CS2013 – Final Project – Ines Roman Gracia, 123123969

1. Definition of classes, subclasses in pygame and circuitpython

All the variables defined in the Food, Ghost and Pacman classes are instance variables, in order to give more flexibility to the code, since all these variables are going to change throughout the execution of the code.

Ghost

The ghost moves upwards, downwards, leftwards or rightwards according to the input variable *direction*. In the method that moves the ghost, we add or subtract to its x-position or y-position (depending on the direction) the current speed of the ghost in that axis. However, the ghost cannot be completely off screen, so whenever its position either in the x-position or y-position :

- is lower or equal to its radius (this means off screen) we add one, so that at least one pixel is on the screen.
- is greater or equal to the display boundary plus its radius (this means off screen) we subtract 1, so that at least one pixel it's on the screen.
  Example of management of the boundaries in the y-axis:

```
if direction == "UP":
    self.__y = self.__y - self.__speed_y
    if self.__y < - self.__radius + 1:
        self.__y = - self.__radius + 1
 elif direction == "DOWN":
    self.__y = self.__y + self.__speed_y
    if self.__y > display_height + self.__radius - 1:
        self.__y = display_height + self.__radius - 1
```

Pacman

This class contains four methods that move Pacman in the four different directions: upwards, leftwards, downwards and rightwards. The boundary management is different from the ghost one. Pacman can be off screen in all directions, its limits in the four screen boundaries are the display border plus Pacman's radius. Whenever the Pacman crosses these boundaries it will reappear on the other side of the display, also off screen. For example if Pacman's x-position is display_width plus its radius and it continues to go right, it will reappear on the other side and its x-positions will be minus its radius:

```
# Move Pacman rightwards on the display.
    def move_right(self, display_width):
        self.__x = self.__x + self.__speed_x
        if self.__x > display_width + self.__radius:
            self.__x = - self.__radius
```

SensorLightDisplay

The class **LightDisplay** manages pixel brightness and auto-write behavior and contains a method that creates light patterns (explained in feature 2) that will be displayed by the Neopixels of the Circuit Playground Express.

As in the second assignment, variable *__lights_off* is set as a class variable because it is not going to change from one instance to another: turning the lights off will always be [0, 0, 0] in rgb code. The variable ***__pixels_count*** is set as an instance variable because it gives more flexibility to the code: the number of pixels may change depending on the board.

The **SensorLightDisplay** subclass extends the functionality of its superclass by incorporating sensor data. Through its method *advanced_control_feedback*, it interprets acceleration data from sensors to display the 4 different light patterns implemeted in its superclass. The logic of this method will be explained in the feature 2 section.

2. Pygame main code

Some important features to comment on this part are:

- **keyboard events:** after drawing all the objects on the screen, we read the key presses from *pygame.events*, we are interested in reading the 4 arrows that control the movement of Pacman. There are four boolean variables, each one indicating whether Pacman is moving in one direction or not (up, down, right, and left). Pacman can only move in one direction in every iteration, then only one variable can be set to True at the same time. Whenever an arrow is pressed, the variable that controls the movement in the direction of that arrow will be set to True, and the rest to False (in case there was an arrow that had been pressed before and not yet released). And whenever an arrow is released, the variable that controls the movement in the direction of the arrow will be set to false.
- **collisions:** this part of the code determines whether the coordinates of Pacman and the other objects intersect on both the y-axis and the x-axis of the game display. A collision occurs if the maximum coordinate of one object surpasses the minimum coordinate of the other, and vice versa, on both axes. In case of a collision between Pacman and the food, the food will change size and will be relocated. If there is a collision between Pacman and the ghost, both will be relocated and the speed of the ghost will change.
- **next ghost position:** after checking for collisions, if none have occurred, Pacman and the ghost are updated to their next positions. For Pacman, its direction is determined by the last pressed arrow key, allowing it to move either vertically or horizontally. Meanwhile, the ghost's movement logic is based on advancing towards Pacman. If moving vertically would bring the ghost closer to Pacman's current position, it will move in that direction. Similarly, if moving horizontally would reduce the distance to Pacman, the ghost will move horizontally. We calculate this by comparing the absolute distances between the current positions of the ghost and Pacman to the absolute distance between Pacman and the ghost if it moved in every direction.

3. Circuitpython main code

First of all, the modules that will be used are imported. Then, an instance for the SensorLightDisplay and an instance for the Keyboard class are initialized.

The main loop is executed every 0.1 seconds and it does the following:

1. The acceleration values in the x-axis and y-axis of the Circuit Playground Express are read and stored in variables x and y.
2. Based on the values of x and y, the corresponding arrow keys are pressed or released using the Keyboard object kbd. For instance, if x is less than -3 and smaller than y, the left arrow key is pressed, and the other arrow keys are released. Similar logic applies for the other directions.
3. The *advanced_control_feedbac*k method of the SensorLightDisplay object *sld* is called, passing in the current acceleration data. This method will provide a light pattern displayed by the Neopixels as feedback.

4. Feature 1: Ghost and Food interactions in pygame application

Food

I have implemented a method in the Food class that randomly assigns values within a range both to the width and the height of the food. This method will be called in the *main.py* code when Pacman and the food collide. Also in the *main.py* code I have created two global variables for the limits of the size of the food:

```
SIZE_FOOD_LOWER_LIMIT = 5
SIZE_FOOD_UPPER_LIMIT = 20
```

Ghost

I have implemented a method in the Ghost class that randomly assigns a value within a range to the ghost's speed in the x-axis. This method will be called in the *main.py* code when Pacman and the ghost collide. Also in the *main.py* code I have created two global variables for the limits of the speed of the ghost:

```
SPEED_GHOST_X_LOWER_LIMIT = 2
SPEED_GHOST_X_UPPER_LIMIT = 6
```

5. Feature 2: creating and calling a new method for controller feedback in SensorLightDisplay class

Method: light_scale

This method is responsible for setting the intensity of two levels of lights based on a given blue value and side parameter. If the value of the parameter side is 0 then we don't light on any pixels.

When lighting on the pixels of the x-axis, the first level contains one pixel (index 7 on the left and index 2 on the right) and the second level consists of the two pixels to the sides of the first-level pixel. For the y-axis, the first level consists of two pixels (indexes 4 and 5 on the top and 0 and 9 on the bottom) and the second level contains the adjacent pixels on both sides.

The parameter blue that ranges from 0 to 510 indicates the sum of blue for the two levels of pixels:

- if blue is lower or equal to 255, only the first level of pixels will be lighted on to that value.
- If blue is greater than 255, once the first level is set to the maximum (255), the second level will be lighted in blue to the number of the parameter blue minus 255.

Method: advanced_control_feedback

This method adjusts the lights by using the *light_scale* method based on accelerometer readings. It reads the acceleration values of the x and y axes, and it checks if they are in the range [-9.81, 9.81]. If so, it checks for further conditions of the values of x and y, which determine to what side the Circuit Playground is tilted. The parameter *side* of the *light_scale* method indicates which side has to be lighted (0 if none).

The value of the parameter *blue* of the *light_scale* method is calculated following a proportion rule. The maximum level of acceleration is 9.81 and the maximum number of the sum of blues of the pixels is 510. Given the absolute value of the current acceleration, the sum of the blues to illuminate will be: *acceleration * total_blue / limit_acceleration*. However, starting from the acceleration equal to 3, we need to subtract 3 from the limit acceleration and the current acceleration. This way the value of blue starts from 0 when the acceleration is 3 and it will increase proportionally until the acceleration reaches its limit of 9.81.