SASTA OLIPHANT

SCIENCE AWARDS

SOUTH AUSTRALIAN SCIENCE TEACHERS ASSOCIATION

Prize Winner

Programming, Apps & Robotics

Year 11-12

Stuart Vass

Pembroke School - Middle School

ROWE SCIENTIFIC PTY. LTD. www.rowe.com.au

Australian Government Department of Defence

Government of South Australia Department for Education

SASTA South Australian Science Teachers Association

Oliphant Science Awards 2023

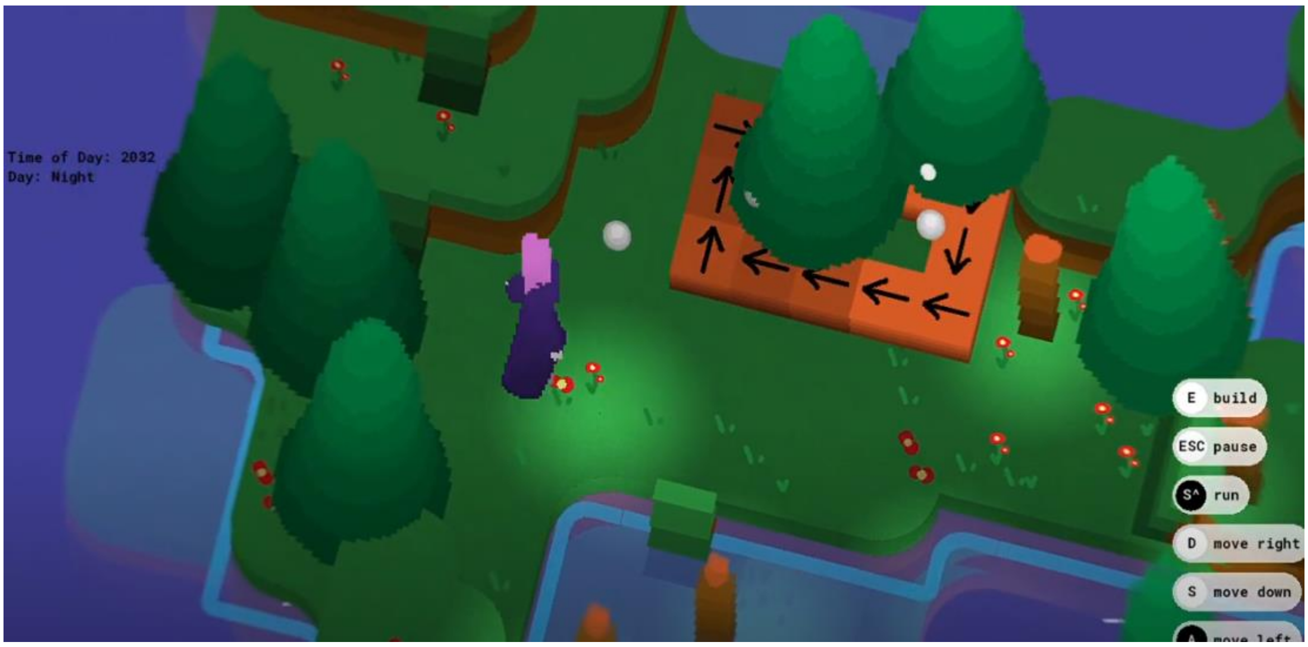
Entry 0486-014 Programming, Apps & Robotics
Stuart Vass, Year 12

Charmful

Charmful is an app where you wander an island, looking adventure, running with the wind, playing with the trees.

The landscape and all graphics generate on the fly. From the movement of your character to the way the trees sway in the wind, is simulated behaviour using scientific understanding of interactions. Mathematical equations are used to create stacked 2D images ("sprite stacking") that appear to look and behave like moving 3-Dimensions.

This comprehensive yet simple program could be further extended and applied to an RPG/top-down game. This creates a unique style that isn't used in many games, and because it's pseudo 3D, the rest of the game can be coded as if it was a regular 2D RPG.



Instructions

You play as the main character wandering the island, running with the wind, playing with the trees. It is a game developed for all ages, but particularly 8- to 12-year-olds.

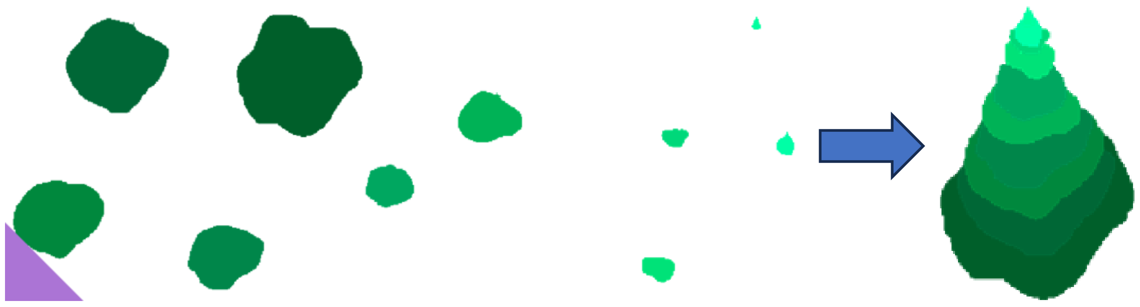
Device: Any computer that can run .exe – typically Windows. Install like a regular program with the .exe file. Link: https://www.dropbox.com/s/gzrrdx2fyg1f7nw/Charmful%20V0.0.5.2.2%20OLIPHANT%20SCIENCE.exe?dl=0

Development

The game has been authored using Game Maker Studio 2. Using an education account, same as the now free version, and can export as a .exe file.

It was designed to try to create self-generating 3D trees that move realistically in the wind.

It is my work, including most of the assets; some royalty-free assets are credited. Thank you to my teachers and School for their support throughout my schooling.



Sprite stacking is a relatively straight forward way of creating depth. In Game Maker you can have a sprite, which is a series of images where you draw them from bottom to top with a negative y offset each time:

```
var length = sprite_get_number(sprite_index);
var interval = 10;
for (var i = 0; i < length; i++) {
    draw_sprite(sprite_index, i, x, y-i*interval);
}
```

But if multiple objects (such as pine trees) do this, although each individual tree has depth, it looks like a bunch of flat images still.

Instead, what we can do is tell the program to draw every objects bottom layer first, then every objects second layer next, instead of drawing one whole object before starting on the next.

```
var length = 200;
var interval = 1;
for (var i = 0; i < length; i++) {
    with(parent_sprite_stacking) {
        draw_sprite(sprite_index, i, x, y-i*interval);
    }
}
```

One of the best benefits of using sprite stacking though is that you can program a camera to move in 3D space by just rotating the draw window around.

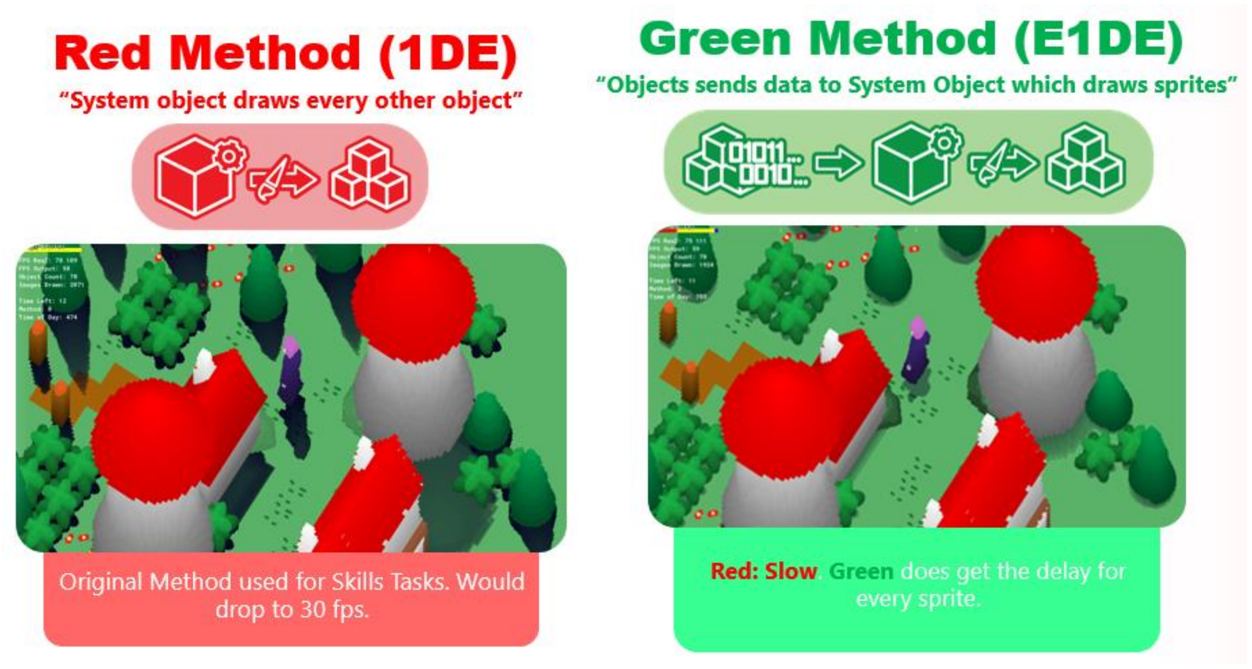
```
var length = 200;
var interval = 1;

var cam = view_camera[0];
var cam_a = camera_get_view_angle(cam);
var cam_i = lengthdir_x(1, 90-cam_a);
var cam_j = lengthdir_y(1, 90-cam_a);

for (var i = 0; i < length; i++) {
    with(parent_sprite_stacking) {
        draw_sprite(sprite_index, i, x-i*cam_i*interval, y-i*cam_j*interval);
    }
}
```

The notation cam_i and cam_j is used as what we're creating is unit vectors upwards in relation to the camera which in maths is referred to as i and j. Already we're putting into practice maths which most people consider not useful in regular life!

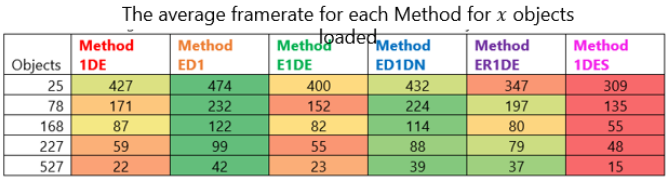
Here is a video of the final program: https://www.youtube.com/watch?v=VucqDvPuJnY

Everything else in the program is built upon this bit of code – well, used to. As part my journey in creating a sprite stacking program which was efficient enough to fit my purposes, I developed 6 different methods of drawing sprites to the screen. You can see these methods here: https://www.youtube.com/watch?v=9JmyDRLT7ac

My final program actually uses the Green method opposed to the method (red) I've been describing.

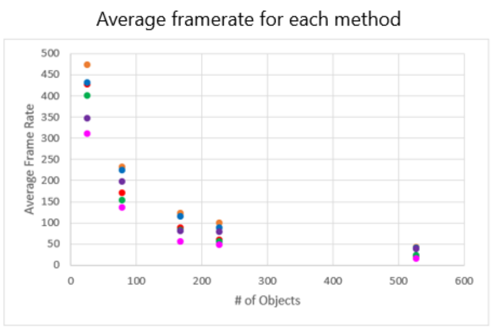

Red Method (1DE)
"System object draws every other object"
Original Method used for Skills Tasks. Would drop to 30 fps.
Green Method (E1DE)
"Objects sends data to System Object which draws sprites"
01011... 0010...
Red: Slow. Green does get the delay for every sprite.

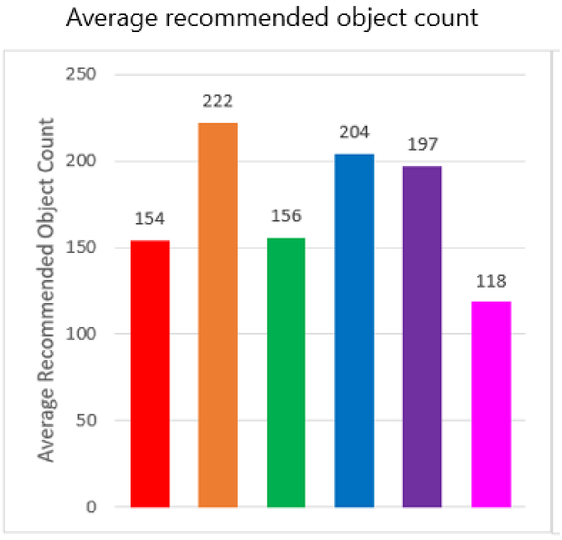
The efficiency of each method can be found in the table and graph:



The average framerate for each Method for *x* objects loaded

Objects	Method 1DE	Method ED1	Method E1DE	Method ED1DN	Method ER1DE	Method 1DES
25	427	474	400	432	347	309
78	171	232	152	224	197	135
168	87	122	82	114	80	55
227	59	99	55	88	79	48
527	22	42	23	39	37	15

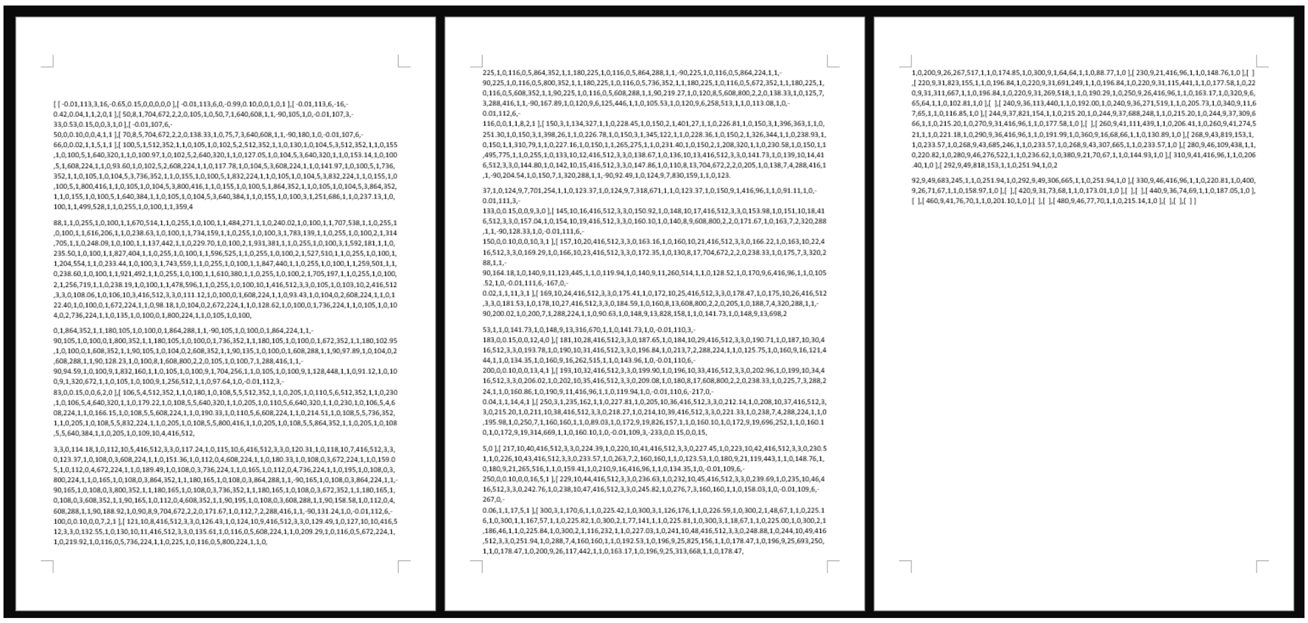




Although the Orange Method was the most efficient, those efficiencies came with visual downsides, so the Green Method was chosen.

Instead of opening every object and drawing the objects in layers from bottom to top, instead all the objects add information to a very long list with details on how to draw the sprite and in what order.

The list:



```
[ [ -0.01,113,3,16,-0.65,0.15,0,0,0,0,0 ],[ -0.01,113,6,0,-0.99,0.10,0,0,1,0,1 ],[ -0.01,113,6,-16,-
0.42,0.04,1,1,2,0,1 ],[ 50,8,1,704,672,2,2,0,105,1,0,50,7,1,640,608,1,1,-90,105,1,0,-0.01,107,3,-
33,0.53,0.15,0,0,3,1,0 ],[ -0.01,107,6,-
50,0,0.10,0,0,4,1,1 ],[ 70,8,5,704,672,2,2,0,138.33,1,0,75,7,3,640,608,1,1,-90,180,1,0,-0.01,107,6,-
66,0,0.02,1,1,5,1,1 ],[ 100,5,1,512,352,1,1,0,105,1,0,102,5,2,512,352,1,1,0,130,1,0,104,5,3,512,352,1,1,0,155
,1,0,100,5,1,640,320,1,1,0,100.97,1,0,102,5,2,640,320,1,1,0,127.05,1,0,104,5,3,640,320,1,1,0,153.14,1,0,100
,5,1,608,224,1,1,0,93.60,1,0,102,5,2,608,224,1,1,0,117.78,1,0,104,5,3,608,224,1,1,0,141.97,1,0,100,5,1,736,
352,1,1,0,105,1,0,104,5,3,736,352,1,1,0,155,1,0,100,5,1,832,224,1,1,0,105,1,0,104,5,3,832,224,1,1,0,155,1,0
,100,5,1,800,416,1,1,0,105,1,0,104,5,3,800,416,1,1,0,155,1,0,100,5,1,864,352,1,1,0,105,1,0,104,5,3,864,352,
1,1,0,155,1,0,100,5,1,640,384,1,1,0,105,1,0,104,5,3,640,384,1,1,0,155,1,0,100,3,1,251,686,1,1,0,237.13,1,0,
100,1,1,499,528,1,1,0,255,1,0,100,1,1,359,4

88,1,1,0,255,1,0,100,1,1,670,514,1,1,0,255,1,0,100,1,1,484,271,1,1,0,240.02,1,0,100,1,1,707,538,1,1,0,255,1
,0,100,1,1,616,206,1,1,0,238.63,1,0,100,1,1,734,159,1,1,0,255,1,0,100,3,1,783,139,1,1,0,255,1,0,100,2,1,314
,705,1,1,0,248.09,1,0,100,1,1,137,442,1,1,0,229.70,1,0,100,2,1,931,381,1,1,0,255,1,0,100,3,1,592,181,1,1,0,
235.50,1,0,100,1,1,827,404,1,1,0,255,1,0,100,1,1,596,525,1,1,0,255,1,0,100,2,1,527,510,1,1,0,255,1,0,100,1,
1,204,554,1,1,0,233.44,1,0,100,3,1,743,559,1,1,0,255,1,0,100,1,1,847,440,1,1,0,255,1,0,100,1,1,259,501,1,1,
0,238.60,1,0,100,1,1,921,492,1,1,0,255,1,0,100,1,1,610,380,1,1,0,255,1,0,100,2,1,705,197,1,1,0,255,1,0,100,
2,1,256,719,1,1,0,238.19,1,0,100,1,1,478,596,1,1,0,255,1,0,100,10,1,416,512,3,3,0,105,1,0,103,10,2,416,512
,3,3,0,108.06,1,0,106,10,3,416,512,3,3,0,111.12,1,0,100,0,1,608,224,1,1,0,93.43,1,0,104,0,2,608,224,1,1,0,1
22.40,1,0,100,0,1,672,224,1,1,0,98.18,1,0,104,0,2,672,224,1,1,0,128.62,1,0,100,0,1,736,224,1,1,0,105,1,0,10
4,0,2,736,224,1,1,0,135,1,0,100,0,1,800,224,1,1,0,105,1,0,100,

0,1,864,352,1,1,180,105,1,0,100,0,1,864,288,1,1,-90,105,1,0,100,0,1,864,224,1,1,-
90,105,1,0,100,0,1,800,352,1,1,180,105,1,0,100,0,1,736,352,1,1,180,105,1,0,100,0,1,672,352,1,1,180,102.95
,1,0,100,0,1,608,352,1,1,90,105,1,0,104,0,2,608,352,1,1,90,135,1,0,100,0,1,608,288,1,1,90,97.89,1,0,104,0,2
,608,288,1,1,90,128.23,1,0,100,8,1,608,800,2,2,0,105,1,0,100,7,1,288,416,1,1,-
90,94.59,1,0,100,9,1,832,160,1,1,0,105,1,0,100,9,1,704,256,1,1,0,105,1,0,100,9,1,128,448,1,1,0,91.12,1,0,10
0,9,1,320,672,1,1,0,105,1,0,100,9,1,256,512,1,1,0,97.64,1,0,-0.01,112,3,-
83,0,0.15,0,0,6,2,0 ],[ 106,5,4,512,352,1,1,0,180,1,0,108,5,5,512,352,1,1,0,205,1,0,110,5,6,512,352,1,1,0,230
,1,0,106,5,4,640,320,1,1,0,179.22,1,0,108,5,5,640,320,1,1,0,205,1,0,110,5,6,640,320,1,1,0,230,1,0,106,5,4,6
08,224,1,1,0,166.15,1,0,108,5,5,608,224,1,1,0,190.33,1,0,110,5,6,608,224,1,1,0,214.51,1,0,108,5,5,736,352,
1,1,0,205,1,0,108,5,5,832,224,1,1,0,205,1,0,108,5,5,800,416,1,1,0,205,1,0,108,5,5,864,352,1,1,0,205,1,0,108
,5,5,640,384,1,1,0,205,1,0,109,10,4,416,512,

3,3,0,114.18,1,0,112,10,5,416,512,3,3,0,117.24,1,0,115,10,6,416,512,3,3,0,120.31,1,0,118,10,7,416,512,3,3,
0,123.37,1,0,108,0,3,608,224,1,1,0,151.36,1,0,112,0,4,608,224,1,1,0,180.33,1,0,108,0,3,672,224,1,1,0,159.0
5,1,0,112,0,4,672,224,1,1,0,189.49,1,0,108,0,3,736,224,1,1,0,165,1,0,112,0,4,736,224,1,1,0,195,1,0,108,0,3,
800,224,1,1,0,165,1,0,108,0,3,864,352,1,1,180,165,1,0,108,0,3,864,288,1,1,-90,165,1,0,108,0,3,864,224,1,1,-
90,165,1,0,108,0,3,800,352,1,1,180,165,1,0,108,0,3,736,352,1,1,180,165,1,0,108,0,3,672,352,1,1,180,165,1,
0,108,0,3,608,352,1,1,90,165,1,0,112,0,4,608,352,1,1,90,195,1,0,108,0,3,608,288,1,1,90,158.58,1,0,112,0,4,
608,288,1,1,90,188.92,1,0,90,8,9,704,672,2,2,0,171.67,1,0,112,7,2,288,416,1,1,-90,131.24,1,0,-0.01,112,6,-
100,0,0.10,0,0,7,2,1 ],[ 121,10,8,416,512,3,3,0,126.43,1,0,124,10,9,416,512,3,3,0,129.49,1,0,127,10,10,416,5
12,3,3,0,132.55,1,0,130,10,11,416,512,3,3,0,135.61,1,0,116,0,5,608,224,1,1,0,209.29,1,0,116,0,5,672,224,1,
1,0,219.92,1,0,116,0,5,736,224,1,1,0,225,1,0,116,0,5,800,224,1,1,0,
```

```
225,1,0,116,0,5,864,352,1,1,180,225,1,0,116,0,5,864,288,1,1,-90,225,1,0,116,0,5,864,224,1,1,-
90,225,1,0,116,0,5,800,352,1,1,180,225,1,0,116,0,5,736,352,1,1,180,225,1,0,116,0,5,672,352,1,1,180,225,1,
0,116,0,5,608,352,1,1,90,225,1,0,116,0,5,608,288,1,1,90,219.27,1,0,120,8,5,608,800,2,2,0,138.33,1,0,125,7,
3,288,416,1,1,-90,167.89,1,0,120,9,6,125,446,1,1,0,105.53,1,0,120,9,6,258,513,1,1,0,113.08,1,0,-
0.01,112,6,-
116,0,0,1,1,8,2,1 ],[ 150,3,1,134,327,1,1,0,228.45,1,0,150,2,1,401,27,1,1,0,226.81,1,0,150,3,1,396,363,1,1,0,
251.30,1,0,150,3,1,398,26,1,1,0,226.78,1,0,150,3,1,345,122,1,1,0,228.36,1,0,150,2,1,326,344,1,1,0,238.93,1,
0,150,1,1,310,79,1,1,0,227.16,1,0,150,1,1,265,275,1,1,0,231.40,1,0,150,2,1,208,320,1,1,0,230.58,1,0,150,1,1
,495,775,1,1,0,255,1,0,133,10,12,416,512,3,3,0,138.67,1,0,136,10,13,416,512,3,3,0,141.73,1,0,139,10,14,41
6,512,3,3,0,144.80,1,0,142,10,15,416,512,3,3,0,147.86,1,0,110,8,13,704,672,2,2,0,205,1,0,138,7,4,288,416,1
,1,-90,204.54,1,0,150,7,1,320,288,1,1,-90,92.49,1,0,124,9,7,830,159,1,1,0,123.

37,1,0,124,9,7,701,254,1,1,0,123.37,1,0,124,9,7,318,671,1,1,0,123.37,1,0,150,9,1,416,96,1,1,0,91.11,1,0,-
0.01,111,3,-
133,0,0.15,0,0,9,3,0 ],[ 145,10,16,416,512,3,3,0,150.92,1,0,148,10,17,416,512,3,3,0,153.98,1,0,151,10,18,41
6,512,3,3,0,157.04,1,0,154,10,19,416,512,3,3,0,160.10,1,0,140,8,9,608,800,2,2,0,171.67,1,0,163,7,2,320,288
,1,1,-90,128.33,1,0,-0.01,111,6,-
150,0,0.10,0,0,10,3,1 ],[ 157,10,20,416,512,3,3,0,163.16,1,0,160,10,21,416,512,3,3,0,166.22,1,0,163,10,22,4
16,512,3,3,0,169.29,1,0,166,10,23,416,512,3,3,0,172.35,1,0,130,8,17,704,672,2,2,0,238.33,1,0,175,7,3,320,2
88,1,1,-
90,164.18,1,0,140,9,11,123,445,1,1,0,119.94,1,0,140,9,11,260,514,1,1,0,128.52,1,0,170,9,6,416,96,1,1,0,105
.52,1,0,-0.01,111,6,-167,0,-
0.02,1,1,11,3,1 ],[ 169,10,24,416,512,3,3,0,175.41,1,0,172,10,25,416,512,3,3,0,178.47,1,0,175,10,26,416,512
,3,3,0,181.53,1,0,178,10,27,416,512,3,3,0,184.59,1,0,160,8,13,608,800,2,2,0,205,1,0,188,7,4,320,288,1,1,-
90,200.02,1,0,200,7,1,288,224,1,1,0,90.63,1,0,148,9,13,828,158,1,1,0,141.73,1,0,148,9,13,698,2

53,1,1,0,141.73,1,0,148,9,13,316,670,1,1,0,141.73,1,0,-0.01,110,3,-
183,0,0.15,0,0,12,4,0 ],[ 181,10,28,416,512,3,3,0,187.65,1,0,184,10,29,416,512,3,3,0,190.71,1,0,187,10,30,4
16,512,3,3,0,193.78,1,0,190,10,31,416,512,3,3,0,196.84,1,0,213,7,2,288,224,1,1,0,125.75,1,0,160,9,16,121,4
44,1,1,0,134.35,1,0,160,9,16,262,515,1,1,0,143.96,1,0,-0.01,110,6,-
200,0,0.10,0,0,13,4,1 ],[ 193,10,32,416,512,3,3,0,199.90,1,0,196,10,33,416,512,3,3,0,202.96,1,0,199,10,34,4
16,512,3,3,0,206.02,1,0,202,10,35,416,512,3,3,0,209.08,1,0,180,8,17,608,800,2,2,0,238.33,1,0,225,7,3,288,2
24,1,1,0,160.86,1,0,190,9,11,416,96,1,1,0,119.94,1,0,-0.01,110,6,-217,0,-
0.04,1,1,14,4,1 ],[ 250,3,1,235,162,1,1,0,227.81,1,0,205,10,36,416,512,3,3,0,212.14,1,0,208,10,37,416,512,3,
3,0,215.20,1,0,211,10,38,416,512,3,3,0,218.27,1,0,214,10,39,416,512,3,3,0,221.33,1,0,238,7,4,288,224,1,1,0
,195.98,1,0,250,7,1,160,160,1,1,0,89.03,1,0,172,9,19,826,157,1,1,0,160.10,1,0,172,9,19,696,252,1,1,0,160.1
0,1,0,172,9,19,314,669,1,1,0,160.10,1,0,-0.01,109,3,-233,0,0.15,0,0,15,

5,0 ],[ 217,10,40,416,512,3,3,0,224.39,1,0,220,10,41,416,512,3,3,0,227.45,1,0,223,10,42,416,512,3,3,0,230.5
1,1,0,226,10,43,416,512,3,3,0,233.57,1,0,263,7,2,160,160,1,1,0,123.53,1,0,180,9,21,119,443,1,1,0,148.76,1,
0,180,9,21,265,516,1,1,0,159.41,1,0,210,9,16,416,96,1,1,0,134.35,1,0,-0.01,109,6,-
250,0,0.10,0,0,16,5,1 ],[ 229,10,44,416,512,3,3,0,236.63,1,0,232,10,45,416,512,3,3,0,239.69,1,0,235,10,46,4
16,512,3,3,0,242.76,1,0,238,10,47,416,512,3,3,0,245.82,1,0,276,7,3,160,160,1,1,0,158.03,1,0,-0.01,109,6,-
267,0,-
0.06,1,1,17,5,1 ],[ 300,3,1,170,6,1,1,0,225.42,1,0,300,3,1,126,176,1,1,0,226.59,1,0,300,2,1,48,67,1,1,0,225.1
6,1,0,300,1,1,167,57,1,1,0,225.82,1,0,300,2,1,77,141,1,1,0,225.81,1,0,300,3,1,18,67,1,1,0,225.00,1,0,300,2,1
,186,46,1,1,0,225.84,1,0,300,2,1,116,232,1,1,0,227.03,1,0,241,10,48,416,512,3,3,0,248.88,1,0,244,10,49,416
,512,3,3,0,251.94,1,0,288,7,4,160,160,1,1,0,192.53,1,0,196,9,25,825,156,1,1,0,178.47,1,0,196,9,25,693,250,
1,1,0,178.47,1,0,200,9,26,117,442,1,1,0,163.17,1,0,196,9,25,313,668,1,1,0,178.47,
```

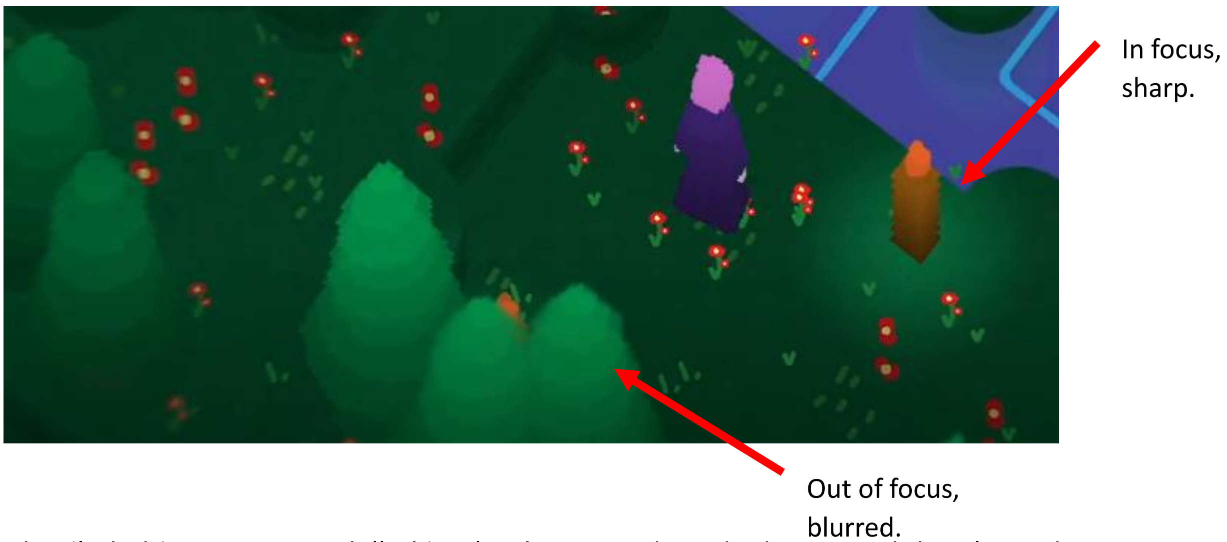
```
1,0,200,9,26,267,517,1,1,0,174.85,1,0,300,9,1,64,64,1,1,0,88.77,1,0 ],[ 230,9,21,416,96,1,1,0,148.76,1,0 ],[ ]
,[ 220,9,31,823,155,1,1,0,196.84,1,0,220,9,31,691,249,1,1,0,196.84,1,0,220,9,31,115,441,1,1,0,177.58,1,0,22
0,9,31,311,667,1,1,0,196.84,1,0,220,9,31,269,518,1,1,0,190.29,1,0,250,9,26,416,96,1,1,0,163.17,1,0,320,9,6,
65,64,1,1,0,102.81,1,0 ],[ ],[ 240,9,36,113,440,1,1,0,192.00,1,0,240,9,36,271,519,1,1,0,205.73,1,0,340,9,11,6
7,65,1,1,0,116.85,1,0 ],[ 244,9,37,821,154,1,1,0,215.20,1,0,244,9,37,688,248,1,1,0,215.20,1,0,244,9,37,309,6
66,1,1,0,215.20,1,0,270,9,31,416,96,1,1,0,177.58,1,0 ],[ ],[ 260,9,41,111,439,1,1,0,206.41,1,0,260,9,41,274,5
21,1,1,0,221.18,1,0,290,9,36,416,96,1,1,0,191.99,1,0,360,9,16,68,66,1,1,0,130.89,1,0 ],[ 268,9,43,819,153,1,
1,0,233.57,1,0,268,9,43,685,246,1,1,0,233.57,1,0,268,9,43,307,665,1,1,0,233.57,1,0 ],[ 280,9,46,109,438,1,1,
0,220.82,1,0,280,9,46,276,522,1,1,0,236.62,1,0,380,9,21,70,67,1,1,0,144.93,1,0 ],[ 310,9,41,416,96,1,1,0,206
.40,1,0 ],[ 292,9,49,818,153,1,1,0,251.94,1,0,2

92,9,49,683,245,1,1,0,251.94,1,0,292,9,49,306,665,1,1,0,251.94,1,0 ],[ 330,9,46,416,96,1,1,0,220.81,1,0,400,
9,26,71,67,1,1,0,158.97,1,0 ],[ ],[ 420,9,31,73,68,1,1,0,173.01,1,0 ],[ ],[ ],[ 440,9,36,74,69,1,1,0,187.05,1,0 ],
[ ],[ 460,9,41,76,70,1,1,0,201.10,1,0 ],[ ],[ ],[ 480,9,46,77,70,1,1,0,215.14,1,0 ],[ ],[ ],[ ] ]
```

This list includes lists (and that list used to also include lists). Near the end of development, it turns out embedded lists and lists saved to a temporary variable cause memory leaks. This is because when a list embeds another list or when a list is saved to a variable the ID of that list is actually what is being saved compared to the actual list. So, when the parent list or the variable is deleted it only deletes the reference to the list, not the list itself, and when 120,000 lists are made in this way every second problems start up fast. Thankfully this was resolved after 3 days of debugging trial and error and searching the internet.

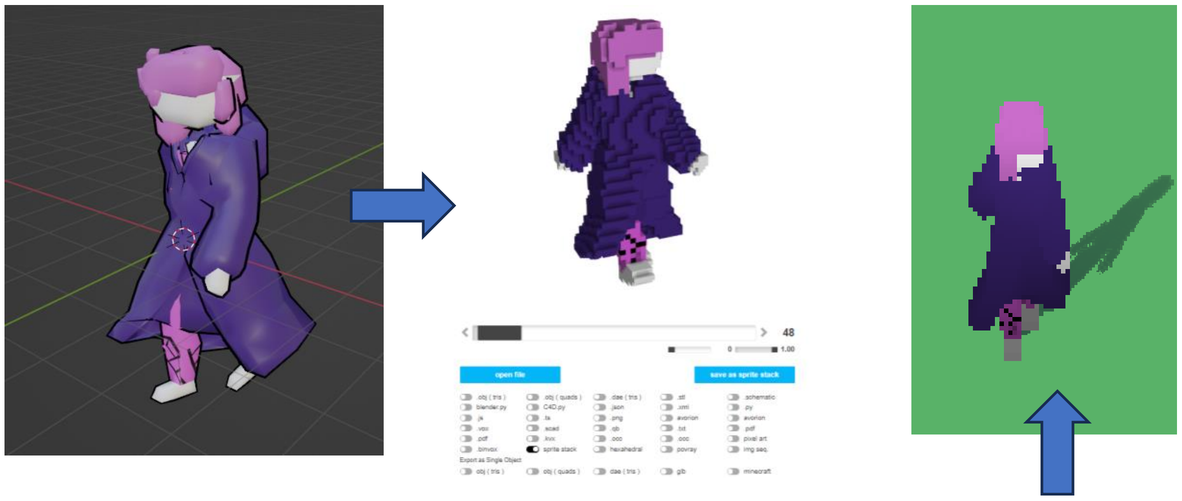
Another effect in the game is the blur. Photos of miniature things have a strong depth of field, where the subject is in focus but almost immediately everything else is blurred. I wanted to recreate this effect. I do this by first finding the sprites that are at the bottom and the top of the screen by taking their (x,y,z) positions and rotating it through a rotation matrix to find what their (x,y,z) positions are in relation to the camera.

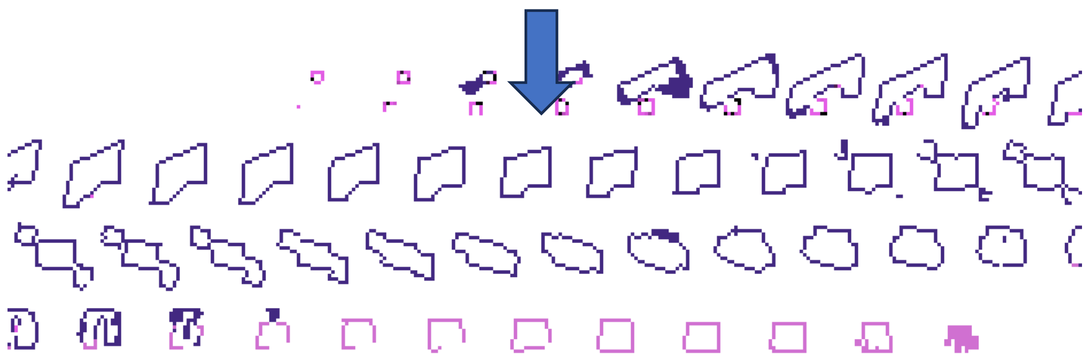
$$\begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} \cos(-camA) & -\sin(-camA) \\ \sin(-camA) & \cos(-camA) \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

This is then mixed with a motion blur filter that blurs the border of the window outwards.



The more detailed objects were modelled in Blender 3.4 such as the house and the player then converted to sliced images which I can then use in Game Maker using a website called Voxeliser.





The light on each object was determined by its distance to a light source and the time of day:

$$light = \frac{20}{distance} + 0.75 + 0.25\sin(time\ of\ day)$$

```
var light = 20/light_distance+0.75+0.25*sin(global.time_of_day*2*pi/2400);
```

Mixing those things together along with a smoothed camera (using x = x ×0.9+new_x×0.1); self-correcting terrain (by defining a rectangle and it smooths the corners); particles (by using drawing images straight from a list); and water distortion (by offsetting every image by a sine function when underwater). You get a program that is just made of 2D images but creates a 3D world.

It was all made by me over the course of about 50 hours over several months. There is about 2,000 lines of code.

Thank you to the Oliphant Science Award organisers, sponsors, judges for supporting students. As well as my teachers and the school for the opportunity to learn and develop programming.

Bibliography

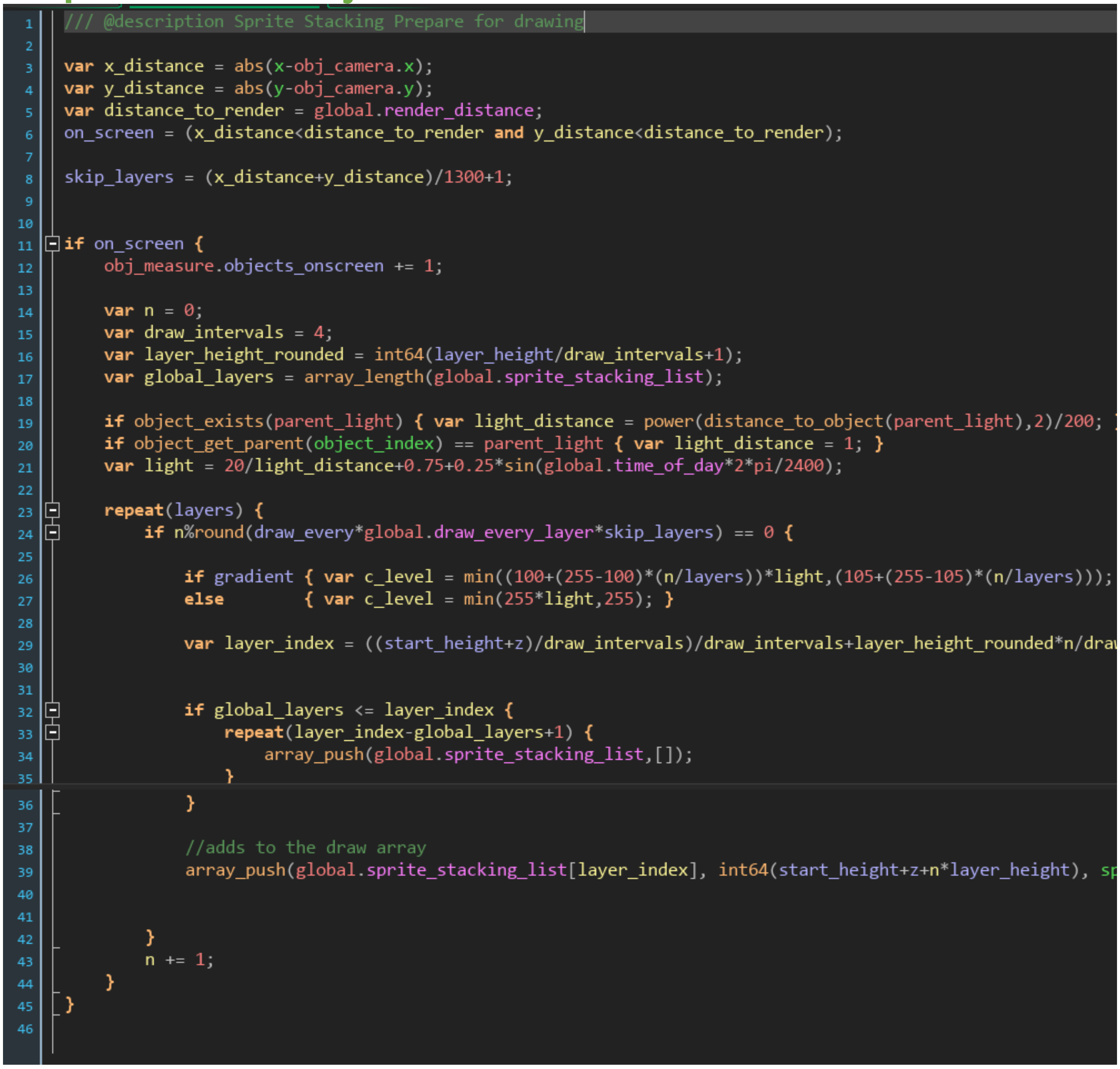
HeartBeast (16 September 2017) *GameMaker Studio 2 - 3D Racecar - Sprite Stacking*, YouTube, accessed 2 April 2023. https://www.youtube.com/watch?v=sNgb3guusMA

Gizmo199 (23 November 2019) *Fake 3d | Gamemaker Studio 2 | Depth issue solved*, YouTube, accessed 2 April 2023. https://www.youtube.com/watch?v=iLdGVN4h_tY

Gizmo199 X (5 September 2020) *Sprite Stacking 3D | Smooth 360 camera & Basic setup | Game Maker Studio 2.3 tutorial [EP 1]*, YouTube, accessed 2 April 2023. https://www.youtube.com/watch?v=VIDN-nG3EOU

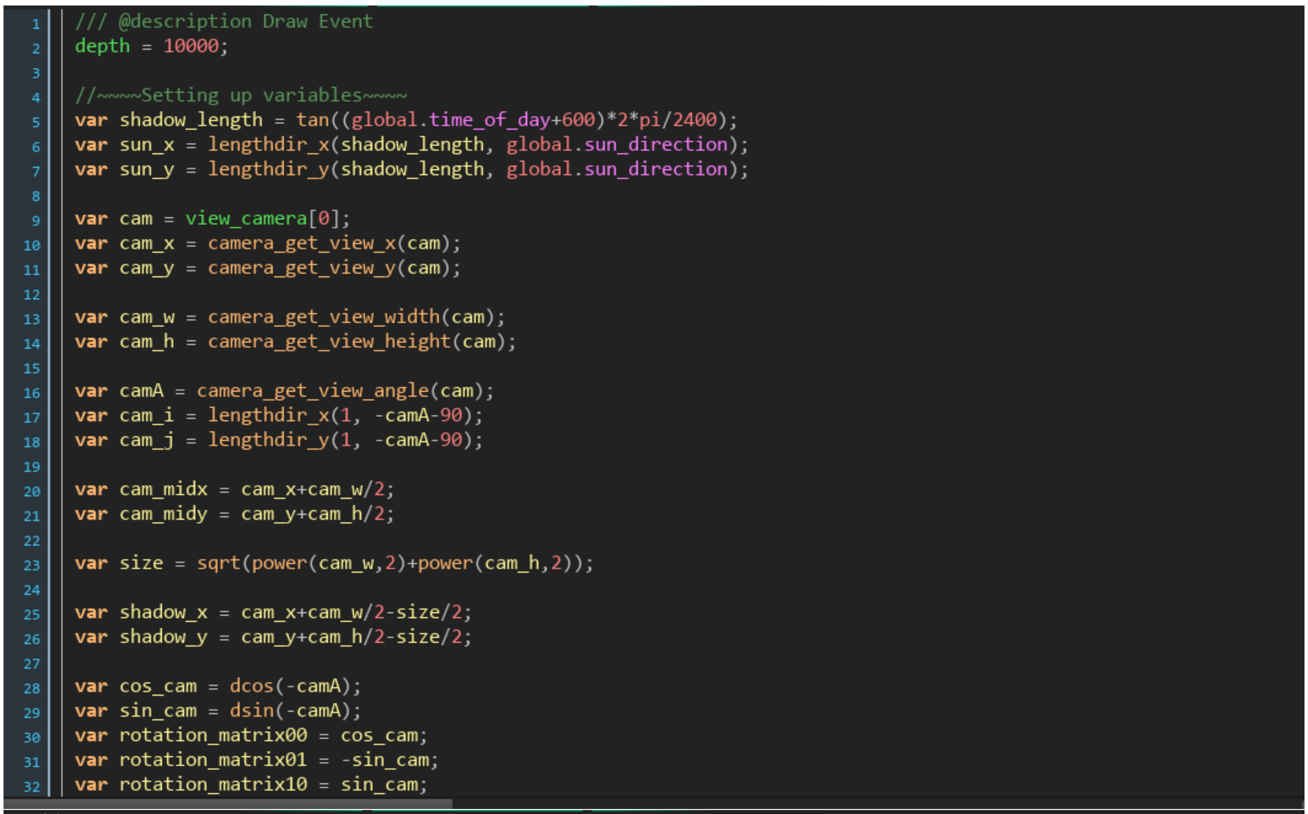
Noonz (10 March 2021) *2d Sprite Stacking - Example/Tutorial*, YouTube, accessed 2 April 2023. https://www.youtube.com/watch?v=1xFVVvoT6eg

Step Event for Objects to be drawn

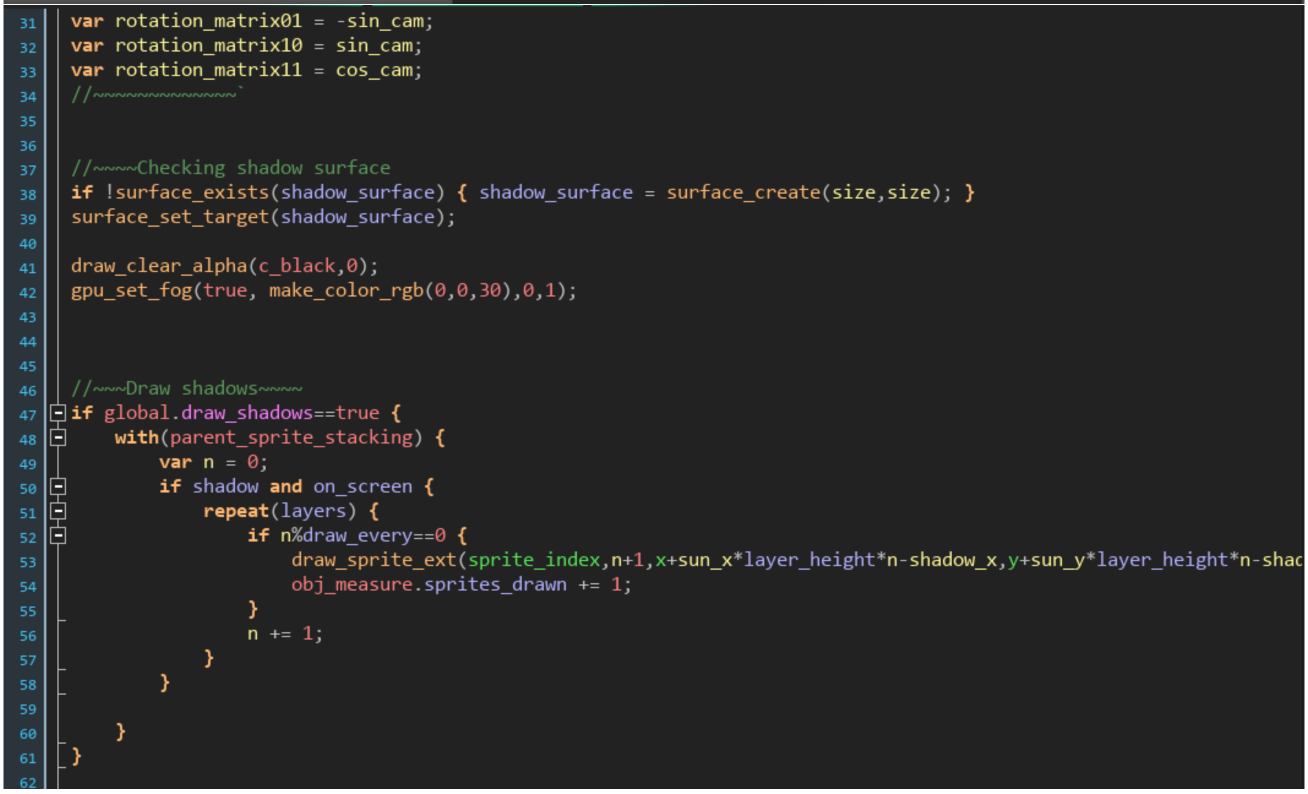


```
/// @description Sprite Stacking Prepare for drawing

var x_distance = abs(x-obj_camera.x);
var y_distance = abs(y-obj_camera.y);
var distance_to_render = global.render_distance;
on_screen = (x_distance<distance_to_render and y_distance<distance_to_render);

skip_layers = (x_distance+y_distance)/1300+1;


if on_screen {
    obj_measure.objects_onscreen += 1;

    var n = 0;
    var draw_intervals = 4;
    var layer_height_rounded = int64(layer_height/draw_intervals+1);
    var global_layers = array_length(global.sprite_stacking_list);

    if object_exists(parent_light) { var light_distance = power(distance_to_object(parent_light),2)/200;
    if object_get_parent(object_index) == parent_light { var light_distance = 1; }
    var light = 20/light_distance+0.75+0.25*sin(global.time_of_day*2*pi/2400);

    repeat(layers) {
        if n%round(draw_every*global.draw_every_layer*skip_layers) == 0 {

            if gradient { var c_level = min((100+(255-100)*(n/layers))*light,(105+(255-105)*(n/layers)));
            else        { var c_level = min(255*light,255); }

            var layer_index = ((start_height+z)/draw_intervals)/draw_intervals+layer_height_rounded*n/dra


            if global_layers <= layer_index {
                repeat(layer_index-global_layers+1) {
                    array_push(global.sprite_stacking_list,[]);
                }
            }

            //adds to the draw array
            array_push(global.sprite_stacking_list[layer_index], int64(start_height+z+n*layer_height), sp


        }
        n += 1;
    }
}
```

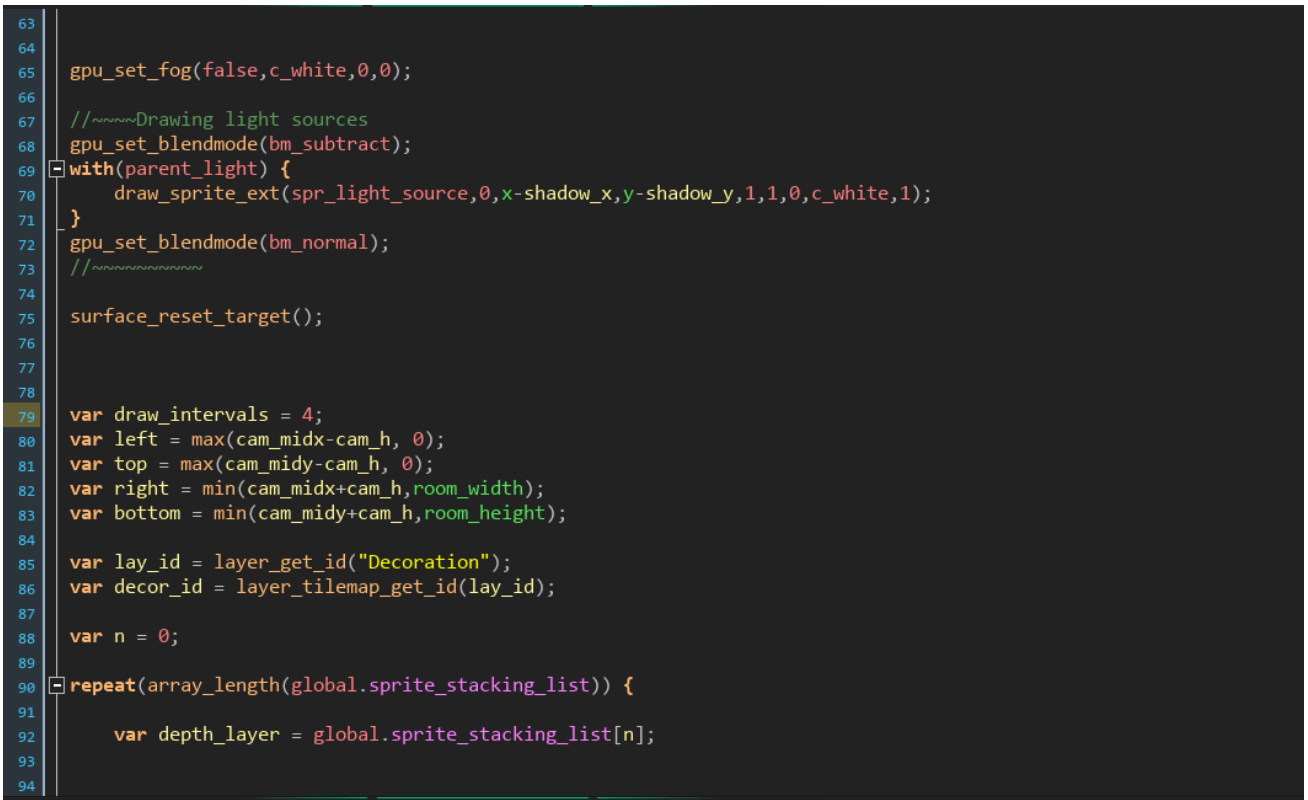
The Draw Event which draws EVERTHING to the screen



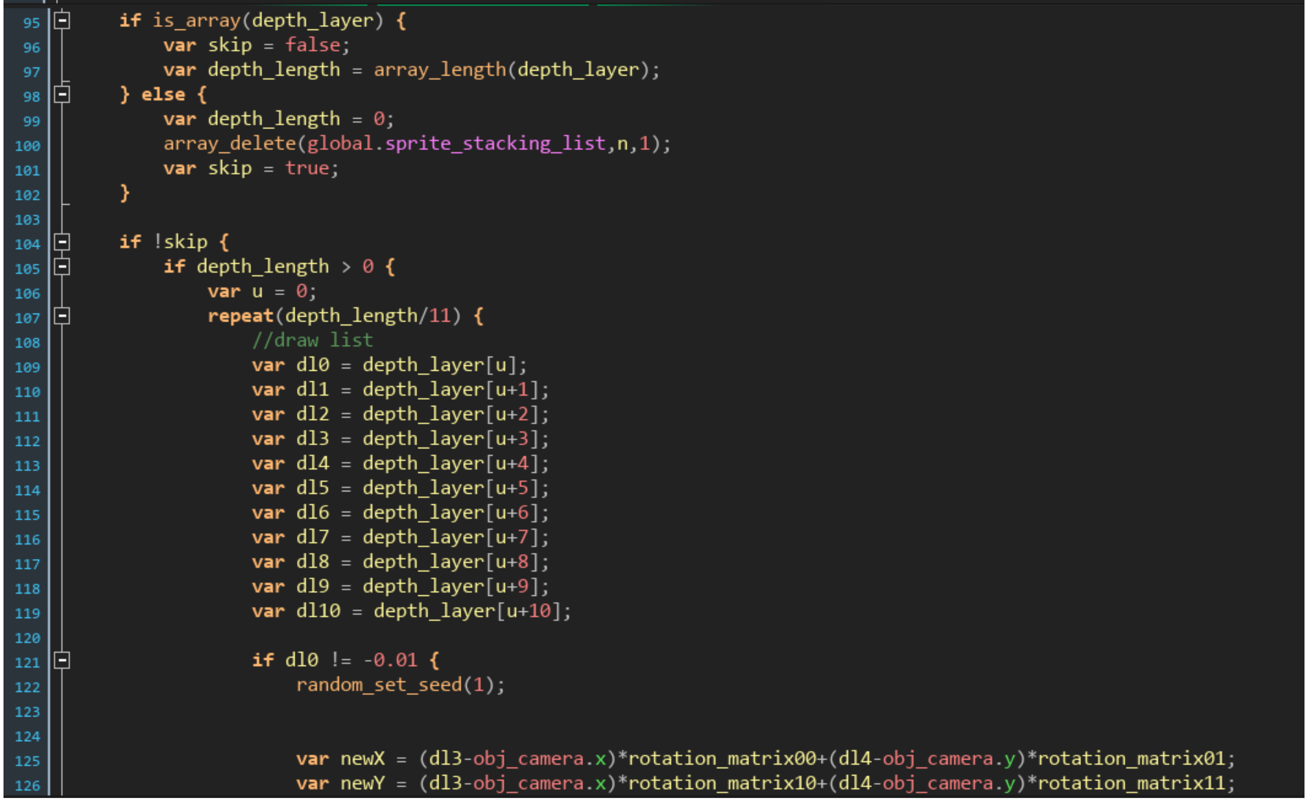
```
/// @description Draw Event
depth = 10000;

//~~~~Setting up variables~~~~
var shadow_length = tan((global.time_of_day+600)*2*pi/2400);
var sun_x = lengthdir_x(shadow_length, global.sun_direction);
var sun_y = lengthdir_y(shadow_length, global.sun_direction);

var cam = view_camera[0];
var cam_x = camera_get_view_x(cam);
var cam_y = camera_get_view_y(cam);

var cam_w = camera_get_view_width(cam);
var cam_h = camera_get_view_height(cam);

var camA = camera_get_view_angle(cam);
var cam_i = lengthdir_x(1, -camA-90);
var cam_j = lengthdir_y(1, -camA-90);

var cam_midx = cam_x+cam_w/2;
var cam_midy = cam_y+cam_h/2;

var size = sqrt(power(cam_w,2)+power(cam_h,2));

var shadow_x = cam_x+cam_w/2-size/2;
var shadow_y = cam_y+cam_h/2-size/2;

var cos_cam = dcos(-camA);
var sin_cam = dsin(-camA);
var rotation_matrix00 = cos_cam;
var rotation_matrix01 = -sin_cam;
var rotation_matrix10 = sin_cam;
```



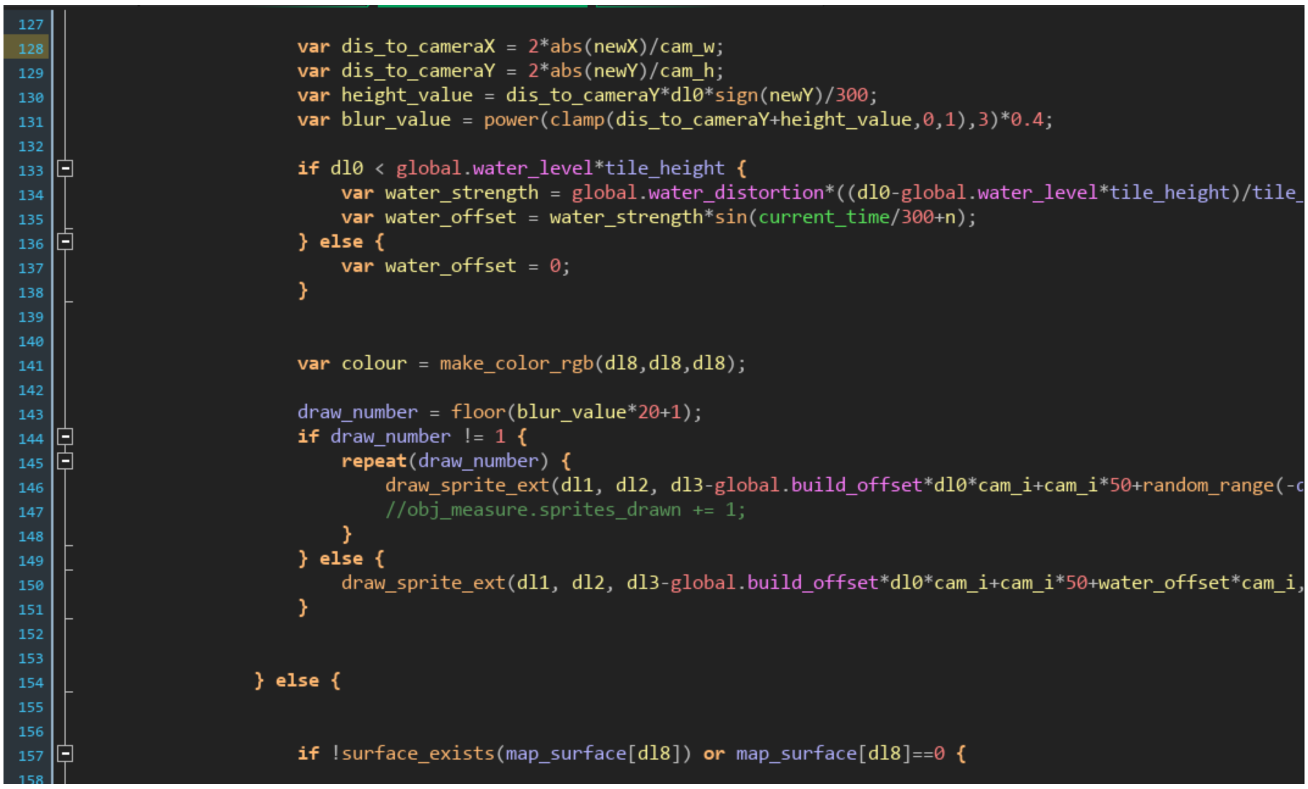
```
var rotation_matrix01 = -sin_cam;
var rotation_matrix10 = sin_cam;
var rotation_matrix11 = cos_cam;
//~~~~~~~~~~~~~~~`


//~~~~Checking shadow surface
if !surface_exists(shadow_surface) { shadow_surface = surface_create(size,size); }
surface_set_target(shadow_surface);

draw_clear_alpha(c_black,0);
gpu_set_fog(true, make_color_rgb(0,0,30),0,1);



//~~~Draw shadows~~~~
if global.draw_shadows==true {
    with(parent_sprite_stacking) {
        var n = 0;
        if shadow and on_screen {
            repeat(layers) {
                if n%draw_every==0 {
                    draw_sprite_ext(sprite_index,n+1,x+sun_x*layer_height*n-shadow_x,y+sun_y*layer_height*n-shad
                    obj_measure.sprites_drawn += 1;
                }
                n += 1;
            }
        }

    }
}
```



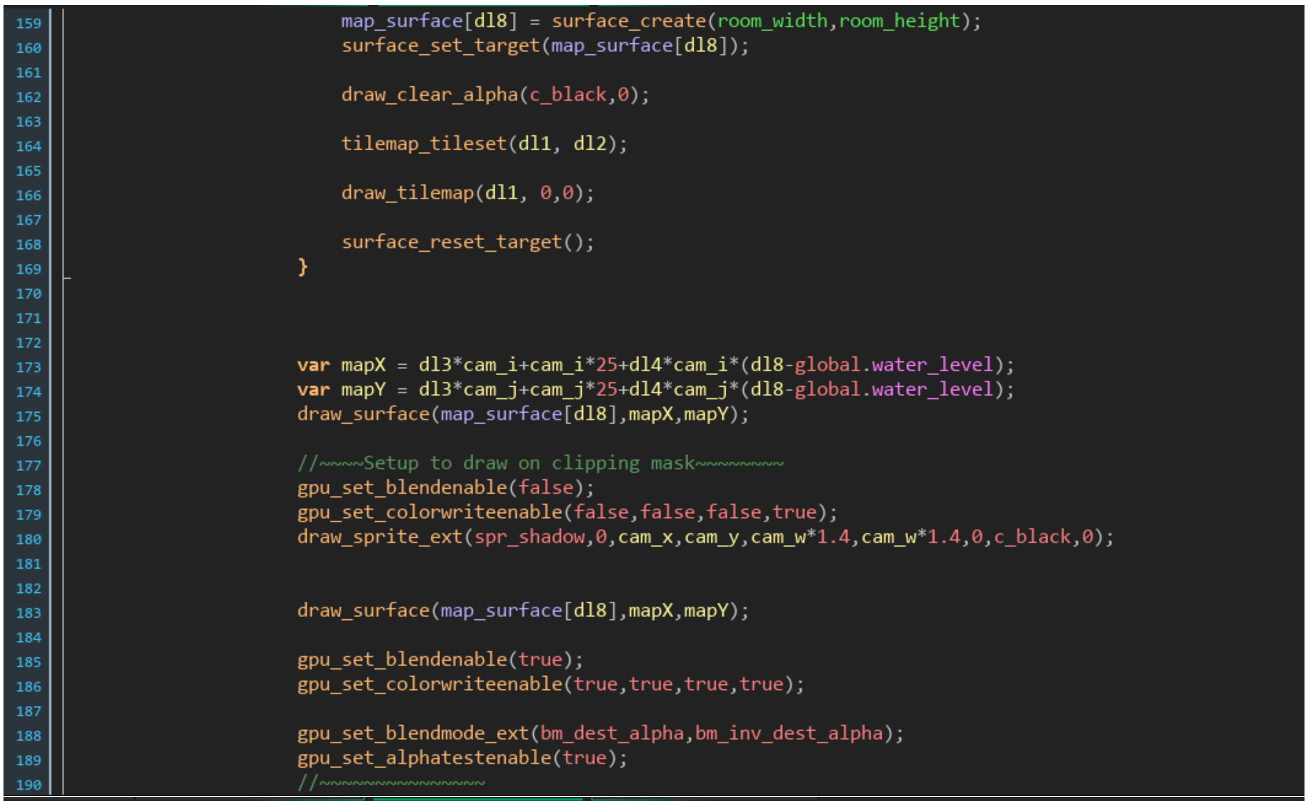
```


gpu_set_fog(false,c_white,0,0);

//~~~~Drawing light sources
gpu_set_blendmode(bm_subtract);
with(parent_light) {
    draw_sprite_ext(spr_light_source,0,x-shadow_x,y-shadow_y,1,1,0,c_white,1);
}
gpu_set_blendmode(bm_normal);
//~~~~~~~~~~~

surface_reset_target();



var draw_intervals = 4;
var left = max(cam_midx-cam_h, 0);
var top = max(cam_midy-cam_h, 0);
var right = min(cam_midx+cam_h,room_width);
var bottom = min(cam_midy+cam_h,room_height);

var lay_id = layer_get_id("Decoration");
var decor_id = layer_tilemap_get_id(lay_id);

var n = 0;

repeat(array_length(global.sprite_stacking_list)) {

    var depth_layer = global.sprite_stacking_list[n];

```



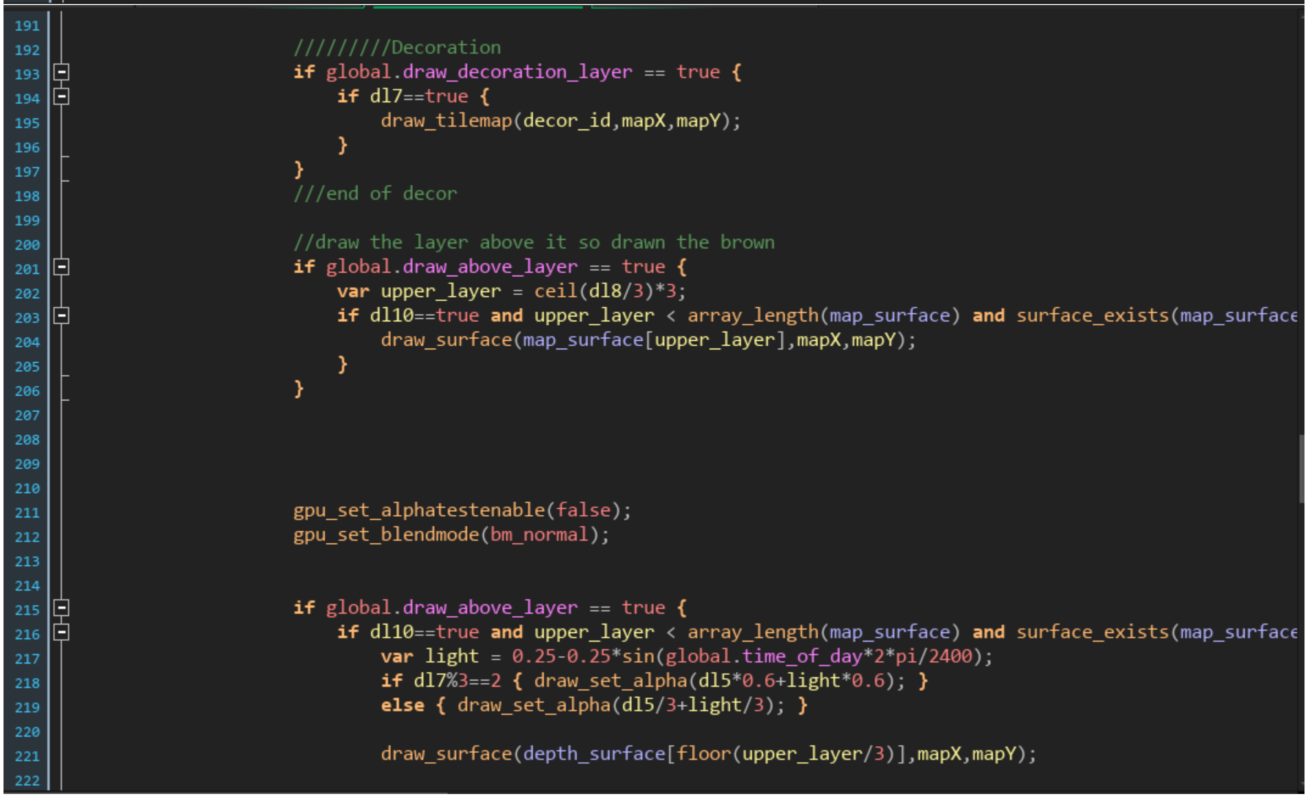
```
    if is_array(depth_layer) {
        var skip = false;
        var depth_length = array_length(depth_layer);
    } else {
        var depth_length = 0;
        array_delete(global.sprite_stacking_list,n,1);
        var skip = true;
    }

    if !skip {
        if depth_length > 0 {
            var u = 0;
            repeat(depth_length/11) {
                //draw list
                var dl0 = depth_layer[u];
                var dl1 = depth_layer[u+1];
                var dl2 = depth_layer[u+2];
                var dl3 = depth_layer[u+3];
                var dl4 = depth_layer[u+4];
                var dl5 = depth_layer[u+5];
                var dl6 = depth_layer[u+6];
                var dl7 = depth_layer[u+7];
                var dl8 = depth_layer[u+8];
                var dl9 = depth_layer[u+9];
                var dl10 = depth_layer[u+10];

                if dl0 != -0.01 {
                    random_set_seed(1);


                    var newX = (dl3-obj_camera.x)*rotation_matrix00+(dl4-obj_camera.y)*rotation_matrix01;
                    var newY = (dl3-obj_camera.x)*rotation_matrix10+(dl4-obj_camera.y)*rotation_matrix11;
```



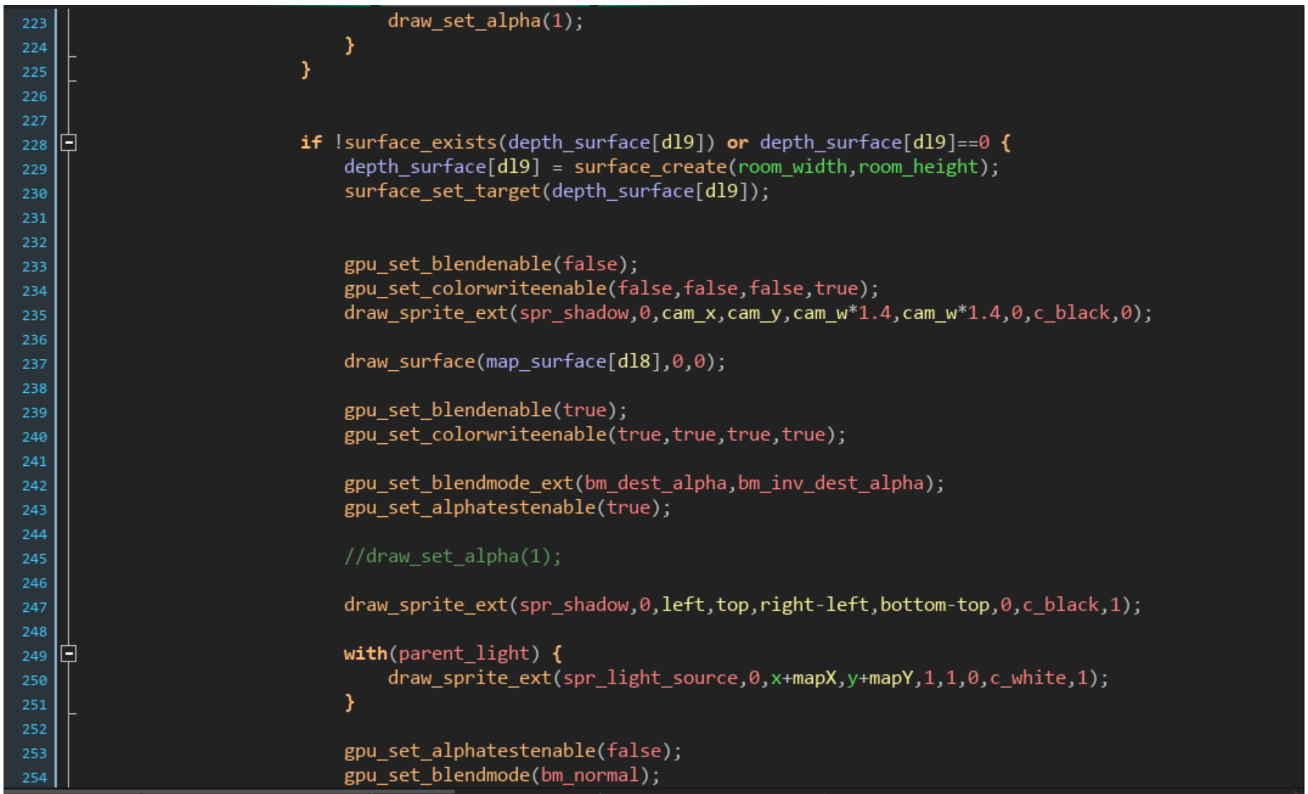
```
    var dis_to_cameraX = 2*abs(newX)/cam_w;
    var dis_to_cameraY = 2*abs(newY)/cam_h;
    var height_value = dis_to_cameraY*dl0*sign(newY)/300;
    var blur_value = power(clamp(dis_to_cameraY+height_value,0,1),3)*0.4;

    if dl0 < global.water_level*tile_height {
        var water_strength = global.water_distortion*((dl0-global.water_level*tile_height)/tile_
        var water_offset = water_strength*sin(current_time/300+n);
    } else {
        var water_offset = 0;
    }


    var colour = make_color_rgb(dl8,dl8,dl8);

    draw_number = floor(blur_value*20+1);
    if draw_number != 1 {
        repeat(draw_number) {
            draw_sprite_ext(dl1, dl2, dl3-global.build_offset*dl0*cam_i+cam_i*50+random_range(-c
            //obj_measure.sprites_drawn += 1;
        }
    } else {
        draw_sprite_ext(dl1, dl2, dl3-global.build_offset*dl0*cam_i+cam_i*50+water_offset*cam_i,
    }


} else {


    if !surface_exists(map_surface[dl8]) or map_surface[dl8]==0 {
```



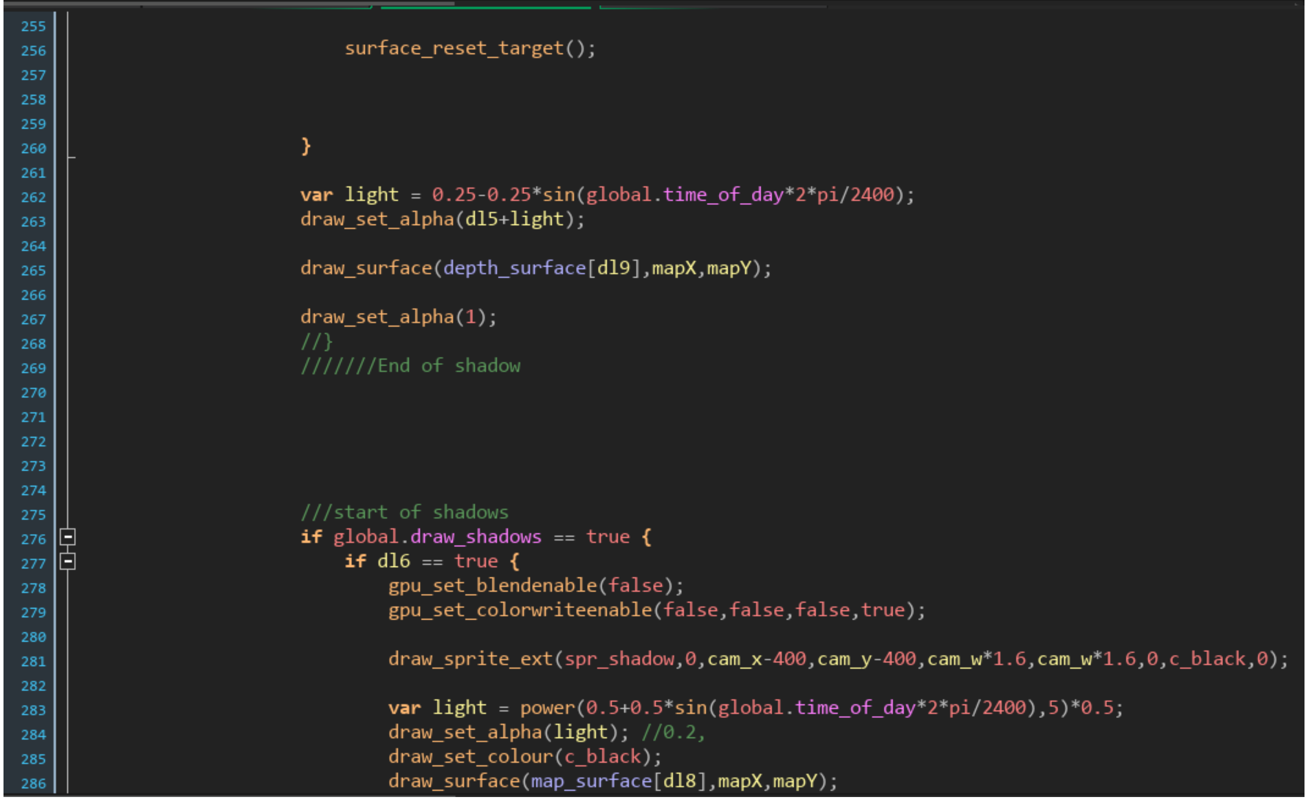
```
    map_surface[dl8] = surface_create(room_width,room_height);
    surface_set_target(map_surface[dl8]);

    draw_clear_alpha(c_black,0);

    tilemap_tileset(dl1, dl2);

    draw_tilemap(dl1, 0,0);

    surface_reset_target();
}



var mapX = dl3*cam_i+cam_i*25+dl4*cam_i*(dl8-global.water_level);
var mapY = dl3*cam_j+cam_j*25+dl4*cam_j*(dl8-global.water_level);
draw_surface(map_surface[dl8],mapX,mapY);

//~~~~Setup to draw on clipping mask~~~~~~~~
gpu_set_blendenable(false);
gpu_set_colorwriteenable(false,false,false,true);
draw_sprite_ext(spr_shadow,0,cam_x,cam_y,cam_w*1.4,cam_w*1.4,0,c_black,0);


draw_surface(map_surface[dl8],mapX,mapY);

gpu_set_blendenable(true);
gpu_set_colorwriteenable(true,true,true,true);

gpu_set_blendmode_ext(bm_dest_alpha,bm_inv_dest_alpha);
gpu_set_alphatestenable(true);
//~~~~~~~~~~~~~~~~~
```



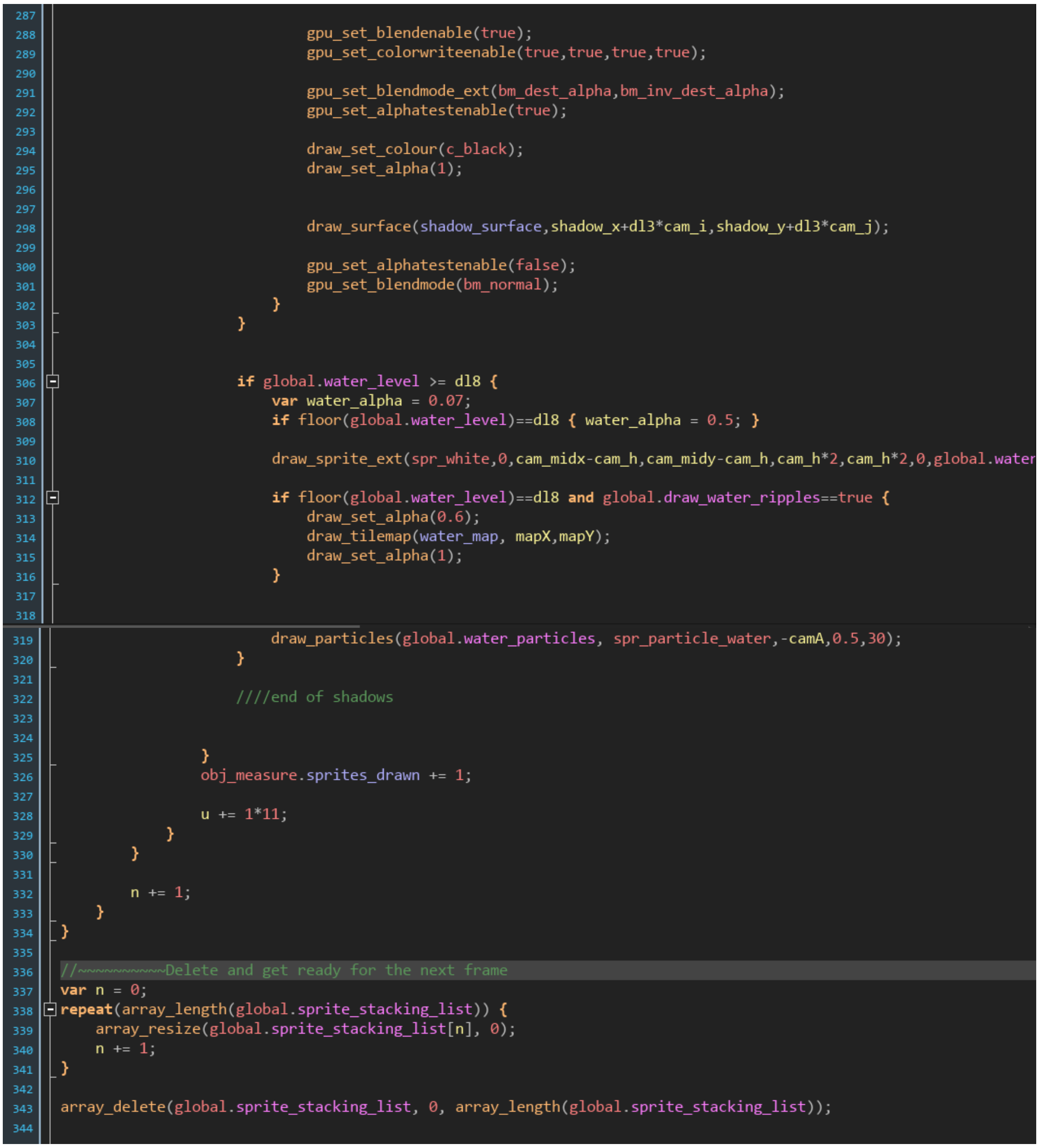
```

/////////Decoration
if global.draw_decoration_layer == true {
    if dl7==true {
        draw_tilemap(decor_id,mapX,mapY);
    }
}
///end of decor

//draw the layer above it so drawn the brown
if global.draw_above_layer == true {
    var upper_layer = ceil(dl8/3)*3;
    if dl10==true and upper_layer < array_length(map_surface) and surface_exists(map_surface
        draw_surface(map_surface[upper_layer],mapX,mapY);
    }
}




gpu_set_alphatestenable(false);
gpu_set_blendmode(bm_normal);


if global.draw_above_layer == true {
    if dl10==true and upper_layer < array_length(map_surface) and surface_exists(map_surface
        var light = 0.25-0.25*sin(global.time_of_day*2*pi/2400);
        if dl7%3==2 { draw_set_alpha(dl5*0.6+light*0.6); }
        else { draw_set_alpha(dl5/3+light/3); }

        draw_surface(depth_surface[floor(upper_layer/3)],mapX,mapY);

```



```
        draw_set_alpha(1);
    }
}


if !surface_exists(depth_surface[dl9]) or depth_surface[dl9]==0 {
    depth_surface[dl9] = surface_create(room_width,room_height);
    surface_set_target(depth_surface[dl9]);


    gpu_set_blendenable(false);
    gpu_set_colorwriteenable(false,false,false,true);
    draw_sprite_ext(spr_shadow,0,cam_x,cam_y,cam_w*1.4,cam_w*1.4,0,c_black,0);

    draw_surface(map_surface[dl8],0,0);

    gpu_set_blendenable(true);
    gpu_set_colorwriteenable(true,true,true,true);

    gpu_set_blendmode_ext(bm_dest_alpha,bm_inv_dest_alpha);
    gpu_set_alphatestenable(true);

    //draw_set_alpha(1);

    draw_sprite_ext(spr_shadow,0,left,top,right-left,bottom-top,0,c_black,1);

    with(parent_light) {
        draw_sprite_ext(spr_light_source,0,x+mapX,y+mapY,1,1,0,c_white,1);
    }

    gpu_set_alphatestenable(false);
    gpu_set_blendmode(bm_normal);
```



```

    surface_reset_target();



}

var light = 0.25-0.25*sin(global.time_of_day*2*pi/2400);
draw_set_alpha(dl5+light);

draw_surface(depth_surface[dl9],mapX,mapY);

draw_set_alpha(1);
//}
///////End of shadow






///start of shadows
if global.draw_shadows == true {
    if dl6 == true {
        gpu_set_blendenable(false);
        gpu_set_colorwriteenable(false,false,false,true);

        draw_sprite_ext(spr_shadow,0,cam_x-400,cam_y-400,cam_w*1.6,cam_w*1.6,0,c_black,0);

        var light = power(0.5+0.5*sin(global.time_of_day*2*pi/2400),5)*0.5;
        draw_set_alpha(light); //0.2,
        draw_set_colour(c_black);
        draw_surface(map_surface[dl8],mapX,mapY);
```



```
                            gpu_set_blendenable(true);
                            gpu_set_colorwriteenable(true,true,true,true);

                            gpu_set_blendmode_ext(bm_dest_alpha,bm_inv_dest_alpha);
                            gpu_set_alphatestenable(true);

                            draw_set_colour(c_black);
                            draw_set_alpha(1);


                            draw_surface(shadow_surface,shadow_x+dl3*cam_i,shadow_y+dl3*cam_j);

                            gpu_set_alphatestenable(false);
                            gpu_set_blendmode(bm_normal);
                        }
                    }


                    if global.water_level >= dl8 {
                        var water_alpha = 0.07;
                        if floor(global.water_level)==dl8 { water_alpha = 0.5; }

                        draw_sprite_ext(spr_white,0,cam_midx-cam_h,cam_midy-cam_h,cam_h*2,cam_h*2,0,global.water

                        if floor(global.water_level)==dl8 and global.draw_water_ripples==true {
                            draw_set_alpha(0.6);
                            draw_tilemap(water_map, mapX,mapY);
                            draw_set_alpha(1);
                        }

                        draw_particles(global.water_particles, spr_particle_water,-camA,0.5,30);
                    }

                    ////end of shadows


                }
                obj_measure.sprites_drawn += 1;

                u += 1*11;
            }
        }

        n += 1;
    }
}

//~~~~~~~~~~~Delete and get ready for the next frame
var n = 0;
repeat(array_length(global.sprite_stacking_list)) {
    array_resize(global.sprite_stacking_list[n], 0);
    n += 1;
}

array_delete(global.sprite_stacking_list, 0, array_length(global.sprite_stacking_list));

```