Week 5: Snake / PyGame

Introduction

This worksheet will guide you through how to make your own version of **snake** from scratch, allowing you to build up your Python skills and create something quite fun!

During this you will use a very popular Python package **pygame**. This provides us with the resources to display a screen on our computer where we can colour pixels and constantly update it. Pygame has a lot more to offer past this worksheet so if you are interested there are a lot of cool projects you can do with it!

Getting Started

The only preparation needed for this worksheet is to download pygame - just run the following command in terminal:

```
pip3 install pygame
```

TASK 0: The Template

Let's start by having a look at the template code given.

Screen Size

The first 3 properties of the game class are:

1. Width

2. Height
3. Scale

These will represent the size of your game window displayed on your screen.
As Snake is a pixelated game, we are going to work with our own grid of size (Width, Height). The scale represents the size of each cell in our grid in terms of pixels on our screen. So if we are working with a 20x20 grid and a scale of 30, each cell in our grid will have 30x30 pixels in, meaning our overall game screen is 600x600 pixels.

Colours

The next properties are the colours which we will work with in our game:

1. BackgroundColour
2. TextColour
3. FoodColour
4. SnakeColour

Digitally, colours are often stored in **RGB format** ranging from **0-255**. So the corresponding 3 channels represent how much red, green and blue you want (respectively) where 0 means that colour isn't present and 255 means we want all of that colour. You can create any colour using a mix of colours and intensities.

For example, we can create orange using mixes of red and green, e.g (255,150,0). Black is represented as (0,0,0) and white is (255,255,255).

You do not need to worry about the rest of the properties for now, you will come across these later on.

PyGame

When working with a live game, there are 2 things we must do:

1. Initialise the game
2. Create a game loop in which our display is updated

As you can see in the template code, the game is initialised inside our initialisation of the game class - this is all set-up for you. Our screen is saved as a property called 'self.Display' so we can access it wherever inside the class!

You can observe the game loop inside the 'Play' function. This consistently monitors to see if any buttons are pressed on your keyboard or if you quit the game - you are free to decide what happens when any event occurs. Currently, we stop the game loop if you quit the window and, later on, we will be interested in choosing actions when the keyboard arrows are pressed (this code is commented out for now).

Also, inside the game loop we call on the 'DisplayFrame' function, which has been provided for you. This is where we will tell pygame which pixels to colour - note that this must be done after we reset from the previous frame and before we update the screen.

TO DO:

1. Choose your size for your screen and pixels (you can test this out by running the template code)
2. Choose colours for your objects (you won't be able to test these colours until we start displaying the objects on the screen)
3. Uncomment the two lines of code at the end of the template

TASK 1: Colouring Cells

To colour a cell we can use the code below. It takes a coordinate (as a list, i.e [1,3]) in terms of our grid system and a colour in RGB format.

```python
def ColourCell(self, Position, Colour):
        # Scale up the coordinates so we get the true pixel coordinates on our screen
        X,Y = Position
        X = X * self.Scale
        Y = Y * self.Scale
        # Draw a rectangle covering all the screen pixels for our 'cell'
        pygame.draw.rect(
            self.Display,
            Colour,
            [X, Y, self.Scale, self.Scale])
```

TO DO:

1. Copy and paste this function into the template function given to you on line 56.
2. Try calling this function within DisplayFrame - where do you need to call it within the function? Which way round are the axes for pygame?

TASK 2: The Snake Class

In your template code you have a Snake class with the following properties:

- Head: a coordinate (list of size 2). Either start with [0,0] or add an input and make it the middle.

- Body: a list of coordinates (lists of size 2). The body will be a list of ALL coordinates that the snake embodies (**including the head**). The head will be in position 0 and as the snake grows, these coordinates will be appended onto the end.
- PreviousCell: a coordinate (list of size 2). This will be updated every time the snake moves to store the previous position of the end of the snake.

TO DO:

1. Choose where you want your snake to start and edit your self.Head property.
2. Before your game starts in your play function, create an instance of your snake.
3. Edit your DisplayFrame function and use the ColourCell function so that your snake can be seen (you will want to make it an input to the function). Note that your snake might have a length larger than 1 and you will want to colour all parts of it.

TASK 3: Moving

Now, it's time to make our snake move. We will first write a function to make our Snake move a singular step (i.e only move one block) in a specified direction and then connect this up to the arrow keys through our game loop.

TO DO:

1. Copy and paste the following into your 'Move' function within your Snake class. This function will only move the snake if we input Direction = "Left", Direction = "Up" etc. This function will move your snake one block at a time in the specified direction, making the body follow the head.

```python
    def Move(self, Direction):
        # Making copies to work with instead
        Head = self.Head.copy()
        Body = self.Body.copy()

        # Moving the position of the head
        if Direction == "Left":
            # Move the x-coordinate
            Head[0] += -1
        if Direction == "Right":
            # Move the x-coordinate
            Head[0] += 1
        if Direction == "Up":
            # Move the y-coordinate
            Head[1] += -1
        if Direction == "Down":
            # Move the y-coordinate
            Head[1] += 1

        # Save the last block position for later
        self.Previous = Body[-1]

        # Delete the last element from our body
        n = len(Body)
        Body = Body[0:n-1]

        # Add the head to the front of the body
        Body = [Head] + Body

        # Update the properties
        self.Body = Body
        self.Head = Head

```

2. Uncomment the code in the game loop in 'Play'.
3. Add code to call your move function when the corresponding button is pressed so that your Snake steps one position each time the arrows are pressed.

TASK 4: Time and Speed

We now want to add a speed to our snake so that when we press left once it constantly moves left, instead of having to press the button for each step.

TO DO:

1. Change your game loop so that the 'Move' function is called with each iteration of the loop. This will ensure that the Snake is constantly moving - the buttons should only change the direction of the Snake.
2. Change your code so that the snake only starts moving when the first arrow button is pressed - using a Boolean variable should work for this!
3. Once this has been completed, you will notice that your snake moves very quickly! We will use PyGame's own clock system to slow this down. Initialise this using the following code (you may do this in the __init__ function or before your game loop):

```python
    self.Clock = pygame.time.Clock()
```

4. In your game class properties, set the 'GameSpeed' property to a value (I personally like 15)
5. Call the following code in your game loop:

```python
    self.Clock.tick(self.GameSpeed)
```

6. Play around to find the speed that you like! You may also want to add in later code that makes the game speed up as you get longer.

TASK 5: Food

Now is time to incorporate the food into our game. We want to have food placed randomly on our grid and also for it to disappear and for new food to appear when it is eaten.

TO DO:

1. Complete the 'NewFood' function. This function should choose a random cell in our grid and update our FoodPosition property to store the coordinate. *Note: The 'random' package is already imported at the top of the code.*
2. Call upon 'NewFood' in initialisation of the game so we can start off with some food.
3. Edit your DisplayFrame function to show the food on our screen.
4. Complete the function 'EatenFood' in the Snake class. This function should return True if the snake has just eaten the food and False otherwise. You will only need to observe the positioning of the head for this (not the whole body).
5. Edit your game loop so that we constantly check to see if our Snake has eaten the food with each move. If it has, we:
   (a) Update our food positioning
   (b) Add a block onto the end of our snake (Hint: use the PreviousCell property within our snake by appending it onto the end of the Snake's body)
   (c) Add one to our score property

TASK 6: Game-Over

Currently, our snake is able to go off the screen and is able to cross over itself - two things we do not want it to do.

TO DO:

1. Complete the function 'OffScreen' in the Snake class. This function will return True if the head of the snake has gone outside of our viewing grid and False otherwise.
2. Complete the 'EatenSelf' function in the Snake class. This function will check to see if it has eaten itself, returning a Boolean value. Again, consider the head of the snake and where you wouldn't want to find it.
3. Copy and paste the following code into your main game loop (you may need to change variable names so it matches your code). This bit of code 'pauses' your game when you lose - if you press any key it restarts to a new game.

```
if Snake.OffScreen(self.Width,self.Height) or Snake.EatenSelf():
            GamePaused = True
            while GamePaused:
                self.DisplayFrame(Snake)
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        GamePaused = False
                        GameOver = True
                    if event.type == pygame.KEYDOWN:
                        NewGame = GameClass()
                        NewGame.Play()
                        return
```

TASK 7: Text

Hopefully now your game play fully works! All that is left to do is to add some text to our game to show our score and also when the game is over.

A function has been given to you to display text on the screen. It has an input of the Message you which to display and the point (in terms of our grid, not screen pixels). The font and size has been set for you, but please feel free to change these!

TO DO:

1. Edit your 'DisplayFrame' function by adding an input (which will be the text you wish to display) and call on the 'DisplayText' function, deciding on a position for your text too.
2. Edit your game loop (where you call 'DisplayFrame') so that the score of the game is displayed.
3. Edit the code in the loop for when the game is over so that the text displayed is the end score plus some text saying "Press any key to restart".
4. Enjoy your completed game!

Topic Suggestions

If you have any topics which you would like covered in these sessions then feel free to let me know by clicking here.
This may be content that isn't covered within your units which you would find useful or a particular topic within your current units that you feel you would benefit from more help with!