SASTA OLIPHANT SCIENCE AWARDS

SOUTH AUSTRALIAN SCIENCE TEACHERS ASSOCIATION

Highly Commended

Programming, Apps & Robotics

Year 11 - 12

Angus Hunt

Pembroke School - Middle School

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

Government of South Australia
Department for Education

SASTA South Australian Science Teachers Association

Oliphant almost done - GameMaker

File Edit Build Windows Tools Marketplace Layouts Help Source Control

IDE v2024.14.4.222 Runtime v2024.14.4.268

Windows | GMS2 VM | Default | Default

Inspector

Layers - rm_end

Assets_2

Instances_1

Assets_1

Instances

Background

Inspector: rm_end

Search...

Room

rm_end

Room Settings

Persistent

Clear Display Buffer

3D in gamemaker oliphant version 0.01

10 items

100%

ENG US

12:52 PM 25/07/2026

ideapad Slim 3 (14")

High Active Area Ratio

Privacy Shutter

Intelligent Thermal with FN+Q

eSupport

Lenovo

Programming, Apps & Robotics

Year 11-12

Angus Hunt

Pembroke School – Senior School

Oliphant Science Awards 2026

Entry xxx-xxx Programming, Apps & Robotics Angus Hunt, Year 11

Making 3D environments with GameMaker:

The main point of the game is for the player to get to the end of the map. In order to do this, the player has to walk a large amount of distance in any direction. The landscape throughout the world is uniform with mountains being evenly spaced across. The landscape changes every time the player restarts the game, with things such as the distance between the mountains, the actual size of the mountains and the distance the player has to go to finish the game all being randomised.

Table of the randomised values within the game:

The randomised feature:	Value Min	Value Max	All possible values
The size of Mountains	3	8	3,4,5,6,7,8
Mountain's spacing	12	22	12,14,16,18,20,22
Player vision distance	6	16	6,7,8,9,10,11,12,13,14,15,16
Player movement speed (in 100s of pixels travelled per second/blocks)	3.2	4.0	3.2, 3.4, 3.6, 3.8, 4.0
Distance to the end(in 100s of pixels/blocks)	750	900	750,800,850,900

This game wasn't made for being entertaining like most games, but instead it's more of a template to anyone who wants to look into experimenting with or making a proper game with 3D in Gamemaker. The code throughout the game is clearly commented, so anyone who's looking at it can easily see how different things work and how they can be changed.

How to play the game:

The way that the player can move throughout the game is the same as how players can move through most 3D games. To move forwards, the player can press W or the up-arrow key. To move backwards, the player can press the S key or down arrow key. To move left, the player can press the A or left arrow key. To move right, the player can press the D or right arrow key. The main point of the game is trying to get far away from the spawn point, which will cause the game to end. If the player looks in the exact same direction for in the exact same direction for too long, the player will start spinning. The intention of this is that it makes the player actually move through the terrain of the game, and doesn't just hold down the W key. Some maths was done to make sure that the spinning would stop the player from being able to beat the game, no matter how fast the player is and no matter how small the map is.

The following equation was used to determine how far the player could actually make it if they just moved forwards.

$$(Speed\ per\ second \times time\ before\ the\ player\ starts\ turning) + \left(\frac{speed\ per\ second}{degree\ change\ per\ second\ in\ radians}\right)$$

An example for how this would normally look is.

$$(3.6 \times 45) + \left(\frac{3.6}{3° \times \frac{\pi}{180}}\right) = 230.8$$

Speed	Distance from start (blocks)	Radius (blocks)	Total furthest distance from the start	Percentage of the way to the minimum distance (750 blocks)
3.2	144	61.1	205.1	27.3%
3.4	153	64.9	217.9	29.1%
3.6	162	68.8	230.8	30.8%

3.8	171	72.6	243.6	32.5%
4	180	76.4	256.4	34.2%

Game Link: Oliphant Game finished | GX Games

Development:

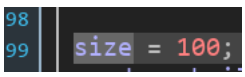
The game has been created using the most recent version Game Maker Studio 2. The game is my work, but some credits are owed to the people who have helped me learn more about and improve my skills in Gamemaker throughout my schooling experience. (Mr Izzo, Stuart Vass, Archibald Nagel, Ms Robbins). Some credit is also given to the youtuber (**DragoniteSpam**) who created the tutorials to and inspired me to start experimenting with 3D in Gamemaker.

How the game actually works:

The way that the terrain is generated is using cubes that are 100 pixels in size. The cubes are made from 2 triangles. Triangles were selected because they are the only shape where you can cut them into 2 pieces, where the pieces are always that of the same shape.

Here is an example of how the top face of one of the cubes being generated looks like.

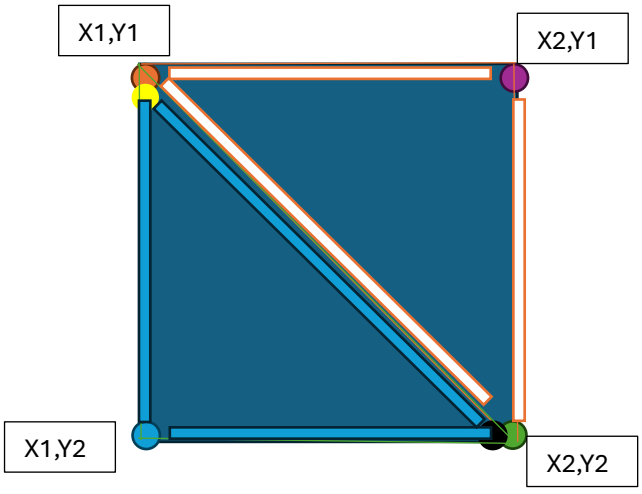
In this case size is 100 pixels



```
113
114     var x1 = x;
115     var x2 = x + size;
116
117     var y1 = y;
118     var y2 = y + size;
119
120     var z1 = z;
121     var z2 = z - size;
122
```

```
123 //Top face
124 vertex_add_point(vbuffer, x1, y1, z1, 0,0,1, 0,0, c_white, 1);
125 vertex_add_point(vbuffer, x2, y1, z1, 0,0,1, 0,0, c_white, 1);
126 vertex_add_point(vbuffer, x2, y2, z1, 0,0,1, 0,0, c_white, 1);
127 vertex_add_point(vbuffer, x2, y2, z1, 0,0,1, 0,0, c_white, 1);
128 vertex_add_point(vbuffer, x1, y2, z1, 0,0,1, 0,0, c_white, 1);
129 vertex_add_point(vbuffer, x1, y1, z1, 0,0,1, 0,0, c_white, 1);
```

Visualization of how it works:



Bibliography:

Getting Started with 3D in GameMaker 2020, Youtube.com, viewed 10 May 2026, https://www.youtube.com/watch?v=ojfN--tdSNM&list=PL_hT--4HOvrcML9uqHe4fwBVTm650Vy3V

Vertex Formats and Vertex Buffers - 3D Games in GameMaker 2020, Youtube.com, viewed 10 May 2026, https://www.youtube.com/watch?v=DH0gjTemUxg&list=PL_hT--4HOvrcML9uqHe4fwBVTm650Vy3V&index=2

Using Textures - 3D Games in GameMaker 2020, Youtube.com, viewed 10 May 2026, https://www.youtube.com/watch?v=XLpcn0XQJj8&list=PL_hT--4HOvrcML9uqHe4fwBVTm650Vy3V&index=3

First Person Cameras - 3D Games in GameMaker 2020, Youtube.com, viewed 10 May 2026, https://www.youtube.com/watch?v=rrx6EW2tUlM&list=PL_hT--4HOvrcML9uqHe4fwBVTm650Vy3V&index=4

Third Person Cameras - 3D Games in GameMaker 2020, Youtube.com, viewed 11 May 2026, https://www.youtube.com/watch?v=JdsxJqC4S4o&list=PL_hT--4HOvrcML9uqHe4fwBVTm650Vy3V&index=5

Importing 3D Object Files in GameMaker 2024, Youtube.com, viewed 11 May 2026, https://www.youtube.com/watch?v=rKLp38gI1HI&list=PL_hT--4HOvrcML9uqHe4fwBVTm650Vy3V&index=6

All the code:

Camera Create event:

```
//randomises the values with a randomised seeds
randomise(){
    //randomises the size of tehe tiles
 tiles_s = choose(0, 1,2,3,4,5,6,7,8,9,10)
 //Randomise the distance the player has to go
 game_distance = choose(75000,80000,85000,90000)
 }
 //14,17,19
if tiles_s = 0{

 tiles_away  = 6;
 }
if tiles_s = 1{
 tiles_away  = 7;
 }
if tiles_s = 2{
 tiles_away  = 8;
 }
if tiles_s = 3{

 tiles_away  = 9;
 }
if tiles_s = 4{
 tiles_away  = 10;
 }
if tiles_s = 5{
 tiles_away  = 11;
```

```
}
if tiles_s = 5{
tiles_away  = 11;
}
if tiles_s = 6{

tiles_away  = 12;
}
if tiles_s = 7{
tiles_away  = 13;
}
if tiles_s = 8{
tiles_away  = 14;
}
if tiles_s = 9{

tiles_away  = 15;
}
if tiles_s = 10{
tiles_away  = 16;
}

//Draw objects that are infront of objects behind them over the objects that are behind them
gpu_set_ztestenable(true);
gpu_set_zwriteenable(true);

//Vertex format
```

```
//Vertex format
vertex_format_begin();
vertex_format_add_position_3d();
vertex_format_add_normal();
vertex_format_add_texcoord();
vertex_format_add_colour();
//Varible definitons (not entirly essential, but usign them instead of the
//full thing saves time).
vertex_format = vertex_format_end();
vbuffer = vertex_create_buffer();
vertex_begin(vbuffer, vertex_format);
//Randomize the game seed, this basically makes it so all the things
//that can happen with the game are truly random
//6 possible mountain distance amounts (0-5)
global.mountain_type = choose(0, 1,2,3,4,5);
//Spacing between mountains
//Mountain centers are 12 cubes away from each other
mountain_spacing = 1200;
//Mountain centers are 14 cubes away from each other
mountain_spacing2 = 1400;
//Mountain centers are 16 cubes away from each other
mountain_spacing3 = 1600;
//Mountain centers are 18 cubes away from each other
mountain_spacing4 = 1800;
//Mountain centers are 20 cubes away from each other
mountain_spacing5 = 2000;
//Mountain centers are 22 cubes away from each other
```

```
mountain_spacing5 = 2000;
//Mountain centers are 22 cubes away from each other
mountain_spacing6 = 2200;
//Define where the player strts at
start_x = Player.x;
start_y = Player.y;
xx = start_x;
yy = start_y;
center_x = choose(round(xx / mountain_spacing) * mountain_spacing,(round(xx /
mountain_spacing2) * mountain_spacing2));
center_y = choose(round(yy / mountain_spacing) * mountain_spacing,(round(yy /
mountain_spacing2) * mountain_spacing2));
//instance_create_depth(0,0,0,Player);
xdist = 70;
ydist = 70;
size = 100;
z1 = -100
z2 = z1 - size;


height2 = choose(3,4,5,6,7) + 1;
var max_height =0;
height_chosen = false;
time = 0;

vertex_begin(vbuffer, vertex_format);

var tile_size = 100;
```

```
var tile_size = 100;




for (var xx = 0; xx < room_width /2; xx += tile_size)
{
    for (var yy = 0; yy < room_height; yy += tile_size)
    {
        vertex_add_cube(
            vbuffer,
            xx,
            yy,
            200,
            tile_size
        );
    }
}
dist = 300;
function vertex_add_cube(vbuffer, x, y, z, size, tex)
{


    var x1 = x;
    var x2 = x + size;

    var y1 = y;
    var y2 = y + size;
```

```
    var y1 = y;
    var y2 = y + size;

    var z1 = z;
    var z2 = z - size;
//Top face
vertex_add_point(vbuffer, x1, y1, z1, 0,0,1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y1, z1, 0,0,1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z1, 0,0,1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z1, 0,0,1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y2, z1, 0,0,1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y1, z1, 0,0,1, 0,0, c_white, 1);
//Bottem Face
vertex_add_point(vbuffer, x1, y1, z1 - 100, 0,0,-1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y1, z1 - 100, 0,0,-1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z1 - 100, 0,0,-1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z1 - 100, 0,0,-1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y2, z1 - 100, 0,0,-1, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y1, z1 - 100, 0,0,-1, 0,0, c_white, 1);
//Left Face
vertex_add_point(vbuffer, x1, y1, z1, -1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y2, z1, -1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y2, z2, -1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y2, z2, -1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y1, z2, -1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y1, z1, -1,0,0, 0,0, c_white, 1);
//Right Face
vertex_add_point(vbuffer, x2, y1, z1, 1,0,0, 0,0, c_white, 1);
```

```
//Right Face
vertex_add_point(vbuffer, x2, y1, z1, 1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z1, 1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z2, 1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z2, 1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y1, z2, 1,0,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y1, z1, 1,0,0, 0,0, c_white, 1);
//Front Face
vertex_add_point(vbuffer, x1, y1, z1, 0,-1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y1, z1, 0,-1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y1, z2, 0,-1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y1, z2, 0,-1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y1, z2, 0,-1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y1, z1, 0,-1,0, 0,0, c_white, 1);
//Back face
vertex_add_point(vbuffer, x1, y2, z1, 0,1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z1, 0,1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z2, 0,1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x2, y2, z2, 0,1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y2, z2, 0,1,0, 0,0, c_white, 1);
vertex_add_point(vbuffer, x1, y2, z1, 0,1,0, 0,0, c_white, 1);
}

```

Camera Step event:

```
//The distance between different cubes
var tile_size = 100;
//If reloading (every frame)m
if (needs_rebuild = true)
{
vertex_begin(vbuffer, vertex_format);
// If the tiles away variable isnt defined for some reason
//or another, it is set to 19 as a backup to avoid the game from crashing.
if (!variable_instance_exists(id, "tiles_away"))
{
    tiles_away = 19;
}
//If the tiles away varible does existance, define
//the loading distance as the size of the tiles (100) times
//the render distance (tiles away)
if (variable_instance_exists(id, "tiles_away"))
{
var load_distance = tile_size * tiles_away;
}
//The maximum height that a mountain can be,

//Defines how far the map goes
var start_x = floor((Player.x - load_distance) / tile_size) * tile_size;
var end_x   = floor((Player.x + load_distance) / tile_size) * tile_size;
var start_y = floor((Player.y - load_distance) / tile_size) * tile_size;
var end_y   = floor((Player.y + load_distance) / tile_size) * tile_size;

for (var xx = start_x; xx <= end_x; xx += tile_size)
```

```

for (var xx = start_x; xx <= end_x; xx += tile_size)
{
    for (var yy = start_y; yy <= end_y; yy += tile_size)
    {
        var dx = xx - Player.x;
        var dy = yy - Player.y;

        // circular loading
        if (dx * dx + dy * dy <= load_distance * load_distance)
        {


    // If it is the first type of Mountian
    //12 blocks between mountains
      if (global.mountain_type == 0)
{



    center_x = (round(xx / mountain_spacing) * mountain_spacing
    );

    center_y = round(yy / mountain_spacing) * mountain_spacing;
}
//If it is the second type of Mountain,
//14 blocks between the centers mountains
else if (global.mountain_type == 1)
```

```
//14 blocks between the centers mountains
else if (global.mountain_type == 1)
{
    center_x = round(xx / mountain_spacing2) * mountain_spacing2;
    center_y = round(yy / mountain_spacing2) * mountain_spacing2;
}
//If it is the third type of Mountain,
//16 blocks between the centers mountains
else if (global.mountain_type == 2)
{
     center_x = round(xx / mountain_spacing3) * mountain_spacing3;
    center_y = round(yy / mountain_spacing3) * mountain_spacing3;
}
//If it is the fourth type of Mountain,
//16 blocks between the centers mountains

else if (global.mountain_type == 3)
{
     center_x = round(xx / mountain_spacing4) * mountain_spacing4;
    center_y = round(yy / mountain_spacing4) * mountain_spacing4;
}
//If it is the fifth type of Mountain,
//18 blocks between the centers mountains
else if (global.mountain_type == 4)
{
     center_x = round(xx / mountain_spacing5) * mountain_spacing5;
    center_y = round(yy / mountain_spacing5) * mountain_spacing5;
}
//If it is the sixth type of Mountain,
```

```
//If it is the sixth type of Mountain,
//20 blocks between the centers mountains
else if (global.mountain_type == 5)
{
     center_x = round(xx / mountain_spacing6) * mountain_spacing6;
    center_y = round(yy / mountain_spacing6) * mountain_spacing6;
}
//If it is the seventh type of Mountain,
//22 blocks between the centers mountains
else if (global.mountain_type == 6)
{
     center_x = round(xx / mountain_spacing6) * mountain_spacing6;
    center_y = round(yy / mountain_spacing6) * mountain_spacing6;
}


            //Defines the distance from mountain center
            var mdx = xx - center_x;
            var mdy = yy - center_y;

            var dist = point_distance(0, 0, mdx, mdy);

            //Piryamid falloff
            var height = height2 - floor(dist / 120);

            //clamps the height
            height = clamp(height, 1, height2);
```

```
108
109
110             //Stacks the cubes upward
111             for (var h = 0; h < height; h++)
112
113             {
114                 vertex_add_cube(
115                     vbuffer,
116                     xx,
117                     yy,
118                     200 - (h * tile_size),
119                     tile_size
120                 );
121             }
122         }
123     }
124 }
125
126 //Ends the vertex
127 vertex_end(vbuffer);
128 }
129
```

Camera Draw event:

```
1  //Sets the skybox as black (can be any colour)
2  draw_clear(c_black);
3  //Defines the camera
4  var camera = camera_get_active();
5  //Defines zfrom and yfrom
6  var xfrom = Player.x;
7  var yfrom = Player.y;
8  //zfrom is slightly different to the other two, sinceit isnt an inbuilt varibl
9  //e like y and x, and could be called things like height or depth
10 var zfrom = Player.z;
11 //Are controlled by the playerd look direction,
12 var xto = xfrom + dcos(Player.look_dir);
13 var yto = yfrom - dsin(Player.look_dir);
14 //Isn't controlled  by the player look direction,since you can't really look up to go up.
15 var zto = zfrom - dsin(Player.look_pitch);
16 //Sets camera view etc
17 camera_set_view_mat(camera,matrix_build_lookat(xfrom,yfrom,zfrom,xto,yto,zto,0,0,1));
18 camera_set_proj_mat(camera, matrix_build_projection_perspective_fov
19 (60, window_get_width() / window_get_height(),1,32000));
20 camera_apply(camera);
21
22 //Draws everything
23 vertex_submit(vbuffer, pr_trianglelist, sprite_get_texture(Sprite1,0));
24
```

Function:

```
function vertex_add_point(){


    var vbuffer = argument0;
var xx = argument1;
var yy = argument2;
var zz = argument3;
var nx= argument4;
var ny = argument5;
var nz= argument6;
var utex= argument7;
var vtex= argument8;
var color = argument9;
var alpha = argument10;

vertex_position_3d(vbuffer,xx,yy,zz);
vertex_normal(vbuffer, nx,ny,nz);
vertex_texcoord(vbuffer,utex,vtex);
vertex_colour(vbuffer,color,alpha);
}


```

Player create event:

```
// Z position of the player, since blocks technically spawn at -200, the player camera is
//-136 pixels, or about 1.4 blocks above the ground
z = -64;
//Start off looking in 0 degrees horozontally
look_dir = 0;
//sets the orignal look directio to the current look direction (0)
old_look_dir = look_dir;
//Sets moving as false
moving = false;
//Start off looking 0 degrees vertically
look_pitch = 0;
//Create the camera
instance_create_layer(x,y,"Instances",Camera)
//Randomise the seed, this basically makes it so randomised values are actually
//randomised between different loadings of the game (important for the camera)
randomise();
//Sets the amount of pixels that the player moves per frame (to calculate
//it just times the numbers by the framerate (30)
move_speed = choose(10.6,11.3,12,12.6,13.3)
//define the time since the player last changed directions as 0
time_since_last_movement_DIR = 0;
```

Player step event:

```
//Define look direction as where the players mouse is pointing on the screen.
//the ending bit (the /10) defines the mouse sensitivity
look_dir -= (window_mouse_get_x() - window_get_width() / 2) / 10;

//Define old look direction as where the player is looking.
old_look_dir = look_dir;

//If the player is moving and they arent changing the direction that they are looking in.
if look_dir = old_look_dir and moving = true
{
time_since_last_movement_DIR +=1;
}
//If the player starts looking in a different direction, records the time
//since they last looked in a new direction as 0
if (look_dir != old_look_dir)
{
    time_since_last_movement_DIR = 0;
}
//If the player is not moving, set the time since last direction change to 0,
//because the camera shouldn't spin if the player isn't actually doing anything
if moving = false
{
time_since_last_movement_DIR = 0;
}
//Defines the verticle camera sensitivity (z)
look_pitch -= (window_mouse_get_y() - window_get_height() / 2) / 10;

look_pitch -= (window_mouse_get_y() - window_get_height() / 2) / 10;

//Defines the camera to not be able to go above or below 80 degrees,
//this is because if the camera gets to high or low, the world will start to look weird.
look_pitch = clamp(look_pitch,-80,80);
//Sets the cursor to the middle of the screen, by getting the screen
//height and width and cutting them in half to find the exact center.
window_mouse_set(window_get_width() / 2, window_get_height() / 2);


//Allows the player to use the escape key to end the game once they are done with it.
if (keyboard_check_direct(vk_escape)){
game_end();
}

//If shift is pressed, double the movement speed
if keyboard_check(vk_shift){
move_speed = 24;
}
//If shift isn't being pressed, set the default movement speed as being 12
else if !keyboard_check_pressed(vk_shift) {
move_speed = 12;
}
//Strafing? I think that is the term idk
if keyboard_check(ord("A") or keyboard_check(vk_left)){
        //Move across the X and Y axes proportional to the direction that the player is looking in
        //Go backwards on the x axis (Does not matter much)
        x-=dsin(look_dir) * move_speed;
        //Go backwards on the y axis (matters)
```

```
            //Go backwards on the y axis (matters)
        y-=dcos(look_dir) * move_speed;
        Camera.xdist += move_speed;
    }


    if keyboard_check(ord("D")or keyboard_check(vk_right)){
            //Move across the X and Y axes proportional to the direction that
            //the player is looking in
            //Go forwards on the x axis (does not matter much)
                x+=dsin(look_dir) * move_speed;
                //Go foorward on the y axis (matters)
        y+=dcos(look_dir) * move_speed;
        Camera.xdist -= move_speed;
    }

    //Regular forward and backward movement
    //If pressing the W key
    if keyboard_check(ord("W")or keyboard_check(vk_up)){
        //Move across the X and Y axes proportional to the direction
        //that the player is looking in
        //Go forward on the x axis (matters)
        x+=dcos(look_dir) * move_speed;
        //Go backwards on the y axis (does not matter much)
        y-=dsin(look_dir) * move_speed;
        Camera.ydist += move_speed;
    }
    //Same as the W code, but since S is taking you in the
    //opposiet direction of W, the x and y values are flipped
```

```
    //If pressing the S key
    if keyboard_check(ord("S") or keyboard_check(vk_down)){
            //Move across the X and Y axes proportional to the direction that the player is
            //Go backwards on the x axis (matters)
        x-=dcos(look_dir) * move_speed;
        //Go forwards on the y axis (Does not matter much)
        y=+dsin(look_dir) * move_speed;
        Camera.ydist -= move_speed;
    }
    //If far away enough, end the game
    if distance_to_object(Start) > Camera.game_distance{
        room_goto(rm_end)
    }
    //if the player has been looking in the exact same direction
    //for 45 seconds, start spinning them.
    if time_since_last_movement_DIR > 45 * 30
    {
        //Spin 3 degrees per frame, (3 x 30) div 360 = quarter of
        //a full rotation per second
        look_dir += 3;
    }
    //If the player is giving inputs, set moving as true.
    if (keyboard_check(ord("W"))) or (keyboard_check(ord("A"))) or (keyboard_check(ord("S")
    {
    moving = true;
    }//If the player is not pressing any of these keys, set moving as false.
    else
    {
    moving = false;
```