3D GameStudio

Workshop Kampfspiele

für A5 Engine 5.10
von Alain Brégeon August 2001

Aktuelle Neuigkeiten, Demos, Updates, diverses Werkzeug sowie das Anwender-Forum und alles zum jährlichen Wettbewerb findet sich auf den GameStudio Internetseiten unter **http://www.3dgamestudio.com**.

Inhalt

Vorwort

Lieber Leser,

Indem ich diesen Workshop erstellt habe, wollte ich Ihnen helfen, die Frage "Wie mache ich ein Kampfspiel mit 3D GameStudio?" zu beantworten. Die dabei verwendeten Features sind ab der Version **4.25** oder neuer verfügbar.

Wie andere Kurse zuvor zielt auch dieser Workshop auf Anwender ab, die über eine gewisse Vorerfahrung mit 3D GameStudio verfügen. Ich setze vorraus, dass Sie die Tutorials durchgearbeitet haben und mit den diversen Werkzeugen (WED,MED and WDL) umzugehen wissen.

Dieser Text ergänzt die zu 3D GameStudio gehörige Dokumentation und kann diese nicht ersetzen. Sollte Ihnen irgendetwas in diesem Workshop unklar sein, lesen Sie daher bitte im mitgelieferten Manual nach. Für eventuelle Unklarheiten im Ausdruck, fehlerhaften Code, Irrtümer oder Versäumnisse entschuldige ich mich hiermit bereits im Voraus.

Ich hoffe, dieser Workshop wird Ihnen neue Erkenntnisse bringen und ausserdem Spass machen.

Alain Brégeon
mailto:alainbregeon@hotmail.com

Die Kampfphilosophie, die diesem Tutorial zugrunde liegt, wurde stark von dem dem Demo C ++ SDK von Microsoft © beiliegenden 'Rokem' beeinflusst.

Besorgen Sie sich die neueste Version

Bevor Sie loslegen vergewissern Sie sich bitte, dass Sie auch die neueste Version von 3D GameStudio (**4.25** oder neuer) haben.

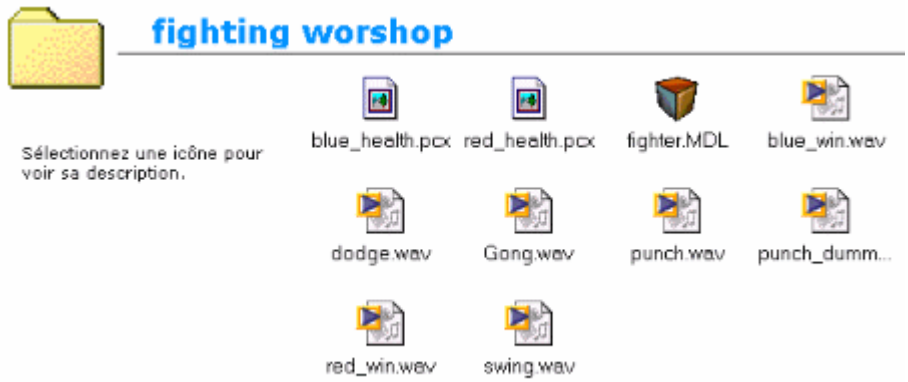
Bereiten Sie Ihre Arbeitsumgebung vor

Erstellen Sie einen Ordner namens "fighting Workshop" in Ihrem GStudio-Verzeichnis. Das ist nun der Ordner in dem Sie sämtliche Spielelemente verwalten werden.

Als erstes werden wir in diesen Ordner die Entities, die wir für unser Spiel brauchen hineinkopieren. Sollten Sie diese noch nicht haben, finden Sie sie auf der Download-Seite von Conitec unter http://www.conitec.net / a4update.htm.

Entzippen Sie den Inhalt in Ihren Ordner, er sollte dann die folgenden Dateien enthalten:

```
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
```



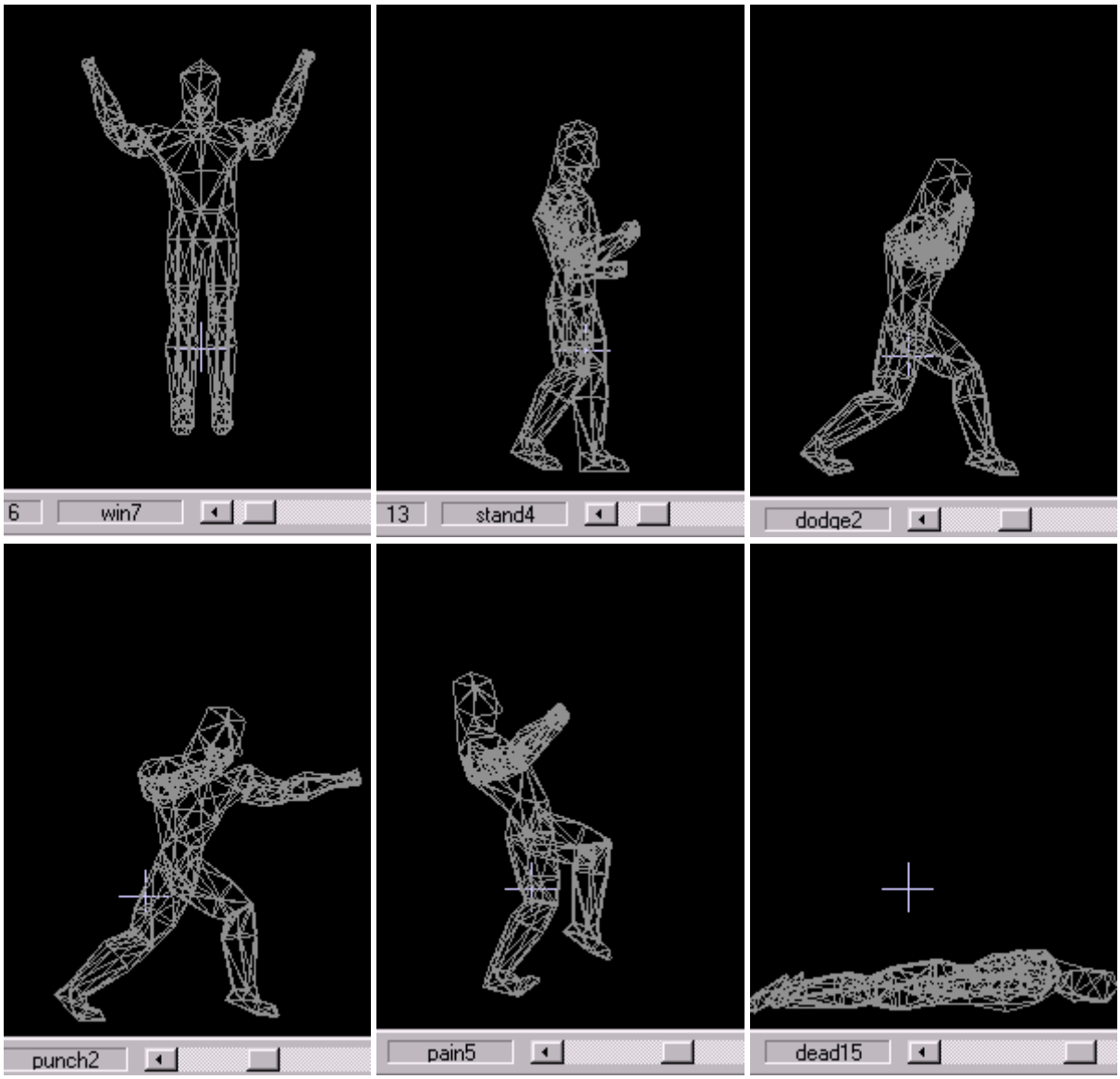
Fighting Workshop-Ordnerr

Unser Kämpfer

Es ist allgemein bekannt, dass Kämpfe dadurch vereitelt wurden, dass es keine Kämpfer gab. Wenn wir also ein Kampfspiel erstellen wollen, brauchen wir zuallererst mal Kämpfer.

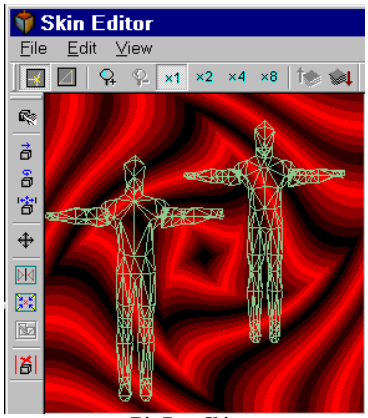
Unsere Figur sollte zu folgendem in der Lage sein: Ausweichen, Schlagen, Hiebe einstecken, Warten, Sterben und Gewinnen. Die diesem Tutorial beigefügte Figur kann das alles. was die 'Skin' betrifft, habe ich mich für ein leichteres Verständnis des Tutorials an das simpelste gehalten: eine rote und eine blaue 'Skin'.

Öffnen Sie den MED, laden Sie **`fighter.mdl`** und betrachten Sie sich all die Animationen, zu denen dieser Kämpfer fähig ist.



Die Fighter.mdl

Natürlich könnte er noch einiges mehr aber da wir diesen Workshop ja nicht überfrachten wollen, sollten wir uns mit diesen 6 Animationen zufriedengeben.



Die Rote Skin

Leveldesign für's Kampfspiel

Wir sehen uns genötigt zu wählen: ein Boxring, eine Strasse, der Mond... Die einzige Bedingung, die wir zu erfüllen haben, ist es genug Platz für die Kamera mit einzubeziehen. Hier meine persönliche Wahl:

Öffnen Sie WED und gehen auf **File-> New**

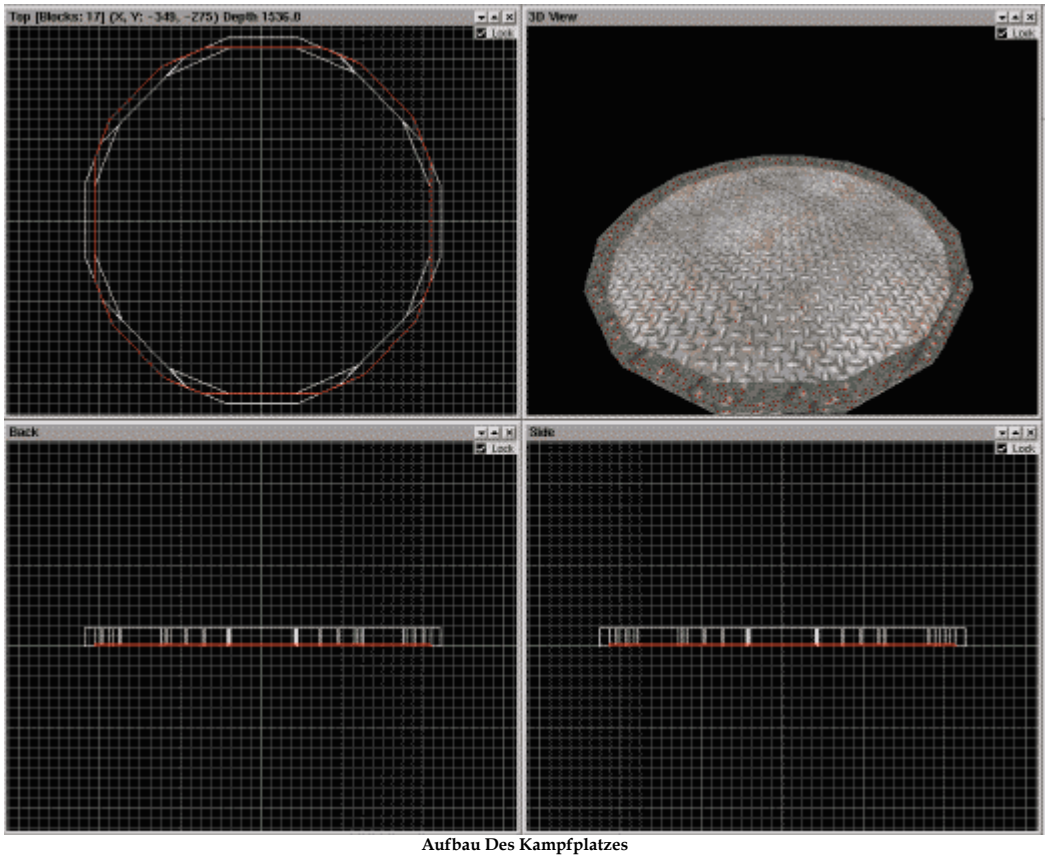
Öffnen Sie den Textur-Manager (**Texture -> Texture Manager**). Wählen Sie **add WAD**, und fügen die **`standard.wad`** ein. Schliessen Sie den Textur-Manager wieder.

Wählen Sie dann **Add Primitive - > Cylinder -> 16**.

Vergrössern Sie den Kreis soweit, bis Sie etwa 5 grosse Quadrate im Durchmesser haben.

Höhlen Sie den Block mit **Alt+H** aus , womit Sie seine Höhe auf 2 kleine Quadrate reduzieren. Gehen Sie nun eine Ebene tiefer (**scope down**), selektieren Sie die Deckenwand Ihrer Box und löschen Sie diese.

An Texturen habe ich hierfür Standardtexturen genommen: Für die Seiten **`metalrivet2`** und für dem Boden habe ich mich für **`metalribbed5`** entschieden.

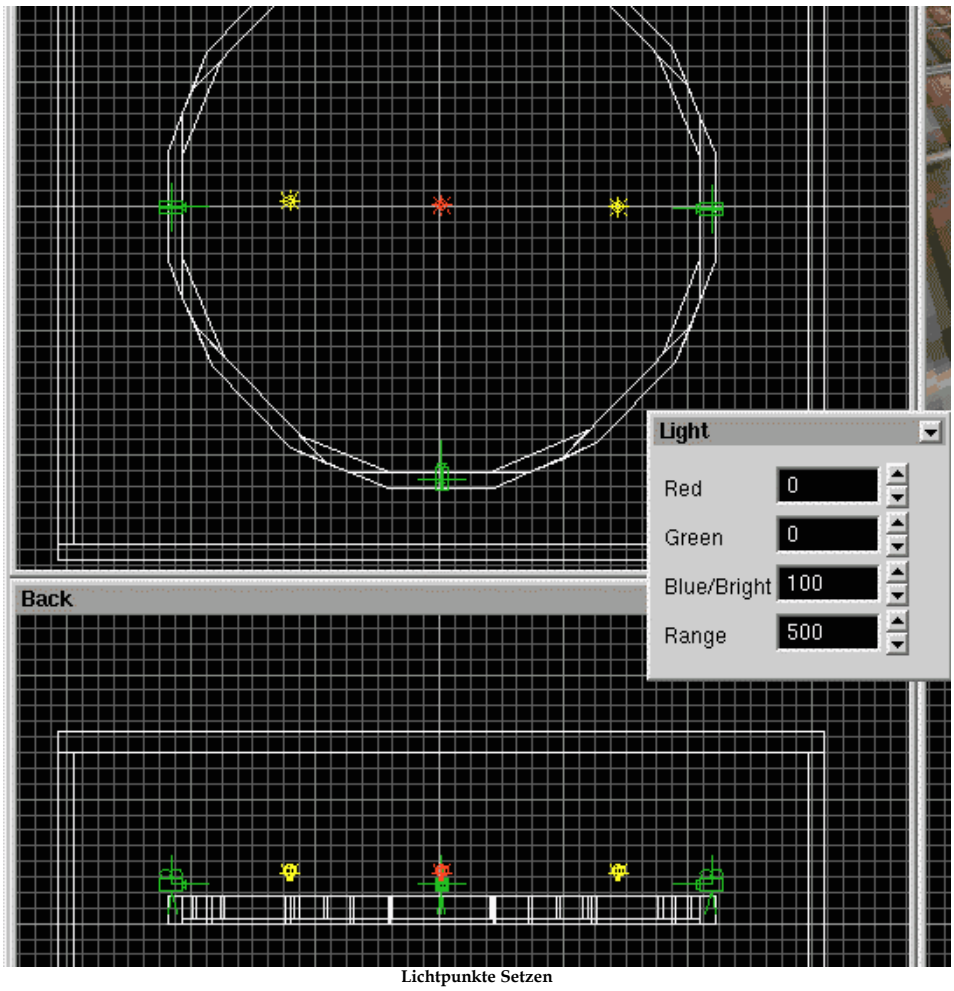


Aufbau Des Kampfplatzes

Nun erstellen wir noch einen 'Sky', den wir zwar nicht unbedingt brauchen aber um den guten Ton zu wahren tun wir es eben doch.

Wählen Sie **Add Primitive -> Cube (large)** und halten Sie die Seiten Ihres Würfels vom Zylinder fern. Höhlen Sie den Block mit **Alt+H** aus und bringen Sie eine Textur an (z. B. **`gate1`**).

Nun fügen wir noch 2 oder 3 Leuchtquellen (**Add Light**) hinzu. Dabei müssen wir ziemlich acht geben: Das erste Licht setzen wir in die linke Ecke unseres Rings, das Zweite in die rechte Ecke und das Dritte kommt in die Mitte.



Lichtpunkte Setzen

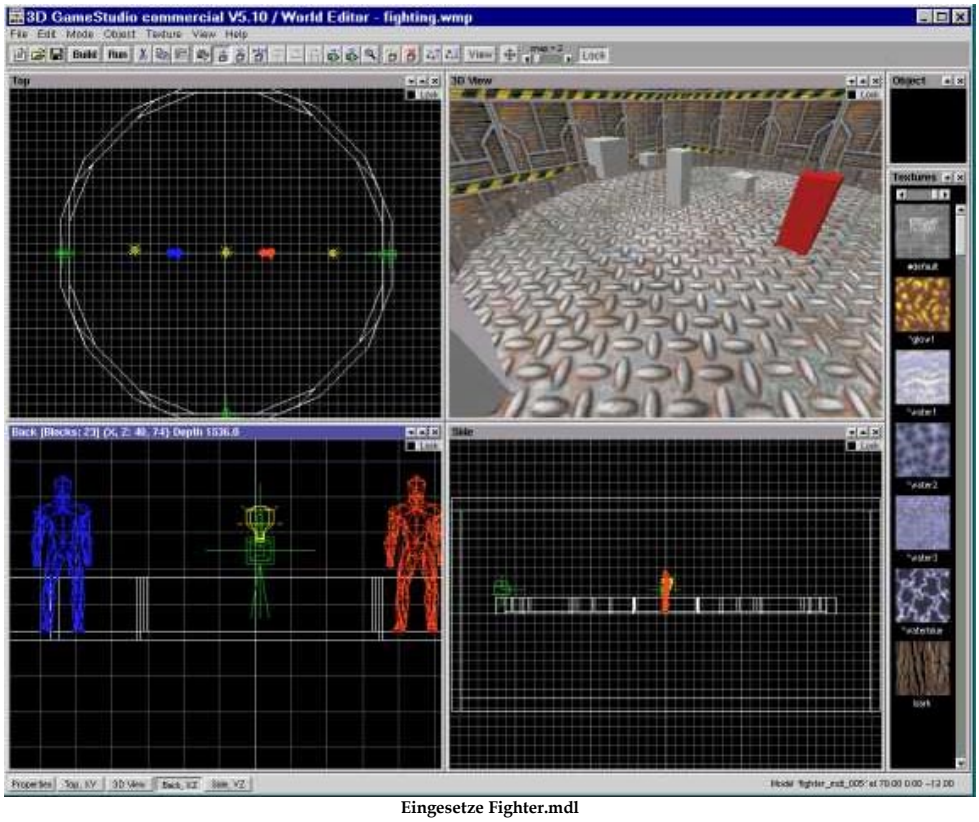
Ich merke schon, Sie sind ungeduldig und wollen endlich betrachten, wie Ihr Level aussieht. Also gut, speichern Sie es unter dem Namen **`fighting`**, **build**en Sie es mit markiertem **fly-thru = enable.**

Hier das Ergebnis: Ist es nicht hübsch?

Die Beleuchtete Arena

Kämpfer hinzunehmen

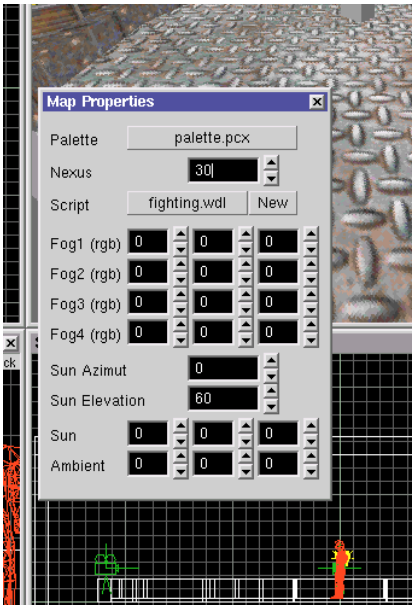
Alles was wir noch tun müssen, ist es, zwei Kämpfer zu plazieren und wir wären soweit fertig. Wählen Sie also **Object-> Load Entity** und suchen Sie nach **`fighter.mdl`**.Öffnen Sie es und das Model erscheint in Ihrer Welt. Setzen Sie es ins Zentrum auf den Boden.



Eingesetze Fighter.mdl

Erstellen Sie Ihr Skript

Erstellen Sie nun ein Skript für Ihr Level. Dazu öffnen Sie einfach Ihr Eigenschaftsfenster (**File > MapProperties**) und klicken auf den **new-**Knopf. Der Knopf daneben müsste nun von **`ndef`** auf **`fighting.wdl`** umspringen.



Neues Skript Im Eigenschaftsfenster

Öffnen Sie Ihren Levelordner, wählen und öffnen (Doppelklick) Sie die Datei **`fighting.wdl`**. Falls Windows fragt mit welcher Applikationssoftware es die Datei öffnen soll, suchen Sie sich am besten "Notepad" (oder eben einen anderen reinen Texteditoren) aus.

Sie werden feststellen, dass die Standard Spiele-"Templates" bereits für sie erstellt wurden. Das ist für die meisten Projekt sehr schön aber wir wollen ja etwas für Fortgeschrittene machen. Darum nur zu, markieren Sie alles (in Microsoft Notepad use **Edit->Select All**) und drücken Sie die Löschtaste. So, jetzt fangen Sie bei Null an!

Pfade einfügen

Lassen Sie uns mit dem Definieren der Pfade beginnen, die unser Programm benutzen wird. Diese Pfade werden dazu gebraucht, der Engine zu sagen, wo sie die in unserem Projekt verwendeten Dateien finden kann (Bilder, Sounds, andere Skripte usw.). Der Ordner in dem wir uns befinden("Fighting Workshop") ist bereits eingefügt, wir müssen daher in unserem Fall nur noch die **`template`**-Ordner hinzufügen.

Schreiben Sie die folgende Zeile:

```
path  "..\\template";    // Pfad zum WDL Templates-Unterverzeichnis
```

Beachten Sie, dass sämtliche Pfade in Relation zu unserem Levelordner stehen. Die obige Zeile besagt folgendes: "Gehe ein Verzeichnis höher ("...") und dort in den Template-Ordner (**`\\template`**)".

'Includes' einfügen

Nachdem wir unsere Pfade eingerichtet haben, sollten wir unsere sogenannten Include-Dateien hinzunehmen. Schreiben Sie also unter **path**:

```
include <movement.wdl>;  // Bibliothek von WDL-Funktionen
include <messages.wdl>;
include <menu.wdl>;      // menu muss VOR doors and weaponsincludet werden
include <particle.wdl>; // wegnehmen, wenn keine Partikel gebraucht werden
```

Der **include**-Befehl sagt der Engine, dass sie diese Zeile durch den Inhalt zwischen den beiden spitzen Klammern (<...>) ersetzen soll. Es ist dasselbe, als wenn Sie in die betreffende Datei gehen, den gesamten Code kopieren und an der Stelle in Ihrem Skript einfügen würden.

Dies ist ein sehr wirksames Werkzeug, denn es erlaubt uns Code aus anderen Projekten wieder zu verwenden und die Vorteile von Überarbeitungen innerhalb der **include**-Dateien zu nutzen ohne den eigenen Code umschreiben zu müssen. Die Version 4.19 z.B. beinhaltete Code, der das Schwimmen in Wasser ermöglichte. Seither kann in jedem Projekt, das die **movement.wdl** 'included'' hat dieser neue Schwimmcode verwendet werden.

Da einige Skripte Werte verwenden, die in anderen Skripten definiert sind, ist die Reihenfolge der **include**-Zeilen genauso wichtig wie bei den Pfaden. Die **actors.wdl** z.B. verwendet die Variable **force**, die aber in der **movement.wdl** definiert ist. Setzen wir also die **actors.wdl** vor die **movement.wdl**, kriegen wir hässliche Fehler.

Es ist vollkommen in Ordnung Skripte auch dann zu '**include**en', wenn Sie daraus gar keine Features in Ihrem Code verwenden. Die meisten vorgefertigten Dateien (Templates) sind voneinander abhängig, so dass Sie, sollten Sie vorhaben eine zu verwenden, zur Sicherheit am besten gleich alle per **include** in Ihren Code aufnehmen sollten. Die Ausnahme zu dieser Regel heisst **venture.wdl**. Auf dieses Skript wird von keinem anderen verwiesen, es selbst allerdings benutzt einige der anderen.

Startwerte für die Engine

Nun werden wir einige zur Bestimmung der Simulationsdarstellung wichtige Werte festlegen. Diese Werte betreffen die Auflösung, Farbtiefe, Framerate und Helligkeit. Fügen Sie also die folgenden Zeilen unterhalb der **Include**-Zeilen ein:

```
// Startwerte der Engine
ifdef lores;
var video_mode = 4; // 320x240
ifelse;
var video_mode = 6; // 640x480
endif;
var video_depth = 16; // D3D, 16 bit Aufloesung
var fps_max = 50; // 50 fps max
```

Unsere Spielevarianten

Hier ist der Platz an dem wir unsere Spielevarianten je nach Notwendigkeit eingeben:

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

Anzeigen des A4/A5-Logos

Zum Anzeigen als Logo nehmen wir das aus dem Templates-Verzeichnis:

```
////////////////////////////////////////////////////////////
// definiere einen Splash-Screen mit dem gewünschten A4/A5-Logo
bmap splashmap = <logodark.bmp>; // das default A5 logo in templates
panel splashscreen { bmap = splashmap; flags = refresh,d3d; }
```

Die Haupt ('Main')-Function

Für jedes Projekt brauchen Sie eine Haupt- oder **main**-Funktion. Das ist die allererste Funktion, die beim Programmstart aufgerufen wird. In den meisten Fällen ist die **main**-Funktion ziemlich einfach und auch unsere **main** ist da keine Ausnahme. Schreiben Sie also bitte die folgenden Zeilen hinter Ihre bisher letzten:

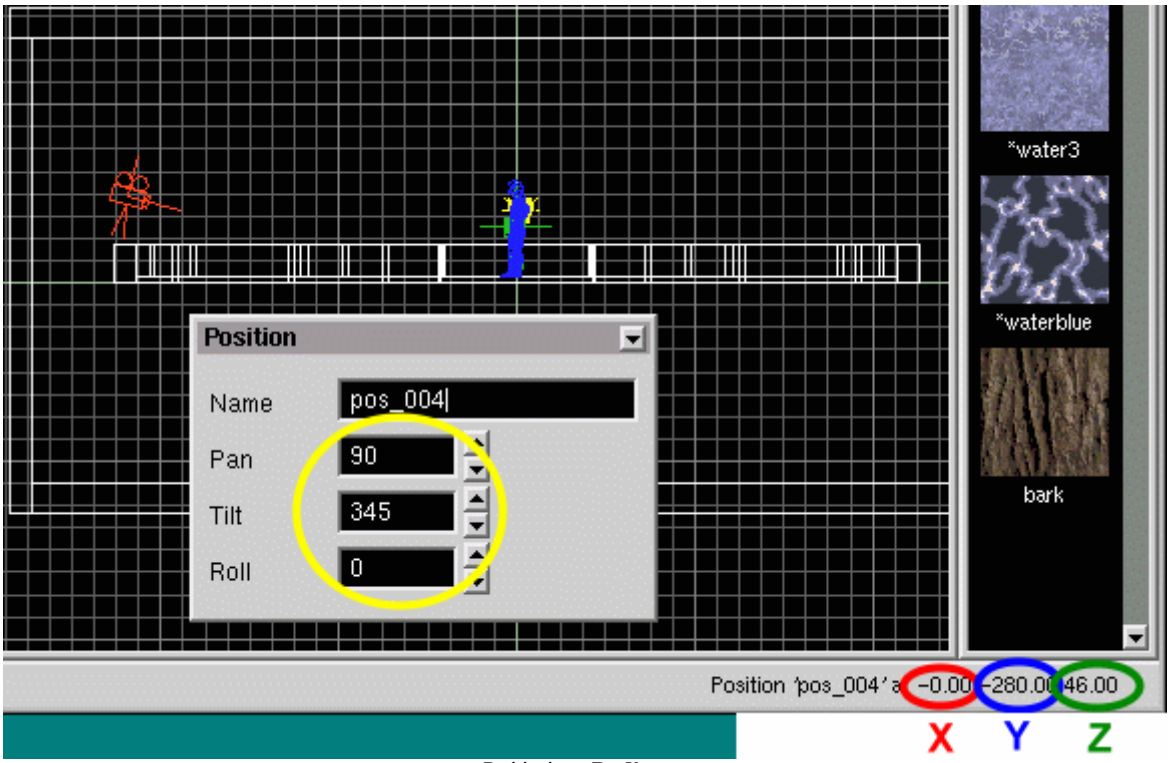
```
function main()
{
   fps_max = 50;
   warn_level = 2;    // melde schlechte Texturgroessen und fehlerhaften wdl code
   tex_share = on;    // map entities teilen ihre Texturen

// nentriere den Splash-screen für nicht-640x480 Auflösungen
   splashscreen.pos_x = (screen_size.x - bmap_width(splashmap))/2;
   splashscreen.pos_y = (screen_size.y - bmap_height(splashmap))/2;
   splashscreen.visible = on; // mache ihn sichtbar
   wait(3); // warte 3 Frames (für dreifaches Buffering) bis gerendert und zum Vordergrund geflippt ist.
// jetzt lade den Level
   load_level (<fighting.wmb>);
// warte die benötigte Sekunde und schalte dann den Splash-Screen aus.
   waitt(16);
   splashscreen.visible = off;
   bmap_purge(splashmap);   // Nimm die Logo-Bitmap aus dem Videospeicher
   load_status(); // lade einige globale Variablen, wie Soundvolumen
```

Jede dieser Zeilen ist mit einem Kommentar versehen, so dass sie keiner weiteren Erklärungen bedürfen.

Installierung der Kameraposition

Um die passenden Koordinaten für unsere Kamera zu bekommen, verwenden wir unsere 3. Lichtposition. Selektieren Sie die Kamera und ziehen Sie sie auf die ensprechenden Koordinaten (Sie finden Sie unten am WED-Fenster angezeigt).

Hier die Werte, die wir in unserem Beispiel erhalten (bitte beachten Sie die Ihren):



Positionieren Der Kamera

Da unsere Kamera, wie wir später noch sehen werden, dynamisch ist, ist Präzisionsarbeit hier nicht so arg wichtig.

Als nächstes übernehmen wir diese Werte und tragen sie in unser Hauptmodul ein:

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

Nun ist es an der Zeit, die Gesundheits-Variablen zu implementieren.

Tippen Sie bitte die folgenden Zeilen:

```
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;
}
```

dann, unter der **main**-Funktion, geben wir diese Zeilen nach der Kamera ein:

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

Die erste Zeile ermöglicht das Anzeigen der Gesundheitspunkte innerhalb von 220 Pixeln vom rechten Bildschirmrand aus(200 Pixel beträgt die Bitmapbreite + 20 Pixel).

Machen Sie Kämpfer aus den Entities!

Zur Vereinfachung der Lesbarkeit nennen wir den blauen Spieler “Left” und den roten “Right”. Also tippen wir unterhalb unserer Variablen:

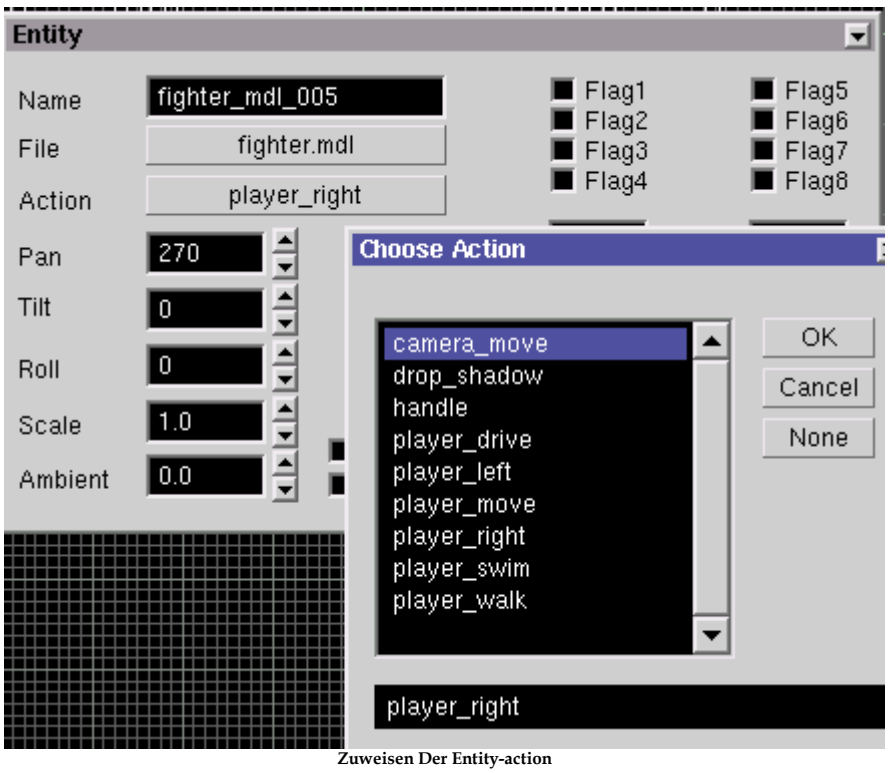
```
synonym left {type entity;}
synonym right {type entity;}

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

Nun erstellen wir für jede der Entities je eine Action, die wir hinter die Function **main()** plazieren.

```
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;}
}
```

Nun ist es höchste Zeit unsere Datei zu speichern, in WED zurückzukehren und uns die Früchte unserer Arbeit zu betrachten.

Um die betreffende Action festzulegen, selektieren wir die 'Left'-Figur und öffnen mit einem Rechtsklick das Eigenschaftsfenster. Hier weisen wir **`player_left`** als Action zu. Machen Sie dann dasselbe mit **`player_right`**.



Zuweisen Der Entity-action

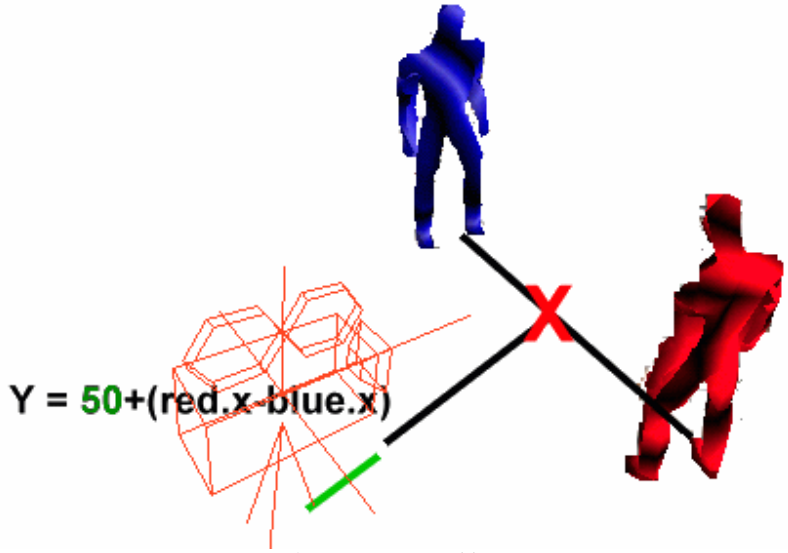
Speichern Sie das Level jetzt, **build**en und starten Sie es. Wenn Sie es gut gemacht haben, sollte es in etwa so aussehen:

Zwei Kämpfer In Der Arena

Steuerung der Kamera

Wie wir auf dem Bild sehen, würde jeden der beiden Spieler ein Schritt zurück ausserhalb der Kamerasicht manövrieren. Darum werden wir nun eine kleine Funktion schreiben, die wir innerhalb einer jeden Playerbewegung zum Zwecke einer Repositionierung der Kamera aufrufen werden.

Als X-Position nehmen wir den Mittelpunkt zwischen den beiden Figuren. Unser Y-Punkt errechnet sich aus der Distanz zwischen ihnen plus 50.



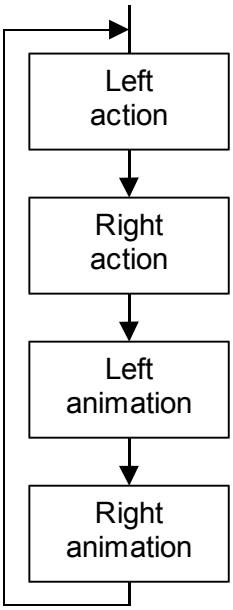
Errechnen Der Kamerapositionen

Diese Funktion schreiben wir ganz ans Ende unseres Skriptes:

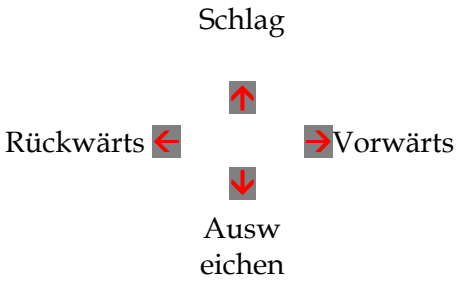
```
function pos_camera()
{
   camera.z = 90;
   camera.x = ((right.x+left.x)/2);
   camera.y = - ((right.x-left.x) + 50);
   wait (1);
}
```

Der Spielverlauf

Das Spiel kan in 4 Hauptphasen unterteilt werden:



Die Aktionen des "Left"- Players werden über die Pfeiltasten wie folgt gesteuert:



Die Aktionen des "Right"-Players setzen künstliche Intelligenz vorraus, welche in unserem Fall doch recht kurz zu halten sein wird. Wir wollen 3 mögliche Zustände für diesen Spieler:

- Prudent
- Defensive
- Offensive

Die Übergänge zwischen diesen Zuständen hängen teilweise von unserem Verhalten und zu einem weiteren Teil von einem eingesetzten Zufallsfaktor ab.

Folglich nehmen wir die folgenden Variablen in unser Skript auf:

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

Die Zustände der Spieler

Wie wir bei der Modelauswahl bereits beschlossen haben, bleiben uns 6 Zustände und zwar:

Warten (waiting)	**Ausweichen (dodging)**	**Schlagen (punching)**
Gewinnen (win)	**Schmerz (pain)**	**Tod (dead)**

Daher schreiben wir die folgenden Variablen ins Skript:

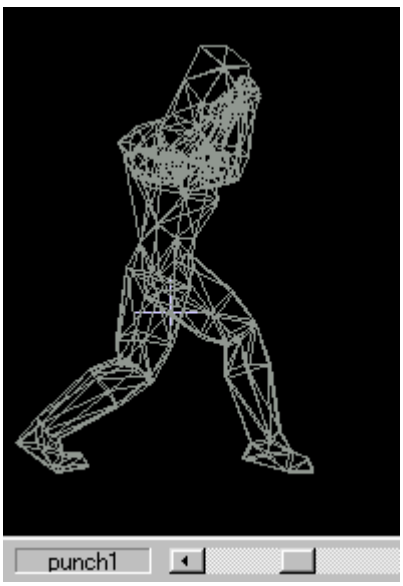
```
var waiting = 1;
var dodging = 2;
var punching = 3;
var pain = 4;
var dead = 5;
var win = 6;

var right_action_state;
var left_action_state;
```

Ausserdem wollen wir im Verlaufe einiger dieser Aktionen noch die passende Vertonung hören. Fügen wir also noch die folgenden Sounds hinzu:

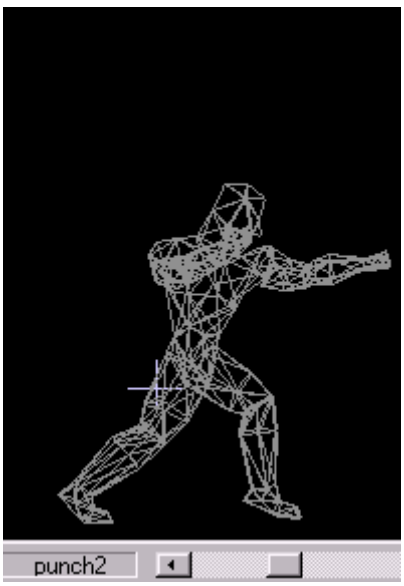
```
sound dead_sound = <dead.wav>;
sound punch_sound = <punch.wav>;
sound punch_dummy_sound = <punch_dummy.wav>;
sound left_win_sound = <left_win.wav>;
sound dodge_sound = <dodge.wav>;
sound lark_sound = <lark.wav>;
sound right_win_sound = <right_win.wav>;
sound gong_sound = <gong.wav>;
sound swing_sound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

Die Animation

Für die Animation verwenden wir die Anweisungen **`ent_frame()`** und **`ent_cycle(),`**wenn wir aber innerhalb einer Animation eingreifen müssen, brauchen wir, angepasst für jede Entity, den Parameter **`frame.`** Ein Beispiel: wenn der rechte Spieler nach Ihnen schlägt, können Sie dem Schlag beim ersten Frame noch ausweichen, beim zweiten Frame ist es aber zu spät, der Schlag wurde bereits fertig ausgeführt.



Noch Können Sie Dem Schlag Ausweichen ...



....zu Spät Zum Ausweichen

Dies erfordert die Analyse eines jeden Animationszyklus um so die Frame-Nummern für Anfang, Ende und, falls nötig, auch irgendwo in der Mitte herauszufinden.

Sie haben Glück, mit diesem Model habe ich das bereits für Sie erledigt. Für die anderen Models müssen Sie es leider selber machen.

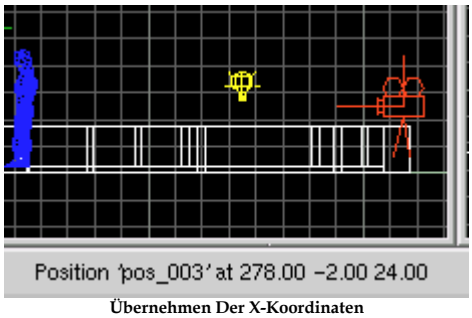
Definieren wir also die folgenden Variablen:

```
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; //warten
var end_left_frame = 48;
var start_right_frame = 10; //warten
var end_right_frame = 12;
```

Tatsächlich korrespondiert das Endframe mit dem letzten Frame +1.

Nun sind noch ein paar Variablen übrig, die wir definieren müssen, wie die Umrandung des Platzes oder die Distanz innerhalb derer der Schlag eines Players den Gegner trifft .

Um die Ecken unseres Territoriums abzustecken, bedienen wir uns wieder unserer beiden Links-Rechts-Licht-Positionen in WED und verwenden diese als X-Koordinaten.



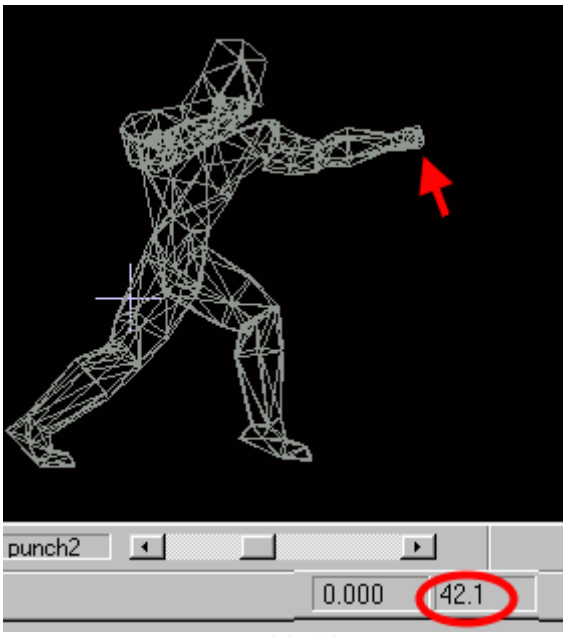
Übernehmen Der X-Koordinaten

Wir notieren also 278 und definieren die beiden folgenden Variablen:

```
var left_edge = -250; //auf der Levelbreite basierend
var right_edge = 250; //auf der Levelbreite basierend
```

Um die Minimaldistanz zwischen den beiden Gegnern herauszufinden, öffnen wir den Frame **punch2** (ausgestreckter Arm) in MED. Nun plazieren wir die Maus an der äussersten Stelle des

gestreckten Arms und schon haben wir unseren X-Wert.



Ausmessen Des Minimaldistanz-Wertes

Erstellen wir nun die folgenden beiden Variablen:

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

Und wir komplettieren unsere Variablen indem wir die beiden Folgenden, deren Nutzen sich noch im Weiteren erklären wird, festlegen:

```
var left_state_attack_block = 0; // 1 = angreifen 2 = abblocken;
var alea;
```

Nun sind wir endlich fertig zum Kampf.

Ich weiss, wie frustrierend es ist, wenn man Code schreibt, ohne ihn testen zu können. Wir sind also zuerst am blauen Spieler interessiert und werfen uns mit wechselnden Animationen ins Gewühl.

Das Spiel

Dafür kreieren wir nun unsere Spielefunktion **`function game()`** und positionieren unserer Labels.

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

```
   end_right_frame = end_wait;
   play_sound (gong_sound,100);

   while (1)
   {
      //hier left action

      //hier right action

left_anime:
      //hier left animation

right_anime:
      //hier right animation

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

Und fügen Sie am Schluss der **main**-Funktion ...

```
game();
```

.... hinzu.

Jetzt wollen wir das Ganze mal testen. Wenn alles stimmt, hören Sie den Gong und die Gegner stehen fertig zum Kampf in Position.

Der Bewegungsablauf des “Left”-Spielers

Tippen Sie die folgenden Zeilen:

```
      //hier animation
   dist_between_right_left = right.x - left.x;    //left bewegt sich vorwärts falls er weder gewonnen
                                                   // hat, noch tot ist

   if ((key_cur == 1) && (left_action_state != win) && (left_action_state != dead))
   {
      //achten sie darauf, dass left sich nach rechts bewegen kann
      if (dist_between_right_left > min_dist_between_right_left)
      {
         left.x += 2;
         pos_camera();
      }
   }

   //left bewegt sich rückwärts falls er nicht gewonnen hat, noch tot ist
   if ((key_cul == 1) && (left_action_state != win) && (left_action_state != dead))
   {
      //achten sie darauf, dass left nicht über die linke Kante kann
      if (left.x > left_edge)
      {
         left.x -= 2;
         pos_camera();
      }
   }
```

Probieren Sie das ganze jetzt aus. Bewegen Sie sich vorwärts und zurück, überprüfen Sie ob Sie nicht doch irgendein gesetztes Limit überschreiten können. Falls dem so ist, passen Sie die **left_edge** und **right_edge**-Werte neu an. Bewundern Sie auch die Kamerabewegungen. Sobald der "Right"-Spieler Sie k.o. schlägt, ist dann allerdings gar keine Zeit mehr, irgendwas zu betrachten.

Wir müssen in der Tat zugeben, dass sich unser Player in einer etwas eigenartiger Weise vor- und zurück bewegt. Um dem abzuhelfen fügen Sie die folgenden Zeilen **nach left_anime** ein:

```
//hier 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;
}
```

Sie haben es begriffen und ich brauche Ihnen eigentlich nicht zu sagen, dass Sie das in jeder Phase machen können.

Ich weiss, Sie sind so ungeduldig wie ich. Lassen Sie uns also unserem Gegner ein wenig Leben einhauchen. Dazu tippen Sie das folgende **nach 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;
}
```

Wir werden uns verteidigen!

Um das zu tun, müssen wir den folgenden Code in unseren Abschnitt **left_action** und zwar **nach** den Bewegungen einfügen:

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

   //links ausweichen
   if ((key_cud == 1) && ( left_state_attack_block == 0 ))
   {
      if ((left_action_state != dodging) && (left_action_state != dead) && (left_action_state != win))
      {
         //Left weicht aus
         left_action_state = dodging; //ausweichen
         left_state_attack_block = 2; //blocken
         start_left_frame = start_dodge;
         left.frame = start_left_frame;
         end_left_frame = end_dodge;

      }
```

```
    }
```

Und in unserer **left_anim** fügen wir **nach** der Warte-Aktion (waiting action) das folgende ein:

```
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;
}
```

Wen wir das unten stehende eintippen, können wir Schläge austeilen. Allerdings sollten wir uns noch ein wenig gedulden, denn die Animation verlangt doch etwas mehr an Erklärung.

Gehen Sie also in die **left_action** und machen sie weiter mit:

```
    //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;}
```

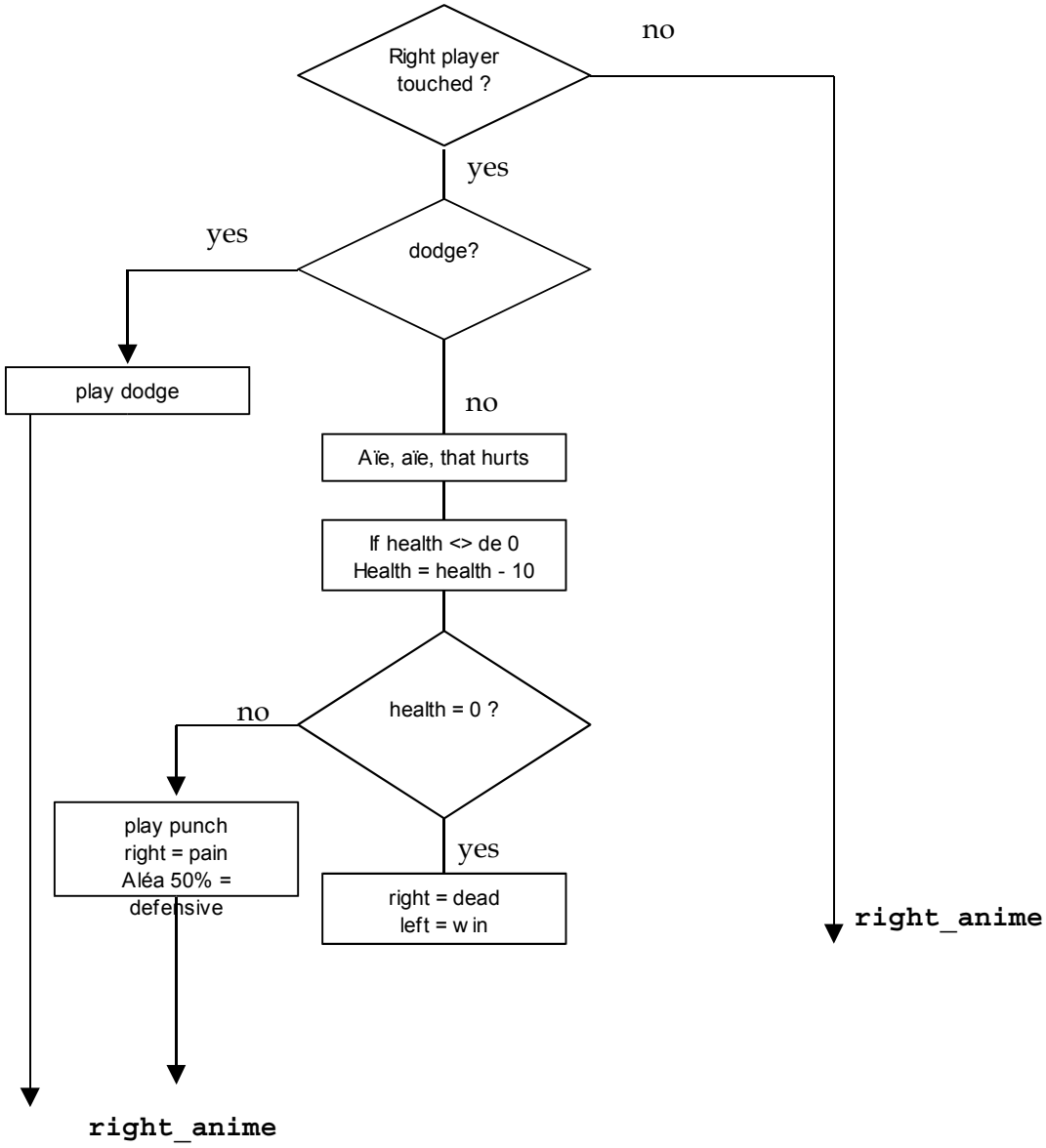
Nun machen wir mit der **left_animation** weiter und kommentieren die Schlaganimation:

```
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; //warten
      left.frame = start_left_frame;
      end_left_frame = end_wait;
```

Soweit ist alles in Ordnung. Einer schlägt und der andere wartet. Wenn Sie das unbedingt ausprobieren müssen, bitte, tun Sie sich keinen Zwang an und nutzen Sie den Vorteil: zertrümmern Sie seinen Schädel - bald wird er sich wehren können! Beenden Sie mit 2 Klammern

und vergessen Sie nicht, diese einzurücken.

Und so machen wir weiter:



Und so schreiben wir es auf:

```
//haben wir den Right-Spieler getroffen?
if (dist_between_right_left < min_dist_between_right_left+ 4)
        {
   //Right könnte ausweichen
   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 ist tot
                right_action_state = dead;
                start_right_frame = start_dead;
                right.frame = start_right_frame;
                end_right_frame = end_dead;

                //Left hat gewonnen
                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;

            //was soll Right machen?
            if (random (2) < 1){right_state_of_mind = defensive;}
         }
      }

      goto right_anime;
   }
```

Und ich verbiete Ihnen, ihn zusammen zu schlagen...

Auch unser "Left"-Spieler kann Schläge einstecken.

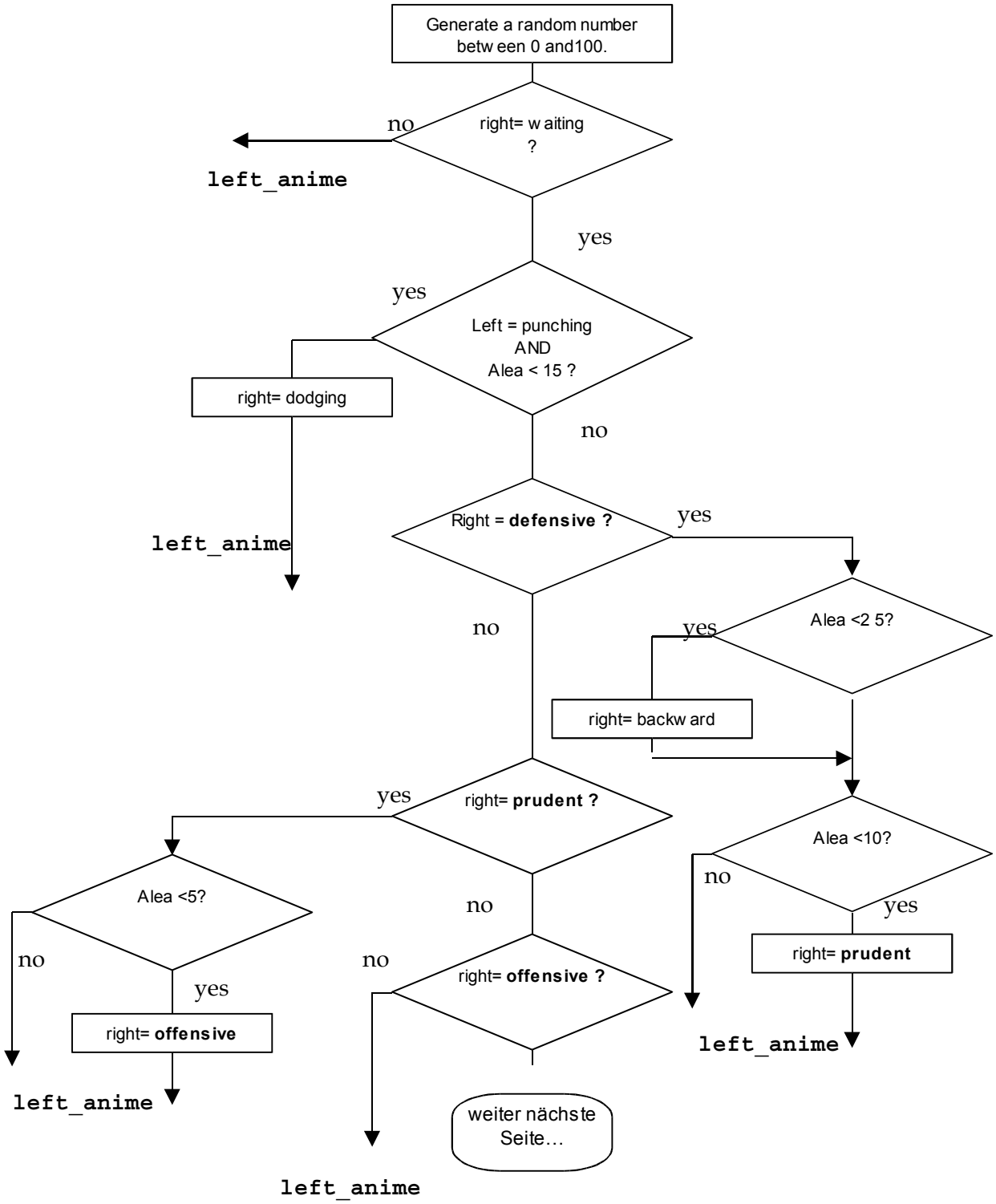
Führen Sie die **`left_animation`** weiter und tippen Sie:

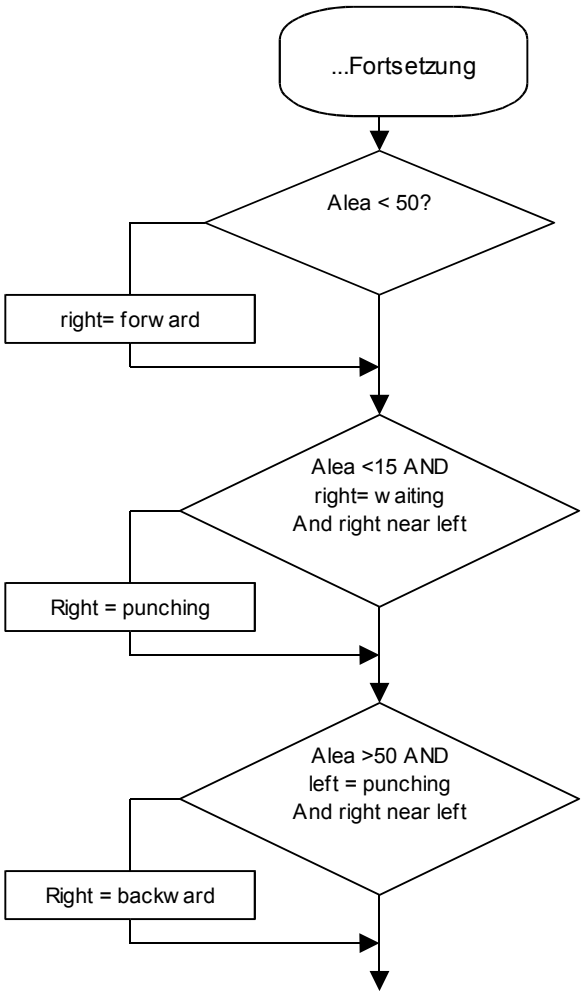
```
if (left_action_state== pain)
{
   if (left.frame == start_left_frame)
   {
      play_sound (punch_sound,50);
      if (left.x > left_edge){left.x -= 10;}
   }
   left.frame += .5*time;
   left.next_frame = 0;

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

Wir sind mit dem "Left"-Spieler fertig (was damit endet, dass er tot ist, dazu sind wir allerdings

nicht gezwungen).

Was den "Right"-Spieler betrifft, sollten wir nun seine künstliche Intelligenz analysieren.





left_anime

Das Gleiche in WDL-Code:

```
if (right_action_state != waiting) {goto left_anime;}
   // Zufallswerte, die festlegen was "Right" auf seinem Zustand basierend tun sollte
   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
   {
      //achten Sie darauf, dass Right sich nach links bewegen kann
      if (dist_between_right_left > min_dist_between_right_left)
      {
         right.x -= 2;
         pos_camera();
         //son de pas
      }
   }

   //Angriff oder Nicht-Angriff? das ist hier die Frage
   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 rückwärts oder nicht, falls Left schlägt
   if ((left_action_state == punching) && (alea > 50) && (right_action_state== waiting)
      && (dist_between_right_left < min_dist_between_right_left+ 10))
   {
      //achten Sie darauf, dass Right nicht über die rechte Kante entwischt
      if (right.x < right_edge)
      {
         right.x += 1;
         pos_camera();
      }
   }
}
```

Die Animation des roten Spielers ist fast identisch mit der des Blauen. Wir tippen also die folgenden Zeilen in die Aktion **`right_anime`** und zwar **nach** dem '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; //warten
      right.frame = start_right_frame;
      end_right_frame = end_wait;

      //haben wir den Left-Spieler getroffen?
      if (dist_between_right_left < min_dist_between_right_left+ 4)
               {
         //Right könnte ausweichen
         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 ist tot
            left_action_state = dead;
            start_left_frame = start_dead;
            left.frame = start_left_frame;
            end_left_frame = end_dead;

            //Right hat gewonnen
            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; //warten
         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; //warten
         right.frame = start_right_frame;
         end_right_frame = end_wait;
      }
      goto fin;
   }
```

Nun bleibt uns noch Tod und Sieg für unsere beiden Spieler zu verwalten.

Und das geht so:

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 gewinnt
         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;
   }
```

... und in **right_anime** :

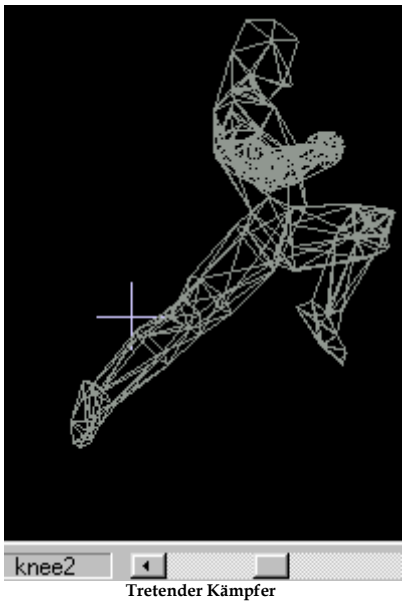
```
if (right_action_state== dead)
{
   right.frame += time;
   right.next_frame = 0;

   //son quand touche sol

   if (right.frame >= end_right_frame)
   {
      //Left gewinnt
      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; //warten
         left.frame = start_left_frame;
         end_left_frame = end_wait;
         right_action_state = waiting;
         start_right_frame = start_wait; //warten
         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;
}
```

Schlussfolgerung

Ich würde ja gerne sagen “das war's” einigermassen stimmt es ja auch. Trotzdem können wir das nicht einfach so lassen.

Was kann man denn noch hinzufügen?

Das Erste zeigt uns ein Blick auf unseren Kämpfer in MED. Wie wir sehen kann er auch mit den Knien treten. Wenn Ihnen das gefällt, nur zu, legen Sie los und fügen Sie das selber ein - das kann nun ja nicht mehr wirklich schwierig sein.



Tretender Kämpfer

Zusätzliches

Jetzt beabsichtige ich das Ganze zu vervollständigen und mit ein bischen Spass zu garnieren.

Als da z.B. verschiedene Kamera-Ansichten wären, Vogelstimmen, die mit einer Animation für den am Boden liegenden kombiniert sind, ein Startpanel, das die Auswahl der Spielerfarbe ermöglicht, Hintergrundmusik, Publikum etc.

Kameraeffekte

Neben der Kamerasicht, die wir bereits haben, planen wir noch zwei Weitere. Eine zeigt den Blick von hinter dem “Left”-Spieler, die zweite sitzt auf seiner Schulter. Eine dritte Sicht wird immer dann unseren Kopf zeigen, wenn wir einen Treffer abbekommen. Alles in allem haben wir schliesslich vier Ansichten.

Dazu definieren wir eine Variable:

```
var cam_view = 0; //défaut
```

Dann schreiben wir ans Ende unseres Skripts:

```
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();
}
```

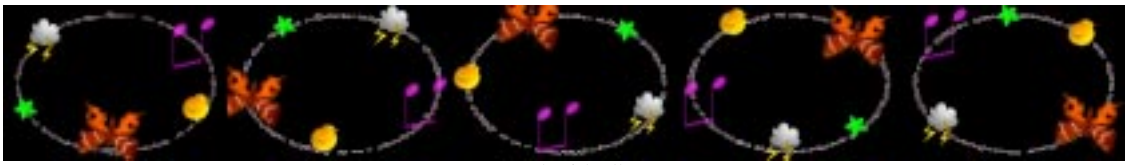
Nun ersetzen wir unsere Funktion **pos_camera** durch diese:

```
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;}
}
```

Ein bischen Animation

Begeben wir uns in die Situation wenn kleine Vöglein zwitschern. Ich habe eine Bitmap mit dem Namen **ko+5.pcx** erstellt und die werden wir jetzt einbinden und dann im richtigen Moment anzeigen.



Es Dreht Sich Alles

Wir fügen diese Zeilen zur Definition unserer Variablen am Beginn des Szenarios ein:

```
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;
}
```

Nun definieren wir einen sogenannten Handle für unseren Sound (gleich nach den Sounddefinitionen), über den wir prüfen können, ob das Abspielen des Sounds beendet ist und wir weiter machen können:

```
var larkhandle = 0;
```

Dann erstellen wir die Funktion **cui_cui** und setzen sie ans Ende:

```
function show_cui_cui()
{
   while (1)
   {
      cui_cui_pos.x += 84;
      if (cui_cui_pos.x >= 400){cui_cui_pos.x =0;}
      waitt (10);
   }
}
```

Nun modifizieren (gesperrte, rote zeilen) wir unsere **pos_camera**-Funktion wie folgt:

```
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;}
}
```

Und folglich modifizieren wir auch unseren Tod:

```
      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)
            …
```

und

```
      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 gewinnt
if (right.frame == start_right_frame){play_sound (right_win_sound,100);}

right.frame += .5*time;

right.next_frame = 0;
```

Ein Startbildschirm

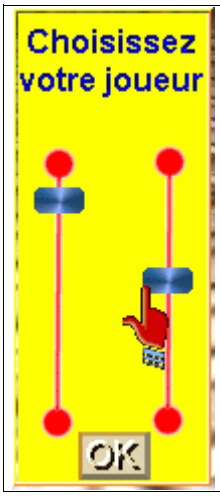
Dieses Panel wird uns das Auswählen einer Playerfarbe erlauben. Also müssen wir zunächst einmal neue Skins für unsere Figuren erstellen. Öffnen Sie MED, exportieren Sie eine unserer beiden Skins in Ihr bevorzugtes Malprogramm und erstellen Skins in anderen Farben.

Ich für meinen Teil habe die folgenden Skins hinzugenommen (in der gegebenen Reihenfolge):

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

Für den Fall, dass Sie nicht genügend ausgerüstet sind, habe ich dieses Model unter dem Namen **`fighter1.mdl`**vorbereitet.

Damit wir auch wählen können, ist es jetzt an der Zeit unser Panel zu erstellen. Bitte schön, hier ist es:



Es besteht aus 5 Elementen:

1. Der Panelhintergrund (**`choix.pcx`**)*
2. Einem Curser, der zweimal verwendet wird (**`cursor.pcx`**)
3. Einem Knopf 'o.k an' (**`button_ok_on.pcx`**)
4. Einem Knopf 'o.k aus' (**`button_ok_off.pcx`**)
5. Einem Mauszeiger (**`pointeur.pcx`**)

* französische Version
nehmen Sie **`choix_e.pcx`** für's englische Panel
oder **`choiw_d.pcx`** für's deutsche Panel

Definiert werden diese Elemente so:

```
bmap choix_map,<choix.pcx>; // oder choix_e.pcx (english version) oder choix_d.pcx (deutsche 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>;
```

Auch definieren wir die beiden Variablen, die von **vslider** eingegeben werden:

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

Nun definieren wir unser 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;
}
```

Gemäss dieser Definition merken Sie sich bitte:

1 – Indem wir auf den 'OK'-Knopf klicken, rufen wir die Funktion **game()** auf.
2 – Der linke Regler schaltet durch die Werte 1 bis 3 (3 mögliche Skins für unseren “Left”).
3 – Der rechte Regler schaltet durch die Werte 4 bis 6 (die 3 möglichen Skins für unsere “Right”-Figur).

Das Einzige, was wir jetzt noch tun müssen, ist es, das Ganze zum Laufen zu bringen:

In der **main()**-Funktion ersetzen wir **game();** durch

```
choix();
```

Dann schreiben wir diese Funktion **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);
   }
}
```

Die beiden ersten Zeilen zentrieren das Panel in Abhängigkeit zur Auflösung am Bildschirm.

Dann stellen wir das Panel dar und anschliessend die Maus. Zum Schluss weisen wir dem "Left"- bzw. "Right"- Spieler aufgrund der von den beiden Reglern zurückgelieferten Werte die betreffende Skinfarbe zu.

Die Zufallszeile ist ein alter Trick den ich immer benutzte wenn Computer keine interne Uhr hatten. Tatsächlich, Sie haben es vielleicht bemerkt, ist es so, dass die Zufallsanweisung bei jeder Ausführung des Programms immer dieselbe Reihe von Zahlen ausgibt. In unserem Fall werden wir niemals zweimal die gleiche Zeit in unserer Schleife verbringen. Da die Zeit von dem Moment abhngängt, an dem jemand auf 'OK' klickt, haben wir garantiert zufällige Zufallszahlen.

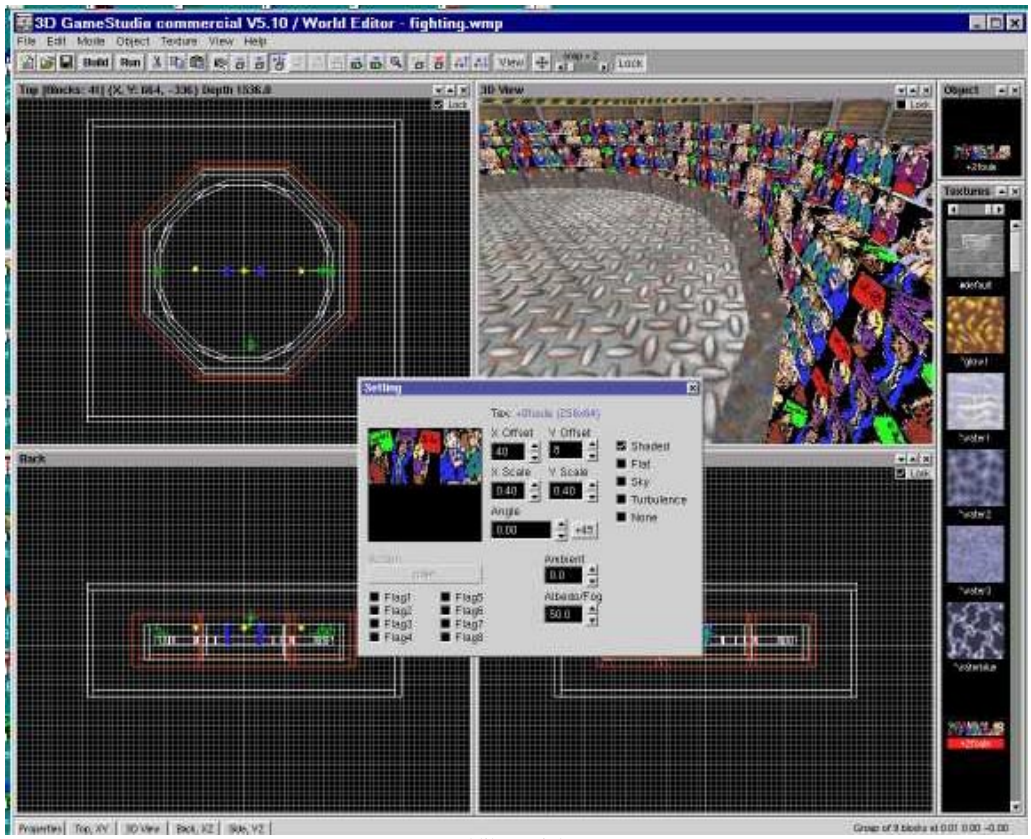
Jetzt fügen wir rasch noch diese beiden Zeilen am Anfang unserer **`game()`**-Funktion ein und schon können wir mit den Farben unserer Wahl zu spielen.

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

Ich überlasse es als Übungsaufgabe Ihnen, die Farbe des Gesundheitsbalkens anzupassen. Es geht genauso wie das Anpassen der Spielerfarben (vorerst bleiben wir noch bei rot und blau).

Das Publikum

Für das Publikum werde ich keine besonderen Details anbringen. Ich füge lediglich 2 hohle Zylinder für die Tribünen hinzu und weise eine animierte Textur zu (Die finden Sie im Verzeichnis: **`+0foule.pcx`** in **`+3foule.pcx`**).



Publikumseinlass

Das Ergebnis Ist Doch Ganz Nett

Die Hintergrundmusik

Auch die Hintergrundmusik ist eher problemlos:

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

// spiele einen Song ab, wenn die Lautstärke am beginn gesetzt ist
function start_song()
{
   wait(4); // wait until load_info has loaded the last volume setting
   if (midi_vol > 0) {
      play_song(amb_song,80);
   }
}
```

Tippen Sie die folgende Zeile in **`main()`** nach **`load_status:`**

```
start_song();
```

Schluss (2)

Ich denke, dieses Mal sind wir fertig. Wir haben nichts weiter zu tun als uns zu verabschieden...

Dennoch, denken Sie vielleicht, es wäre doch ganz nett, wenn jeder Kerl passend zu seiner Farbe auch eine eigene Persönlichkeit und infolge dessen auch unterschiedliche Reaktionen hätte. Was für eine absurde Idee! Machen Sie das mal...

Naja, ich habe schon Lust, Ihnen ein bischen zu helfen noch etwas mehr in die prächtige Welt des

WED hineinzuwachsen. Also abgemacht, aber danach liegt es an Ihnen.

Wir machen es einfach:

- **Erster Spieler**:
- Der Schwache, dessen Treffer den Gegner lediglich 5 Lebenspunkte kostet, aber jeder eingesteckte Schlag zieht ihm 5 Punkte mehr ab, als dem Gegner in dem Fall. Aber seien Sie so nett und lassen Sie ihm eine Chance auf einen glücklichen Sieg: wir werden ihn mit einen geheimen Stiefel ausstatten! Ein Kniestoss ist demnach unhaltbar und schlägt den Gegner zu Boden. (Der Kniekick ersetzt den Hieb in unvorhersehbarer Weise: z.B. mit einer Wahrscheinlichkeit von 1 zu 10).
-
- **Zweiter Spieler:**
- Der, den wir bereits kennen, wir sollten also gar nichts ändern.
-
- **Dritter Spieler:**
- Der Brutale, jeder Treffer von ihm kostet den Gegner 15 Gesundheitspunkte. Steckt er einen Schlag ein, so verliert er 5 Punkte weniger, als der andere wenn er getroffen wird. Einem Kniestritt kann er mit einer schnellen Rückwärtsbewegung ausweichen.

Also haben wir:

Schlag ausgeteilt von	Schlag eingesteckt von	Bonus/malus	abzuziehende Gesundheitspunkte
Schwach = 5 points	Schwach	+5	10
	Normal	0	5
	Brutal	-4	1
Normal = 10 points	Schwach	+5	15
	Normal	0	10
	Brutal	-4	6
Brutal = 15 points	Schwach	+5	20
	Normal	0	15
	Brutal	-4	11

Die Treffer, die der "Left"-Player landet **`left.skin*5`** bedeuten 5 für den Schwachen, 10 für den Normalen und 15 für den Brutalen.
Die Treffer, die der "Right"-Player landet **`(right.skin-3)*5`** bedeuten 5 für den Schwachen, 10 für den Normalen und 15 für den Brutalen.

Wir erstellen zwei Variablen:

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

Dann berechnen wir ihre Werte: (fügen Sie diese Zeilen am Anfang der Funktion **`game()`** ein)

```
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;}
```

Für die Gesundheitspunkte ersetzen wir einfach die im folgenden blauen Zeilen durch die roten:

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

            if (right_health == 0)
```

durch :

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

            if (right_health <= 0)
```

und

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

if (left_health == 0)
```

durch:

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

            if (left_health <= 0)
```

und

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

durch:

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

Tritte mit dem Knie

Der "Right"-Spieler kann uns mit dem Knie treten. Lieber als eine zusätzliche Aktion zu schreiben, ziehe ich es vor die Schlagaktion zu modifizieren: eines von zehn Malen wird der Hieb durch den Knietritt ersetzt.

Zuerst definieren wir die folgenden Variable, die wir hinter die bereits bestehenden plazieren.

```
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 ;
```

Dann fügen wir die Zeilen in Rot hinzu:

```
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)
                       {
```

Was den "Left"-Player betrifft, können wir uns entscheiden, ob wir entweder dasselbe machen (Hieb durch Kick ersetzen) oder eine weiter Taste zur Kontrolle der Tritte hinzunehmen. Ich habe mich für die zweite Möglichkeit entschieden und so kann man durch Drücken der **STRG.**-Taste mit dem Knie kicken.

Wir tippen also die folgenden Zeilen **vor** der **`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;}
     //hier right action
---------- - - -  - -- - - --- -
```

Dann verändern wir diese Zeilen:

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

   if (right_health <= 0)
           {
```

Zum guten (3.) Schluss

Tatsächlich, wir sind wirklich fertig! Schliesslich sagen wir nun doch 'und tschüss'...

Und was lässt sich nach einem gut zu spielenden Game sonst noch machen?

- Richten Sie eine Bestenliste ein und messen Sie sich mit Freunden oder Ihrer Familie.
- Machen Sie Aufschlagpunkte durch die Verwendung von Partikeln sichtbar
- Variieren Sie die Animationsgeschwindigkeit gemäss der Spielerstärke: ein schwacher Player hat langsamere Bewegungen, als ein starker.
- Das gleiche gilt für den Sound. auch der sollte proportional zur Kraft des Spielers eingestellt sein.
-
- Etc.

Der Kampf Kann Beginnen

PS: Sie finden die kompletten Datein dieses Szenarios unter dem Namen **`fighting.all`** .