HAND IN MODULE 4

OLE K LARSEN, MARIUS HAAKONSEN

1. Task 1

The demo of interpolation will show a circle shape moving from the top left corner to the bottom right corner of the screen. The shape will begin as a small, dark line, and end up being a large, lightly colored oval shape.

Using the 'float lerp' function we can return the interpolated version of P0 and P1 based on the variable 'tid'.

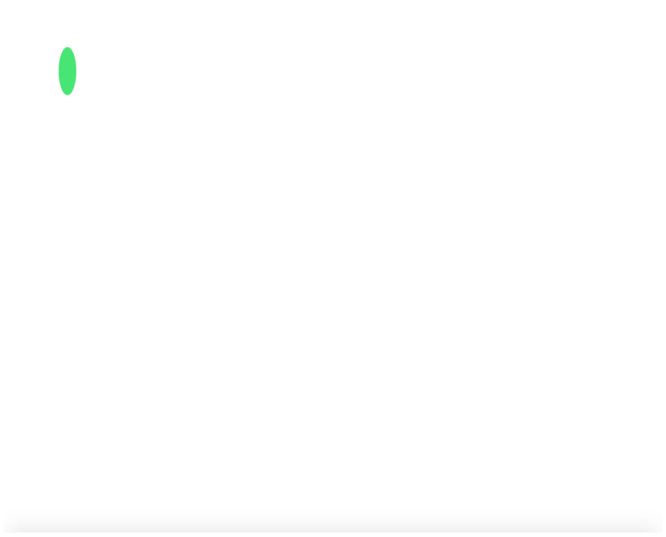
```
float lerp(float tid, float P0, float P1) { // Returns interpolated float.
    return (P1 - P0) * tid + P0;
}
```

The following variables are used to interpolate to and from.

```
float minX = 5.0f;
float minY = 5.0f; // X and Y pos. for where the circle begins

float maxX = WIDTH - 150.0f;
float maxY = HEIGTH - 150.0f; // X and Y pos. for where the circle ends

float minRadius = 5.0f;     // Beginning radius
float maxRadius = 300.0f;   // Ending radius

float minCol = 1.0f;    // Beginning color
float maxCol = 255.0f;  // Ending color

float minScl = 0.0f;  // Beginning scale
float maxScl = 1.0f;  // Ending scale
```

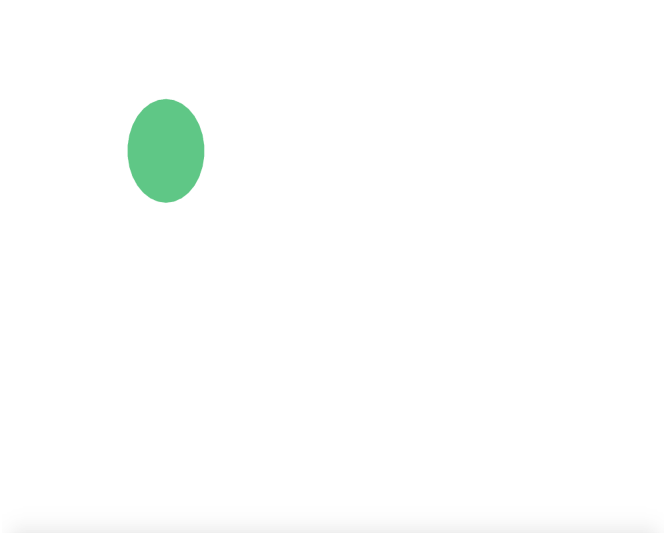
Changing the poisition utilizes CircleShape.setPosition, taking an x and y value.

```
// Interpolating position
c.setPosition(lerp(t, minX, maxX),lerp(t, minY, maxY));
```



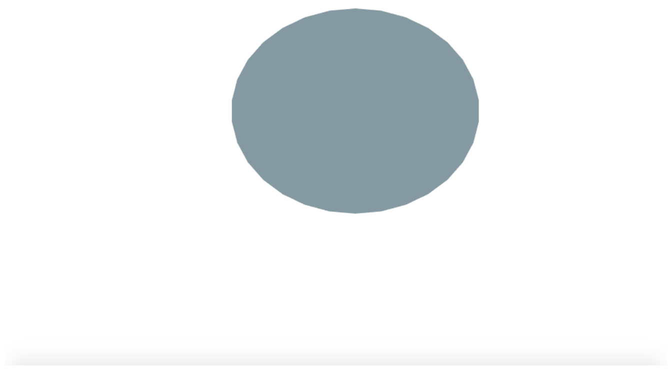
Changing the size utilizes CircleShape.setRadius, taking a single value that slowly increases.

```
// Interpolating size
c.setRadius(lerp(t, minRadius, maxRadius*0.75));
```



Changing the color utilizes CircleShape.setFillColor. This function takes 3 variables. In this example they are scaled with +50 and +100 to vary the color from only white to black.

```
// Interpolating color
c.setFillColor(sf::Color(lerp(t, minCol+50, maxCol),
                         lerp(t, maxCol, minCol),
                         lerp(t, minCol+100, maxCol)));
```



Changing the shape utilizes CircleShape.setScale, taking 2 values: the size in the x direction and the size in the y direction.

```
// Interpolating shape
c.setScale(lerp(t, minScl, maxScl)*5, lerp(t, minScl, maxScl)+minX/4);
```

The time variable is incremented by 0.001 each game loop, and reseting when t exceeds 1.

```
t += 0.001f;
  if(t > 1) {
      t = 0;
  }
```

2. Task 4

a)

Defining an array of lines 'lines', capable of storing a maximum 1025 lines. The first and last line is set to the middle of the screen.

```
sf::VertexArray lines(sf::LinesStrip, WIDTH+1);

    lines[0].position = sf::Vector2f(0,HEIGTH/2);
    lines[WIDTH].position = sf::Vector2f(WIDTH, HEIGTH/2);
```

A float array of datapoints is declared to store 1025 values.

The first and last value in the data structure is set to 0.0f.

```
    float data[WIDTH+1];
    data[0] = 0.0f;
    data[WIDTH] = 0.0f;
```

The first loop happens 512 times, dividing stepsize (=1024) by 2 each time. The nested loop with 'int i' moves 1 stepsize each increment.

```
for (int stepsize = WIDTH; stepsize > 1; stepsize/=2) {
    for (int i = stepsize/2; i < WIDTH; i += stepsize) {
```

The following bits of code happens inside the nested for-loops.

Creating a random value 'val' and scaling it to a suitable size based on screen heigth.

```
    val = dis(gen);
    val *= HEIGTH/2.5;
```

The data[i] value will function as the corresponding y-value for line[i], using the midpoint displacement algorithm.

When exiting the nested loop, alpha (0.5) is divided by beta (2.0).

```
            data[i] = ((data[i-stepsize/2] + data[i+stepsize/2])/2 + alpha*val);

            lines[i].position = sf::Vector2f(WIDTH/(WIDTH/2)*i, data[i]+HEIGTH/2);

        } alpha /= beta;
    }
```