## Report

## Definition of classes, subclasses in pygame and circuitpython

### A ) Bubble Class

### For the purpose of this report I will describe how the Bubble class was created in the pygame application and explain how the class works using OOP principles. The Bubble class is instantiated as an object called bubble\_1. Here I pass in the objects attributes as variables which I have predefined. I utilised variables here as from the programmers perspective it’s far easier to understand the implications of changes to values passed into object when variables names denote variable usage. For example bubble\_radius clearly holds the value for the radius size. When these variables are passed into the class template, they are assigned within the constructor method to appropriate private variables. All the necessary attributes to create a circle are passed in as they will later be used to draw and move the bubble within my methods. By assigning these variables to private class variables I support encapsulation whereby we restrict access to an objects attributes outside the class. This prevents unwanted access to variables. Private instance variables within my constructor were also use to enhance customisability across instances of a class. My aim was that another programmer should be able to change the appearance and behaviour of objects across different multiple instances. This also supports object reusability and the creation of multiple Ball objects. Class variables on the other hand could have been used if there were attributes that I sought to keep constant across instances of a class.

Within the class template I have created methods which support the functionality of Ball object. The string representation method provides a user-friendly description of the bubble instance. It includes the all the attributes of the bubbles instance. It proves useful for other programmers to garner information about how the class works. The draw\_Bubble method draws the bubble to screen. The display window onto which the bubble will be drawn is passed into this method. To draw we pass in the display window as well as the bubble colour, x and y coordinates and radius. The move method moves the bubble diagonally down the display, we polarise the speed of the bubble when the balls x\_coordinate is greater or equal the right bound of the screen or less than or equal to the left bound. When it reaches the line (passed in as the vertical bound), we call the relocation method which generates a random x value for the ball to relocate at the top of the screen. To generate a random starting point for the bubble at the beginning of the game I set up a counter which increments when the if statement executes from then onwards the ball is relocated on collision. The if statement executes if the counter equals 0. Once these if statements are checked I increment the x and y coordinates of the bubble giving the appearance of diagonal movement. I then update the bubble coordinates as we pass in the coordinates of the bubble as a pair of values. The relocate method as described above generates a random value between the bounds of the screen. In this case between zero and the width of the display minus the radius of the bubble. I use random uniform here as other users may not pass in whole number positions for display width. I then set up a number of getters and setters for all the objects attributes. This allows me to update variables outside the scope of the class. Getters and setters allow a degree of control for the user of the class template but constrain the user its harder to make unwanted changes to variable values. Getters and setters offer a layer of abstraction away from the complexity of the code inside the class meaning programmers can use this class template without necessarily needing to understand all the code within it.

### B) SensorLightDisplay

The Sensor LightDisplay class is imported from the sensorlightdisplay.py file. It is instantiated as the sensorlightdisplay\_1 object. The brightness of the neopixels is passed in as an argument and will be directly assigned to the neopixels brightness in the constructor method within the class. This class lacks somewhat in reusability and flexibility as from the users perspective the only changeable attribute is pixel brightness. Some variables within the constructor cannot be created as instance variables as they directly access the board. Difference instances of board sensors and actuators cannot be created (we cannot yet simply create new neopixels by creating new instances). We create the colour black as an instance variable so it can be accessed and modified throughout the programme. We set pixel autowrite to false. This ensures that pixels are not lit up as they are iterated over when using for loops. We can execute a for loop and then light the correct number of neopixels following execution. We access the board here, and when changing the brightness by doing this we are directly altering attributes of the board. For this project we only use one of the methods in this class is used – control\_feedback. The acceleration value of the board and the colour used to light up the pixels are passed into this method. The acceleration value allows us to monitor the angle of the board in relation to the horizontal plane. It is used within the class to light the neopixels based on how much the board is tilted.

## Pygame main code

All elements that are to be drawn to the display window are created within the game loop. The loop is updated every frame until the user closes the display window. Within the loop we call the draw methods for the paddle and bubble which are drawn onto the passed in display. I then create a line which will serve as the vertical bound for the bubble. When the bubble y coordinates are the same or greater than the line we relocate the bubble. I then create an event listener using the pygame.event.get() function. If the user presses the user close the window, we change run\_game to false causing the while loop to break and the game to end. We also listen for keydown events. When this event occurs, we check if the key was the space bar. If it is then we increment a counter which cause a below if statement to be satisfied, the bullet is drawn to screen and the bullet trigger is set to true causing it to move upwards. The bubble move method is called the bubble to moves diagonally across the screen, its speed is polarised when it collides with the horizontal bounds of the screen. The paddle responds to the input of the user. I pass in the left and right key into the pair of move methods. Within the left key method, we check the left key against the status of all the keys. If the left key is pressed the key status in this list will be set to true. If the left key is true, we decrement the paddle x position. As long as the left key is pressed the paddle will move until the x coordinate of the paddle is less than or equal to zero. If the right key is pressed, we increment x coordinate of the paddle. This value can increment until it reaches the horizontal bound of the screen minus the width of the paddle. The bullet object when not triggered rests atop the paddle. Its position is relative to the paddle. To centre the bullet on top of the paddle we give it an x coordinate of the paddle plus half the paddle width. We subtract the bullet radius from the paddles y coordinate to position the bullet on the y axis. This works as the bullet is drawn from its centre. We run the wall\_collision method to check whether the bullet y coordinate is equal to the back wall of the display. If a collision occurs triggered is set to False. When the move method runs again its check if this variable is true. If its true it continues to move the bullet upwards. If its false we the bullet returns to its position of rest and moves relative to the paddle once more. In the else part of this if block we update the position of the bullet before it is has been triggered for the first time as it hasn’t been drawn.

## Circuitpython main code

From the circuit python express library we first import cp (circuit playground). We will use cp to refer the boards sensors and Neopixels. I import time which can be used for the addition of delays. We instantiate the imported sensorlight display class. Into which we pass the pixel brightness. Next, we import the usb\_hib library from which we can import the keyboard which allows us to access the system keyboard. The while loop is a forever loop which will run indefinitely. It continuously reads in the acceleration values of the board. These values range from -9.8 to + 9.8 with noise to be accounted for at either end. Within this if statement I nest a series of if statements that allow actions on the board to execute keystrokes on my computer. Firstly, we access the switch on the board. If the switch is set to on it checks the nested if’s. This allows us to quickly switch of the boards ability to access the computer keystrokes. If the switch is true we.. check if the button is pressed. If its pressed the space key on the computer Is executed causing the bullet to launch. If the x acceleration value of the board is less than -3 we press the left key and if the acceleration value is greater than 3 we press the right key. Otherwise we release both keys. To prove the user feedback on their movements we call the sensorlightdisplay classes control feedback method which is explained in more detail below.

### Feature 1: Bubble and Bullet interactons in pygame application

As part of the gameplay collisions between the bullet and bubble cause changes in the appearance and functionality of both objects. The collision\_check function is called in the main code. I pass in the two objects as arguments with the aim of enhancing reusability. We could potentially pass in different instances of bullet and bubble objects. If a consistent naming convention was used across objects, we could check collisions for any displayed circles. The function continuously checks if objects have collided. It does this by finding the Euclidean distance between the two centre points of the circles. If this distance is less than the sum of the radii of the two objects, then they have collided. In the case of collision a number of actions occur. We call the relocate method which gives the bubble a random starting point at the top of the screen. We set the bullet trigger to false causing it to move relative to the paddle again. We call the change\_size method in the bubble class. This method accepts a upper and lower limit for the size of the bubble. Within the method I decrement the radius of the bubble until it reaches the lower limit. I have hardcoded the decrementer variable as I think this value should be kept constant. In future versions this could be made into an instance variable so that another programmer could control how much the size is decremented by. On collision the change\_speed method is called. This method slows the speed of the bullet it, accepts a lower and upper limit as input. Within the method the speed of the bullet is decremented if the speed of the bullet is greater than the lower limit and less than the upper limit.

### Feature 2: Creating and calling a new method for controller feedback in SensorLightDisplay class

The SesnorLightDisplay class is instantiated in the main code and a value for the brightness is passed in. For the purpose of this assignment only the control\_feedback method is used. This method is called in the main code. The acceleration x value and list of colours is passed in. If the acceleration value is between -9.81 and 9.81 we execute. This disregards x values that are considered noise. If the acceleration values are between 2 and -2 we don’t light up any pixels so that when the board is flat no neopixels light up. If the acceleration value is greater than 2 we… Map the x value to range 2 to 9.81. To convert the current acceleration into a percentage of the total range divide it by 7.81. To map our percentage to the corresponding neopixels we then multiply this by the number of neopixels available. This value is rounded so it can be used to access neopixel indexes. I create a new list and set up a while loop. In which I add elements to a list from 4 to the rounded acceleration value. Thus the list begins at 4 and decrements by one until it reaches the acceleration value. Using this list, I light up the corresponding pixels accessing them via index. Within this method I have created another mapping calculation to map the acceleration value to colour. To do this first subtract 2 from the acceleration value as I want the range to span from 2 to 9.81 I then divide this number by the overall range (7.81) to make acceleration a percentage of the total range. I then multiply this figure by 127. By multiplying the percentage by 127 I get the equivalent colour value that corresponds to the acceleration value. Therefore, as I tilt the board and my acceleration value gets bigger and so does my overall percentage and thereby amount of green shown in my neopixels. For the left-hand side of the board, I polarise the negative acceleration values as rgb values are positive. Similarly I subtract 2 from the acceleration value as the range is from 2 to 9.81 I divide by 7.81 to make acceleration a percentage of the range(2 -> 9.81). I then multiply by 127 to map to the neo pixels.