Introduction to Computer Programming

Exercises - Week 10: Pygame

1. **Review last week's exercise sheet**

   - Last week's exercise sheet was on developing your own classes and using inheritance. This will be important for today's exercise sheet, so please do work through that sheet first.

2. **Making an 'enemy' class.**
   The goal of this exercise is to write a class that produces the behaviour of an enemy sprite, which will wander randomly and will steal a random amount of gold bricks from any player it touches. Start by downloading the `PyGame_base_code.py` program from Blackboard, as we will use this as our base.

   - Write a class called `Enemy` that inherits from the `pygame.sprite.Sprite` class. We want to create a class that is very similar in structure to the `Player` class i.e. creating a square sprite, colouring it, initialising its speed variables to 0 etc.
   - Start by implementing an `__init__()` method that follows the general layout from the `Player` class:
     (a) Add a new argument to the `__init__()` method to set the size of the enemy sprite as well as the starting x and y coordinates.
     (b) Make the default colour of the enemy sprite be `RED` (you will have to define a new colour at the top of the code).
     (c) Set the `self.size` variable to store the size argument.
     (d) The rest should be kept the same.
   - Implement the same `update()` method, as the game requires that all sprites implement an update method.
   - Next, we're going to add a new method to our `Enemy` class which, given an `EnemySpeed` and the length and width of the window, has our enemy move randomly within the window:
     (a) Because we want our Enemy to change direction at random times, rather than every time we loop through the main code, we need to begin by creating a `delay` variable inside the Enemy class (but not within the new method – usually at the top underneath the class definition) and setting it to 0.
     (b) Create a method with the following signature:
     `moveRandomly(self, speed, xLimit, yLimit):`
     where `speed` will be the speed at which the enemy moves. `xLimit` and `yLimit` will be the window's width and height, to act as bounds within which our enemy will move.
     (c) Then, `IF delay == 0`, we will set a random direction for our enemy:
       – For each possible compass direction e.g. a random direction in
       ["**N**", "**NE**", "**E**", "**SE**", "**S**", "**SW**", "**W**", "**NW**"]
       pick a random direction in the list and have the sprite change its `change_x` and `change_y` values to move in that direction based on its speed.

    - Then, after picking an initial direction to travel in (as delay is set to 0 when the sprite is created), we need to pick a random value for delay e.g. a value between 5 and 100.

(d) `ELSE`
    - decrement the `delay` variable by 1

(e) Then, we need to "bounce" the enemy. This can be tricky but the gist is that:
    - If the enemy position is too far left (< 0) or too far right ($>$ `xLimit`), reverse the x-direction e.g.
      `change_x = change_x * -1`
    - Similarly, if the enemy position is too high (< 0) or too low ($>$ yLimit), reverse the y-direction e.g.
      `change_y = change_y * -1`
    - To be more precise, you will need to adjust the conditions checking the xLimit and yLimit to account for the size of the sprite.

- Create a new instance of the `Enemy` class, assigning it to a variable called `Enemy1` and giving it a starting position and size e.g. $(400, 400)$ and 30. Then add the enemy to the `AllSpritesList`, so that it will be drawn on-screen with the other sprites.
- After the input loop, but **before** `AllSpritesList.update()`, add a call to our Enemy's `moveRandomly()` method, passing in the window's dimensions as follows:
  `Enemy1.moveRandomly(EnemySpeed, 800, 600)` where `EnemySpeed` should probably be a lower value than the `SPEED` variable we set for the players e.g. 5. We could increase the difficulty later by increasing the `EnemySpeed`.

3. **Adding collision detection to the enemy class.**
   Now that we have our enemy moving in random directions, we need to have the `Enemy` sprite detect when it touches a player, and then reduce the player's score by stealing a random amount of gold bricks.

   We can use the existing collision detection code as a basis for having the enemy detect when it touches a player.

   - First, we will need to add the players to a new sprite group, called `PlayerList`, so that we can detect collisions for **only** the players, and not the gold bars.
   - Then, add both `Player1` and `Player2` to our `PlayerList`.
   - Next, in the same location as we detect if players are touching the gold bars, we need to check if `Enemy1` is touching any players in `PlayerList`, and store the result in a list called `EnemyHitList`.
   - Finally, we can set an amount to be stolen as `stealAmount`, and loop through the list. For each player in the list, reduce their score by `stealAmount`. You should also check that if `player.score < 0`, that you set their score to 0 so that they cannot possess a negative score.
   - **Extension:** If you wanted, you could even implement a score for the enemy sprite and display the total amount of stolen gold bricks between the scores of Player1 and Player2.