

Practical Session 2: Game Structure and Sprites in Pygame

Overview

In this session we are going to look at some basic game mechanics, including: Game Structure, Sprites; User input; Backgrounds; and Collision detection. You will need to download the “Scratch/Pygame Resources” and “Pygame Scripts” from the class materials wiki page.

Task 2.1

First of all we need to get Pygame up and running. Copy the Pygame Scripts and resources all into the same folder; rename “Framework.py” to “Spacegame.py”. This code has been structured to give a basic framework for a simple game, but feel free to use your own framework. Comments are provided to help you understand the code, but the tutors are there to help if there is anything you are unsure of. Let’s think about a typical game structure:

Initialisation

When the game starts up, initialise variables, load images, set starting positions for Sprites etc.

It is important when creating games that all your users have a similar experience, unless you intentionally create dynamic environments that change with each play, but this is a slightly more advanced topic. It is also dangerous to proceed with altering values held by variables without first initialising them as this can yield unpredictable results.

Updating (Game loop)

Typically, the game loop is processed repeatedly, updating all sprites, user input, game events etc. according to the desired frame rate. In our example, this is accomplished by using a While True loop which processes the event queue and calls the tick() and render() functions once per frame.

Rendering

Part of the game loop, after updating all the sprites positions and other game elements, we render to the screen. Pygame uses double-buffering, so all images are rendered to a back-buffer before being presented to the screen. If you are unfamiliar with the concept of double-buffering, please ask a tutor.

Object Orientation

I have used minimal object orientation in this basic framework to keep things simple, although I will use it more in following weeks. The basis of object orientation is that objects are responsible for their own data and functionality. Don't worry if this is an unfamiliar topic to you, as it will be introduced slowly, and it will be covered in your programming modules.

Task 2.2

"Sprite" is generally used to describe a graphical object that can be moved around a game surface. We want to create a Sprite using the "ship.png" image. A sprite class has been created with minimal required functionality for you. There are a few steps to follow:

1. Create a variable for the ship sprite.
2. Create the sprite by specifying the image filename and start position
3. Draw the sprite each frame

Add the following code to your script, using the comments as indications of where each part of code should be placed:

Spacegame.py

```
# global variables section
ship = None

# during initialisation, after the screen has been created
# remember to use the global variables
ship = Sprite("ship.png", size[0]/2, size[1]/2) # positions at centre of screen

# in the render() function
ship.render(screen)
```

If all goes well, running the script now should show the spaceship in the centre of the screen. If something goes wrong, have a look at your code, think about what could be wrong, and try to fix it. Only if you have made an attempt to fix the problem should you ask a tutor for help. Asking for help when something goes wrong, without attempting to fix the problem yourself won't help you learn.

To add user input for the sprite, again there are a few steps:

1. Check for key presses
2. Move or rotate the sprite as appropriate

3. Update the Sprite

Add the following code to your script, using the comments as indications of where each part of code should be placed:

Spacegame.py

```
global ship

angle = 0
move = 0

keystate = pygame.key.get_pressed()
# rotate the ship
if((keystate[K_LEFT])):
    angle += 1
if((keystate[K_RIGHT])):
    angle -= 1
# move the ship
if((keystate[K_UP])):
    move += 1
if((keystate[K_DOWN])):
    move -= 1

ship.moveRel(move)
ship.rotate(angle)
ship.tick()
```

You should now be able to move the ship about the screen. Again, make an attempt to fix any problems yourself before asking a tutor, but do ask if you can't get it working.

Task 2.3

A spaceship moving about on a plain white background doesn't look very good, so let's add in a background. The file "starfield.png" in the downloaded resources should do for this. We don't need to use a Sprite for the background, so add the following code using comments as guidelines:

Spacegame.py

```
# global variables section
```

```
starfield = None
```

```
# during initialisation, after the screen has been created
```

```
# remember to use the global variables
```

```
starfield = pygame.image.load("starfield.png")
```

```
# in the render() function
```

```
screen.blit(starfield, [0, 0])
```

Be careful in which order you render images to the screen, or the background could be rendered on top of the spaceship.

Task 2.4

The pilot of the spaceship has nothing to do, so let's give him some work. His task is to pick-up a strange looking object represented by the image "collect.png". You should create a new sprite named "pickup" using the image file.

After updating both sprites, you should check to see if they have collided. To do this, use the *isColliding* method of the sprite class, passing the other sprite as an argument. If there has been a collision, a simple response would be to set the pickup as inactive (*pickup.active = False*).

Homework Task 2.1

Have a go at adding some of the following functionality to the project you have been working on today, either in the labs if you have time, or at home before next week's practical session:

1. Create a score variable, initially set to 0, that is incremented when a Pick-up is collected. The score should be displayed for the user to see.
2. When a Pick-up is collected, show it again at a random position on the screen.
3. Allow the spaceship to 'wrap' around the screen i.e. goes out left edge, comes in right and vice versa.
4. Add inertia to the spaceship so it continues to move for a short time after the user releases the forward key. Tip: Try creating a speed variable
5. When the space key is pressed, the spaceship should fire a bullet that moves off in the direction it is currently facing.