COMP1551: Sprites and User Interaction

Preparation

1. In a browser, visit http://www.pygame.org/docs/. Keep this page open, as you will need to refer to it later.

2. Create a new directory for this week's lab work and download `pygamelab02.zip` into it. You will find this Zip archive in the VLE. Unpack the archive into your directory using the unzip command.

3. Make sure you use the command p3 in any terminal windows that you use in this lab session. before running python. This is to ensure you are using python 3 and not 2.

Important note: type in the code fragments that appear in these instructions; do not attempt to copy and paste from the PDF document into your text editor.

1 A Simple Pygame Sprite

1. You have been given a file called `sprites.py`. Open this file in an editor and examine it. The sprites module defines a function `draw_disc`, which returns an image containing a coloured disc. It also defines a sprite class called `Disc`, based on the superclass `Sprite` defined in the `pygame.sprite` module. Study the function and the class constructor carefully.

Copy `main.py` to a new file, `ex1.py`. Open `ex1.py` in a text editor and study it. The code should be familiar from last week's lab. Run `ex1.py` and you should see a Pygame window with a grey background appear.

2. At the position in `ex1.py` indicated by the first 'TODO' comment, add code to create a `Disc` sprite. Use a diameter of 50 pixels and the variable `red` to specify the colour.

Beneath this line of code, add another line that creates a sprite group containing the new sprite:
`group = pygame.sprite.Group(disc)`

3. Find the second 'TO DO' comment and add beneath it three lines of code to clear, update and redraw sprites in the group:

```
        group.clear(screen, background)
        group.update()
        group.draw(screen)
```

Now run `ex1.py` again. This time, you should see a red disc appear in the center of the window.

4. The sprite doesn't move because we haven't yet overridden its update method. To rectify this, edit `sprites.py` and define an update method below the constructor of the `Disc` class. Your method will need to update the `rect` instance variable of a `Disc` with

```
        self.rect.centerx += self.vy
```

plus a similar line for the `centery` property of `self.rect`. You will also need to test whether the `rect` instance variable is moving off any of the screen edges and flip the sign of `self.vx` or `self.vy` accordingly (see Exercise 5 from last week's lab for how to do this).

Run `ex1.py` again and this time the disc should move, bouncing off the edges of the screen when it touches them.

2 Managing Multiple Sprites

The real benefit of using sprite groups is felt when there is more than one sprite in a program.

1. Copy `ex1.py` to a new file, `ex2.py`. Edit this file and add code that creates a few additional Disc sprites. Give these sprites different diameters, colours and initial velocities. (For the latter, add a keyword argument like `velocity=(6, 3)`.)

2. Add these new sprite objects to the sprite group created earlier. This is all you need to do; there is no need to modify the code that clears, updates and redraws the contents of the sprite group.

3. Run `ex2.py`. You should now have several Disc sprites moving independently on the screen.

Depending on how many sprites you have created, you may notice that sprites are not layered on top of each other in the same order as they were added to the group. If ordering matters to you, just use `pygame.sprite.OrderedUpdates` instead of `pygame.sprite.Group`.

3 Using easypg Sprites

The `easypg` package provides two classes to simplify sprite creation in cases where sprite images are stored in a file; `SimpleSprite` can be used when the sprite consists of a single static image; Sprite can be used for animated sprites, represented by one or more sequences of images.

1. Start a Python interpreter, then enter the following commands at the >>> prompt:

```
from easypg.sprites import SimpleSprite
help(SimpleSprite)
```

   Read the documentation provided.

2. Add the `import` statement given above to the file `sprites.py`, just after the import statement for the `pygame` package.

3. At the end of sprites.py, define a new class called `Monster`, inheriting from `SimpleSprite`.
   Add the following constructor definition to your class:

```
    def __init__(self, screen, velocity=(3, 3)):
        super().__init__(screen, 'monster.png')
        self.vx, self.vy = velocity
```

4. Define a move method in your Monster class. Copy into it the first two lines of code from the update method of `Disc` (the lines that modify the `centerx` and `centery` properties of `self.rect`).

5. Define a check_bounds method in your Monster class. Copy into it the other lines of code from the update method of `Disc` (the ones that check whether the sprite is going off the edge of the screen and adjust `self.vx` and `self.vy` accordingly).

6. Copy `ex1.py` to a new file `ex3.py`. Edit `ex3.py` and modify it so that it creates a `Monster` object instead of a `Disc` object. Run `ex3.py` and there should now be a monster bouncing around the screen, instead of a disc.

7. Make the following changes to `sprites.py`:

   - Change the import statement so that it imports `Sprite` from `easypg.sprites`
   - Change the `Monster` class so that it inherits from `Sprite` rather than `SimpleSprite`
   - Add the following class variable definition just inside the definition of `Monster`: `images = {}`
   - Change the first line of the `Monster` constructor to

     ```
         super().__init__(screen, 'monster.zip', state='angry')
     ```

   Run the program again. The monster should now be animated.

4 User Interaction

The sprite classes developed thus far have all moved independently of user interaction. How do we create a sprite that is controlled by the user?

1. Edit `sprites.py` once again. Create another class called `Player` by copying and modifying the code from `Monster`. Aside from renaming the class, you will need to change the source of sprite images from `monster.zip` to `player.zip`.

   Next, replace the body of the `move` method with the following:

   ```
   keys = pygame.key.get_pressed()
   if keys[pygame.K_LEFT]:
       self.rect.centerx -= self.vx
   if keys[pygame.K_RIGHT]:
       self.rect.centerx += self.vx
   if keys[pygame.K_UP]:
       self.rect.centery -= self.vy
   if keys[pygame.K_DOWN]:
       self.rect.centery += self.vy
   ```

This checks for key presses on every update of the sprite, adjusting its position in accordance with which arrow keys are currently pressed.

You will also need to replace the body of the `check_bounds` with different code. In this case, we want to ensure that if the player moves their sprite off the right of screen, the sprite will reappear on the left and vice versa. We want similar behaviour for the top and bottom edges of the screen, too. This can be achieved with the following code:

```
    width, height = self.screen.get_size()
    if self.rect.centerx < 0:
        self.rect.centerx = width - 1
    elif self.rect.centerx >= width:
        self.rect.centerx = 0
    if self.rect.centery < 0:
        self.rect.centery = height - 1
    elif self.rect.centery >= height:
        self.rect.centery = 0
```

2. Copy `ex3.py` to a new file `ex4.py`. Edit this new file and write code to create a `Player` sprite and add it to the sprite group. Run `ex4.py` and you should now see two animated sprites: a red one moving independently and a blue one that moves in response to key presses.

5 Collision Detection

Suppose that you want the computer-controlled sprites in your game to be destroyed when the player- controlled sprite touches them. This is fairly easy to implement in Pygame, thanks to its comprehensive support for collision detection.

1. Start by copying `ex4.py` to a new file `ex5.py`. Edit this new file and modify it so that it creates three or four different `Monster` sprites. Use the `position` and `velocity` keyword arguments to given each of them different starting positions and velocities.

2. To make destruction of the computer-controlled sprites simpler, it is a good idea to put them in a separate sprite group from the player-controlled sprite. Do this now. Put the Monster sprites in a group called monsters and the Player sprite in another group called other.

3. Modify the event loop of `ex5.py` so that it calls the clear method on both sprite groups, then the update method on both groups, then the draw method on both groups. Run `ex5.py` and make sure that it behaves as expected.

4. The collision detection code itself is very simple. Just add the following to the event loop in `ex5.py`, after the code that updates the sprites and before the call to `pygame.display.flip`:

```
pygame.sprite.spritecollide(player, monsters, True)
```

This tests for collision between `player` and any of the sprites in group `monsters`. Passing in `True` as the third argument indicates that colliding sprites should be removed from the group. The function call will return a list containing references to the colliding sprites, or an empty list if no collisions are detected. We have no need of such a list here, so you can ignore it.

5. Run `ex5.py` once more and try steering the player's sprite with the arrow keys so that it collides with and destroys the monsters.