



Ah movement. An important thing for all of your gaming projects. You want to see your characters move around and be able to steer them where you want them to go. But like me once it's possible this lite\_c programming is all new to you. Snippets of codes don't make any sense to you yet but after this tiny workshop/tutorial you will be understanding the basic a bit more and be able to use it in your own projects.

With this download you got a full work folder including a test level. Open this level and the script that comes with it. Then we will walk through all different actions we can assign to the main character SID. Keep in mind I am no master programmer so things could be done in a different perhaps even better way. All scripts are tested and do work so you will be able to use it.

The first thing we do is make Sid walk just straight ahead without using any of his animations. We program this action first so we can assign it to the model (Entity) in WED later.

### **Simply moving script**

```
action simply_moving()
{
set(my,SHADOW);
while(1)
{
c_move (my, vector (2 * time_step, 0, 0), nullvector, GLIDE);
wait(1);
}
}
```

The name of the action always ends with () and begins with the word action.  
open the first instruction with a {  
give the model some shadow  
starts a loop. All in the loop will happen continuously until the loop gets broken  
open a new instruction with {  
c\_move is used by all movement instruction, my vector is the models place at begin 2 is the speed of movement on the x axis, \*time\_step makes sure it has the same speed on any pc (Framerate), 0,0 means no speed, movement on y and z axis, nullvector placement start, GLIDE makes entities glide against level blocks. You will see that the wall will stop Sid's movement. End instruction line with a ; as also is used after wait(1)

So you use the x,y,z axis in numbers here x= 2, y and z are 0

wait 1 frame  
close all with 2 }} since you used also 2 {{ to open instructions.

Assign this action to the Sid model and run your level. If all goes well Sid will move straight ahead without using any animations. You just learned a very simple movement script and from here we will expand it.



Let's continue by using this same script but adding some animation to it. In MED you can see the models animation with the names of the animation frames. We're going to use that in our next script.

### **Simply moving script with animation.**

```
action simply_animmoving()
{
  set(my,SHADOW);
  var walking_percentage = 0;
  while(1)
  {
    walking_percentage %= 100;
    walking_percentage += 3 * time_step;
    ent_animate(my, "walk", walking_percentage, ANM_CYCLE);
    c_move (my, vector (2 * time_step, 0, 0), nullvector, GLIDE);
    wait(1);
  }
}
```

I'll skip the lines that you already learned in the first script so pay attention to the explanation of the rest. Here it goes.

A variable for the animation is created. This always starts with var then the name of the variable and we set it on 0. End the line with a ;

We set the variable's percentage to 100% play all the animation frames. We set the playing speed of this animation percentage on +3 the time\_step you should understand. End this line with a ;

ent\_animate is the instruction used to activate the models animation. My is the model itself, the animation frame names used is walk and ANM\_CYCLE plays the animation continuously. End this line with a ;

If you use NULL or 0 instead of ANM\_CYCLE the animation will play only once.

Assign this action to the Sid model and run your level. If all goes well Sid will move straight ahead and is now using it's walk animation. You just learned a very simple movement with animation script that is a good basic to continue with. You can change the speed of the animation by changing the 3 into any speed you like. Change the movement on the x-axis by replacing the 2 with any number you like and experiment with them. Try to use another animation frame series by changing the name. For example replace walk with fly.



Let's try to make the model move on the press of the cursor up key (cuu). I'll use the previous code and one more line of script is added. Here is how to do it.

```
action keya_moving()
{
  set(my,SHADOW);
  var walking_percentage = 0;
  while (!key_cuu) wait (1);
  while(1)
  {
    walking_percentage %= 100;
    walking_percentage += 3 * time_step;
    ent_animate(my, "walk", walking_percentage, ANM_CYCLE);
    c_move (my, vector (2 * time_step, 0, 0), nullvector, GLIDE);
    wait(1);
  }
}
```

Like previous scripts I'll only explain what is new to the script.

Before the while (loop) we add another loop. While( if everything is else than the press on the cursor up key (cuu) just wait one frame and end the line with a ; The ! before the key\_cuu stands for if it's different than cuu.

Assign the action to the Sid model and run the level. You will see that Sid starts moving when you have pressed the cursor up key.

Fairly easy to do right ? Shall we continue then ?

By now you should understand how to make a model move on a key press and using its animations. Let's see if we can make Sid move around like Pac-man.

Each entity (model) has a pan. The pan is nothing else than the direction that the model is facing. So if we make Sid move automatically all we have to do to make it change direction is to change its pan on the different key presses. Here is how.



```
action simply_packman()
{
  set(my,SHADOW);
  var walking_percentage = 0;
  while(1)
  {
    if(key_cul){
      my.pan =90;
    }
    if(key_cur){
      my.pan =-90;
    }
    if(key_cud){
      my.pan =-180;
    }
    if(key_cuu){
      my.pan =0;
    }
    walking_percentage %= 100;
    walking_percentage += 10 * time_step;
    ent_animate(my, "walk", walking_percentage, ANM_CYCLE);
    c_move (my, vector (10 * time_step, 0, 0), nullvector, GLIDE);
    wait(1);
  }
}
```

If stands in this case for when a mentioned key is pressed execute do the next instruction.

So on press cursor left execute by setting the pan on 90 degrees. Place the instruction between {}  
And ass always end your instruction line with a ;  
On cursor right the pan will be set on minus 90 degrees. Cursor down sets it on minus 180 degrees and the cursor up holds the models pan on 0 degrees.

The four cursor keys up down left and right are set by using cuu-cud-cul-cur. If you want to see how to use all different keys check the manual you will find it under the title key mapping.

Assign this action to the Sid model and run the level. Sid moves like Pac-man. It changes direction only by pressing the assigned keys and moves automatically. Well done.

So we know now how to steer the model by using key press and make it move with animation automatically. But what if we don't want it to move right away ? What if you want Sid to move only on a key press and uses for example a stand animations for when no key is pressed at all ?  
Let's do this right now it will be so easy.



```

action adventure_movement()
{
    set(my,SHADOW);
    var walking_percentage = 0;
    var standing_percentage = 0;
    while(1)
    {
        if(key_cul){
            my.pan =90;
        }
        if(key_cur){
            my.pan =-90;
        }
        if(key_cud){
            my.pan =-180;
        }
        if(key_cuu){
            my.pan =0;
        }
        if(!key_cuu - key_cud - key_cul - key_cur)
        {
            standing_percentage %= 100;
            standing_percentage += 3 * time_step;
            ent_animate(my, "stand", standing_percentage, ANM_CYCLE);
            wait(1);
        }
        else
        {
            walking_percentage %= 100;
            walking_percentage += 10 * time_step;
            ent_animate(my, "walk", walking_percentage, ANM_CYCLE);
            c_move (my, vector (10 * time_step, 0, 0), nullvector, GLIDE);
            wait(1);
        }
    }
}

```

I added a variable for the standing animation. Remember always start with var followed by the name of the animation and set it on 0.

Now if the cursor keys are not pressed it plays its standing animation and will not move at all. else (meaning if all before is not the case then) move and use the walking animation. You should understand by now how to use animation. Assign this action to the Sid model and run the level. Now Sid will only move when you use the cursor keys and it uses the standing animation when you are not pressing the cursor keys. A basic movement for a platform game is created. Experiment and change the keys to use in WASD. See if you can do that.



Pretty cool isn't it ☺ now let's figure out if we can use the last script with a so called joypad (also known under the names Controller or joystick).

We used keymapping before and when using a joypad we need the instructions for it. For my example I am using a Microsoft USB game controller. Let's see if we can make it work. The scripting will be different then what we used before.

```

action joystick_movement()
{
    set(my,SHADOW);
    var walking_percentage = 0;
    var standing_percentage = 0;
    while(1)
    {
        if(joy_force.x < 0 && my.pan != 90){ my.pan = 90; }
        if(joy_force.x > 0 && my.pan != 270){ my.pan = 270; }
        if(joy_force.y > 0 && my.pan != 0){ my.pan = 0; }
        if(joy_force.y < 0 && my.pan != 180){ my.pan = 180; }
        result = c_move(me, nullvector, vector(10 * joy_force.y * time_step, -10 * joy_force.x * time_step, 0), GLIDE |
        IGNORE_ME | IGNORE_PASSABLE);
        if(result > 0)
        {
            walking_percentage %= 100;
            walking_percentage += 5 * time_step;
            ent_animate(my, "walk", walking_percentage, ANM_CYCLE);
            wait(1);
        }
        else{
            standing_percentage %= 100;
            standing_percentage += 5 * time_step;
            ent_animate(my, "stand", standing_percentage, ANM_CYCLE);
            wait(1);
        }
    }
}

```

The pan direction is now activated by the movement of the controllers steering pad. Up down left and right will change the pan of the model. New is the line that says `result = c_move`. It gives the movement instruction on x and y axis by a speed of 10 using `joy_force x` and `y`. When this result is bigger than 0 (>) it will use the walking animation and its speed. If no steering is happening so the result is not bigger than 0 it will stop moving and using the standing animation.

Assign this action to the Sid model and run your level. You should now be able to move Sid with your controller. Let the game creation begin.

A simple test of your controller buttons can be done by assigning an instruction to the press of one of this buttons. How ? Here it comes.



We will use the previous script and add a few lines. On the press of a button a text will appear. If you understand how to do this you can add later other stuff to the different buttons. For example making the character jump.

```

action joystick2_movement()
{
    set(my,SHADOW);
    var walking_percentage = 0;
    var standing_percentage = 0;
    while(1)
    {
        if(joy_force.x < 0 && my.pan != 90){ my.pan = 90; }
        if(joy_force.x > 0 && my.pan != 270){ my.pan = 270; }
        if(joy_force.y > 0 && my.pan != 0){ my.pan = 0; }
        if(joy_force.y < 0 && my.pan != 180){ my.pan = 180; }
        result = c_move(me, nullvector, vector(10 * joy_force.y * time_step, -10 * joy_force.x * time_step, 0), GLIDE |
        IGNORE_ME | IGNORE_PASSABLE);
        if(result > 0)
        {
            walking_percentage %= 100;
            walking_percentage += 5 * time_step;
            ent_animate(my, "walk", walking_percentage, ANM_CYCLE);
            wait(1);
        }
        else{
            standing_percentage %= 100;
            standing_percentage += 5 * time_step;
            ent_animate(my, "stand", standing_percentage, ANM_CYCLE);
            if(joy_1)
            draw_text("YAY THE BUTTON WORKS ! ",100,10,vector(100,100,255));
            wait(1);
        }
    }
}

```

When pressing joy\_1 (button 1) draw\_text (show the following text) place the text at 100 on x and 10 on y screen and use the colour code 100,100,255. This colour code will be red. Change it into any colour you like. Try changing it into a different colour for example 255,255,255.

Assign this script to Sid and run the level. Move around using your controller and on the press of the button the text will appear. Well done you know now how to use keys or controllers for your projects.



Next part of this tutorial I will try to explain how to make your character jump.

I will use the adventure movement script for this and add new functions on top of it needed for jumping. If you understand this last piece of scripting you should be able to convert it yourself and use a controller instead of the keyboard. Let's do this. In order to make jumping work we start with creating 2 functions. One that makes Sid jump and a second that creates gravity.

```
VECTOR absdist = 50;
var jump_time = 1;
var jump_height = 50;
var my_height = 50;
function my_jump()
{
    if(jump_time != 1){ return; }
    while(jump_time > -1){
        jump_time -= 0.2 * time_step;
        absdist.z = jump_height * jump_time * time_step;
        wait(1);
    }
    while(my_height > 5){ wait(1); }
    jump_time = 1;
}
```

We create a vector absdist(distance) and set it on 50

We create 3 variables jump\_time, Jump\_height, and my\_height

A function always starts with the word function followed by the name of the function and ends with()

We open the instruction with {

When the variable jump\_time is different then 1 we return to start function with the command return ending with a ; and closing the instruction with a }

When jump\_time variable is bigger than -1 do the next instruction

Open the instruction with {

Make jump\_time variable number go down on speed 0.2 and time\_step you know what it does.

In the mean time make the absdist(ance) variable set on jump\_height and jump\_time (Go up)

Wait one frame

Close this instruction with a }

And as last when my\_height variable is bigger than 5 wait one frame

Set jump\_time variable back to 1 (Ready for a new jump)

So we created the my\_jump function. Now let's create the gravity function that we place under the my\_jump function.





```
function my_gravity()
{
    VECTOR temp;
    vec_fill(temp.x, 0);
    vec_set(temp.x, my.x);
    temp.z -= 2000;
    trace_mode = IGNORE_PASSABLE | IGNORE_ME | USE_BOX;
    my_height = c_trace(my.x, temp.x, trace_mode);
    if(my_height > 5 || my_height <= 0){
        accelerate(absdist.z, -20 * time_step, 0.3);
    }
    Else
    {
        if(jump_time == 1){
            absdist.z = 0;
        }
    }
}
```

Remember a function always starts with the word function followed by the name of the function and ends with()

Open the instruction with {

A temporary vector is created.

Vec\_fill is temp.x set on 0

Vec\_set = temp.x the entities x.

Temp.z = minus 2000 (under the entity)

We use trace and it ignores passable objects as well as the model itself. A box around the model is used for the tracing.

My\_height variable = tracing the models x and temp.x position when tracing

When my\_height variable is bigger and my\_height is smaller or equal to 0

Accelerate the distance minus 20 and use time\_step 0.3

Close the instruction with }

Else so if none of above

Open instruction with {

If the jump\_time variable is equal to 1

Set the absdist.z back to 0

Close the instructions with }}}

So the gravity function is done. Now let's make our gravity and the jump function work.



```

action adventure2_movement()
{
    set(my,SHADOW);
    var walking_percentage = 0;
    var standing_percentage = 0;
    on_space = my_jump;
    while(1)
    {
        my_gravity();
        wait(1);
        if(key_cul){
            my.pan =90;
        }
        if(key_cur){

            my.pan =-90;
        }
        if(key_cud){

            my.pan =-180;
        }
        if(key_cuu){

            my.pan =0;
        }
        if(!key_cuu - key_cud - key_cul - key_cur)
        {
            standing_percentage %= 100;
            standing_percentage += 3 * time_step;
            ent_animate(my, "stand", standing_percentage, ANM_CYCLE);
            c_move (my, vector (0 * time_step, 0, absdist.z), nullvector, IGNORE_PASSABLE | IGNORE_ME | GLIDE);
            wait(1);
        }
        else
        {
            walking_percentage %= 100;
            walking_percentage += 20 * time_step;
            ent_animate(my, "walk", walking_percentage, ANM_CYCLE);
            c_move (my, vector (20 * time_step, 0, absdist.z), nullvector, IGNORE_PASSABLE | IGNORE_ME | GLIDE);
            wait(1);
        }
    }
}

```

Before the while we put the line that makes a press on spacebar activate the jump function.

In the while loop we activate the gravity function.

While standing there is no movement but the absdist.z variable is used so we can jump.

While walking there is movement and the absdist.z variable is used so we can jump.

Assign this script to Sid and run the level. Now Sid can walk and jump on the press of the spacebar.

As soon as Sid lands on the ground he can jump again.



If you want to play the jump animation while jumping we add some more lines of script.

```

action adventure3_amovement()
{
set(my,SHADOW);
var walking_percentage = 0;
var standing_percentage = 0;
var jumping_percentage =0;
on_space = my_jump;
while(1)
{
my_gravity();
while(key_space){
ent_animate(my, "jump", jumping_percentage, 0); // "death" one-shot animation
jumping_percentage += 4 * time_step;
c_move (my, vector (10 * time_step, 0, absdist.z), nullvector, IGNORE_PASSABLE | IGNORE_ME | GLIDE);
wait(1);
}
if(key_cul){
my.pan =90;
}
if(key_cur){
my.pan =-90;
}
if(key_cud){
my.pan =-180;
}
if(key_cuu){
my.pan =0;
}
if(!key_cuu - key_cud - key_cul - key_cur )
{
standing_percentage %= 100;
standing_percentage += 3 * time_step;
ent_animate(my, "stand", standing_percentage, ANM_CYCLE);
c_move (my, vector (0 * time_step, 0, absdist.z), nullvector, IGNORE_PASSABLE | IGNORE_ME | GLIDE);
wait(1);
}
else
{
walking_percentage %= 100;
walking_percentage += 10 * time_step;
ent_animate(my, "walk", walking_percentage, ANM_CYCLE);
c_move (my, vector (10 * time_step, 0, absdist.z), nullvector, IGNORE_PASSABLE | IGNORE_ME | GLIDE);
wait(1);
}
}
    
```

I added a variable for the jumping animation.

When space\_bar is pressed it activates the my\_jump function but it also activates the animation.

Ent\_animate is used to activate the animation. Jump is used and it will play only 1 time during press spacebar.

The speed of the animation is set on 4.

The c\_move is also here because it still needs movement and after a jump we should be able to jump again.

Assign this action to the Sid model and run the level. Every time you press the spacebar Sid will jump and use its animation.



As final part of this workshop we will do a very basic car vehicle movement. We will make the car model move and steer by using the cursor keys. Also we will be able to make it speed up and down and use reverse driving. Ready ? Let's see how it works.

```

action car_movement()
{
    set(my,SHADOW,METAL);
    var movement_speed = 0;
    var rotation_speed = 15;
    var car_maxspeed = 20;
    VECTOR car_speed, temp;
    c_setminmax(me);
    while(1)
    {
        my.pan += rotation_speed * (key_cul - key_cur) * time_step;
        vec_set(car_speed.x, accelerate (movement_speed, 5 * (key_cuu - key_cud), 0.1));
        car_speed.y = 0;
        vec_set (temp.x, my.x);
        temp.z -= 10000;
        car_speed.z = -c_trace (my.x, temp.x, IGNORE_ME | USE_BOX) ;
        c_move (my, car_speed.x, nullvector, GLIDE);
        wait(1);
        if (movement_speed >= car_maxspeed){
            movement_speed = 20;
        }
        if(movement_speed <=-20){
            movement_speed = -20;
        }
    }
}
    
```

Like previous scripts I will try to explain the new lines.

We know how to give shadow to a model. By adding METAL it will have a metallic look.

I created 3 variables one for movement\_speed set on 0. The second is rotation speed set on 15 (Pan direction).

The third is car\_maxspeed set on 20.

A vector is created for the car called car\_speed,temp

c-setminmax(me); makes sure it uses the full models size. (Better collision this way)

Then in the while we make sure that the cursor left and right changes the direction of the car it will use the speed setting set in the variable rotation\_speed.

Now on cursor up and down it will accelerate the car speed temp from 0 (variable movement speed) and it built up in steps of 0.1. The higher the number the faster it accelerates.

Now car\_speed on y is needed so we make sure to set it to 0

The temp vector is set on x axis and z is set to minus 10000 (car can fall down)

It uses c\_trace (remember) and it uses the box

When the movement\_speed variable is bigger or equal to car\_maxspeed we set the movement speed to 20 (max\_carspeed)

When the movement\_speed variable is smaller or equal to car\_maxspeed we set the movement speed to minus 20 (max\_carspeed)

This is a very simple basic vehicle movement. You can change it to your own likings but it gives you a clear start. Assign this action to the car model and run the level. Now you can drive around speed up, slow down or stand still.

I hope you found this useful and I am looking forward to see your game projects coming to life.

Happy game creation. René Pol aka Realspawn

