

```

Физический движок смотрит какие кнопки управления мы нажимаем и переместит игрока, если только он не врежется в стену.

Рассмотрим еще один нужный метод при разработке игр. Это метод on_update(), который вызывается примерно 60 раз в секунду. Мы указываем физическому движку переместить нашего игрока на основе значений change_x и change_y.

```
def on_update(self, delta_time):
  self.physics_engine.update()
```

Проверяем и как мы видим по результату просчет столкновений у нас уже реализован физическим движком и он не дает нам пройти сквозь указанные объекты.
Данный пример бы лучше подошел для игр с видом сверху, нам же все таки, как и в прошлой игре нам не хватает гравитации. Давайте рассмотрим, как нам ее позволит реализовать наш движок.
Создадим переменные отвечающие за прыжок и гравитацию

```
GRAVITY = 1
PLAYER_JUMP_SPEED = 20
```

и изменим строку с настройкой движка в setup()

```
self.physics_engine = arcade.PhysicsEnginePlatformer(
  self.player_sprite, gravity_constant=GRAVITY, walls=self.scene["Walls"]
)
```

Мы указываем список объектов, с которыми игрок должен столкнуться в параметр walls физического движка. Физический движок платформера имеет параметр платформ и стен. Разница между ними очень важна. Статические неподвижные списки спрайтов всегда следует указывать в параметре walls, а движущиеся спрайты в параметре platforms.

Затем измените обработчики нажатия клавиш вверх и вниз

```
def on_key_press(self, key, modifiers):
  if key == arcade.key.UP or key == arcade.key.W:
    if self.physics_engine.can_jump():
      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
```

В данном случае за реализацию прыжка отвечает метод can_jump(), который производит смещение по координате y, а также решает проблему двойного прыжка.

Давайте еще разберем, как можно взаимодействовать с камерой в arcade.
Мы можем добавить в наше приложение камеру, которая будет, как пример следовать за игроком, либо же перемещать ее согласно задуманному сценарию. В методе __init__ создадим переменную

```
self.camera = None
```

В методе setup создадим нашу камеру и укажем ее размеры

```
self.camera = arcade.Camera(self.width, self.height)
```

и активируем ее в on_draw()

```
self.camera.use()
```

После запуска ничего нового мы пока не увидим. Мы можем использовать функцию перемещения камеры, чтобы переместить ее в другое положение либо же сделать так, чтобы камера следовала за игроком

Создадим метод для вычисления координат центра нашего игрока относительно экрана, а затем переместим камеру на него. Затем мы можем вызвать этот метод в on_update, чтобы камера следовала постоянно. Новая позиция будет отрисована в методе в on_draw

```
def center_camera_to_player(self):
  screen_center_x = self.player_sprite.center_x - (self.camera.viewport_width / 2)
  screen_center_y = self.player_sprite.center_y - (
    self.camera.viewport_height / 2
  )
  if screen_center_x < 0:
    screen_center_x = 0
  if screen_center_y < 0:
    screen_center_y = 0
  player_centered = screen_center_x, screen_center_y
  self.camera.move_to(player_centered)
```

В данном методе мы с вами сначала рассчитываем координаты центра по x и y для камеры, вычитая из центра игрока половину размера камеры. Также прописываем условия, чтобы камера не выходила за пределы экрана и перемещаем камеру с помощью метода move_to()

Ну и запускаем данный метод в on_update()

```
self.center_camera_to_player()
```

Как мы видим камера следуем за персонажем. Давайте добавим врага и если мы его коснулись, то игра будет завершаться и выводится в консоль сообщение о проигрыше, а также добавим выход в конец, который будет выводить сообщение о победе.

```
self.enemy = None
```

```
enemy_sprite = ":resources:images/animated_characters/zombie/zombie_idle.png"
self.enemy = arcade.Sprite(enemy_sprite, CHARACTER_SCALING)
self.enemy.center_x = 400
self.enemy.center_y = 128
self.scene.add_sprite("Enemy", self.enemy)
```

Для проверки столкновения двух спрайтов существует метод check_for_collision, который принимает два спрайта, от которых мы ожидаем столкновение. Данный метод возвращает булево значение True или False.

```
def on_update(self, delta_time):
  self.physics_engine.update()
  self.center_camera_to_player()
  if arcade.check_for_collision(self.player_sprite, self.enemy):
    self.player_sprite.kill()
    arcade.close_window()
    print('Вы проиграли')
```

Для уничтожения объекта мы используем kill(), для закрытия приложения close_window(). Так как приложение закрывается очень быстро, то мы не успеем увидеть как исчезает игрок. Можно закомментировать строку arcade.close_window() и мы увидим, что это работает.

Теперь давайте добавим выход для нашего игрока. Пока что он будет тоже просто закрывать приложение и выводить сообщение о победе

```
self.exit_player = None
```

```
exit_sprite = ":resources:images/tiles/signExit.png"
self.exit_player = arcade.Sprite(exit_sprite, CHARACTER_SCALING)
self.exit_player.center_x = 1200
self.exit_player.center_y = 128
self.scene.add_sprite("Exit", self.exit_player)
```

```
if arcade.check_for_collision(self.player_sprite, self.exit_player):
  arcade.close_window()
  print('Вы победили')
```

Готово. Теперь мы знаем, как проверить коллизию двух спрайтов, но если же у нас их больше, например более одного врага, то данный метод уже не подойдет.

Для начала создадим стандартный минимальный шаблон

```
import arcade

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
SCREEN_TITLE = 'Космический шутер'


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

  def setup(self):
    pass

  def on_draw(self):
    self.clear()


def main():
  game = Game()
  game.setup()
  arcade.run()
```

Раз уж мы собрались делать с вами игру, то давайте добавим какой нибудь фон.

```
def __init__(self):
  super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)
  self.background_texture = arcade.load_texture(':resources:images/backgrounds/stars.png')

def on_draw(self):
  self.clear()
  arcade.draw_texture_rectangle(SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2, SCREEN_WIDTH,
SCREEN_HEIGHT, self.background_texture)
```

Отлично. Теперь создадим необходимые переменные и создадим игрока

```
SPRITE_SCALING_PLAYER = 0.5
```

```
class Game(arcade.Window):
  def __init__(self):
    super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)
    self.background_texture = arcade.load_texture(':resources:images/backgrounds/stars.png')
    self.player_sprite = None

  def setup(self):
    self.player_sprite = Player()


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

```
    arcade.draw_texture_rectangle(SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2, SCREEN_WIDTH,
SCREEN_HEIGHT,
                                  self.background_texture)
    self.player_sprite.draw()
```

В данном случае физика нам не нужна, поэтому мы сразу с вами приступим к управлению и стрельбе нашего корабля. Управление и стрельбу мы сделаем с помощью мышки.

```
def on_mouse_motion(self, x: int, y: int, dx: int, dy: int):
  self.player_sprite.center_x = x
  self.player_sprite.center_y = y
```

Метод on_mouse_motion отвечает за смещение курсора нашей мыши. Он принимает координаты x, y мыши, а также расстояние на которое мышь сместилась по этим координатам с последнего вызова.
Не забудем для удобства сделать курсор мыши невидимым

```
self.set_mouse_visible(False)
```

Неплохо будет сделать, чтобы наш корабль не залетал в самый верх экрана. Ограничим его серединой игрового окна

```
def on_mouse_motion(self, x: int, y: int, dx: int, dy: int):
  self.player_sprite.center_x = x
  self.player_sprite.center_y = y
  if self.player_sprite.center_y >= SCREEN_HEIGHT / 2:
    self.player_sprite.center_y = SCREEN_HEIGHT / 2
```

Отлично. Управление готово. Осталось реализовать стрельбу. Для этого воспользуемся методом on_mouse_press. Данный метод принимает координаты мыши, нажатую кнопку, а также модификаторы. Пока у нас нет объекта патрона выведем просто в консоль

```
def on_mouse_press(self, x: int, y: int, button: int, modifiers: int):
  if button == arcade.MOUSE_BUTTON_LEFT:
    print('Shot')
```

При нажатии на левую кнопку мыши у нас срабатывает условие и сообщение выводится в консоль. Создадим нашу пулю, а вернее лазер.

```
class Laser(arcade.Sprite):
  def __init__(self):
    super().__init__(':resources:images/space_shooter/laserBlue01.png', SPRITE_SCALING_LASER)
```

В методе init создадим необходимые переменные.

```
self.laser_sprite = None
self.laser_sprite_list = None
```

В данном случае так как выстрелов может быть достаточно большое количество, мы объекты нашего лазера будем помещать в список при нажатии на левую кнопку мыши

```
def on_mouse_press(self, x: int, y: int, button: int, modifiers: int):
  if button == arcade.MOUSE_BUTTON_LEFT:
    self.laser_sprite = Laser()
    self.laser_sprite.bottom = self.player_sprite.top
    self.laser_sprite.center_x = self.player_sprite.center_x
    self.laser_sprite_list.append(self.laser_sprite)
```

И уже данный список отрисовывать

```
self.laser_sprite_list.draw()
```

Запускаем и видим, что наш лазер не летит и он повернут в другую сторону. Исправим для начала его положение

```
super().__init__(':resources:images/space_shooter/laserBlue01.png', SPRITE_SCALING_LASER, angle=90)
```

Дополнительным параметром мы указали угол наших лазеров. Теперь добавим движение вверх игрового окна

```
class Laser(arcade.Sprite):
  def __init__(self):
    super().__init__(':resources:images/space_shooter/laserBlue01.png', SPRITE_SCALING_LASER,
angle=90)
    self.change_y = 2

  def update(self):
    self.center_y += self.change_y
```

```
def on_update(self, delta_time: float):
  self.laser_sprite_list.update()
```

Все работает, но единственная проблема, что наши выстрелы не уничтожаются, когда долетают до верхнего края. Исправим это

```
def update(self):
  self.center_y += self.change_y
```

```
if self.bottom >= SCREEN_HEIGHT:
    self.kill()
```

В примере выше мы указали, что если наш патрон долетит до верхней части и положение нижней части спрайта будет больше, либо равно высоте экрана, то мы его уничтожим.

Реализация для персонажа готова. Теперь можно приступить и к врагам. Мы сделаем 30 врагов, которые должны будем уничтожить. Для начала создадим класс врага

```
class Enemy(arcade.Sprite):
  def __init__(self):
    super().__init__(':resources:images/space_shooter/playerShip3_orange.png', SPRITE_SCALING_ENEMY)
```

Добавим все необходимые переменные

```
SPRITE_SCALING_ENEMY = 0.5
```

```
self.enemy_sprite = None
self.enemy_sprite_list = None
```

```
def setup(self):
  self.player_sprite = Player()
  self.laser_sprite_list = arcade.SpriteList()
  self.enemy_sprite_list = arcade.SpriteList()
```

В данном случае создавать наши объекты мы будем в методе setup. Так как их требуется достаточно большое количество, то использовать мы будем цикл

```
self.enemy_sprite_list = arcade.SpriteList()
for _ in range(1, 31):
  self.enemy_sprite = Enemy()
  self.enemy_sprite.center_x = random.randint(0, SCREEN_WIDTH)
  self.enemy_sprite.center_y = SCREEN_HEIGHT + _ * 50
  self.enemy_sprite_list.append(self.enemy_sprite)
```

Отрисуем наш список

```
self.enemy_sprite_list.draw()
```

И добавим движения для наших врагов, а также развернем их в нужном направлении

```
class Enemy(arcade.Sprite):
  def __init__(self):
```

```
    super().__init__(':resources:images/space_shooter/playerShip3_orange.png', SPRITE_SCALING_ENEMY,
angle=180)
    self.change_y = 1

  def update(self):
    self.center_y -= self.change_y
```

Не забываем вызвать метод update

```
def on_update(self, delta_time: float):
  self.laser_sprite_list.update()
  self.enemy_sprite_list.update()
```

Ну и приступим к тому, что необходимо реализовать обработку коллизий нашего лазера и врага. В методе on_update мы переберем список наших лазеров, далее сравним было ли у какого то из них столкновение со списков врагов. Если столкновение было, то уничтожим найденный объект лазера

```
for laser in self.laser_sprite_list:
  shot_list = arcade.check_for_collision_with_list(laser, self.enemy_sprite_list)
  if shot_list:
    laser.kill()
```

Для проверки столкновений объекта и списка, мы используем метод check_for_collision_with_list, который принимает объект и список объектов и возвращает список столкновений. Теперь осталось уничтожить также и наших врагов. Добавляем в последнее условие перебор нашего списка столкновений и уничтожение данного врага.

```
for laser in self.laser_sprite_list:
  shot_list = arcade.check_for_collision_with_list(laser, self.enemy_sprite_list)
  if shot_list:
    laser.kill()
    for enemy in shot_list:
      enemy.kill()
```

Отлично. Теперь добавим победу. Если мы уничтожили все вражеские объекты, то наша игра просто будет останавливаться. Добавим переменную для указания статуса игры

```
self.status = True
```

Теперь возьмем все в методе on_update в условие, что если статус True

```
def on_update(self, delta_time: float):
  if self.status:
    self.laser_sprite_list.update()
    self.enemy_sprite_list.update()
```

```
    for laser in self.laser_sprite_list:
        shot_list = arcade.check_for_collision_with_list(laser, self.enemy_sprite_list)
        if shot_list:
            laser.kill()
            for enemy in shot_list:
                enemy.kill()
```

То же самое сделаем для управления и выстрела

```
def on_mouse_motion(self, x: int, y: int, dx: int, dy: int):
  if self.status:
      self.player_sprite.center_x = x
      self.player_sprite.center_y = y
      if self.player_sprite.center_y >= SCREEN_HEIGHT / 2:
          self.player_sprite.center_y = SCREEN_HEIGHT / 2

def on_mouse_press(self, x: int, y: int, button: int, modifiers: int):
  if self.status:
      if button == arcade.MOUSE_BUTTON_LEFT:
          self.laser_sprite = Laser()
          self.laser_sprite.bottom = self.player_sprite.top
          self.laser_sprite.center_x = self.player_sprite.center_x
          self.laser_sprite_list.append(self.laser_sprite)
```

Ну и теперь нам останется переключать статус в False, если список наших врагов пуст в методе on_update

```
if not self.enemy_sprite_list:
  self.status = False
```