Материалы к занятию

Сегодня мы научимся создавать стартовые экраны, а также экраны проигрыша для нашей игры.

Представления позволяют легко переключать «виды» на то, что мы показываете в окне. Мы можем использовать это для поддержки добавления таких экранов, как:

- Стартовый экран
- Экран инструкций
- Экран проигрыша
- Экран паузы

Для начала создадим простой шаблон игры, где пользователь управляет мышью персонажем и собирает монетки.

```
import random
import arcade

# --- Constants ---
SPRITE_SCALING_PLAYER = 0.5
SPRITE_SCALING_COIN = .25
COIN_COUNT = 50

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
SCREEN_TITLE = "Implement Views Example"


class MyGame(arcade.Window):
   """ Our custom Window Class"""

   def __init__(self):
       super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT,
SCREEN_TITLE)

       self.player_list = None
       self.coin_list = None

       self.player_sprite = None
       self.score = 0

       self.set_mouse_visible(False)

       arcade.set_background_color(arcade.color.AMAZON)

   def setup(self):

       self.player_list = arcade.SpriteList()
       self.coin_list = arcade.SpriteList()

       self.score = 0
```

```
        self.player_sprite =
arcade.Sprite(":resources:images/animated_characters/female_perso
n/femalePerson_idle.png",
                                              SPRITE_SCALING_PLAYER)
        self.player_sprite.center_x = 50
        self.player_sprite.center_y = 50
        self.player_list.append(self.player_sprite)

        for i in range(COIN_COUNT):

            coin =
arcade.Sprite(":resources:images/items/coinGold.png",
                                    SPRITE_SCALING_COIN)

            coin.center_x = random.randrange(SCREEN_WIDTH)
            coin.center_y = random.randrange(SCREEN_HEIGHT)

            self.coin_list.append(coin)

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

        output = f"Score: {self.score}"
        arcade.draw_text(output, 10, 20, arcade.color.WHITE, 14)

    def on_mouse_motion(self, x, y, dx, dy):

        self.player_sprite.center_x = x
        self.player_sprite.center_y = y

    def on_update(self, delta_time):

        self.coin_list.update()

        coins_hit_list =
arcade.check_for_collision_with_list(self.player_sprite,
self.coin_list)

        for coin in coins_hit_list:
            coin.remove_from_sprite_lists()
            self.score += 1


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


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

```

```

Давайте для начала изменим название нашего основного класса на другой и укажем, что он наследуется от класса представления

```
class GameView(arcade.View):
```

Также изменим конструктор класса

```
super().__init__()
```

Отключим отображение мыши в окне в методе __init__

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

Теперь мы изменим функцию main, в которой мы будем создавать окно представление, а потом отображать его

```
def main():
   window = arcade.Window(SCREEN_WIDTH, SCREEN_HEIGHT,
SCREEN_TITLE)
   start_view = GameView()
   window.show_view(start_view)
   start_view.setup()
   arcade.run()
```

Запускаем и наша игра работает как и прежде. Теперь добавим окно инструкции

```
class InstructionView(arcade.View):
   pass
```

Добавим в наш новый метод, в котором укажем цвет фона нашего окна

```
def on_show_view(self):
   arcade.set_background_color(arcade.csscolor.DARK_SLATE_BLUE)
   arcade.set_viewport(0, self.window.width, 0,
self.window.height)
```

Теперь добавим метод on_draw, в котором будет указываться необходимый текст и его координаты

```
def on_draw(self):
   self.clear()
   arcade.draw_text("Instructions Screen", self.window.width / 2,
self.window.height / 2,
                    arcade.color.WHITE, font_size=50,
anchor_x="center")
   arcade.draw_text("Click to advance", self.window.width / 2,
self.window.height / 2 - 75,
                    arcade.color.WHITE, font_size=20,
anchor_x="center")
```

и добавим метод on_mouse_press, который будет переключать нас в окно игры по щелчку мыши

```
def on_mouse_press(self, _x, _y, _button, _modifiers):
   game_view = GameView()
   game_view.setup()
   self.window.show_view(game_view)
```

Наконец изменим функцию main.

```
def main():
   window = arcade.Window(SCREEN_WIDTH, SCREEN_HEIGHT,
SCREEN_TITLE)
   start_view = InstructionView()
   window.show_view(start_view)
   arcade.run()
```

Проверяем все работает. Теперь добавим экран проигрыша

```
class GameOverView(arcade.View):

   def __init__(self):
       super().__init__()
       self.texture = arcade.load_texture("game_over.png")

       arcade.set_viewport(0, SCREEN_WIDTH - 1, 0, SCREEN_HEIGHT
- 1)

   def on_draw(self):
       self.clear()
       self.texture.draw_sized(SCREEN_WIDTH / 2, SCREEN_HEIGHT /
2,
                               SCREEN_WIDTH, SCREEN_HEIGHT)

   def on_mouse_press(self, _x, _y, _button, _modifiers):
       game_view = GameView()
```

```
        game_view.setup()
        self.window.show_view(game_view)
```

Укажем в методе on_update, если мы собрали все монеты, то будет показываться экран проигрыша

```
if len(self.coin_list) == 0:

  view = GameOverView()

  self.window.show_view(view)
```

Таким образом, мы можем создавать различные экраны и в нужный момент переключаться между ними

```
import os

import arcade

SCREEN_WIDTH = 1000
SCREEN_HEIGHT = 650
SCREEN_TITLE = "Platformer"

TILE_SCALING = 0.5
CHARACTER_SCALING = TILE_SCALING * 2
COIN_SCALING = TILE_SCALING
SPRITE_PIXEL_SIZE = 128
GRID_PIXEL_SIZE = SPRITE_PIXEL_SIZE * TILE_SCALING

PLAYER_MOVEMENT_SPEED = 7
GRAVITY = 1.5
PLAYER_JUMP_SPEED = 30

PLAYER_START_X = SPRITE_PIXEL_SIZE * TILE_SCALING * 2
PLAYER_START_Y = SPRITE_PIXEL_SIZE * TILE_SCALING * 1

RIGHT_FACING = 0
LEFT_FACING = 1

LAYER_NAME_MOVING_PLATFORMS = "Moving Platforms"
LAYER_NAME_PLATFORMS = "Platforms"
LAYER_NAME_COINS = "Coins"
LAYER_NAME_BACKGROUND = "Background"
LAYER_NAME_LADDERS = "Ladders"
LAYER_NAME_PLAYER = "Player"
```

```
def load_texture_pair(filename):
   return [
       arcade.load_texture(filename),
       arcade.load_texture(filename, flipped_horizontally=True),
   ]


class PlayerCharacter(arcade.Sprite):
   def __init__(self):
       super().__init__()

       self.character_face_direction = RIGHT_FACING

       self.cur_texture = 0
       self.scale = CHARACTER_SCALING

       self.jumping = False
       self.climbing = False
       self.is_on_ladder = False

       main_path =
":resources:images/animated_characters/male_person/malePerson"

       self.idle_texture_pair = load_texture_pair(f"{main_path}_idle.png")
       self.jump_texture_pair = load_texture_pair(f"{main_path}_jump.png")
       self.fall_texture_pair = load_texture_pair(f"{main_path}_fall.png")

       self.walk_textures = []
       for i in range(8):
           texture = load_texture_pair(f"{main_path}_walk{i}.png")
           self.walk_textures.append(texture)

       self.climbing_textures = []
       texture = arcade.load_texture(f"{main_path}_climb0.png")
       self.climbing_textures.append(texture)
       texture = arcade.load_texture(f"{main_path}_climb1.png")
       self.climbing_textures.append(texture)

       self.texture = self.idle_texture_pair[0]

       self.hit_box = self.texture.hit_box_points

   def update_animation(self, delta_time: float = 1 / 60):

       if self.change_x < 0 and self.character_face_direction ==
RIGHT_FACING:
           self.character_face_direction = LEFT_FACING
       elif self.change_x > 0 and self.character_face_direction ==
LEFT_FACING:
           self.character_face_direction = RIGHT_FACING

       if self.is_on_ladder:
           self.climbing = True
       if not self.is_on_ladder and self.climbing:
           self.climbing = False
       if self.climbing and abs(self.change_y) > 1:
           self.cur_texture += 1
           if self.cur_texture > 7:
               self.cur_texture = 0
```

```
        if self.climbing:
            self.texture = self.climbing_textures[self.cur_texture // 4]
            return

        if self.change_y > 0 and not self.is_on_ladder:
            self.texture =
self.jump_texture_pair[self.character_face_direction]
            return
        elif self.change_y < 0 and not self.is_on_ladder:
            self.texture =
self.fall_texture_pair[self.character_face_direction]
            return

        if self.change_x == 0:
            self.texture =
self.idle_texture_pair[self.character_face_direction]
            return

        self.cur_texture += 1
        if self.cur_texture > 7:
            self.cur_texture = 0
        self.texture = self.walk_textures[self.cur_texture][
            self.character_face_direction
        ]


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

        file_path = os.path.dirname(os.path.abspath(__file__))
        os.chdir(file_path)

        self.left_pressed = False
        self.right_pressed = False
        self.up_pressed = False
        self.down_pressed = False
        self.jump_needs_reset = False

        self.tile_map = None

        self.scene = None

        self.player_sprite = None

        self.physics_engine = None

        self.camera = None

        self.gui_camera = None

        self.end_of_map = 0

        self.score = 0

        self.collect_coin_sound =
arcade.load_sound(":resources:sounds/coin1.wav")
        self.jump_sound = arcade.load_sound(":resources:sounds/jump1.wav")
        self.game_over =
arcade.load_sound(":resources:sounds/gameover1.wav")
```

```
   def setup(self):
       self.camera = arcade.Camera(self.width, self.height)
       self.gui_camera = arcade.Camera(self.width, self.height)

       map_name = ":resources:tiled_maps/map_with_ladders.json"

       layer_options = {
           LAYER_NAME_PLATFORMS: {
               "use_spatial_hash": True,
           },
           LAYER_NAME_MOVING_PLATFORMS: {
               "use_spatial_hash": False,
           },
           LAYER_NAME_LADDERS: {
               "use_spatial_hash": True,
           },
           LAYER_NAME_COINS: {
               "use_spatial_hash": True,
           },
       }

       self.tile_map = arcade.load_tilemap(map_name, TILE_SCALING,
layer_options)

       self.scene = arcade.Scene.from_tilemap(self.tile_map)

       self.score = 0

       self.player_sprite = PlayerCharacter()
       self.player_sprite.center_x = PLAYER_START_X
       self.player_sprite.center_y = PLAYER_START_Y
       self.scene.add_sprite(LAYER_NAME_PLAYER, self.player_sprite)

       self.end_of_map = self.tile_map.width * GRID_PIXEL_SIZE

       if self.tile_map.background_color:
           arcade.set_background_color(self.tile_map.background_color)

       self.physics_engine = arcade.PhysicsEnginePlatformer(
           self.player_sprite,
           platforms=self.scene[LAYER_NAME_MOVING_PLATFORMS],
           gravity_constant=GRAVITY,
           ladders=self.scene[LAYER_NAME_LADDERS],
           walls=self.scene[LAYER_NAME_PLATFORMS]
       )

   def on_draw(self):
       self.clear()

       self.camera.use()

       self.scene.draw()

       self.gui_camera.use()

       score_text = f"Score: {self.score}"
       arcade.draw_text(
           score_text,
           10,
```

```
            10,
            arcade.csscolor.BLACK,
            18,
        )

    def process_keychange(self):

        if self.up_pressed and not self.down_pressed:
            if self.physics_engine.is_on_ladder():
                self.player_sprite.change_y = PLAYER_MOVEMENT_SPEED
            elif (
                self.physics_engine.can_jump(y_distance=10)
                and not self.jump_needs_reset
            ):
                self.player_sprite.change_y = PLAYER_JUMP_SPEED
                self.jump_needs_reset = True
                arcade.play_sound(self.jump_sound)
        elif self.down_pressed and not self.up_pressed:
            if self.physics_engine.is_on_ladder():
                self.player_sprite.change_y = -PLAYER_MOVEMENT_SPEED

        if self.physics_engine.is_on_ladder():
            if not self.up_pressed and not self.down_pressed:
                self.player_sprite.change_y = 0
            elif self.up_pressed and self.down_pressed:
                self.player_sprite.change_y = 0

        if self.right_pressed and not self.left_pressed:
            self.player_sprite.change_x = PLAYER_MOVEMENT_SPEED
        elif self.left_pressed and not self.right_pressed:
            self.player_sprite.change_x = -PLAYER_MOVEMENT_SPEED
        else:
            self.player_sprite.change_x = 0

    def on_key_press(self, key, modifiers):

        if key == arcade.key.UP or key == arcade.key.W:
            self.up_pressed = True
        elif key == arcade.key.DOWN or key == arcade.key.S:
            self.down_pressed = True
        elif key == arcade.key.LEFT or key == arcade.key.A:
            self.left_pressed = True
        elif key == arcade.key.RIGHT or key == arcade.key.D:
            self.right_pressed = True

        self.process_keychange()

    def on_key_release(self, key, modifiers):

        if key == arcade.key.UP or key == arcade.key.W:
            self.up_pressed = False
            self.jump_needs_reset = False
        elif key == arcade.key.DOWN or key == arcade.key.S:
            self.down_pressed = False
        elif key == arcade.key.LEFT or key == arcade.key.A:
            self.left_pressed = False
        elif key == arcade.key.RIGHT or key == arcade.key.D:
            self.right_pressed = False

        self.process_keychange()
```

```
   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, 0.2)

   def on_update(self, delta_time):

       self.physics_engine.update()

       if self.physics_engine.can_jump():
           self.player_sprite.can_jump = False
       else:
           self.player_sprite.can_jump = True

       if self.physics_engine.is_on_ladder() and not
self.physics_engine.can_jump():
           self.player_sprite.is_on_ladder = True
           self.process_keychange()
       else:
           self.player_sprite.is_on_ladder = False
           self.process_keychange()

       self.scene.update_animation(
           delta_time, [LAYER_NAME_COINS, LAYER_NAME_BACKGROUND,
LAYER_NAME_PLAYER]
       )

       self.scene.update([LAYER_NAME_MOVING_PLATFORMS])

       coin_hit_list = arcade.check_for_collision_with_list(
           self.player_sprite, self.scene[LAYER_NAME_COINS]
       )

       for coin in coin_hit_list:

           if "Points" not in coin.properties:
               print("Warning, collected a coin without a Points
property.")
           else:
               points = int(coin.properties["Points"])
               self.score += points

           coin.remove_from_sprite_lists()
           arcade.play_sound(self.collect_coin_sound)

       self.center_camera_to_player()


def main():
   window = MyGame()
```

```
   window.setup()
   arcade.run()


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

Давайте добавим сюда врагов.

Вынесем подключение текстур в отдельный класс, также создадим отдельный общий класс врага и классы типов врагов

```
class Entity(arcade.Sprite):
   def __init__(self, name_folder, name_file):
       super().__init__()
       self.facing_direction = RIGHT_FACING

       self.cur_texture = 0
       self.scale = CHARACTER_SCALING
       self.character_face_direction = RIGHT_FACING

       main_path =
f":resources:images/animated_characters/{name_folder}/{name_file}
"

       self.idle_texture_pair =
load_texture_pair(f"{main_path}_idle.png")
       self.jump_texture_pair =
load_texture_pair(f"{main_path}_jump.png")
       self.fall_texture_pair =
load_texture_pair(f"{main_path}_fall.png")

       self.walk_textures = []
       for i in range(8):
           texture =
load_texture_pair(f"{main_path}_walk{i}.png")
           self.walk_textures.append(texture)

       self.climbing_textures = []
       texture = arcade.load_texture(f"{main_path}_climb0.png")
       self.climbing_textures.append(texture)
       texture = arcade.load_texture(f"{main_path}_climb1.png")
       self.climbing_textures.append(texture)

       self.texture = self.idle_texture_pair[0]
       self.hit_box = self.texture.hit_box_points


class Enemy(Entity):
   def __init__(self, name_folder, name_file):
       super().__init__(name_folder, name_file)
```

```
class RobotEnemy(Enemy):
    def __init__(self):
        super().__init__("robot", "robot")


class ZombieEnemy(Enemy):
    def __init__(self):
        super().__init__("zombie", "zombie")
```

Изменим __init__ класса игрока и укажем, что он наследуется от нашего основного класса с анимациями

```
class PlayerCharacter(Entity):
    def __init__(self):
        super().__init__("male_person", "malePerson")

        self.jumping = False
        self.climbing = False
        self.is_on_ladder = False
```

Запускаем и наша игра все также работает. Отлично. Теперь пора создать врагов

```
LAYER_NAME_ENEMIES = "Enemies"
```

В методе setup создадим врагов

```
enemies_layer = self.tile_map.object_lists[LAYER_NAME_ENEMIES]

for my_object in enemies_layer:
    cartesian = self.tile_map.get_cartesian(
        my_object.shape[0], my_object.shape[1]
    )
    enemy_type = my_object.properties["type"]
    if enemy_type == "robot":
        enemy = RobotEnemy()
    elif enemy_type == "zombie":
        enemy = ZombieEnemy()
    else:
        raise Exception(f"Unknown enemy type {enemy_type}.")
    enemy.center_x = math.floor(
        cartesian[0] * TILE_SCALING * self.tile_map.tile_width
    )
```

```
   enemy.center_y = math.floor(
       (cartesian[1] + 1) * (self.tile_map.tile_height *
TILE_SCALING)
   )
   self.scene.add_sprite(LAYER_NAME_ENEMIES, enemy)
```

добавим стартовое меню и меню проигрыша при столкновении с врагом

Создадим стартовое меню

```
class MainMenu(arcade.View):

    def on_show_view(self):
        arcade.set_background_color(arcade.color.WHITE)

    def on_draw(self):
        self.clear()
        arcade.draw_text(
            "Main Menu - Click to play",
            SCREEN_WIDTH / 2,
            SCREEN_HEIGHT / 2,
            arcade.color.BLACK,
            font_size=30,
            anchor_x="center",
        )

    def on_mouse_press(self, _x, _y, _button, _modifiers):
        game_view = GameView()
        self.window.show_view(game_view)
```

Экран прогрыша

```
class GameOverView(arcade.View):

    def on_show_view(self):
        arcade.set_background_color(arcade.color.BLACK)

    def on_draw(self):
        self.clear()
        arcade.draw_text(
            "Game Over - Click to restart",
            SCREEN_WIDTH / 2,
            SCREEN_HEIGHT / 2,
            arcade.color.WHITE,
            30,
```

```
            anchor_x="center",
        )

    def on_mouse_press(self, _x, _y, _button, _modifiers):
        game_view = GameView()
        self.window.show_view(game_view)
```

Добавим метод вызывающий setup

```
def on_show_view(self):
   self.setup()
```

Добавим в on_update коллизии и вызов экрана проигрыша

```
player_collision_list = arcade.check_for_collision_with_lists(
   self.player_sprite,
   [
       self.scene[LAYER_NAME_COINS],
       self.scene[LAYER_NAME_ENEMIES],
   ],
)

for collision in player_collision_list:

   if self.scene[LAYER_NAME_ENEMIES] in collision.sprite_lists:
       arcade.play_sound(self.game_over)
       game_over = GameOverView()
       self.window.show_view(game_over)
       return
   else:
       if "Points" not in collision.properties:
           print("Warning, collected a coin without a Points
property.")
       else:
           points = int(collision.properties["Points"])
           self.score += points

       collision.remove_from_sprite_lists()
       arcade.play_sound(self.collect_coin_sound)
```