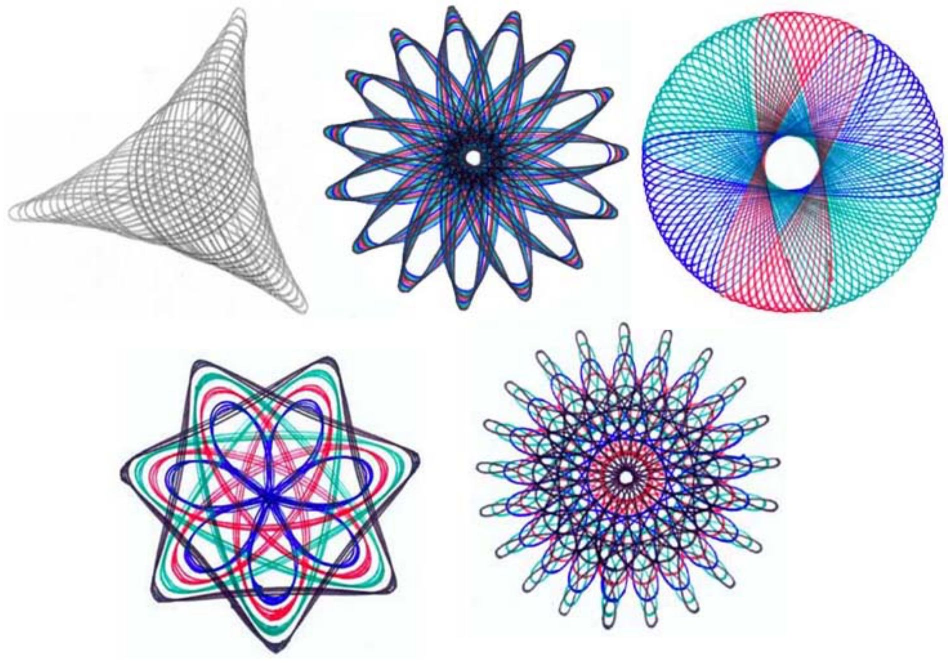
Turtle Graphics Assignment

Introduction

Turtle graphics is a common method for introducing children to programming. It was created by Wally Feurzeig, Seymour Papert, and Cynthia Solomon in 1967 as part of the original Logo programming language.

Picture a mechanical turtle that moves under the control of a set of precise instructions.
The turtle holds the pen in either the up or down position. While the pen is lowered, the turtle sketches out forms as it moves; when the pen is raised, the turtle moves freely without writing.
The turtle possesses a location, an orientation (or direction), and a pen. These are its three attributes.
Turtle Graphics gives you the ability to design any kind of pattern you can imagine, from the simplest to the most intricate.



(Python Turtle Spirograph, n.d.)

What I needed to do?

Turtle Methods

Pen control:

- **Drawing state:** pendown or penup

Turtle motion:

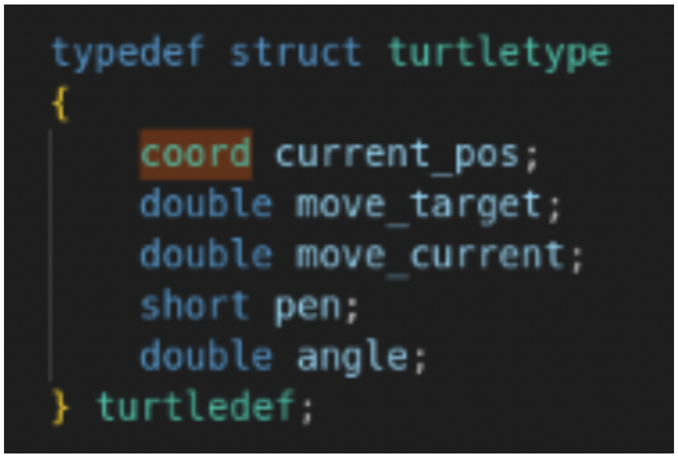
- **Tell turtle state:** position, towards x-coordinate, y-coordinate, heading, distance
- **Setting and measurement:** degrees, radians

Screen Methods

- **Window control:** background colour, clear, screen size
- **Animation control:** delay

The program is supposed to ask the user which of the pre-organized patterns he would like to be displayed on the screen and to pick a background colour.

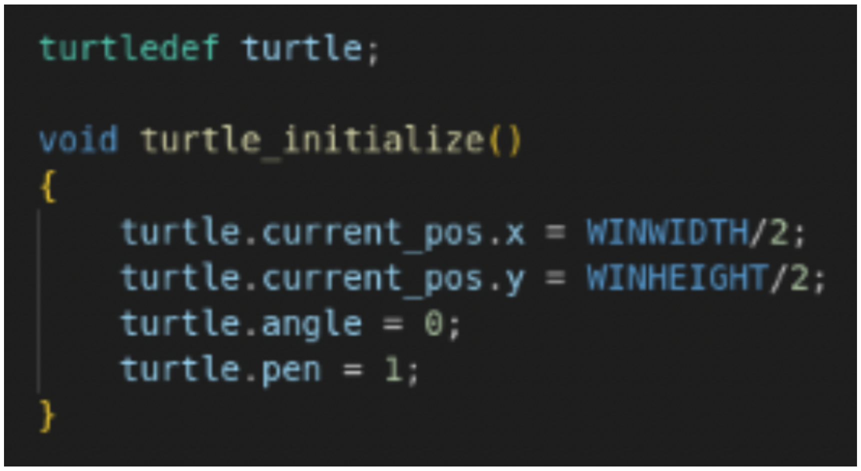
First thing I needed to do is to define a turtle type that will house the properties required for the initialization of the turtle



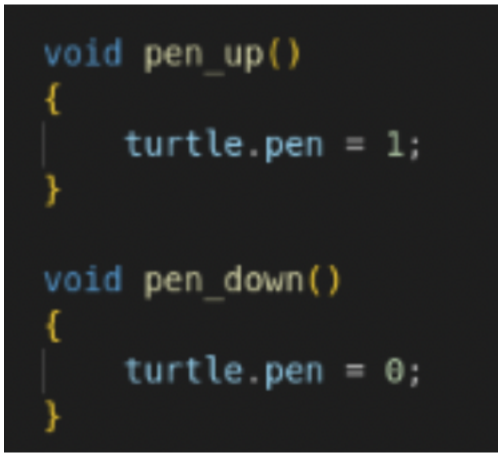
```
typedef struct turtletype
{
    coord current_pos;
    double move_target;
    double move_current;
    short pen;
    double angle;
} turtledef;
```

Then I initialized the turtle by assigning values to the turtle object's properties:

- The 'turtle' initial position will be the middle of the screen, hence why its 'x' and 'y' position values are half of the screen's width and length respectively.
- The 'turtle angle will be 0 degrees.
- The 'turtle' pen's value will be 1, which means it is set not to draw a line until being commanded otherwise.

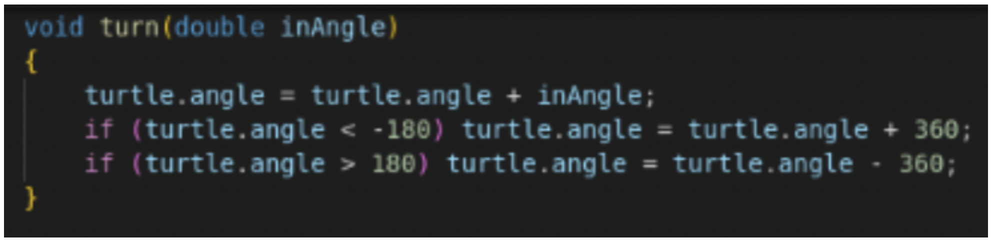


```
turtledef turtle;

void turtle_initialize()
{
    turtle.current_pos.x = WINWIDTH/2;
    turtle.current_pos.y = WINHEIGHT/2;
    turtle.angle = 0;
    turtle.pen = 1;
}
```



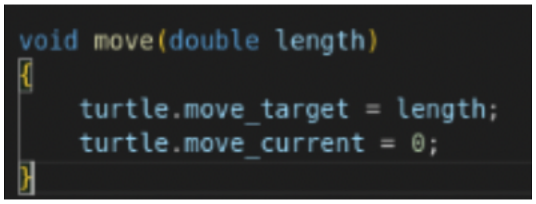
```
void pen_up()
{
    turtle.pen = 1;
}

void pen_down()
{
    turtle.pen = 0;
}
```

A **pen_up** and **pen_down** functions that can be called to determine whether the pen is up (no line will be drawn) or the pen is down (a line will be drawn) on the screen.



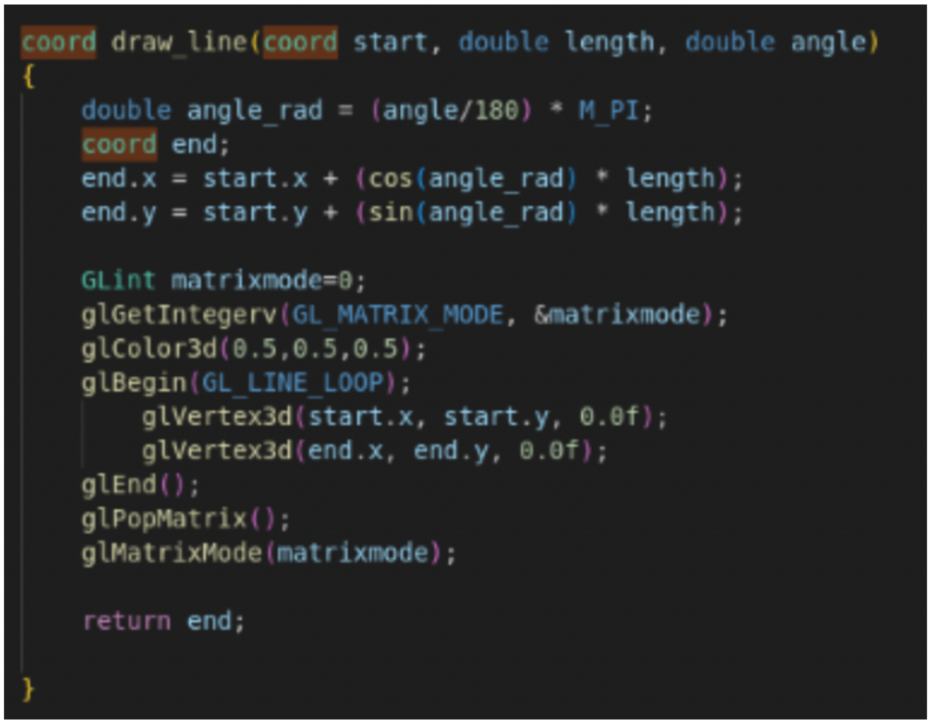
```
void turn(double inAngle)
{
    turtle.angle = turtle.angle + inAngle;
    if (turtle.angle < -180) turtle.angle = turtle.angle + 360;
    if (turtle.angle > 180) turtle.angle = turtle.angle - 360;
}
```

A **turn** function that gets the angle value and makes sure its maximum value is 360 degrees.



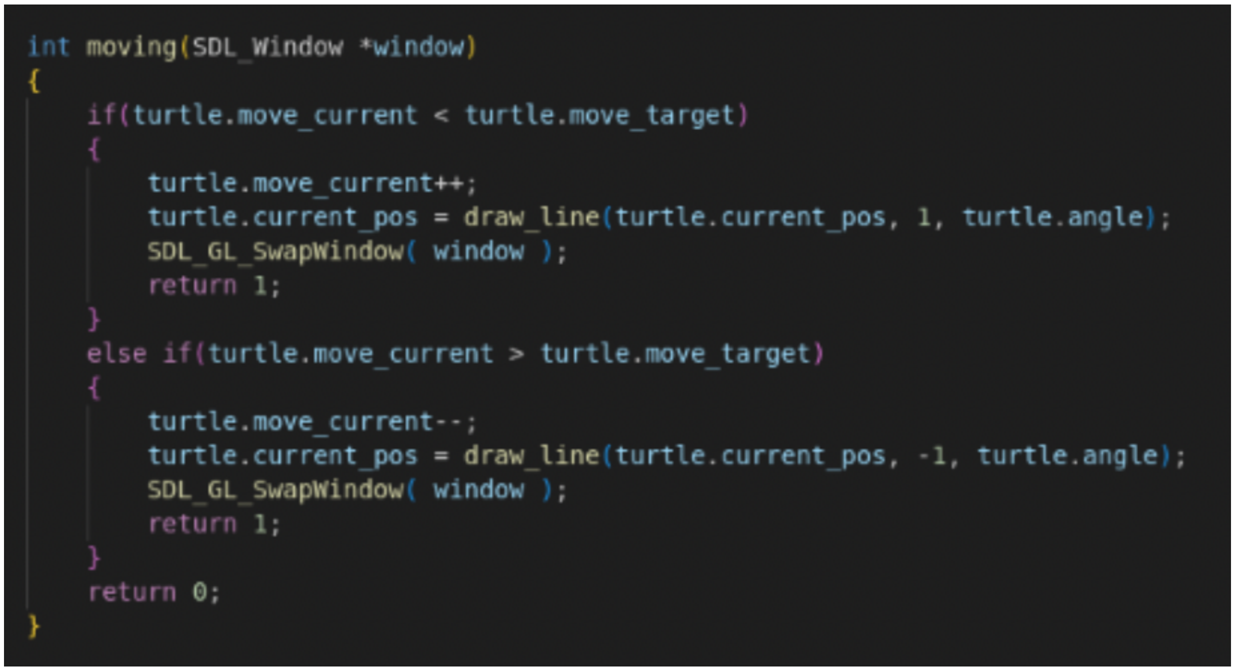
```
void move(double length)
{
    turtle.move_target = length;
    turtle.move_current = 0;
}
```

A **move** function that gets the length value and by that determines the **move_target** value. **move_current** is set to 0 and in another function, **moving**, (which I will explain later here) it increases until it is equal to the **move_target** value. this creates the animation of the 'turtle' moving.



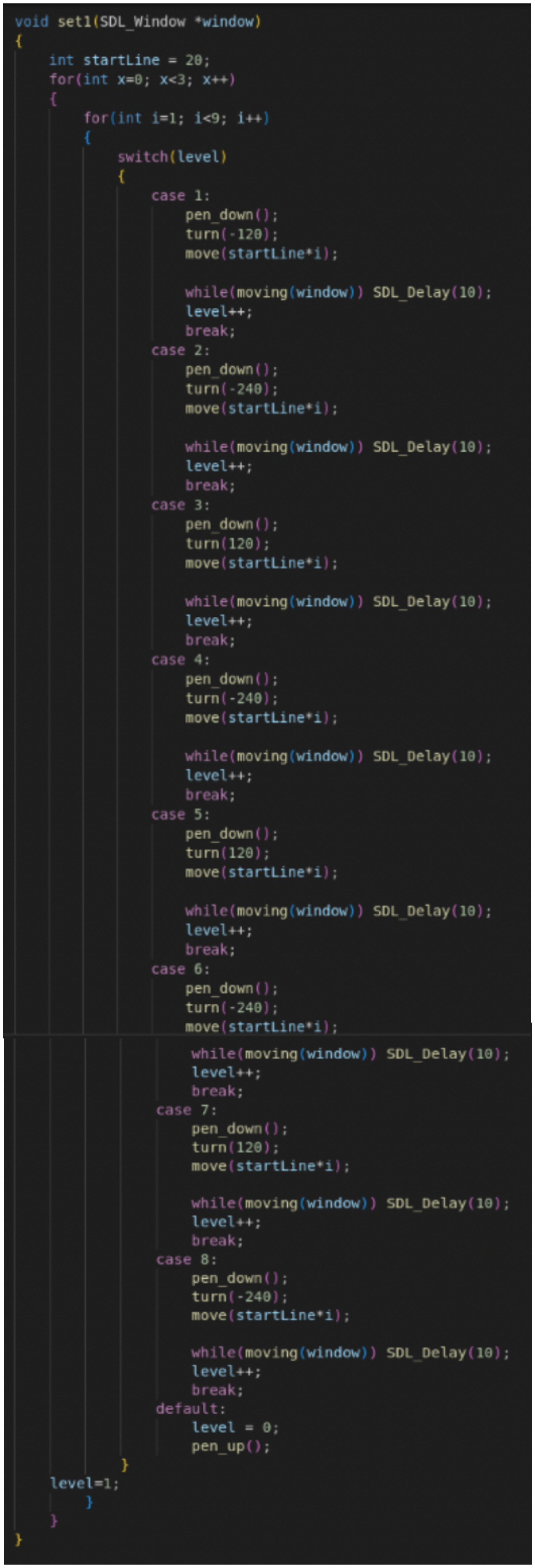
```
coord draw_line(coord start, double length, double angle)
{
    double angle_rad = (angle/180) * M_PI;
    coord end;
    end.x = start.x + (cos(angle_rad) * length);
    end.y = start.y + (sin(angle_rad) * length);

    GLint matrixmode=0;
    glGetIntegerv(GL_MATRIX_MODE, &matrixmode);
    glColor3d(0.5,0.5,0.5);
    glBegin(GL_LINE_LOOP);
        glVertex3d(start.x, start.y, 0.0f);
        glVertex3d(end.x, end.y, 0.0f);
    glEnd();
    glPopMatrix();
    glMatrixMode(matrixmode);

    return end;

}
```

A **draw_line** function that gets a start, length and angle values and calculates the coordinate in which the 'turtle' needs to be in at the end of its movement.



```
int moving(SDL_Window *window)
{
    if(turtle.move_current < turtle.move_target)
    {
        turtle.move_current++;
        turtle.current_pos = draw_line(turtle.current_pos, 1, turtle.angle);
        SDL_GL_SwapWindow( window );
        return 1;
    }
    else if(turtle.move_current > turtle.move_target)
    {
        turtle.move_current--;
        turtle.current_pos = draw_line(turtle.current_pos, -1, turtle.angle);
        SDL_GL_SwapWindow( window );
        return 1;
    }
    return 0;
}
```

A **moving** function that increases the **move_current** value until it is equal to the **move_target** value. that is, there is the final coordinate that the 'turtle' needs to get to at the end of every line, and the **move_current** value increases until it gets to the target – the 'turtle' moves until reaching its destination.



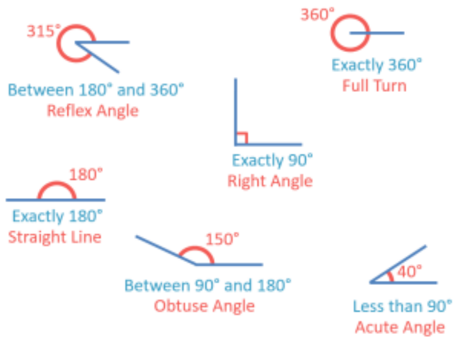
```
void set1(SDL_Window *window)
{
    int startLine = 20;
    for(int x=0; x<3; x++)
    {
        for(int i=1; i<9; i++)
        {
            switch(level)
            {
                case 1:
                    pen_down();
                    turn(-120);
                    move(startLine*i);

                    while(moving(window)) SDL_Delay(10);
                    level++;
                    break;
                case 2:
                    pen_down();
                    turn(-240);
                    move(startLine*i);

                    while(moving(window)) SDL_Delay(10);
                    level++;
                    break;
                case 3:
                    pen_down();
                    turn(120);
                    move(startLine*i);

                    while(moving(window)) SDL_Delay(10);
                    level++;
                    break;
                case 4:
                    pen_down();
                    turn(-240);
                    move(startLine*i);

                    while(moving(window)) SDL_Delay(10);
                    level++;
                    break;
                case 5:
                    pen_down();
                    turn(120);
                    move(startLine*i);

                    while(moving(window)) SDL_Delay(10);
                    level++;
                    break;
                case 6:
                    pen_down();
                    turn(-240);
                    move(startLine*i);
                    while(moving(window)) SDL_Delay(10);
                    level++;
                    break;
                case 7:
                    pen_down();
                    turn(120);
                    move(startLine*i);

                    while(moving(window)) SDL_Delay(10);
                    level++;
                    break;
                case 8:
                    pen_down();
                    turn(-240);
                    move(startLine*i);

                    while(moving(window)) SDL_Delay(10);
                    level++;
                    break;
                default:
                    level = 0;
                    pen_up();
            }
    level=1;
        }
    }
}
```

A **set1** function that is responsible to draw the pattern on the screen. There are two more functions like that: **set2** and **set3** as there are 3 patterns for the user to choose from.

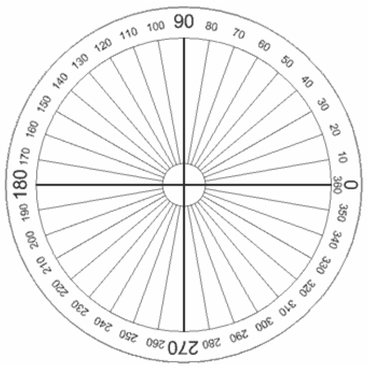
Every pattern is a certain shape composed of lines in different angles and lengths. Then this shape is drawn repeatedly in a curtain number of times and a pattern is created.

Therefore, instead of writing a command for every line in the pattern, I wrote commands for the lines of one shape and then repeated it using 'for loops', so the code can be more practical and easier to read.

In this example, the pattern is made of 3 triangles, and every triangle is made of 8 lines in which every next line is longer than the previous one by the initial length.
The first loop repeats 3 times, for the 3 triangles and inside this loop there is another loop that repeats 8 times to draw the lines of the shape.
Then inside this loop, there is 'switch…case'. The case is the 'level' value that is initially set to 1.
Every case determines that the pen is down, that is, the line is drawn and visible, then determines an angle and length.
The initial length is 20 (int startLine).
And finally increases the 'level' by 1 so it can move forward to the next case.
After all the lines of the first shape are drawn, the loop ends, level is set to 1 again and another shape is being drawn, creating eventually the pattern.

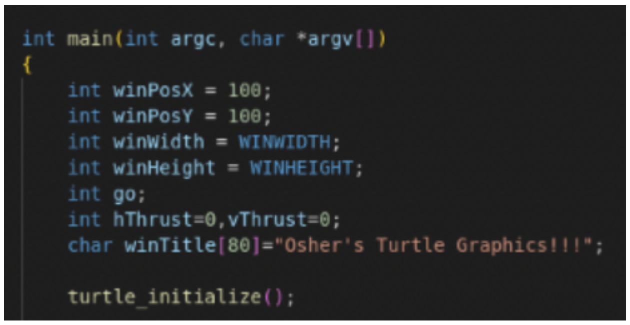


(Classifying Angles as Acute, Obtuse, Right or Reflex)



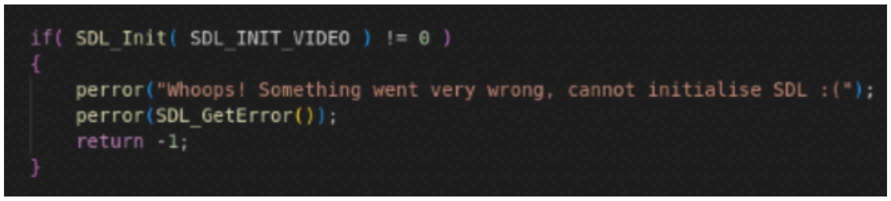
(Measuring Degrees)

I also used the images above in order to help me calculate and understand the degrees of the shapes and implement it in my code



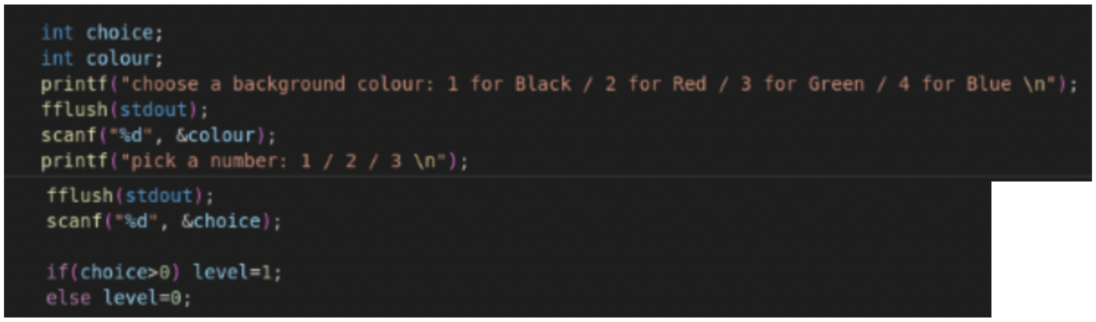
```
int main(int argc, char *argv[])
{
    int winPosX = 100;
    int winPosY = 100;
    int winWidth = WINWIDTH;
    int winHeight = WINHEIGHT;
    int go;
    int hThrust=0,vThrust=0;
    char winTitle[80]="Osher's Turtle Graphics!!!";

    turtle_initialize();
```

The determination of the screen values in the **main** function



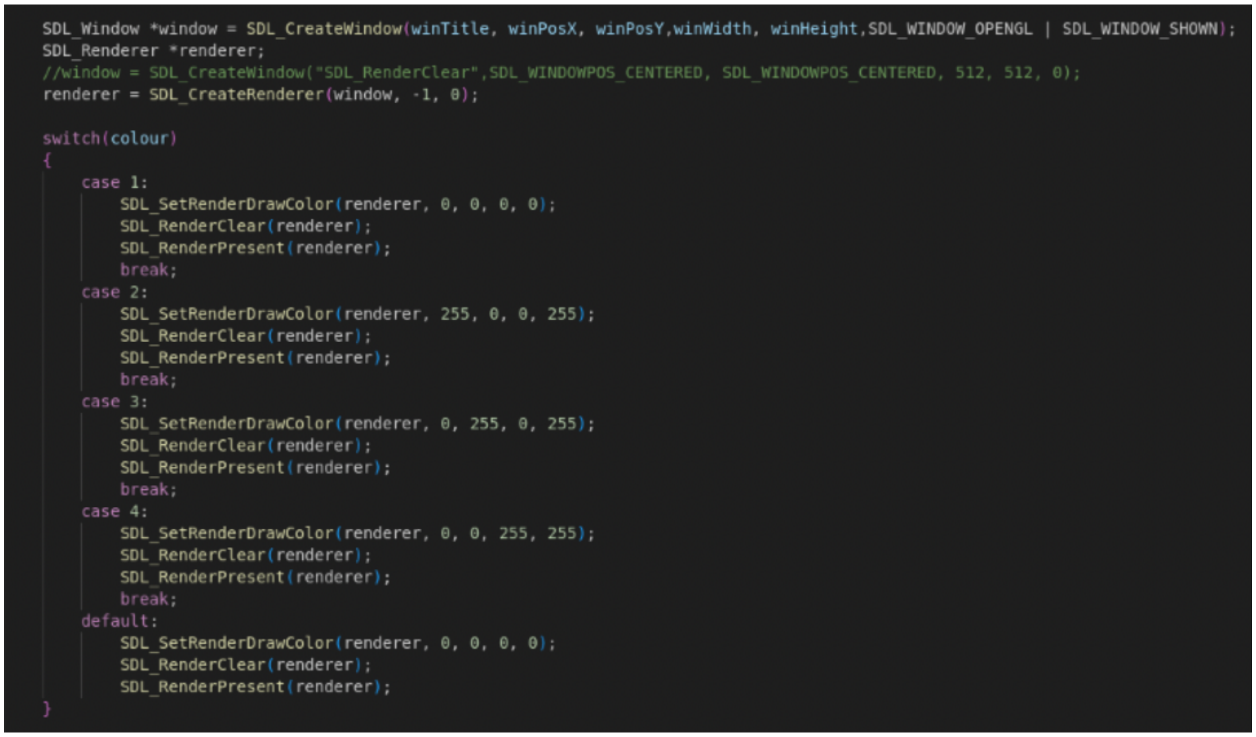
```
if( SDL_Init( SDL_INIT_VIDEO ) != 0 )
{
    perror("Whoops! Something went very wrong, cannot initialise SDL :(");
    perror(SDL_GetError());
    return -1;
}
```

SDL_Init is the primary function involved in the initiation process for SDL.
It accepts a 'flag' input that we utilize to inform SDL of the systems we intend to use. We wish to initialize everything, therefore we set the corresponding flag. This method also provides an error value when an error occurs.



```
int choice;
int colour;
printf("choose a background colour: 1 for Black / 2 for Red / 3 for Green / 4 for Blue \n");
fflush(stdout);
scanf("%d", &colour);
printf("pick a number: 1 / 2 / 3 \n");
 fflush(stdout);
 scanf("%d", &choice);

 if(choice>0) level=1;
 else level=0;
```

The program prints messages for the user in order for him to choose a background colour and a pattern.



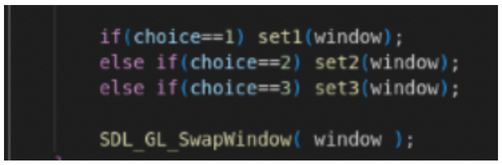
```
SDL_Window *window = SDL_CreateWindow(winTitle, winPosX, winPosY,winWidth, winHeight,SDL_WINDOW_OPENGL | SDL_WINDOW_SHOWN);
SDL_Renderer *renderer;
//window = SDL_CreateWindow("SDL_RenderClear",SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, 512, 512, 0);
renderer = SDL_CreateRenderer(window, -1, 0);

switch(colour)
{
    case 1:
        SDL_SetRenderDrawColor(renderer, 0, 0, 0, 0);
        SDL_RenderClear(renderer);
        SDL_RenderPresent(renderer);
        break;
    case 2:
        SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
        SDL_RenderClear(renderer);
        SDL_RenderPresent(renderer);
        break;
    case 3:
        SDL_SetRenderDrawColor(renderer, 0, 255, 0, 255);
        SDL_RenderClear(renderer);
        SDL_RenderPresent(renderer);
        break;
    case 4:
        SDL_SetRenderDrawColor(renderer, 0, 0, 255, 255);
        SDL_RenderClear(renderer);
        SDL_RenderPresent(renderer);
        break;
    default:
        SDL_SetRenderDrawColor(renderer, 0, 0, 0, 0);
        SDL_RenderClear(renderer);
        SDL_RenderPresent(renderer);
}
```

‘switch…case’ state to determine the background color based on what the user chose.

RGB Colour Model
is a type of colour model known as an additive colour model. In this model, the red, green, and blue fundamental colours of light are combined in a variety of different ways to produce a wide range of colours.

RGB Colour System:
creates all possible hues by mixing **R**ed **G**reen and **B**lue colours.
Each of the red, green, and blue components employ 8 bits, and their integer values range from 0 to 255.

I decided to keep the program simple and therefore put into option only 4 colors: red, green, blue, and black (also as a default).



```
    if(choice==1) set1(window);
    else if(choice==2) set2(window);
    else if(choice==3) set3(window);

    SDL_GL_SwapWindow( window );
```

An ‘if’ state checking which of the patterns was chosen and then calling the relevant function in order to execute it.

Bibliography

File:Turtle-Graphics Polyspiral.svg. (n.d.). Retrieved from wikimedia commons: https://commons.wikimedia.org/wiki/File:Turtle-Graphics_Polyspiral.svg

Python Turtle Graphics Design Ideas. (n.d.). Retrieved from YouTube: https://www.youtube.com/watch?v=Z_FEJ_JrJJE

Python Turtle Spirograph. (n.d.). Retrieved from 101 computing: https://www.101computing.net/python-turtle-spirograph/

Classifying Angles as Acute, Obtuse, Right or Reflex. (n.d.). Retrieved from Maths With Mum: https://www.mathswithmum.com/classifying-angles/

Measuring Degrees. (n.d.). Retrieved from Maths is fun: https://www.mathsisfun.com/geometry/degrees.html

RGB color Codes Chart. (n.d.). Retrieved from RapidTables: https://www.rapidtables.com/web/color/RGB_Color.html

RGB color model. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/RGB_color_model

Turtle graphics. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Turtle_graphics

turtle - Turtle graphics. (n.d.). Retrieved from Python: https://docs.python.org/3/library/turtle.html

SDL Wiki. (n.d.). Retrieved from SDL Wiki: https://wiki.libsdl.org/SDL2/FrontPage

I also used the codes uploaded to BrightSpace as a reference and left relevant bits of them in my code.