Python: Intro to Graphics

Computer Science ICS20

The pygame module

In order to be able to draw in a window we will be using the pygame module. Please make sure that you have this installed by importing the pygame module as follows:

```
import pygame
```

If you get an error message indicating that you don't have pygame installed please see your teacher.

Defining a Function

The basic structure of all pygame programs is the following:

```
import pygame
pygame.init()     # initializes the graphics module
window = pygame.display.set_mode((800,600))    # define window size
pygame.display.set_caption('Intro to PyGame')  # title of program that
                                                    # appears on window
                                                    # frame
clock = pygame.time.Clock()     # used to track time within the game (FPS)
quit = False
while not quit:                             # main program loop
    for event in pygame.event.get():        # check if there were any events
        if event.type == pygame.QUIT:       # check if user clicked the upper
            quit = True                     # right quit button

      # your code that draws to the window goes here


    pygame.display.update()         # refresh your display
    clock.tick(60)                  # wait a certain amount of time that
                                    # ensures a frame rate of 60 fps
pygame.quit()                       # shutdown module
```

Try typing this in your favourite IDE and see if it runs. You should see a black window pop up with the title that you set using the set_caption() function. ("Intro to PyGame" in our example above)

If you want to change the background colour, you can use the **fill()** function and pass it a new colour, but first you must define a new colour.

Colours

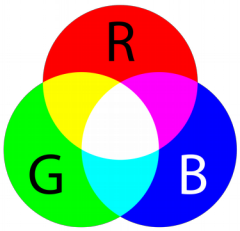
Defining colours on a computer is different from creating colours using paint. You still use 3 primary colours but they are RED, GREEN and BLUE instead of the traditional RED, YELLOW and BLUE used with paints. Can you think of why the primary colours are different on computers?

In order to create a colour we mix different amounts of the 3 primary colours. The amount is indicated by a number between 0 and 255, with 0 being the absence of all colour and 255 being the most intense colour. So to create a RED colour we can type the following:

```
red = [255, 0, 0]
```

The above line creates a red colour. Notice that the colour is specified with 3 items in a list, with each item representing the amount of RED, GREEN, BLUE in that order. The first amount is 255, which is the largest value we can have for RED, followed by zeros for the GREEN and BLUE components indicating that there is no green or blue in our mixture.

Here are a few other colours we can define:




```
green = [0, 255, 0]

blue = [0, 0, 255]

black = [0, 0, 0]

white = [255, 255, 255]
```

Notice that black is the absence of all colours and white has equal amounts of each primary colour in maximum amounts. Can you make other colours by mixing different amounts of R,G,B?

Let's try changing the background colour of our window to white:

```
gameDisplay.fill(white)
```

Notice that the fill function is a part of the gameDisplay object. This is because we can have multiple windows and we need to be able to tell pygame which window's background is to be changed.

Drawing Basic Shapes

Drawing in pygame is accomplished by using various drawing functions within the **pygame.draw** sub module. Here are a list of the more common drawing functions:

pygame.draw.line(screen, color, start_pos, end_pos, width=1)

- draws a line, connecting the points from start_pos to end_pos
- thickness is the thickness of the line (in pixels).
- Example: `pygame.draw.line(screen, black, (100,100), (150,200), 1)`

pygame.draw.lines(screen, color, closed, pointlist, thickness)

- draws a series of lines, connecting the points specified in pointlist
- pointlist is a list of tuples, specifying a series of points, e.g. to draw a V you might use [(100,100), (150,200), (200,100)], with closed = False
- closed should be either True or False, indicating whether to connect the last point back to the first
- thickness is the thickness of the line (in pixels).

- Example: `pygame.draw.lines(screen, black, False, [(100,100), (150,200), (200,100)], 1)`

`pygame.draw.rect``(screen, color, (x,y,width,height), thickness)`

- draws a rectangle
- (x,y,width,height) is a Python tuple
- x,y are the coordinates of the upper left hand corner
- width, height are the width and height of the rectangle
- thickness is the thickness of the line. If it is zero, the rectangle is filled

`pygame.draw.circle``(screen, color, (x,y), radius, thickness)`

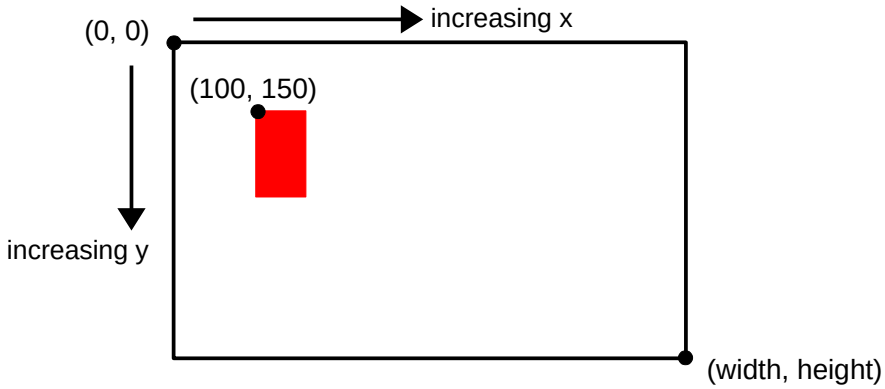
- draws a circle
- (x,y) is a Python tuple for the center, radius is the radius
- thickness is the thickness of the line. If it is zero, the rectangle is filled

`pygame.draw.arc``(screen, color, (x,y,width,height), start_angle, stop_angle, thickness)`

- draws an arc (portion of an ellipse)
- (x,y,height,width) are the coordinates of a rectangle that the arc would fit inside if it were drawn all the way around
- if height and width are equal, then the rectangle is a square, and the arc will be a portion of a circle
- start_angle and stop_angle are the angle on the unit circle in radians (not degrees) where the arc stops and starts

So, for example if we want to draw a rectangle of dimensions 100 x 150 at a certain location on the screen, we need to understand how pygame specifies the location of points on the screen. Normally points are objects without any dimensions; they are infinitely small objects. However, although the screen may seem to be made from infinitely small points, it is actually made up of finitely sized picture elements or **pixels.** So every screen's dimensions can be specified by its horizontal resolution and its vertical resolution in pixels. In the previous example, the window size was defined to be 800 x 600 pixels.

When we draw something on the screen we are essentially changing the colour of certain pixels but we need to be able to tell pygame which pixels to alter. Pygame treats the entire surface similar to a Cartesian plane, only using integers. So the address of each pixel is specified by a coordinate pair (x, y). The origin of the plane is at the top left corner of the screen with increasing x values to the right and with increasing y values downwards.



If we want to draw a rectangle using the rect() function, we need to specify the location and dimensions as a **tuple**. A tuple is the same as a list except that it is immutable (cannot be changed) and round brackets are used instead of square brackets. The following is a tuple:

(213, 450)

So, to draw a red rectangle of dimensions 100 x 150 at location (200, 100) on the screen we would call the rect function as follows:

```
pygame.draw.rect(window, red, (100, 150, 200, 100))
```

The location of the rectangle is specified at the top left corner. Refer to the picture above to see this. Note that the first parameter which indicates which sreen to draw to is the value returned by the pygame.display.set_mode() function.

Exercises

1. Write a Python program that draws a single green 200x200 square in the centre of the window.

2. Write a Python program that draws three differently coloured 100x100 squares in a triangle arrangement like this:

3. Write a Python function that will draw a triangle of squares as in question 2. The function signature is

```
drawTriSquare(screen, location, square_size, height, colour)
```

where: *screen* is the screen or window that is used to draw into

location is a tuple representing the (x, y) location of the top left corner of the top square.

square_size is the dimension of each square in pixels

height is the height of the triangle or number of rows of squares (height = 2 in the above drawing)

colour is the RGB colour of the squares

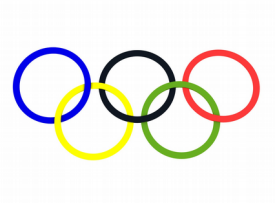
If the colour is not specified, the function should randomly choose a different colour for each square.

4. Starting with a black square, draw a row of alternating black and white coloured squares of size 50x50 pixels across the top of the screen.

5. Extend question 3 to draw another row of black and white squares immediately beneath the first row but this time starting with a white square.

6. Continue drawing drawing rows of black and white squares until you have filled the entire screen. Will your program work if you opened a window of different dimensions? If not, fix your program so that it does. You should then have a program that draws a checker board pattern on the window.

7. Write a Python program to draw the Olympics 5-ring logo as shown below:



8. Write a Python program to continuously draw 250 lines all starting from the same random point on the screen and each ending at a different random point. Each line should be randomly coloured from a list of colours defined as

```
[black, white, green, red, blue, yellow, cyan, magenta]
```

You should slow down the refresh rate to 1 frame per second (fps) to properly see the lines.

9. Write a Python program that plots a quadratic function of the form $y = a(x - h)^2 + k, A \leq x \leq B$and where (h, k) is the vertex of the parabola. Since this is a more complicated program it is a good idea to break it down into smaller problems using functions. Here is a suggestion of how the program could be broken down:

- quadratic(x, a, h, k) – returns the value of the quadratic evaluated at x
- drawGrid(screen, num_rows, num_cols) – draws a grid on the screen with the given number of rows and the given number of columns so that the entire screen is covered.
- drawAxis(screen, x, y) – draw the axis on the given screen with a slightly thicker line, where (x, y) is the location of the origin.
- drawFunction(screen, colour, A, B) – draws the quadratic function on the screen in the requested colour for all x values between A and B.
- Plus the standard pygame template code that sets up pygame for drawing to a screen and to call the above functions.