

OOP in Arcade

Videogames Technology
Asignatura transversal

Departamento de Automática

Objectives

1. Understand the OO API in Arcade
2. Use sprites and sprites sheets with Arcade
3. Handle user input
4. Understand some multimedia file formats
5. Introduce the Window, View and Sprite classes

Bibliography

1. Paul Craven. The Arcade Book. Chapter 18: Using the Window class. ([link](#)).
2. Paul Craven. The Arcade Book. Chapter 19: User control. ([link](#)).
3. Paul Craven. The Arcade Book. Chapter 21: Spriters and collisions. ([link](#)).
4. Paul Craven. Using Views for Start/End Screens. ([link](#)).

Table of Contents

1. The Window class

- Introduction
- Basic methods
- Main methods and attributes
- Background
- User control methods
- Other methods

2. Using a game controller

3. Life-cycle management

- Graphs
- Finite States Machines
- The View class

- The View class: Example

4. Sprites

- Introduction
- Spritesheets
- Data formats
- The `Sprite` class
- Examples
- Collision detection
- Other classes
- Sprite lists
- Locating sprites

5. Resources management

The Window class

Introduction

Arcade has an OOP API

- More features than structured API
- Easy to use API

The Window class

Introduction (II)

```

1  class MyGame(arcade.Window):
2      def __init__(self, width, height, title):
3          """ Initialize everything """
4
5          # Initialize the parent class
6          super().__init__(width, height, title)
7
8          arcade.set_background_color(arcade.color.AMAZON)
9
10     def setup(self):
11         """ Create the sprites and set up the game """
12         pass
13
14     def on_draw(self):
15         """ Render the screen. """
16
17         arcade.start_render()
18         # TODO: Drawing code goes here

```

The Window class

Introduction (III)

```

1 def main () :
2     """ Main method """
3     game = MyGame (SCREEN_WIDTH, SCREEN_HEIGHT, "My Game Title")
4     game.setup ()
5     arcade.run ()
6
7
8 if __name__ == "__main__":
9     main ()

```

The Window class

Constructor

Constructor

```
class arcade.Window(  
    width: int = 800,  
    height: int = 600,  
    title: Optional[str] = 'Arcade Window',  
    fullscreen: bool = False,  
    resizable: bool = False,  
    update_rate: Optional[float] = 0.016666666666666666,  
    antialiasing: bool = True)
```

Remember to use reference documentation

- (arcade.Window reference)
- $\frac{1}{0,016666666666666666} = 60\text{Hz}$
- Antialiasing: Smoothing transitions between colors and shapes

The Window class

Main methods and attributes

arcade.Window

Methods

- `setup()`.
Initialization
- `on_draw()`.
Drawing
- `on_update(delta_time: float)`.
Move everything. Perform collision checks. Do all the game logic here

Attributes

- `background_color`.

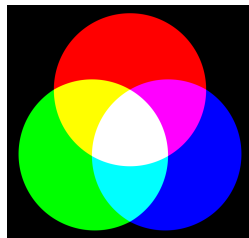
The Window class

Background (I)

```
1 # Use Arcade's built in color values
2 window.background_color = arcade.color.AMAZON
```

















(List of colors)

```
1 # Specify RGB value directly (red)
2 window.background_color = 255, 0, 0
```



The Window class

Background (II)

| Color | Color name | (R,G,B) | Hex |
|---|------------|---------------|---------|
|  | Black | (0,0,0) | #000000 |
|  | White | (255,255,255) | #FFFFFF |
|  | Red | (255,0,0) | #FF0000 |
|  | Lime | (0,255,0) | #00FF00 |
|  | Blue | (0,0,255) | #0000FF |
|  | Yellow | (255,255,0) | #FFFF00 |
|  | Cyan | (0,255,255) | #00FFFF |
|  | Magenta | (255,0,255) | #FF00FF |
|  | Silver | (192,192,192) | #C0C0C0 |
|  | Gray | (128,128,128) | #808080 |
|  | Maroon | (128,0,0) | #800000 |
|  | Olive | (128,128,0) | #808000 |
|  | Green | (0,128,0) | #008000 |
|  | Purple | (128,0,128) | #800080 |
|  | Teal | (0,128,128) | #008080 |
|  | Navy | (0,0,128) | #000080 |

(Source)

The Window class

User control methods (I)

User control methods

- `on_key_press(key)` Called when the user presses key.
- `on_key_release(symbol: int, modifiers: int)`. Called when the user presses key.
- `on_mouse_press(x: float, y: float, button: int, modifiers: int)`.
Called when the user presses a mouse button.
- `on_mouse_release(x: float, y: float, button: int, modifiers: int)`.
Called when the user releases a mouse button.
- `on_mouse_motion(x, y, delta_x, delta_y)`. Called whenever the mouse moves.

The Window class

User control methods: Examples

Capturing a key

```
1 def on_key_press(self, key, modifiers):  
2     if key == arcade.key.LEFT:  
3         print("Left key hit")  
4     elif key == arcade.key.A:  
5         print("The 'a' key was hit")
```

Capturing a mouse click

```
1 def on_mouse_press(self, x, y, button, modifiers):  
2     """ Called when the user presses a mouse button. """  
3  
4     if button == arcade.MOUSE_BUTTON_LEFT:  
5         print("Left mouse button pressed at", x, y)  
6     elif button == arcade.MOUSE_BUTTON_RIGHT:  
7         print("Right mouse button pressed at", x, y)
```

(More info about keys)

The Window class

Other methods

Other methods

- `activate()`.
- `center_window()`.
- `close()`.
- `get_location()`.
- `maximize()` and `minimize()`.
- `set_fullscreen(fullscreen: bool = True)`.
- `set_location(x, y)`.
- `set_viewport(left: float, right: float, bottom: float, top: float)`.
Set the coordinates we can see

The Window class

Using a game controller (I)

First, get the controllers with `arcade.get_joysticks()`

Get game controllers

```
1 joysticks = arcade.get_joysticks()
2
3 if joysticks:
4     self.joystick = joysticks[0]
5     self.joystick.open()
6 else:
7     print("There are no joysticks.")
8     self.joystick = None
```

Read game controller value

```
1 def update(self, delta_time):
2     # Update the position according to the game controller
3     if self.joystick:
4         print(self.joystick.x, self.joystick.y)
```

The Window class

Using a game controller (II)

Read game controller value

```
1 def update(self, delta_time):  
2     # Update the position according to the game controller  
3     if self.joystick:  
4         print(self.joystick.x, self.joystick.y)
```



Centered (0, 0)



Down (0, 1)



Down-left (-1, 1)

(Source)



Up (0, -1)

(Interesting example)

Life-cycle management

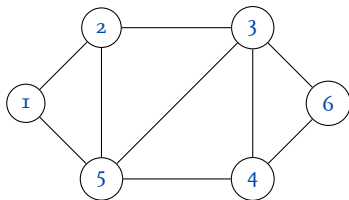
Graphs

Graph: Data structure with **nodes** and **edges**

- Widely used in programming, AI and videogames
- Huge number of applications

Central role in **path planning**

- (Video)
- Navigation mesh

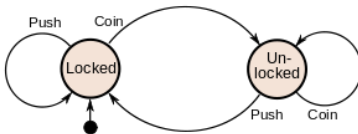


Life-cycle management

Finite States Machines (I)

Finite-State Machine (FSM)

A graph whose nodes represent states, usually associated with behaviours



(Source)

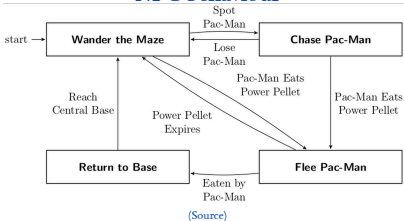
Life-cycle management

Finite States Machines (II)

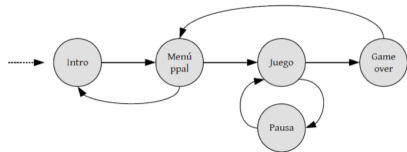
FSMs have many applications

- Central role in Theory of Computation
- Good to model behaviours ... such as a NPC or an entire videogame

NPC behaviour



Game life-cycle (also state)



Life-cycle management

The View class (I)

Videogames use several screens, or states

- Start screens
- Instruction screens
- Game over screens
- Pause screens

Arcade provides the `View` class

- Very much like the `Window` class
- It has the `on_draw()` and `on_update()` methods



(Source)

Life-cycle management

The View class (II)

Our class must derive from `arcade.View`

```
class MyGame(arcade.Window):
```



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

The view does not control the window size, so

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



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

Life-cycle management

The View class (III)

Finally, we need to create a window, a view and show that view

```
def main():
    """ Main function """

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

Life-cycle management

The View class: Example (I)

GameOverView view

```

1  class GameOverView(arcade.View):
2      """ View to show when game is over """
3
4      [...]
5
6      def on_draw(self):
7          """ Draw this view """
8          self.clear()
9          self.texture.draw_sized(SCREEN_WIDTH / 2, SCREEN_HEIGHT
10                                 / 2, SCREEN_WIDTH, SCREEN_HEIGHT)
11
12     def on_mouse_press(self, _x, _y, _button, _modifiers):
13         """ If the user presses the mouse button, re-start the
14             game. """
15         game_view = GameView()
16         game_view.setup()
17         self.window.show_view(game_view)

```

Life-cycle management

The View class: Example (II)

Game view

```

1  def on_update(self, delta_time):
2      """ Movement and game logic """
3
4      [ ... ]
5
6      # Check length of coin list. If it is zero, flip to the
7      # game over view.
8      if len(self.coin_list) == 0:
9          view = GameOverView()
10         self.window.show_view(view)

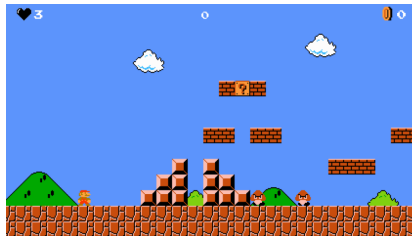
```

Sprites

Introduction

Sprite

A sprite is a 2D image used in videogames



Sprites

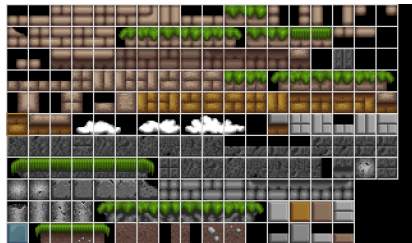
Spritesheets

A videogame contains many sprites

- Difficult maintenance
- Solution: Spritesheets

Advantages

- One file contains many sprites
- Less I/O operations \Rightarrow Better performance
- Less memory consumption



Sprites

Data formats (I)

In general, any data can be stored in three forms

- Not compressed
- Compressed with loss
- Compressed without loss

| | Image format | Sound format | Binary data |
|-------------------------|--------------|-----------------|---------------------|
| Not compressed | BMP | WAV | ZIP, bzip, rar, ... |
| Compressed with loss | JPG | MP ₃ | |
| Compressed without loss | PNG, GIF | - | |

Sprites

Data formats (II)

Attending to what information is stored in image format, there are two types of image formats:

- Bitmap: stores each pixel
 - Scales bad
 - Formats: JPG, PNG, BMP, GIF
- Vectorial: stores coordinates
 - Scales well
 - Not supported by Arcade
 - Formats: SVG, EPS

Many open assets for your games!

- (Kenney)

Sprites

The Sprite class (I)

You will need to provide a **path** to the file

- **Absolute path:** Starts from the root directory
 - Example (Windows):
`c:\\Users\\atreides\\Desktop\\mygame\\assets\\sprites\\mario.png`
 - Example (Linux):
`/home/atreides/mygame/assets/sprites/mario.png`
- **Relative path:** Relative to the project's directory
 - Example (Windows): `assets\\sprites\\mario.png`
 - Example (linux): `assets/sprites/mario.png`

Always use relative paths in your projects!!!

Sprites

The Sprite class (II)

Sprites are a fundamental concept in Arcade

Creating a sprite

```
character = arcade.Sprite('images/character.png')
```

Placing a sprite

```
character.center_x = 300  
character.center_y = 200
```

Sprite

The Sprite class (III)

(Reference documentation)

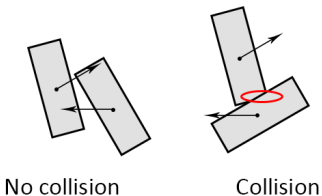
Constructor

```
class arcade.Sprite(  
    filename: Optional[str] = None,  
    scale: float = 1,  
    image_x: float = 0, # offset within sprite sheet  
    image_y: float = 0, # offset within sprite sheet  
    image_width: float = 0,  
    image_height: float = 0,  
    center_x: float = 0,  
    center_y: float = 0,  
    repeat_count_x: int = 1,  
    repeat_count_y: int = 1,  
    flipped_horizontally: bool = False,  
    flipped_vertically: bool = False,  
    flipped_diagonally: bool = False,  
    hit_box_algorithm: Optional[str] = 'Simple',  
    angle: float = 0)
```

Sprite

The Sprite class (IV)

Sprites in Arcade implement **collision detection** and **handling**



Three values for `hit_box_algorithm`: 'None', 'Simple' and 'Detailed'



'None'



'Simple'



'Detailed'

Sprite

The Sprite class (V)

arcade.Sprite

Methods

- `on_update(delta_time: float = 0.016).`
- `draw().`
- `append_texture(Texture: arcade.texture.Texture.`
Appends a new texture (image)
- `set_texture(texture_no: int).`
- `update_animation(delta_time: float = 0.016)).`
- `set_position(center_x: float, center_y: float).`
- `turn_left(theta: float = 90.0)`
- `turn_right(theta: float = 90.0)`
- `stop()`

Attributes

- `alpha: int.`
- `angle: float.`
- `bottom: float` and `bottom: float.`
- `center_x: float` and `center_y: float.`
Center of the sprite
- `change_x: float` and `change_y: float.`
Velocity in X and Y
- `height: float.`
- `visible: bool.`

(Reference documentation)

Sprites

The Sprite class: Examples

```
1 # Make the sprite invisible
2 sprite.visible = False
3
4 # Change back to visible
5 sprite.visible = True
6
7 # Toggle visible
8 sprite.visible = not sprite.visible
```

Sprites

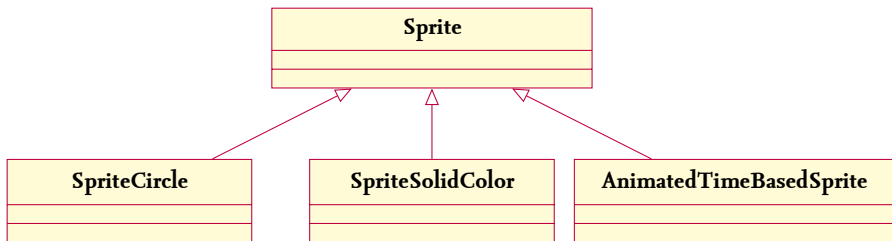
Collision detection

Collision detection methods

- `collides_with_point(point: Union[Tuple[float, float], List[float]])`
→ `bool`
- `draw_hit_box()`.
- `collides_with_list(sprite_list: SpriteList)` → `bool`

Sprite

Other classes



- ([SpriteCircle documentation](#))
- ([SpriteSolidColor documentation](#))
- ([SpriteAnimatedTimeBasedSprite documentation](#))

Sprites

Sprite lists (I)

Arcade stores sprites in lists

```
wall = arcade.Sprite('images/boxCrate.png')  
wall.center_x = 300  
wall.center_y = 300  
wall_list = arcade.SpriteList()  
wall_list.append(wall)
```

Lists can be manipulated as a whole

```
wall_list.draw()
```

And sprites can be removed from a list

```
wall.remove_from_sprite_lists()
```

Sprites

Sprite lists (II)

Lists in Arcade also implement collision detection

```
hit_list =  
arcade.check_for_collision_with_list(player_sprite,  
coin_list)
```

Functional example in (example)

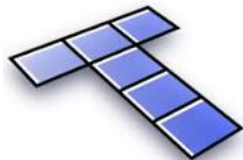
Sprites

Locating sprites

Locating sprites in the game is a tough work

- Closely related to **level design**
- There are tools that ease this task

(Tiled Map Editor)



Resources management (I)



Arcade comes along a collection of built-in resources

- Good for testing
- No need of files in disk
- Images, music, sounds, tiled maps and video

The path is something like `'resources:/path/to/resource'`

```
arcade.Sprite("resources:images/items/coinGold.png")
```

(List of resources)

(Arcade source code)

Resources management (II)

Adding new resources handlers

```

1 arcade.resources.add_resource_handle("characters", "resources /
   characters")
2 arcade.resources.add_resource_handle("maps", "resources / maps")
3 arcade.resources.add_resource_handle("data", "resources / data")
4 arcade.resources.add_resource_handle("sounds", "resources / sounds
   ")
5 arcade.resources.add_resource_handle("misc", "resources / misc")

```

Using resources handlers

```

1 self.player_sprite = PlayerSprite(":characters:Female/Female
   r8-4.png")

```