Assignment 5 – Interaction Part II

This short lab assignment is worth 3% of the module.

You are expected to spend 1-4 hours (including the scheduled lab class)

You will get half the marks (1.5%) for any reasonable attempt (even if it doesn’t quite work).

NOTE: the following assumes you attended the lecture, which provided a verbal walkthrough of the steps. You may also need to look at the concepts discussed in the lecture.

If anything is not clear, PLEASE DO NOT HESITATE TO CONTACT ME IN-LAB OR BY EMAIL.
In this lab..

- **Programming concepts**
  - If statement
  - Loops*
  - Arrays*

- **Processing concepts**
  - colour + transparency
  - more shapes: rect
  - random

- **Visual computing concepts**
  - Animation
  - A particle (system*)

*: optional for this lab, but you will need these later so it would be beneficial to try these out now
A “Particle”
Essentially a small simple animated object that moves around for visual effect

- Minimum requirements
  - Particles have a position.
    - In 2D: an x and y position
      Essentially we keep track of the position and draw the particle at a certain position *each frame*
  - Particles have a speed
    - In 2D its x-speed and its y-speed: i.e. how much the x and y position change *each frame*
    - To update the particle add increase x by x-speed
STEP 1: Minimal program
(As with last week) start with functions to set window size and clear the screen

```cpp
void setup()
{
    //set the viewport size to 600 x 600 pixels
    size (600, 600);
}

void draw()
{
    //clear the screen to blue
    background(0, 0, 255);
}
```
STEP 2: Draw A Particle

- For now let's just draw it at the mouse position using a coloured ellipse
  - The following should work
    ```
    fill(255, 0, 0);
    ellipse(mouseX, mouseY, 50, 50);
    ```
  - but, for animation we will want to CHANGE the colour and the radius (size) of the ellipse:
  - In fact let's **randomize** these a bit

```java
void draw()
{
    float redness = random(255);
    fill(redness, 0, 0);
    float circle_size = random(10, 100);
    ellipse(mouseX, mouseY, circle_size, circle_size);
}
```
random( ... )

- In processing...
  - `random( max )` returns float* value between 0 and max
  - `random( min, max)` returns a float value between min and max
  - Note that `float` is a number type which can have decimal points. Usually used for measuring and positioning things e.g. 1.5, -32.4, 0

```cpp
float distance_right = random(width);
float circle_size = random(10, 100);
```
STEP 3: A Particle’s Position

During an animation, objects change their state (some variable property) from one frame to the next; to mimic motion. For the particle the state is mainly it’s position and velocity (next slide)

- Create (or declare) a variable to store the X position (how far across) of the particle and another to store the Y position (how far down) of the particle
  - For each of these, use a variable of type float and make it GLOBAL variable (outside of the body of any function)
  - In setup() you should give some initial values for the position of your particle
  - Now each time you redraw the scene, you should draw a particle at the position you stored

```cpp
float distance_right; float distance_down;
void setup()
{
  size (600, 200);
  distance_right = 300;
  distance_down = 100;
}
void draw()
{
  //add code from previous slides here
  ellipse (distance_right, distance_down, circle_size, circle_size);
}
```
STEP 4: A Particle’s Speed

- Next declare a variable to store the X speed (how fast it moves to the right) of the particle and another to store the Y speed (how fast it moves down) of the particle.
  - Give some initial values for the distances and speeds.
  - Declare a function `update()` where you will update the particles position.
  - Call the function at the start of `draw()`.
  - Now each time you update the animation you should change the position of the particle (e.g. add `speed_right` to `distance_right`).

- If done correctly the particle should be animated now when you run the program.

```cpp
//add code from previous slides here
float speed_right;
float speed_down;

void setup()
{
    //add code from previous slides here
    speed_right = 14;
}

void draw()
{
    update();
    //add code from previous slides here
}

void update()
{
    distance_right = distance_right + speed_right;
}
```

NOTE: this example just updates positions in the X direction. See if you can also move in Y as well.
STEP 5: Collisions (bouncing the particle)

Unfortunately the particle will move towards the right then disappear off the screen. To keep it in we need to bounce it off the walls.

- Check if the x position is greater than the width of the window (here’s where the IF statement comes in)
  - Recall: The system variable `width` holds the size of the window
  - You should compare this to the X position (`distance_right`) of the particle
  - If it is too far to the right, “bounce” the particle by REVERSING IT’S SPEED. i.e. multiply the X speed `-1` (that’s negative one)
- You also need to check if the x position is less than 0

```java
void update()
{
    distance_right = distance_right + speed_right;
    if (distance_right > width)
    {
        speed_right = -speed_right;
    }
    if (distance_right < 0 )
    {
        speed_right = -speed_right;
    }
}
```

NOTE: this example just updates movement in the X direction. See if you can also move in Y
STEP 6: Customization

- Give it a bit of your own style by changing the look of the particle
- Try some of the following
  - Set the background to white at the beginning of draw to clear the screen
  - Change its color
  - Change its size
  - Change its shape (ellipses that are not circular, rectangles, lines, small images)
ADVANCED STEPS

NOTE: You will score full marks if you successfully complete everything up to Step 6.

The following is entirely optional but will look a bit nicer.
STEP 7: Some Visual Tricks

- Instead of clearing the screen with `background(...)`, repeatedly draw a large semi-transparent rectangle over the screen.
  - Remove any call to `background(...)` in `draw` or move it to `setup`.
  - Instead, add the following at the start of `draw()`:
    ```
    fill (255, 255, 255, 50)
    rect( 0, 0, width, height );
    ```
  - The effect of this is that the old positions of the particle remain on the screen for a little while and fade away.

- Blur the screen each time it is drawn to soften “older” particles. More on such operations later but the following code added after the above code will do this:
  ```
  filter(BLUR, 1);
  ```
STEP 7: Arrays of Particles

- For now* create separate arrays for x, y positions and x, y speeds
- You will need to declare (Global) arrays to hold the values
  ```c
  float[] distance_right;
  float[] speed_right;
  ```
- Then you should initialize them to some value (you will need a for loop)
  ```c
  Do this in setup
  distance_right = new float[20];  //20 particles
  for (int i=0; i<20; i++)
  {
      distance_right[i] = random(width);
  }
  ```

*: This may be more efficient with classes.

NOTE: this example just deals with the X position. See if you can also move in Y as well
STEP 8: Drawing and Updating the Array of Particles

- Access the array using a loop to find out where to draw each particle
  - Do this in the draw function
    ```cpp
    for (int i=0; i<20; i++)
    {
      ellipse(distance_right[i], distance_down[i], rad, rad);
    }
    ```
- Use another for loop in the update function to update each particle’s position based on its speed. You will also need to BOUNCE each particle off the walls
  ```cpp
  for (int i=0; i<20; i++)
  {
    distance_right[i] = distance_right[i] + speed_right[i];
    if (distance_right[i] > width)
    {
      speed_right[i] = -speed_right[i];
    }
  }
  ```

NOTE: this example just deals with the X position. See if you can also move in Y as well
Further things to try

- Increase number of particles

- Add force. Instead of increasing the positions, this should increase speeds eg.
  - Gravity: increase the speed very slightly in the downward direction each frame
  - Wind: increase speed towards right or left

- Add some interactivity based on mouse or keyboard events e.g.
  - Restart or reset velocities of particles when key is pressed
  - Particles get repulsed/created at mouse position