3D GameStudio

Workshop Fighting

for A5 Engine 5.10
by Alain Brégeon August 2001

The latest news, demos, updates and tools, as well as the Users' Magazine, the Users' Forum and the annual Contest are available at the GameStudio main page **http://www.3dgamestudio.com**.

Contents

Foreword

Dear Reader,

I created this workshop in order to help you to find an answer to the question "How to make a fighting game with 3D GameStudio?" The features used in this workshop are available with the version 4.25 or later.

This workshop, like others before is targeted at users who have some previous experience with 3D GameStudio. I assume that you have worked through the tutorials and know, how to use the tools (WED,MED and WDL).

This text improves the documentation which comes with 3DGameStudio, and cannot replace it. If something in this workshop is unclear to you, please read through the manuals enclosed. I apologize in advance for any unclear wording, faulty code, errors or omissions.

I hope you find the workshops informative and enjoyable.

Alain Brégeon
mailto:alainbregeon@hotmail.com

The philosophy of fighting this tutorial is based on, is deeply inspired by the demo 'Rockem' supplied that comes with the C ++ SDK by Microsoft ©.

Get the latest version

Before you begin, please make sure to have the latest version of 3DGameStudio (**4.25** or later).

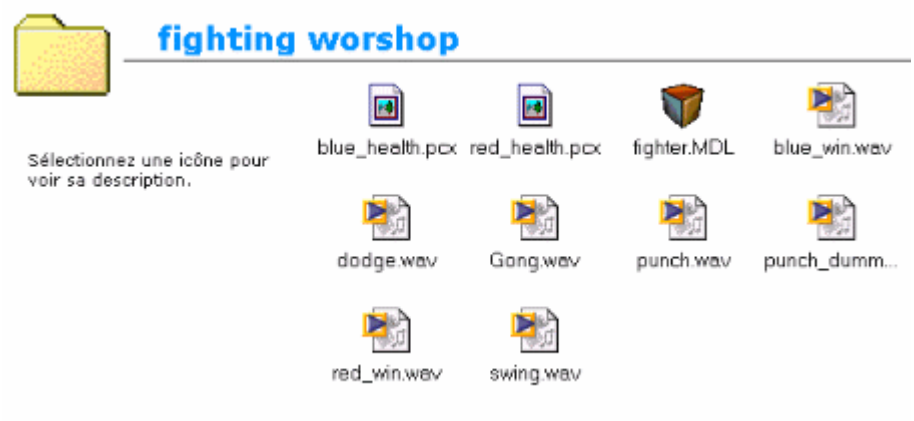
Prepare your workspace

Create a folder called "fighting Workshop" in your GSTUDIO folder. This is the directory where you will store all the game elements.

The first thing that we are going to do, is to add the entities which we need for our game into the folder. If you do not have them yet, visit conitec's download site ((http://www.conitec.net / a4update.htm) and get them. Unzip the contents into your folder.

Your folder should now contain the following files:

```
left_health.pcx
right_health.pcx
fighter.mdl
left_win.wav
dodge.wav
gong.wav
punch.wav
punch_dummy.wav
right_win.wav
swing.wav
```



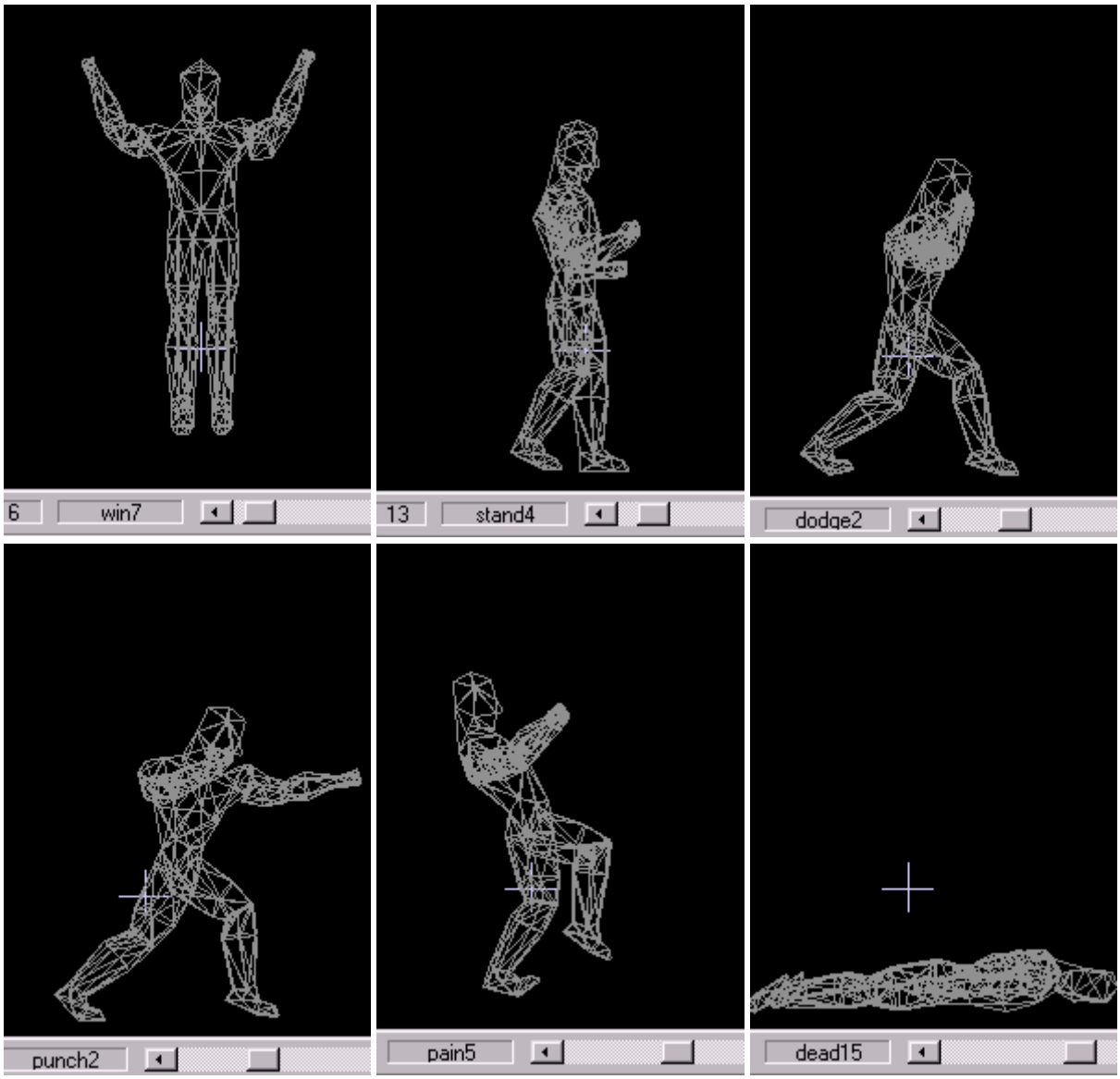
Fighting Workshop folder

Our Fighter

It is well known that fights stopped for the lack of fighters. So, if we want to create a fighting game, first of all we need fighters.

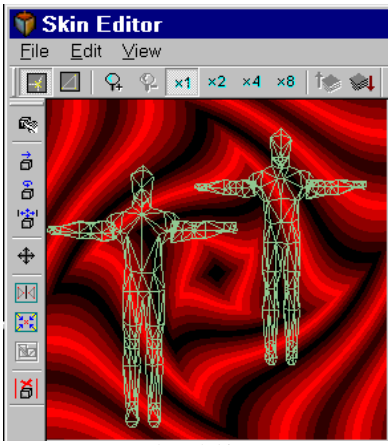
Our character should be able to dodge, to punch, to get hits, wait, die and win. The character that comes with this tutorial does all of it. Regarding the skins, I did the simplest for a good understanding of the tutorial: a red skin and a blue one.

Open MED, load **`fighter.mdl`** and have a look at all animations that this fighter can do.



the fighter.mdl

Of course he could do much more, but in order not to overload this tutorial, we should be satisfied with these 6 animations.



the red skin

Design the fighting level

We find ourselves compelled to choose: a boxing ring, a street, the moon... The only condition we have to meet, is to leave enough space for the camera. Here is my personal choice:

Open WED and select **File-> New**

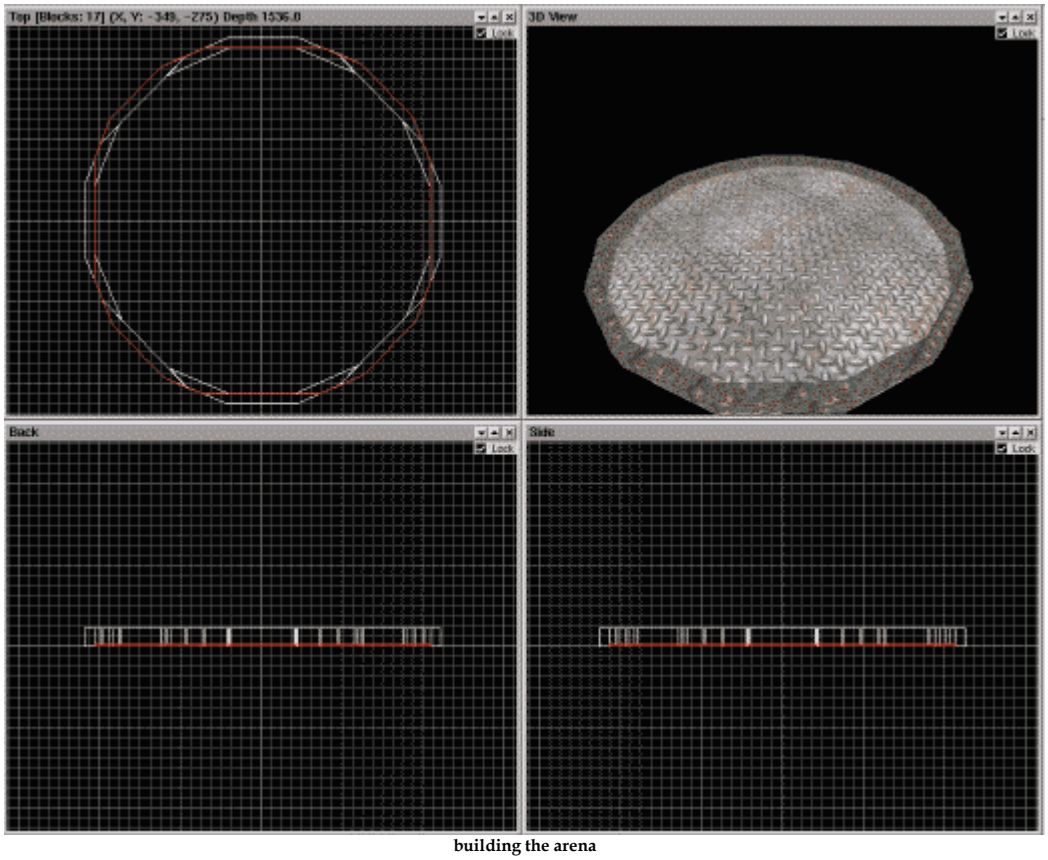
Open the texture manager (**Texture > Texture Manager**). Select **add WAD**, and add the **`standard.wad`**. Close the texture manager.

Choose **Add Primitive - > Cylinder -> 16**.

Enlarge the circle until you get about 5 big squares in diameter.

Hollow the block by **Alt+H**, which reduces its height to 2 small squares, **scope down**, select the top wall of your box and delete it.

I used the standard textures. On sides I took the texture **`metalrivet2`** and on the ground I have choosen the texture **`metalribbed5`**.

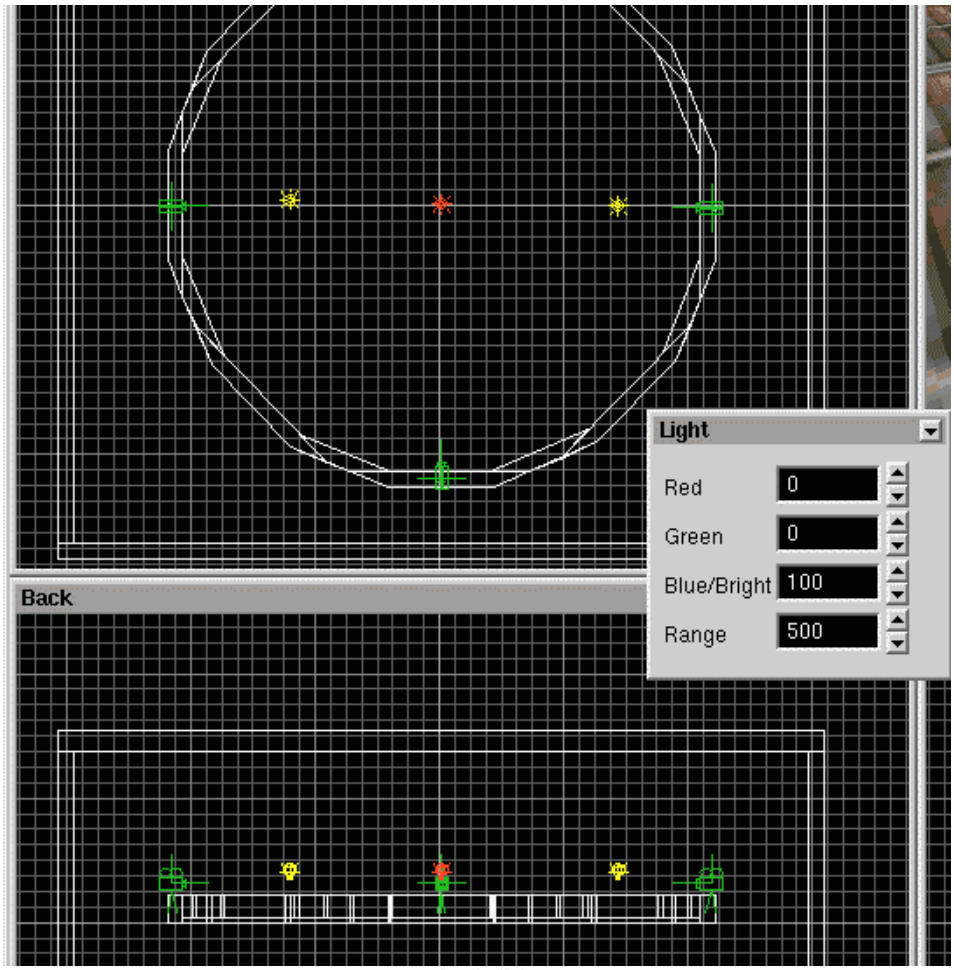


building the arena

Now we create a sky which is not really necessary, but we do it to keep the proper etiquettes.

Choose **Add Primitive -> Cube (large)** and keep the sides of your cube clear of the cylinder. Hollow the block by **Alt+H** and apply a texture (**`gate1`** for example).

We add 2 or 3 lights (**Add Light**) and we have to take good care with it: The first one we put into the left corner of our ring, the second into the right one and the third light will be set into the midle.



setting the lights

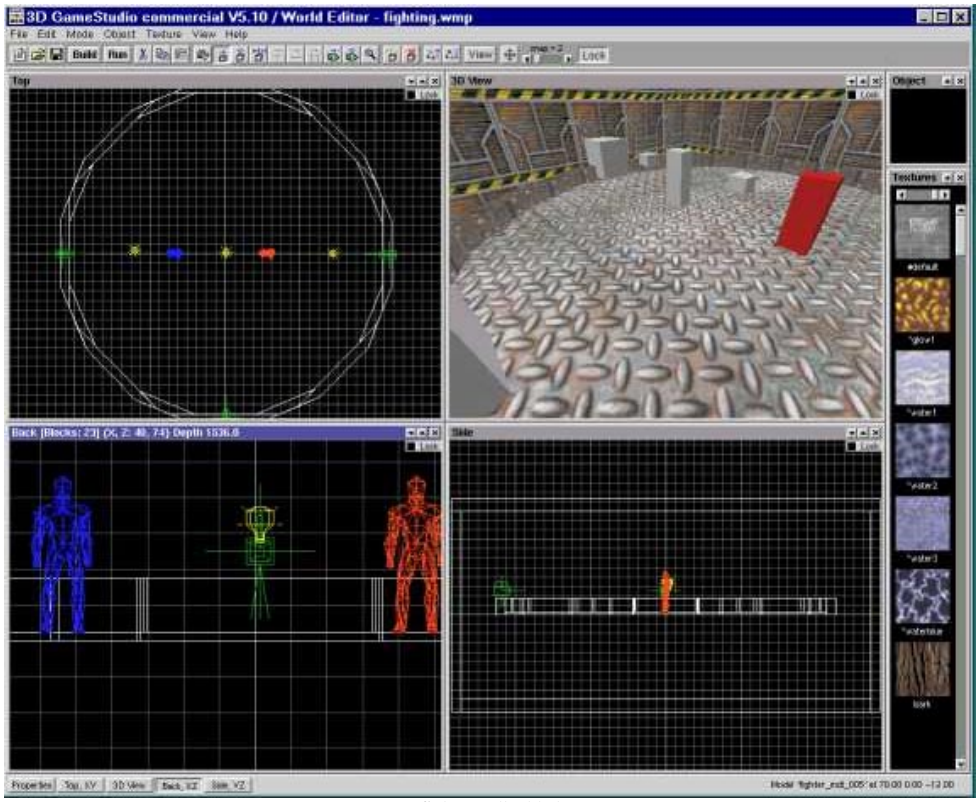
I feel you impatient to see what your level looks like. So go on, save it under the name of **`fighting`** then **build** with **fly-thru = enable** marked. Here is the result. Isn't it nice?

The illuminated arena

Adding the fighters

All we have left do do, is to place the two fighters and we are done so far.

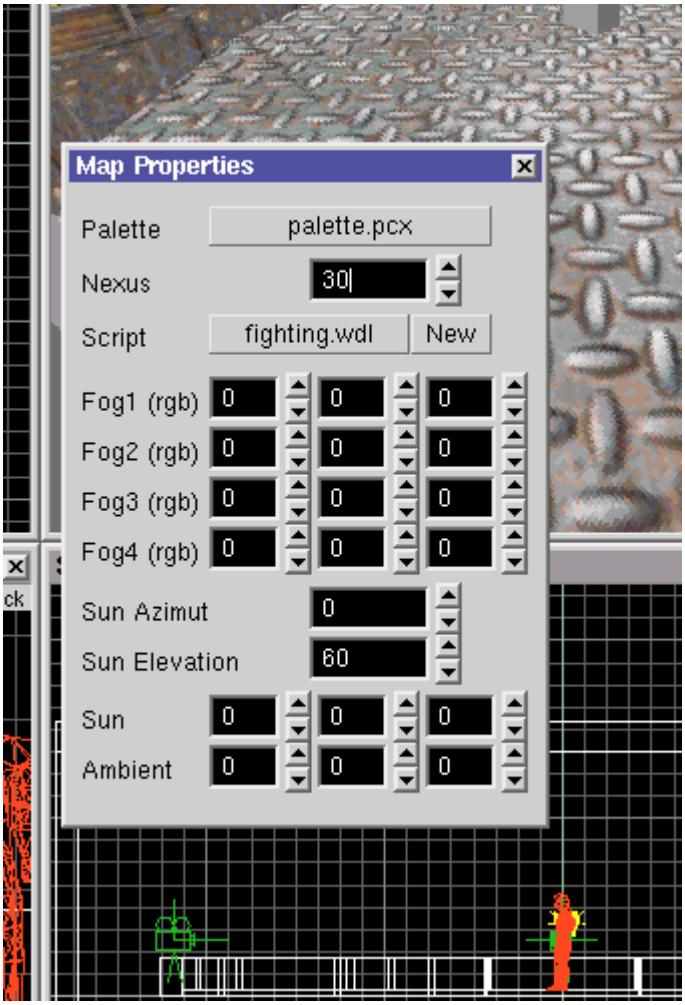
Select **Object-> Load Entity** and browse the navigator to find the model **`fighter.mdl`**.Get it and the model appears in your world. Place it near the center onto the ground.



fighter.mdl added

Create your script

Create a script for your level. Just open your property window (**File > MapProperties**) and click the button **new**. The button next to it should change from **`ndef`** into **`fighting.wdl.`**



New Script in Map Properties

Open your level folder, select and open (double-click) the file **`fighting.wdl`**. If Windows asks which application to use to open the file, choose "notepad" (or any other plain text editor).

You will notice that the standard game “Templates” have been created for you. This is fine for most projects but we want to do something more advanced this time. Go ahead and select everything (in Microsoft Notepad use **Edit->Select All**) and hit the delete key. Now you're starting from ground zero!

Adding Paths

Lets start by defining the paths our program is going to use. Paths are needed to tell the engine where it can locate the files used in our project (images, sounds, other scripts, etc.). The folder we are in ("Fighting Workshop") is already included so, in our case, we only need to add the **`templatel`** folders.

Type in the following line:

```
path  "..\\template";    // Path to WDL templates subdirectory
```

Note that all paths are relative to our level folder. The line above says the following, "Go 'up' one directory ("...") and go into the template folder (**\\template**)".

Adding Includes

After we set up our paths we should add our include files. Type in the following below **path**:

```
include <movement.wdl>;  // libraries of WDL functions
include <messages.wdl>;
include <menu.wdl>;      // menu must be included BEFORE doors and weapons
include <particle.wdl>; // remove when you need no particles
```

The **include** command tells the engine to replace this line with the contents of the file between the angular brackets (<...>). It is as if you went to the file in question and copied all the code from it and pasted it into you script.

This is a powerful tool because it allows us to reuse code from other projects and take advantage of updates in the included files without having to rewrite your code. For example, the 4.19 update included code to allow the player to swim in water. So now any project that includes **movement.wdl** can use this new swimming code.

Just like paths, the order of the **include** lines is important. Since some scripts use values that are defined in other scripts. For example: **actors.wdl** uses the variable **force** which is defined in the **movement.wdl** If we put **actors.wdl** before **movement.wdl** we would get errors.

It's okay to **include** scripts even if you don't use features from them in you code. Most of the files in the template are interdependent on each other so if you plan to use one of them, you should **include** all of them just to be safe. The exception to this rule is the **venture.wdll** script, which is not used by any of the other scripts, but uses many of them.

Starting engine values

Now we are going to set some important values that will help to determine how the simulator will be displayed. These values effect the resolution, color depth, frame rate, and lighting. Add the following lines beneath the **include** lines:

```
 // Starting engine values
ifdef lores;
var video_mode = 4; // 320x240
ifelse;
var video_mode = 6; // 640x480
endif;
var video_depth = 16; // D3D, 16 bit resolution
var fps_max = 50; // 50 fps max
```

Our game variables

It is here that we shall enter our game variables according to our needs:

```
//our skills **************************************************
```

Display the A4/A5 logo

We apply the display of the logo, which is located in the templates directory:

```
////////////////////////////////////////////////////////////
// define a splash screen with the required A4/A5 logo
bmap splashmap = <logodark.bmp>; // the default A5 logo in templates
panel splashscreen { bmap = splashmap; flags = refresh,d3d; }
```

The main function

In any project you need a **main** function. This is the very first function to be called when the program starts. In most cases the **main** function is very simple, our **main** is no exception. Please enter the following lines (below your last line):

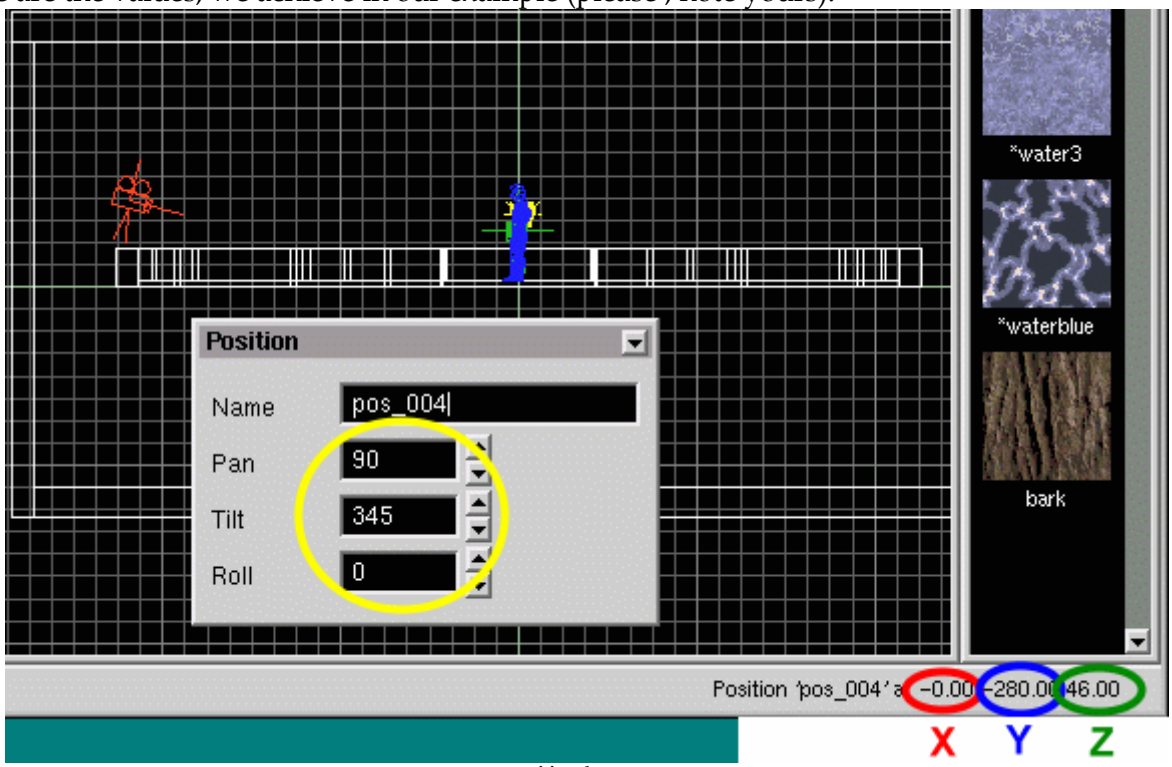
```
function main()
{
   fps_max = 50;
   warn_level = 2;    // announce bad texture sizes and bad wdl code
   tex_share = on;    // map entities share their textures

// center the splash screen for non-640x480 resolutions
   splashscreen.pos_x = (screen_size.x - bmap_width(splashmap))/2;
   splashscreen.pos_y = (screen_size.y - bmap_height(splashmap))/2;
   splashscreen.visible = on; // set it visible
   wait(3); // wait 3 frames (for triple buffering) until it is renderight and flipped to the foreground
// now load the level
   load_level (<fighting.wmb>);
// wait the required second, then switch the splashscreen off.
   waitt(16);
   splashscreen.visible = off;
   bmap_purge(splashmap);   // remove logo bitmap from video memory
   load_status(); // load some global variables, like sound volume
```

Each of these lines is commented, so there is no need of further explanations.

Initialize the camera position

We are going to use our third light position, to get the coordinates of our camera. Select the camera and drag it onto the coordinates of that position (you find them displayed at the bottom of the WED window).

Here are the values, we achieve in our example (please , note yours):



position the camera

It is not so much important to work very precisely here because, as we see later, our camera is dynamic.

Next we overtake these values and enter them into our main module:

```
camera.pan = 90;
camera.tilt = 345;
camera.x = 0;
camera.y = -280;
camera.z = 46;
```

Now it is time to implement the health variables.

Please type the following lines:

```
var right_health = 100;
var left_health = 100;
bmap left_health_map = <left_health.pcx>;
bmap right_health_map = <right_health.pcx>;

panel left_health_pan
{
   pos_x = 20;
   pos_y = 30;
   layer = 1;
```

```
   hbar = 0,0,200,left_health_map,2,left_health;
   flags = d3d,overlay,refresh;
}

panel right_health_pan
{
   pos_y = 30;
   layer = 1;
   hbar = 0,0,200,right_health_map,2,right_health;
   flags = d3d,overlay,refresh;
}
```

Then, below the **main** function, we enter these lines after the camera:

```
   right_health_pan.pos_x = screen_size.x - (bmap_width(right_health_map) + 20);
   right_health_pan.visible = on;
   left_health_pan.visible = on;
}
```

The first line allows to display the health points within 220 pixels (200 pixels is the width of the bitmap + 20 pixels) off the right edge of the screen.

Make the Entities Fighters!

To make it better readable, we called the blue player “left” and the red one “right”.
So underneath our variables we type:

```
synonym left { type entity; }
synonym right { type entity; }

var skin_right = 1;
var skin_left = 2;
```

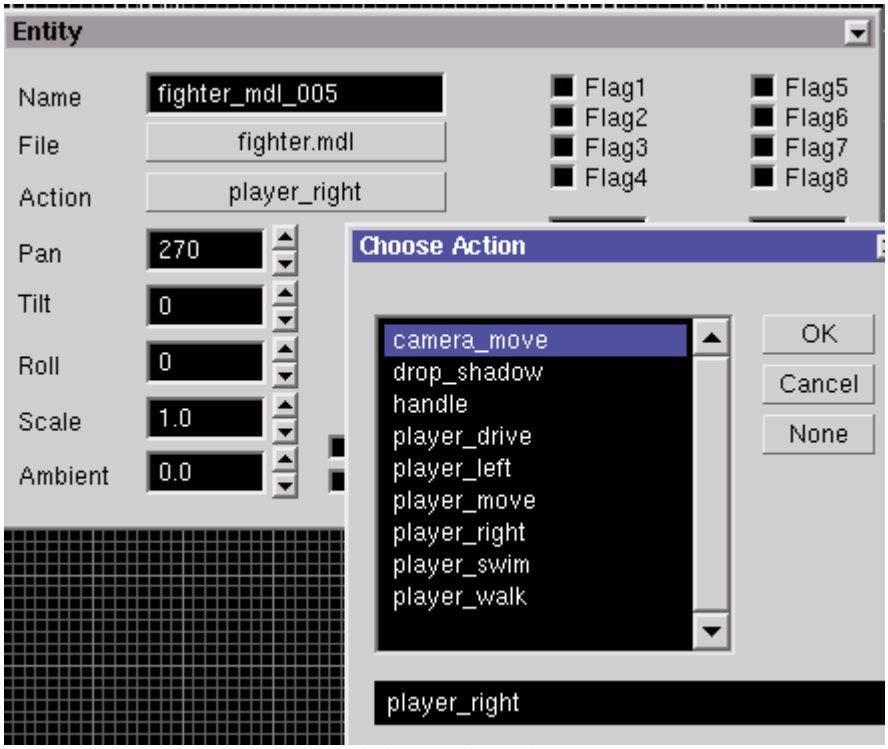
Then we create an action for each of the entities that we place after the function **main()**

```
action player_right
{
   right = my;
   my.skin = skin_right;
   my.pan = 180;
   if (my.shadow == off) { drop_shadow(); }
   while ((right == null) || (left == null)){wait 1;}
}

action player_left
{
   left = my;
   my.skin = skin_left;
   my.pan = 0;
   if (my.shadow == off) { drop_shadow(); }
   while ((right == null) || (left == null)){wait 1;}
}
```

It's high time now to save our file and return in WED to see the fruits of our work.

To determine the relevant action, we select the left person then right click to bring up the entity's properties panel where we assign **player_left** as action. Do the same now with

`player_right`.



assigning the entity's action

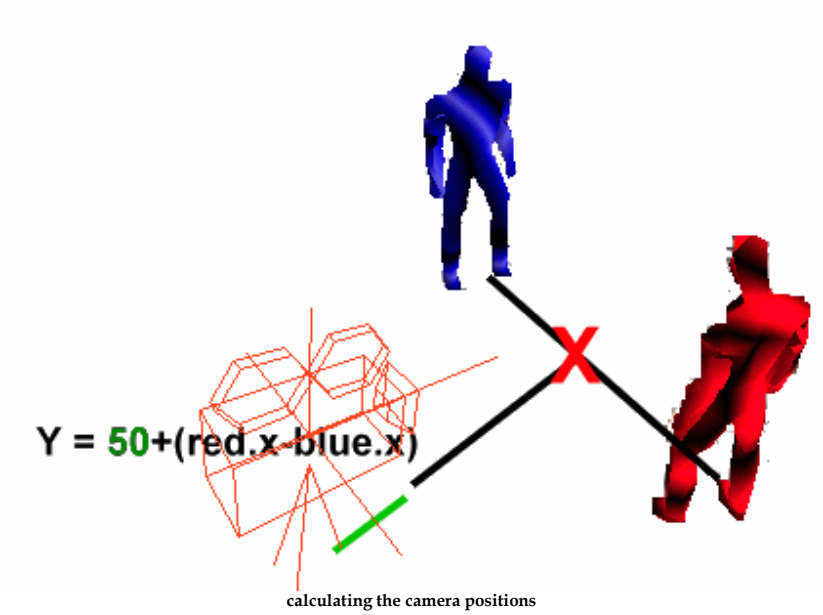
Now save the level, **build** and run it. If you did fine, it should look like this:

two fighters in the arena

Management of the camera

As we see on the picture, a step backward of one of the players would bring him outside the camera's view. That's why we are going to write a little funktion which we will call within every movement of a player to reposition the camera.

As position X we take the point amidst the two players. Our point Y will be the distance between them plus 50.



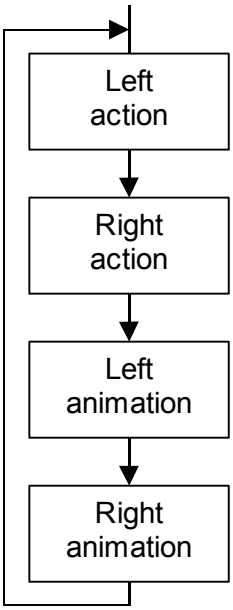
calculating the camera positions

We write this function at the very end of our script:

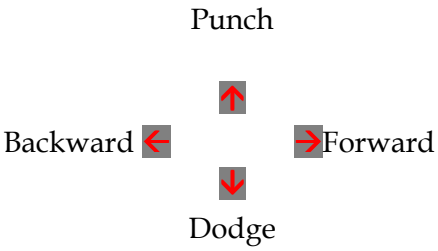
```
function pos_camera()
{
   camera.z = 90;
   camera.x = ((right.x+left.x)/2);
   camera.y = - ((right.x-left.x) + 50);
   wait (1);
}
```

The Game Course

The game can be subdevided into 4 main phases:



The "left" player's action is controled by the arrow keys as follows:



The action of the "right" player requires artificial intelligence, which in our case will be quite short. We plan three possible states for this player:

- Prudent
- Defensive
- Offensive

The transition between these states depends partially on our behavior and on the other hand by a random factor included.

Therefore we write the following variables in our script:

```
var prudent = 1;
var defensive = 2;
var offensive = 3;
var right_state_of_mind;
```

The states of the players

As we decided at the beginning by choosing our model, we kept 6 states which are:

waiting	**dodging**	**punching**
win	**pain**	**dead**

So we write the following variables in our script:

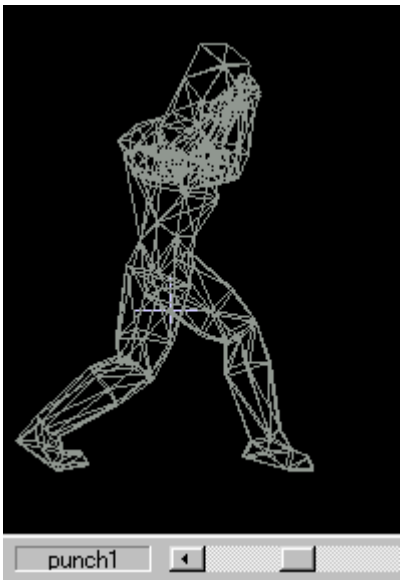
```
var waiting = 1;
var dodging = 2;
var punching = 3;
var pain = 4;
var dead = 5;
var win = 6;

var right_action_state;
var left_action_state;
```

As well, we want to hear voices during some of these actions, so let us add some sounds as follows:

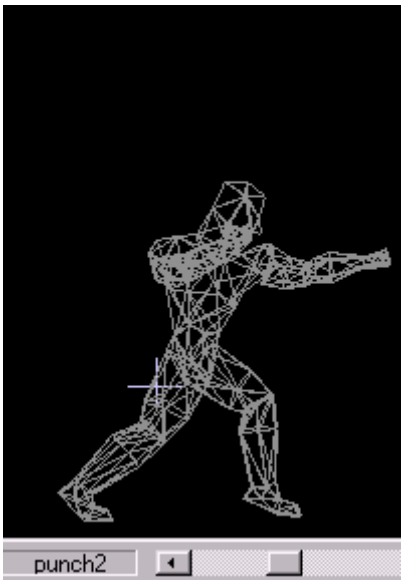
```
sound dead_sound = <dead.wav>;
sound punch_sound = <punch.wav>;
sound punch_dummy_sound = <punch_dummy.wav>;
sound left_win_sound = <left_win.wav>;
sound dodge_sound = <dodge.wav>;
sound lark_sound = <lark.wav>;
sound right_win_sound = <right_win.wav>;
sound gong_sound = <gong.wav>;
sound swing_sound = <swing.wav>;
```

The animation

For the animation, we use the instructions **`ent_frame()`** and **`ent_cycle()`**, but if we have to intervene inside an animation, we have to use the parameter **`frame`** in a proper way for every entity. For example: when the right player punches to you, you can dodge the blow at the first frame. But at the second frame it is too late, the blow is already completed.



You can dodge the blow...



....It's too late to dodge

This requires to analyze each animation cycle and find the frame numbers for the beginning, the end and if necessary somewhere in the middle.

You are lucky I did it for you with this model, but for the other models, you have to do it yourself.

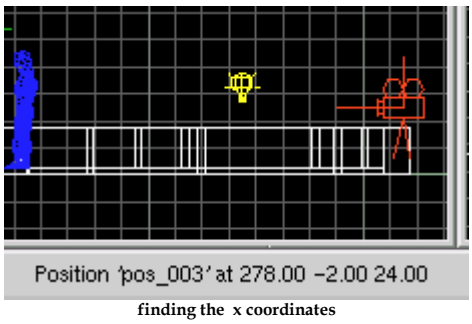
So we type the following variables:

```
var start_wait = 44;
var end_wait = 48;
var start_dodge = 40;
var end_dodge = 44;
var start_punch = 48;
var end_punch = 52;
var start_pain = 64;
var end_pain = 73;
var start_win = 1;
var end_win = 10;
var start_dead = 73;
var end_dead = 88;

var start_left_frame = 44; //wait
var end_left_frame = 48;
var start_right_frame = 10; //wait
var end_right_frame = 12;
```

Actually the endframe corresponds to the last frame + 1.

We have left some variables to be defined like the boards of the territory and the distance within the punch of a player reaches the opponent.

For the edges of the territory, we use again our two left and right (light-) positions in WED and apply them as the X coordinates.

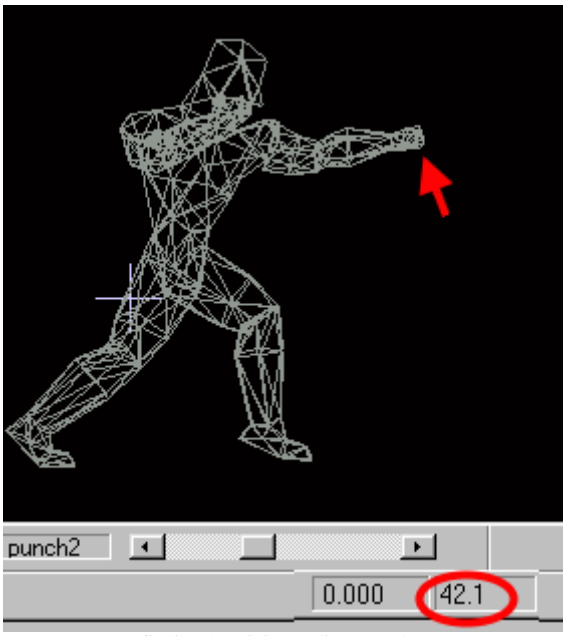


finding the x coordinates

So we note 278 and create the following 2 variables:

```
var left_edge = -250; //dependend on width level
var right_edge = 250; //dependend on width level
```

For the minimum distance between the players we open the frame **punch2** (spreaded arm) in MED, and place the mouse at the extremity of the outstreched arm. The given coordinates determine our value X.



finding the minimum distance value

Now we create the following two variables:

```
var min_dist_between_right_left = 42;
var dist_between_right_left;
```

And we compled our variables by determing the two following ones whose use will become clear with further explanations.

```
var left_state_attack_block = 0; // 1 = attacking 2 = blocking;
var alea;
```

Finally we are ready to fight.

I know how much frustrating it is to write code without being able to test it. So first of all, we are interested in the blue player and jump into action with alternately animations.

The game

For this we are going to create our game **`function game()`** and position our labels.

```
function game()
{
   wait (100);
   right_state_of_mind = prudent;
   left_action_state = waiting;
   start_left_frame = start_wait; //waiting
   left.frame = start_left_frame;
   end_left_frame = end_wait;
   right_action_state = waiting;
   start_right_frame = start_wait; //waiting
   right.frame = start_right_frame;
   end_right_frame = end_wait;
   play_sound (gong_sound,100);

   while (1)
   {
```

```
      //here left action

      //here right action

left_anime:
      //here left animation

right_anime:
      //here right animation

fin:
      wait(1);
   }
}
```

And add

```
game();
```

.... at the final end of the **main** function

Now let us try out. If everything is all right, you hear the gong and the players are in position ready to fight.

The movements of the left player

Type the following lines:

```
      //here left animation
   dist_between_right_left = right.x - left.x;    //left move forward if not win and not dead

   if ((key_cur == 1) && (left_action_state != win) && (left_action_state != dead))
   {
      //make sure left can move to right
      if (dist_between_right_left > min_dist_between_right_left)
      {
         left.x += 2;
         pos_camera();
      }
   }

   //left move backward if not win and not dead
   if ((key_cul == 1) && (left_action_state != win) && (left_action_state != dead))
   {
      //make sure left can not out left edge
      if (left.x > left_edge)
      {
         left.x -= 2;
         pos_camera();
      }
   }
```

Then try it out. Move forward, move backward, check out if you don't exceed any given limit. If so, adjust the **left_edge** and **right_edge** values. Also admire the camera movements. As soon as the right Player knocks you out, there is no time left at all to look at anything.

Actually we must admit, that our player moves back and forth in a quite funny way. To fix this,

insert the following lines **after `left_anime`**:

```
//here left animation
dist_between_right_left = right.x - left.x;

if (left_action_state== waiting)
{
   left.frame += .3*time;
   if (left.frame >= end_left_frame-1) {left.next_frame = start_left_frame;}
   else {left.next_frame = 0;}
   if (left.frame >= end_left_frame) {left.frame = start_left_frame;}
   goto right_anime;
}
```

You understand it well and there is no need to tell you, that you can do that in any phase.

I know, you are as impatient as me. So let us breath some life into our opponent. Therefore type the following **after `right_anime:`**

```
dist_between_right_left = right.x - left.x;

if (right_action_state == waiting)
{
   right.frame += .3*time;
   if (right.frame >= end_right_frame-1) {right.next_frame = start_right_frame;}
   else {right.next_frame = 0;}
   if (right.frame >= end_right_frame) {right.frame = start_right_frame;}
   goto fin;
}
```

Let us defend ourselves!

To do so, we need do add the following code into our section (**`left_action`**) **after** the movements:

```
if ((key_cuu == 0) && ( left_state_attack_block == 1 )){left_state_attack_block = 0;}

   //left dodge
   if ((key_cud == 1) && ( left_state_attack_block == 0 ))
   {
      if ((left_action_state != dodging) && (left_action_state != dead) && (left_action_state != win))
      {
         //left = dodging
         left_action_state = dodging; //dodging
         left_state_attack_block = 2; //blocking
         start_left_frame = start_dodge;
         left.frame = start_left_frame;
         end_left_frame = end_dodge;

      }
   }
```

And into our **`left_anim`** we insert the following **after** the waiting action:

```
if (left_action_state== dodging)
{
   left.frame += .5*time;
   left.next_frame = 0;
   if (left.frame >= end_left_frame)
```

```
      {
         left_action_state = waiting;
         start_left_frame = start_wait; //waiting
         left.frame = start_left_frame;
         end_left_frame = end_wait;
      }
      goto right_anime;
   }
```

We are ready to deal blows by typing what follows but we should be a little more patient because the animation needs some more explanations.

So go on in **left_action** and continue with:

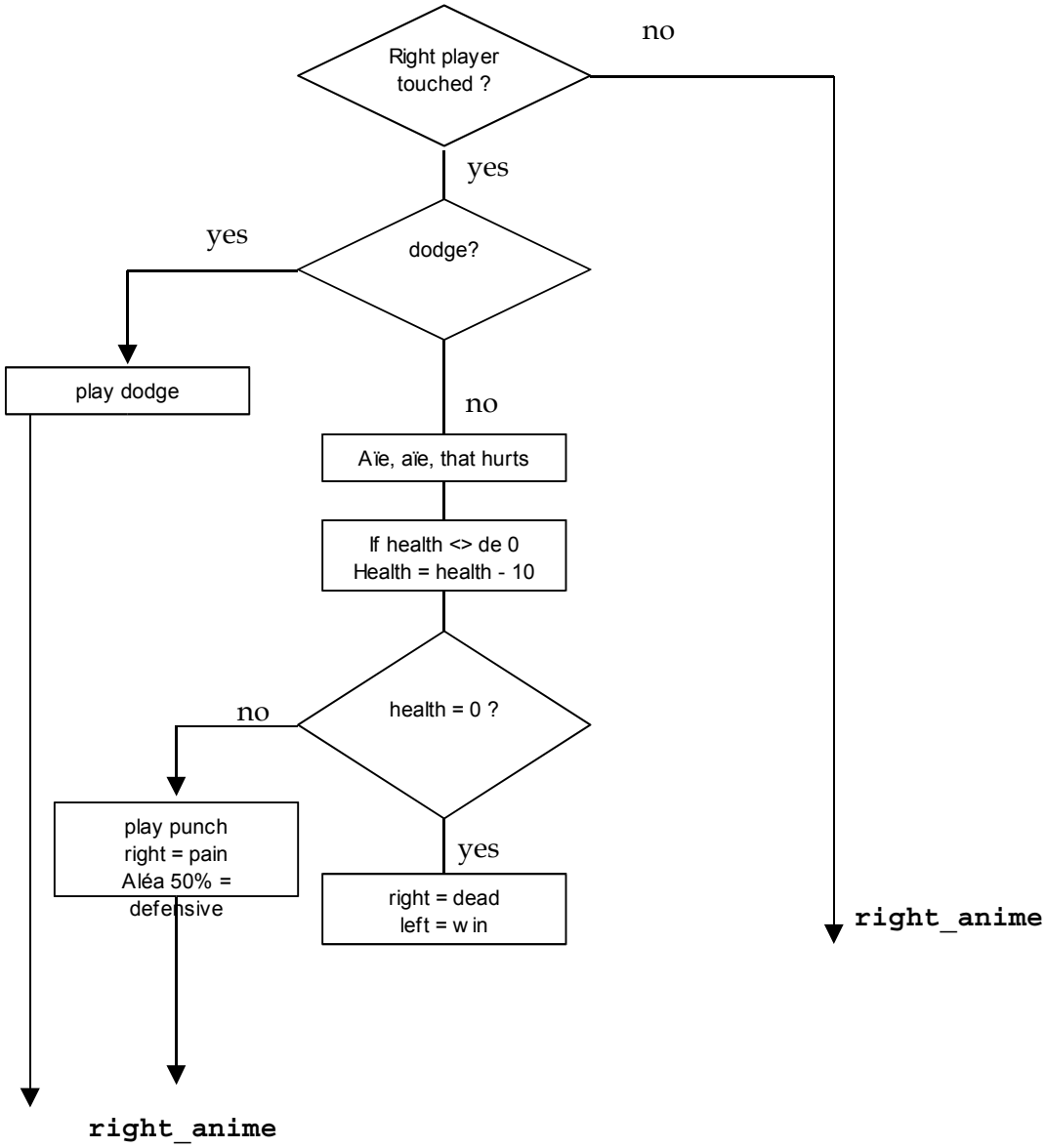
```
      //left attack
      if ((key_cuu == 1) && ( left_state_attack_block == 0 ))
      {

         if ((left_action_state != punching) && (left_action_state != win) && (left_action_state != 
dead))
         {
            //left = punching
            left_action_state = punching;
            left_state_attack_block = 1; //attacking
            start_left_frame = start_punch;
            left.frame = start_left_frame;
            end_left_frame = end_punch;
         }
      }

     if ((key_cuu == 0) && ( left_state_attack_block == 1 )){left_state_attack_block = 0;}
```

Now let us continue the **left_animation** and write and comment the punch animation:

```
   if (left_action_state== punching)
   {
      if (left.frame == start_left_frame){play_sound (punch_dummy_sound,40);}
      left.frame += .5*time;
      left.next_frame = 0;
      if (left.frame >= end_left_frame)
      {
         left_action_state = waiting;
         start_left_frame = start_wait; //waiting
         left.frame = start_left_frame;
         end_left_frame = end_wait;
```

This far everything is all right. One punches and the other keeps on waiting. If you want to try that out (You're all the same big babies), go ahead and take the advantage to smash his head - soon he will be able to defend himself! Close with two brackets and don't forget to indent them.

This is how we continue:



And this is how we write it:

```
//have we hit right player ?
if (dist_between_right_left < min_dist_between_right_left+ 4)
        {
   //right may be dodging
   if ((right_action_state == dodging) && (right.frame > start_dodge + 1))
             {
      play_sound (dodge_sound,100);
      goto right_anime;
   }

   if (right_health == 0){goto right_anime;}

   if (right_health > 0) {right_health -= 10;}

               if (right_health == 0)
               {
```

```
            //right has died
            right_action_state = dead;
            start_right_frame = start_dead;
            right.frame = start_right_frame;
            end_right_frame = end_dead;

            //left has won
            left_action_state = win;
            start_left_frame = start_win;
            left.frame = start_left_frame;
            end_left_frame = end_win;

            goto right_anime;
         }

               play_sound (punch_sound,50);
         pos_camera();

         right_action_state = pain;
         start_right_frame = start_pain;
         right.frame = start_right_frame;
         end_right_frame = end_pain;

         //what should right do ?
         if (random (2) < 1){right_state_of_mind = defensive;}
      }
   }

   goto right_anime;
}
```

And I forbid you to knock him down....

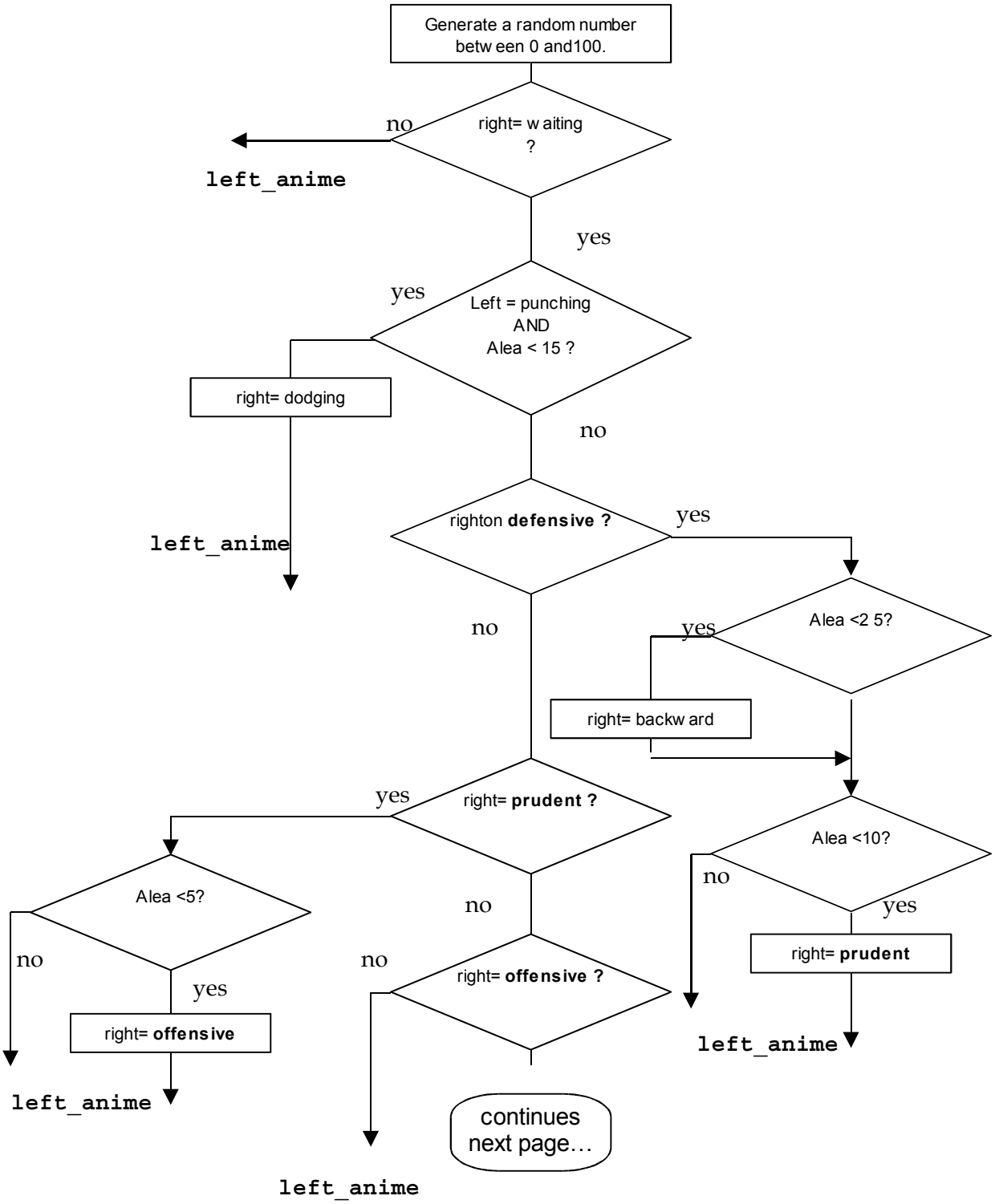
Also our left player can take blows.

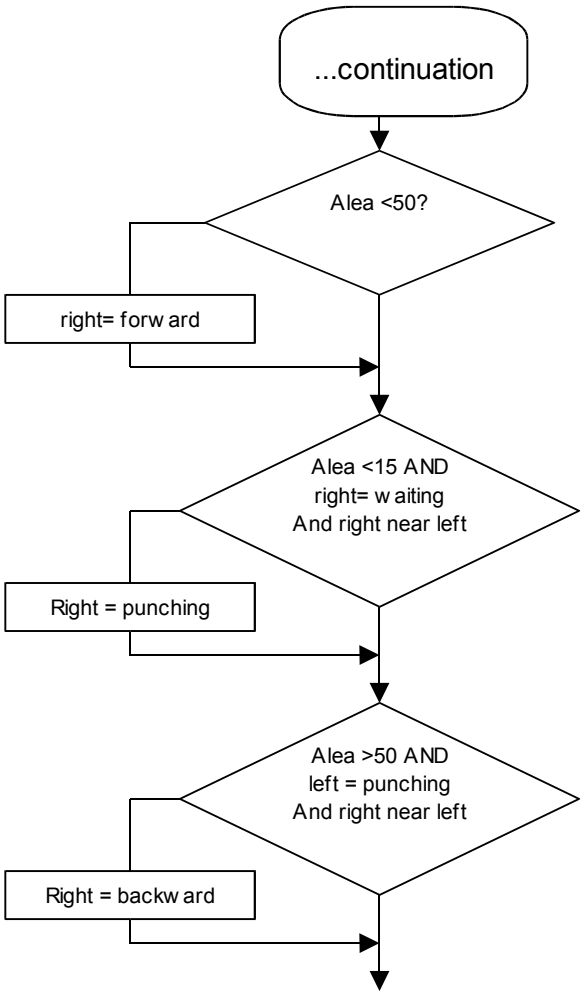
Continue the **`left_animation`** by typing:

```
if (left_action_state== pain)
{
   if (left.frame == start_left_frame)
   {
      play_sound (punch_sound,50);
      if (left.x > left_edge){left.x -= 10;}
   }
   left.frame += .5*time;
   left.next_frame = 0;

   pos_camera();
   if (left.frame >= end_left_frame)
   {
      left_action_state = waiting;
      start_left_frame = start_wait; //waiting
      left.frame = start_left_frame;
      end_left_frame = end_wait;
   }
   goto right_anime;
}
```

We are finished with our left player. (what turned out that he died, but we are not forced to do so).

Concerning the right player we now should analyze his artificial intelligence.





left_anime

The same in WDL code:

```
if (right_action_state != waiting) {goto left_anime;}
   // Random value to determine what right should do based upon its state
   alea = random (100);//0 to 100

   dist_between_right_left = right.x - left.x;

   if ((alea < 15) && (left_action_state == punching) && (right_action_state== waiting))
   {
      right_action_state= dodging;
      start_right_frame = start_dodge;
      right.frame = start_right_frame;
      end_right_frame = end_dodge;
      goto left_anime;
   }

   if (right_state_of_mind == defensive)
   {
      if (alea < 25) //backward
      {
         if (right.x < right_edge)
```

```
      {
         right.x += 2;
         pos_camera();
      }
   }
   if (alea < 10) {right_state_of_mind = prudent;}
   goto left_anime;
}

if (right_state_of_mind == prudent)
{
   if (alea < 5) {right_state_of_mind = offensive;}
   goto left_anime;
}

if (right_state_of_mind == offensive)
{
   if (alea < 50) //forward
   {
      //make sure right can move to left
      if (dist_between_right_left > min_dist_between_right_left)
      {
         right.x -= 2;
         pos_camera();
         //son de pas
      }
   }

   //attack or not attack that is the question ?
   if ((alea < 15) && (right_action_state== waiting) &&
      (left_action_state != dead) && (dist_between_right_left < min_dist_between_right_left + 10 ))
            {
      right_action_state= punching;
      start_right_frame = start_punch;
      right.frame = start_right_frame;
      end_right_frame = end_punch;
   }

   //right backward or not if left = punching
   if ((left_action_state == punching) && (alea > 50) && (right_action_state== waiting)
      && (dist_between_right_left < min_dist_between_right_left+ 10))
   {
      //make sure right can not exceed right edge
      if (right.x < right_edge)
      {
         right.x += 1;
         pos_camera();
      }
   }
}
```

The animation of the red player is almost identical with the blue one, so we type the following lines in **`right_anime` after** the waiting:

```
if (right_action_state== punching)
{
   if (right.frame == start_right_frame){play_sound (swing_sound,40);}
   right.frame += .5*time;
   right.next_frame = 0;
   if (right.frame >= end_right_frame)
   {
      right_action_state = waiting;
```

```
      start_right_frame = start_wait; //waiting
      right.frame = start_right_frame;
      end_right_frame = end_wait;

      //have we hit left player ?
      if (dist_between_right_left < min_dist_between_right_left+ 4)
               {
         //right may be dodging
         if ((left_action_state== dodging) && (left.frame > start_dodge + 1))
                  {
            play_sound (dodge_sound,100);
            goto fin;
         }

         if (left_health <= 0){goto fin;}

         if (left_health > 0) {left_health -= 10;}

                  if (left_health == 0)
                  {
            //left has died
            left_action_state = dead;
            start_left_frame = start_dead;
            left.frame = start_left_frame;
            end_left_frame = end_dead;

            //right has won
            right_action_state = win;
            start_right_frame = start_win;
            right.frame = start_right_frame;
            end_right_frame = end_win;

            goto fin;
         }

         //play_sound (punch_sound,50);
         pos_camera();

         if (left_action_state != pain)
         {
            left_action_state = pain;
            start_left_frame = start_pain;
            left.frame = start_left_frame;
            end_left_frame = end_pain;

         }
      }
   }

   goto fin;
}

if (right_action_state== pain)
{
   if (right.frame == start_right_frame) //rouge recule
   {
      if (right.x < right_edge){right.x += 10;}

   }
   right.frame += .5*time;
   right.next_frame = 0;
   pos_camera();
```

```
      if (right.frame >= end_right_frame)
      {
         right_action_state = waiting;
         start_right_frame = start_wait; //waiting
         right.frame = start_right_frame;
         end_right_frame = end_wait;
      }
      goto fin;
   }

   if (right_action_state== dodging)
   {
      right.frame += .5*time;
      right.next_frame = 0;
      if (right.frame >= end_right_frame)
      {
         right_action_state = waiting;
         start_right_frame = start_wait; //waiting
         right.frame = start_right_frame;
         end_right_frame = end_wait;
      }
      goto fin;
   }
```

Now we have nothing left to do but to manage death and victory for the both of our players.

And here we go:

In **`left_anime`** :

```
   if (left_action_state== dead)
   {
      right.pan = 0; //pour le salut
      left.frame += 1.2*time;
      left.next_frame = 0;
      //son quand touche sol
      if (left.frame >= end_dead)
      {
         //right wins
         left.frame = end_left_frame;
         if (right.frame == start_right_frame){play_sound (right_win_sound,100);}
         right.frame += .5*time;
         right.next_frame = 0;
         if (right.frame >= end_right_frame)
         {
            right_action_state = waiting;
            start_right_frame = start_wait; //waiting
            right.frame = start_right_frame;
            end_right_frame = end_wait;
            right.pan = 185;
            left_action_state = waiting;
            start_left_frame = start_wait; //waiting
            left.frame = start_left_frame;
            end_left_frame = end_wait;
            left_health = 100; //pret à recommencer
            wait(100);
            play_sound (gong_sound,100);
            pos_camera();
         }
      }
      goto right_anime;
   }
```

... and in **right_anime** :

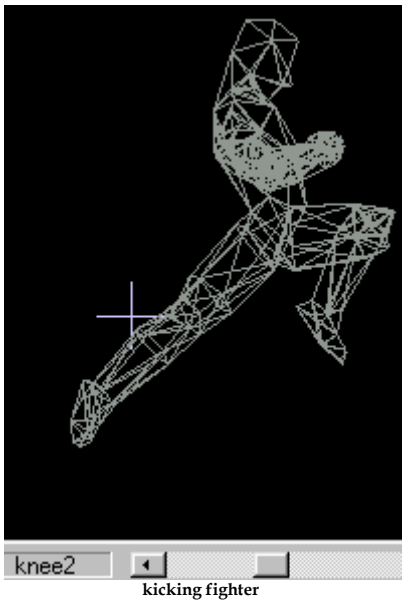
```
if (right_action_state== dead)
{
   right.frame += time;
   right.next_frame = 0;

   //son quand touche sol

   if (right.frame >= end_right_frame)
   {
      //left wins
      right.frame = end_right_frame ;
      if (left.frame == start_left_frame){play_sound (left_win_sound,100);}
      left.frame += .3*time;
      left.next_frame = 0;
      if (left.frame >= end_left_frame)
      {

         left_action_state = waiting;
         start_left_frame = start_wait; //waiting
         left.frame = start_left_frame;
         end_left_frame = end_wait;
         right_action_state = waiting;
         start_right_frame = start_wait; //waiting
         right.frame = start_right_frame;
         end_right_frame = end_wait;
         right_health = 100; //pret à recommencer
         wait(100);
         play_sound (gong_sound,100);
         pos_camera();
      }
   }
   goto fin;
}
```

Conclusion

I would like to say "that's it" which is fairly true. However, we are not going to leave it at that.

What else could be added?

The first thing results from a look at our fighter in MED. As we see, he also can use his knees to kick. If you like to, go ahead and include that on our own - it cannot be really difficult after all.



kicking fighter

In Addition

Now I intend to complete the whole thing and trim it with some fun.
As there are for example different camera views, bird sounds combined with animation for the one who has fallen to the ground, a start screen which offers the choice to select the player's color, background music, audience etc.

Camera effects

Besides the camera view, we already have, we plan two supplementary camera views. One from behind the left player, the second one with the camera on his shoulder. A third view will show our haed every time, we get hit. So we obtain four views alltogether.

Therefore we create a variable:

```
var cam_view = 0; //default
```

Then at the end of our script we write:

```
on_f1 = cam_view1;
on_f2 = cam_view2;
on_f3 = cam_view3;
on_f4 = cam_view4;
```

```
function cam_view1()
{
   if (cam_view > 3){cam_view = 4;}
   else {cam_view = 0;}

   //cam_view = 0;
   pos_camera();
}

function cam_view2()
{
   if (cam_view > 3){cam_view = 5;}
   else {cam_view = 1;}
   //cam_view = 1;
   pos_camera();
}

function cam_view3()
{
   if (cam_view > 3){cam_view = 6;}
   else {cam_view = 2;}
   //cam_view = 2;
   pos_camera();
}

function cam_view4()
{
   if (cam_view > 3){cam_view -=4;}
   else {cam_view +=4;}
   pos_camera();
}
```

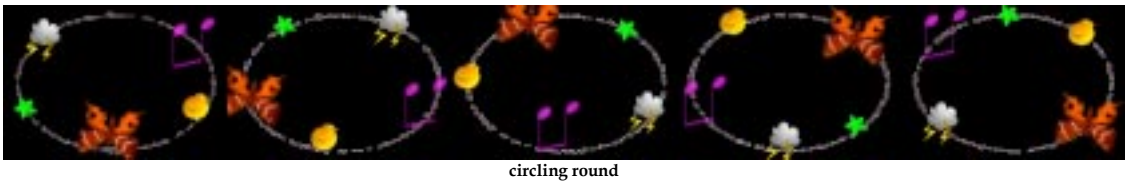
Then we replace our function **pos_camera** by this one:

```
function pos_camera()
{
   cam_left.x = right.x;
   cam_left.y = right.y;
   cam_left.z = 60;
   cam_left.pan = 180;
   cam_left.arc = 40;
   cam_left.pos_x = screen_size.x - 150;


   if ((cam_view == 1)|| (cam_view == 5))
   {
      camera.pan=0;
      camera.z = 70;
      camera.x = left.x;
      camera.y = left.y;
      //cam_left.visible = off;

      wait (1);
   }

   if ((cam_view == 2)|| (cam_view == 6))
   {
      camera.pan=15;
      camera.z = 70;
      camera.x = left.x-70;
      camera.y = left.y-60;
```

```
      //cam_left.visible = off;

      wait (1);
   }

   if ((cam_view == 0)|| (cam_view == 4))
   {
      camera.pan=90;
      camera.z = 70;
      camera.x = ((right.x+left.x)/2);
      camera.y = -((right.x-left.x) + 50);
      //cam_left.visible = off;

      wait (1);
   }
   if (cam_view > 3){cam_left.visible = on;}
   else {cam_left.visible = off;}
}
```

A little bit of animation

Let us catch the moment when small birds twitter. I craeted a bitmap named **ko+5.pcx** which we are going to include now and then display at the right moment.



circling round

We include these lines to define our variables at the beginning of the scenario:

```
bmap cui_cui_map,<ko+5.pcx>;
var cui_cui_pos = 0;
var  cui_cui_x_left;
var  cui_cui_y_left;
var  cui_cui_x_right;
var  cui_cui_y_right;

panel cui_cui_pan
{
      layer = 1;
   window = 0,0,84,58,cui_cui_map,cui_cui_pos.x,0;
         flags = d3d,overlay,refresh;
}
```

Now we are creating a handle for our sound (right after sound definitions), which enables us to check whether the sound has stopped playing so we can continue:

```
var larkhandle = 0;
```

Then we create the function **cui_cui** which we place at the end:

```
function show_cui_cui()
{
   while (1)
   {
      cui_cui_pos.x += 84;
```

```
      if (cui_cui_pos.x >= 400){cui_cui_pos.x =0;}
      waitt (10);
   }
}
```

Then we modify (spaced red lines) our function **pos_camera** as follows:

```
function pos_camera()
{
   cam_left.x = right.x;
   cam_left.y = right.y;
   cam_left.z = 60;
   cam_left.pan = 180;
   cam_left.arc = 40;
   cam_left.pos_x = screen_size.x - 150;

   if ((cam_view == 1)|| (cam_view == 5)) //touche F2
   {
      camera.pan=0;
      camera.z = 70;
      camera.x = left.x;
      camera.y = left.y;
      //cam_left.visible = off;

      cui_cui_x_left = 380;
      cui_cui_y_left = 400;
      cui_cui_x_right = 180;
      cui_cui_y_right = 400;

      wait (1);
   }

   if ((cam_view == 2)|| (cam_view == 6))//touche F3
   {
      camera.pan=15;
      camera.z = 70;
      camera.x = left.x-70;
      camera.y = left.y-60;
      //cam_left.visible = off;

      cui_cui_x_left = 160;
      cui_cui_y_left = 320;
      cui_cui_x_right = 100;
      cui_cui_y_right = 400;

      wait (1);
   }

   if ((cam_view == 0)|| (cam_view == 4))// touche F1
   {
      camera.pan=90;
      camera.z = 70;
      camera.x = ((right.x+left.x)/2);
      camera.y = -((right.x-left.x) + 50);
      //cam_left.visible = off;

      cui_cui_x_left = 380;
      cui_cui_y_left = 400;
      cui_cui_x_right = 180;
      cui_cui_y_right = 400;
```

```
      wait (1);
   }
   if (cam_view > 3){cam_left.visible = on;}
   else {cam_left.visible = off;}
}
```

And we modify our death in consequence:

```
      if (right_action_state== dead)
      {

         right.frame += time;
         right.next_frame = 0;

         //son quand touche sol

         if (right.frame >= end_right_frame)
         {
            //left wins
            right.frame = end_right_frame ;
            if (left.frame == start_left_frame)
            {
               if (larkhandle ==0) //le son n'est pas joué
               {
                  play_sound (lark_sound,100);
                  larkhandle = result;
                  cui_cui_pan.visible = on;
                  cui_cui_pan.pos_x = cui_cui_x_right;
                  cui_cui_pan.pos_y = cui_cui_y_right;
                  show_cui_cui();
               }
               if (snd_playing(larkhandle) == 1){goto fin;}
               cui_cui_pan.visible = off;
               larkhandle = 0;
            }

            if (left.frame == start_left_frame){play_sound (left_win_sound,100);}
            left.frame += .3*time;
            left.next_frame = 0;
            if (left.frame >= end_left_frame)
            …
```

and

```
      if (left_action_state== dead)
      {
         //if (left.frame >= start_dead+10){play_sound (lark_sound,100);}
         right.pan = 0; //pour le salut
         left.frame += 1.2*time;
         left.next_frame = 0;
         //son quand touche sol
         if (left.frame >= end_dead)
         {
            left.frame = end_left_frame;
            if (right.frame == start_right_frame)
            {
               if (larkhandle ==0) //le son n'est pas joué
               {
                  play_sound (lark_sound,100);
                  larkhandle = result;
                  cui_cui_pan.visible = on;
```

```
      cui_cui_pan.pos_x = cui_cui_x_left;
      cui_cui_pan.pos_y = cui_cui_y_left;
      show_cui_cui();
   }
   if (snd_playing(larkhandle) == 1){goto right_anime;}
   cui_cui_pan.visible = off;
   larkhandle = 0;
}
//right wins
if (right.frame == start_right_frame){play_sound (right_win_sound,100);}

right.frame += .5*time;

right.next_frame = 0;
```

A start screen

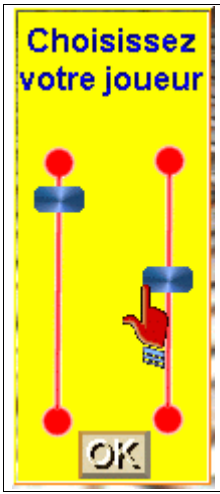
This screen is going to allow us to choose the player's color. So first of all, we need to create new skins for our persons. Open MED, export one of our two skins to your favorite paint program. Now go ahead and create skins of other colors.

I added for my part the following skins (in this order):

```
red (of origin) = 1
blue (of origin) = 2
grey =3
green = 4
yellow = 5
cyan = 6
```

If you are not equipped, I also provided this model under the name of **`fighter1.mdl`**.

Now it's time to create our panel to give us the choice. Well, here you are:



It consists of 5 elements:

1. The background of the panel (**`choix.pcx`**)*
2. A cursor used twice (**`cursor.pcx`**)
3. A button 'o.k on' (**`button_ok_on.pcx`**)
4. A button 'o.k off' (**`button_ok_off.pcx`**)
5. A mouse pointer (**`pointeur.pcx`**)

* french version
take **`choix_e.pcx`** for the english panel
or **`choiw_d.pcx`** for the german panel

We define them as follows:

```
bmap choix_map,<choix.pcx>; // or choix_e.pcx (english version) or choix_d.pcx (german version)
bmap button_ok_on_map,<button_ok_on.pcx>;
bmap button_ok_off_map,<button_ok_off.pcx>;
bmap cursor_map,<cursor.pcx>;
bmap pointeur,<pointeur.pcx>;
```

We also define the two variables which will be fed by **vslider** :

```
var coul_left=1;
var coul_right=4;
```

Now we define our panel:

```
panel choix_pan
{
      layer = 1;
      bmap = choix_map;
   button = 33,212,button_ok_on_map,button_ok_off_map,button_ok_off_map,game,null,null;
   vslider = 11,90,100,cursor_map,1,3,coul_left;
   vslider = 65,90,100,cursor_map,4,6,coul_right;
   mouse_map = pointeur;
         flags = d3d,overlay,refresh;
}
```

According to this definition you can note that:

1 – By clicking the button 'OK', we call the function **game()**
2 – the left slider toggles through the values 1 - 3 (3 possible skins for our left player)
3 – The right slider toggles through the values 4 - 6 (3 possible skins for our right player).

The only thing we are left to do is to put it all to work:

In the **main()** function, we replace **game();** by

```
choix();
```

Then we write our function **choix()**.

```
function choix()
{
   choix_pan.pos_x = (screen_size.x/2) -50;
   choix_pan.pos_y = (screen_size.y/2) -100;

   choix_pan.visible = on;
   mouse_mode = 1;
   mouse_on();
   mouse_spot.x = 18;
   mouse_spot.y = 6;
   while(mouse_mode != 0)
   {
      left.skin = int(coul_left);
      right.skin = int(coul_right);
      alea = random(100);
   wait (1);
   }
}
```

The first two lines fix the panel in the center of the screen depending on the resolution. Then we display the panel and next the mouse. At least we assign the skin color to the left respectively the right player depent on the value returned by the two sliders.

The random line is an old trick which I allways used if computers had no internal clock. Actually, you might have raelized it, the random instruction allways gives the same sequence of numbers

at every execution of the program. In our case we will never spend twice the same time in our loop. Because the time depends on the moment at which one clicks on OK, we are assured to have really random random numbers.

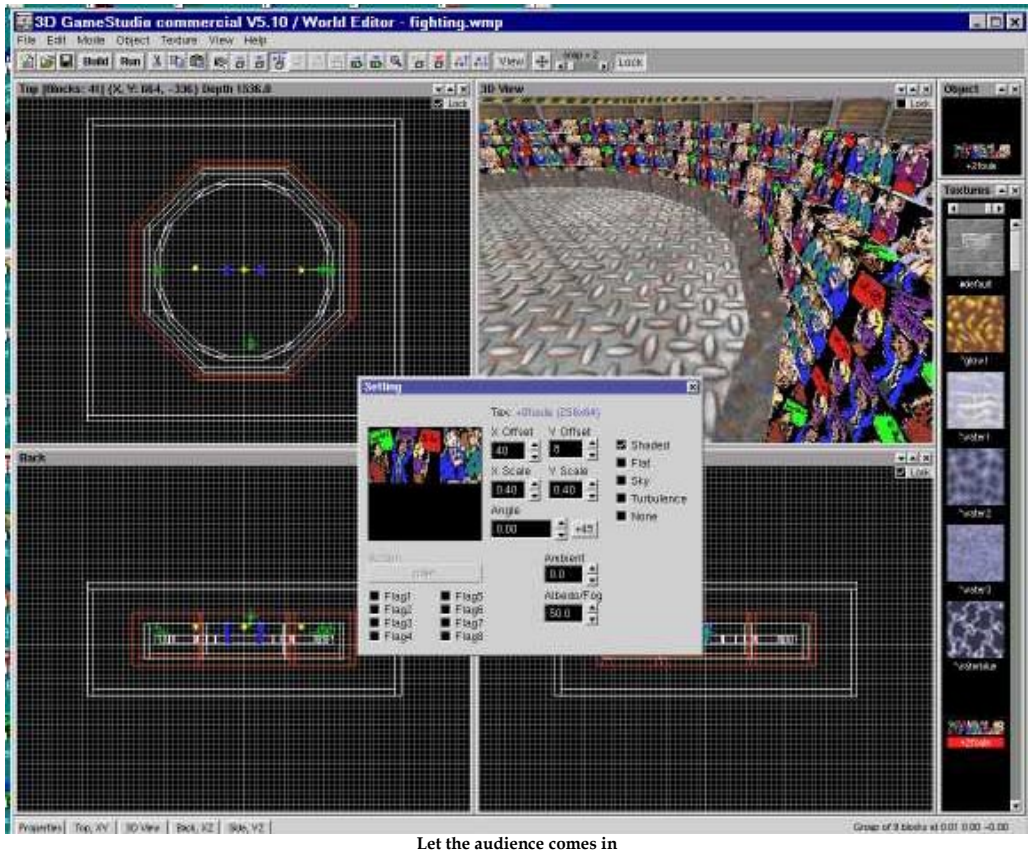
Now we quickly include these two lines at the beginning of our function **`game()`** and we are ready to play, using the color of our choice.

```
mouse_mode = 0;
choix_pan.visible = off;
```

I leave it to you as an exercise to adapt the color of the health bar. It is identical with the adaption of the player's color (for the moment, we keep to red and blue).

The audience

As for the audience, I am not going to apply details. I just added 2 hollow cylinders for the stands and attached an animated texture (in your directory files: **`+0foule.pcx`** in **`+3foule.pcx`**).



Let the audience comes in

The result is nevertheless pleasant

The background music

The background music does not make much problems either:

```
music amb_song = <ambiance.mid>;

// play a song if volume was set at beginning
function start_song()
{
   wait(4); // wait until load_info has loaded the last volume setting
   if (midi_vol > 0) {
      play_song(amb_song,80);
   }
}
```

Type the following line in **`main()`** after **`load_status:`**

```
start_song();
```

Conclusion (2)

This time we're done, I think. There is nothing left, than to say goodbuye...

However, you might think it would be nice if every guy had his own personality that goes with his color and therefore different reactions. What a stupid idea! Just go ahead...

Well, I feel like helping you a little to grow some more into WED's magnificent world. It's a deal, but afterwards it's up to you.

We are going to make it simple:

- **First player** :
- The weak one, whose hits deduct only 5 of the opponent's life points and every received blow makes him loose 5 more points than the oppont in this case. But be so kind to leave him the chance of a lucky win: we are going to give him a secret boot! So a kick with the knee is unstoppable and knocks down the opponent. (The kick with the knee replaces the punch in an unpredictable way: f.i. with a probability of 1 to 10).
-
- **Second player**:
- the one that we know at present , so we shall change nothing.
-
- **Third player:**
- the brutal one, every hit causes a deduction of 15 off the oppontnt's life points, received blows let him loose 5 points less than than the other. He can dodge the kick of a knee by rapid backward movement.

What gives us:

Blow given by	Blow received by	Bonus/malus	Points of life to be removed
Weak = 5 points	Weak	+5	10
	Normal	0	5
	Brutal	-4	1
Normal = 10 points	Weak	+5	15
	Normal	0	10
	Brutal	-4	6
Brute = 15 points	Weak	+5	20
	Normal	0	15
	Brutal	-4	11

The hits given by the left player **`left.skin*5`** result in 5 for the weak, 10 for the normal and 15 for the brutal one.
The hits given by the right player **`(right.skin-3)*5`** result in 5 for the weak, 10 for the normal and 15 for the brutal one.

We create two variables:

```
var coup_recu_left ;
var coup_recu_right ;
```

Then we calculate their values: (include these lines at the beginning of the function **`game()`**)

```
if (left.skin == 1){coup_recu_left = ((right.skin-3)*5)+5;}
if (left.skin == 2){coup_recu_left = (right.skin-3)*5;}
if (left.skin == 3){coup_recu_left = ((right.skin-3)*5)-4;}

if (right.skin == 4){coup_recu_right = (left.skin*5)+5;}
if (right.skin == 5){coup_recu_right = (left.skin*5);}
if (right.skin == 6){coup_recu_right = (left.skin*5)-4;}
```

For the haelth points, we simply replace the following blue lines by the red ones:

```
if (right_health > 0) {right_health -= 10;}

            if (right_health == 0)
```

by :

```
if (right_health > 0) {right_health -= coup_recu_right;}

            if (right_health <= 0)
```

and

```
if (left_health > 0) {left_health -= 10;}

if (left_health == 0)
```

by:

```
if (left_health > 0) {left_health -= coup_recu_left;}

            if (left_health <= 0)
```

and

```
bmap choix_map,<choix.pcx>;
```

by:

```
bmap choix_map,<choix1.pcx>;
```

The kicks with the knees:

The right player can kick us with his his knees. Rather than to write a supplementary action, I prefer to modify the punch action: once of 10 times the punch is replaced by the knee.

At first we create the following variables which we place after the existing ones:

```
var start_win = 1;
var end_win = 10;
var start_dead = 73;
var end_dead = 88;
var start_knee = 56;
var end_knee = 64;

var pain = 4;
var dead = 5;
var win = 6;
var knee = 7;
var knee_state = 0 ;
```

Then we add the lines in red:

```
if ((alea < 15) && (right_action_state== waiting) &&
      (left_action_state != dead) && (dist_between_right_left < min_dist_between_right_left + 10 ))
            {
      right_action_state= punching;
      start_right_frame = start_punch;
      right.frame = start_right_frame;
      end_right_frame = end_punch;
```

```
        if ((random(10) < 1) && (right.skin == 4))
        {
        right_action_state= punching;
        knee_state = 1;
        start_right_frame = start_knee;
        right.frame = start_right_frame;
        end_right_frame = end_knee;
        }
     }
.....

              if (left_health > 0) {left_health -= coup_recu_left+(50*knee_state);}
if (knee_state == 1) {knee_state = 0;}

                 if (left_health <= 0)
                        {
```

Concerning the left player we can decide either to do it the same way (replacing punch by kick) or to add another key to control the kicks. I have chosen the second option and so you can kick with the knee by hitting the **CTRL.** -key.

We type the following lines before the **`right`** action.

```
     //left attack with knee
     if ((key_ctrl == 1) && ( left_state_attack_block == 0 ) && (left.skin
== 1))
     {


       if ((left_action_state != punching) && (left_action_state != win)
&& (left_action_state != dead))
       {
         //left = punching
         left_action_state = punching;
         left_state_attack_block = 1; //attacking
         knee_state = 1;
         start_left_frame = start_knee;
         left.frame = start_left_frame;
         end_left_frame = end_knee;
       }
     }



     if   ((key_cuu   ==   0)   &&   (   left_state_attack_block   ==   1
)){left_state_attack_block = 0;}
     //here right action
---------- - - -  - -- - - --- -
```

Then we are going to modify these lines:

```
if (right_health > 0) {right_health -= coup_recu_right+(20*knee_state);}
     if (knee_state == 1) {knee_state = 0;}

   if (right_health <= 0)
           {
```

Conclusion (3)

Actually wa are done for real! At last we say goodbuye...

And after a well playing game What else is there to do?

- Establish a high score list to compete against your friends or family.
- Show impact points by using particles
- Vary the animation speed according to the the player's force: a weak character has slower motions than a strong one.
- The same applies to the sound: it should be proportional to the player's force as well.
- Etc.

let the fight begin!

PS: you will find the complete files of scenario under the name of **`fighting.all`** .