3D GameStudio

Les Ateliers de combat

pour A5 Engine 5.10
par Alain Brégeon August 2001

Les dernières nouvelles, les démonstrations, les mises à jour et les outils, aussi bien que le Magazine des Utilisateurs, le Forum des Utilisateurs et le concours annuel sont disponibles à la page principale GameStudio **http://www.3dgamestudio.com**.

Contente

Avant propos de la part de l'auteur

Cher Lecteur,

J'ai produit cet atelier pour vous aider à répondre à la question "Comment faire un jeu de combat avec 3DGameStudio ?". Cet atelier utilise des possibilités disponibles à partir de la version (4.22) ou supérieure.

Cet atelier, comme les autres ateliers avant cela, vise surtout les utilisateurs qui ont un peu d'expérience de 3DGameStudio. Je suppose que vous avez travaillé les différents tutoriaux et savez comment employer les outils (WED, MED et WDL).

Ce texte complète la documentation qui va avec 3DGameStudio, et ne la remplace pas. Si quelque chose dans cet atelier est peu clair lisez s'il vous plait les manuels qui sont fournis avec 3DGameStudio. Je fais par avance des excuses pour des formulations que vous trouveriez peu claires, un code défectueux, des erreurs ou des omissions.

J'espère que vous trouvez les ateliers informatifs et agréables.

Alain Brégeon
mailto:alainbregeon@hotmail.com

La philosophie du combat employée dans ce tutorial est largement inspirée de l'exemple Rockem fourni avec le SDK C++ de Microsoft©.

Obtenez la dernière version

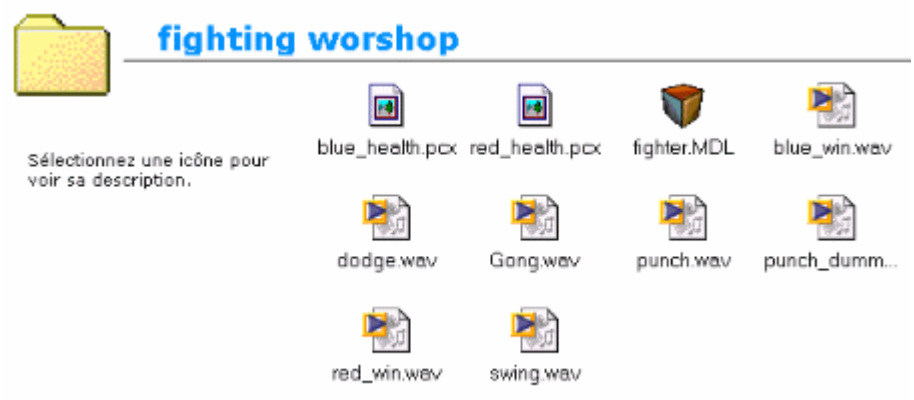
Avant de commencer, assurez-vous d'avoir la dernière version de 3DGameStudio (**4.25** ou au-dessus) puisque nous allons nous servir de plusieurs des nouvelles particularités qui ont été ajoutées. Aussi vous aurez besoin du dernier constructeur de monde, parce que pour un simulateur de vol nous utiliserons un monde extrêmement énorme.

Préparez votre workspace

Créez un dossier appelé "fighting Workshop" dans votre dossier GSTUDIO. C'est le répertoire ou vous stockerez tous les éléments du jeu.

La première chose que nous allons ajouter à notre dossier est le modèle de jeu. Si vous ne les avez pas déjà, allez à la page de téléchargement de Conitec (http: // www.conitec.net/a4update.htm) et récupérez le niveau de Fighting. Décompressez le contenu dans votre dossier. Votre dossier doit maintenant contenir au moins les fichiers:

```
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
```



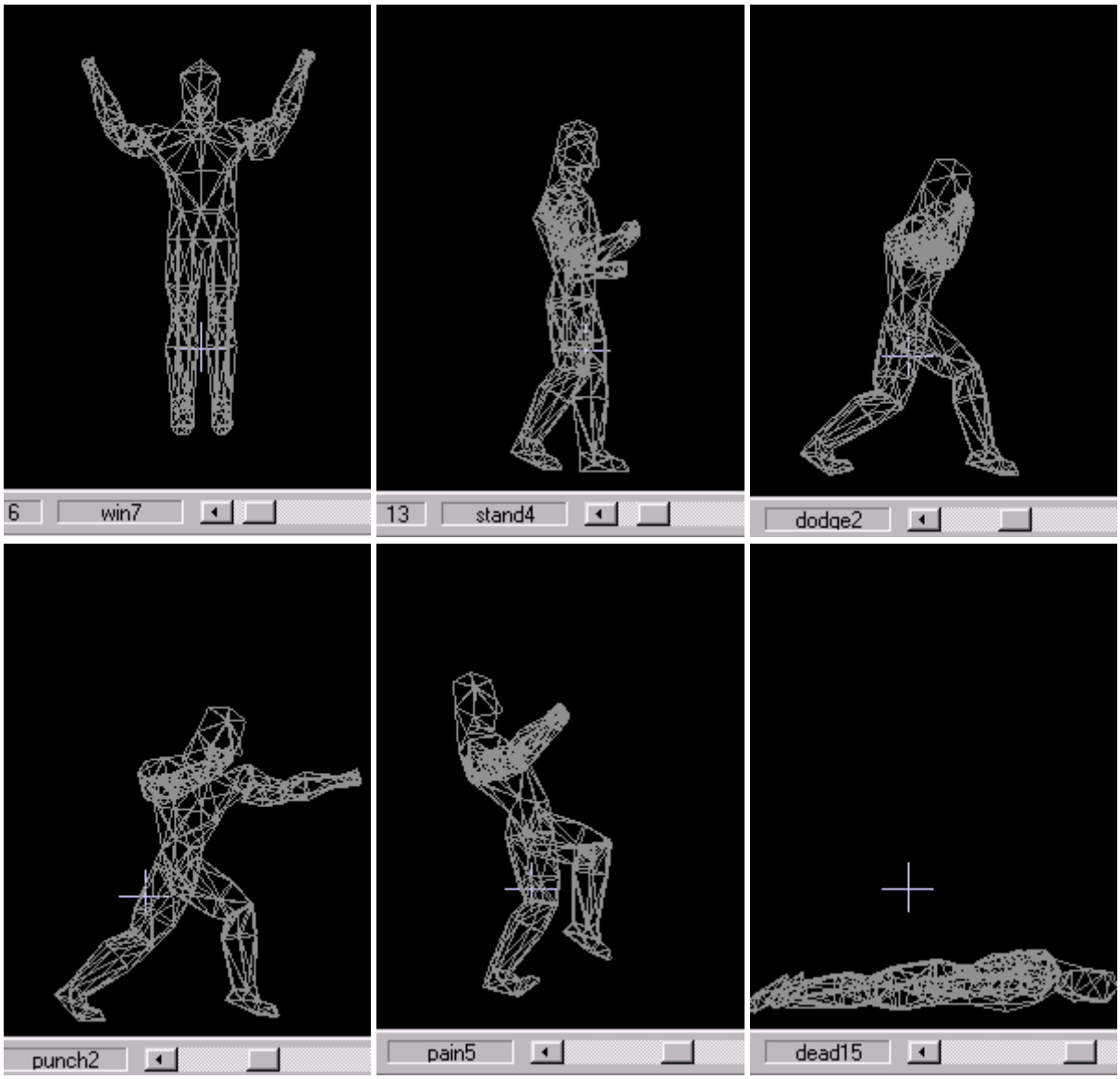
répertoire Fighting Workshop

Notre Combattant

Il est bien connu que faute de combattants, les combats cessèrent. Si nous voulons créer un jeu de combat il va donc nous falloir des combattants.

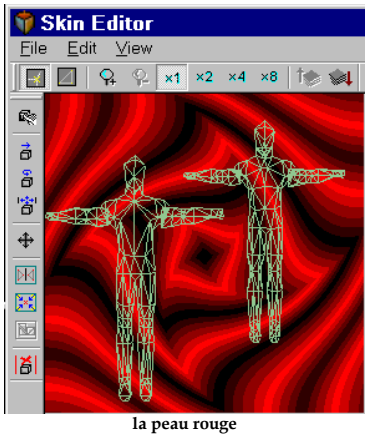
Notre personnage doit pouvoir esquiver, donner des coups, recevoir des coups, attendre, mourir et gagner. Le personnage que je fournis fait tout cela. Pour les peaux et pour la bonne compréhension du tutorial, j'ai fait au plus simple, une peau rouge et une peau bleue.

Ouvrez MED, chargez **`fighter.mdl`** et observez tout ce que peut faire ce combattant.



fighter.MDL

Il est bien entendu qu'il pourrait faire beaucoup plus mais pour ne pas alourdir ce tutorial nous nous contenterons de ces 6 actions.



la peau rouge

Création du niveau de combat

Nous avons l'embarras du choix, un ring, la rue, la lune … La seule contrainte imposée est de disposer de suffisamment de champ pour la caméra. Voici ce que j'ai choisi personnellement :

Ouvrez WED et choisissez **File-> New**

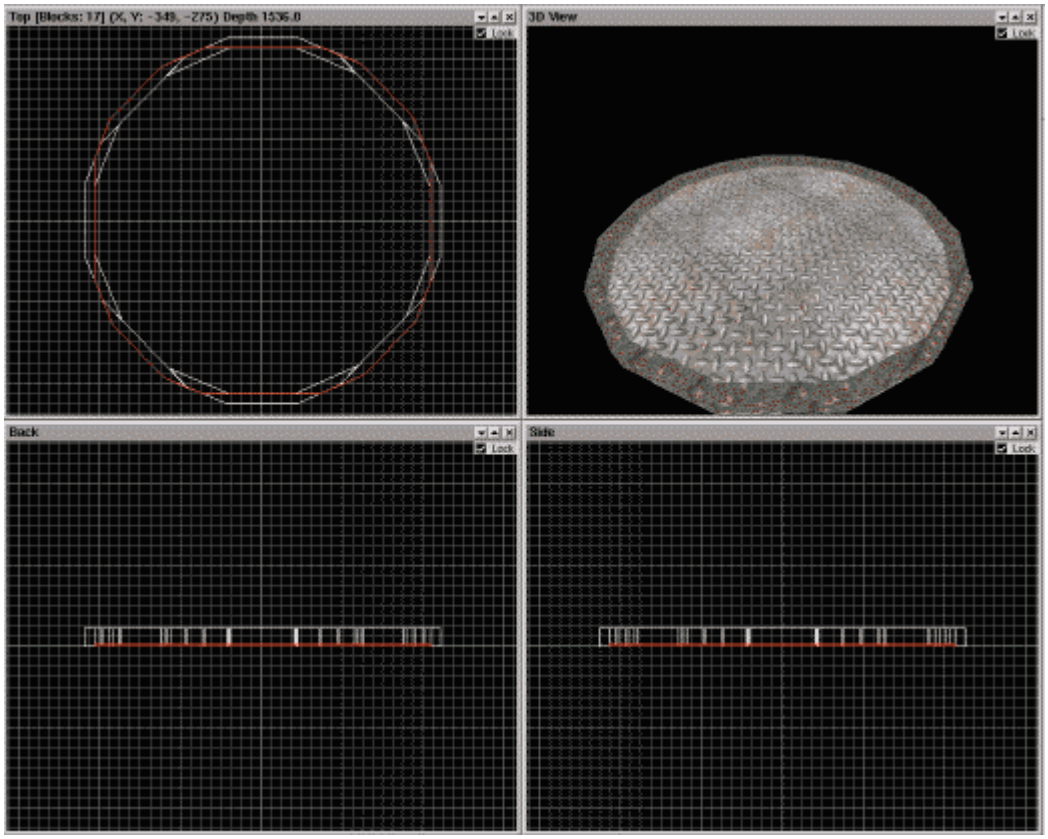
Ouvrez le manager de texture (**Texture-> Texture Manage**). Sélectionnez **add WAD**, et ajoutez le **`standard.wad`**. Fermez le manager de texture.

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

Agrandissez le cercle pour obtenir un diamètre de 5 grands carrés environ.

On le creuse par **Alt H**, on réduit sa hauteur à 2 petits carrés, scope down, on supprime le plafond

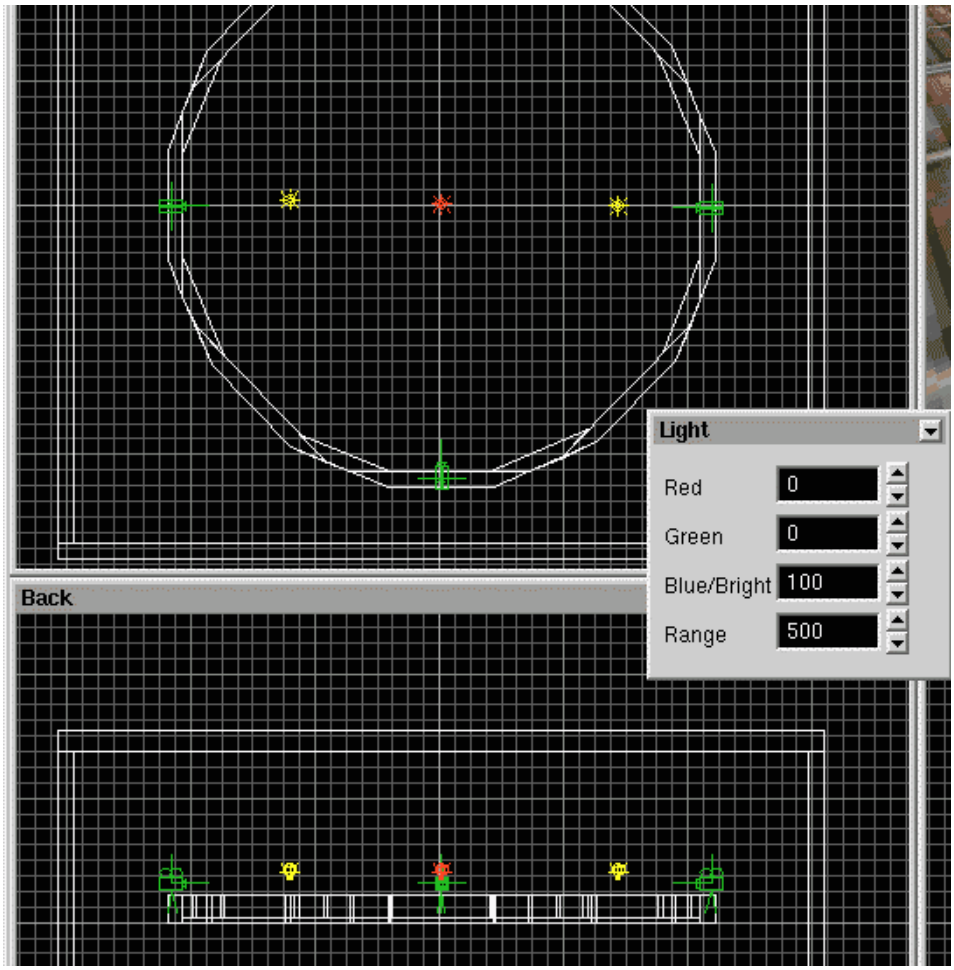
On utilise les textures standard, sur les côtés j'ai mis la texture **`metalrivet2`** et sur le sol j'ai mis la texture **`metalribbed5`**.



Puis nous créons une boite 'ciel' qui n'est pas obligatoire mais il faut garder les bonnes habitudes.

Choisissez **Add Primitive -> Cube large** et recouvrez votre cylindre de chaque coté en laissant un peu de hauteur. Creusez le par **alt+H** et appliquez une texture (**`gate1`** par exemple)

Nous positions 2 ou 3 lumières. **Add Light** que nous positionnons au-dessus de la piste. Nous allons à présent positionner 3 positions avec le plus grand soin. Première position sur le bord gauche de notre piste, deuxième position sur le bord droit et troisième position sur le bord du bas, comme ci-dessous.

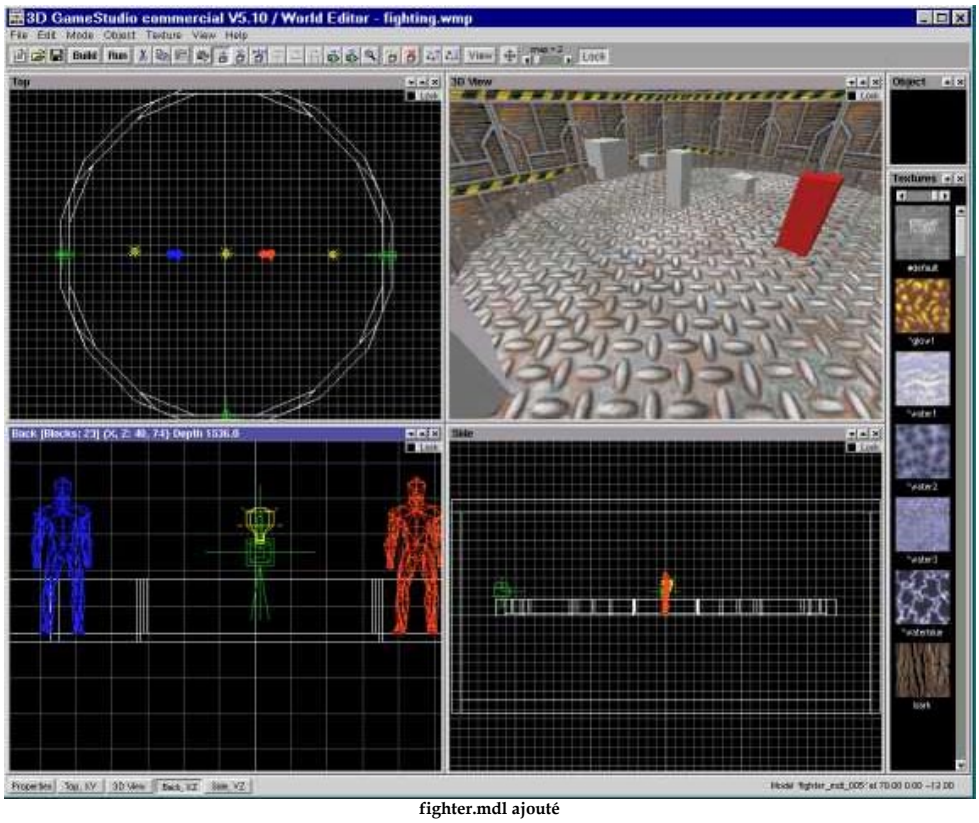


Je vous sens impatient de voir à quoi ressemble votre niveau, allons-y, sauvegardez sous le nom de **fighting** puis **build** en cochant **fly-thru = enable**. Voici le résultat. Plutôt sympa, non ?

Ajout des combattants

Il ne nous reste plus qu'à placer les 2 combattants et nous aurons pratiquement terminé. (Pas tout à fait quand même).

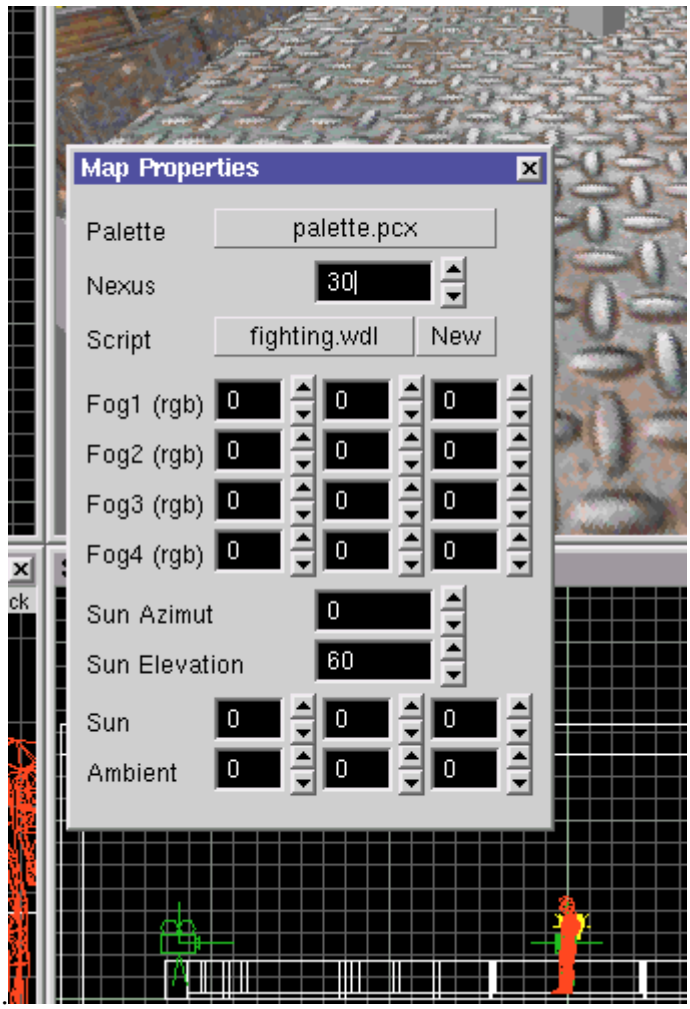
Select **Object-> Load Entity** et utilisez les fenêtres de navigation de fichier pour trouver le modèle **`fighter.mdl`**. Le modèle doit apparaître dans votre monde. Placez-le près du centre de votre sol pour qu'il repose sur la surface.



fighter.mdl ajouté

Création de votre script

Créez un scénario pour votre niveau. Ouvrez votre fenêtre de Propriétés de Carte (**File-> MapProperties**) et appuyez sur bouton **new**. Le bouton à côté du Scénario doit changer de **`ndef`** à **`fighting.wdl`**



Nouveau Scénario dans Carte Properties

Ouvrez votre dossier de niveau sélectionnez et ouvrez (double-clic) le fichier **`fighter.wdl`**. Si Windows demande quelle application employer pour ouvrir le fichier, choisissez **bloc-notes** (ou n'importe quel éditeur de texte simple).

Vous devez remarquer que le jeu typique, "modèle" a été créé pour vous. C'est suffisant pour la plupart des projets mais nous voulons faire quelque chose de plus avancé cette fois. Allons-y et sélectionnez tout (dans le bloc-notes de Microsoft employez **Edit-> select All**) et cliquez sur la touche "suppr". Maintenant nous allons tout recommencer à zéro.

Ajout d'un chemin

Commençons en définissant les chemins que notre programme va employer. Les chemins sont employés pour dire au moteur où il peut placer les fichiers employés dans notre projet (des images, des sons, d'autres scénarios, etc.). Le dossier ou nous sommes ("Flight Sim Workshop") est déjà inclus, aussi, dans notre cas, nous devons seulement ajouter le répertoire **`template1`**.

Tapez la ligne suivante :

```
path  "..\\template";    // Chemin du sous répertoire templates de WDL
```

Notez que tous les chemins sont relatifs à notre dossier de niveau. La ligne ci-dessus dit ceci, "remonte d'un niveau (" ..") et rightescend dans le dossier template (**\\ template**)".

Ajoutez les "include"

Après que nous ayons créé nos chemins nous devons ajouter nos fichiers "include". Tapez sous **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
```

La commande **include** dit au moteur de remplacer cette ligne avec le contenu du fichier entre (<...>). C'est comme ci nous étions allés chercher le fichier en question, avions copié tout son code, et l'avions collé dans le scénario.

C'est un outil puissant parce qu'il nous permet de réutiliser le code d'autres projets et permet de faire des mises à jour dans les fichiers inclus sans devoir récrire votre code. Par exemple, la mise à jour 4.19 inclut du code pour permettre au joueur de nager dans l'eau. Ainsi n'importe quel projet qui inclut **movement.wdl** peut employer ce nouveau code pour la nage.

Comme pour les chemins, l'ordre des lignes **include** est important. Puisque quelques scénarios utilisent des valeurs qui sont déclarées dans d'autres scénarios. Par exemple : **actors.wdl** emploie la variable, **force** qui est déclarée dans **movement.wdl**. Si nous mettons **actors.wdl** avant **movement.wdl** nous obtiendrions des erreurs.

C'est bien d'INCLURE des scénarios même si vous n'employez aucune de leurs particularités dans votre code. La plupart des fichiers dans template sont interdépendants l'un sur l'autre aussi si vous faites des projets pour utiliser l'un d'entre eux, vous devez les **inclure** tous pour être sûrs. L'exception à cette règle est le scénario **venture.wdl**, qui n'est employé par aucun des autres scénarios, mais emploie chacun d'eux.

Les valeurs de départ du moteur

Maintenant nous allons mettre quelques valeurs importantes qui aideront à décider comment le simulateur sera montré. Ces valeurs déterminent la résolution, la profondeur des couleurs, le taux de rafraîchissement et l'éclairage. Ajoutez les lignes suivantes au-dessous des lignes **include**:

```
 // Valeurs de départ du moteur
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
```

Nos variables de jeux

C'est ici que nous entrerons nos variables de jeux au fur et à mesure de nos besoins

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

L'affichage du logo A4/A5

Nous préparons l'affichage du logo qui se trouve dans le répertoire templates

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

La Fonction Principale "Main"

Dans n'importe quel projet vous avez besoin de la fonction **main** (principale). C'est la première fonction à être appelée lorsque le programme démarre. Dans la plupart des cas la fonction principale est très simple, la notre ne fera pas exception. Entrez s'il vous plaît les lignes suivantes (sous votre dernière ligne) :

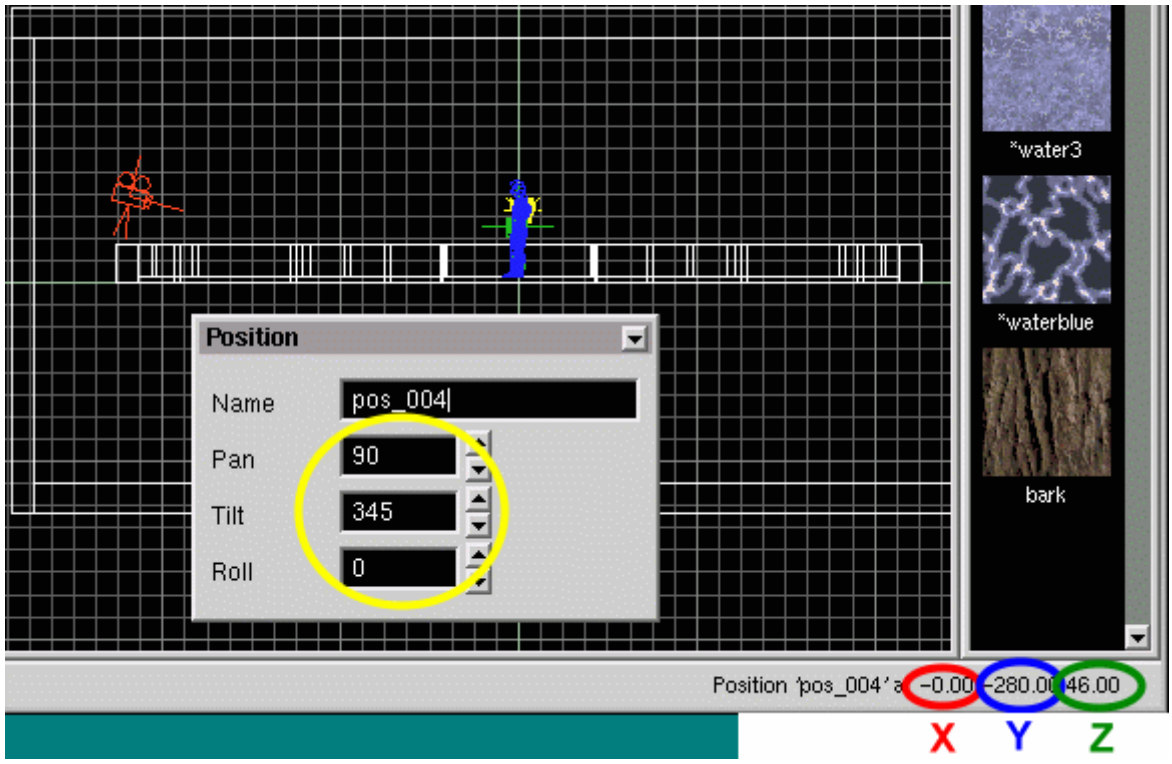
```
function main()
{
   fps_max = 50;
   warn_level = 2;    // announce bad texture sizes and bad wdl code
   tex_share = on;    // map entities share their textures

// 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;
// set it visible
   splashscreen.visible = on;
// wait 3 frames (for triple buffering) until it is renderight and flipped to the foreground
   wait(3);

// now load the level
   load_level (<fighting.wmb>);
// wait the requiright second, then switch the splashscreen off.
   waitt(16);
   splashscreen.visible = off;
   bmap_purge(splashmap);   // remove logo bitmap from video memory

// load some global variables, like sound volume
   load_status();
```

Chacune de ces lignes est commentée et n'appelle pas d'explications supplémentaires.

Position initiale de notre caméra

Nous allons utiliser notre position 3, celle qui se trouve en bas du cylindre pour obtenir les coordonnées de notre caméra. Nous la sélectionnons et repérons les coordonnées qui s'affichent en bas de la fenêtre WED.

Dans notre exemple voici les valeurs que nous obtenons (notez les votres svp)



La précision n'est pas très importante car comme nous le verrons plus loin, notre caméra est dynamique.

Nous reportons ces valeurs dans notre module main à la suite.

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

Nous allons à présent préparer l'affichage des points de santé
Dans nos variables, nous saisissons les lignes suivantes :

```
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;
}
```

puis à la suite de la fonction main, après camera, nous entrons les lignes suivantes :

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

La première ligne permet d'avoir l'affichage des points de vie à 220 pixels (200 pixels de largeur d'image + 20 pixels) du bord droit quelle que soit la résolution de l'écran.

Création de nos entités 'combattants'

Pour une meilleure lisibilité nous avons appelé le joueur bleu : 'left' et le joueur rouge 'right'
A la suite de nos variables nous saisissons :

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


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

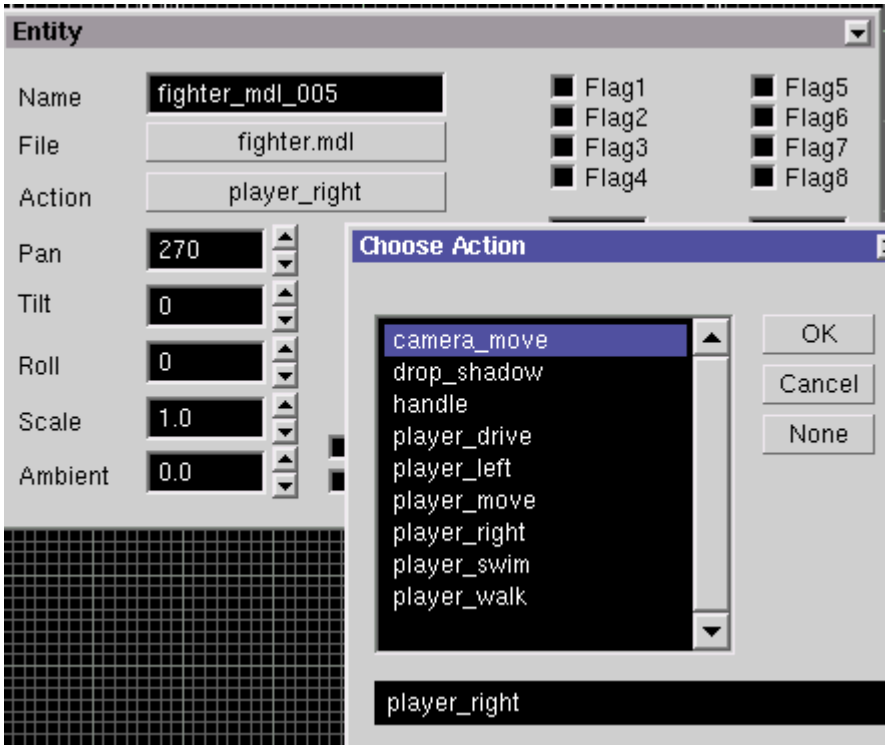
puis nous créons une action pour chacune des entités que nous plaçons après la fonction **`main()`**

```
action player_right
{

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


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

Il est grand temps de sauvegarder notre fichier et de revenir dans WED pour voir le fruit de notre

travail.

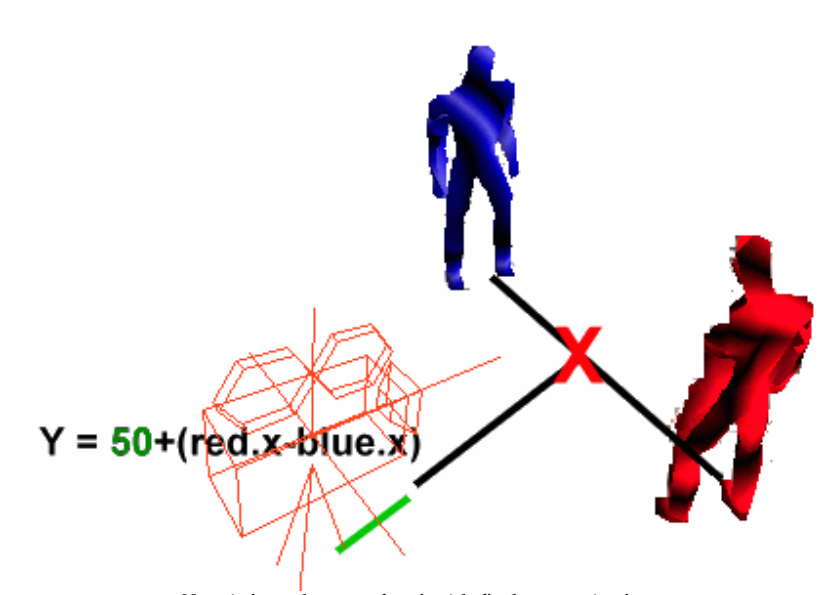
Nous sélectionnons le personnage de gauche puis click droit de la souris, propriétés et nous assignons **`player_left`** comme action.
Avec le joueur de droite nous assignons **`player_right`**.



Nous sauvegardons notre niveau, le compilons et l'exécutons. Si vous n'avez pas fait d'erreur vous devriez obtenir ceci :

Gestion de la camera

Comme nous le voyons sur cette image, dés qu'un des joueurs va reculer, il va sortir du champ de la caméra, nous allons donc écrire une petite fonction que nous appellerons à chaque déplacement d'un joueur pour repositionner la caméra.
En ce qui concerne la position X, nous prendrons comme référence le point milieu situé entre les deux joueurs.
Pour la position Y, nous prenons la distance entre les 2 joueurs + 50.

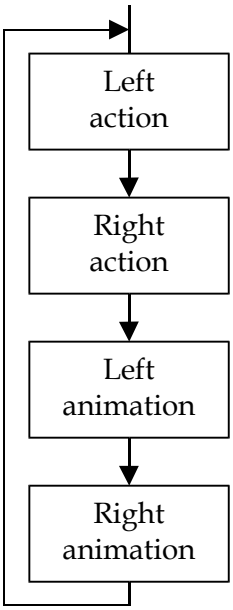


Nous écrivons donc cette fonction à la fin de notre scénario :

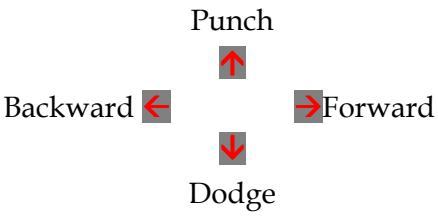
```
function pos_camera()
{
   camera.z = 90;
   camera.x = ((right.x+left.x)/2);
   camera.y = - ((right.x-left.x) + 50);
   wait (1);
}
```

Déroulement du jeu

Le jeu peut se décomposer en 4 phases principales :



L'action du joueur gauche se commande à partir des flèches de direction comme suit :



L'action du joueur droit nécessite de l'intelligence artificielle, qui dans notre cas sera succincte. Nous prévoyons 3 états possibles pour ce joueur :

- Prudent
- Défensif
- Offensif

Le passage de l'un à l'autre de ces états se faisant pour une part en fonction de notre comportement et pour une autre part en faisant intervenir un facteur aléatoire.
Nous écrivons donc les variables suivantes dans notre scénario :

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

Les états des joueurs

Ces différents contrôles vont nous permettre d'attribuer un état à chacun des joueurs. Comme nous l'avons vu au début en choisissant notre modèle, nous avons retenu 6 états qui sont :

Attente (waiting)	**Esquive (dodging)**	**Attaque (punching)**
Gagne (win)	**Reçoit un coup (pain)**	**Meurt (dead)**

Nous écrivons donc les variables suivantes dans notre scénario :

```
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;
```

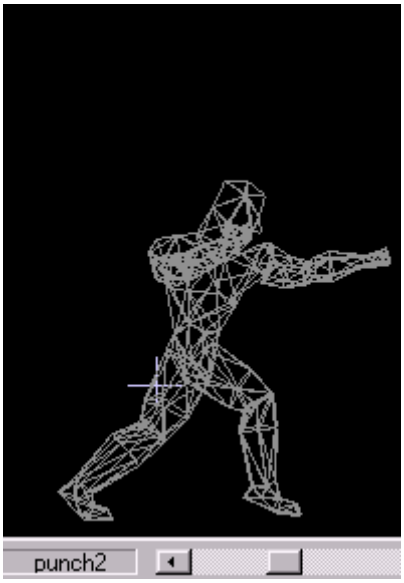
Nous allons bien entendu sonoriser certaines de ces actions nous ajoutons donc les variables sons suivantes :

```
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>;
```

L'animation

Pour l'animation, nous aurions pu utiliser les instructions **`ent_frame()`** et **`ent_cycle()`**, mais dans la mesure ou nous sommes appelés à intervenir à l'intérieur même d'une animation, nous utiliserons le paramètre **`frame`** propre à chaque entité. Je m'explique :
Par exemple lorsque le joueur rouge vous donne un punch, à la première frame vous pouvez encore esquiver, à la deuxième il est trop tard le poing est parti.

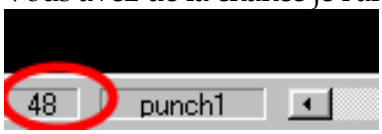
...Vous pouvez encore esquiver



....Il est trop tard pour esquiver

Ceci nous amène donc à repérer chacune des actions (début, fin) et éventuellement intermédiaire quand c'est nécessaire. Dans MED il suffit de relever les numéros de la frame dans la boite à outils animation.

Vous avez de la chance je l'ai fait pour vous mais ce sera à faire si vous utilisez d'autres modèles.

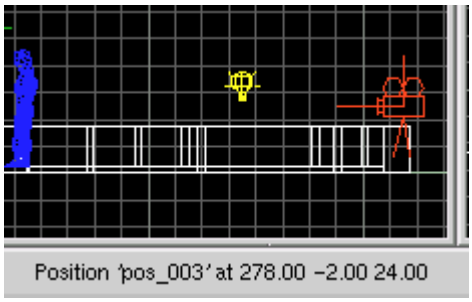


Nous saisissons donc les variables suivantes :

```
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;
```

La frame de fin correspond en fait à frame de fin +1

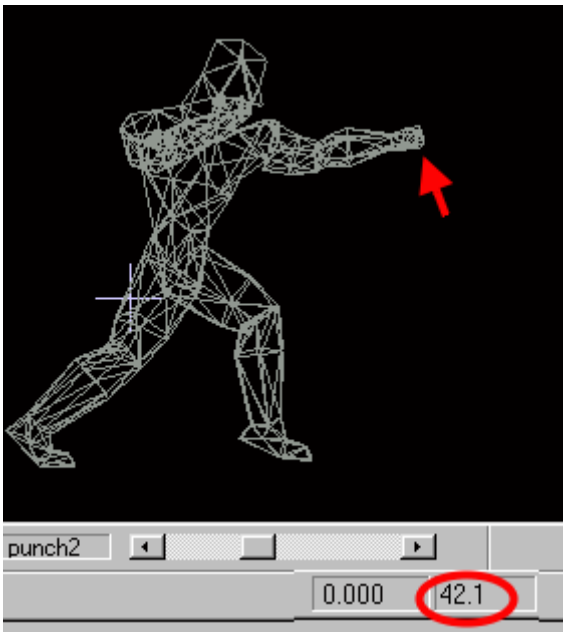
Il nous reste quelques variables à définir comme les bords extérieurs du terrain ou la distance entre joueur permettant de dire que le punch touche l'adversaire.
Pour les bords du terrain nous allons utiliser nos 2 autres positions dans WED (gauche et droite) et repérer les coordonnées X.



Nous notons 278 et créons les 2 variables suivantes :

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

Pour la distance minimale entre joueurs nous allons dans MED, nous nous positionnons sur **punch2** (bras déployé), nous plaçons la souris sur l'extrémité de la main et repérons la valeur X



Nous créons les 2 variables suivantes :

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

Et nous terminons avec nos variables en saisissant les 2 suivantes dont nous verrons l'utilité dans la suite des explications.

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

Nous sommes enfin prêts à nous battre.

Comme je sais combien c'est frustrant d'écrire des lignes sans pouvoir les tester, nous allons tout d'abord nous intéresser au joueur bleu et nous sauterons de l'action à l'animation alternativement.

Le jeu

Pour cela nous allons créer notre fonction **`game()`** et positionner nos 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);
   }
}
```

Ajoutons

```
game();
```

à la fin de notre fonction **main**

Et essayons. Si tout va bien, vous entendez le gong et les joueurs sont en position de combat.

Les déplacements du joueur gauche

Saisissez les lignes suivantes :

```
//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();
        }
     }
```

Et essayez. Avancez, reculez, vérifiez que vous ne sortez pas de la limite gauche sinon ajustez les valeurs **left_edge** et **right_edge**. Admirez également les mouvements de la caméra. Lorsque le joueur rouge vous tapera dessus vous n'aurez plus le temps de regarder tout cela.

C'est bien mais notre joueur avance et recule d'une drôle de façon. Remédions à cela. Saisissez les lignes suivantes **après** `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;
}
```

Vous l'avez bien compris, vous pouvez essayer à chaque étape, inutile de vous le dire, je vous sais aussi impatient que moi.

Donnons un peu de vie à notre adversaire : (à saisir après `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;
}
```

Défendons-nous !

A ajouter dans notre section : `left_action`, **après** les déplacements

```
//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;

   }
} if ((key_cud== 0) && ( left_state_attack_block == 2 )){left_state_attack_block = 0;}
```

et à ajouter dans notre section `left_anim`, **après** l'action waiting :

```
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;
      }
```

Nous allons nous préparer à donner des coups en saisissant ce qui suit mais nous devrons patienter car l'animation mérite quelques explications.

A saisir dans **left_action** à la suite :

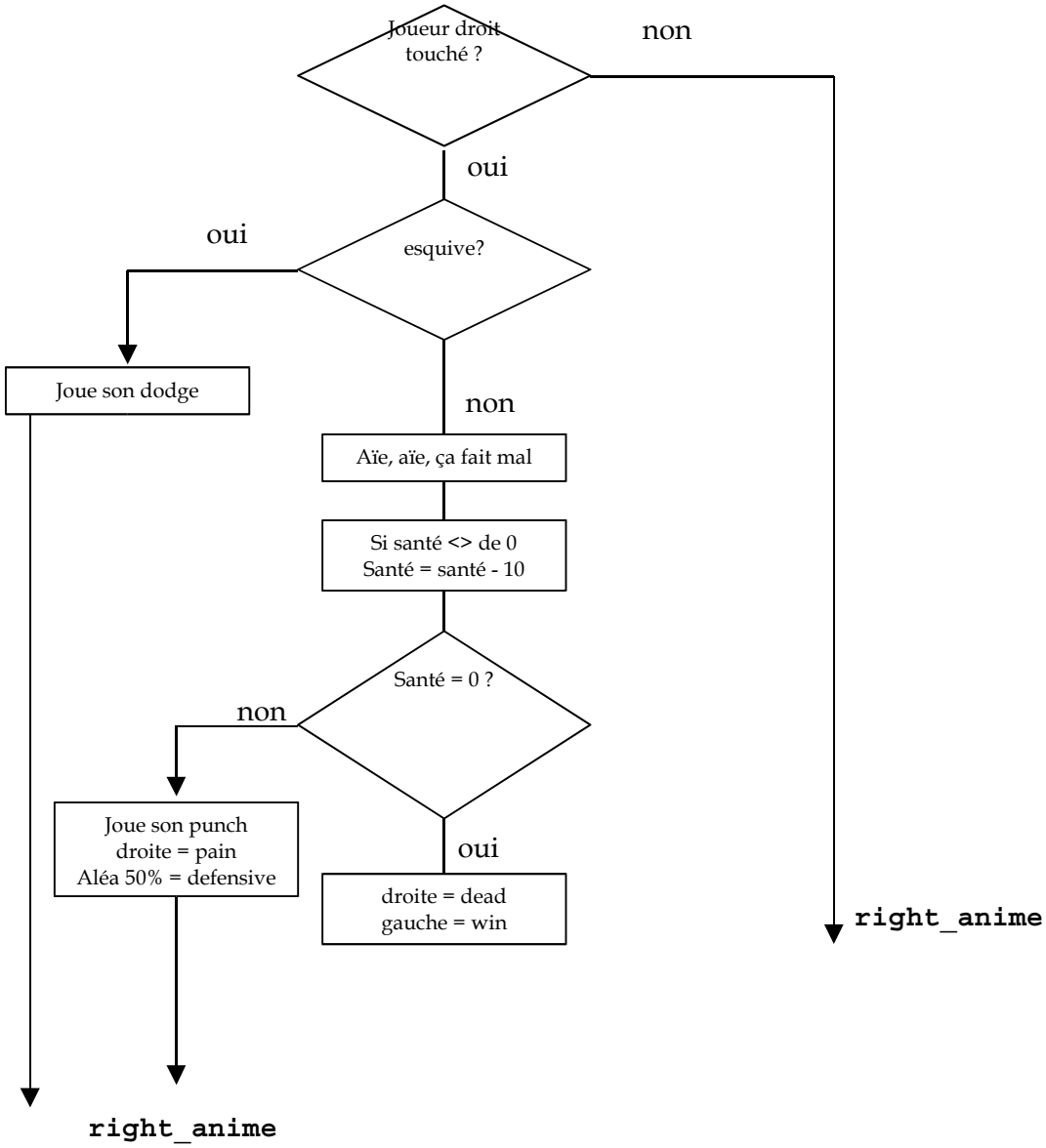
```
   //left attack
   if ((key_cuu == 1) && ( left_state_attack_block == 0 ))
   {

      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;}
```

Saisissons et commentons l'animation punching (à la suite dans **left_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;
```

Jusque là tout va bien, on punch et on revient en attente, si vous voulez vraiment essayer (vous êtes quand même de grands enfants ! Et profitez-en pour lui casser la figure, bientôt il pourra se défendre.) Fermez par 2 accolades et pensez à les enlever pour continuer.

Voici la suite :



Ce que nous écrivons comme ceci :

```
//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 win
            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;
}
```

Et je vous interdis d'aller lui taper dessus…

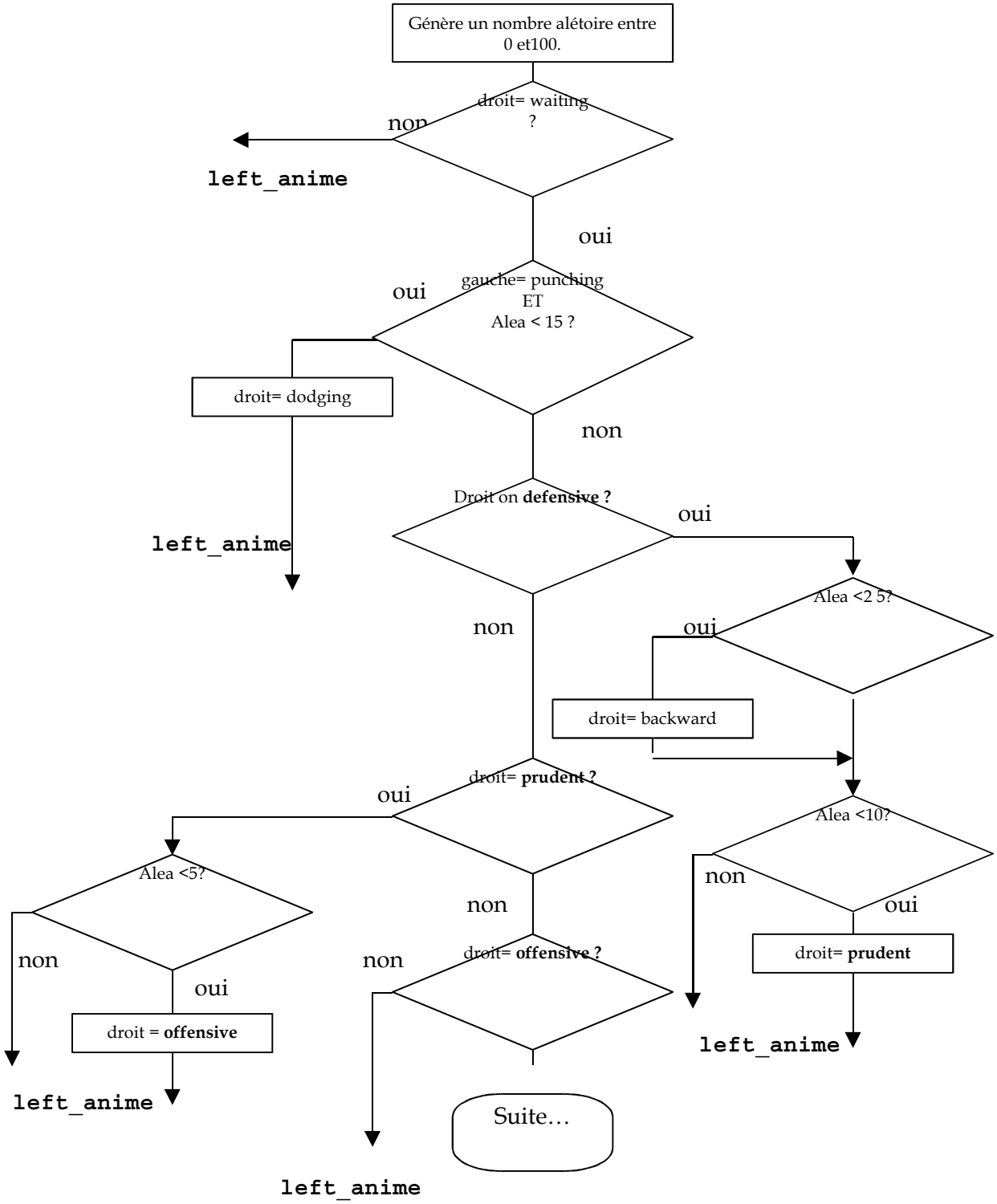
Notre joueur gauche peut aussi prendre des coups :

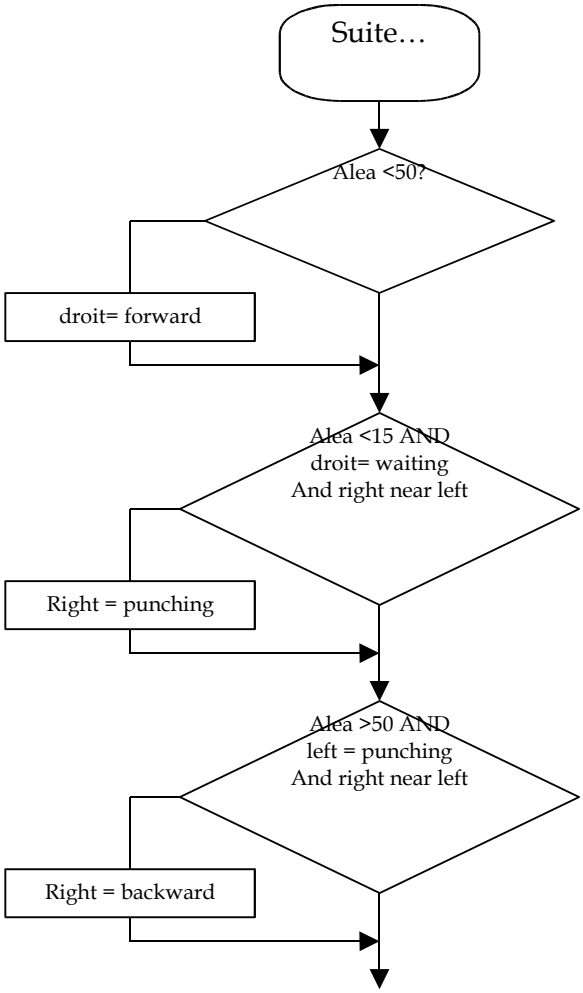
A saisir dans **`left_animation`** à la suite

```
   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;
   }
```

Nous en avons terminé avec notre joueur gauche. (Il reste à le faire mourir mais nous ne sommes pas pressé).

En ce qui concerne le joueur droit nous devons analyser son intelligence artificielle.





left_anime

Ou en langage WDL :

```
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 out right edge
         if (right.x < right_edge)
         {
            right.x += 1;
            pos_camera();
         }
      }


   }
```

L'animation du joueur rouge est à peu près identique à celle du joueur bleu, nous saisissons donc les lignes suivantes dans **`right anime`** **après** le 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;
}
```

Il ne nous reste plus qu'à gérer la mort et la victoire respectivement pour chaque joueur, ce qui nous donne :

Dans **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 win
      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;
   }
```

et dans **right_anime**:

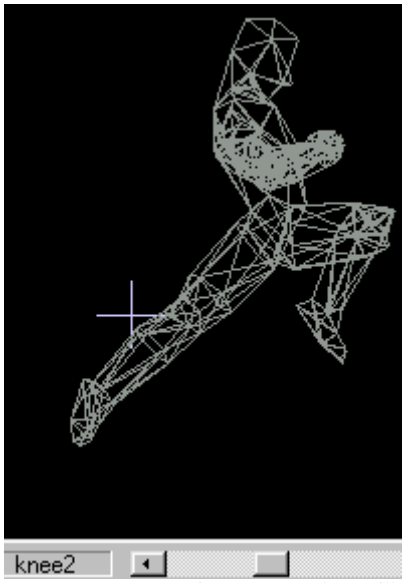
```
   if (right_action_state== dead)
   {
      right.frame += time;
      right.next_frame = 0;

      //son quand touche sol

      if (right.frame >= end_right_frame)
      {
         //left win
         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

J'ai envie de dire, ça y est nous en avons terminé, ce qui est vrai d'ailleurs, mais nous n'allons pas nous quitter comme ça.

Que peut-on ajouter ?

Tout d'abord si nous regardons bien notre combattant dans MED, nous voyons qu'il peut aussi donner des coups de genoux, je vous laisse inclure ceci si vous le souhaitez.



Après tout, ce n'est pas toujours au même de travailler.

Additional

Non moi ce que je souhaiterais faire avec vous c'est un peu de plaisir.
Comme par exemple des vues de caméras différentes, un son d'oiseau avec animation pour celui qui est à terre, un écran d'accueil avec possibilité de choisir la couleur de son joueur, une musique de fond, des spectateurs etc.

Les effets de caméra

En plus de la vue caméra que nous avons nous allons proposer 2 vues supplémentaires.
Une vue avec caméra au ¾ arrière du joueur bleu, une autre avec caméra sur l'épaule du joueur bleu et une troisième qui montrera notre tête à chaque fois que nous prenons un coup dans la figure. Ce qui nous fait donc 4 vues en tout.

Nous créons donc une variable

```
var cam_view = 0; //défaut
```

Puis à la fin de notre scénario nous écrivons :

```
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();
}
```

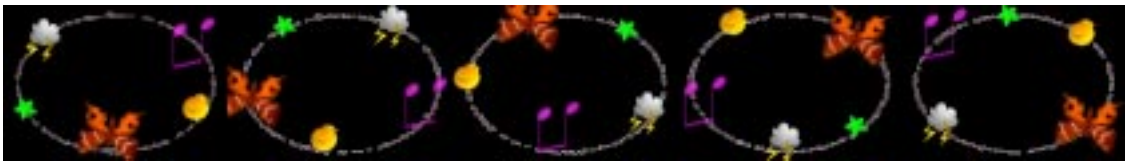
puis nous remplaçons notre fonction **`pos_camera`** par celle-ci

```
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;}

}
```

Un peu d'animation

Passons à présent aux petits oiseaux, cui-cui, j'ai créé une image bitmap KO+5.pcx que nous allons définir et afficher au moment opportun.



Nous incluons ces lignes au début du scénario avec les définitions de nos variables.

```
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;
}
```

Puis nous créons un pointeur pour notre son, c'est grâce à lui que l'on saura si le son est fini de jouer pour pouvoir passer à la suite.

Nous le mettons dans nos variables, juste après les définitions sound

```
var larkhandle = 0;
```

Nous créons ensuite la fonction **`cui_cui`** que nous plaçons à la fin.

```
function show_cui_cui()
{
   while (1)
   {
```

```
      cui_cui_pos.x += 84;
      if (cui_cui_pos.x >= 400){cui_cui_pos.x =0;}
      waitt (10);
   }
}
```

Puis nous modifions notre fonction **pos_camera** comme suit :
Les lignes en rouge sont les modifs, les lignes en noir existent déjà.

```
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;}
}
```

Et nous modifions notre mort en conséquence.

```
      if (right_action_state== dead)
      {

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


         //son quand touche sol

         if (right.frame >= end_right_frame)
         {
            //left win
            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)
            …
```

et

```
      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 win
      if (right.frame == start_right_frame){play_sound (right_win_sound,100);}
      right.frame += .5*time;
      right.next_frame = 0;
```

Un écran d'accueil

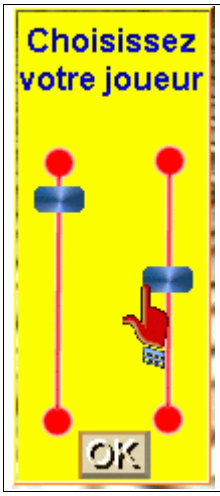
Cet écran va nous permettre de choisir la couleur des joueurs donc avant toute chose nous devons créer de nouvelles peaux pour notre personnage, allons dans MED. Exportons une de nos deux peaux, ouvrons notre logiciel de peinture favori et créons d'autres couleurs de peau.

J'ai pour ma part ajouté les peaux suivantes (dans cet ordre) :

```
rouge (d'origine) = 1
bleu (d'origine) = 2
gris =3
vert = 4
jaune = 5
bleu clair = 6
```

Si vous n'êtes pas équipé, j'ai fourni également ce personnage sous le nom de **`fighter1.mdl`**.

Nous allons à présent créer notre panneau qui nous permettra de choisir. Ce panneau ressemble à ceci :



Il est composé de 5 éléments :

1. Le fond du panneau (choix.pcx)
2. De 1 bouton curseur utilisé 2 fois(cursor.pcx)
3. De 1 bouton 'OK' on (button_ok_on.pcx)
4. De 1 bouton 'OK' off (button_ok_off.pcx)
5. D'un pointeur de souris (pointeur.pcx)

Nous les définissons comme suit :

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

Nous définissons également les 2 variables qui seront alimentées par **vslider** :

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

Nous définissons à présent notre panneau :

```
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;
}
```

D'après cette définition vous pouvez noter que :

1 – Lorsque nous appuyons sur le bouton OK, nous appelons la fonction **game()**
2 – le slider de gauche navigue entre les valeurs 1 à 3 (les 3 peaux possibles pour notre joueur de gauche.
3 – le slider de droite navigue entre les valeurs 4 à 6 (les 3 peaux possibles pour notre joueur de droite.

Il ne nous reste plus qu'à faire fonctionner tout cela :

Dans la fonction main() nous remplaçons **game();** par

```
choix();
```

Puis nous écrivons notre 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);
   }
}
```

Les 2 premières lignes sont faites pour que notre panneau soit toujours centré quelle que soit la résolution de l'écran de jeu. Nous faisons ensuite apparaître le panneau puis la souris et mettons la couleur de peau au joueur gauche et droite en fonction de la valeur retournée par les 2 sliders.

La ligne random est une vieille astuce que j'utilisais lorsque les ordinateurs n'avaient pas d'horloge

interne. En effet vous l'avez peut-être constaté, dans un programme linéaire, les instructions random donnent toujours les mêmes suites de valeurs à chaque exécution du programme. Dans notre cas comme nous ne passerons jamais deux fois le même temps dans notre boucle, temps qui dépend du moment on va cliquer sur ok, nous sommes donc assurés d'avoir des nombres aléatoires vraiment aléatoires.

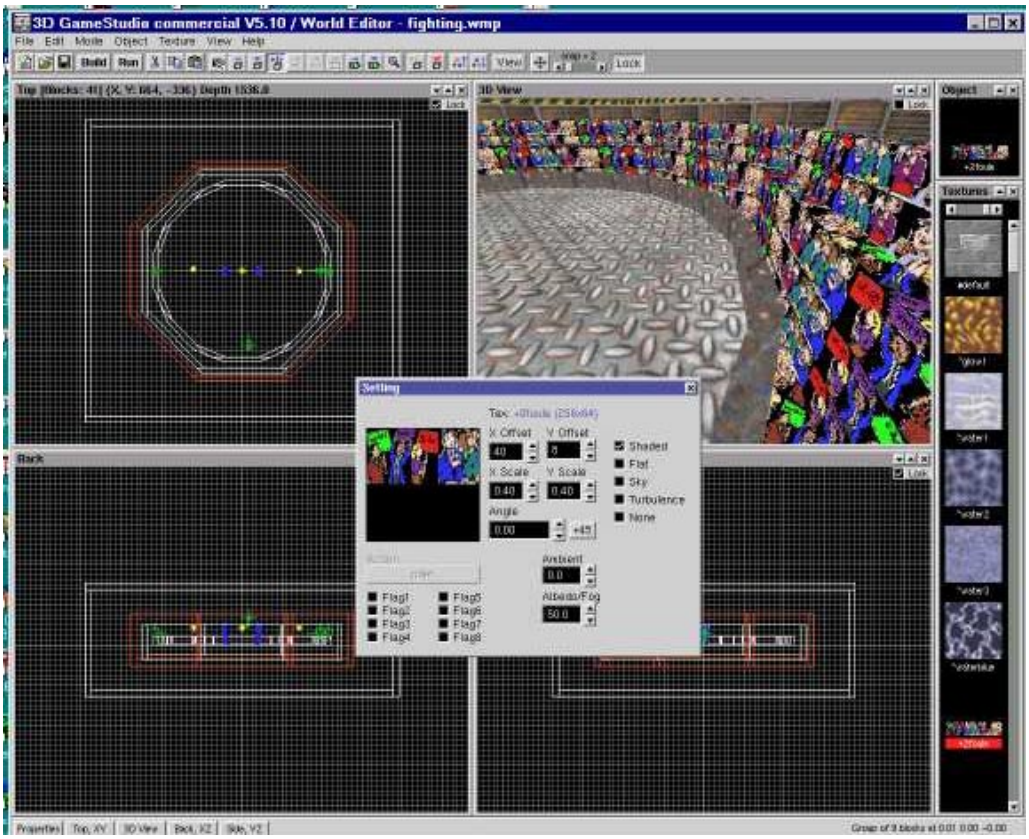
Nous ajoutons ensuite ces 2 lignes au début de notre fonction **`game()`** et nous sommes prêts à jouer avec la couleur de notre choix.

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

Je vous laisse comme exercice l'adaptation de la couleur de la barre de santé comme celle du joueur. (Pour l'instant nous gardons bleu et rouge).

La foule

Pour ce qui est de la foule, je ne vais pas entrer dans les détails, vous ferez au moins aussi bien que moi. J'ai ajouté 2 cylindres creux pour faire les estrades et j'ai appliqué une texture animée foule. (vous trouverez dans votre répertoire les fichiers **`+0foule.pcx`** à **`+3foule.pcx`**)



Le résultat est somme toute agréable.

La musique de fond

La musique de fond ne pose pas beaucoup plus de problèmes.

Dans **`main()`** après **`load_status`**, entrez les lignes suivantes :

```
   start_song();

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);
   }
}
```

Conclusion (2)

Cette fois-ci je pense que nous en avons terminé. Il ne nous reste plus qu'à nous séparer…

Quoique, peut-être vous êtes-vous dit que ce serait sympa si, en fonction de sa couleur, chaque personnage avait sa propre personnalité et donc des réactions différentes. Ca n'est pas bête comme idée. Je vous laisse faire, mais non je plaisante, encore un petit coup de main pour vous aider à grandir dans le monde merveilleux de WED, d'accord mais après ce sera à vous.

En ce qui concerne notre joueur nous allons faire simple :

- **Premier joueur :**
- le faible c'est à dire que chaque coup porté ne fait perdre que 5 points de vie à l'adversaire et chaque coup pris lui fait perdre 5 points de plus que ce que donne l'adversaire. Mais comme il faut lui laisser une chance de gagner, on va lui donner une botte secrète, le coup de genou, imparable et qui terrasse l'adversaire. (coup de genou qui remplacera le punch de façon aléatoire, 1 chance sur 10 par exemple)
- **Deuxième joueur**:
- celui que nous connaissons actuellement, donc nous ne changerons rien.
- **Troisième joueur:**
- la brute, chaque coup porté fait perdre 15 points de vie, chaque coup reçu lui fait perdre 5 de moins que ce que donne l'adversaire. Il peut, lui, esquiver le coup de genou, s'il recule assez vite.

Ce qui nous donne :

Coup donné par	Coup reçu par	Bonus/malus	Points de vie à enlever
Faible = 5 points	Faible	+5	10
	Normal	0	5
	Fort	-4	1
Normal = 10 points	Faible	+5	15
	Normal	0	10
	Fort	-4	6
Fort = 15 points	Faible	+5	20
	Normal	0	15
	Fort	-4	11

Les coups donnés par joueur gauche : **`left.skin*5`** soit 5 pour faible, 10 pour normal et 15 pour fort
Les coups donnés par joueur droit : **`(right.skin-3)*5`** soit 5 pour faible, 10 pour normal et 15 pour fort.
Nous créons deux variables :

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

Puis nous calculons leur valeur : (nous incluons ces lignes au début de notre fonction **`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;}
```

Pour les points de vie, c'est très simple nous remplaçons dans les lignes suivantes le bleu par le rouge :

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

   if (right_health == 0)
```

par :

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

        if (right_health <= 0)
```

et

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

        if (left_health == 0)
```

par

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

        if (left_health <= 0)
```

et

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

par

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

Les coups de genoux :

Le joueur de droite peut nous donner des coups de genoux, plutôt que de faire une action supplémentaire, je modifie l'action punch pour faire qu'une fois sur dix le punch soit remplacé par un knee.
Nous créons tout d'abord les variables suivantes, à mettre à la suite des variables déjà existantes

```
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 ;
```

puis nous ajoutons les lignes en rouge :

```
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)
                       {
```

Pour le joueur de gauche nous pouvons opter pour le même système, valeur aléatoire substituant le coup de genou au punch out ajouter une autre touche. J'ai opté pour la deuxième solution et vous pourrez donner des coups de genoux en utilisant la touche **CTRL**.

Nous saisissons les lignes suivantes juste avant **`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
---------- - - -  - -- - - --- -
```

Puis modifier ces lignes

```
            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)

Cette fois-ci je pense que nous en avons terminé. Il ne nous reste plus qu'à nous séparer…

Ce qu'il vous reste à faire après que vous ayez bien joué :

- Instaurer un système de nom avec affichage des scores pour vous mesurer à vos amis ou à votre famille
-
- Visualiser les points d'impacts en utilisant les particules
-
- Varier la vitesse d'animation en fonction de la force du joueur, un joueur faible ayant des gestes plus lents qu'un joueur fort.
-
- Dans le même esprit, faire que le son soit proportionnel à la force
-
- Etc.

PS : vous trouverz le fichier complet de scénario sous le nom de fighting.al1 qu'il vous suffit soit de renommer en fighter.wdl soit de lire avec notepad et de faire du copier coller.