Move a car down the street then teleport to start point and do it again.

```
action street_car() // attach this action to your car model
{
VECTOR init_pos;
vec_set(init_pos.x, my.x);
while (1)
{
// 10 sets the speed of the car
c_move (my, vector (10 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
// the car has move 2000 quants away from the starting point? (play with 2000)
if (vec_dist (my.x, init_pos.x) > 2000)
vec_set(my.x, init_pos.x); // then teleport it at its starting point
wait (1);
}
}
```

MP3 file that plays one time after pressing a key. If the sound is still playing and you press the same key, the sound must start over

```
var track_handle;
function play_track()
{
if (track_handle) // the sound is playing already?
media_stop(track_handle); // then stop it!
track_handle = media_play("track1.wav", NULL, 100); // play the sound track from its beginning
}
function init_startup()
{
on_p = play_track;
}
```

can I make certain entities in my level to be visible even if they are behind walls?

```
action hidden_entities()
{
while (1)
{
// press the "v" key to make the entities visible
if (key_v)
set(my, ZNEAR);
else
reset(my, ZNEAR);
wait (1);
}
}
```

Main menu buttons slide down and stop at their final positions.

```
function new_game();
function exit_game();
BMAP* newgame1_pcx = "newgame1.pcx";
BMAP* newgame2_pcx = "newgame2.pcx";
BMAP* exit1_pcx = "exit1.pcx";
BMAP* exit2_pcx = "exit2.pcx";
PANEL* main_pan =
{
layer = 10;
bmap = "main.jpg";
pos_x = 0;
pos_y = 0;
button (250, 0, newgame2_pcx, newgame1_pcx, newgame2_pcx, new_game, NULL, NULL);
button (250, 0, exit2_pcx, exit1_pcx, exit2_pcx, exit_game, NULL, NULL);
flags = SHOW;
}
function buttons_startup()
{
var button_offset = 0; // the first button will move until its y position is 200
fps_max = 75; // limit the frame rate to 75 fps in order to keep the panel sliding smoothly
wait (-3); // give the computer a few seconds to display the main panel bitmap
// the buttons will slide downwards until the first one is place at y = 250
// the second button will be placed 100 quants below the first one, etc
while (button_offset < 250)
{
pan_setbutton(main_pan, 1, 0, 250, button_offset, newgame2_pcx, newgame1_pcx, newgame2_pcx, NULL, new_game, NULL, NULL);
// notice that the second button has an extra 100 pixels offset on the y axis
pan_setbutton(main_pan, 2, 0, 250, button_offset + 100, exit2_pcx, exit1_pcx, exit2_pcx, NULL, exit_game, NULL, NULL);
button_offset += 1; // add 1 pixel / frame (about 75 pixels per second) to the offset
wait (1);
}
}
function new_game()
{
wait (1); // use your own code here
}
function exit_game()
{
sys_exit(NULL);
}
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

a menu to pop up when I right click a button on a panel.

```
BMAP* mainpanel_pcx = "mainpanel.pcx";
BMAP* options_pcx = "options.pcx";
BMAP* options1_png = "options1.png";
BMAP* options2_png = "options2.png";
BMAP* menuoption1_png = "menuoption1.png";
BMAP* menuoption2_png = "menuoption2.png";
BMAP* pointer_tga = "pointer.tga";
function mouse_over();
function cancel_options();
PANEL* main_pan =
{
bmap = mainpanel_pcx;
layer = 15;
button(40, 10, options2_png, options1_png, options2_png, NULL, NULL, mouse_over);
flags = SHOW;
}
PANEL* options_pan =
{
bmap = options_pcx;
pos_x = 300;
pos_y = 200;
layer = 15;
button(100, 150, menuoption2_png, menuoption1_png, menuoption2_png, cancel_options, NULL, NULL);
}
function mouse_over()
{
// the loop below will run for as long as the mouse pointer wasn't moved away from the button
while (event_type != EVENT_RELEASE)
{
if (mouse_right) // the right mouse button was clicked while the cursor was placed over the button?
set(options_pan, SHOW);
wait (1);
}
}
function cancel_options()
{
reset(options_pan, SHOW);
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

use the mouse button to move the barrels in my warehouse one by one to another position

```
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function move_entity()
{
VECTOR temp;
var entity_offset = 0;
my.skill1 += 1;
temp.z = 200; // the default value places the entities
if ((my.skill1 % 2) == 1) // clicked the entity?
{
while (mouse_left) {wait (1);} // wait until the player releases the left mouse button
while (!mouse_left) // move the entity until the player presses the mouse button again
{
temp.x = mouse_cursor.x;
temp.y = mouse_cursor.y;
temp.z = 200 + entity_offset;
entity_offset += mickey.z; // use the mouse wheel to bring the entity closer or farther away from the camera
vec_for_screen(temp.x, camera);
vec_set (my.x, temp.x);
set (my, PASSABLE);
wait (1);
}
}
else // drop the object here
{
vec_set (temp.x, my.x);
temp.z -= 3000; // trace up to 3,000 quants below
// make sure to drop the object on the ground
my.z -= c_trace (my.x, temp.x, IGNORE_ME + IGNORE_SPRITES + USE_BOX);
reset (my, PASSABLE);
}
}
action my_barrels()
{
my.skill1 = 0;
my.emask |= ENABLE_CLICK;
my.event = move_entity;
}
```

to create an entity that follows the player.

```
function player_follower();
action players_code() // attach this action to your player model
{
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
ent_create("guard.mdl", nullvector, player_follower);
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = 0;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50;
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
wait (1);
}
}
function player_follower()
{
var walk_percentage;
while (1)
{
// play with the numerical values; they give the xyz offset of the follower in relation with the player
vec_set(my.x, vector(150, -60, 0));
vec_rotate(my.x, you.pan);
vec_add(my.x, you.x);
my.pan = you.pan;
ent_animate(my, "walk", walk_percentage, ANM_CYCLE); // play the "stand" aka idle animation
walk_percentage += 5 * (key_w - key_s) * time_step; // increase walk_percentage only if the player is moving
wait (1);
}
}
```

player to shoot fireballs that get destroyed when they collide with other entities

```
BMAP* fire_tga = "fire.tga";
STRING* fireball_mdl = "fireball.mdl";
SOUND* fireball_wav = "fireball.wav";
SOUND* destroyed_wav = "destroyed.wav";
function fire_fireball();
function shoot_fireball();
function remove_fireball();
function fade_fire(PARTICLE *p);
function fire_effect(PARTICLE *p);
function init_startup()
{
on_mouse_left = fire_fireball;
}
function fire_fireball()
{
ent_create (fireball_mdl, player.x, shoot_fireball);
snd_play (fireball_wav, 50, 0);
}
function shoot_fireball()
{
VECTOR fireball_speed, temp;
my.emask |= (ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_fireball;
set (my, PASSABLE);
my.pan = camera.pan;
my.tilt = camera.tilt;
my.skill20 = 0;
vec_set(fireball_speed.x, nullvector);
fireball_speed.x = 100 * time_step;
my.skill10 = 0;
while (my.skill20 < 500)
{
effect(fire_effect, 10, my.x, nullvector);
if (vec_dist (player.x, my.x) > 100)
reset (my, PASSABLE);
if ((my.skill10 == 0) && (vec_dist (my.x, player.x) > 100))
{
// play with 500, sets the damage range
c_scan(my.x, my.pan, vector(40, 60, 500), IGNORE_ME | SCAN_ENTS);
}
my.skill20 += 1 * time_step;
c_move (my, fireball_speed.x, nullvector, IGNORE_FLAG2 | IGNORE_PASSABLE);
wait (1);
}
remove_fireball();
}
function remove_fireball()
{
wait (1);
my.event = NULL;
ent_playsound (my, destroyed_wav, 1000); // play the explosion sound
set (my, INVISIBLE); // hide the fireball, keep ent_playsound playing
wait (-1.5); // wait for 1.5 seconds
ent_remove(me); // now remove it
}
function fade_fire(PARTICLE *p)
{
p.alpha -= 4 * time_step; // fade out the fire particles
if (p.alpha < 0)
p.lifespan = 0;
}
function fire_effect(PARTICLE *p)
{
set (my, PASSABLE);
p->vel_x = 1 - random(2);
p->vel_y = 1 - random(2);
p->vel_z = 3 - random(6);
p.alpha = 20 + random(50);
p.bmap = fire_tga;
p.size = 10 + random(5); // gives the size of the flame particles
p.flags |= (BRIGHT | MOVE);
p.event = fade_fire;
}
```

the best way to make an NPC disappear at the end of his path and go back at the beginning over and over

```
var entity_speed = 3;
var movement_enabled = 0;
var dist_to_node;
var current_node = 1;
var angle_difference = 0;
ANGLE temp_angle, pos_node;
function move_target()
{
while(1)
{
if(movement_enabled)
{
entity_speed = minv(5, entity_speed + 0.5 * time_step);
c_move(my, vector(entity_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x));
}
wait(1);
}
}
action my_entity() // attach this action to a model
{
VECTOR init_pos;
vec_set (init_pos.x, my.x); // store the initial coordinates
move_target();
result = path_scan(me, my.x, my.pan, vector(360, 180, 1000));
if (result) {movement_enabled = 1;}
path_getnode (my, 1, pos_node, NULL);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate towards the node
while(1)
{
dist_to_node = vec_dist(my.x, pos_node);
if(dist_to_node < 50) // close to the node?
{
current_node = path_nextnode(my, current_node, 1);
if (!current_node) // reached the end of the path? Then start over!
{
current_node = 1; // reset the node
vec_set (my.x, init_pos.x); // and then teleport the player at its initial position in the level
}
path_getnode (my, current_node, pos_node, NULL);
}
wait(1);
}
}
```

control a robot by reading the movement instructions from a text file

```
var string_handle;
var eof_reached = 0; // will be set to -1 when the end of the file (eof) is reached
var temp_coords;
STRING* direction_str = "#10";
ENTITY* robot;
action my_robot()
{
robot = my;
}
function read_startup()
{
while (!robot) {wait (1);} // wait until the robot model is loaded
wait (-5); // give the player some time to prepare for the show (some monitors need a lot of time to switch to the proper resolution)
string_handle = file_open_read("commands.txt"); // open the commands.txt file
while (eof_reached != -1) // the end of the file wasn't reached yet?
{
eof_reached = file_str_read(string_handle, direction_str); // read a string from the file
if (str_cmpi(direction_str, "left")) // read a "left" command? Then let's increase the y coordinate
{
temp_coords = robot.y;
while (robot.y < temp_coords + 50)
{
c_move(robot, nullvector, vector(0, 5 * time_step, 0), IGNORE_PASSABLE);
wait (1);
}
}
if (str_cmpi(direction_str, "right")) // read a "right" command? Then let's decrease the y coordinate
{
temp_coords = robot.y;
while (robot.y > temp_coords - 50)
{
c_move(robot, nullvector, vector(0, -5 * time_step, 0), IGNORE_PASSABLE);
wait (1);
}
}
if (str_cmpi(direction_str, "forward")) // read a "forward" command? Then let's increase the x coordinate
{
temp_coords = robot.x;
while (robot.x < temp_coords + 50)
{
c_move(robot, nullvector, vector(5 * time_step, 0, 0), IGNORE_PASSABLE);
wait (1);
}
}
if (str_cmpi(direction_str, "back")) // read a "back" command? Then let's decrease the x coordinate
{
temp_coords = robot.x;
while (robot.x > temp_coords - 50)
{
c_move(robot, nullvector, vector(-5 * time_step, 0, 0), IGNORE_PASSABLE);
wait (1);
}
}
wait (-1); // wait a second after each command
}
file_close (string_handle); // all the strings are read here, so close the file
}
```

the best way to display a dynamic text in 3D

```
STRING* display_str = "#40"; // backup string
STRING* temp_str = "#40"; // backup string
var str_count;
var string_index = 0;
var position_y = 500;
function letter_function()
{
set(my, PASSABLE);
my.pan = 270; // orient the letters according to your needs
}
function letters_startup()
{
str_cpy (display_str, "great graphics");
while (!player) {wait (1);} // wait until the player is loaded (if needed)
position_y = 800;
str_count = str_len(display_str);
while (string_index < str_count)
{
str_cpy (temp_str, display_str); // copy display_str to temp_str (we don't want to destroy it)
str_clip(temp_str, string_index); // cut the needed number of characters from the beginning of the string
str_trunc(temp_str, str_count - string_index - 1); // and cut from the end as well
// here we've got the letters that create our string, one by one
// in this example we create them along the y axis (the value of y is decreased at all times until all the letters are in place)
if (str_cmpi(temp_str, "a"))
ent_create("a.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "b"))
ent_create("b.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "c"))
ent_create("c.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "d"))
ent_create("d.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "e"))
ent_create("e.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "f"))
ent_create("f.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "g"))
ent_create("g.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "h"))
ent_create("h.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "i"))
ent_create("i.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "j"))
ent_create("j.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "k"))
ent_create("k.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "l"))
ent_create("l.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "m"))
ent_create("m.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "n"))
ent_create("n.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "o"))
ent_create("o.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "p"))
ent_create("p.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "q"))
ent_create("q.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "r"))
ent_create("r.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "s"))
ent_create("s.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "t"))
ent_create("t.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "u"))
ent_create("u.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "v"))
ent_create("v.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "w"))
ent_create("w.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "x"))
ent_create("x.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "y"))
ent_create("y.mdl", vector(-100, position_y, -100), letter_function);
if (str_cmpi(temp_str, "z"))
ent_create("z.mdl", vector(-100, position_y, -150), letter_function);
position_y -= 120; // set an offset of 120 quants between 2 consecutive letters
string_index += 1; // increase string_index
}
}
```

a teleport, an entity that teleports you to another place in the level when you walk on it

```
action my_teleport()
{
set(my, POLYGON); // use polygon-based collision detection
// your player action or function should include a "player = my;" line of code
while (!player) {wait (1);} // wait until the player model is loaded
while (1)
{
if (vec_dist(player.x, my.x) < 100) // trigger distance, play with 100
{
while (key_any) {wait (1);} // wait until the player releases all the keys
// now teleport the player to its new position (choose the vector coordinates properly)
vec_set (player.x, vector (400, 700, -150));
}
wait (1);
}
}
```

a working health bar code for the player and an enemy

```
var players_health = 100;
var enemy_health = 100;
BMAP* health_pcx = "health.pcx";
PANEL* health_pan =
{
pos_x = 10;
pos_y = 20;
layer = 10;
window(50, 0, 40, 20, health_pcx, players_health, 0);
flags = SHOW;
}
PANEL* enemy_pan =
{
pos_x = 10;
pos_y = 20;
layer = 10;
window(500, 0, 40, 20, health_pcx, enemy_health, 0);
flags = SHOW;
}
// just for testing, use the 1... 4 keys to increase / decrease the health for the player and for the enemy
function health_startup()
{
while (1)
{
if (key_1) players_health += 2 * time_step;
if (key_2) players_health -= 2 * time_step;
if (key_3) enemy_health += 2 * time_step;
if (key_4) enemy_health -= 2 * time_step;
wait (1);
}
}
```

top view camera to point towards any entity I'm touching with the mouse pointer

```
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function camera_startup()
{
camera.z = 600; // play with the z value
camera.tilt = -80; // with the tilt angle
camera.arc = 100; // and with the arc value until you see all the entities at once
VECTOR temp;
while (1)
{
if (mouse_ent)
{
vec_set(temp, mouse_ent.x);
vec_sub(temp, camera.x);
vec_to_angle(camera.pan, temp);
}
wait (1);
}
}
```

a lite-C script for a mini camera? The camera would be fixed somewhere in a level and the movie would be shown at a screen somewhere else at the level.

```
var fixed_camera = 1;
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
// activates the fixed camera for as long as fixed_camera is set to 1
function camera_startup()
{
while (1)
{
if (fixed_camera)
{
camera.x = -700;
camera.y = -365;
camera.z = 450;
camera.pan = 40;
camera.tilt = -35;
}
wait (1);
}
}
// attach this action to a sprite or a model
// if you are using a sprite, make sure that it has the same size (in pixels) with the size of the movie
action movie_display()
{
media_loop ("movie.wmv", bmap_for_entity (my, 0), 100); // use your own movie name
}
```

to get and display the xyz position of an object in a room, on which i am pointing with my mouse cursor.

```
BMAP* pointer_tga = "pointer.tga";
FONT* arial_font = "Arial#14";
PANEL* entity_pan = // displays the xyz coordinates of the entity
{
pos_x = 10;
pos_y = 10;
layer = 15;
digits(0, 10, "Entity x coordinate: %.f", arial_font, 1, mouse_ent.x);
digits(0, 30, "Entity y coordinate: %.f", arial_font, 1, mouse_ent.y);
digits(0, 50, "Entity z coordinate: %.f", arial_font, 1, mouse_ent.z);
flags = SHOW;
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

show an entity skill on a panel?

```
var enemy_skill1, enemy_skill2, enemy_skill3;
FONT* arial_font = "Arial#14";
ENTITY* my_enemy;
PANEL* entity_pan = // displays the xyz coordinates of the entity
{
pos_x = 10;
pos_y = 10;
layer = 15;
digits(0, 10, "Enemy skill1: %.f", arial_font, 1, enemy_skill1);
digits(0, 30, "Enemy skill2: %.f", arial_font, 1, enemy_skill2);
digits(0, 50, "Enemy skill3: %.f", arial_font, 1, enemy_skill3);
flags = SHOW;
}
action enemy_action()
{
my_enemy = my;
my_enemy.skill1 = 123;
enemy_skill1 = my_enemy.skill1;
my_enemy.skill2 = 456;
enemy_skill2 = my_enemy.skill2;
my_enemy.skill3 = 789;
enemy_skill3 = my_enemy.skill3;
}
```

scroll a text that's displayed above a panel

```
#include <strio.c> // need to include this library
var string_limit = 50; // don't display more than 50 characters from message_str at once
STRING* message_str = "This is a very long text that will be scrolled because its size exceeds the size of the panel";
STRING* temp_str = "";
STRING* output_str = "";
PANEL* hud_pan =
{
pos_x = 10;
pos_y = 10;
layer = 15;
bmap = "map.tga";
flags = SHOW;
}
TEXT* hud_txt =
{
pos_x = 20;
pos_y = 20;
layer = 20; // the text is displayed on top of hud_pan
string(output_str);
flags = SHOW;
}
function cut_startup()
{
var i = 0, j;
while (1)
{
str_cpy(temp_str, message_str);
j = i + string_limit;
str_cut(output_str, temp_str, i, j); // using a function from the strio.c library to cut the string
wait (-0.3);
i++;
j++;
if (i > str_len(message_str)) // the long string ended?
i = 0;
}
}
```

make a sprite appear on a key press? The sprite needs to fade in.

```
function display_sprite();
function create_sprites()
{
VECTOR sprite_coords;
sprite_coords.x = random(200) - 400; // the x of the sprite will be in the -200... 200 quants interval
sprite_coords.y = random(200) - 400; // the y of the sprite will be in the -200... 200 quants interval
sprite_coords.z = random(200) - 350; // the z of the sprite will be in the -150... 200 quants interval
ent_create ("heart2.tga", sprite_coords, display_sprite);
}
function display_sprite()
{
set(my, PASSABLE | TRANSLUCENT);
my.alpha = 0;
while (my.alpha < 100)
{
my.alpha += 7 * time_step; // 7 gives the fade-in speed
wait (1);
}
my.alpha = 100;
reset(my, TRANSLUCENT);
}
function sprites_startup()
{
on_c = create_sprites; // press the "C" key to create a new sprite
}
```

make a sphere model be filled with mist

```
BMAP* mist_tga = "mist.tga";
function fade_mist(PARTICLE *p)
{
p.alpha -= 0.5 * time_step; // fade out the particles
if (p.alpha < 0)
p.lifespan = 0;
}
function mist_function(PARTICLE *p)
{
p.alpha = 10 + random(20); // play with all these values
p->vel_x = (1 - random(2)) / 3; // play with all these values
p->vel_y = (1 - random(2)) / 3;
p->vel_z = (1 - random(2)) / 3;
p.bmap = mist_tga;
p.size = 5 + random(4); // sets the size of the mist particles
p.flags |= (BRIGHT | MOVE);
p.event = fade_mist;
}
action misty_sphere() // attach this action to your sphere model
{
set (my, PASSABLE | TRANSLUCENT); // make the sphere passable and transparent
my.alpha = 40; // play with this value
while (1)
{
effect(mist_function, 5, my.x, nullvector);
wait (1);
}
}
```

scan for dynamic lights

```
action players_code() // sample player code
{
var anim_percentage, distance_to_ground;
VECTOR movement_speed, temp;
player = my;
while (1)
{
c_scan(my.x, my.pan, vector(360, 180, 300), SCAN_ENTS | SCAN_LIMIT | IGNORE_ME); // scan for lights
camera.x = player.x - 250 * cos(player.pan);
camera.y = player.y - 250 * sin(player.pan);
camera.z = player.z + 150;
camera.pan = player.pan;
camera.tilt = -20;
my.pan += 6 * (key_a - key_d) * time_step;
vec_set (temp, my.x);
temp.z -= 5000;
distance_to_ground = c_trace (my.x, temp.x, IGNORE_ME | USE_BOX);
movement_speed.x = 5 * (key_w - key_s) * time_step;
movement_speed.y = 0;
movement_speed.z = - (distance_to_ground - 17); // 17 = experimental value
movement_speed.z = maxv (-35 * time_step, movement_speed.z); // 35 = falling speed
c_move (my, movement_speed.x, nullvector, GLIDE);
if (!key_w && !key_s)
{
ent_animate(my, "stand", anim_percentage, ANM_CYCLE);
}
else // the player is moving?
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
}
anim_percentage += 5 * time_step; // 5 = animation speed
wait (1);
}
}
function change_color()
```

```
{
vec_set(my.blue, vector(0, 0, 255)); // change to red light
}
action my_light()
{
vec_set(my.blue, vector(255, 255, 255)); // white light
my.emask = ENABLE_SCAN;
my.event = change_color;
while(1)
{
my.lightrange = 500; // set a light range of 500 quants
wait (1);
}
}
```

be able to set the colors for the cars in my racing game by clicking them, and the choosing one of the options from a menu

```
VECTOR* car_pos[3];
BMAP* pointer_tga = "pointer.tga";
BMAP* red1_pcx = "red1.pcx";
BMAP* red2_pcx = "red2.pcx";
BMAP* green1_pcx = "green1.pcx";
BMAP* green2_pcx = "green2.pcx";
BMAP* blue1_pcx = "blue1.pcx";
BMAP* blue2_pcx = "blue2.pcx";
ENTITY* active_car;
function set_red();
function set_green();
function set_blue();
PANEL* carcolor_pan =
{
bmap = "carcolor.pcx";
button = 0, 12, red2_pcx, red1_pcx, red2_pcx, set_red, NULL, NULL;
button = 0, 24, green2_pcx, green1_pcx, green2_pcx, set_green, NULL, NULL;
button = 0, 36, blue2_pcx, blue1_pcx, blue2_pcx, set_blue, NULL, NULL;
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function set_red()
{
active_car.red = 255;
active_car.green = 0;
active_car.blue = 0;
}
function set_green()
{
active_car.red = 0;
active_car.green = 255;
active_car.blue = 0;
}
function set_blue()
{
active_car.red = 0;
active_car.green = 0;
active_car.blue = 255;
}
function display_menu()
{
active_car = my;
vec_set(car_pos, my.x);
vec_to_screen(car_pos, camera);
carcolor_pan.pos_x = car_pos.x;
carcolor_pan.pos_y = car_pos.y;
carcolor_pan.flags |= SHOW;
}
action colored_car()
{
my.emask |= ENABLE_CLICK;
my.event = display_menu;
set (my, LIGHT);
}
```

ball that moves freely in the level and kills all the enemies that it is touching.

```
function ball_collides()
{
// the ball has collided with one of the entities in the level?
if (event_type == EVENT_ENTITY)
{
if (you.skill1 == 999) // collided with one of the enemies?
{
you.skill1 = 0; // then remove it!
}
}
// change the direction even if the ball has only collided with a level block
vec_to_angle(my.pan,bounce);
// add a slightly random value to the pan angle each time, in order to prevent the ball from getting stuck
my.pan += 10 - random(20);
}
action bouncing_ball()
{
my.emask |= (ENABLE_BLOCK | ENABLE_ENTITY); // make the entity sensitive to collisions with level blocks and entities
my.event = ball_collides;
while(1)
{
c_move(my, vector(10 * time_step, 0, 0),nullvector, NULL); // 10 gives the ball movement speed
wait(1);
}
}
action my_enemies()
{
my.skill1 = 999; // this value identifies an enemy
while (my.skill1 == 999) {wait (1);}
ent_remove (my);
}
```

a piece of code, so that when I close the game using Esc it saves the current date to a text file automatically

```
STRING* log_str = " "; // ex: August 25, 2011
STRING* temp_str = " "; // temporary string
function write_data()
{
var filehandle;
filehandle = file_open_write("date.txt");
if (sys_month == 1)
str_cpy(log_str, "January ");
if (sys_month == 2)
str_cpy(log_str, "February ");
if (sys_month == 3)
str_cpy(log_str, "March ");
if (sys_month == 4)
str_cpy(log_str, "April ");
if (sys_month == 5)
str_cpy(log_str, "May ");
if (sys_month == 6)
str_cpy(log_str, "June ");
if (sys_month == 7)
str_cpy(log_str, "July ");
if (sys_month == 8)
str_cpy(log_str, "August ");
if (sys_month == 9)
str_cpy(log_str, "September ");
if (sys_month == 10)
str_cpy(log_str, "October ");
if (sys_month == 11)
str_cpy(log_str, "November ");
if (sys_month == 12)
str_cpy(log_str, "December ");
if (sys_day < 10)
{
str_cat(log_str, "0");
}
str_for_num(temp_str, sys_day);
str_cat(log_str, temp_str);
str_cat(log_str, ", ");
str_for_num(temp_str, sys_year);
str_cat(log_str, temp_str);
file_str_write(filehandle, log_str);
file_close(filehandle);
wait (-1); // wait for a second
sys_exit(NULL); // shut down the engine
}
function init_startup()
{
on_esc = write_data;
}
```

pass a variable into an action like I would with a function

```
ENTITY* my_entity;
action tall_guy()
{
var height_data;
my_entity = my;
while (1)
{
height_data = my.skill99;
my.scale_z = 1 + my.skill99;
wait (1);
}
}
function init_startup()
{
while (!my_entity) {wait (1);}
while (1)
{
my_entity.skill99 = random(10); // pass the data using the 99th skill of the entity
wait (-2); // change the height of the entity every 2 seconds
}
}
```

a button that toggles my game from full screen to window mode and back

```
var toggled = 0;
BMAP* pointer_tga = "pointer.tga";
BMAP* picture1_pcx = "picture1.pcx";
BMAP* picture2_pcx = "picture2.pcx";
function toggle_fullscreen();
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
PANEL* time_pan =
{
// no need to use a bitmap for the panel this time
layer = 15;
button(40, 10, picture2_pcx, picture1_pcx, picture2_pcx, toggle_fullscreen, NULL, NULL);
flags = SHOW;
}
function toggle_fullscreen()
{
toggled +=1;
wait (3);
if (toggled % 2)
{
video_switch(0,0,2); // change to window mode
}
else
{
video_switch(0,0,1); // change to full screen mode
}
}
```

a simple way to trigger an event when a mouse button is released

```
var toggle_id;
function toggle_ambient() // runs each time the player presses the right mouse button
{
while (mouse_right) {wait (1);} // wait until the right mouse button is released
// the line of code below will be executed as soon as the player releases the right mouse button
toggle_id += 1;
if (toggle_id % 2)
{
camera.ambient = 100;
}
else
{
camera.ambient = -100;
}
}
function mouse_startup()
{
on_mouse_right = toggle_ambient;
}
```

an NPC that rotates towards the player at all times, rotating gently until it reaches the final position

```
action npc1()
{
VECTOR temp;
while (!player) {wait (1);}
// wait until the player model is loaded - your player action should include a player = my; line of code
while (player) // run this loop for as long as the player pointer exists
{
vec_set (temp.x, player.x);
vec_sub (temp.x, my.x);
vec_to_angle (my.skill99, temp);
// rotate the npc towards the player slowly - 0.05 gives the rotation speed
my.pan -= ang(my.pan - my.skill99) * 0.05 * time_step;
wait (1);
}
}
```

to play several sounds randomly, but I don't want them to overlap

```
SOUND* sound1_wav = "sound1.wav";
SOUND* sound2_wav = "sound2.wav";
SOUND* sound3_wav = "sound3.wav";
SOUND* sound4_wav = "sound4.wav";
SOUND* sound5_wav = "sound5.wav";
var soundhandle, sound_id;
function sounds_startup()
{
while (1)
{
sound_id = integer(random(5));
switch (sound_id)
{
case 0:
soundhandle = snd_play(sound1_wav, 80, 0);
break;
case 1:
soundhandle = snd_play(sound2_wav, 80, 0);
break;
case 2:
soundhandle = snd_play(sound3_wav, 80, 0);
break;
case 3:
soundhandle = snd_play(sound4_wav, 80, 0);
break;
case 4:
soundhandle = snd_play(sound5_wav, 80, 0);
break;
}
while (snd_playing(soundhandle)) {wait (1);}
wait (1);
}
}
```

congratulate the player at the end of the level, displaying a bunch of "congratulations" messages

```
function create_panels()
{
var digits_value, congrats_x, congrats_y, i;
for (i = 0; i < 10; i++)
{
congrats_x = 600 - random(300);
congrats_y = 200 + random(400);
PANEL* my_panel = pan_create(NULL,0);
FONT* my_font = font_create("Arial#20b");
pan_setstring(my_panel, 0, congrats_x, congrats_y, my_font, str_create("Congratulations!"));
set(my_panel,SHOW);
}
}
function panels_startup()
{
on_p = create_panels; // press the "P" key to create 10 congratulation panels at random positions on the screen
}
```

the player to be attacked only by the enemies that are closer to a certain distance to it

```
function scan_event()
{
if (event_type == EVENT_SCAN)
{
my.skill99 = 1;
}
}
action my_enemies() // attach this action to your enemies
{
var anim_percentage;
VECTOR temp;
my.emask |= ENABLE_SCAN;
my.event = scan_event;
while (1)
{
while (my.skill99 == 0) {wait (1);}
vec_set(temp.x, player.x);
vec_sub(temp.x, my.x);
vec_to_angle(my.pan, temp); // rotate the enemy towards the player
c_move(my, vector(2 * time_step, 0, 0), nullvector, GLIDE);
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 2 * time_step;
my.skill99 = 0;
wait (1);
}
}
action players_code() // sample player code
{
var anim_percentage, distance_to_ground;
VECTOR movement_speed, temp;
player = my;
while (1)
{
c_scan(my.x, my.pan, vector(360, 0, 300), SCAN_ENTS | SCAN_LIMIT | IGNORE_ME);
camera.x = player.x - 250 * cos(player.pan);
camera.y = player.y - 250 * sin(player.pan);
camera.z = player.z + 150;
camera.pan = player.pan;
camera.tilt = -20;
my.pan += 6 * (key_a - key_d) * time_step;
vec_set (temp, my.x);
temp.z -= 5000;
distance_to_ground = c_trace (my.x, temp.x, IGNORE_ME | USE_BOX);
movement_speed.x = 5 * (key_w - key_s) * time_step;
movement_speed.y = 0;
movement_speed.z = - (distance_to_ground - 17); // 17 = experimental value
movement_speed.z = maxv (-35 * time_step, movement_speed.z); // 35 = falling speed
c_move (my, movement_speed.x, nullvector, GLIDE);
if (!key_w && !key_s)
{
ent_animate(my, "stand", anim_percentage, ANM_CYCLE);
}
else // the player is moving?
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
}
anim_percentage += 5 * time_step; // 5 = animation speed
wait (1);
}
}
```

a snippet which reads the text file, switches the columns keeping the data intact, and then writes them to a new text file

```
var read_result = 1;
STRING* buffer1_str = "#200"; // the strings in the text file can have up to 200 characters each
STRING* buffer2_str = "#200"; // increase these values if you have longer strings
function process_data_startup()
{
var file_handle_read, file_handle_write;
file_handle_read = file_open_read("data.txt");
file_handle_write = file_open_write("output.txt"); // create the file if it doesn't exist
while (1) // reads the data until the end of the file is reached
{
// read the first string
read_result = file_str_read(file_handle_read, buffer1_str);
// read the second string
read_result = file_str_read(file_handle_read, buffer2_str);
// both strings were read here, so let's write them to the new file
// didn't read an empty string (the end of the file wasn't reached yet)? Then write the strings to output.txt!
if (read_result != -1)
{
file_str_write(file_handle_write, buffer2_str); // write the first string (was the second in its row)
file_str_write(file_handle_write, ","); // separate the consecutive strings with a comma
file_str_write(file_handle_write, buffer1_str); // write the second string (was the first in its row)
// move on to the following row in the output.txt file
file_asc_write (file_handle_write, 13); // write the following strings on a new line
file_asc_write (file_handle_write, 10); // using asc(13) = carriage return + asc(10) = line feed
}
else // reached the end of the file?
{
break; // then let's get out of this loop!
}
wait (1);
}
// the data reading and writing has finished here, so close the input and ouput files
file_close(file_handle_read); // close data.txt
file_close(file_handle_write); // close output.txt
}
```

the VIEW commands

```
VIEW* top_view =
{
// sets the camera layer value, allowing it to be placed on top of other views that have a lower layer value
layer = 10;
// sets the position of the view on the x axis of the screen
pos_x = 10;
// sets the position of the view on the y axis of the screen
pos_y = 10;
// sets the size of the view on the x axis of the screen
size_x = 128;
// sets the size of the view on the y axis of the screen
size_y = 256;
// sets the camera.arc value (zoom factor) of the view
arc = 45;
// set the ambient value of the view (bigger values will make the view brighter than the default camera view)
ambient = 100;
// displays the view
flags = SHOW;
}
function attached_startup() // attaches the view to the player, placing it 200 quants above it
{
// your player code must include this line of code: "player = my;" (without using the quotes)
while (!player) {wait (1);} // wait until the player model is loaded in the level
top_view.tilt = -90; // make the new view look downwards
while (1)
{
vec_set(top_view.x, player.x);
top_view.z += 200; // place the view 200 quants above the player
top_view.pan = player.pan; // rotate the view together with the player
wait (1);
}
}
```

an in-game menu that is toggled on / off by pressing the "T" key

```
var toggled = 0;
PANEL* main_pan =
{
bmap = "main.tga";
pos_x = 10;
pos_y = 20;
}
function toggle_menu_startup()
{
while (1)
{
if (key_t)
{
while (key_t) {wait (1);}
toggled += 1;
if (toggled % 2) // time to show the menu?
{
set (main_pan, SHOW); // then do it!
}
else // got to hide the menu?
{
reset (main_pan, SHOW); // then do it!
}
}
wait (1);
}
}
```

export from Gamestudio to Excel

```
var file_handle;
BMAP* pointer_tga = "pointer.tga";
STRING* product_str = "Input the product name";
STRING* quantity_str = "Input the quantity";
TEXT* product_txt =
{
pos_x = 30;
pos_y = 50;
layer = 20;
string(product_str);
}
TEXT* quantity_txt =
{
pos_x = 500;
pos_y = 50;
layer = 20;
string(quantity_str);
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function quit_input() // press the esc key to stop the program and save the data
{
file_close(file_handle); // close inventory.csv
sys_exit(NULL); // and then shut down the engine
}
function input_startup()
{
on_esc = quit_input;
wait (-5); // wait for 5 seconds
file_handle = file_open_write("inventory.csv"); // create the file if it doesn't exist
file_str_write(file_handle, "Products");
file_str_write(file_handle, ","); // separate the consecutive strings with a comma
file_str_write(file_handle, "Quantities");
file_asc_write (file_handle, 13); // write the following strings on a new line
file_asc_write (file_handle, 10); // using asc(13) = carriage return + asc(10) = line feed
// run this loop until the player types "quit" for the product name
while (1)
{
set(product_txt, SHOW);
wait(-3);
str_cpy(product_str, "#20"); // reset the string, allow up to 20 characters to be input for the product name
inkey(product_str);
file_str_write(file_handle, product_str); // write the product name to the file
file_str_write(file_handle, ","); // separate the consecutive strings with a comma
reset(product_txt, SHOW);
set(quantity_txt, SHOW);
wait(-3);
str_cpy(quantity_str, "#4"); // reset the string, allow up to 9,999 units (4 digits) to be input for the quantity
inkey(quantity_str);
file_str_write(file_handle, quantity_str); // write the product name to the file
file_asc_write (file_handle, 13); // write the following strings on a new line
file_asc_write (file_handle, 10); // using asc(13) = carriage return + asc(10) = line feed
reset(quantity_txt, SHOW);
}
}
```

1,234,567. Is it possible to show the number with its digits separated by commas

```
STRING* money_str = "#10";
STRING* temp_str = "#10";
STRING* copy_str = "#10";
TEXT* money_txt =
{
pos_x = 20;
pos_y = 20;
string(money_str); // displays the properly formatted string
flags = SHOW;
}
function money_startup()
{
var money = 1234567;
var length;
str_for_num(temp_str, money); // convert the number to a string
str_cpy(copy_str, temp_str);
length = str_len(temp_str); // count the number of digits
if (length <= 3) // no need to add commas here?
str_cpy(money_str, temp_str); // then copy the converted number to money_str directly!
if (length == 4) // got a 4 digit number here?
{
str_trunc(temp_str, 3); // remove the last 3 digits
str_cpy(money_str, temp_str); // copy the first digit to money_str;
str_cat(money_str, ","); // add a comma
str_cpy(temp_str, copy_str); // restore temp_str
str_clip(temp_str, 1); // remove the first digit
str_cat(money_str, temp_str); // add the last 3 digits
}
if (length == 5) // got a 5 digit number here?
{
str_trunc(temp_str, 3); // remove the last 3 digits
str_cpy(money_str, temp_str); // copy the first digit to money_str;
str_cat(money_str, ","); // add a comma
str_cpy(temp_str, copy_str); // restore temp_str
str_clip(temp_str, 2); // remove the first 2 digits
str_cat(money_str, temp_str); // add the last 3 digits
}
if (length == 6) // got a 6 digit number here?
{
str_trunc(temp_str, 3); // remove the last 3 digits
str_cpy(money_str, temp_str); // copy the first digit to money_str;
str_cat(money_str, ","); // add a comma
str_cpy(temp_str, copy_str); // restore temp_str
str_clip(temp_str, 3); // remove the first 3 digits
str_cat(money_str, temp_str); // add the last 3 digits
}
if (length == 7) // got a 7 digit number here?
{
str_trunc(temp_str, 6); // remove the last 6 digits
str_cpy(money_str, temp_str); // copy the first digit to money_str;
str_cat(money_str, ","); // add a comma
str_cpy(temp_str, copy_str); // restore temp_str
str_trunc(temp_str, 3); // remove the last 3 digits
str_clip(temp_str, 1); // remove the first digit
str_cat(money_str, temp_str); // add the following 3 digits
str_cat(money_str, ","); // add a comma
str_cpy(temp_str, copy_str); // restore temp_str
str_clip(temp_str, 4); // remove the first 4 digits
str_cat(money_str, temp_str); // add the last 3 digits
}
}
```

a simple algorithm for camera shaking

```
SOUND* explosion_wav = "explosion.wav";
function roll_startup()
{
var shaking_time = 3; // shake the camera for 3 seconds
while (!key_e) {wait (1);} // press the "E" key to start the explosion (camera shaking)
while (key_e) {wait (1);} // waut until the player releases the "E" key
snd_play (explosion_wav, 100, 0);
while (shaking_time > 0)
{
shaking_time -= time_step / 16; // subtract a value of 1 from shaking_time each second
camera.roll = 3 - random(6); // shake the camera by changing its roll angle, play with the numerical values
wait (1);
}
camera.roll = 0; // the explosion has ended here, so let's reset camera's roll angle
}
```

player reaches a certain point in the level, the game saves, and if the player dies, he starts again at the save point

```
var players_health = 100;
VECTOR player_position;
ANGLE player_angle;
action players_code() // attach this action to your player model
{
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
temp.z = 0;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50;
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
while (players_health <= 0) // the player is dead?
{
camera.roll = 70; // then let's make the camera look weird
camera.z = player.z - 30; // this while loop will prevent the player from moving
wait (1);
}
wait (1);
}
}
action checkpoint() // attach this to your autosave model (a portal, a gate, etc)
{
while (!player) {wait (1);} // wait until the player model is loaded in the level
// wait until the player comes closer than 200 quants to the autosave model
while (vec_dist (player.x, my.x) > 200) {wait (1);}
vec_set (player_position, player.x); // store player's position
vec_set (player_angle, player.pan); // store player's angles
// put a nice "Game Saved!" panel here, make it visible for a few seconds, and then remove it
}
function restore_startup() // restores player's health and angles
{
while (1)
{
while (players_health > 0) // wait until the player is dead
{
wait (1);
}
// now let's restore player's health
players_health = 100;
vec_set (player.x, player_position); // and its autosave position
vec_set (player.pan, player_angle); // and angles
camera.roll = 0; // restore camera's roll angle as well
wait (1);
}
}
function take_health()
{
if (you == player) // collided with the player?
players_health = 0; // then make it die!
}
action health_taker() // run into this entity to make the player lose its health
{
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY); // this entity is sensitive to impact with entities
my.event = take_health; // and runs this function when it is hit
}
```

check if a video is still being played

```
var movie_handle;
function movie_playing_startup()
{
movie_handle = media_play("mymovie.avi", NULL, 70);
while (media_playing (movie_handle)) {wait (1);}
beep(); beep(); // the movie has stopped here, so do what you need
}
```

ent_create to attach a flare to a key model, so that player can find the key easier in the level. Now I want to remove the flare as soon as the key model is removed

```
var got_key = 0;
function attach_flare()
{
set (my, PASSABLE);
while (you) // this loop will run for as long as the key model exists
{
my.x = you.x; // place the flare at the same position with the origin of the key model
wait (1);
}
ent_remove (my); // the key is gone now, so let's remove the flare sprite as well
}
action glowing_key()
{
set (my, PASSABLE);
ent_create ("flare.tga", my.x, attach_flare);
while (!player) {wait (1);}
while (vec_dist (player.x, my.x) > 70) // wait until the player comes close to the key
{
my.pan += 3 * time_step;
wait (1);
}
got_key = 1; // set the got_key variable, we will use it later in the game to open the corresponding door
ent_remove (my); // and then remove it
}
```

cars rotate (change their pan) in a car selection screen, but only do that while they are touched by the mouse

```
BMAP* arrow_pcx = "arrow.pcx";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = arrow_pcx;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function rotate_me()
{
proc_kill(1);
while (event_type != EVENT_RELEASE)
{
my.pan += 3 * time_step;
wait (1);
}
}
action rotating_car()
{
my.emask |= ENABLE_TOUCH | ENABLE_RELEASE;
my.event = rotate_me;
}
```

camera to follow the player at all times, changing to a camera that rotates around slowly, showing the nearby areas, if the player doesn't move at all for 10 seconds

```
action player_and_camera() // attach this action to your player model
{
var camera_time = 10; // the camera starts rotating around after 10 seconds
var camera_increments = 0;
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
while (1)
{
if (key_any) // the player has pressed one of the keys? Then let's reset camera_increments!
{
camera_increments = 0;
}
else // no key is being pressed? Then let's increase camera_increments!
{
camera_increments += time_step / 16; // add 1 to camera_increments each second
}
my.pan -= 7 * mouse_force.x * time_step;
temp.z = 0;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50;
if (camera_increments > 10) // the player didn't touch any of the keys for at least 10 seconds?
{
camera.pan += 0.8 * time_step;
}
else
{
camera.pan = my.pan;
}
camera.tilt += 5 * mouse_force.y * time_step;
wait (1);
}
}
```

a door that opens automatically (sliding vertically) when the player approaches it and closes automatically when the player has moved away from it

```
var trigger_range = 100;
SOUND* sliding_wav = "sliding.wav";
action my_door()
{
var initial_z;
// make sure to include a "player = my;" line of code to your player action
while (!player) {wait (1);}
initial_z = my.z;
while (1)
{
// wait until the player comes closer than 100 quants to the door, play with trigger_range
while (vec_dist (player.x, my.x) > 100) {wait (1);}
snd_play (sliding_wav, 100, 0);
// slide the door upwards by 200 quants, play with this value
while (my.z < (initial_z + 200))
{
my.z += 5 * time_step; // 5 gives the upwards sliding speed
wait (1);
}
// wait until the player moves away 500 quants from the door (5 * trigger_range)
while (vec_dist (player.x, my.x) < (5 * trigger_range)) {wait (1);}
snd_play (sliding_wav, 100, 0);
while (my.z > initial_z) // slide the door downwards, until it reaches the initial height
{
my.z -= 5 * time_step; // 5 gives the downwards sliding speed
wait (1);
}
// make sure that the door returns to the exact initial position regardless of the frame rate value
my.z = initial_z;
wait (1);
}
}
```

a weapon which fires bullets that don't disappear, but bounce from wall to wall indefinitely

```
var front_offset = 30; // animates player's weapon while it is firing by moving it backwards
var init_offset = 30; // stores front_offset's value
ENTITY* weapon1;
STRING* bullet_mdl = "bullet.mdl";
SOUND* bullet_wav = "bullet.wav";
function fire_bullets();
function move_bullets();
function redirect_bullets();
action players_weapon() // place your weapon model in the level and attach it this action
{
weapon1 = my; // I'm weapon1
VECTOR player1_pos; // stores the initial position of the player
VECTOR player2_pos; // stores the position of the player after a frame
VECTOR weapon_offset;
var weapon_height;
set (my, PASSABLE); // the weapon model is passable
while (!player) {wait (1);} // wait until the player is created
while (vec_dist (player.x, my.x) > 50) {wait (1);} // wait until the player comes close to pick up the weapon
my.roll = 0;
while (1)
{
vec_set (player1_pos.x, player.x); // store the initial player position
// place the weapon 30 quants in front, 18 quants to the right and 20 quants below camera's origin
vec_set (weapon_offset.x, vector (front_offset, -18, -20));
if (vec_dist (player1_pos.x, player2_pos.x) != 0) // the player has moved during the last frame?
{
```

```
weapon_height += 30 * time_step; // then offset weapon_height (30 = weapon waving speed)
weapon_offset.z += 0.3 * sin(weapon_height); // (0.3 = weapon waving amplitude)
}
// rotate weapon_offset according to the camera angles
vec_rotate (weapon_offset.x, vector (camera.pan, camera.tilt, 0));
vec_add (weapon_offset.x, camera.x); // add the camera position to weapon_offset
vec_set (my.x, weapon_offset.x); // set weapon's coords to weapon_offset
my.pan = camera.pan; // use the same camera angles for the weapon
my.tilt = camera.tilt;
vec_set (player2_pos.x, player.x); // store the new player coordinates after 1 frame
wait (1);
}
}
function weapon_startup()
{
on_mouse_left = fire_bullets; // call function fire_bullets() when the left mouse button is pressed
}
function fire_bullets() // includes the code that animates the weapon as well
{
var temp_seconds = 0;
VECTOR temp;
proc_kill(4); // don't allow more than 1 copy of this function to run
while (mouse_left) // this loop runs for as long as the left mouse button is pressed
{
front_offset -= 1; // move the weapon backwards a bit
vec_for_vertex (temp.x, weapon1, 29); // get the coordinates for the bullets' origin
// create the bullet at camera's position and attach it the "move_bullets" function
ent_create (bullet_mdl, temp.x, move_bullets);
temp_seconds = 0;
snd_play (bullet_wav, 100, 0); // play the bullet sound at a volume of 100
while (temp_seconds < 0.5)
{
temp_seconds += time_step / 16; // adds 0.5 to temp_seconds in 0.5 seconds)
front_offset -= 0.1; // restore the position of the weapon
wait (1);
}
while (front_offset < init_offset)
{
front_offset += 0.1; // restore the position of the weapon
wait (1);
}
front_offset = init_offset;
}
}
function move_bullets()
{
VECTOR bullet_speed; // stores the speed of the bullet
my.skill30 = 1; // I'm a bullet
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = redirect_bullets; // when it collides with something, its event function (redirect_bullets) will run
my.pan = camera.pan; // the bullet has the same pan
my.tilt = camera.tilt; // and tilt with the camera
bullet_speed.x = 200 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // then don't allow the gravity to have its ways with the bullet
while (my) // this loop will run for as long as the bullet exists (it isn't "null")
{
// move the bullet ignoring the passable entities and store the result in distance_covered
c_move (my, bullet_speed, nullvector, IGNORE_PASSABLE);
wait (1);
}
}
function redirect_bullets() // this function runs when the bullet collides with something
{
vec_to_angle(my.pan, bounce); // change the angle of the bullet
}
```

display a sphere made out of particles

```
BMAP* particle_tga = "particle.tga";
function keep_particle(PARTICLE *p)
{
p.lifespan = 99999; // use a big value here
}
function particle_effect(PARTICLE *p)
{
p.bmap = particle_tga;
p.size = 2; // gives the size of the particles
p.flags |= BRIGHT;
p.event = keep_particle;
}
action sphere() // attach this action to a sphere model
{
set (my, PASSABLE | INVISIBLE);
var particle_pos[3];
while (my.skill1 < ent_vertices (my))
{
my.skill1 += 1;
vec_for_vertex(particle_pos, my, my.skill1);
effect(particle_effect, 1, particle_pos, nullvector);
}
}
```

playing a soundtrack and after a specified interval fades it out, while at the same time fading in a new soundtrack

```
var track1_volume = 100;
var track2_volume = 1;
function music_startup()
{
var track1_handle, track2_handle;
track1_handle = media_play("track1.wav", NULL, track1_volume);
wait (-20); // play the first track for 20 seconds
track2_handle = media_play("track2.wav", NULL, track2_volume); // now start the second track at a tiny volume (1)
while (track1_volume > 0) // now change the volume - decrease it gently
{
media_tune(track1_handle, track1_volume, 100, 0);
media_tune(track2_handle, track2_volume, 100, 0);
track1_volume -= 0.6 * time_step; // 0.6 gives the fade-out speed
track1_volume = maxv(track1_volume, 0);
track2_volume += 0.8 * time_step; // 0.8 gives the fade-in speed
track2_volume = minv(track2_volume, 100);
wait (1);
}
wait (-20); // play the second track for 20 seconds
media_stop (track1_handle); // now stop the sound tracks for good
media_stop (track2_handle);
}
```

a simple AI which triggers an event when it is near a specific model or the player

```
// make sure that player's action includes the "player = my;" line of code
// set skill1 to 1234 for all the entities that should be detected (besides the player)
action enemy_detector()
{
while (1)
{
// scan for any entities that are closer than 200 quants to this entity
c_scan(my.x, my.pan, vector(360, 180, 200), IGNORE_ME | SCAN_ENTS);
if (you) // a target was detected?
{
if ((you == player) || (you.skill1 == 1234))
{
// an entity that verifies the conditions was detected, so do what you want here
my.scale_z = 3; // I'll just make this entity 3 times taller while it has detected something
}
}
else // didn't detect any entity here, so let's restore the initial scale
{
my.scale_z = 1;
}
wait (1);
}
}
```

a virtual dice. I got 6 pictures resembling the numbers of the dice; how can I show them in a random way for 5 seconds, and then stop at one picture

```
var dice_rolling = 0;
var rolling_time = 0;
var random_value;
PANEL* one_pan =
{
bmap = "01.tga";
layer = 15;
pos_x = 300;
pos_y = 200;
}
PANEL* two_pan =
{
bmap = "02.tga";
layer = 15;
pos_x = 300;
pos_y = 200;
}
PANEL* three_pan =
{
bmap = "03.tga";
layer = 15;
pos_x = 300;
pos_y = 200;
}
PANEL* four_pan =
{
bmap = "04.tga";
layer = 15;
pos_x = 300;
pos_y = 200;
}
PANEL* five_pan =
{
bmap = "05.tga";
layer = 15;
pos_x = 300;
pos_y = 200;
}
PANEL* six_pan =
{
```

```
bmap = "06.tga";
layer = 15;
pos_x = 300;
pos_y = 200;
}
function roll_dice()
{
if (dice_rolling) return; // don't allow the dice to roll again before ending the current roll
dice_rolling = 1;
rolling_time = 0;
while (rolling_time < 5) // this loop runs for 5 seconds
{
rolling_time += time_step / 1.6; // add a value of 1 to rolling_time each second
random_value = integer(random(6)) + 1; // generate an integer number that ranges from 1 to 6
if (random_value == 1)
{
set(one_pan, SHOW); // display the 01.tga bitmap
reset(two_pan, SHOW); // hide the other bitmaps
reset(three_pan, SHOW); // hide the other bitmaps
reset(four_pan, SHOW); // hide the other bitmaps
reset(five_pan, SHOW); // hide the other bitmaps
reset(six_pan, SHOW); // hide the other bitmaps
}
if (random_value == 2)
{
set(two_pan, SHOW); // display the 01.tga bitmap
reset(one_pan, SHOW); // hide the other bitmaps
reset(three_pan, SHOW); // hide the other bitmaps
reset(four_pan, SHOW); // hide the other bitmaps
reset(five_pan, SHOW); // hide the other bitmaps
reset(six_pan, SHOW); // hide the other bitmaps
}
if (random_value == 3)
{
set(three_pan, SHOW); // display the 01.tga bitmap
reset(two_pan, SHOW); // hide the other bitmaps
reset(one_pan, SHOW); // hide the other bitmaps
reset(four_pan, SHOW); // hide the other bitmaps
reset(five_pan, SHOW); // hide the other bitmaps
reset(six_pan, SHOW); // hide the other bitmaps
}
if (random_value == 4)
{
set(four_pan, SHOW); // display the 01.tga bitmap
reset(two_pan, SHOW); // hide the other bitmaps
reset(three_pan, SHOW); // hide the other bitmaps
reset(one_pan, SHOW); // hide the other bitmaps
reset(five_pan, SHOW); // hide the other bitmaps
reset(six_pan, SHOW); // hide the other bitmaps
}
if (random_value == 5)
{
set(five_pan, SHOW); // display the 01.tga bitmap
reset(two_pan, SHOW); // hide the other bitmaps
reset(three_pan, SHOW); // hide the other bitmaps
reset(four_pan, SHOW); // hide the other bitmaps
reset(one_pan, SHOW); // hide the other bitmaps
reset(six_pan, SHOW); // hide the other bitmaps
}
if (random_value == 6)
{
set(six_pan, SHOW); // display the 01.tga bitmap
reset(two_pan, SHOW); // hide the other bitmaps
reset(three_pan, SHOW); // hide the other bitmaps
reset(four_pan, SHOW); // hide the other bitmaps
reset(five_pan, SHOW); // hide the other bitmaps
reset(one_pan, SHOW); // hide the other bitmaps
}
wait (10); // generate 6 random numbers per second (this loop running at 60 frames per second, see below)
}
// the dice have rolled for 5 seconds here; the loop has stopped
```

```
dice_rolling = 0; // the dice can be rolled again
}
function dice_startup()
{
fps_max = 60; // limit the frame rate to 60
random_seed(0); // generate random number sequences each time
on_d = roll_dice; // press the "D" key to roll the dice
}
```

randomize questions in Gamestudio

```
BMAP* generate1_pcx = "generate1.pcx";
BMAP* generate2_pcx = "generate2.pcx";
BMAP* pointer_tga = "pointer.tga";
function generate_questions();
TEXT* question1_txt =
{
layer = 20;
pos_x = 200;
pos_y = 20;
string("Is this the 1st question?");
}
TEXT* question2_txt =
{
layer = 20;
pos_x = 200;
pos_y = 20;
string("Is this the 2nd question?");
}
TEXT* question3_txt =
{
layer = 20;
pos_x = 200;
pos_y = 20;
string("Is this the 3rd question?");
}
TEXT* question4_txt =
{
layer = 20;
pos_x = 200;
pos_y = 20;
string("Is this the 4th question?");
}
TEXT* question5_txt =
{
layer = 20;
pos_x = 200;
pos_y = 20;
string("Is this the 5th question?");
}
PANEL* questions_pan =
{
bmap = "mainpanel.pcx";
layer = 15;
button(40, 10, generate2_pcx, generate1_pcx, generate2_pcx, generate_questions, NULL, NULL);
flags = SHOW;
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function generate_questions()
{
var random_value;
random_value = integer(random(5)) + 1; // generate an integer number that ranges from 1 to 5
if (random_value == 1)
{
set(question1_txt, SHOW); // display the first question
reset(question2_txt, SHOW); // hide the other questions
reset(question3_txt, SHOW); // hide the other questions
reset(question4_txt, SHOW); // hide the other questions
reset(question5_txt, SHOW); // hide the other questions
}
if (random_value == 2)
{
set(question2_txt, SHOW); // display the 2nd question
reset(question1_txt, SHOW); // hide the other questions
reset(question3_txt, SHOW); // hide the other questions
reset(question4_txt, SHOW); // hide the other questions
reset(question5_txt, SHOW); // hide the other questions
}
if (random_value == 3)
{
set(question3_txt, SHOW); // display the 3rd question
reset(question1_txt, SHOW); // hide the other questions
reset(question2_txt, SHOW); // hide the other questions
reset(question4_txt, SHOW); // hide the other questions
reset(question5_txt, SHOW); // hide the other questions
}
if (random_value == 4)
{
set(question4_txt, SHOW); // display the 4th question
reset(question1_txt, SHOW); // hide the other questions
reset(question2_txt, SHOW); // hide the other questions
reset(question3_txt, SHOW); // hide the other questions
reset(question5_txt, SHOW); // hide the other questions
}
if (random_value == 5)
{
set(question5_txt, SHOW); // display the 5th question
reset(question1_txt, SHOW); // hide the other questions
reset(question2_txt, SHOW); // hide the other questions
reset(question3_txt, SHOW); // hide the other questions
reset(question4_txt, SHOW); // hide the other questions
}
}
```

prevent the camera to show the inside of my model, when i use a gun in the view like a FPS

```
ENTITY* actor_screen =
{
type = "ak47.mdl";
layer = 1;
flags2 = SHOW;
x = 30;
y = -3;
z = -5;
}
```

set health/hitpoints for enemies? Let's say there are 3 types of enemies in my map; I want all zombies to have 75 health, all mutants to have 100 health and all terrorists to have 125 health

```
STRING* bullet_mdl = "bullet.mdl";
function fire_bullets(); // creates the bullets
function remove_bullets(); // removes the bullets after they've hit something
function got_shot();
function move_bullets();
function move_enemy_bullets();
#define idle 1
#define attacking 2
#define dead 3
#define status skill10
#define health skill80
function fire_bullets()
{
proc_kill(4); // don't allow more than 1 instance of this function to run
while (mouse_left) {wait (1);} // wait until the player releases the mouse button
ent_create (bullet_mdl, camera.x, move_bullets); // create the bullet at camera's position and attach it the "move_bullets" function
}
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bullet
}
action my_enemy() // attach this action to your enemies
{
while (!player) {wait (1);}
var idle_percentage = 0;
var run_percentage = 0;
var death_percentage = 0;
VECTOR content_right; // tracks the content in front of the player
VECTOR content_left; // tracks the content in front of the player
VECTOR temp;
set (my, POLYGON); // use accurate collision detection
if (my.skill1 == 0)
{
my.health = 100; // set skill1 to the desired value (75, 100 or 125 points) or it will default to
}
else
{
my.health = my.skill1;
}
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY); // the enemy is sensitive to impact with player's bullets
my.event = got_shot; // and runs this function when it is hit
my.status = idle; // that's the same thing with my.skill10 = 1; (really!)
while (my.status != dead) // this loop will run for as long as my.skill10 isn't equal to 3
{
if (my.status == idle) // hanging around?
{
ent_animate(my, "stand", idle_percentage, ANM_CYCLE); // play the "stand" aka idle animation
idle_percentage += 3 * time_step; // "3" controls the animation speed
if (vec_dist (player.x, my.x) < 1000) // the player has come too close?
{
// scanned in the direction of the pan angle and detected the player?
if ((c_scan(my.x, my.pan, vector(120, 60, 1000), IGNORE_ME) > 0) && (you == player))
{
my.status = attacking; // then attack the player even if it hasn't fired at the enemy yet
}
}
}
if (my.status == attacking) // shooting at the player?
{
// the road is clear? then rotate the enemy towards the player
if (c_content (content_right.x, 0) + c_content (content_left.x, 0) == 2)
{
vec_set(temp, player.x);
vec_sub(temp,my.x);
vec_to_angle(my.pan, temp); // turn the enemy towards the player
}
if (vec_dist (player.x, my.x) > 500)
{
vec_set(content_right, vector(50, -20, -15));
vec_rotate(content_right, my.pan);
vec_add(content_right.x, my.x);
if (c_content (content_right.x, 0) != 1) // this area isn't clear?
{
my.pan += 5 * time_step; // then rotate the enemy, allowing it to avoid the obstacle
}
vec_set(content_left, vector(50, 20, -15));
vec_rotate(content_left, my.pan);
vec_add(content_left.x, my.x);
if (c_content (content_left.x, 0) != 1) // this area isn't clear?
{
my.pan -= 5 * time_step; // then rotate the enemy, allowing it to avoid the obstacle
}
c_move (my, vector(10 * time_step, 0, 0), nullvector, GLIDE);
ent_animate(my, "run", run_percentage, ANM_CYCLE); // play the "run" animation
run_percentage += 6 * time_step; // "6" controls the animation speed
}
else
{
ent_animate(my, "alert", 100, NULL); // use the last frame from the "alert" animation here
}
if ((total_frames % 80) == 1) // fire a bullet each second
{
vec_for_vertex (temp, my, 8);
// create the bullet at enemy's position and attach it the "move_enemy_bullets" function
ent_create (bullet_mdl, temp, move_enemy_bullets);
}
if (vec_dist (player.x, my.x) > 1500) // the player has moved far away from the enemy?
{
my.status = idle; // then switch to "idle"
}
}
wait (1);
}
while (death_percentage < 100) // the loop runs until the "death" animation percentage reaches 100%
{
ent_animate(my, "deatha", death_percentage, NULL); // play the "die" animation only once
death_percentage += 3 * time_step; // "3" controls the animation speed
wait (1);
}
set (my, PASSABLE); // allow the player to pass through the corpse now
}
function got_shot()
{
if (you.skill30 != 1) {return;} // didn't collide with a bullet? Then nothing should happen
my.health -= 35;
if (my.health <= 0) // dead?
{
my.status = dead; // stop the loop from "action my_enemy"
my.event = NULL; // the enemy is dead, so it should stop reacting to other bullets from now on
return; // end this function here
}
else // got shot but not (yet) dead?
{
my.status = attacking; // same thing with my.skill10 = 2
}
}
function move_enemy_bullets()
{
VECTOR bullet_speed; // this var will store the speed of the bullet
my.skill30 = 1; // I'm a bullet
// the bullet is sensitive to impact with other entities and to impact with level blocks
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = you.pan; // the bullet has the same pan
my.tilt = you.tilt; // and tilt with the enemy
bullet_speed.x = 50 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // or up / down on the z axis
// the loop will run for as long as the bullet exists (it isn't "null")
while (my)
{
// move the bullet ignoring its creator (the enemy)
c_move (my, bullet_speed, nullvector, IGNORE_YOU);
wait (1);
}
}
```

to stick an object to the camera so that it stays in front of a camera wherever I go

```
function stick_to_camera()
{
while (1)
{
// place the pot model 170 quants in front of the camera, 20 quants sideways and 25 quants below the origin on the z axis
vec_set (my.x, vector(170, 20, -25)); // play with these values
vec_rotate (my.x, camera.pan);
vec_add (my.x, camera.x);
my.pan = camera.pan;
my.tilt = camera.tilt;
wait (1);
}
}
function object_startup()
{
// make sure to include a "player = my;" line of code inside your player action / function
while (!player) {wait (1);}
ent_create("pot1.mdl", nullvector, stick_to_camera);
}
```

input box with "Cancel" and "OK" buttons

```
BMAP* pointer_tga = "pointer.tga";
BMAP* ok1_pcx = "ok1.pcx";
BMAP* ok2_pcx = "ok2.pcx";
BMAP* cancel1_pcx = "cancel1.pcx";
BMAP* cancel2_pcx = "cancel2.pcx";
STRING* name_str = "Click to input your name";
function input_name();
function cancel_name();
TEXT* name_txt =
{
pos_x = 300;
pos_y = 50;
layer = 20;
string(name_str);
flags = SHOW;
}
PANEL* input_pan =
{
bmap = "hud.tga";
pos_x = 280;
pos_y = 40;
layer = 10;
button(40, 10, ok2_pcx, ok1_pcx, ok2_pcx, input_name, NULL, NULL);
button(40, 50, cancel2_pcx, cancel1_pcx, cancel2_pcx, cancel_name, NULL, NULL);
flags = SHOW;
}
function input_name()
{
while (mouse_left) {wait (1);} // wait until the player releases the mouse button
str_cpy(name_str, "#50"); // reset the input string
inkey(name_str); // let's input player's name
reset (input_pan, SHOW); // and then let's hide the panel
}
function cancel_name()
{
printf("Name input was cancelled. System shutting down!");
sys_exit(NULL);
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

removes bitmaps from the video memory

```
BMAP* panel1_pcx = "panel1.pcx";
PANEL* gameover_pan =
{
layer = 15;
pos_x = 300;
pos_y = 200;
bmap = "panel1_pcx";
flags = SHOW;
}
function purge_startup()
{
while (!key_p) {wait (1);} // wait until the player presses the "P" key
while (key_p) {wait (1);} // wait until the player releases the "P" key
reset(gameover_pan, SHOW); // hide the panel
bmap_purge(panel1_pcx); // purge the bitmap used by the panel
}
```

a menu that changes its size automatically depending on the resolution of the resolution using scale_x, scale_y

```
BMAP* main_tga = "main.tga"; // main menu bitmap
BMAP* pointer_tga = "pointer.tga";
BMAP* ok1_pcx = "ok1.pcx";
BMAP* ok2_pcx = "ok2.pcx";
BMAP* cancel1_pcx = "cancel1.pcx";
BMAP* cancel2_pcx = "cancel2.pcx";
function start_game();
function exit_game();
PANEL* main_pan =
{
bmap = main_tga;
layer = 15;
button(40, 10, ok2_pcx, ok1_pcx, ok2_pcx, start_game, NULL, NULL);
button(40, 50, cancel2_pcx, cancel1_pcx, cancel2_pcx, exit_game, NULL, NULL);
flags = SHOW;
}
// makes the main menu panel fill the entire screen, regardless of the screen size and / or the bitmap resolution
function menu_startup()
{
while (1)
{
main_pan.scale_x = screen_size.x / bmap_width(main_tga);
main_pan.scale_y = screen_size.y / bmap_height(main_tga);
wait (1);
}
}
function start_game()
{
```

```
beep(); // start your game here
}
function exit_game()
{
sys_exit(NULL); // shut down the engine
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

interact with a panel that's placed on top of a patrolling entity

```
BMAP* pointer_tga = "pointer.tga";
PANEL* temp_pan;
function show_value();
action boss_entity() // attach this action to the boss
{
var anim_percentage;
VECTOR temp_pos;
temp_pan = pan_create("bmap = panel.pcx;", 10); // create a panel that uses the panel.pcx bitmap and has a layer of 10
temp_pan.scale_x = 5 + random(10); // set a random scale value for the enemy panel each time
temp_pan.flags |= SHOW; // and then make it visible
temp_pan.event = show_value; // this is the function that runs whenever the player clicks the panel
while (1)
{
c_move (my, vector(3 * time_step, 0, 0), nullvector, GLIDE); // "3" controls the walking speed
my.pan += 2 * time_step; // this line makes the entity walk in a circle
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 4 * time_step; // "4" controls the "walk" animation speed
vec_set (temp_pos, my.x); // copy the xyz coordinates of the entity to temp_pos
temp_pos.z += 60; // play with this value, it sets the distance between the entity's origin and the panel
vec_to_screen (temp_pos, camera); // convert the 3D position to 2D screen coordinates
temp_pan.pos_x = temp_pos.x; // then set the position of the panel
temp_pan.pos_y = temp_pos.y; // on x and y
wait (1);
}
}
function show_value() // open your menu here
{
beep();
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

to terminate all particle and shaders functions before changing the level

```
// set loading_new_level to 1 before loading a new level
var loading_new_level = 0;
action particle_generator()
{
while (!loading_new_level)
{
generate_particles_here();
wait (1);
}
// the particles won't be generated anymore here
}
action entity_shader()
{
my.material = my.superb_shader;
while (!loading_new_level) {wait (1);}
my.material = NULL;
}
```

smoke particles that are positioned

```
BMAP* smoke_tga = "smoke.tga";
ENTITY* car;
function fade_smoke(PARTICLE *p)
{
p.alpha -= 5 * time_step; // fade in the smoke particles
if (p.alpha < 0)
p.lifespan = 0;
}
function smoke_effect(PARTICLE *p)
{
p->vel_x = 0.5 - random(1);
p->vel_y = 0.5 - random(1);
p->vel_z = 0.5 - random(1);
p.alpha = 10 + random(20);
p.bmap = smoke_tga;
p.size = 2; // 2 gives the size of the smoke particle
p.flags |= (MOVE);
p.lifespan = 350; // lifespan for the smoke particles
p.event = fade_smoke;
}
function smoke_startup()
{
VECTOR temp;
while (!car) {wait (1);}
while (1)
{
vec_for_vertex (temp, car, 301); // generate smoke from the 301th vertex on the car (use your own number here)
// create more smoke particles on more powerful computers
effect (smoke_effect, 5, temp.x, normal); // generate 5 smoke particles each frame
wait (1);
}
}
action my_car()
{
car = my;
// put the rest of your car code here
}
```

several panels with buttons on them. When you click on a button, a new panel with buttons should appear. But after the first panel's buttons are clicked, the subsequent panel's buttons do not respond at all

```
BMAP* pointer_tga = "pointer.tga";
BMAP* panel1_pcx = "panel1.pcx";
BMAP* panel2_pcx = "panel2.pcx";
BMAP* newgame1_pcx = "newgame1.pcx";
BMAP* newgame2_pcx = "newgame2.pcx";
BMAP* quitgame1_pcx = "quitgame1.pcx";
BMAP* quitgame2_pcx = "quitgame2.pcx";
BMAP* start1_pcx = "start1.pcx";
BMAP* start2_pcx = "start2.pcx";
BMAP* previous1_pcx = "previous1.pcx";
BMAP* previous2_pcx = "previous2.pcx";
function second_panel();
function exit_game();
function init_game();
function first_panel();
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
PANEL* one_pan =
{
layer = 15;
pos_x = 100;
pos_y = 200;
bmap = panel1_pcx;
button(140, 110, newgame2_pcx, newgame1_pcx, newgame2_pcx, second_panel, NULL, NULL);
button(140, 160, quitgame2_pcx, quitgame1_pcx, quitgame2_pcx, exit_game, NULL, NULL);
flags = SHOW;
}
PANEL* two_pan =
{
layer = 15;
pos_x = 100;
pos_y = 200;
bmap = panel1_pcx;
button(140, 110, start2_pcx, start1_pcx, start2_pcx, init_game, NULL, NULL);
button(140, 160, previous2_pcx, previous1_pcx, previous2_pcx, first_panel, NULL, NULL);
}
function second_panel()
{
while (mouse_left) {wait (1);} // wait until the player releases the left mouse button
one_pan.flags &= ~SHOW;
two_pan.flags |= SHOW;
wait (1);
}
function exit_game()
{
sys_exit(NULL);
}
function init_game()
{
beep(); // initialize and start your game here
}
function first_panel()
{
while (mouse_left) {wait (1);} // wait until the player releases the left mouse button
two_pan.flags &= ~SHOW;
one_pan.flags |= SHOW;
}
```

a countdown timer that will turn into red if its near the end

```
var countdown_timer = 100; // this timer will run for 100 seconds
FONT* arial_font = "Arial#48b";
PANEL* timer_pan =
{
layer = 15;
digits(200, 120, 4 ,arial_font, 1, countdown_timer);
flags = SHOW;
}
function countdown_startup()
{
while (1)
{
countdown_timer -= time_step / 16;
countdown_timer = maxv(countdown_timer, 0); // don't allow the timer to go below zero
if (countdown_timer <= 10) // counting down the last 10 seconds? Then let's make the digit red!
vec_set(timer_pan.blue, vector(0, 0, 255)); // setting the blue, green, red vector color components
wait (1);
}
}
```

enemies change their colors to red when the player is coming very close to them

```
function init_startup() // start with a random sequence of numbers each time
{
random_seed(0);
}
function ball_event()
{
vec_to_angle (my.pan, bounce);
}
action bouncing_enemy()
{
while (!player) {wait (1);} // wait until the player model is loaded
my.pan = random(360); // each ball starts with a random pan angle
// make the ball sensitive to impacts with the level blocks or other entities
my.emask |= (ENABLE_BLOCK | ENABLE_IMPACT | ENABLE_ENTITY);
my.event = ball_event;
set(my, LIGHT); // allow this entity to change its color
my.green = 0;
my.blue = 0;
while(1)
{
c_move (my, vector(3 * time_step, 0, 0), nullvector, IGNORE_PASSABLE);
if (vec_dist(player.x, my.x) < 150) // this ball is really close to the player?
my.red = 255; // then let's make it change its color to red
else // the ball is away from the player
my.red = 0; // then let's restore its normal color
wait(1);
}
}
```

compare two objects based on their rank? I'd like the one that has a bigger score to win

```
ENTITY* barrel_black;
ENTITY* barrel_red;
action black_barrel()
{
barrel_black = my;
my.scale_z = 1;
while (my.scale_z < 10)
{
my.scale_z += 0.01 * (1 + random(2)) * time_step;
wait (1);
}
}
action red_barrel()
{
barrel_red = my;
my.scale_z = 1;
while (my.scale_z < 10)
{
my.scale_z += 0.01* (1 + random(2)) * time_step;
wait (1);
}
}
function check_startup() // checks to see which one of the barrels is the winner
{
random_seed(0); // start with a random sequence of numbers each time
wait (3); // wait until the level is loaded
while (1)
{
if (barrel_black.scale_z >= 10)
{
printf ("Black Barrel Wins!"); // display a message
sys_exit(NULL);
}
if (barrel_red.scale_z >= 10)
{
printf ("Red Barrel Wins!"); // display a message
sys_exit(NULL);
}
wait (1);
}
}
```

display an image when the player touches an object

display strings and numbers on a panel

```
FONT* arial_font = "Arial#20";
PANEL* display_pan =
{
pos_x = 100;
pos_y = 200;
layer = 15;
digits(10, 10, "Number of seconds: %.f", arial_font, 1, sys_seconds);
flags = visible;
}
```

2.5D sidescroller game and I'd like to have an enemy that shoots towards the player if it comes close enough to it

```
function move_bullet();
function remove_bullet();
function initialize_startup()
{
fps_max = 75; // limit the frame rate to 75 fps
```

```
}
action players_code() // simple player action
{
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
my.skill99 = 100; // the player starts with 100 health points
while (my.skill99 > 0) // this loop will run for as long as the player is alive
{
my.pan -= 7 * mouse_force.x * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50;
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
wait (1);
}
camera.roll = 20; // the player is dead here
}
action simple_enemy()
{
VECTOR temp;
while (!player) {wait (1);}
while (1)
{
// the player has come too close to the enemy
if (vec_dist (player.x, my.x) < 1000) // 1000 gives the enemy's range
{
vec_set(temp, player.x);
vec_sub(temp, my.x);
vec_to_angle(my.pan, temp); // rotate the enemy towards the player
// the enemy fires a bullet every 2 seconds (150 = 2 x 75 frames, the frame rate was limited to 75 fps)
if ((total_frames % 150) == 1)
{
ent_create("bullet.mdl", my.x, move_bullet);
}
}
wait (1);
}
}
function move_bullet()
{
VECTOR bullet_speed; // this var will store the speed of the bullet
my.skill30 = 1; // I'm a bullet
// the bullet is sensitive to impact with other entities and to impact with level blocks
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_bullet; // when it collides with something, its event function (remove_bullet) will run
my.pan = you.pan; // the bullet has the same pan
my.tilt = you.tilt; // and tilt with the enemy
bullet_speed.x = 50 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // or up / down on the z axis
// the loop will run for as long as the bullet exists (it isn't "null")
while (my)
{
// move the bullet ignoring its creator (the enemy)
c_move (my, bullet_speed, nullvector, IGNORE_YOU);
wait (1);
}
}
function remove_bullet() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
if (you) // collided with an entity?
```

```
{
if (you == player) // collided with the player? Then let's do some damage!
player.skill99 -= 10; // each shot takes 10 health points from the player
}
ent_remove (my); // and then remove the bullet
}
```

the best way to implement zoom in / out using the mouse scrolling wheel

```
// play with the values below - they depend on the actual size of your level
var min_offset_x = -1000; // the level starts at about -1000 quants
var max_offset_x = 1500; // and goes up to 1500 quants on the x axis in this example
var min_offset_y = -1200; // the level starts at about -1200 quants
var max_offset_y = 1300; // and goes up to 1300 quants on the y axis in this example
var horizontal_speed = 10; // horizontal scrolling speed
var vertical_speed = 8; // vertical scrolling speed
BMAP* pointer_tga = "pointer.tga";
function camera_startup()
{
wait (-1);// wait until the level is loaded
camera.tilt = -60; // make the camera look downwards
camera.pan = 90; // set the proper pan angle
camera.x = (min_offset_x + max_offset_x) / 2; // set the initial camera position close to the center of the level
camera.y = (min_offset_y + max_offset_y) / 2; // on both axis
mouse_mode = 2;
mouse_map = pointer_tga;
camera.arc = 90; // set the default zoom factor
while (1)
{
vec_set(mouse_pos, mouse_cursor);
if ((mouse_pos.x < 1) && (camera.x > min_offset_x))
camera.x -= horizontal_speed * time_step;
if ((mouse_pos.x > screen_size.x - 2) && (camera.x < max_offset_x))
camera.x += horizontal_speed * time_step;
if ((mouse_pos.y > screen_size.y - 2) && (camera.y > min_offset_y))
camera.y -= vertical_speed * time_step;
if ((mouse_pos.y < 1) && (camera.y < max_offset_y))
camera.y += vertical_speed * time_step;
if (mickey.z < -1) // the player has moved the mouse wheel because he / she wants to zoom in?
{
camera.arc -= 30;
}
if (mickey.z > 1) // the player has moved the mouse wheel because he / she wants to zoom out?
camera.arc += 30;
camera.arc = clamp(camera.arc, 30, 150); // each zoom step adds / subtracts 30 from camera.arc
wait (1);
}
}
```

create several models faster and faster

```
var number_of_models = 100; // generate 100 models
var pause_time = 200;
function move_ball()
{
while (my.z < 10000)
{
my.z += 15 * time_step;
wait (1);
}
ent_remove(my);
}
// attach this action to any model - it will become invisible
action model_generator()
{
set (my, PASSABLE | INVISIBLE);
wait (-5); // wait for 5 seconds
while (number_of_models > 0)
{
// use your own model file here
ent_create("ball.mdl", my.x, move_ball);
pause_time -= 15 * time_step;
pause_time = maxv(10, pause_time);
number_of_models -= 1;
wait (pause_time * time_step);
}
}
```

create models in random positions

```
var number_of_balls = 100; // generate 100 balls
VECTOR ball_position;
function moving_ball()
{
while (1)
{
my.z += 5 * sin(0.05 * total_ticks) * time_step;
wait (1);
}
}
function balls_startup()
{
wait (-3); // wait for 3 seconds
while (number_of_balls > 0)
{
ball_position.x = 400 - random(800);
ball_position.y = 500 - random(1000);
ball_position.z = 100;
// use your own model file here
ent_create("ball.mdl", ball_position.x, moving_ball);
number_of_balls -= 1;
wait (1);
}
}
```

an UFO hover above player's head, dropping bombs when it manages to get right above it.

```
function framerate_startup()
{
fps_max = 75; // limit the frame rate to 75 fps
}
function remove_bomb()
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bomb
}
function move_bomb()
{
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_bomb;
while (my)
{
// the bombs move downwards with the speed given by 10
c_move(my, vector(0, 0, -10 * time_step), nullvector, IGNORE_PASSABLE);
wait (1);
}
}
action evil_ufo()
{
while (!player) {wait (1);} // wait until the player model is loaded
while (1)
{
if (abs(player.x - my.x) > 1)
my.x += (player.x - my.x) * 0.03 * time_step; // 0.03 = tracking speed along the x axis
if (abs(player.y - my.y) > 1)
my.y += (player.y - my.y) * 0.03 * time_step; // 0.03 = tracking speed along the y axis
// the ufo is close enough to the player? Then let's drop some bombs!
if ((abs(player.x - my.x) < 50) && (abs(player.y - my.y) < 50))
{
if ((total_frames % 150) == 1) // drop a bomb every 2 seconds (75 fps x 2)
{
ent_create ("bomb.mdl", vector(my.x, my.y, my.z - 40), move_bomb);
}
}
wait(1);
}
}
```

separate VIEW* for my cut scenes. I'd like this view to activate and point towards an entity 3 seconds after the player has come close enough to it

```
VIEW* cutscene_camera =
{
pos_x = 0;
pos_y = 0;
layer = 10; // appears on top of the default camera view
}
function init_startup()
{
cutscene_camera.size_x = screen_size.x;
cutscene_camera.size_y = screen_size.y;
while (1)
{
wait (1);
}
}
action my_spaceship()
{
VECTOR temp;
while (!player) {wait (1);}
vec_set(temp, my.x);
vec_sub(temp, cutscene_camera.x);
vec_to_angle(cutscene_camera.pan, temp); // the cut scene camera was rotated towards the spaceship already, but it isn't visible yet
while (vec_dist (player.x, my.x) > 300) {wait (1);} // wait until the player comes close enough to the player
wait (-3); // wait for 3 more seconds
set(cutscene_camera, SHOW); // now display the cut scene camera
}
```

a car that has its wheels attached at runtime

```
STRING* wheel_mdl = "wheel.mdl";
function front_left_wheel()
{
VECTOR wheel_offset;
set (my, PASSABLE);
while (1)
{
vec_set(wheel_offset, vector(43, -20, -15));
vec_rotate(wheel_offset, you.pan);
vec_add(wheel_offset.x, you.x);
vec_set(my.x, wheel_offset.x);
my.pan = you.pan;
my.roll = you.roll;
wait (1);
}
}
function front_right_wheel()
{
VECTOR wheel_offset;
set (my, PASSABLE);
while (1)
{
vec_set(wheel_offset, vector(43, 20, -15));
vec_rotate(wheel_offset, you.pan);
vec_add(wheel_offset.x, you.x);
vec_set(my.x, wheel_offset.x);
my.pan = you.pan;
my.roll = you.roll;
wait (1);
}
}
function back_left_wheel()
{
VECTOR wheel_offset;
set (my, PASSABLE);
while (1)
{
vec_set(wheel_offset, vector(-32, -20, -15));
vec_rotate(wheel_offset, you.pan);
vec_add(wheel_offset.x, you.x);
vec_set(my.x, wheel_offset.x);
my.pan = you.pan;
my.roll = you.roll;
wait (1);
}
}
function back_right_wheel()
{
VECTOR wheel_offset;
set (my, PASSABLE);
while (1)
{
vec_set(wheel_offset, vector(-32, 20, -15));
vec_rotate(wheel_offset, you.pan);
vec_add(wheel_offset.x, you.x);
vec_set(my.x, wheel_offset.x);
my.pan = you.pan;
my.roll = you.roll;
wait (1);
}
}
action car_body() // attach this action to your car body model
{
my.ambient = -40;
ent_create(wheel_mdl, my.x, front_left_wheel); // create the front left wheel model
ent_create(wheel_mdl, my.x, front_right_wheel); // create the front right wheel model
ent_create(wheel_mdl, my.x, back_left_wheel); // create the back left wheel model
ent_create(wheel_mdl, my.x, back_right_wheel); // create the back right wheel model
while (1)
{
// simple snippet, moves the car in a circle
c_move (my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.pan += 2 * time_step; // 2 sets the radius of the rotation circle
wait (1);
}
}
```

remove a button from a panel after I've clicked it

```
BMAP* main_pcx = "main.pcx";
BMAP* button_pcx = "button.pcx";
BMAP* pointer_tga = "pointer.tga";
function remove_button();
PANEL* main_pan =
{
layer = 15;
pos_x = 300;
pos_y = 200;
bmap = main_pcx;
flags = SHOW;
}
PANEL* button_pan =
{
layer = 20;
pos_x = 320;
pos_y = 210;
bmap = button_pcx;
flags = SHOW;
on_click = remove_button;
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function remove_button()
{
while(mouse_left) {wait (1);}
reset(button_pan, SHOW);
beep(); // do your one-time operation here
}
```

a looping soundtrack in my main menu. When the player presses a button, the loop should fade out and the game should start

```
var loop_volume = 100;
var soundtrack_handle;
BMAP* main_pcx = "main.pcx";
BMAP* pointer_tga = "pointer.tga";
BMAP* button1_pcx = "button1.pcx";
BMAP* button2_pcx = "button2.pcx";
function fade_music();
PANEL* main_pan =
{
layer = 15;
pos_x = 300;
pos_y = 200;
bmap = main_pcx;
button(40, 10, button2_pcx, button2_pcx, button1_pcx, fade_music, NULL, NULL);
flags = SHOW;
}
function mouse_startup()
{
soundtrack_handle = media_loop("soundtrack.wav", NULL, loop_volume);
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
```

```
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function fade_music()
{
while (mouse_left) {wait (1);} // wait until the mouse button is released
while (loop_volume > 1)
{
media_tune(soundtrack_handle, loop_volume, 0, 0);
loop_volume -= 1.5 * time_step; // 1.5 = fading speed
wait (1);
}
snd_stop (soundtrack_handle);
mouse_mode = 0; // hide the mouse pointer
reset(main_pan, VISIBLE); // hide the main panel
// start your game here: load a level, etc
}
```

a rocket that increases its speed until it reaches a certain value. Then, it should stop doing that and continue to fly at its current speed

```
var front_offset = 30; // animates player's weapon while it is firing by moving it backwards
var init_offset = 30; // stores front_offset's value
var distance_covered; // global variable, so that we can display it using a digit on a panel
var got_weapon1 = 0;
ENTITY* weapon1;
STRING* bullet_mdl = "bullet.mdl";
SOUND* bullet_wav = "bullet.wav";
function fire_bullets();
function move_bullets();
function remove_bullets();
action players_weapon() // place your weapon model in the level and attach it this action
{
weapon1 = my; // I'm weapon1
VECTOR player1_pos; // stores the initial position of the player
VECTOR player2_pos; // stores the position of the player after a frame
VECTOR weapon_offset;
var weapon_height;
set (my, PASSABLE); // the weapon model is passable
while (!player) {wait (1);} // wait until the player is created
while (vec_dist (player.x, my.x) > 50) {wait (1);} // wait until the player comes close to pick up the weapon
got_weapon1 = 1;
my.roll = 0;
while (1)
{
vec_set (player1_pos.x, player.x); // store the initial player position
// place the weapon 30 quants in front, 18 quants to the right and 20 quants below camera's origin
vec_set (weapon_offset.x, vector (front_offset, -18, -20));
if (vec_dist (player1_pos.x, player2_pos.x) != 0) // the player has moved during the last frame?
{
weapon_height += 30 * time_step; // then offset weapon_height (30 = weapon waving speed)
weapon_offset.z += 0.3 * sin(weapon_height); // (0.3 = weapon waving amplitude)
}
// rotate weapon_offset according to the camera angles
vec_rotate (weapon_offset.x, vector (camera.pan, camera.tilt, 0));
vec_add (weapon_offset.x, camera.x); // add the camera position to weapon_offset
vec_set (my.x, weapon_offset.x); // set weapon's coords to weapon_offset
my.pan = camera.pan; // use the same camera angles for the weapon
my.tilt = camera.tilt;
vec_set (player2_pos.x, player.x); // store the new player coordinates after 1 frame
wait (1);
}
}
function weapon_startup()
{
on_mouse_left = fire_bullets; // call function fire_bullets() when the left mouse button is pressed
}
function fire_bullets() // includes the code that animates the weapon as well
{
if (!got_weapon1) return; // didn't get the weapon? Then don't allow the player to fire bullets!
var temp_seconds = 0;
VECTOR temp;
proc_kill(4); // don't allow more than 1 copy of this function to run
while (mouse_left) // this loop runs for as long as the left mouse button is pressed
{
front_offset -= 1; // move the weapon backwards a bit
vec_for_vertex (temp.x, weapon1, 29); // get the coordinates for the bullets' origin
// create the bullet at camera's position and attach it the "move_bullets" function
ent_create (bullet_mdl, temp.x, move_bullets);
temp_seconds = 0;
snd_play (bullet_wav, 100, 0); // play the bullet sound at a volume of 100
while (temp_seconds < 0.5)
{
temp_seconds += time_step / 16; // adds 0.5 to temp_seconds in 0.5 seconds)
front_offset -= 0.1; // restore the position of the weapon
wait (1);
}
while (front_offset < init_offset)
{
front_offset += 0.1; // restore the position of the weapon
wait (1);
}
front_offset = init_offset;
}
}
function move_bullets()
{
VECTOR bullet_speed; // stores the speed of the bullet
my.skill30 = 1; // I'm a bullet
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = camera.pan; // the bullet has the same pan
my.tilt = camera.tilt; // and tilt with the camera
bullet_speed.x = 2 * time_step; // set the initial speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // then don't allow the gravity to have its ways with the bullet
while (my) // this loop will run for as long as the bullet exists (it isn't "null")
{
if (distance_covered < 100 * time_step) // increase bullet's speed until it reaches 100 * time_step;
bullet_speed.x += 3 * time_step; // 3 gives the speed growth / frame
// move the bullet ignoring the passable entities and store the result in distance_covered
distance_covered = c_move (my, bullet_speed, nullvector, IGNORE_PASSABLE);
wait (1);
}
}
```

```
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bullet
}
PANEL* distance_pan =
{
layer = 15;
digits(600, 20, 4.3 ,* , 1, distance_covered);
flags = SHOW;
}
```

create a mini map of a level? I'd like to have a circular view that shows the player in the center of the mini map

```
VIEW* minimap_view;
ENTITY* minimap_model =
{
x = 100; // tweak these values until you set the desired position and size of the minimap view
y = -40;
z = 20;
tilt = -90;
type = "teler.mdl"; // use your own model here
layer = 20;
flags2 = SHOW; // show the model
}
function minimap_startup()
{
while (!minimap_model) {wait (1);}
minimap_view = view_create(10); // set the layer value for the view to 10
wait (2);
set (minimap_view, SHOW);
// the view will have 256x256 pixels; increase these values and the skin of the minimap model if you need a higher resolution
minimap_view.size_x = 256;
minimap_view.size_y = 256;
minimap_view.bmap = bmap_for_entity(minimap_model, 0);
minimap_view.arc = 90; // play with this value
vec_set(minimap_view.pan, vector(0, -90, 0)); // make the minimap view point downwards (tilt = -90)
while (1)
{
// set the position of the minimap 300 quants above the player - use a bigger value here if you'd like to see a bigger area around the player
vec_set(minimap_view.x, vector(player.x, player.y, player.z + 300));
wait (1);
}
}
```

two panels that stay in the left / right upper corners of the screen regardless of the video resolution

```
BMAP* left_pcx = "left.pcx";
BMAP* right_pcx = "right.pcx";
PANEL* left_pan =
{
bmap = left_pcx;
layer = 15;
pos_x = 0;
pos_y = 0;
flags = SHOW;
}
PANEL* right_pan =
{
bmap = right_pcx;
layer = 15;
flags = SHOW;
}
function panels_startup()
{
while (1)
{
right_pan.pos_x = screen_size.x - bmap_width(right_pcx);
wait (1);
}
}
```

make the player collide with the walls and not pass through them

```
action players_code() // attach this action to your player model
{
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
while (1)
{
// simple player movement code with gravity - use WSAD to move the player and the mouse to look around / rotate
my.pan -= 7 * mouse_force.x * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
// bonus - simple camera code! ;)
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50;
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
wait (1);
}
}
```

attach a weapon to one of my soldiers

```
STRING* soldiergun_mdl = "solgun.mdl";
function attach_weapon()
{
proc_mode = PROC_LATE;
set (my, PASSABLE);
while(you)
{
vec_set(my.x, you.x);
vec_set(my.pan, you.pan);
my.frame = you.frame;
my.next_frame = you.next_frame;
wait(1);
}
ent_remove(my);
}
action my_soldier()
{
var anim_percentage;
ent_create(soldiergun_mdl, nullvector, attach_weapon); // give the soldier a gun
while (1)
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 4 * time_step; // "1" controls the "stand" animation speed
c_move (my, vector(3 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.pan += 2 * time_step; // "2" controls the rotation speed
wait (1);
}
}
```

player leaving a trail behind everywhere he changes direction

```
BMAP* dust_tga = "dust.tga";
function fade_dust(PARTICLE *p)
{
p.alpha -= 2 * time_step; // dust fading speed = 2
if (p.alpha < 0) {p.lifespan = 0;}
}
function dust_particle(PARTICLE *p)
{
p->vel_x = 0; // slightly random
p->vel_y = 0; // horizontal speed
p->vel_z = random(1); // small vertical speed
p.alpha = 70; // initial transparency, play with this value
p.bmap = dust_tga;
p.size = 10;
p.flags |= (BRIGHT | MOVE);
p.event = fade_dust;
}
action players_code() // attach this action to your player model
{
var movement_speed = 4;
var anim_percentage;
VECTOR temp;
player = my;
while (1)
{
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_cuu - key_cud) * time_step;
my.pan -= movement_speed * (key_cur - key_cul) * 0.3 * time_step;
temp.y = 0;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 500;
camera.pan = my.pan;
camera.tilt = -90;
if (key_cuu + key_cud) // one of the movement keys is being pressed?
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 4 * time_step; // "1" controls the "stand" animation speed
if (key_cul + key_cur) // one of the rotation keys is being pressed?
effect (dust_particle, 3, vector(my.x, my.y, my.z - 30), normal); // generate dust particles 30 quants below the origin of the model
wait (1);
}
}
```

how to open a door if the player has got the security card

```
var got_card = 0;
SOUND* gotcard_wav = "gotcard.wav";
action my_card() // attach this action to your card model
{
set (my, PASSABLE);
while (!player) {wait (1);}
while (vec_dist (player.x, my.x) > 100) // wait until the player comes close to the card
{
my.pan += 4 * time_step; // rotate the card around its pan angle until the player gets it
wait (1);
}
snd_play(gotcard_wav, 60, 0);
set (my, INVISIBLE); // the player has picked up the card, so hide it
got_card = 1;
}
action my_door() // attach this action to your door entity
{
VECTOR door_coords;
vec_set (door_coords.x, my.x); // store the initial door position
while (!got_card) {wait (1);} // wait until the player picks up the card
while (vec_dist (player.x, my.x) > 150) {wait (1);} // wait until the player comes closer than 150 quants to the door
while (my.x < door_coords.x + 80) // 80 gives the sideway movement (door offset) - play with this value
{
my.x += 3 * time_step; // 3 gives the door sliding speed - play with it
wait (1);
}
}
```

panel fade in and out. I have a main menu and buttons; when I press a certain button, I'd like the main menu to fade out and another panel to fade in

```
var main_alpha = 100; // stores the transparency value for the "main" panel
var options_alpha = 0; // stores the transparency value for the "options" panel
BMAP* pointer_tga = "pointer.tga";
function new_game();
function exit_game();
PANEL* main_pan =
{
layer = 10;
bmap = "main.jpg";
pos_x = 0;
pos_y = 0;
button (250, 100, "newgame2.pcx", "newgame1.pcx", "newgame2.pcx", new_game, NULL, NULL);
button (250, 150, "exit2.pcx", "exit1.pcx", "exit2.pcx", exit_game, NULL, NULL);
alpha = 100;
flags = SHOW;
}
PANEL* options_pan =
{
layer = 15;
bmap = "options.pcx";
pos_x = 0;
pos_y = 0;
alpha = 0;
flags = TRANSLUCENT;
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function new_game()
{
main_pan.flags |= TRANSLUCENT; // the player has clicked the "options" button, so let's make the main panel transparent
options_pan.flags |= SHOW; // show the options panel
while (main_alpha > 5) // decrease main panel's alpha value, making it less and less visible
{
main_alpha -= 4 * time_step; // 4 gives main pan's fade out fading speed
main_pan.alpha = main_alpha;
options_alpha += 4 * time_step; // 4 gives option pan's fade in speed
options_pan.alpha = options_alpha;
wait (1);
}
main_pan.flags &= ~SHOW; // options_pan is visible, so let's hide the main panel
options_pan.alpha = 100; // and then let's make options_pan fully visible by setting its alpha value to 100
}
function exit_game()
{
sys_exit(NULL);
}
```

render a view on a target like a model, a texture or a bmap? I'd like to use that as a mirror, for example

```
ENTITY* mirrorsprite;
VIEW* mirror_view;
action mirror_sprite() // attach this action to a sprite / model that will be used for the mirror
{
mirrorsprite = my;
set (my, PASSABLE | DECAL);
}
function mirror_startup()
{
while (!mirror_sprite) {wait (1);}
mirror_view = view_create(10); // set the layer value for the new view to 10
wait (2);
set (mirror_view, SHOW);
mirror_view.size_x = 128;
mirror_view.size_y = 128; // the mirror sprite has 128x128 pixels
mirror_view.bmap = bmap_for_entity(mirrorsprite, 0);
vec_set(mirror_view.x, vector(1000, 500, 100)); // set the position of the mirror "eye"
vec_set(mirror_view.pan, vector(0, -10, 0)); // set the angles of the mirror "eye"
}
```

cut a view into a circle

```
VIEW* cutscene_view;
ENTITY* cutscene_model =
{
x = 100; // tweak these values until you set the desired position and size of the circular view
y = -40;
z = 20;
type = "teler.mdl"; // use your own model name here
layer = 20;
flags2 = SHOW; // show the model
}
function cut_scene_startup()
{
while (!cutscene_model) {wait (1);}
cutscene_view = view_create(10); // set the layer value for the view to 10
wait (2);
set (cutscene_view, SHOW);
// the view will have 256x256 pixels; increase these values and the skin of the model if you need a higher resolution
cutscene_view.size_x = 256;
cutscene_view.size_y = 256;
cutscene_view.bmap = bmap_for_entity(cutscene_model, 0);
cutscene_view.arc = 90; // play with this value
vec_set(cutscene_view.x, vector(30, 120, -175)); // set the position of the cutscene view "eye"
vec_set(cutscene_view.pan, vector(0, 10, 0)); // set the angles of the cutscene "eye"
}
```

display an image when my player touches an object

```
BMAP* pointer_tga = "pointer.tga";
PANEL* temp_pan;
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function tooltips_startup()
{
var panel_on = 0;
while (1)
{
if (mouse_ent)
{
// use different skill1 values and expand this piece of code if you want to display different images for different entities
if (mouse_ent.skill1 == 1)
{
if (panel_on == 0) // the image isn't visible already? Then let's create it!
{
panel_on = 1;
temp_pan = pan_create("bmap = circle.tga;", 10); // create a panel that uses the circle.tga bitmap and has a layer of 10
temp_pan.pos_x = 30; // set your desired pos_x and pos_y values for the panel
temp_pan.pos_y = 50;
temp_pan.flags |= SHOW; // and then make it visible
}
}
}
else // the mouse pointer didn't touch any entity?
{
if (panel_on == 1) // the panel was created? (no need to remove it more than once)
{
panel_on = 0; // reset panel_on
ptr_remove(temp_pan); // and then let's remove the panel
}
}
wait (1);
}
}
```

a text box in order to ask for player's name

```
BMAP* pointer_tga = "pointer.tga";
STRING* name_str = "Click to input your name";
function input_name();
TEXT* name_txt =
{
pos_x = 300;
pos_y = 50;
layer = 20;
string(name_str);
flags = SHOW;
}
PANEL* input_pan = // used only as a background for the text box
{
bmap = "hud.tga";
pos_x = 280;
pos_y = 40;
layer = 10;
on_click = input_name;
flags = SHOW;
}
function input_name()
{
while (mouse_left) {wait (1);} // wait until the player releases the mouse button
str_cpy(name_str, "#50"); // reset the input string
inkey(name_str); // let's input player's name
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

clouds using particles

```
BMAP* cloud_tga = "cloud.tga";
function fade_cloud(PARTICLE *p)
{
p.lifespan = 100; // comment this line to have the coulds disappear after a few seconds and
// p.alpha -= 0.1 * time_step; // remove the comments for these 2 lines to have the clouds fade away as the time passes
// if (p.alpha < 0) {p.lifespan = 0;}
}
function cloud_particle(PARTICLE *p)
{
p->vel_x = 0.9 * (1 - random(2)); // slightly random
p->vel_y = 0.8 * (1 - random(2)); // horizontal speed
p->vel_z = 0.5 * (1 - random(2)); // and vertical speed
p.alpha = 30 + random(40); // cloud transparency, play with this value
p.bmap = cloud_tga;
p.size = 150 + random(50); // slightly random cloud size
p.flags |= (BRIGHT | MOVE);
p.event = fade_cloud;
}
// Attach this action to several entities placed up high in the sky - they will become your invisible cloud generators
action cloud_particles()
{
set (my, INVISIBLE);
var i;
VECTOR temp;
for (i = 0; i < 100; i++) // generate 100 clouds for each entity
{
temp.x = my.x + 500 - random(1000);
temp.y = my.y + 500 - random(1000);
temp.z = my.z - 100 + random(200);
effect (cloud_particle, 1, temp.x, normal);
}
}
```

create a day / night sky system that changes in real time

```
ENTITY* my_sky;
function sky_startup()
{
my_sky = ent_createlayer("skycube+6.tga", SKY | CUBE | SHOW, 1);
while (1)
{
// allow the sky colors to change from 1 to 255
my_sky.red = 128 + 127 * sin(total_ticks * 0.003); // 0.003 gives the day / night transition speed
my_sky.green = my_sky.red;
my_sky.blue = my_sky.red;
wait (1);
}
}
```

check if a certain entity was found using c_scan

```
TEXT* normal_txt =
{
pos_x = 100;
pos_y = 30;
string("Normal entity detected!");
}
TEXT* special_txt =
{
pos_x = 300;
pos_y = 30;
string("Special entity detected!");
}
function got_scanned()
{
while (event_type == EVENT_SCAN)
{
if (my.skill99 == 1) // I'm the special entity?
{
set(special_txt, SHOW);
}
else // detected one of the regular entities?
{
set (normal_txt, SHOW);
}
wait (1);
}
reset (normal_txt, SHOW);
reset (special_txt, SHOW);
}
action my_regular_entities() // attach this action to your regular entities
{
my.emask |= ENABLE_SCAN; // make the entity sensitive to scanning
my.event = got_scanned;
}
action my_special_entity() // attach this action to your special entity
{
my.skill99 = 1; // this makes your entity special (different)
my.emask |= ENABLE_SCAN; // make the entity sensitive to scanning
my.event = got_scanned;
}
action players_code() // simple player / camera code, includes scanning code
{
player = my; // I'm the player
set (my, INVISIBLE); // no need to see player's model in 1st person mode
while (1)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, GLIDE);
vec_set (camera.x, player.x); // use player's x and y for the camera as well
camera.z += 30; // place the camera 30 quants above the player on the z axis (approximate eye level)
camera.pan -= 5 * mouse_force.x * time_step; // rotate the camera around by moving the mouse
camera.tilt += 3 * mouse_force.y * time_step; // on its x and y axis
player.pan = camera.pan; // the camera and the player have the same pan angle
// the following line makes the player scan up to 300 quants around it
c_scan(my.x, my.pan, vector(360, 180, 300), IGNORE_ME);
wait (1);
}
}
```

place a panel over an entity in my game

```
PANEL* temp_pan;
action panel_guy() // attach thi action to the entity that should display the panel above its head
{
wait (-3);
var anim_percentage;
VECTOR temp_pos;
temp_pan = pan_create("bmap = block.pcx;", 10); // create a panel that uses the block.pcx bitmap and has a layer of 10
temp_pan.flags |= SHOW; // and then make it visible
while (1)
{
c_move (my, vector(5 * time_step, 0, 0), nullvector, GLIDE); // "5" controls the walking speed
my.pan += 4 * time_step; // this line makes the entity walk in a circle
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 4 * time_step; // "4" controls the "walk" animation speed
vec_set (temp_pos, my.x); // copy the xyz coordinates of the entity to temp_pos
temp_pos.z += 60; // play with this value, sets the distance between the entity's origin and the panel
vec_to_screen (temp_pos, camera); // convert the 3D position to 2D screen coordinates
temp_pan.pos_x = temp_pos.x; // and then set the position of the text
temp_pan.pos_y = temp_pos.y; // on x and y
wait (1);
}
}
```

several panels with individual pointers and I need to change their position uniformly

```
PANEL* next_pan;
PANEL* panel_test =
{
layer = 15;
pos_x = 300;
pos_y = 200;
bmap = "test.tga";
flags = SHOW;
}
PANEL* panel_test2 =
{
layer = 15;
pos_x = 400;
pos_y = 250;
bmap = "test.tga";
flags = SHOW;
}
PANEL* panel_test3 = // feel free to add as many panels as you want
{
layer = 15;
pos_x = 500;
pos_y = 300;
bmap = "test.tga";
flags = SHOW;
}
// go through all the panels
function move_panels(offset_x, offset_y)
{
next_pan = ptr_first(panel_test);
while (next_pan)
{
next_pan.pos_x += offset_x;
next_pan.pos_y += offset_y;
next_pan = next_pan.link.next;
}
}
function move_startup()
{
while (1)
{
if (key_1)
{
while (key_1) {wait (1);}
move_panels(20, 40); // add 20 pixels on the x axis and 40 pixels on the y axis to all the panels
}
wait (1);
}
}
```

countdown timer that shuts down the game after 10 minutes of play

```
var time_left = 600; // allow the player to run the game for 600 seconds (10 minutes)
var minutes_left = 0;
var seconds_left = 0;
STRING* time_str = "#30";
STRING* temp_str = "#10";
TEXT* time_txt =
{
pos_x = 20;
pos_y = 20;
string(time_str);
flags = SHOW;
}
function countdown_startup()
{
while (time_left >= 0)
{
minutes_left = integer(time_left / 60);
seconds_left = time_left - minutes_left * 60;
str_cpy(time_str, "Time left to play: ");
str_for_num(temp_str, minutes_left);
str_cat(time_str, temp_str);
str_cat(time_str, ":");
str_for_num(temp_str, seconds_left);
if (seconds_left < 10)
str_cat(time_str, "0");
str_cat(time_str, temp_str);
time_left -= 1;
wait (-1);
}
sys_exit(NULL);
}
```

adjust the frame rate through a slider on a panel, thus increasing and decreasing the navigation speed in my level

```
var entity_speed = 3;
var movement_enabled = 0;
var dist_to_node;
var current_node = 1;
var angle_difference = 0;
VECTOR temp_angle;
VECTOR pos_node[3]; // stores the position of the node
BMAP* slider_tga = "slider.tga";
BMAP* pointer_tga = "pointer.tga";
PANEL* speed_pan =
{
bmap = "speed_panel.tga";
pos_x = 0;
pos_y = 0;
hslider(16, 24, 298, slider_tga, 0, 10, time_factor);
flags = SHOW;
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function move_target()
{
while(1)
{
if(movement_enabled)
{
entity_speed = minv(5, entity_speed + 0.5 * time_step);
c_move(my, vector(entity_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x));
}
wait(1);
}
}
action move_on_path() // attach this action to a model
{
move_target();
result = path_scan(me, my.x, my.pan, vector(360, 180, 1000));
if (result) {movement_enabled = 1;}
path_getnode (my, 1, pos_node, NULL);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate towards the node
while(1)
{
dist_to_node = vec_dist(my.x, pos_node);
if(dist_to_node < 50) // close to the node?
{
current_node = path_nextnode(my, current_node, 1);
if (!current_node) {current_node = 1;} // reached the end of the path? Then start over!
path_getnode (my, current_node, pos_node, NULL);
}
wait(1);
}
}
```

a door scan its surroundings and slide horizontally when the player approaches it

```
var slide_once = 0;
action sliding_door()
{
while (!player) {wait (1);}
while (1)
{
// the door scans on a range of 300 quants, trying to detect the "player" entity
if ((c_scan(my.x, my.pan, vector(360, 180, 300), IGNORE_ME) > 0) && (you == player))
{
slide_once += 1;
if (slide_once == 1) // do this only once
{
// this door slides along the x axis, feel free to use the y and z axis if they serve you better
my.skill90 = my.x;
while (my.x < my.skill90 + 100) // the door slides 100 quants along the x axis
{
my.x += 3 * time_step; // 3 gives the sliding speed
wait (1);
}
}
}
else // the player has moved away from the door here
{
slide_once = 0; // reset slide_once
while (my.x > my.skill90) // let's move the door back to its initial position
{
my.x -= 3 * time_step; // 3 gives the sliding speed
wait (1);
}
my.x = my.skill90; // make sure that the end x position is the same with the initial x position
}
wait (1);
}
}
action players_code() // simple player code snippet
{
var movement_speed = 10; // movement speed
VECTOR temp;
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2; // play with 2
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

a car selection screen, where the player uses two buttons to go through all the available car models

```
var car_type = 1; // showing the first car model at game start
BMAP* arrowleft_pcx = "arrowleft.pcx";
BMAP* arrowright_pcx = "arrowright.pcx";
BMAP* pointer_tga = "pointer.tga";
STRING* tinylevel_wmb = "test.wmb";
ENTITY* car;
function cars_left();
function cars_right();
function set_current_car();
PANEL* leftclick_pan =
{
bmap = "arrowleft.pcx";
pos_x = 0;
pos_y = 0;
layer = 15;
on_click = cars_left;
flags = SHOW;
}
PANEL* rightclick_pan =
{
bmap = "arrowright.pcx";
pos_x = 750;
pos_y = 0;
layer = 15;
on_click = cars_right;
flags = SHOW;
}
void main()
{
fps_max = 70;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
video_screen = 1; // start in full screen mode
level_load (tinylevel_wmb);
while (!car) {wait (1);} // wait until the car model is loaded
// choose a proper camera position that allows you to see the cars perfectly
vec_set (camera.x, vector(1000, 300, 200));
// and maybe set the camera pan, tilt and roll angles as well
vec_set (camera.pan, vector(100, -20, 5));
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function cars_left()
{
car_type -= 1;
car_type = maxv(1, car_type);
set_current_car();
}
function cars_right()
{
car_type += 1;
car_type = minv(3, car_type);
set_current_car();
}
function set_current_car()
{
if(car_type == 1)
ent_morph(car, "car1.mdl");
if(car_type == 2)
ent_morph(car, "car2.mdl");
if(car_type == 3)
ent_morph(car, "car3.mdl");
// add as many cars as you want here
}
// attach this action to the car1.mdl model (the default car)
action my_cars()
{
car = my;
while (1)
{
my.pan += 3 * time_step;
wait (1);
}
}
```

use the string1 and string2 strings from within Wed

```
BMAP* pointer_tga = "pointer.tga";
STRING* info_str = "#32";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
TEXT* info_txt =
{
pos_x = 50;
pos_y = 20;
string (info_str);
flags = SHOW;
}
function strings_startup()
{
while (1)
{
if (mouse_ent)
{
if (mouse_left)
str_cpy(info_str, mouse_ent.string1); // put your text in entity's string1 in Wed
if (mouse_right)
str_cpy(info_str, mouse_ent.string2); // put your text in entity's string1 in Wed
set(info_txt, SHOW);
}
else
{
reset(info_txt, SHOW);
}
wait (1);
}
}
```

the easiest way to make drop shadows

```
#include <acknex.h>
#include <default.c>
#include <mtlView.c> // contains the stencil_blur() function
STRING* test_wmb = "test.wmb";
function main()
{
camera.ambient = 70;
fps_max = 70;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
video_screen = 1; // start in full screen mode
level_load (test_wmb);
wait (2);
ent_createlayer("skycube+6.tga", SKY | CUBE | SHOW, 1);
shadow_stencil = 4; // use values in the 0...4 range
shadow_lod = 2; // use the second LOD stage for stencil shadows
stencil_blur(1); // activate blurred shadows (included in mtlView.c, look better)
}
action shadow_model() // simple action that rotates this model in a circle
{
set(my, SHADOW);
while (1)
{
```

```
c_move (my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.pan += 2 * time_step; // 2 sets the radius of the circle
wait (1);
}
}
```

make the number pad keys (2, 4, 6, 8) control player's movement

```
action players_code() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44);
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_pressed(72) - key_pressed(80)) * time_step;
temp.y = movement_speed * (key_pressed(75) - key_pressed(77)) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

create a health bar for my player character

```
var players_health;
PANEL* health_pan =
{
bmap = "redbar.pcx"; // a bitmap with x = 2, y = 12 pixels
pos_x = -10;
pos_y = 0;
layer = 10;
flags = SHOW;
}
function health_startup()
{
while (1)
{
health_pan.scale_x = maxv(0.001, players_health * 3.5); // 3.5 = experimental value
wait (1);
}
}
action players_code() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
players_health = 100; // the player starts with 100 health points
while (players_health > 0)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44);
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, GLIDE);
wait (1);
}
// the player is dead here
camera.z -= 20;
camera.roll = -35;
}
action health_taker()
{
while (!player) {wait (1);}
while (1)
{
if (vec_dist(player.x, my.x) < 200)
players_health -= 2 * time_step;
wait (1);
}
}
```

<u>a camera that can rotate around the player whenever I want it to</u>

```
function camera_startup()
{
var orbit_radius = 200; // use your own value here
var orbit_speed = 4; // and here
var temp_angle;
VECTOR orbit_center, temp;
while (!player) {wait (1);}
vec_set (orbit_center.x, player.x); // store the orbit center position
while(1)
{
camera.x = orbit_center.x + sin(temp_angle) * orbit_radius;
camera.y = orbit_center.y + cos(temp_angle) * orbit_radius;
camera.z = orbit_center.z + 50; // play with 50
camera.tilt = -20; // play with -20
// use the "O" (not zero!) and "P" keys to rotate the camera around the player
temp_angle += 1 * (key_o - key_p) * orbit_speed * time_step; // 1 gives the camera rotation speed
vec_set(temp, player.x);
vec_sub(temp, camera.x);
vec_to_angle(camera.pan, temp); // make the camera look at the player all the time
wait(1);
}
}
action players_code() // attach this action to your player
```

```
{
var movement_speed = 20;
VECTOR temp;
player = my;
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) + 20; // 20 gives the distance to ground
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

a particle under a moving entity

```
BMAP* fire_tga = "fire.tga";
function fade_jet(PARTICLE *p)
{
p.alpha -= 40 * time_step; // fade out the fire particles, play with this value
if (p.alpha < 0)
p.lifespan = 0;
}
function jet_effect(PARTICLE *p)
{
p->vel_x = 1 - random(2);
p->vel_y = 1 - random(2);
p->vel_z = 1 + random(1);
p.lifespan = 10; // play with this value
p.alpha = 25 + random(50);
p.bmap = fire_tga;
p.size = 25; // gives the size of the fire particles
p.flags |= (BRIGHT | MOVE);
p.event = fade_jet;
}
action my_player() // dummy player action
{
VECTOR jet_offset;
while (1)
{
// put your own player code here
// the example below simply makes the player rotate in a circle
c_move (my, vector(20 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.pan += 2 * time_step; // 2 sets the radius of the circle
// place the particle jet at the proper position, play with the offset values below
vec_set(jet_offset.x, vector(-100, -10, -20));
vec_rotate(jet_offset.x, my.pan);
vec_add(jet_offset.x, my.x);
effect(jet_effect, 1, jet_offset.x, nullvector); // generate a jet particles per frame
wait (1);
}
}
```

code for a physics car, but with a camera that follows the car

```
#include <acknex.h>
#include <default.c>
//vars, vectors, entities and angles
var constr_front_left;
var constr_front_right;
var constr_back_left;
var constr_back_right;
VECTOR mom_speed;
var max_angle;
var stear_contr;
var max_speed = 1500;
var ang_force;
var speed_contr;
VECTOR* vec_gravity = {x = 0; y = 0; z = -1800;}
ENTITY* chassis;
function wheel_physics_init()
{
set (my, SHADOW);
phent_settype (my, PH_RIGID, PH_SPHERE);
phent_setmass (my, 30, PH_SPHERE);
phent_setgroup (my, 2);
phent_setfriction (my, 100);
phent_setdamping (my, 20, 20);
phent_setelasticity (my, 0, 100);
}
// front wheels
function front_tyre_left()
{
wheel_physics_init();
constr_front_left = phcon_add (PH_WHEEL, chassis, my);
phcon_setparams1 (constr_front_left, my.x, vector (0,0,1), nullvector);
phcon_setparams2 (constr_front_left, nullvector, nullvector, nullvector);
}
function front_tyre_right()
{
wheel_physics_init();
constr_front_right = phcon_add (PH_WHEEL, chassis, my);
phcon_setparams1 (constr_front_right, my.x, vector (0,0,1), nullvector);
phcon_setparams2 (constr_front_right, nullvector, nullvector, nullvector);
}
// rear wheels
function back_tyre_left()
{
wheel_physics_init();
constr_back_left = phcon_add (PH_WHEEL, chassis, my);
phcon_setparams1 (constr_back_left, my.x, nullvector, vector (0,1,0));
phcon_setparams2 (constr_back_left, nullvector, nullvector, nullvector);
}
function back_tyre_right()
{
wheel_physics_init();
constr_back_right = phcon_add (PH_WHEEL, chassis, my);
phcon_setparams1 (constr_back_right, my.x, nullvector, vector (0,1,0));
phcon_setparams2 (constr_back_right, nullvector, nullvector, nullvector);
}
function chassis_init()
{
chassis = my;
set (chassis, SHADOW);
//init physics variables
phent_settype (chassis, PH_RIGID, PH_BOX); // rigid and box as collision hull
phent_setmass (chassis, 15, PH_BOX); // lighter than wheels to avoid tilt
phent_setgroup (chassis, 2); // all parts of the car in one group --> no collision
phent_setfriction (chassis, 20);
phent_setdamping (chassis, 5, 5);
phent_setelasticity (chassis, 10, 100); // only little bouncing
//init car wheels
//back
ent_create ("car_tyre_left.mdl", vector (chassis.x - 65, chassis.y - 45, chassis.z - 20), back_tyre_left);
ent_create ("car_tyre_right.mdl", vector (chassis.x - 65, chassis.y + 45, chassis.z - 20), back_tyre_right);
//front
ent_create ("car_tyre_left.mdl", vector (chassis.x + 65, chassis.y - 45, chassis.z - 20), front_tyre_left);
ent_create ("car_tyre_right.mdl", vector (chassis.x + 65, chassis.y + 45, chassis.z - 20), front_tyre_right);
wait(1);
}
void main()
{
level_load ("test.wmb");
//initialize the physics sub-system:
ph_setgravity (vec_gravity);
ph_fps_max_lock = 300;
//init chassis
ent_create ("chassis.mdl", vector (0,0, 100), chassis_init);
while (!chassis) {wait (1);} // wait until the chassis appears in the level
// speed and steering control
while (1)
{
// steering control
stear_contr = (key_d - key_a);//d = 1, a = -1, a & d = 0
max_angle += stear_contr * 3 * time_step;
max_angle = clamp (max_angle, -30, 30);
if (stear_contr != 0)
{
phcon_setparams2 (constr_front_left, vector (max_angle, max_angle, 0), nullvector, vector (95000, 500, 0));
phcon_setparams2 (constr_front_right, vector (max_angle, max_angle, 0), nullvector, vector (95000, 500, 0));
}
else
{
max_angle = max_angle * (1 - (0.25 * time_step));
if (abs(max_angle) < 1)
max_angle = 0;
phcon_setparams2 (constr_front_left, vector (max_angle, max_angle, 0), nullvector, vector (95000, 500, 0));
phcon_setparams2 (constr_front_right, vector (max_angle, max_angle, 0), nullvector, vector (95000, 500, 0));
}
// speed control
speed_contr = (key_w - key_s);//w = 1, s = -1, w & s = 0
ang_force = speed_contr * 1000 * time_step;
phcon_setmotor (constr_back_left, nullvector, vector(-ang_force, max_speed, 0), nullvector);
phcon_setmotor (constr_back_left, nullvector, vector(-ang_force, max_speed, 0), nullvector);
camera.x = chassis.x - 250 * cos(chassis.pan);
camera.y = chassis.y - 250 * sin(chassis.pan);
camera.pan = chassis.pan; // the camera and the car have the same pan angle
camera.z = chassis.z + 50; // place the camera above the car, play with this value
camera.tilt = -15; // look downwards
wait(1);
}
}
```

a simple bike that moves and lean as it turns

```
action my_bike() // attach this action to your bike model
{
var movement_speed = 0; // initial movement speed
var rotation_speed = 3; // rotation speed
VECTOR bike_speed, temp;
player = my;
while (1)
{
// change the pan angle of the bike if the player presses the left / right cursor keys
my.pan += rotation_speed * (key_cul - key_cur) * time_step;
// also change the roll angle of the bike if the player presses left / right
if (key_cul) // the player has pressed the left cursor key?
{
if (my.roll > -20)
{
my.roll -= 5 * time_step;
}
}
else
{
if (my.roll < 0)
{
my.roll += 5 * time_step;
}
}
if (key_cur) // the player has pressed the right cursor key?
{
if (my.roll < 20)
{
my.roll += 5 * time_step;
}
}
else
{
if (my.roll > 0)
{
my.roll -= 5 * time_step;
}
}
// 15 gives the acceleration, 0.1 gives the friction
vec_set(bike_speed.x, accelerate (movement_speed, 15 * (key_cuu - key_cud), 0.1));
bike_speed.y = 0;
vec_set (temp.x, my.x);
temp.z -= 10000;
// 12 gives the distance to the ground, play with it
bike_speed.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) + 12;
c_move (my, bike_speed.x, nullvector, IGNORE_PASSABLE | GLIDE);
camera.x = my.x - 250 * cos(my.pan); // put the camera 250 quants behind the bike
camera.y = my.y - 250 * sin(my.pan);
camera.pan = my.pan; // the camera and the bike have the same pan angle
camera.z = my.z + 70; // place the camera above the bike, play with this value
camera.tilt = -10; // look downwards
wait (1);
}
}
```

retrieve the distance between a ball and the floor and have a continuous update of its distance displayed on my screen. The ball is moving up and down

```
var distance_to_ground = 0;
FONT* arial_font = "Arial#20b";
ENTITY* ball;
action my_ball() // simple ball action
{
ball = my; // for further use
VECTOR temp;
while (1)
{
my.z += 0.3 * sin(total_ticks);
vec_set (temp.x, my.x);
temp.z -= 10000;
distance_to_ground = c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
wait (1);
}
}
PANEL* info_pan =
{
layer = 15;
digits(30, 40, "Distance to ground: %.f", arial_font, 1, distance_to_ground);
flags = SHOW;
}
```

a sprite attached to a model

```
var entity_speed = 3;
var movement_enabled = 0;
var dist_to_node;
var current_node = 1;
VECTOR temp_angle;
VECTOR pos_node; // stores the position of the node
VECTOR temp;
function attach_sprite() // attaches a sprite to guard's left thumb
{
proc_mode = PROC_LATE; // this is the key instruction - it eliminates the lagging
// the sprite must be made passable; otherwise it would hinder the movement of the guard
set (my, PASSABLE | BRIGHT); // no need to make the sprite bright if you don't want to
my.scale_x = 0.2; // set the scale of the sprite according to your needs
my.scale_y = my.scale_x;
while (1)
{
vec_set(my.x, temp.x); // get the updated coordinates of guard's left thumb each frame
wait (1);
}
}
function move_target()
{
var stand_percentage, walk_percentage;
ent_create("muzzle.tga", my.x, attach_sprite); // create the sprite at player's origin initially
while(1)
{
if(movement_enabled)
{
entity_speed = minv(5, entity_speed + 0.5 * time_step);
c_move(my, vector(entity_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
walk_percentage += 4 * time_step;
ent_animate(my, "walk", walk_percentage, ANM_CYCLE); // play the "walk" animation
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x));
}
else // standing still? (no path could be found)
{
stand_percentage += 2 * time_step;
ent_animate(my, "stand", stand_percentage, ANM_CYCLE); // play the "stand" animation
}
vec_for_vertex (temp, my, 500); // get the coordinates of guards 500th vertex (its left thumb)
wait(1);
}
}
action guard_with_sprite() // attach this action to the guard model
{
move_target();
result = path_scan(me, my.x, my.pan, vector(360, 0, 500)); // scan the area
if (result) {movement_enabled = 1;}
path_getnode (my, 1, pos_node, NULL);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate towards the node
while(1)
{
dist_to_node = vec_dist(my.x, pos_node);
if(dist_to_node < 50) // close to the node?
{
current_node = path_nextnode(my, current_node, 1);
if (!current_node) {current_node = 1;} // reached the end of the path? Then start over!
path_getnode (my, current_node, pos_node, NULL);
}
wait(1);
}
}
```

a rocket that appears and moves towards an enemy

```
#include <acknex.h>
#include <default.c>
SOUND* rocket_wav = "rocket.wav";
SOUND* destroyed_wav = "destroyed.wav";
STRING* heatseek_wmb = "heatseek.wmb";
STRING* rocket_mdl = "rocket.mdl";
function fire_rocket();
function shoot_rocket();
function remove_rocket();
function enemy_event();
function main()
{
fps_max = 70;
camera.ambient = 100;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
video_screen = 1; // start in full screen mode
level_load (heatseek_wmb);
wait (3);
on_mouse_left = fire_rocket;
}
action player_moves() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44);
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
action enemy() // attach this action to your enemies (they need to be sensitive to scanning)
{
my.emask |= (ENABLE_SCAN);
my.event = enemy_event;
}
function fire_rocket()
{
ent_create (rocket_mdl, player.x, shoot_rocket);
snd_play (rocket_wav, 50, 0);
}
function shoot_rocket()
{
VECTOR rocket_speed, temp;
my.emask |= (ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_rocket;
set (my, PASSABLE);
my.pan = camera.pan;
my.tilt = camera.tilt;
my.skill20 = 0;
vec_set(rocket_speed.x, nullvector);
rocket_speed.x = 10 * time_step;
my.skill10 = 0;
while (my.skill20 < 500)
{
if (vec_dist (player.x, my.x) > 200)
reset (my, PASSABLE);
if ((my.skill10 == 0) && (vec_dist (my.x, player.x) > 100))
{
c_scan(my.x, my.pan, vector(40, 60, 500), IGNORE_ME | SCAN_ENTS); // play with 500
}
my.skill20 += 1 * time_step;
c_move (my, rocket_speed.x, nullvector, IGNORE_FLAG2 | IGNORE_PASSABLE);
wait (1);
}
remove_rocket();
}
function enemy_event()
{
VECTOR temp;
if (event_type == EVENT_SCAN)
{
you.skill10 = 1; // stop scanning
while (you != NULL) // as long as the rocket exists
{
vec_set (temp.x, my.x);
vec_sub (temp.x, you.x);
vec_to_angle (you.pan, temp.x); // rotate it towards the target
wait (1);
}
}
}
function remove_rocket()
{
wait (1);
my.event = NULL;
ent_playsound (my, destroyed_wav, 1000); // play the explosion sound
set (my, INVISIBLE); // hide the rocket, keep ent_playsound playing
wait (-1.5); // wait for 1.5 seconds
ent_remove(me); // now remove it
}
```

trigger the EVENT_SHOOT event. I'd like to have a text displayed when the moving "ent1" hits "ent2".

```
#include <acknex.h>
#include <default.c>
SOUND* rocket_wav = "rocket.wav";
SOUND* destroyed_wav = "destroyed.wav";
STRING* rocket_mdl = "rocket.mdl";
function fire_rocket();
function shoot_rocket();
function remove_rocket();
function enemy_event();
function main()
{
fps_max = 70;
camera.ambient = 100;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
video_screen = 1; // start in full screen mode
level_load ("test.wmb");
wait (3);
on_mouse_left = fire_rocket;
}
TEXT* gotme_txt =
{
```

```
pos_x = 20;
pos_y = 20;
flags = SHOW;
}
action player_moves() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp;
set (my, INVISIBLE);
player = my;
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44);
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
function fire_rocket()
{
ent_create (rocket_mdl, player.x, shoot_rocket);
snd_play (rocket_wav, 50, 0);
}
function shoot_rocket()
{
VECTOR rocket_speed, temp;
my.emask |= (ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_rocket;
set (my, PASSABLE);
my.pan = camera.pan;
my.tilt = camera.tilt;
my.skill20 = 0;
vec_set(rocket_speed.x, nullvector);
rocket_speed.x = 10 * time_step;
my.skill10 = 0;
while (my.skill20 < 500)
{
if (vec_dist (player.x, my.x) > 100)
reset (my, PASSABLE);
if ((my.skill10 == 0) && (vec_dist (my.x, player.x) > 100))
{
c_scan(my.x, my.pan, vector(40, 60, 500), IGNORE_ME | SCAN_ENTS); // play with 500
}
my.skill20 += 1 * time_step;
c_move (my, rocket_speed.x, nullvector, IGNORE_FLAG2 | IGNORE_PASSABLE);
wait (1);
}
remove_rocket();
}
function remove_rocket()
{
wait (1);
my.event = NULL;
ent_playsound (my, destroyed_wav, 1000); // play the explosion sound
set (my, INVISIBLE); // hide the rocket, keep ent_playsound playing
wait (-1.5); // wait for 1.5 seconds
ent_remove(me); // now remove it
}
action enemy() // attach this action to your sensitive entities
{
my.emask |= (ENABLE_IMPACT); // they need to be sensitive to impact
my.event = enemy_event;
```

```
}
function enemy_event()
{
if (event_type == EVENT_IMPACT) // collided with another entity?
{
my.event = NULL; // don't react to other events for a while
str_cpy ((gotme_txt.pstring)[0], "Ouch! That hurt, man!");
wait (-3); // display the "Ouch" text for 3 seconds
str_cpy ((gotme_txt.pstring)[0], ""); // reset the string (hides the message)
my.event = enemy_event; // the enemy is sensitive to events again
}
}
```

a dead player doesn't react to mouse button clicks, pressed keys, etc

```
var players_health = 100;
action dead_player() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp;
my.skill40 = 0;
while (players_health > 0)
{
// decrease player's health slowly - this should be done by the enemy hits, of course
players_health -= 0.4 * time_step;
if (key_w || key_s || key_a || key_d)
{
ent_animate(my, "walk", my.skill40, ANM_CYCLE); // play the "walk" animation
my.skill40 += 5 * time_step; // "walk" animation speed
}
else
{
ent_animate(my, "stand", my.skill40, ANM_CYCLE); // play the "stand" animation
my.skill40 += 3 * time_step; // "stand" animation speed
}
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
if (mouse_left)
beep(); // just a beep here, you would have a firing function in real-life situations
wait (1);
}
// the player is dead here, so the movement keys and the left mouse button won't work anymore
my.skill40 = 0; // skill40 controls the "death" animation
while (my.skill40 < 95) // don't play all the animation frames because the result doesn't always look good
{
ent_animate(my, "death", my.skill40, NULL); // play the "death" animation
my.skill40 += 2 * time_step; // "death" animation speed
wait (1);
}
set (my, PASSABLE); // the corpse will be passable from now on
}
function zoom_startup() // allows the player to zoom in by pressing the "Z" key only if the player is alive
{
while (1)
{
// if (key_z) // this line would keep the zooming code active even after player's death, so don't use it
if ((key_z) && (players_health > 0)) // this line takes into account player's health as well
{
camera.arc -= 10 * time_step; // 10 gives the zoom-in speed
camera.arc = maxv(15, camera.arc); // make sure that camera.arc's value doesn't go below 15
}
else
{
if (camera.arc < 60) // camera.arc is below the default value?
camera.arc += 12 * time_step; // 12 gives the zoom-out speed
}
wait (1);
}
}
```

a sprite for the engine exhaust, not particles

```
VECTOR engine_offset;
function engine_sprite()
{
set(my, BRIGHT);
my.scale_x = 0.6; // set a proper scale for the engine sprite
my.scale_y = my.scale_x;
while (1)
{
my.pan = you.pan;
vec_set (my.x, engine_offset.x);
// choose a random frame for the engine - comment the following line if your sprite isn't animated
my.frame = 1 + random(4);
wait (1);
}
}
action my_ship() // attach this action to your space ship
{
ent_create("explo+5.tga", engine_offset.x, engine_sprite); // create the animated engine sprite
while (1)
{
// put your own flight code here
// the example below simply makes the plane fly in a circle
c_move (my, vector(20 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.pan += 2 * time_step; // 2 sets the radius of the rotation circle
// end of the simply flying example code
// the engine sprite code starts below
// place the engine sprite at the proper position, play with the numerical values below
vec_set(engine_offset.x, vector(-85, 0, -10));
vec_rotate(engine_offset.x, my.pan);
vec_add(engine_offset.x, my.x);
wait (1);
}
}
```

a simple example of a working player / passable water entity combination

```
#define amplitude skill1
#define water_speed skill2
#define number_of_vertices 1089 // the terrain entity used for the water has 33 x 33 vertices = 1089
var vertex_array[number_of_vertices];
var counter;
var index;
action players_code() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp, players_sensor;
set (my, INVISIBLE);
player = my;
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44);
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
action passable_water()
{
VECTOR temp;
set (my, TRANSLUCENT | PASSABLE);
my.alpha = 75;
my.ambient = -100;
if (my.amplitude == 0)
{
my.amplitude = 2; // default wave amplitude value
}
if (my.water_speed == 0)
{
my.water_speed = 10; // default wave speed value
}
counter = 0;
while (counter < number_of_vertices)
{
vertex_array[counter] = random(360); // set random values in vertex_array
counter += 1;
}
while (1)
{
index = 0;
while (index < number_of_vertices)
{
vec_for_mesh(temp, my, index); // store the vertex coordinates in temp
temp.z = sin(counter + vertex_array[index]) * my.amplitude; // change the z component
vec_to_mesh(temp, my, index); // deform the terrain entity
index += 1;
}
counter += my.water_speed * time_step;
wait(1);
}
}
```

an enemy that follows a path and if the player comes close, it starts chasing the player. When he is close enough to the player, he must shoot the player

```
var entity_speed = 3;
var movement_enabled = 0;
var dist_to_node;
var current_node = 1;
var angle_difference = 0;
VECTOR temp_angle;
VECTOR pos_node; // stores the position of the node
SOUND* hit_wav = "hit.wav";
function remove_bullets()
{
my.event = NULL;
ent_playsound (my, hit_wav, 1000);
wait (-0.1);
ent_remove (me);
}
function move_bullets()
{
my.pan = you.pan; // the bullet and the enemy have the same pan angle
my.emask |= (ENABLE_ENTITY | ENABLE_IMPACT | ENABLE_BLOCK); // the bullet is sensitive to other entities and level blocks
my.event = remove_bullets;
while (my)
{
my.skill1 = 15 * time_step; // bullet speed
my.skill2 = 0;
my.skill3 = 0;
move_mode = IGNORE_YOU + IGNORE_PASSABLE + IGNORE_PUSH; // ignores the enemy (its creator = you)
ent_move(my.skill1, nullvector);
wait (1);
}
}
function move_target()
{
while(1)
{
if(movement_enabled)
{
entity_speed = minv(5, entity_speed + 0.5 * time_step);
ent_animate(my, "walk", my.skill46, ANM_CYCLE); // play its "walk" frames animation
my.skill46 += 5 * time_step;
my.skill46 %= 100; // loop the animation
c_move(my, vector(entity_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x));
}
wait(1);
}
}
action my_enemy() // attach this action to your enemy model
{
VECTOR temp, bullet_coords;
while (!player) {wait (1);}
move_target();
result = path_scan(me, my.x, my.pan, vector(90, 80, 400)); // scan the area looking for the player
if (result) {movement_enabled = 1;}
path_getnode (my, 1, pos_node, NULL);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate towards the node
while(1)
{
if ((c_scan(my.x, my.pan, vector(120, 90, 1000), IGNORE_ME) > 0) && (you == player)) // detected the player?
break; // then get out of this loop!
dist_to_node = vec_dist(my.x, pos_node);
if(dist_to_node < 50) // close to the node?
{
current_node = path_nextnode(my, current_node, 1);
if (!current_node) {current_node = 1;} // reached the end of the path? Then start over!
path_getnode (my, current_node, pos_node, NULL);
}
wait(1);
}
// the enemy has spotted the player here
while (1) // so it rotates towards it and starts chasing it
{
vec_set(temp, player.x);
vec_sub(temp, my.x);
vec_to_angle(my.pan, temp);
my.tilt = 0;
if (vec_dist (player.x, my.x) < 400) // the enemy has come close enough to the player?
{
movement_enabled = 0; // then it should stop walking
my.skill46 = 0;
if (random(1) > 0.92) // play with 0.92
{
vec_set(bullet_coords.x, vector(25, -7, 14)); // play with these values
vec_rotate(bullet_coords.x, my.pan);
vec_add(bullet_coords.x, my.x);
ent_create ("bullet.mdl", bullet_coords.x, move_bullets); // create a bullet
while (my.skill46 < 100)
{
ent_animate(my, "standshoot", my.skill46, ANM_CYCLE); // play its "walk" frames animation
my.skill46 += 15 * time_step; // "shoot" animation speed
wait (1);
}
}
}
else // the player has moved away again?
```

```
{
movement_enabled = 1; // then start chasing him again
}
wait (1);
}
}
```

a sprite that fades softly in when I come near it and fades out gently when I move away from it

```
var fading_distance = 1000;
action fading_entity()
{
set (my, TRANSLUCENT);
my.alpha = 100;
// make sure that the player code includes the "player = my;" line
while (!player) {wait (1);}
while (1)
{
while(vec_dist (player.x, my.x) < fading_distance) {wait (1);}
while (my.alpha > 0)
{
my.alpha -= 5 * time_step;
wait (1);
}
// the entity is invisible here
while(vec_dist (player.x, my.x) > fading_distance) {wait (1);}
while (my.alpha < 100)
{
my.alpha += 5 * time_step;
wait (1);
}
}
}
```

an enemy that shoots the player and creates a health box (that can be picked up by the player) when it dies

```
STRING* bullet_mdl = "bullet.mdl";
SOUND* gothealth_wav = "gothealth.wav";
function fire_bullets(); // creates the bullets
function remove_bullets(); // removes the bullets after they've hit something
function got_shot();
function move_enemy_bullets();
#define idle 1
#define attacking 2
#define dead 3
#define status skill1
#define health skill10
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bullet
}
function health_box()
{
set (my, PASSABLE);
while (!player) {wait (1);}
// wait until the player comes close to the health box
while (vec_dist (player.x, my.x) > 60)
```

```
{
my.pan += 5 * time_step;
wait (1);
}
player.skill99 += 50; // player's health would be stored by its skill99 in this example
snd_play (gothealth_wav, 50, 0);
wait (-1);
ent_remove(my);
}
action my_enemy() // attach this action to your enemies
{
wait (-7);
while (!player) {wait (1);}
var idle_percentage = 0;
var run_percentage = 0;
var death_percentage = 0;
VECTOR content_right; // tracks the content in front of the player
VECTOR content_left; // tracks the content in front of the player
VECTOR temp;
set (my, POLYGON); // use accurate collision detection
my.health = 100;
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY); // the enemy is sensitive to impact with player's bullets
my.event = got_shot; // and runs this function when it is hit
my.status = idle; // that's the same thing with my.skill1 = 1; (really!)
while (my.status != dead) // this loop will run for as long as my.skill1 isn't equal to 3
{
if (my.status == idle) // hanging around?
{
ent_animate(my, "stand", idle_percentage, ANM_CYCLE); // play the "stand" aka idle animation
idle_percentage += 3 * time_step; // "3" controls the animation speed
if (vec_dist (player.x, my.x) < 1000) // the player has come too close?
{
// scanned in the direction of the pan angle and detected the player?
if ((c_scan(my.x, my.pan, vector(120, 60, 1000), IGNORE_ME) > 0) && (you == player))
{
my.status = attacking; // then attack the player even if it hasn't fired at the enemy yet
}
}
}
if (my.status == attacking) // shooting at the player?
{
// the road is clear? then rotate the enemy towards the player
if (c_content (content_right.x, 0) + c_content (content_left.x, 0) == 2)
{
vec_set(temp, player.x);
vec_sub(temp,my.x);
vec_to_angle(my.pan, temp); // turn the enemy towards the player
}
if (vec_dist (player.x, my.x) > 500)
{
vec_set(content_right, vector(50, -20, -15));
vec_rotate(content_right, my.pan);
vec_add(content_right.x, my.x);
if (c_content (content_right.x, 0) != 1) // this area isn't clear?
{
my.pan += 5 * time_step; // then rotate the enemy, allowing it to avoid the obstacle
}
vec_set(content_left, vector(50, 20, -15));
vec_rotate(content_left, my.pan);
vec_add(content_left.x, my.x);
if (c_content (content_left.x, 0) != 1) // this area isn't clear?
{
my.pan -= 5 * time_step; // then rotate the enemy, allowing it to avoid the obstacle
}
c_move (my, vector(10 * time_step, 0, 0), nullvector, GLIDE);
ent_animate(my, "run", run_percentage, ANM_CYCLE); // play the "run" animation
run_percentage += 6 * time_step; // "6" controls the animation speed
}
else
{
ent_animate(my, "alert", 100, NULL); // use the last frame from the "alert" animation here
}
if ((total_frames % 80) == 1) // fire a bullet each second
{
vec_for_vertex (temp, my, 8);
// create the bullet at enemy's position and attach it the "move_enemy_bullets" function
ent_create (bullet_mdl, temp, move_enemy_bullets);
}
if (vec_dist (player.x, my.x) > 1500) // the player has moved far away from the enemy?
{
my.status = idle; // then switch to "idle"
}
}
wait (1);
}
while (death_percentage < 100) // the loop runs until the "death" animation percentage reaches 100%
{
ent_animate(my, "deatha", death_percentage, NULL); // play the "die" animation only once
death_percentage += 3 * time_step; // "3" controls the animation speed
wait (1);
}
set (my, PASSABLE); // allow the player to pass through the corpse now
ent_create ("healthbox.mdl", my.x, health_box);
wait (3);
ent_remove (my);
}
function got_shot()
{
if (you.skill30 != 1) {return;} // didn't collide with a bullet? Then nothing should happen
my.health -= 35;
if (my.health <= 0) // dead?
{
my.status = dead; // stop the loop from "action my_enemy"
my.event = NULL; // the enemy is dead, so it should stop reacting to other bullets from now on
return; // end this function here
}
else // got shot but not (yet) dead?
{
my.status = attacking; // same thing with my.skill1 = 2
}
}
function move_enemy_bullets()
{
VECTOR bullet_speed; // this var will store the speed of the bullet
my.skill30 = 1; // I'm a bullet
// the bullet is sensitive to impact with other entities and to impact with level blocks
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = you.pan; // the bullet has the same pan
my.tilt = you.tilt; // and tilt with the enemy
bullet_speed.x = 50 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // or up / down on the z axis
// the loop will run for as long as the bullet exists (it isn't "null")
while (my)
{
// move the bullet ignoring its creator (the enemy)
c_move (my, bullet_speed, nullvector, IGNORE_YOU);
wait (1);
}
}
```

a high score table; how do I let the player input his name and then attach it to his score so that the score is next to his name

```
var high_score = 0;
var current_score = 0;
STRING* name_str = "#50";
STRING* input_str = "#50";
STRING* temp_str = "#10"; // just a temporary string
```

```
TEXT* name_txt =
{
pos_x = 30;
pos_y = 30;
string(name_str);
flags = SHOW;
}
TEXT* input_txt =
{
pos_x = 10;
pos_y = 450;
string(input_str);
flags = SHOW;
}
PANEL* score_txt =
{
pos_x = 0;
pos_y = 0;
digits (450, 30, "Current score: %.0f", *, 1, current_score);
flags = SHOW;
}
function init_startup()
{
var i;
while (1)
{
while (!mouse_right) {wait (1);}
while (mouse_right) {wait (1);}
str_cpy(input_str, "#50"); // reset the input string
current_score = 0;
i = 0;
while (i < 5)
{
if (mouse_left) {current_score += 1;}
while (mouse_left) {wait (1);}
i += time_step / 16;
wait (1);
}
if (current_score > high_score) // a new high score was achieved?
{
high_score = current_score;
str_cpy(input_str, "New high score! Please type in your name!");
wait (-3); // display the message for 3 seconds
str_cpy(input_str, "#50"); // reset the string
inkey(input_str); // now let's input player's name
str_cpy(name_str, input_str); // let's update the high score name on the screen as well
str_cat(name_str, " ---- "); // these characters separate the name and the actual high score
str_for_num(temp_str, high_score); // convert the numerical value to a string
str_cat(name_str, temp_str); // now add the high score to the string that contains the name of the player
}
str_cpy(input_str, "The test is over! Press the right mouse button to try again.");
wait (1);
}
}
```

create a mouse sensitivity snippet that allows you to see the change in real-time

```
var mouse_sensitivity = 2;
var mouse_type = 0;
BMAP* slider_pcx = "slider.pcx";
BMAP* background_pcx = "background.pcx";
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function toggle_mouse()
{
mouse_type += 1;
mouse_type %= 2;
mouse_mode = mouse_type * 2;
}
function init_startup()
{
on_mouse_right = toggle_mouse;
}
PANEL* gameover_pan =
{
bmap = background_pcx;
layer = 15;
pos_x = 20;
pos_y = 30;
// create a slider that changes mouse_sensitivity between 1 and 10 over 256 pixels
hslider(22, 0, 256, slider_pcx, 1, 10, mouse_sensitivity);
flags = SHOW;
}
action players_code() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp, players_sensor;
set (my, INVISIBLE);
player = my;
while (1)
{
my.pan -= 3 * mouse_force.x * mouse_sensitivity * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44);
camera.pan = my.pan;
camera.tilt += 2.5 * mouse_force.y * mouse_sensitivity * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

enemies to attack only when they can see the player

```
SOUND* alarm_wav = "alarm.wav";
action players_code() // attach this action to your player
{
var movement_speed = 20; // movement speed
VECTOR temp, players_sensor;
set (my, INVISIBLE); // 1st person player
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2; // play with 2
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

```
action my_enemy() // attach this action to your enemies
{
while (!player) {wait (1);} // wait until the player model is loaded in the level
var alarm_handle, walk_percentage = 0;
while (1)
{
// scans a sector that's got 120 degrees horizontally, 60 degrees vertically and a length of 1000 quants
// the player will be detected only if its action includes this line of code: "player = my;" (without the quotes)
if ((c_scan(my.x, my.pan, vector(120, 60, 1000), IGNORE_ME) > 0) && (you == player))
{
if (!snd_playing(alarm_handle)) // the alarm sound isn't playing already?
{
alarm_handle = snd_play(alarm_wav, 70, 0); // then let's play it!
}
}
c_move (my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.pan += 1 * time_step; // make the enemy rotate in a circle
ent_animate(my, "walk", walk_percentage, ANM_CYCLE); // play the "walk" animation in a loop
walk_percentage += 2 * time_step; // 2 controls the walk animation speed
wait (1);
}
}
```

add a sound to my fireplace entity

```
var fire_handle;
SOUND* fire = "fire.wav";
action my_fire() // attach this action to your fireplace entity
{
fire_handle = ent_playloop(my, fire, 80);
}
```

press a key and have an animated sprite show up, go through all its frames and then disappear

```
function animated_sprite()
{
set(my, BRIGHT);
my.alpha = 100;
my.frame = 1; // start with the first frame
while (my.frame < 20) // change this value if your animated sprite doesn't contain 20 frames
{
my.frame += 1.2 * time_step; // 1.2 gives the animation speed
wait (1);
}
// let's fade out the last animation frame (remove the "while" loop below if you don't want that to happen
while (my.alpha > 0)
{
my.alpha -= 0.7 * time_step; // 0.7 gives the fade-out animation speed
}
ent_remove (my); // remove the sprite
}
function create_sprite()
{
// creates the sprite 300 quants in front of the camera in this example; feel free to use your own values
// use your own animated sprite here
VECTOR sprite_offset;
vec_set (sprite_offset.x, vector(300, 0, 0));
vec_rotate (sprite_offset.x, camera.pan);
vec_add (sprite_offset.x, camera.x);
ent_create("muzzle+20.tga", sprite_offset.x, animated_sprite);
}
function init_startup()
{
on_c = create_sprite; // press the "C" key to create the sprite
}
```

attach a rotating entity to the player, but it must perform collision detection as it rotates around the player

```
ENTITY* dummy_ent;
// the function below moves the "real" rotating entity towards the position of the dummy entity (if possible)
function rotate_guard()
{
proc_mode = PROC_LATE;
while (!dummy_ent) {wait (1);}
VECTOR offset_speed;
while (1)
{
vec_diff(offset_speed.x, dummy_ent.x, my.x);
// normalize the speed with which the "real" entity follows the dummy entity, play with 10
vec_normalize(offset_speed, 10 * time_step);
c_move(my, nullvector, offset_speed.x, IGNORE_PASSABLE | GLIDE);
my.pan = dummy_ent.pan;
wait (1);
}
}
// the function below creates a dummy entity that rotates around the player at all times, penetrating the walls, etc
function rotate_around_player()
{
// set (my, INVISIBLE | PASSABLE); // make the dummy entity invisible and passable
ent_create("guard.mdl", my.x, rotate_guard); // use your own model here
var rotation_angle;
VECTOR my_speed, temp;
while (!player) {wait (1);} // wait until the player model is loaded
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY);
while (1)
{
my.x = player.x - 250 * cos(rotation_angle);
my.y = player.y - 250 * sin(rotation_angle);
my.z = player.z + 30; // play with 30
rotation_angle += 0.5 * time_step;
vec_set(temp.x, player.x);
vec_sub(temp.x, my.x);
vec_to_angle(my.pan, temp);
wait (1);
}
}
action players_code() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp, players_sensor;
set (my, INVISIBLE);
player = my;
dummy_ent = ent_create("dummy.mdl", nullvector, rotate_around_player); // put this line inside your player action
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44);
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

an NPC script in my rpg level

```
// use a wave file that contains something like "Stay away from the red rabbit, stranger!"
SOUND* advice_wav = "advice.wav";
action my_npc()
{
VECTOR front_pos;
var distance_covered = 0;
var sound_once = 1;
while (!player) {wait (1);} // wait until the player model is loaded
while (1)
{
vec_set(front_pos.x, vector(50, 0, 0)); // compute a position that's placed 50 quants in front of the player
// rotate "front_pos" in the direction (angles) given by the entity
vec_rotate(front_pos.x, my.pan);
vec_add(front_pos.x, my.x); // add the resulting vector to entity's position
// front_pos isn't touching any walls and the player didn't move 500 quants without pausing yet?
if ((c_content(front_pos.x, 0) == 1) && (distance_covered < 500))
{
// 5 gives the movement speed
distance_covered += c_move(my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.skill22 += 5 * time_step; // 5 gives the "walk" animation speed
ent_animate(my, "walk", my.skill22, ANM_CYCLE);
// if the player is closer than 150 quants to the npc and advice_wav wasn't played yet
if ((vec_dist(player.x, my.x) < 150) && (sound_once == 1))
{
sound_once = 2; // don't allow the sound to be played more than once per walking cycle
ent_playsound(my, advice_wav, 50);
}
}
else // the player is close to a wall or it has moved more than 500 quants?
{
sound_once = 1; // reset sound_once
distance_covered = 0; // reset distance_covered!
my.skill99 = 0;
while (my.skill99 < 5) // the npc stays in front of the wall for 5 seconds
{
my.skill99 += time_step / 16;
ent_animate(my, "stand", my.skill22, ANM_CYCLE);
my.skill22 += 3 * time_step; // 3 gives the "stand" animation speed
wait (1);
}
my.skill99 = my.pan;
my.skill99 += random(180); // and then adds a random pan angle to its initial pan, in order to avoid the wall
while (my.pan < my.skill99) // rotate the npc towards the direction given by its new pan angle
{
my.pan += 10 * time_step;
wait (1);
}
}
wait (1);
}
}
```

get the coordinates of a point on an entity's surface that was clicked using the left mouse button

```
VECTOR hit_coords;
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
action players_code() // attach this action to your player
{
var movement_speed = 20;
VECTOR temp, players_sensor;
set (my, INVISIBLE);
player = my;
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44);
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x);
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
function getcoords_startup()
{
VECTOR pos1, pos2;
while (1)
{
while (!mouse_left) {wait (1);}
while (mouse_left) {wait (1);}
beep();
pos1.x = mouse_pos.x;
pos1.y = mouse_pos.y;
pos1.z = 0;
vec_for_screen (pos1, camera);
pos2.x = mouse_pos.x;
pos2.y = mouse_pos.y;
pos2.z = 20000; // use a big value here
vec_for_screen (pos2, camera);
c_trace (pos1.x, pos2.x, IGNORE_PASSABLE | SCAN_TEXTURE);
// now "hitvertex" holds the coordinates of the closest vertex to the hit point
if (mouse_ent) // the mouse has clicked an entity?
vec_for_vertex(hit_coords, mouse_ent, hitvertex); // then get the xyz coordinates of "hitvertex"
wait (1);
}
}
PANEL* coords_pan = // displays the xyz coordinates of "hitvertex"
{
layer = 15;
digits(50, 20, 6 ,* , 1, hit_coords.x);
digits(50, 40, 6 ,* , 1, hit_coords.y);
digits(50, 60, 6 ,* , 1, hit_coords.z);
flags = SHOW;
}
```

allows my enemies to respawn; when one of the enemies dies, another is created

```
ENTITY* enemy;
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
```

```
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function destroy_enemy() // this function runs if the player clicks the enemy using the left mouse button
{
enemy = NULL; // free the enemy pointer
ent_remove(my); // remove the enemy from the level; another one will be created
}
function move_enemy()
{
var anim_percentage = 0;
my.emask = ENABLE_CLICK; // the enemy is sensitive to mouse clicks
my.event = destroy_enemy;
my.z += 50; // create the enemy 50 quants above the origin of the cave model
my.pan = random(360); // and give it a random pan angle
while (1)
{
c_move (my, vector(5 * time_step, 0, 0), nullvector, GLIDE); // "5" controls the walking speed
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 3 * time_step; // "3" controls the animation speed
anim_percentage %= 100;
wait (1);
}
}
action enemy_respawner() // attach this action to your cave model, etc
{
set (my, PASSABLE);
while (1)
{
while (enemy) {wait (1);} // don't do anything if the enemy exists already
// create the enemy and attach it the function named move_enemy
enemy = ent_create("guard.mdl", my.x, move_enemy);
}
}
```

example that uses four panels (one for each corner of the screen) and keeps them aligned at any resolution

```
BMAP* upperleft_tga = "upperleft.tga";
BMAP* upperright_tga = "upperright.tga";
BMAP* lowerleft_tga = "lowerleft.tga";
BMAP* lowerright_tga = "lowerright.tga";
PANEL* upperleft_pan =
{
bmap = upperleft_tga;
layer = 15;
flags = SHOW;
}
PANEL* upperright_pan =
{
bmap = upperright_tga;
layer = 15;
flags = SHOW;
}
PANEL* lowerleft_pan =
{
bmap = lowerleft_tga;
layer = 15;
flags = SHOW;
}
```

```
PANEL* lowerright_pan =
{
bmap = lowerright_tga;
layer = 15;
flags = SHOW;
}
function align_startup()
{
while (1)
{
// that's the easy part; the panel from the upper left corner will always be placed at x = 0 and y = 0;
upperleft_pan.pos_x = 0;
upperleft_pan.pos_y = 0;
// the panel from the upper right corner has x = horizontal resolution - panel bitmap width and y = 0;
upperright_pan.pos_x = screen_size.x - bmap_width(upperright_tga);
upperright_pan.pos_y = 0;
// the panel from the lower left corner has x = 0 and y = vertical resolution - panel bitmap height
lowerleft_pan.pos_x = 0;
lowerleft_pan.pos_y = screen_size.y - bmap_height(lowerleft_tga);
// the panel from the lower right corner x = horizontal resolution - panel bitmap width
// and y = vertical resolution - panel bitmap height
lowerright_pan.pos_x = screen_size.x - bmap_width(lowerright_tga);
lowerright_pan.pos_y = screen_size.y - bmap_height(lowerright_tga);
wait (1);
}
}
```

a ball to trigger an event when it passes through a passable sprite

```
var movement_speed = 10;
ENTITY* ball;
action simple_ball() // attach this action to your ball entity
{
VECTOR temp;
ball = my;
while (1)
{
temp.x = movement_speed * (key_cuu - key_cud) * time_step;
temp.y = movement_speed * (key_cul - key_cur) * 0.6 * time_step;
temp.z = 0; // use gravity, etc here
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
action passable_sprite() // attach this action to your passable sprite
{
set (my, PASSABLE);
while (!ball) {wait (1);} // wait until the player model is loaded
while (1)
{
if (vec_dist (ball.x, my.x) < 50) // play with 50
{
beep(); // do what you need here
// get rid of the "break" line if you want to trigger the event several times
break; // get out of the while loop
}
wait (1);
}
}
```

two spheres and I would like to create a 3D line that connects them

```
ENTITY* s1;
ENTITY* s2;
action sphere1() // attach this line to the first sphere
{
s1 = my; // I'm the first sphere
}
action sphere2() // attach this line to the second sphere
```

```
{
s2 = my; // I'm the second sphere
}
function line_startup()
{
while (!s1 || !s2) {wait (1);}
while (1)
{
// set the starting point for the 3D line
draw_line3d(vector(s1.x, s1.y, s1.z), NULL, 100);
// draw a blue line that connects s1 and s2
draw_line3d(vector(s2.x, s2.y, s2.z), vector(255, 0, 0), 100);
wait (1);
}
}
```

detect if the player is entering a certain room

```
STRING* detected_str = "#50";
TEXT* detected_txt =
{
pos_x = 200;
pos_y = 20;
string(detected_str);
flags = SHOW;
}
// attach this action to an entity that is placed inside the room, near the door
action detect_player()
{
set (my, PASSABLE);
// set (my, INVISIBLE); // remove this comment after you have set the proper scanning value
while (!player) {wait (1);}
VECTOR temp[3];
while (1)
{
// scan for the player every 5 frames to save a bit of cpu power
// the player will be detected if it comes closer than 200 quants to this entity
c_scan(my.x, my.pan, vector(360, 90, 200), IGNORE_ME | SCAN_ENTS); // play with 200
if (you == player) // detected the player?
{
beep(); beep(); // do what you need here (a level_load instruction, etc)
str_cpy((detected_txt.pstring)[0], "The player has entered the room");
break; // get out of the loop, the player has entered the room
}
else
{
str_cpy((detected_txt.pstring)[0], "The player is outside the room");
}
wait (5);
}
}
```

disable the camera movement when the user presses the zero key? And how can I enable the camera movement without having to press the zero key

```
// place a model in your level using Wed and then attach it this action
action my_camera()
{
var speed_factor = 5;
VECTOR camera_speed;
set (my, INVISIBLE); // make the camera entity invisible
while (1)
{
// use the left and right mouse buttons to move forward and backward; 5 gives the movement speed
camera_speed.x = speed_factor * (mouse_left - mouse_right);
// press and hold the shift key to increase the movement speed 6 times, useful for large areas
if (key_shift)
{
speed_factor = 30; // 5 * 6
}
else
{
speed_factor = 5; // restore the default movement speed if the shift key isn't pressed
}
// no need to change the y and z components of the camera_speed vector
camera_speed.y = 0;
camera_speed.z = 0;
my.pan -= 5 * mouse_force.x * time_step; // 5 = horizontal rotation speed
my.tilt -= 3 * mouse_force.y * time_step; // 3 = vertical rotation speed
// now move the entity in the direction given by its pan and tilt angles
c_move (my, camera_speed.x, nullvector, IGNORE_PASSABLE | GLIDE);
// the camera will inherit the same position and angles with the entity
vec_set (camera.x, my.x);
camera.pan = my.pan;
camera.tilt = my.tilt;
wait(1);
}
}
```

scan all the entities and check their skills rather then stopping at the closest one

```
var entity_id = -1;
var value[100];
function show_info() // runs if the entities are scanned by the scanner
{
while (event_type == EVENT_SCAN)
{
my.skill1 = 123; // set skill1 to 123 while the entity is being scanned
value[my.skill2] = my.skill1; // and copy it to the "value" array
my.scale_z = 4; // also increase the z scale of the entity that is being scanned
wait (1);
}
my.scale_z = 1; // restore the original z scale of the entity if it isn't being scanned anymore
my.skill1 = 0; // set skill1 to zero if the entity isn't scanned
value[my.skill2] = 0; // reset "value"
}
action scanner() // attach this action to the entity that will scan the area
{
while (1)
{
// scan using a pan angle of 120 degrees, a tilt of 60 degrees and a range of 500 quants
c_scan(my.x, my.pan, vector (120, 60, 500), IGNORE_ME | SCAN_ENTS | SCAN_LIMIT);
c_move(my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE); // 5 = movement speed
my.skill22 += 5 * time_step; // 5 gives the "walk" animation speed
my.pan += 1 * time_step;
ent_animate(my, "walk", my.skill22, ANM_CYCLE);
wait (1);
}
}
action scanned_objects() // attach this action to several entities
{
entity_id += 1;
my.skill2 = entity_id;
my.emask |= ENABLE_SCAN; // the scanned entities are sensitive to c_scan instructions
my.event = show_info;
}
PANEL* values_pan = // displays the first 10 "value" values
{
layer = 15;
digits(20, 20, 4 ,* , 1, value[0]);
digits(20, 40, 4 ,* , 1, value[1]);
digits(20, 60, 4 ,* , 1, value[2]);
digits(20, 80, 4 ,* , 1, value[3]);
digits(20, 100, 4 ,* , 1, value[4]);
digits(20, 120, 4 ,* , 1, value[5]);
digits(20, 140, 4 ,* , 1, value[6]);
digits(20, 160, 4 ,* , 1, value[7]);
digits(20, 180, 4 ,* , 1, value[8]);
digits(20, 200, 4 ,* , 1, value[9]);
flags = SHOW;
}
```

simple gravity system that works effectively

```
action physics_box() // this should be a function, but it can be used as a standalone action
{
VECTOR kick_speed;
ph_setgravity (vector(0, 0, -386));
set (my, SHADOW);
phent_settype (my, PH_RIGID, PH_BOX);
phent_setmass (my, 20, PH_BOX);
phent_setfriction (my, 50);
phent_setdamping (my, 50, 50);
phent_setelasticity (my, 30, 30);
kick_speed.x = 250; // kick speed
kick_speed.y = 0;
kick_speed.z = 50; // make it move a bit upwards as well
vec_rotate(kick_speed, camera.pan); // kick it depending on the angle of the camera
phent_addvelcentral(my, kick_speed); // kick the box, comment this line to turn the box into a regular one
}
function create_boxes()
{
VECTOR box_pos;
vec_set(box_pos.x, vector(100, 0, 0));
vec_rotate (box_pos.x, camera.pan);
vec_add (box_pos.x, camera.x);
ent_create ("box.mdl", box_pos.x, physics_box);
}
function init_startup()
{
on_c = create_boxes; // creates a box when the player presses the "c" key
}
```

enemy code

```
STRING* bullet_mdl = "bullet.mdl";
function fire_bullets(); // creates the bullets
function remove_bullets(); // removes the bullets after they've hit something
function got_shot();
function move_enemy_bullets();
#define idle 1
#define attacking 2
#define dead 3
#define status skill1
#define health skill10
function fire_bullets()
{
proc_kill(4); // don't allow more than 1 instance of this function to run
while (mouse_left) {wait (1);} // wait until the player releases the mouse button
ent_create (bullet_mdl, camera.x, move_bullets); // create the bullet at camera's position and attach it the "move_bullets" function
}
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bullet
}
action my_enemy() // attach this action to your enemies
{
while (!player) {wait (1);}
var idle_percentage = 0;
var run_percentage = 0;
var death_percentage = 0;
VECTOR content_right; // tracks the content in front of the player
VECTOR content_left; // tracks the content in front of the player
VECTOR temp;
set (my, POLYGON); // use accurate collision detection
my.health = 100;
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY); // the enemy is sensitive to impact with player's bullets
my.event = got_shot; // and runs this function when it is hit
my.status = idle; // that's the same thing with my.skill1 = 1; (really!)
while (my.status != dead) // this loop will run for as long as my.skill1 isn't equal to 3
{
if (my.status == idle) // hanging around?
{
ent_animate(my, "stand", idle_percentage, ANM_CYCLE); // play the "stand" aka idle animation
idle_percentage += 3 * time_step; // "3" controls the animation speed
if (vec_dist (player.x, my.x) < 1000) // the player has come too close?
{
// scanned in the direction of the pan angle and detected the player?
if ((c_scan(my.x, my.pan, vector(120, 60, 1000), IGNORE_ME) > 0) && (you == player))
{
my.status = attacking; // then attack the player even if it hasn't fired at the enemy yet
}
}
}
if (my.status == attacking) // shooting at the player?
{
// the road is clear? then rotate the enemy towards the player
if (c_content (content_right.x, 0) + c_content (content_left.x, 0) == 2)
{
vec_set(temp, player.x);
vec_sub(temp,my.x);
vec_to_angle(my.pan, temp); // turn the enemy towards the player
}
if (vec_dist (player.x, my.x) > 500)
{
vec_set(content_right, vector(50, -20, -15));
vec_rotate(content_right, my.pan);
vec_add(content_right.x, my.x);
if (c_content (content_right.x, 0) != 1) // this area isn't clear?
{
my.pan += 5 * time_step; // then rotate the enemy, allowing it to avoid the obstacle
}
vec_set(content_left, vector(50, 20, -15));
vec_rotate(content_left, my.pan);
vec_add(content_left.x, my.x);
if (c_content (content_left.x, 0) != 1) // this area isn't clear?
{
my.pan -= 5 * time_step; // then rotate the enemy, allowing it to avoid the obstacle
}
c_move (my, vector(10 * time_step, 0, 0), nullvector, GLIDE);
ent_animate(my, "run", run_percentage, ANM_CYCLE); // play the "run" animation
run_percentage += 6 * time_step; // "6" controls the animation speed
}
else
{
ent_animate(my, "alert", 100, NULL); // use the last frame from the "alert" animation here
}
if ((total_frames % 80) == 1) // fire a bullet each second
{
vec_for_vertex (temp, my, 8);
// create the bullet at enemy's position and attach it the "move_enemy_bullets" function
ent_create (bullet_mdl, temp, move_enemy_bullets);
}
if (vec_dist (player.x, my.x) > 1500) // the player has moved far away from the enemy?
{
my.status = idle; // then switch to "idle"
}
}
wait (1);
}
```

```
while (death_percentage < 100) // the loop runs until the "death" animation percentage reaches 100%
{
ent_animate(my, "deatha", death_percentage, NULL); // play the "die" animation only once
death_percentage += 3 * time_step; // "3" controls the animation speed
wait (1);
}
set (my, PASSABLE); // allow the player to pass through the corpse now
}
function got_shot()
{
if (you.skill30 != 1) {return;} // didn't collide with a bullet? Then nothing should happen
my.health -= 35;
if (my.health <= 0) // dead?
{
my.status = dead; // stop the loop from "action my_enemy"
my.event = NULL; // the enemy is dead, so it should stop reacting to other bullets from now on
return; // end this function here
}
else // got shot but not (yet) dead?
{
my.status = attacking; // same thing with my.skill1 = 2
}
}
function move_enemy_bullets()
{
VECTOR bullet_speed; // this var will store the speed of the bullet
my.skill30 = 1; // I'm a bullet
// the bullet is sensitive to impact with other entities and to impact with level blocks
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = you.pan; // the bullet has the same pan
my.tilt = you.tilt; // and tilt with the enemy
bullet_speed.x = 50 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // or up / down on the z axis
// the loop will run for as long as the bullet exists (it isn't "null")
while (my)
{
// move the bullet ignoring its creator (the enemy)
c_move (my, bullet_speed, nullvector, IGNORE_YOU);
wait (1);
}
}
```

any way to restore the sharpness of the images, regardless of the resolution of the screen

```
function main()

{
video_set(sys_metrics(0), sys_metrics(1), 32, 1); // detect and use the best screen resolution
level_load (test_wmb);
// put the rest of your code here
}
```

an enemy that comes towards the player until it is 100 quants away from him, and then maintains that distance. Even the player runs toward it, the enemy should retreat in order to keep those 100 quants between it and the player

```
#define idle 1
#define following 2
#define dead 3
#define status skill1
action my_enemy() // attach this action to your enemies
{
while (!player) {wait (1);}
var idle_percentage = 0;
var run_percentage = 0;
VECTOR temp;
set (my, POLYGON); // use accurate collision detection
my.status = idle; // that's the same thing with my.skill1 = 1; (really!)
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
my.z -= c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 20; // play with 20
while (my.status != dead) // this loop will run for as long as my.skill1 isn't equal to 3
{
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 20; // play with 20
if (my.status == idle) // hanging around?
{
ent_animate(my, "stand", idle_percentage, ANM_CYCLE); // play the "stand" aka idle animation
idle_percentage += 3 * time_step; // "3" controls the animation speed
if (vec_dist (player.x, my.x) < 1000) // the player has come too close?
{
// scanned in the direction of the pan angle and detected the player?
if ((c_scan(my.x, my.pan, vector(120, 60, 1000), IGNORE_ME) > 0) && (you == player))
{
my.status = following; // then attack the player even if it hasn't fired at the enemy yet
}
}
}
if (my.status == following) // shooting at the player?
{
vec_set(temp, player.x);
vec_sub(temp,my.x);
vec_to_angle(my.pan, temp); // turn the enemy towards the player
if (vec_dist (player.x, my.x) > 100) // the distance between player and enemy is bigger than 100 quants?
{
c_move (my, vector(10 * time_step, 0, temp.z), nullvector, GLIDE); // move towards the player
ent_animate(my, "run", run_percentage, ANM_CYCLE); // play the "run" animation
run_percentage += 6 * time_step; // "6" controls the animation speed, using normal animation
}
if (vec_dist (player.x, my.x) < 90) // the distance between player and enemy is smaller than 90 quants?
{
c_move (my, vector(-10 * time_step, 0, temp.z), nullvector, GLIDE); // move away from the player
ent_animate(my, "run", run_percentage, ANM_CYCLE); // play the "run" animation
run_percentage -= 6 * time_step; // "6" controls the animation speed, using a reversed animation
}
if (vec_dist (player.x, my.x) > 500) // the player has moved far away from the enemy?
{
my.status = idle; // then switch to "idle"
}
}
wait (1);
}
set (my, PASSABLE); // allow the player to pass through the corpse now
}
```

a circle on the ground around an entity

```
STRING* circle_tga = "circle.tga";
function adjust_circle()
{
set (my, PASSABLE);
my.tilt = 90;
while (1)
{
vec_set (my.x, you.x);
my.z = you.min_z + 35; // place the circle 35 quants above the entity's min_z value, play with 35
wait (1);
}
}
action my_entity() // moves the entity in a circle
{
var anim_percentage = 0;
ent_create(circle_tga, my.x, adjust_circle);
while (1)
{
anim_percentage += 3 * time_step; // 3 = "walk" animation speed
ent_animate(my, "walk", anim_percentage, ANM_CYCLE); // play the "walk" animation
c_move (my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.pan += 2 * time_step; // 2 sets the radius of the circle
wait (1);
}
}
```

an animated weapon that fires a bullet per second.

```
var front_offset = 30; // animates player's weapon while it is firing by moving it backwards
var init_offset = 30; // stores front_offset's value
ENTITY* weapon1;
STRING* bullet_mdl = "bullet.mdl";
SOUND* bullet_wav = "bullet.wav";
function fire_bullets();
function move_bullets();
function remove_bullets();
action players_weapon() // place your weapon model in the level and attach it this action
{
weapon1 = my; // I'm weapon1
VECTOR player1_pos; // stores the initial position of the player
VECTOR player2_pos; // stores the position of the player after a frame
VECTOR weapon_offset;
var weapon_height;
set (my, PASSABLE); // the weapon model is passable
while (!player) {wait (1);} // wait until the player is created
while (vec_dist (player.x, my.x) > 50) {wait (1);} // wait until the player comes close to pick up the weapon
my.roll = 0;
while (1)
{
vec_set (player1_pos.x, player.x); // store the initial player position
// place the weapon 30 quants in front, 18 quants to the right and 20 quants below camera's origin
vec_set (weapon_offset.x, vector (front_offset, -18, -20));
if (vec_dist (player1_pos.x, player2_pos.x) != 0) // the player has moved during the last frame?
{
weapon_height += 30 * time_step; // then offset weapon_height (30 = weapon waving speed)
weapon_offset.z += 0.3 * sin(weapon_height); // (0.3 = weapon waving amplitude)
}
// rotate weapon_offset according to the camera angles
vec_rotate (weapon_offset.x, vector (camera.pan, camera.tilt, 0));
vec_add (weapon_offset.x, camera.x); // add the camera position to weapon_offset
vec_set (my.x, weapon_offset.x); // set weapon's coords to weapon_offset
my.pan = camera.pan; // use the same camera angles for the weapon
my.tilt = camera.tilt;
vec_set (player2_pos.x, player.x); // store the new player coordinates after 1 frame
wait (1);
}
}
function weapon_startup()
{
on_mouse_left = fire_bullets; // call function fire_bullets() when the left mouse button is pressed
}
function fire_bullets() // includes the code that animates the weapon as well
{
var temp_seconds = 0;
VECTOR temp;
proc_kill(4); // don't allow more than 1 copy of this function to run
while (mouse_left) // this loop runs for as long as the left mouse button is pressed
{
front_offset -= 1; // move the weapon backwards a bit
vec_for_vertex (temp.x, weapon1, 29); // get the coordinates for the bullets' origin
// create the bullet at camera's position and attach it the "move_bullets" function
ent_create (bullet_mdl, temp.x, move_bullets);
temp_seconds = 0;
snd_play (bullet_wav, 100, 0); // play the bullet sound at a volume of 100
while (temp_seconds < 0.5)
{
temp_seconds += time_step / 16; // adds 0.5 to temp_seconds in 0.5 seconds)
front_offset -= 0.1; // restore the position of the weapon
wait (1);
}
while (front_offset < init_offset)
{
front_offset += 0.1; // restore the position of the weapon
wait (1);
}
front_offset = init_offset;
}
}
function move_bullets()
{
VECTOR bullet_speed; // stores the speed of the bullet
my.skill30 = 1; // I'm a bullet
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = camera.pan; // the bullet has the same pan
my.tilt = camera.tilt; // and tilt with the camera
bullet_speed.x = 200 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // then don't allow the gravity to have its ways with the bullet
while (my) // this loop will run for as long as the bullet exists (it isn't "null")
{
// move the bullet ignoring the passable entities and store the result in distance_covered
c_move (my, bullet_speed, nullvector, IGNORE_PASSABLE);
wait (1);
}
}
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bullet
}
```

a muzzle effect in front of an animated weapon model

```
ENTITY* weapon1;
STRING* bullet_mdl = "bullet.mdl";
STRING* muzzle_tga = "muzzle.tga";
STRING* explosion_pcx = "explosion+5.pcx"; // explosion sprite
SOUND* bullet_wav = "bullet.wav";
function fire_bullets();
function move_bullets();
function remove_bullets();
function display_muzzle();
function explosion_sprite();
action players_weapon() // place your weapon model in the level and attach it this action
{
VECTOR weapon_offset;
weapon1 = my; // I'm weapon1
set (my, PASSABLE); // the weapon model is passable
while (!player) {wait (1);} // wait until the player is created
while (1)
{
// place the weapon 30 quants in front, 18 quants to the right and 20 quants below camera's origin
vec_set (weapon_offset.x, vector (30, -18, -20));
// rotate weapon_offset according to the camera angles
vec_rotate (weapon_offset.x, vector (camera.pan, camera.tilt, 0));
vec_add (weapon_offset.x, camera.x); // add the camera position to weapon_offset
vec_set (my.x, weapon_offset.x); // set weapon's coords to weapon_offset
my.pan = camera.pan; // use the same camera angles for the weapon
my.tilt = camera.tilt;
wait (1);
}
}
function weapon_startup()
{
on_mouse_left = fire_bullets; // call function fire_bullets() when the left mouse button is pressed
}
function fire_bullets() // includes the code that animates the weapon as well
{
VECTOR temp;
proc_kill(4); // don't allow more than 1 copy of this function to run
while (mouse_left) // this loop runs for as long as the left mouse button is pressed
{
vec_for_vertex (temp.x, weapon1, 29); // get the coordinates for the bullets' origin
// create the bullet at camera's position and attach it the "move_bullets" function
ent_create (bullet_mdl, temp.x, move_bullets);
ent_create (muzzle_tga, temp.x, display_muzzle); // create the gun muzzle
snd_play (bullet_wav, 100, 0); // play the bullet sound at a volume of 100
wait (-0.1); // fire 10 bullets per second
}
}
function move_bullets()
{
VECTOR bullet_speed; // stores the speed of the bullet
my.skill30 = 1; // I'm a bullet
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = camera.pan; // the bullet has the same pan
my.tilt = camera.tilt; // and tilt with the camera
bullet_speed.x = 200 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // then don't allow the gravity to have its ways with the bullet
while (my) // this loop will run for as long as the bullet exists (it isn't "null")
{
// move the bullet ignoring the passable entities and store the result in distance_covered
c_move (my, bullet_speed, nullvector, IGNORE_PASSABLE);
wait (1);
}
}
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_create (explosion_pcx, my.x, explosion_sprite); // create the explosion sprite
set (my, INVISIBLE | PASSABLE);
wait (-2); // wait until the explosion_sprite() function is over
ent_remove (my); // and then remove the bullet
}
function display_muzzle()
{
set (my, PASSABLE | BRIGHT);
my.roll = 0.01;
my.ambient = 100; // and this line makes it even brighter
my.pan = weapon1.pan; // has the same pan and tilt
my.tilt = weapon1.tilt; // with the weapon
my.roll = random(360); // and a random roll angle (looks nicer)
my.scale_x = 0.3; // we scale it down
my.scale_y = my.scale_x; // on all the axis
my.scale_z = my.scale_z; // this would only be needed for a 3D (model-based) muzzle
weapon1.ambient = 100; // highlight the weapon (makes it look more realistic)
wait (2); // show the muzzle sprite for 2 frames
weapon1.ambient = 0; // restore the normal weapon ambient value
ent_remove (my); // and then remove it
}
function explosion_sprite()
{
set (my, PASSABLE | BRIGHT | TRANSLUCENT);
my.scale_x = 0.15; // we scale it down to 0.15
my.scale_y = my.scale_x; // on both axis
my.ambient = 100; // and we give it an ambient of 100
my.roll = random(360); // we set a random roll angle
my.alpha = 100; // but we set it to be completely opaque for now
while (my.frame < 5) // go through all the animation frames
{
my.frame += 2 * time_step; // animation speed
wait (1);
}
while (my.alpha > 0) // now fade the last frame quickly
{
my.alpha -= 50 * time_step; // 50 = fading speed
wait (1);
}
ent_remove (my);
}
```

change the texture of a block in a certain area of my level

```
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function replace_block()
{
ent_morph(me, "newblock.wmb"); // replaces the old block with the new one; newblock.wmb must exist!
my.event = NULL; // don't react to mouse clicks from now on
}
action special_block() // attach this action to your wmb block
{
// the block will be replaced when the player clicks it using the left mouse button
// you can replace the block when it is shot, etc.
my.emask |= ENABLE_CLICK;
my.event = replace_block;
}
```

check the wall texture and if its name is "steel", I'd like the player to turn its back against the wall automatically

```
action players_code() // attach this action to your player
{
var movement_speed = 20; // movement speed
VECTOR temp, players_sensor;
set (my, INVISIBLE); // 1st person player
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt = player.tilt;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2; // play with 2
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
// place the sensor at x = 50, y = 0, z = 0 in relation to the origin of the player model
vec_set (players_sensor.x, vector(50, 0, 0));
vec_rotate (players_sensor.x, my.pan);
vec_add (players_sensor.x, my.x);
if (c_trace (my.x, players_sensor.x, IGNORE_ME | IGNORE_PASSABLE | SCAN_TEXTURE) > 0) // something was hit?
{
if (str_cmpi(tex_name, "steel") == 1) // detected the steel texture?
vec_to_angle(my.pan, normal); // then rotate the player according to the normal of the surface!
}
wait (1);
}
}
TEXT* message_txt =
{
pos_x = 200;
pos_y = 20;
string(tex_name);
flags = SHOW;
}
```

bullet destroy itself when it hits a wall

```
SOUND* bullet_wav = "bullet.wav";
function remove_bullets() // this function runs when the bullet collides with the wall
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
my.event = NULL; // don't trigger bullet events (for this bullet) anymore
ent_playsound(my, bullet_wav, 2000);
set (my, INVISIBLE); // hide the bullet
wait (-1); // give the sound enough time to be played
ent_remove (my); // and then remove the bullet
}
function move_bullets()
{
VECTOR bullet_speed[3]; // stores the speed of the bullet
// the bullet is sensitive to impacts with entities and level blocks
my.emask = ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK;
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = you.pan; // the bullet has the same pan and tilt angles with its creator
my.tilt = you.tilt;
bullet_speed.x = 50 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // or up / down on the z axis
while (my) // this loop will run for as long as the bullet exists (it isn't "NULL")
{
// move the bullet ignoring its creator (the bullet generator)
c_move (my, bullet_speed, nullvector, IGNORE_PASSABLE | IGNORE_YOU);
wait (1);
}
}
action bullet_generator()
{
VECTOR bullet_pos[3];
while (1)
{
// generate bullets from the 108th vertex of the weapon (get the vertex number from Med)
vec_for_vertex(bullet_pos, my, 108);
ent_create("bullet.mdl", bullet_pos, move_bullets);
wait (-2); // fire a bullet every 2 seconds
}
}
```

player and lift code

```
ENTITY* lift1;
action players_code() // attach this action to your player
{
player = my; // I'm the player
while (!lift1) {wait (1);} // wait until the lift entity is loaded
var movement_speed = 20; // movement speed
var distance_to_ground;
VECTOR temp;
set (my, INVISIBLE); // 1st person player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
distance_to_ground = c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | SCAN_TEXTURE);
// replace "elevator1" with the name of the texture that is used by your lift
if (str_cmpi(tex_name, "elevator1")) // the player is using the lift?
{
my.z = lift1.z + 100; // play with 100
}
else // the player isn't using the lift? Then keep its feet on the ground
{
temp.z = -distance_to_ground + 20; // play with 20
}
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
action my_lift() // attach this action to your lift
{
lift1 = my;
while (!player) {wait (1);} // wait until the player entity is loaded
var init_z;
var max_height = 200; // the lift can move upwards for up to 200 quants
init_z = my.z; // store the initial height of the lift
while (1)
{
while (my.z < init_z + max_height)
{
my.z += 5 * time_step;
wait (1);
}
wait (-3); // wait 3 seconds at the top
while (my.z > init_z)
{
my.z -= 5 * time_step;
wait (1);
}
wait (-5); // wait 5 seconds at the bottom
}
}
```

player's pan angle change depending on the position of the mouse

```
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
action players_code()
{
VECTOR temp, temp2;
player = my; // I'm the player
while (1)
{
```

```
temp.x = mouse_pos.x;
temp.y = mouse_pos.y;
temp.z = 2000;
// create a position 2000 quants below the camera (not a critical value)
vec_for_screen(temp, camera);
vec_set(temp2, temp.x);
vec_sub(temp2, my.x);
vec_to_angle(my.pan, temp2); // player's pan points towards the mouse pointer here
my.tilt = 0; // don't change player's tilt angle, though
// put your own player movement code here
wait (1);
}
}
```

entities stop their animations if they aren't visible on the screen

```
action my_cow()
{
var anim_percentage = 0;
set(my, POLYGON);
while(1)
{
// the cow is visible on the screen? Then play its animation!
if (!(my.eflags & CLIPPED))
{
ent_animate(my, "eatgrass", anim_percentage, NULL);
anim_percentage += 2 * time_step;
anim_percentage %= 100;
}
else // the cow isn't visible on the screen here, so do something else if needed
{
// beep();
}
wait (1);
}
}
```

a weapon that fires bullets when I click the left mouse button, but doesn't allow the player to fire a new bullet until the "shoot" animation is 100% completed

```
var gun_firing = 0;
ENTITY* weapon1;
SOUND* fire_wav = "fire.wav";
function attach_weapon1()
{
weapon1 = my; // I'm the gun
set(my, PASSABLE);
while (1)
{
vec_set (my.x, vector (20, -10, 35)); // set the proper gun offset in relation to the player
vec_rotate (my.x, you.pan);
vec_add (my.x, you.x);
my.pan = you.pan;
my.tilt = camera.tilt;
wait (1);
}
}
action players_code() // simple player and 1st person camera code
{
player = my; // I'm the player
ent_create ("gun1.mdl", nullvector, attach_weapon1);
while (1)
{
```

```
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, GLIDE);
vec_set (camera.x, player.x); // use player's x and y for the camera as well
camera.z += 30; // place the camera 30 quants above the player on the z axis (approximate eye level)
camera.pan -= 5 * mouse_force.x * time_step; // rotate the camera around by moving the mouse
camera.tilt += 3 * mouse_force.y * time_step; // on its x and y axis
player.pan = camera.pan; // the camera and the player have the same pan angle
wait (1);
}
}
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bullet
}
function move_bullets()
{
VECTOR bullet_speed[3]; // stores the speed of the bullet
// the bullet is sensitive to impacts with entities and level blocks
my.emask = ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK;
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = weapon1.pan;
my.tilt = weapon1.tilt;
bullet_speed.x = 150 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // or up / down on the z axis
while (my) // this loop will run for as long as the bullet exists (it isn't "NULL")
{
// move the bullet ignoring its creator (weapon1)
c_move (my, bullet_speed, nullvector, IGNORE_PASSABLE | IGNORE_YOU);
wait (1);
}
}
function fire_bullets()
{
if (gun_firing)
return; // don't allow the player to fire until the previous "shot" animation has finished
snd_play(fire_wav, 100, 0);
gun_firing = 1;
VECTOR bullet_pos[3];
var anim_factor = 0;
// generate bullets from the 952th vertex of the weapon (get the vertex number from Med)
vec_for_vertex(bullet_pos, weapon1, 952);
while (anim_factor < 70) // the gun fires the bullet at 70% of its animation - play with 70
{
ent_animate(weapon1, "shot", anim_factor, NULL);
anim_factor += 7 * time_step;
wait (1);
}
ent_create("bullet.mdl", bullet_pos, move_bullets);
while (anim_factor < 100) // continue with the rest of the "shot" animation
{
ent_animate(weapon1, "shot", anim_factor, NULL);
anim_factor += 6 * time_step;
wait (1);
}
gun_firing = 0; // allow the gun to fire again
}
function bullets_startup()
{
on_mouse_left = fire_bullets;
}
```

a countdown timer that will work while the player is running, and if the timer <= 0 the player is not able to run any more. Also, the timer must recharge so that the player can run again.

```
var countdown_timer = 10;
SOUND* gottime_wav = "gottime.wav";
action players_code() // attach this action to your player
```

```
{
var movement_speed = 20; // movement speed
VECTOR temp;
set (my, INVISIBLE); // 1st person player
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt = player.tilt; //5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2; // play with 2
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
if (countdown_timer > 0) // the countdown time didn't reach zero?
{
temp.x *= 2; // increase the forward movement speed with 100%
temp.y *= 1.5; // increase the sideway movement speed with 50%
}
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
action get_time() // attach this action to the time powerups that can be picked up by the player
{
set (my, PASSABLE);
while (!player) {wait (1);} // wait until the player model is loaded
while (vec_dist (player.x, my.x) > 50) {wait (1);}
snd_play(gottime_wav, 100, 0);
set (my, INVISIBLE);
countdown_timer += 10;
wait (-2);
ent_remove(my);
}
function countdown_startup()
{
while (1)
{
countdown_timer -= time_step / 16;
countdown_timer = maxv(countdown_timer, 0); // don't allow the timer to go below zero
wait (1);
}
}
PANEL* time_pan =
{
layer = 15;
digits(20, 20, 4 ,* , 1, countdown_timer);
flags = SHOW;
}
```

props that only animate when they get destroyed

```
var gun_firing = 0;
ENTITY* weapon1;
SOUND* fire_wav = "fire.wav";
function attach_weapon1()
{
weapon1 = my; // I'm the gun
set(my, PASSABLE);
while (1)
{
vec_set (my.x, vector (20, -10, 35)); // set the proper gun offset in relation to the player
vec_rotate (my.x, you.pan);
vec_add (my.x, you.x);
my.pan = you.pan;
my.tilt = camera.tilt;
wait (1);
}
}
action players_code() // simple player and 1st person camera code
{
player = my; // I'm the player
ent_create ("gun1.mdl", nullvector, attach_weapon1);
while (1)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, GLIDE);
vec_set (camera.x, player.x); // use player's x and y for the camera as well
camera.z += 30; // place the camera 30 quants above the player on the z axis (approximate eye level)
camera.pan -= 5 * mouse_force.x * time_step; // rotate the camera around by moving the mouse
camera.tilt += 3 * mouse_force.y * time_step; // on its x and y axis
player.pan = camera.pan; // the camera and the player have the same pan angle
wait (1);
}
}
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bullet
}
function move_bullets()
{
my.skill99 = 1357; // uniquely identify a bullet
VECTOR bullet_speed[3]; // stores the speed of the bullet
// the bullet is sensitive to impacts with entities and level blocks
my.emask = ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK;
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = weapon1.pan;
my.tilt = weapon1.tilt;
bullet_speed.x = 150 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // or up / down on the z axis
while (my) // this loop will run for as long as the bullet exists (it isn't "NULL")
{
// move the bullet ignoring its creator (weapon1)
c_move (my, bullet_speed, nullvector, IGNORE_PASSABLE | IGNORE_YOU);
wait (1);
}
}
function fire_bullets()
{
if (gun_firing)
return; // don't allow the player to fire until the previous "shot" animation has finished
snd_play(fire_wav, 100, 0);
gun_firing = 1;
VECTOR bullet_pos[3];
var anim_factor = 0;
// generate bullets from the 952th vertex of the weapon (get the vertex number from Med)
vec_for_vertex(bullet_pos, weapon1, 952);
while (anim_factor < 70) // the gun fires the bullet at 70% of its animation - play with 70
{
ent_animate(weapon1, "shot", anim_factor, NULL);
anim_factor += 7 * time_step;
wait (1);
}
ent_create("bullet.mdl", bullet_pos, move_bullets);
while (anim_factor < 100) // continue with the rest of the "shot" animation
{
ent_animate(weapon1, "shot", anim_factor, NULL);
anim_factor += 6 * time_step;
wait (1);
}
gun_firing = 0; // allow the gun to fire again
}
function bullets_startup()
{
```

```
on_mouse_left = fire_bullets;
}
// destroyable crate / barrel code from here on
function explo_sprite()
{
set(my, BRIGHT);
my.alpha = 100;
my.scale_x = 3; // this value gives the scale of the explosion sprite
my.scale_y = my.scale_x;
my.frame = 1;
while (my.frame < 5)
{
my.frame += 0.5 * time_step;
wait (1);
}
while (my.alpha > 0)
{
my.alpha -= 3 * time_step;
wait (1);
}
ent_remove (my);
}
function destroy_crate()
{
// don't do anything if the crate wasn't destroyed by one of player's bullets
// we have set skill99 to 1357 for each bullet inside function move_bullets() - above
if (you.skill99 != 1357)
return;
my.skill1 -= 1;
}
action my_crate()
{
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY);
my.event = destroy_crate;
my.skill1 = 3; // allow the
while (my.skill1 > 1) {wait (1);}
// the crate was shot 3 times here
ent_create("explo+5.pcx", my.x, explo_sprite);
my.alpha = 100;
set (my, TRANSLUCENT);
while (my.alpha > 0)
{
my.alpha -= 7 * time_step;
wait (1);
}
ent_remove (my); // and then remove the crate
}
```

an entity passing through a tunnel and exiting on another tunnel? It's like a portal in Quake 3; when the player enters the portal, it'll exit somewhere else in the map

```
ENTITY* beamer1;
ENTITY* beamer2;
action players_code() // attach this action to your player
{
var movement_speed = 20; // movement speed
VECTOR temp;
set (my, INVISIBLE); // 1st person player
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt = player.tilt; //5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2; // play with 2
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
function beam_me1()
{
my.event = NULL; // don't trigger several events
wait (1);
set (beamer2, PASSABLE); // don't allow beamer2 to teleport the player back to beamer1
vec_set(player.x, beamer2.x);
my.event = beam_me1;
}
action portal1() // attach this action to the first portal
{
beamer1 = my;
while (!player) {wait (1);}
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY);
while (1)
{
if (vec_dist(player.x, my.x) < 10) // the player was just beamed here?
{
set (my, PASSABLE);
my.event = NULL; // then don't allow this beamer to teleport it back
// wait until the player has moved away from this beamer
while (vec_dist (my.x, player.x) < 100) {wait (1);}
}
else
{
reset (my, PASSABLE);
my.event = beam_me1;
}
wait (1);
}
}
function beam_me2()
{
my.event = NULL; // don't trigger several events
wait (1);
set (beamer1, PASSABLE); // don't allow beamer1 to teleport the player back to beamer2
vec_set(player.x, beamer1.x);
my.event = beam_me2;
}
action portal2() // attach this action to the second portal
{
beamer2 = my;
while (!player) {wait (1);}
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY);
while (1)
{
if (vec_dist(player.x, my.x) < 10) // the player was just beamed here?
{
set (my, PASSABLE);
my.event = NULL; // then don't allow this beamer to teleport it back
// wait until the player has moved away from this beamer
while (vec_dist (my.x, player.x) < 100) {wait (1);}
}
else
{
reset (my, PASSABLE);
my.event = beam_me2;
}
wait (1);
}
}
```

3D breakout game

```
#include <acknex.h>
```

```
#include <default.c>
ENTITY* ball; // defined for further use
function control_ball();
function main()
{
video_mode = 8; // 1024 x 768 pixels
video_screen = 1; // full screen mode
level_load("breaker.wmb");
while (!player) {wait (1);}
ball = ent_create("ball.mdl", vector (800, 0, -250), control_ball);
}
action players_paddle()
{
player = my;
set(my, TRANSLUCENT);
my.alpha = 70;
while (1)
{
c_move (my, nullvector, vector(0, -50 * mouse_force.x * time_step, 40 * mouse_force.y * time_step), NULL);
wait (1);
}
}
function ball_was_hit()
{
vec_to_angle(my.pan, bounce); // makes the ball bounce off on collisions
// add some randomness to the ball (we wouldn't want to see it bouncing back and forth between 2 obstacles)
my.pan += 1 - random(2);
}
function control_ball()
{
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = ball_was_hit;
while(1)
{
my.skill1 = c_move (my, vector(50 * time_step, 0, 0), nullvector, IGNORE_PASSABLE);
if (my.skill1 < 5) // the ball got stuck? might be needed under special circumstances
{
my.pan = random(360); // then help it escape by setting a random pan angle for it!
}
wait(1);
}
}
```

play a movie on a surface in a level? I would like to be able to click the surface in order to replay it.

```
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function surface_clicked()
{
media_play ("mymovie.avi", bmap_for_entity (my, 0), 100); // use your own movie name
}
// attach this action to a sprite or a model
// if you are using a sprite, make sure that it has the same size (in pixels) with the size of the movie
action movie_surface()
{
my.emask |= ENABLE_CLICK;
my.event = surface_clicked;
while (!player) {wait (1);} // wait until the player model is loaded
wait (-10); // wait for 10 seconds, and then play the movie once
media_play ("mymovie.avi", bmap_for_entity (my, 0), 100); // use your own movie name
}
```

to fire at the enemy with the smallest health value

```
var minimum_health = 100;
ENTITY* turret;
ENTITY* enemy_target;
action my_turret() // attach this action to your turret
{
turret = my;
VECTOR temp;
while (1)
{
// scan all the entities that are closer than 1000 quants to this entity
c_scan(my.x, my.pan, vector(360, 180, 1000), IGNORE_ME);
if (enemy_target) // an enemy target was detected?
{
vec_set(temp, enemy_target.x);
vec_sub(temp, my.x);
vec_to_angle(my.pan, temp); // rotate towards the enemy target
if (total_frames % 120 == 1) // fire a bullet every 2 seconds
{
// ent_create("bullet.mdl", my.x, move_bullet); // use your own shooting code here
}
}
wait (1);
}
}
function i_am_scanned()
{
if (my.skill1 <= minimum_health)
{
minimum_health = my.skill1;
enemy_target = my;
my.scale_z = 1.5; // highlight the enemy with the minimum health value (make it taller)
}
else
{
my.scale_z = 1;
}
}
action my_enemies() // attach this action to your enemies
{
my.skill1 = random(100); // each enemy has a random health value
my.emask |= ENABLE_SCAN; // make the entity sensitive to scanning
my.event = i_am_scanned;
}
function init_startup()
{
fps_max = 60; // limit the frame rate to 60 fps
random_seed(0); // generate random values at each game run

}
```

light switch, and somewhere else the actual light that needs to be turned on by that switch

```
var light_on = 0; // the light is turned off at first
action my_switch() // attach this action to your light switch
{
while (1)
{
// scan the entities that are closer than 100 quants to the light switch
c_scan(my.x, my.pan, vector(360, 180, 100), IGNORE_ME);
if (you == player) // the player has come closer than 100 quants to the light switch entity?
{
light_on = 1; // then turn the light on!
}
wait (1);
}
}
action my_lightbulb() // attach this action to your light entity
{
while (!light_on) {wait (1);} // wait until the player comes close to the switch
set (my, CAST);
my.lightrange = 500;
my.red = 200;
my.green = 200;
my.blue = 160;
}
```

one entity to the position of another entity, but also over shoot its position and correct itself? This way the entity would look like it's jiggling its position when it gets to its target position

```
ENTITY* my_target;
action my_destination() // attach this action to the target entity
{
my_target = my;
}
action target_seeker() // attach this action to your target seeking entity
{
while (!my_target) {wait (1);} // wait until the target entity is loaded
VECTOR temp, temp_angle;
var temp_time;
vec_set(temp_angle, my_target.x);
vec_sub(temp_angle, my.x);
vec_to_angle(my.pan, temp_angle);
my.tilt = 0;
// the entity was rotated towards the destination here
// let's change its angle a bit, making it miss the target at first
if (random(1) > 0.5)
my.pan += 5; // play with 5
else
my.pan -= 5; // play with -5
// we have added -5 or 5 degrees to the pan angle here, so the entity will miss the target a bit
// the following loop runs until the entity comes close enough to the target
while (vec_dist(my.x, my_target.x) > 100) // play with 100
{
c_move (my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
// keep the entity moving in the wrong direction for 2 more seconds
temp_time = 0;
while (temp_time < 2)
{
c_move (my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
temp_time += time_step / 16;
wait (1);
}
// now rotate towards the target correctly
vec_set(temp_angle, my_target.x);
vec_sub(temp_angle, my.x);
vec_to_angle(my.pan, temp_angle);
while (vec_dist(my.x, my_target.x) > 40) // stop when the entity is closer than 40 quants to the target (play with 40)
{
c_move (my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
// the goal was reached here
}
```

my camera to tilt when the player is going through a slope on a mountain

```
action players_code() // attach this action to your player
{
var movement_speed = 10; // movement speed
VECTOR temp;
ANGLE temp_angles;
set (my, INVISIBLE); // 1st person player
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt = player.tilt;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2; // play with 2
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
temp_angles.tilt = 0;
temp_angles.roll = 0;
temp_angles.pan = -my.pan;
vec_rotate(normal, temp_angles);
temp_angles.tilt = -asin(normal.x);
temp_angles.roll = -asin(normal.y);
my.tilt += 0.1 * ang(temp_angles.tilt - my.tilt); // play with 0.1
my.roll += 0.1 * ang(temp_angles.roll - my.roll); // play with 0.1
wait (1);
}
}
```

trap the mouse pointer in a rectangular area located in the center of the screen

```
// trap the mouse in an area that has 200x100 pixels and is located in the center of the screen
var area_x = 200;
var area_y = 100;
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
VECTOR temp1_pos, temp2_pos;
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
temp1_pos.x = (screen_size.x - area_x) / 2;
temp2_pos.x = (screen_size.x + area_x) / 2;
mouse_pos.x = clamp(mouse_pos.x, temp1_pos.x, temp2_pos.x);
temp1_pos.y = (screen_size.y - area_y) / 2;
temp2_pos.y = (screen_size.y + area_y) / 2;
mouse_pos.y = clamp(mouse_pos.y, temp1_pos.y, temp2_pos.y);
wait(1);
}
}
```

an entity moves for a specified number of seconds when I press a button

```
action moving_puppet()
{
var movement_time = 5; // move for 5 seconds when the "M" key is pressed
var movement_speed = 2; // movement speed
VECTOR temp;
while (1)
{
while (!key_m) {wait (1);} // wait until the player presses the "M" key on the keyboard
while (key_m) {wait (1);} // wait until the player releases the "M" key
while (movement_time > 0)
{
movement_time -= time_step / 16; // decrease 1 from movement_time each second
my.skill1 += 4 * time_step; // 4 gives the animation speed
vec_set (temp.x, my.x); // trace 10,000 quants below the entity
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) + 20; // play with 20
// move the entity with the speed given by movement_speed along the direction pointed by its pan angle
c_move (my, vector(movement_speed, 0, temp.z), nullvector, IGNORE_PASSABLE | GLIDE);
// animate the model using its "walk" animation in a loop
ent_animate(my, "walk", my.skill1, ANM_CYCLE);
wait (1);
}
// the walking is over here, so let's switch to the "stand" animation
ent_animate(my, "stand", 0, 0);
movement_time = 5; // prepare for a new movement session
}
}
```

a sound play once as soon as a comparison instruction is successful

```
STRING* password_str = "iliketoeatalot";
STRING* input_str = "#20"; // this empty string can store up to 20 characters
STRING* messages_str = "30";
SOUND* passok_wav = "passok.wav";
TEXT* input_txt =
{
pos_x = 300;
pos_y = 400;
string(input_str);
}
TEXT* messages_txt =
{
pos_x = 300;
pos_y = 300;
string(messages_str);
}
function password_startup()
{
wait (-3); // wait until the game starts, etc
// the following loop will run until the correct password is typed in
while (1)
{
str_cpy(messages_str, "Type in the correct password");
messages_txt.flags |= SHOW; // show the intro message
wait (-3); // for 3 seconds
messages_txt.flags &= ~SHOW; // now hide the intro message
input_txt.flags |= SHOW; // show the input
str_cpy(input_str, "#20"); // reset the string (if needed)
inkey(input_str);
if (str_cmpi(input_str, password_str) == 1) // the correct password was typed in?
{
snd_play(passok_wav, 90, 0);
break; // get out of the loop
}
}
// the correct password was typed in here
str_cpy(messages_str, "Logged into the system");
messages_txt.flags |= SHOW; // show the outro message
wait (-5); // for 5 seconds
messages_txt.flags &= ~SHOW; // now hide the outro message
sys_exit(NULL); // shut down the engine
}
```

the camera of my game to an object (sphere, etc) so that the camera has a physical behaviour and doesn't go through walls anymore

```
VECTOR movement_force;
action solid_camera() // attach this action to a sphere model
{
set (my, INVISIBLE); // no need to see the sphere model
while (1)
{
// use the W and S keys to move forward / backward, 5 gives the movement speed
movement_force.x = 5 * (key_w - key_s);
// use the A and D keys to move sideways, 3 gives the movement speed
movement_force.y = 3 * (key_a - key_d);
// don't allow the z value of movement_force to change
movement_force.z = 0;
my.pan -= 3 * mouse_force.x * time_step; // 3 = horizontal rotation speed
my.tilt -= 3 * mouse_force.y * time_step; // 3 = vertical rotation speed
// now move the sphere in the direction given by its pan and tilt angles
c_move (my, movement_force.x, nullvector, IGNORE_PASSABLE | GLIDE);
// the camera will inherit the same position and angles with the sphere
vec_set (camera.x, my.x);
camera.pan = my.pan;
camera.tilt = my.tilt;
wait(1);
}
}
```

be able to pick, move and drop an object using the mouse cursor

```
BMAP* pointer_tga = "pointer.tga";
VECTOR temp;
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function pick_or_drop()
{
wait (1);
my.skill1 += 1;
if ((my.skill1 % 2) == 1) // clicked the object?
{
while (mouse_left) {wait (1);} // wait until the player releases the left mouse button
while (!mouse_left) // move the object until the player presses the mouse button again
{
temp.x = mouse_cursor.x;
temp.y = mouse_cursor.y;
temp.z = 200; // move the object 200 quants below the camera, play with this value
vec_for_screen(temp.x, camera);
vec_set (my.x, temp.x);
set (my, PASSABLE);
wait (1);
}
}
else // drop the object here
{
vec_set (temp.x, my.x);
temp.z -= 3000; // trace up to 5,000 quants below the player
// make sure to drop the object on the ground
my.z -= c_trace (my.x, temp.x, IGNORE_ME + IGNORE_SPRITES + IGNORE_MODELS + USE_BOX);
reset (my, PASSABLE);
}
}
action click_and_move()
{
my.skill1 = 0;
my.emask |= ENABLE_CLICK;
my.event = pick_or_drop;
}
```

a dynamic light that can be picked up by pressing space near it and follows the player. The light should always be on, even after a level changing instruction

```
var got_flashlight = 0;
SOUND* gotflashlight_wav = "gotflashlight.wav";
ENTITY* flashlight;
function move_flashlight()
{
set (my, PASSABLE | INVISIBLE | CAST);
my.lightrange = 350;
my.red = 200;
my.green = 200;
my.blue = 160;
while (player) // this loop will run for as long as the player pointer exists
{
vec_set(my.x, player.x);
wait (1);
}
}
action my_flashlight()
{
while (!player) {wait (1);}
while (1)
{
if (vec_dist(player.x, my.x) < 70)
{
if (key_space)
{
snd_play(gotflashlight_wav, 50, 0);
flashlight = ent_create("flashlight.mdl", player.x, move_flashlight);
got_flashlight = 1;
wait (2);
break; // get out of this loop
}
}
wait (1);
}
set(my, PASSABLE | INVISIBLE); // hide the flashlight model on the floor
while (vec_dist(player.x, my.x) < 300) {wait (1);} // wait until the player moves away
// and then remove the flashlight model from the floor (give the sound effect enough time to play)
ent_remove (my);
}
action players_code() // attach this action to your player model in all the levels
{
var movement_speed = 10; // movement speed
VECTOR temp;
set (my, INVISIBLE); // 1st person player
player = my; // I'm the player
// this section recreates the flashlight (if needed)
if (got_flashlight == 1) // the flashlight was picked up in a previous level?
{
flashlight = ent_create("flashlight.mdl", player.x, move_flashlight); // the recreate it!
}
// end of the section that recreates the flashlight (if needed)
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt = player.tilt; //5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2; // play with 2
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
action lever_changer() // attach this action to your level changing entity
{
while (!player) {wait (1);} // wait until the player model is loaded
// wait until the player comes closer than 100 quants to the level changing entity
while (vec_dist (player.x, my.x) > 100) {wait (1);}
// now load the new level; the flashlight will be recreated if it was picked up in a previous level
level_load("level2.wmb");
}
```

a mini math quiz with a text like 5 x 2 that appears on the screen. The users enter an answer and press Enter in order to move on to the following question. If they are wrong their score is decreased

```
var score = 0, number1, number2, my_result;
STRING* question_str = "#20";
STRING* answer_str = "#5";
STRING* temp_str = "#5";
SOUND* goodjob_wav = "goodjob.wav";
SOUND* missed_wav = "missed.wav";
TEXT* question_txt =
{
pos_x = 200;
pos_y = 200;
string(question_str);
flags = SHOW;
}
TEXT* answer_txt =
{
pos_x = 300;
pos_y = 200;
string(answer_str);
flags = SHOW;
}
PANEL* score_pan =
{
layer = 15;
digits(700, 20, 4, * , 1, score);
flags = SHOW;
}
function init_startup()
{
random_seed(0); // generate a random sequence of numbers
wait (-3); // wait until the level is loaded
while (1)
{
str_cpy(question_str, "What is ");
number1 = integer(random(10) + 1); // number 1 ranges from 1 to 10
str_for_num(temp_str, number1); // convert number1 to a string
str_cat(question_str, temp_str); // add number1 to question_str
str_cat(question_str, " x ");
number2 = integer(random(10) + 1); // number 2 ranges from 1 to 10
str_for_num(temp_str, number2); // convert number1 to a string
str_cat(question_str, temp_str); // add number2 to question_str
wait (1);
str_cat(question_str, " = ");
str_cpy(answer_str, "#5");
inkey(answer_str);
my_result = str_to_num(answer_str);
if (my_result == (number1 * number2)) // the player has got the correct answer?
{
score += 1; // then increase the score
snd_play(goodjob_wav, 60, 0);
}
else
{
snd_play(missed_wav, 60, 0);
score -= 1; // then decrease the score
score = maxv(0, score); // don't allow negative score values
}
wait (-3); // wait for 5 seconds before generating a new exercise
}
}
```

a high precision timer for my racing game

```
FONT* arial_font = "Arial#20";
var hours;
var minutes;
var seconds;
var milliseconds;
function timer_startup()
{
while (1)
{
milliseconds += (timer() / 1000);
if (milliseconds > 1000)
{
milliseconds -= 1000;
seconds += 1;
}
if(seconds > 59)
{
minutes += 1;
seconds -= 60;
if(minutes >= 59)
{
hours += 1;
minutes -= 60;
}
}
wait (1);
}
}
PANEL* timer_pan =
{
layer = 15;
digits(100, 20, 2, arial_font , 1, hours);
digits(150, 20, 2, arial_font , 1, minutes);
digits(200, 20, 2, arial_font , 1, seconds);
digits(250, 20, 3, arial_font , 1, milliseconds);
flags = SHOW;
}
```

make an entity appear at a spot on the level where a user clicks

```
* pointer_tga = "pointer.tga";
STRING* target_mdl = "dot.mdl";
function adjust_target()
{
my.z += 5; // play with the target entity here
}
function create_target()
{
VECTOR pos1, pos2;
pos1.x = mouse_pos.x;
pos1.y = mouse_pos.y;
pos1.z = 0;
vec_for_screen (pos1, camera);
pos2.x = mouse_pos.x;
pos2.y = mouse_pos.y;
pos2.z = 20000; // use a big value here
vec_for_screen (pos2, camera);
c_trace (pos1.x, pos2.x, IGNORE_PASSABLE); // now "target" holds the coordinates of the hit point
ent_create (target_mdl, target, adjust_target); // create the entity at the "target" position
}
function mouse_startup()
{
on_mouse_left = create_target; // create a target entity when the player presses the left mouse button
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

entity move to the specified location

```
ENTITY* goal;
action npc_goal() // attach this action to the destination entity
{
goal = my;
}
action goal_tracker() // attach this action to your NPC character
{
while (!goal) {wait (1);} // wait until the goal entity is loaded
VECTOR temp, temp_angle;
var npc_speed = 5;
var covered_dist, i;
while (1)
{
my.skill10 += 6 * time_step; // 6 gives the animation speed
vec_set (temp.x, my.x); // trace 10,000 quants below the npc entity
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) + 20; // play with 20
temp.x = npc_speed * time_step;
temp.y = 0;
ent_animate(my, "walk", my.skill10, ANM_CYCLE);
covered_dist = c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
if (covered_dist < 0.1) // the npc is stuck?
{
my.pan += 90 - random(180); // then add a random angle to its pan angle
i = 0;
while (i < 10) // walk in the new direction for 10 frames, play with 10
{
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
ent_animate(my, "walk", my.skill10, ANM_CYCLE);
i++;
wait (1);
}
}
else // the npc can move? Then rotate it towards the goal again!
{
vec_set(temp_angle, goal.x);
vec_sub(temp_angle, my.x);
vec_to_angle(my.pan, temp_angle);
my.tilt = 0;
}
if (vec_dist(goal.x, my.x) < 50) // the npc has found the goal entity?
break; // then get out of the while loop!
wait (1);
}
// the goal was reached here
ent_animate(my, "stand", 0, 0); // switch to "stand"
}
```

follow the direction of the path smoothly

```
#define car_speed skill55
SOUND* aiskids1_wav = "aiskids1.wav";
SOUND* carai1_wav = "carai1.wav";
ENTITY* dummy1;
ENTITY* carai1_dummy;
ENTITY* carai1;
action car_ai()
{
carai1 = my;
var distance = 0;
var previous_pan;
var following_pan;
var min_speed = 20;
var max_speed;
var carai_engine;
var engine_factor;
var dist_z;
var skids_once = 0;
VECTOR last_pos[3];
VECTOR direction[3];
VECTOR temp[3];
max_speed = 35;
dummy1 = ent_create(NULL, nullvector, NULL);
// create another (carai1_dummy) dummy entity that will move on the path
// the ai car will use the same x, y, pan, tilt and roll with the carai1_dummy entity that moves on the path
// but will compute its own z using c_trace; this way, the nodes don't have to be place precisely above the ground
carai1_dummy = ent_create(NULL, nullvector, NULL);
path_set(dummy1, "path_001"); // make sure to name your path this way
engine_factor = 2 + random(2); // start with a random engine sound frecquency each and every time
carai_engine = ent_playloop(carai1, carai1_wav, 300); // start playing the car ai engine sound in a loop
while(1)
{
previous_pan = carai1.pan;
// place the carai1 entity on the path
path_spline(dummy1, carai1_dummy.x, distance);
distance += dummy1.car_speed * time_step;
// let carai1 look ahead
vec_diff(direction, carai1_dummy.x, last_pos);
vec_to_angle(carai1_dummy.pan, direction);
vec_set(last_pos, carai1_dummy.x);
carai1.pan = carai1_dummy.pan;
carai1.tilt = carai1_dummy.tilt;
carai1.roll = carai1_dummy.roll;
carai1.x = carai1_dummy.x;
carai1.y = carai1_dummy.y;
vec_set (temp.x, carai1_dummy.x);
temp.z -= 2000;
dist_z = c_trace(carai1_dummy.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | IGNORE_MODELS); // ignore the "real" carai
carai1.z = carai1_dummy.z - dist_z + 35; // 35 = experimental value
// sets a variable car ai engine frequency, depending on the speed of the car
snd_tune(carai_engine, 0, dummy1.car_speed * engine_factor, 0);
wait(1);
following_pan = carai1.pan;
if (abs(following_pan - previous_pan) > 1) // sudden direction change? Then lower the speed of the AI car
{
dummy1.car_speed -= 4 * time_step;
if (skids_once == 0)
{
skids_once = 1;
if (random(1) > 0.9)
{
ent_playsound(dummy1, aiskids1_wav, 3000);
}
}
}
else
{
skids_once = 0;
dummy1.car_speed += 2 * time_step;
}
dummy1.car_speed = clamp(dummy1.car_speed, min_speed, max_speed);
}
}
```

a bitmap stay full screen, regardless of the screen resolution

```
BMAP* splash_tga = "splash.tga"; // splash screen bitmap
PANEL* splash_pan =
{
bmap = splash_tga;
layer = 15;
flags = SHOW;
}
// makes the panel fill the entire screen, regardless of the screen size and / or the bitmap resolution
function panel_startup()
{
while (1)
{
splash_pan.scale_x = screen_size.x / bmap_width(splash_tga);
splash_pan.scale_y = screen_size.y / bmap_height(splash_tga);
wait (1);
}
}
```

light moving on a path

```
var entity_speed = 3;
var movement_enabled = 0;
var dist_to_node;
var current_node = 1;
VECTOR temp_angle;
VECTOR pos_node; // stores the position of the node
function move_target()
{
while(1)
{
if(movement_enabled)
{
entity_speed = minv(5, entity_speed + 0.5 * time_step);
c_move(my, vector(entity_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x));
}
wait(1);
}
}
action light_follows_path() // attach this action to your light model
{
// set (my, INVISIBLE | PASSABLE); // if you don't want the light to be visible
vec_set(my.blue, vector(255, 255, 255)); // set a white color for the light
my.lightrange = 300; // and a range of 300 quants
move_target();
result = path_scan(me, my.x, my.pan, vector(360, 0, 500)); // scan the area
if (result) {movement_enabled = 1;}
path_getnode (my, 1, pos_node, NULL);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate towards the node
while(1)
{
dist_to_node = vec_dist(my.x, pos_node);
if(dist_to_node < 50) // close to the node?
{
current_node = path_nextnode(my, current_node, 1);
if (!current_node) {current_node = 1;} // reached the end of the path? Then start over!
path_getnode (my, current_node, pos_node, NULL);
}
wait(1);
}
}
```

jump-pad snippet like in Quake

```
SOUND* jump_wav = "jump.wav";
function jump_now()
{
snd_play(jump_wav, 80, 0);
my.event = NULL; // don't trigger several jumps at once
var vertical_force = 100; // play with 100
while(vertical_force > 1) //while we have some jump force left
{
if (player) // we are controlling the player?
{
c_move(player, vector(0, 0, vertical_force * time_step), nullvector, 0);
}
vertical_force -= 2.5 * time_step; // play with 2.5
wait(1);
}
my.event = jump_now; // allow the jumps again
}
action jumping_pad() // attach this action to your jump pad model / wmb entity
{
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY);
my.event = jump_now;
}
action players_code() // need special player code as well
{
var movement_speed = 10; // movement speed
VECTOR temp;
set (my, INVISIBLE); // 1st person player
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
temp.z = -15 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

a 3D menu, with 3D buttons that can be clicked

```
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function start_game()
{
printf("Starts a new game");
}
function new_game()
{
my.emask |= ENABLE_CLICK;
my.event = start_game;
}
function end_game()
{
sys_exit (NULL); // shut down the engine
}
function quit_game()
{
my.emask |= ENABLE_CLICK;
my.event = end_game;
}
function main()
{
vec_set(screen_color, vector(200, 0, 0)); // make the background color blue
fps_max = 70;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
level_load (NULL); // load an empty level
wait (2); // wait a bit
camera.pan = 90; // point the camera in the right direction
ent_create("background.mdl", vector (0, 200, 0), NULL); // create the background model
ent_create("newgame.mdl", vector (0, 135, 30), new_game); // create the "New game" model
ent_create("quitgame.mdl", vector (0, 135, -30), quit_game); // create the "Quit game" model
}
```

a crosshair who will rotate the camera and also move around the center of the screen as I move the mouse around

```
BMAP* pointer_tga = "pointer.tga";
action players_code() // attach this action to your player
{
var movement_speed = 10; // movement speed
VECTOR temp;
player = my; // I'm the player
set (my, INVISIBLE); // 1st person player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) + 15; // play with 15
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
function mouse_startup()
{
mouse_mode = 1;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
```

a particle that can rotate together with the entity that it is attached to

```
BMAP* effect_tga = "snow.tga";
function my_effect(PARTICLE *p)
{
p.lifespan = 1; // kill the particle right after its creation
p.alpha = 50;
p.bmap = effect_tga;
p.size = 5;
p.flags |= (BRIGHT | MOVE);
p.event = NULL;
}
action rotating_object() // just an example
{
VECTOR temp;
while (1)
{
my.pan += 3 * time_step;
my.tilt += 4 * time_step;
my.roll += 2 * time_step;
vec_for_vertex(temp, my, 10); // generate particles from the 10th vertex of the entity
effect(my_effect, 1, temp.x, nullvector);
wait (1);
}
}
```

check the values for my joystick

```
void main()
{
video_screen = 1; // start in full screen mode
level_load (test_wmb);
fps_max = 70;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
while (1)
{
while (!key_any) {wait (1);} // wait until a key is pressed
while (key_any) {wait (1);} // wait until a key is released
wait (1);
}
}
PANEL* scancodes_pan =
{
digits = 30, 50, 5, *, 1, key_lastpressed; // display the scan code for the last key that was pressed
flags = SHOW;
}
```

collision camera from Morrowing

```
var camera_distance = 200; // distance from the player to the camera in 3rd person mode
var camera_height = 240; // distance from the origin of the player to the camera (on the z axis)
var distance_traced; // distance that is return by a trace instruction
var camera_mode; // starts in 3rd person mode, changes to 1 for first person
var stop_player = 0; // set it to 1 to stop the player until stop_player is set to 0 again
var player_speed = 15;
VECTOR temporary_distance;
SOUND* step_wav = "step.wav";
function avoid_obstacles();
function first_person_camera();
function third_person_camera();
#define shield skill1
#define mace skill2
#define bodyarmor skill3
#define flare skill4
#define strength skill10 // use these skills to store player's abilities
#define experience skill11
#define mana skill12
#define health skill40 // use skill40 to store health for the player and its enemies
action player1()
{
on_f1 = first_person_camera;
on_f3 = third_person_camera;
VECTOR temp;
VECTOR trace_coords;
player = my; // I'm the player
set (my, TRANSLUCENT); // the player is transparent
my.alpha = 100; // but opaque if the camera doesn't run into obstacles
my.skill41 = camera_height; // and its height
my.skill42 = camera_distance; // we store the distance to the camera
camera_mode = 3; // the game starts with the camera in 3rd person mode
my.health = 100;
while (my.health > 0)
{
while (stop_player == 1) {wait (1);}
// player's pan is controlled by the mouse and the "A" and "D" keys
player.pan -= 10 * mouse_force.x * time_step - 1.5 * (key_a - key_d);
camera.x = player.x - camera_distance * cos(player.pan); // keep the camera behind the player
camera.y = player.y - camera_distance * sin(player.pan); // at the distance given by camera_distance
camera.z = player.z + camera_height + 0.8 * sin(my.skill46 * 3.6); // and above the player
camera.pan = player.pan; // the camera has the same pan angle with the player
camera.tilt += 7 * mouse_force.y * time_step; // and can tilt freely
camera_distance = minv(maxv(camera_distance, 5), 500); // camera_distance can have values between 5 and 500
if (key_w + key_s > 0) // if the player is walking
```

```
{
ent_animate(my, "walk", my.skill46, ANM_CYCLE); // play its "walk" frames animation
// the animation speed increases when the player presses the "shift" key
my.skill46 += 5 * (1 + key_shift * 0.7) * time_step;
my.skill46 %= 100; // loop the animation
}
else // if the player is standing
{
my.skill47 = 0; // reset the skill that stores the distance needed for the step sound (not really needed)
ent_animate(my, "stand", my.skill48, ANM_CYCLE); // play the "stand" frames animation
my.skill48 += 2 * time_step; // "stand" animation speed
my.skill48 %= 100; // loop animation
}
if (camera_mode == 3) // if we are in 3rd person mode
{
avoid_obstacles(); // run the function that avoids camera collisions with the relief
}
vec_set (temp, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = - c_trace (my.x, temp, IGNORE_ME | IGNORE_PASSABLE | USE_BOX); // and adjust its height accordingly, placing its feet on the ground
// allow the player to move using the "W" and "S" keys; the speed increases to 200% if the player presses the "shift" key
temp.x = player_speed * (key_w - key_s) * (1 + 1 * key_shift) * time_step;
temp.y = 0;
my.skill47 += c_move (my, temp, nullvector, IGNORE_PASSABLE);
if (my.skill47 > 250) // play with 250 (here we've got a step sound every 50 quants)
{
snd_play(step_wav, 15, 0);
my.skill47 = 0;
}
wait (1);
}
// the player is dead here
my.skill40 = 0; // use the same skill40 (health) to control the "death" animation
while (my.skill40 < 90) // don't play all the animation frames because the result doesn't look too good
{
ent_animate(my, "death", my.skill40, NULL); // play the "death" animation
my.skill40 += 2 * time_step; // "death" animation speed
camera.roll -= 0.3 * time_step;
camera.tilt += 0.8 * time_step;
wait (1);
}
set (my, PASSABLE); // the corpse will be passable from now on
}
function avoid_obstacles()
{
vec_set (temporary_distance.x, camera.x);
temporary_distance.z -= 50; // sets a position closer to the feet of the player; 50 = experimental value
distance_traced = c_trace (player.x, temporary_distance.x, IGNORE_ME | IGNORE_PASSABLE); // trace between the player and temporary_distance
if (distance_traced == 0) // no obstacles on the way?
{
my.alpha = minv(100, my.alpha + 3 * time_step); // then increase player's alpha up to 100
if (player.alpha == 100)
{
reset(player, TRANSLUCENT);
}
else
{
set(player, TRANSLUCENT);
}
if (camera_distance < my.skill40) // if the camera got closer to the player
{
camera_distance += 1; // restore the initial camera_distance slowly
}
}
else // obstacles encountered?
{
distance_traced -= 2; // then bring the camera 2 quants closer to the player!
my.alpha = (distance_traced / (my.skill40 + 1)) * 100; // decrease player's alpha; don't allow a division by zero
camera.x = player.x - distance_traced * cos(camera.pan); // place the camera behind the player
camera.y = player.y - distance_traced * sin(camera.pan); // at the new distance given by distance_traced
}
}
function first_person_camera() // press "F1" to run this function
{
camera_distance = 0; // place the camera at player's position
camera_height = 150; // play with this value
set (player, INVISIBLE); // make the player model invisible
camera_mode = 1; // set the camera_mode variable to 1st person
}
function third_person_camera() // press "F3" to run this function
{
camera_height = player.skill41; // restore the height
camera_distance = player.skill42; // restore the distance from the player to the camera
reset (player, INVISIBLE); // and show the player
camera_mode = 3; // set the camera_mode variable to 3rd person
}
```

make a model transparent if the player is behind it (less than 60% of it is visible)? I need to make the buildings transparent if the player steps behind them

```
var temp_building;
ENTITY* building_ent;
action my_player() // attach this action to your player
{
var movement_speed = 10; // movement speed
VECTOR temp;
VECTOR trace_pos;
player = my; // I'm the player
set(my, FLAG2);
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = 0;
temp.z = 0;
my.pan += 4 * (key_a - key_d) * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
if (key_w + key_s > 0) // if the player is walking
{
ent_animate(my, "walk", my.skill46, ANM_CYCLE); // play its "walk" frames animation
my.skill46 += 5 * time_step;
my.skill46 %= 100; // loop the animation
}
else // if the player is standing
{
ent_animate(my, "stand", my.skill48, ANM_CYCLE); // play the "stand" frames animation
my.skill48 += 2 * time_step; // "stand" animation speed
my.skill48 %= 100; // loop the animation
}
vec_set(trace_pos.x, my.x);
trace_pos.z += 20; // play with this value
c_trace(trace_pos.x, camera.x, IGNORE_FLAG2 | USE_BOX);
if (you)
{
temp_building = handle(you);
set(you, TRANSLUCENT);
you.alpha = 60;
my.skill99 = 1;
}
else
{
if (my.skill99 == 1) // building_ent exists?
{
building_ent = ptr_for_handle(temp_building);
reset (building_ent, TRANSLUCENT);
}
}
wait (1);
}
}
function camera_startup() // camera code sample
{
VECTOR temp;
vec_set(camera.x, vector(0, 0, 1000));
while (!player) {wait (1);}
while (1)
{
vec_set(temp, player.x);
vec_sub(temp, camera.x);
vec_to_angle(camera.pan, temp);
wait (1);
}
}
```

read some names from a text file and sort them alphabetically

```
FONT* arial_font = "Arial#20";
TEXT* names_txt =
{
pos_x = 30;
pos_y = 50;
font(arial_font);
strings = 30; // stores up to 30 names
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
string = " "; // 20 characters
flags = visible;
}
function opendata_startup()
{
var file_handle;
var read_result = 0;
var i = 0;
while (!player) {wait (1);}
file_handle = file_open_read("names.txt");
while (read_result != -1) // read the data until the end of the file is reached
{
read_result = file_str_read(file_handle, (names_txt.pstring)[i]);
i += 1;
}
str_cpy((names_txt.pstring)[i-1], " "); // reset the last string
// all the names are read from the file and stored in the text strings here
file_close(file_handle);
while (!key_n) {wait (1);} // press the "N" key to sort the names
while (key_n) {wait (1);}
txt_sort(names_txt); // the names are sorted here
}
```

create a timer that runs faster that sys seconds? I'd like to have a clock that runs a day in 20...30 minutes.

```
STRING* clock_str = " "; // hh:mm ss = 8 chars
STRING* temp_str = " "; // 2 chars storing temporary data
FONT* arial_font = "Arial#20";
TEXT* clock_text =
{
layer = 20;
pos_x = 40;
pos_y = 30;
font(arial_font);
string(clock_str);
flags = visible;
}
function clock_startup()
{
// limit the frame rate to 60 fps, 1 second for our clock = 1 / 60 real seconds
fps_max = 60; // this clock runs 60 times faster than normal
var my_seconds = 0;
var my_minutes = 0;
var my_hours = 0;
var my_days = 0; // not used, but might come in handy someday
while (1)
{
my_seconds += 1;
if (my_seconds == 60) // a full minute was reached (in fact, a second has passed)?
{
my_seconds = 0;
my_minutes += 1;
}
if (my_minutes == 60) // a full hour was reached (in fact, a minute has passed)?
{
my_minutes = 0;
my_hours += 1;
}
if (my_hours == 24) // a full day has was reached (in fact, 24 minutes have passed)?
{
my_seconds = 0;
my_hours = 0;
my_days += 1;
}
str_for_num(clock_str, my_hours);
str_cat(clock_str, ":"); // add : to create hh:mm
if (my_minutes < 10) // add a "0" if the minutes are displayed with a digit
str_cat(clock_str, "0");
str_for_num(temp_str, my_minutes);
str_cat(clock_str, temp_str);
wait (1);
}
}
```

disable the window border (so that my panel is the border) and use the exit button

```
BMAP* pointer_tga = "pointer.tga";
BMAP* quit1_pcx = "quit1.pcx";
BMAP* quit2_pcx = "quit2.pcx";
STRING* test_wmb = "test.wmb";
function quit_game();
```

```
PANEL* my_pan =
{
layer = 15;
pos_x = 0;
pos_y = 0;
bmap = "mypanel.tga"; // the frame panel
button = 780, 0, quit2_pcx, quit1_pcx, quit2_pcx, quit_game, null, null;
flags = visible;
}
void main()
{
fps_max = 70;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
video_screen = 2; // start in window mode
video_window(NULL, NULL, 1, NULL);
level_load (test_wmb);
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function quit_game()
{
sys_exit(NULL);
}
```

a simple sky-cube

```
STRING* test_wmb = "test.wmb";
void main()
{
fps_max = 70;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
video_screen = 1; // start in full screen mode
level_load (test_wmb);
wait (3);
ent_createlayer("skycube+6.tga", SKY | CUBE | VISIBLE , 1);
}
```

a level loading progress indicator in my game

```
var loading_percentage;
STRING* test_wmb = "test.wmb";
FONT* arial_font = "Arial#20";
function loading_percent(factor)
{
loading_percentage = factor;
loading_percentage = minv(100, loading_percentage); // loaded: 101 / 100 would look bad on the screen
}
```

```
PANEL* time_pan =
{
layer = 15;
digits (300, 300, "Loaded: %.f / 100", arial_font, 1, loading_percentage);
flags = visible;
}
void main()
{
fps_max = 70;
video_mode = 7; // run in 800x600 pixels
video_depth = 32; // 32 bit mode
video_screen = 1; // start in full screen mode
on_level = loading_percent;
level_load (test_wmb);
while (loading_percentage < 100) {wait (1);} // the level is loaded here
reset (time_pan, VISIBLE); // so let's hide the panel
}
```

make my character visible to the camera even if there's a block between the camera and the character, blocking the view

```
action my_player() // attach this action to your player
{
var movement_speed = 10; // movement speed
VECTOR temp;
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = 0;
temp.z = 0;
my.pan += 4 * (key_a - key_d) * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
if (key_w + key_s > 0) // if the player is walking
{
ent_animate(my, "walk", my.skill46, ANM_CYCLE); // play its "walk" frames animation
my.skill46 += 5 * time_step;
my.skill46 %= 100; // loop the animation
}
else // if the player is standing
{
ent_animate(my, "stand", my.skill48, ANM_CYCLE); // play the "stand" frames animation
my.skill48 += 2 * time_step; // "stand" animation speed
my.skill48 %= 100; // loop the animation
}
if (c_trace(camera.x, player.x, USE_BOX)) // the player isn't visible?
{
set(player, ZNEAR);
}
else
{
reset(player, ZNEAR);
}
wait (1);
}
}
```

detect if my bouncing ball has collided with a particular wall

```
function bounce_off() // ball's event function
{
vec_to_angle(my.pan, bounce); // change the direction as expected from a "real" ball
my.pan += 5 - random(10); // add some randomness at each collision (don't allow the ball to get stuck)
if (you) // collided with another entity (maybe with our special wall)?
{
if (you.skill100 == 13579) // if the entity is our special wall indeed
{
```

```
beep(); // do what you need to do here
}
}
}
action my_ball() // attach this action to your ball
{
my.emask |= (ENABLE_BLOCK | ENABLE_ENTITY); // the object is sensitive to block and entity collisions
my.event = bounce_off;
my.skill100 = 13579; // set this skill to a weird value in order to differentiate the object from the others
while(1)
{
c_move(me, vector(5 * time_step, 0, 0), nullvector, 0); // move ahead, in the direction given by the pan angle
wait(1);
}
}
action special_wall() // attach this action to your special wall
{
my.skill100 = 13579; // we will test this value in ball's event function
}
```

make a weapon switch snippet

```
ENTITY* weapon1 = // first weapon
{
type = "weapon1.mdl";
layer = 5;
x = 145; // place this gun 145 quants ahead of the view
y = -70; // 70 quants to the right
z = -30; // and 30 quants below the center of the screen
pan = 7;
tilt = 10;
flags2 = VISIBLE; // the first weapon is visible by default
}
ENTITY* weapon2 = // second weapon
{
type = "weapon2.mdl";
layer = 5;
x = 135; // place this gun 135 quants ahead of the view
y = -30; // 30 quants to the right
z = -20; // and 20 quants below the center of the screen
pan = 3;
tilt = 10;
}
ENTITY* weapon3 = // third weapon
{
type = "weapon3.mdl";
layer = 5;
x = 150; // place this gun 150 quants ahead of the view
y = 20; // 20 quants to the right
z = 10; // and 10 quants above the center of the screen
pan = -4;
tilt = -10;
}
function weapons_startup()
{
var weapon_number = 1;
while (1)
{
if (mickey.z < -1)
{
weapon_number %=3;
weapon_number += 1; // don't allow weapon_number to be smaller than 1 or bigger than 3
if (weapon_number == 1)
{
weapon1.flags2 |= VISIBLE;
weapon2.flags2 &= ~VISIBLE;
weapon3.flags2 &= ~VISIBLE;
}
if (weapon_number == 2)
{
weapon1.flags2 &= ~VISIBLE;
weapon2.flags2 |= VISIBLE;
weapon3.flags2 &= ~VISIBLE;
}
if (weapon_number == 3)
{
weapon1.flags2 &= ~VISIBLE;
weapon2.flags2 &= ~VISIBLE;
weapon3.flags2 |= VISIBLE;
}
}
if (mickey.z > 1)
{
weapon_number %=3;
weapon_number += 1; // don't allow weapon_number to be smaller than 1 or bigger than 3
if (weapon_number == 3)
{
weapon1.flags2 |= VISIBLE;
weapon2.flags2 &= ~VISIBLE;
weapon3.flags2 &= ~VISIBLE;
}
if (weapon_number == 2)
{
weapon1.flags2 &= ~VISIBLE;
weapon2.flags2 |= VISIBLE;
weapon3.flags2 &= ~VISIBLE;
}
if (weapon_number == 1)
{
weapon1.flags2 &= ~VISIBLE;
weapon2.flags2 &= ~VISIBLE;
weapon3.flags2 |= VISIBLE;
}
}
wait (1);
}
}
```

load skins on my entities at runtime

```
action my_entity()
{
my.skill1 = 100; // the entity starts with 100 health points
my.skill2 = 0;
while (my.skill1 > 0)
{
// decrease the health (skill1) at all times; this should be done by the enemies under normal conditions
my.skill1 -= 1 * time_step; // 1 gives the health decreasing speed
my.skill22 += 4 * time_step; // 4 gives the "stand" animation speed
ent_animate(my, "stand", my.skill22, ANM_CYCLE);
if (my.skill1 < 20) // the entity is almost dead?
{
if (my.skill2 == 0)
{
ent_morphskin(my, "bruised.pcx"); // then replace the skin with another one
my.skill2 = 1; // replace the skin only once; no need to waste the cpu power
}
}
wait (1);
}
my.skill22 = 0;
while (my.skill22 < 100)
{
```

```
my.skill22 += 5 * time_step;
ent_animate(my, "death", my.skill22, NULL);
wait (1);
}
set (my, PASSABLE); // the corpse is passable now
}
```

Camera script

```
var camera_number = 1;
function camera_startup()
{
while (!player) {wait (1);}
while (1)
{
if (key_1)
{
camera_number = 1;
}
if (key_2)
{
camera_number = 2;
}
if (key_3)
{
camera_number = 3;
}
if (key_4)
{
camera_number = 4;
}
if (key_5)
{
camera_number = 5;
}
if (camera_number == 1) // top view
{
camera.x = player.x;
camera.y = player.y;
camera.z = player.z + 300; // play with this value
camera.pan = player.pan;
camera.tilt = -90;
camera.roll = 0;
}
if (camera_number == 2) // side view
{
camera.x = player.x + 200 * sin(player.pan); // 200 = distance
camera.y = player.y - 200 * cos(player.pan) ; // same value here
camera.z = player.z + 10; // a little higher
camera.pan = player.pan + 90; // face the player
camera.tilt = 0;
camera.roll = 0;
}
if (camera_number == 3) // isometric view
{
camera.x = player.x - 200 * cos(player.pan); // 200 = distance
camera.y = player.y - 200 * sin(player.pan); // same value here
camera.z = player.z + 200; // above the player
camera.pan = player.pan;
camera.tilt = -30; // look down at the player
camera.roll = 0;
}
if (camera_number == 4) // aidemo view
{
// 200 = front distance, 150 = side distance
camera.x = player.x + 200 * cos(player.pan) + 150 * sin(player.pan);
camera.y = player.y + 200 * sin(player.pan) - 150 * cos(player.pan);
camera.z = player.z;
camera.pan = player.pan + 130; // look back to the player
camera.tilt = 0;
camera.roll = 0;
}
if (camera_number == 5) // 1st person view
{
camera.x = player.x;
camera.y = player.y;
camera.z = player.z + 40; // play with 40
camera.pan = player.pan;
camera.tilt = player.tilt;
camera.roll = player.roll;
}
wait (1);
}
}
action player_code() // attach this action to your player
{
var movement_speed = 10; // movement speed
var anim_percentage;
VECTOR temp;
player = my; // I'm the player
while (1)
{
my.pan -= 7 * mouse_force.x * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) + 20; // play with 20
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
if (!key_w && !key_s) // the player isn't moving?
{
ent_animate(my, "stand", anim_percentage, ANM_CYCLE); // play the "stand" animation
anim_percentage += 5 * time_step; // 5 = animation speed
}
else // the player is moving?
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE); // play the "walk" animation
anim_percentage += 8 * time_step; // 8 = animation speed
}
wait (1);
}
}
```

to have a bird in my game. It should move along the x axis, and when it is above the player it should drop some models on player's head

```
STRING* bomb_mdl = "bomb.mdl";
function set_frames()
{
fps_max = 75; // limit the frame rate to 75 fps
}
function bomb_event()
{
// do something bad to the player here ;)
wait (1);
ent_remove(my);
}
function move_bomb()
{
my.emask |= (ENABLE_BLOCK | ENABLE_IMPACT | ENABLE_ENTITY);
my.event = bomb_event;
while (my)
{
c_move(my, vector(0, 0, -10 * time_step), nullvector, IGNORE_PASSABLE);
wait (1);
}
}
```

```
action bird_enemy()
{
var anim_percentage;
var init_x;
init_x = my.x; // store the initial x value for the bird
while (!player) {wait (1);} // wait until the player model is loaded
while (1)
{
// move up to 1000 quants away from the initial position
while (my.x < init_x + 1000)
{
ent_animate(my, "fly", anim_percentage, ANM_CYCLE); // play the "fly" animation
anim_percentage += 8 * time_step; // 8 = animation speed
my.x += 15 * time_step;
// the player is close to the bird on the x axis? (play with 200)
// then drop a bomb every second!
if ((abs(my.x - player.x) < 200) && (total_frames % 75 == 1))
{
// create the bomb 20 quants below the bird's origin
ent_create (bomb_mdl, vector(my.x, my.y, my.z - 20), move_bomb);
}
wait (1);
}
my.pan += 180; // the bird will turn back
// now return to the initial position
while (my.x > init_x)
{
ent_animate(my, "fly", anim_percentage, ANM_CYCLE); // play the "fly" animation
anim_percentage += 8 * time_step; // 8 = animation speed
my.x -= 15 * time_step;
// the player is close to the bird on the x axis? (play with 200)
// then drop a bomb every second!
if ((abs(my.x - player.x) < 200) && (total_frames % 75 == 1))
{
// create the bomb 20 quants below the bird's origin
ent_create (bomb_mdl, vector(my.x, my.y, my.z - 20), move_bomb);
}
wait (1);
}
my.pan += 180;
// restore the proper pan angle
wait (1);
}

}
```

load the desired level by clicking on its corresponding button on a panel

```
BMAP* levels_tga = "levels.tga";
BMAP* pointer_tga = "pointer.tga";
BMAP* level11_tga = "level11.tga";
BMAP* level12_tga = "level12.tga";
BMAP* level21_tga = "level21.tga";
BMAP* level22_tga = "level22.tga";
BMAP* level31_tga = "level31.tga";
BMAP* level32_tga = "level32.tga";
STRING* level1_wmb = "level1.wmb";
STRING* level2_wmb = "level2.wmb";
STRING* level3_wmb = "level3.wmb";
function load_level1();
function load_level2();
function load_level3();
PANEL* levels_pan =
{
bmap = levels_tga;
layer = 10;
pos_x = 240;
pos_y = 180;
button (100, 70, level11_tga, level11_tga, level12_tga, load_level1, NULL, NULL);
button (100, 110, level21_tga, level21_tga, level22_tga, load_level2, NULL, NULL);
button (100, 150, level31_tga, level31_tga, level32_tga, load_level3, NULL, NULL);
flags = VISIBLE;
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function load_level1()
{
while (mouse_left) {wait (1);}
level_load (level1_wmb);
// remove the comment from the following line if you want to hide the panel after loading the level
// reset(levels_pan, VISIBLE);
}
function load_level2()
{
while (mouse_left) {wait (1);}
level_load (level2_wmb);
// remove the comment from the following line if you want to hide the panel after loading the level
// reset(levels_pan, VISIBLE);
}
function load_level3()
{
while (mouse_left) {wait (1);}
level_load (level3_wmb);
// remove the comment from the following line if you want to hide the panel after loading the level
// reset(levels_pan, VISIBLE);
}
```

touch an entity and makes a text appear

```
BMAP* pointer_tga = "pointer.tga";
TEXT* message_txt =
{
pos_x = 300;
pos_y = 50;
string("Hey! Move the mouse away from me!");
}
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function display_text()
```

```
{
set (message_txt, VISIBLE);
wait (-5);
reset (message_txt, VISIBLE);
}
action my_entity()
{
my.emask |= (ENABLE_TOUCH);
my.event = display_text;
}
```

create a metallic reflection

```
action reflections() // attach this action to your model(s)
{
my.material = mat_metal; // use this material to achieve a quick metallic reflection effect
while (1)
{
my.pan += 5 * time_step; // make the object spin
wait (1);
}
}
```

a simple top down space game and I would like to have the player respawn after its death

```
players_health = 100;
STRING* ship_mdl = "ship.mdl";
PANEL* health_pan = // displays player's health
{
layer = 15;
digits(700, 20, 3 ,* , 1, players_health);
flags = VISIBLE;
}
function damage_player()
{
players_health -= 0.5 * time_step;
}
function control_ship() // control player's ship using the mouse
{
player = my;
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK);
my.event = damage_player;
var temp_speed;
VECTOR player_speed;
while (players_health > 0)
{
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 1000; // place the camera 1000 quants above the ship
camera.pan = my.pan;
camera.tilt = -90; // look down at the ship
my.pan += 4 * mouse_force.x * time_step; // 4 gives the rotation speed
temp_speed = 20 * mouse_force.y; // 20 gives the movement speed
my.skill1 = temp_speed * time_step + maxv(1 - time_step * 0.1, 0) * my.skill1; // 0.1 gives the friction
player_speed.x = my.skill1 * time_step;
player_speed.y = 0;
player_speed.z = 0;
c_move(my, player_speed, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
function respawn_startup()
{
while (1)
{
// place the ship at x = 100, y = 200, z = 50 in the level (just an example)
player = ent_create (ship_mdl, vector (100, 200, 50), control_ship);
// wait until the player dies by colliding with the walls; use enemies to damage the player as well
while (players_health > 0) {wait (1);}
wait (-3); // wait for 3 seconds before respawning the player
players_health = 100; // restore player's health
}
}
```

play a movie inside the level when the player comes close to a certain entity

```
// attach this action to a sprite that has the same size with your movie
action movie_gate()
{
// wait until the player entity is loaded in the level
while (!player) {wait (1);}
// wait until the player comes close to the exit gate
while (vec_dist(player.x, my.x) > 150) {wait (1);}
// now play the movie file
media_play("mymovie.avi", bmap_for_entity (my, 0), 100); // use your own movie name here
}
```

random passwords

```
var password_length = 8; // generate passwords that contain 8 characters, play with this number
STRING* password_str = "#30"; // stores passwords with up to 30 characters
STRING* temp_str = " ";
FONT* arial_font = "Arial#20";
TEXT* password_txt =
{
pos_x = 20;
pos_y = 20;
font = arial_font;
string(password_str);
flags = VISIBLE;
}
// press any key to generate a new password
function passwords_startup()
{
var i, temp;
randomize();
while (1)
{
while (key_any) {wait (1);} // wait until all the keys are released
while (!key_any) {wait (1);} // wait until a key is pressed
str_cpy(password_str, "#30"); // reset password_str
i = 0;
while (i < password_length) // generate a number of 8 characters for our password
{
// generate a random ascii code (visible, printable characters only)
temp = integer(random(95)) + 32;
str_for_asc(temp_str, temp); // convert the random number to its string equivalent
str_cat(password_str, temp_str);
i++;
}
}
}
```

npc to move from A to B, stopping at certain points on the path for a specified number of seconds.

```
ENTITY* my_target;
action patroller()
{
VECTOR temp[3];
while (1)
{
// scan for camera positions that are placed up to 1000 quants away from this entity
c_scan(my.x, my.pan, vector(360, 60, 1000), IGNORE_ME | SCAN_ENTS | SCAN_LIMIT);
if (you) // a target was detected?
{
my_target = you; // the closest position to the npc is set to "you" by c_scan
my_target.scale_z = 4;
vec_set(temp, my_target.x);
vec_sub(temp, my.x);
vec_to_angle(my.pan, temp); // rotate the patroller towards the position
my.tilt = 0; // but don't allow it to change its tilt angle
// the patroller looks at its target now
while (vec_dist (my.x, my_target.x) > 50) // the patroller moves until it comes close to the target
{
my.skill22 += 5 * time_step; // 5 gives the "walk" animation speed
c_move(my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE); // 5 = movement speed
ent_animate(my, "walk", my.skill22, ANM_CYCLE);
wait (1);
}
// the patroller has come close to target here
my.skill99 = 0;
while (my.skill99 < my_target.skill1)
{
my.skill99 += time_step / 16;
ent_animate(my, "stand", my.skill22, ANM_CYCLE);
my.skill22 += 3 * time_step; // 3 gives the "stand" animation speed
wait (1);
}
ent_remove(my_target); // remove the current target and prepare for the following target
}
else // reached the end of the path?
{
ent_animate(my, "stand", my.skill22, ANM_CYCLE); // then default to standing
my.skill22 += 3 * time_step; // 3 gives the "stand" animation speed
}
wait (1);
}
}
action target_init()
{
set (my, PASSABLE | INVISIBLE); // player's movement code should ignore passable entities
my.emask |= ENABLE_SCAN; // the target entity is sensitive to c_scan instructions
}
```

targets (sprites) that rotate around their pan angles when they are shot

```
SOUND* bullet_wav = "bullet.wav";
function fire_bullets()
{
VECTOR trace_coords[3];
vec_set(trace_coords.x, vector(5000, 0, 0)); // trace 5000 quants in front of the player
// rotate "trace_coords"
vec_rotate(trace_coords.x, camera.pan);
vec_add(trace_coords.x, player.x); // add the resulting vector to player's position
snd_play(bullet_wav, 70, 0); // play the bullet sound
if (c_trace (player.x, trace_coords.x, IGNORE_ME + USE_BOX) > 0) // the "c_trace" ray has hit something?
{
if (you) // and the hit object is an entity?
{
if (you.skill1 == 99) // and the hit entity has its skill1 set to 99?
{
you.skill50 = 30; // 30 = maximum rotation speed
}
}
}
}
action players_code()
```

```
{
on_mouse_left = fire_bullets;
player = my; // I'm the player
set (my, INVISIBLE); // no need to see player's model in 1st person mode
while (1)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, GLIDE);
vec_set (camera.x, player.x); // use player's x and y for the camera as well
camera.z += 30; // place the camera 30 quants above the player on the z axis (approximate eye level)
camera.pan -= 5 * mouse_force.x * time_step; // rotate the camera around by moving the mouse
camera.tilt += 3 * mouse_force.y * time_step; // on its x and y axis
player.pan = camera.pan; // the camera and the player have the same pan angle
wait (1);
}
}
action shooting_target()
{
my.skill1 = 99; // this entity will rotate around its axis
while (1)
{
while (my.skill50 <= 0) {wait (1);} // the entity waits until it is shot by the player
while (my.skill50 > 0)
{
my.pan += my.skill50 * time_step; // rotate the entity around its pan angle
my.skill50 -= 0.5 * time_step; // decrease the rotation speed as the time passes
wait (1);
}
}
}
```

have intelligent NPCs that walk, stop in front of walls, and then change their directions and move away

```
action intelligent_npc() // stops in front of the wall, and then changes its direction
{
VECTOR front_pos;
var distance_covered;
while (1)
{
vec_set(front_pos.x, vector(50, 0, 0)); // compute a vector that's 50 quants in front of the player
// rotate "front_pos" in the direction (angles) given by the entity
vec_rotate(front_pos.x, my.pan);
vec_add(front_pos.x, my.x); // add the resulting vector to entity's position
if (c_content(front_pos.x, 0) == 1) // front_pos isn't touching any walls?
{
// 5 gives the movement speed
distance_covered = c_move(my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.skill22 += 5 * time_step; // 5 gives the "walk" animation speed
ent_animate(my, "walk", my.skill22, ANM_CYCLE);
}
else // the player is close to a wall now?
{
my.skill99 = 0;
while (my.skill99 < 5) // stays in front of the wall for 5 seconds
{
my.skill99 += time_step / 16;
ent_animate(my, "stand", my.skill22, ANM_CYCLE);
my.skill22 += 3 * time_step; // 3 gives the "stand" animation speed
wait (1);
}
my.skill99 = my.pan;
my.skill99 += random(180);
while (my.pan < my.skill99)
{
my.pan += 5 * time_step;
wait (1);
}
}
wait (1);
}
}
```

health panel above each entity

```
#define health skill20
function health_indicator()
{
set (my, PASSABLE);
while (you)
{
vec_set (my.x, you.x);
my.z += 60;
my.scale_x = you.health * 0.1; // skill20 aka "health" stores the health for each entity
wait (1);
}
}
action my_enemy() // simple enemy action, makes it rotate in a circle
{
my.health = 10 + random(90);
ent_create ("health.pcx", nullvector, health_indicator);
while (1)
{
c_move(my, vector(5 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE); // 5 = movement speed
my.skill22 += 5 * time_step; // 5 gives the "walk" animation speed
my.pan += 1 * time_step;
ent_animate(my, "walk", my.skill22, ANM_CYCLE);
wait (1);
}
}
function randomize_startup()
{
randomize();
}
```

a solution that allows me to check if certain entities are visible in the current camera view

```
var object_id = -1;
var visible_objects[100]; // displays up to 100 visible objects
var total_visible;
action visible_or_not() // attach this action to all the object that need to be tracked
{
// put the rest of your code here
// .............................
VECTOR my_pos[3];
object_id += 1;
my.skill99 = object_id; // don't use skill99 for something else
while (1)
{
// put the rest of your code here
// .............................
vec_set(my_pos.x, my.x);
if (vec_to_screen(my_pos.x, camera)) // if this entity is visible on the screen
{
visible_objects[my.skill99] = 1;
}
else // the entity isn't visible on the screen
{
visible_objects[my.skill99] = 0;
}
wait (1);
}
}
function total_startup()
{
var i;
while (1)
{
total_visible = 0;
for(i = 0; i < 100; i++)
{
total_visible += visible_objects[i];
}
wait (1);
}
}
PANEL* entities_pan =
{
layer = 15;
digits(20, 20, 3, *, 1, total_visible);
flags = visible;
}
```

three lives and every time player's health hits zero, one of the lives goes down and the game reloads. If no lives are left, the game should jump to a game over panel

```
PANEL* gameover_pan =
{
layer = 15;
pos_x = 300;
pos_y = 200;
bmap = "gameover.pcx";
}
action player_destroyer() // kills the player if the player comes too close
{
set(my, PASSABLE);
while (1)
{
if (player) // the player exists?
{
if (vec_dist(player.x, my.x) < 100) // the player has come close to the destroyer object?
{
player.skill99 -= 5 * time_step; // then decrease player's health
}
}
wait (1);
}
}
function players_action()
{
VECTOR temp;
var movement_speed = 10; // movement speed
set (my, INVISIBLE); // 1st person player
my.skill99 = 100; // player's skill99 stores its health
while (my.skill99 > 0)
{
my.pan -= 7 * mouse_force.x * time_step;
camera.x = my.x;
camera.y = my.y;
camera.z = my.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = my.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX) - 2; // play with 2
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
player = NULL;
wait (1);
ent_remove(my); // remove the player if it is dead
}
function init_startup()
{
wait (-1); // wait until the level is loaded
player = ent_create("guard.mdl", vector (10, 10, 70), players_action); // first life
while (player) {wait (1);} // wait until the player disappears from the level
wait (3);
player = ent_create("guard.mdl", vector (10, 10, 70), players_action); // create it again (that's the second life)
while (player) {wait (1);} // wait until the player disappears from the level
wait (3);
player = ent_create("guard.mdl", vector (10, 10, 70), players_action); // create it again (that's the third life)
wait (3);
while (player) {wait (1);} // wait until the player disappears from the level
// the game is over here, so let's display the "Game Over" panel
set(gameover_pan, VISIBLE);
while (key_any) {wait (1);} // wait until the keys are released
while (!key_any) {wait (1);} // wait until any key is pressed
sys_exit(NULL); // and now let's shut down the engine
}
```

display several parameters for my entities on the screen (for debugging purposes).

```
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function debug_startup()
{
while (1)
{
if(mouse_ent) // the mouse touches an entity?
watched = mouse_ent; // then display information about that entity!
wait (1);
}
}
```

jump with a character

```
action player_code()
{
VECTOR temp[3];
VECTOR movement_speed[3]; // player's movement speed
var anim_percentage; // animation percentage
var jump_percentage; // animation percentage for jumping
var distance_to_ground; // the distance between player's origin and the ground
var jump_height;
var reached_height;
player = my; // I'm the player
while (1)
{
my.pan += 6 * (key_a - key_d) * time_step; // rotate the player using the "A" and "D" keys
vec_set (temp.x, my.x); // copy player's position to temp
temp.z -= 10000; // set temp.z 10000 quants below player's origin
distance_to_ground = c_trace (my.x, temp.x, IGNORE_ME | USE_BOX);
movement_speed.x = 5 * (key_w - key_s) * time_step; // move the player using "W" and "S"
movement_speed.y = 0; // don't move sideways
if (key_space && !reached_height)
{
jump_height = minv(40, jump_height + 5 * time_step); // 40 sets the height, 5 sets the ascending speed
if (jump_height == 40) // reached the maximum height? Then start descending!
{
reached_height = 1;
jump_height = maxv(0, jump_height - 5 * time_step); // 5 sets the falling speed
}
}
else // space isn't pressed anymore?
{
jump_height = maxv(0, jump_height - 3 * time_step); // use a smaller falling speed (3) for smaller jumps
if (!jump_height && !key_space) // the player has touched the ground?
{
reached_height = 0; // then allow it to jump again
}
}
movement_speed.z = -(distance_to_ground - 17) + jump_height; // 17 = experimental value
movement_speed.z = maxv(-35 * time_step, movement_speed.z); // 35 = falling speed
c_move (my, movement_speed.x, nullvector, GLIDE); // move the player
if (!jump_height) // the player isn't jumping?
{
if (!key_w && !key_s) // the player isn't moving?
{
ent_animate(my, "stand", anim_percentage, ANM_CYCLE); // play the "stand" animation
}
else // the player is moving?
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE); // play the "walk" animation
}
anim_percentage += 5 * time_step; // 5 = animation speed
jump_percentage = 0; // always start jumping with the first frame
}
else // the player is jumping
{
jump_percentage += 5 * time_step; // 5 = jump animation speed
ent_animate(my, "jump", jump_percentage, ANM_CYCLE); // play the "jump" animation
}
// camera code
camera.x = player.x - 250 * cos(player.pan);
camera.y = player.y - 250 * sin(player.pan); // use the same value (250) here
camera.z = player.z + 150; // place the camera above the player, play with this value
camera.tilt = -20; // look down at the player
camera.pan = player.pan;
wait (1);
}
}
```

adjust the range of a dynamic light using the mouse wheel

```
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function adjust_light()
{
while (event_type != EVENT_RELEASE)
{
my.lightrange += 0.5 * mickey.z * time_step;
my.lightrange = clamp(my.lightrange, 0, 1000); // limit the light range to 0...1000
wait (1);
}
}
action light_bulb() // attach this action to all the light bulbs
{
vec_set(my.blue, vector(255, 255, 255)); // generate white light
my.lightrange = 300; // default light range value
my.emask |= (ENABLE_TOUCH | EVENT_RELEASE);
my.event = adjust_light;
}
```

create an avatar of the player on screen, filming his face during combat.

```
VIEW* avatar_cam =
{
pos_x = 120;
pos_y = 48;
size_x = 158;
size_y = 119;
layer = 10;
flags = VISIBLE;
}
PANEL* hud_pan =
{
layer = 5;
pos_x = 0;
pos_y = 0;
bmap = "hud.tga";
flags = VISIBLE;
}
function avatarcam_startup()
{
while (!player) {wait (1);}
while (1)
{
vec_set (avatar_cam.x, vector (20, 0, 35)); // play with these values
vec_rotate (avatar_cam.x, player.pan);
vec_add (avatar_cam.x, player.x);
avatar_cam.pan = player.pan + 180; // make avatar_cam look at the player
wait (1);
}
}
```

a panel that shows me how much health I have. I want to make something like the Zelda panels.

```
BMAP* hearts_tga = "hearts.tga"; // that's a stripe that contains 5 small hearts
BMAP* bigheart_tga = "bigheart.tga"; // that's the big heart that appears over the stripe
PANEL* hearts_pan =
{
layer = 15;
pos_x = 0;
pos_y = 20;
bmap = hearts_tga;
flags = visible;
}
PANEL* bigheart_pan =
{
layer = 25; // this panel appear above one of the other panels
pos_x = 200;
pos_y = 15;
bmap = bigheart_tga;
flags = visible;
}
function health_startup()
{
while (!player) {wait (1);} // wait until the player is loaded in the level
player.skill10 = 100;
while (1)
{
if (player.skill10 > 90) // player's health is stored inside its skill10 in this example
bigheart_pan.pos_x = 200;
if ((player.skill10 > 70) && (player.skill10 <= 90))
bigheart_pan.pos_x = 150;
if ((player.skill10 > 50) && (player.skill10 <= 70))
bigheart_pan.pos_x = 100;
if ((player.skill10 > 30) && (player.skill10 <= 50))
bigheart_pan.pos_x = 100;
if ((player.skill10 > 10) && (player.skill10 <= 30))
bigheart_pan.pos_x = 50;
if ((player.skill10 <= 10))
bigheart_pan.pos_x = 0;
wait (1);
}
}
```

detect if two particular objects hit each other

```
function bounce_off()
{
vec_to_angle(my.pan, bounce);
my.pan += 5 - random(10); // add some randomness at each collision
if (you) // collided with another entity?
{
if ((you.skill100 == 13579) && (my.skill100 == 13579)) // you and I are special entities?
{
beep(); // do what you need here
}
}
}
action object1() // this is a special object
{
my.emask |= (ENABLE_BLOCK | ENABLE_ENTITY); // the object is sensitive to block and entity collisions
my.event = bounce_off;
my.skill100 = 13579; // set this skill to a weird value in order to differentiate the object from the others
while(1)
{
c_move(me, vector(5 * time_step, 0, 0), nullvector, 0); // move ahead, in the direction given by the pan angle
wait(1);
}
}
action object2() // this is a special object
{
my.emask |= (ENABLE_BLOCK | ENABLE_ENTITY); // the object is sensitive to block and entity collisions
my.event = bounce_off;
my.skill100 = 13579; // set this skill to a weird value in order to differentiate the object from the others
while(1)
{
c_move(me, vector(5 * time_step, 0, 0), nullvector, 0); // move ahead, in the direction given by the pan angle
wait(1);
}
}
action object3() // this is a regular object (just an example)
{
my.emask |= (ENABLE_BLOCK | ENABLE_ENTITY); // the object is sensitive to block and entity collisions
my.event = bounce_off;
while(1)
{
c_move(me, vector(5 * time_step, 0, 0), nullvector, 0); // move ahead, in the direction given by the pan angle
wait(1);
}
}
```

put views from one or more cameras to small windows and display them on the screen

```
VIEW* camera1 =
{
pos_x = 0;
pos_y = 0;
size_x = 512;
size_y = 384;
flags = VISIBLE;
}
VIEW* camera2 =
{
pos_x = 512;
pos_y = 0;
size_x = 512;
size_y = 384;
flags = VISIBLE;
}
VIEW* camera3 =
{
pos_x = 0;
pos_y = 384;
size_x = 512;
size_y = 384;
flags = VISIBLE;
}
VIEW* camera4 =
{
pos_x = 512;
pos_y = 384;
size_x = 512;
size_y = 384;
flags = VISIBLE;
}
function init_startup()
{
wait (-1);
video_switch(8, 0, 0); // change the resolution to 1024 x 768 pixels
camera.flags &= ~VISIBLE; // now hide the default camera
// set the desired positions and angles for the 4 cameras
vec_set (camera1.x, vector (800, 700, 100));
camera1.pan = 50;
camera1.tilt = 20;
camera1.roll = 0;
// settings for camera 2
vec_set (camera2.x, vector (500, -600, 200));
camera2.pan = 150;
camera2.tilt = -30;
camera2.roll = 20;
// settings for camera 3
vec_set (camera3.x, vector (-1000, 300, 300));
camera3.pan = 110;
camera3.tilt = -20;
camera3.roll = 10;
// settings for camera 4
vec_set (camera4.x, vector (-800, -700, 200));
camera1.pan = 220;
camera1.tilt = -10;
camera1.roll = 30;
}
```

an object to mark the start and goal. I want to start the watch when the player drives near it.

```
var race_over = 0;
var total_time;
action start_counter() // attach this action to your object
{
fps_max = 60; // feel free to use other values (75, etc) here
var time_offset;
while (!player) {wait (1);} // wait until player's car is loaded
// wait until the player comes close to the object; play with 100
while (vec_dist(player.x, my.x) > 100) {wait (1);}
time_offset = total_frames; // ignore the frames that have passed until now
while (!race_over) // this loop will run until the race is over
{
total_time = (total_frames - time_offset) / fps_max; // world's simplest racing timer ;)
wait (1);
}
}
PANEL* time_pan =
{
layer = 150;
```

```
digits(700, 20, 4.3 ,* , 1, total_time);
flags = visible;
}
```

fire particle effect behind the tail of a jet

```
BMAP* fire_tga = "fire.tga";
function fade_fire(PARTICLE *p)
{
p.alpha -= 4 * time_step; // fade out the fire particles
if (p.alpha < 0)
p.lifespan = 0;
}
function fire_effect(PARTICLE *p)
{
p->vel_x = 1 - random(2);
p->vel_y = 1 - random(2);
p->vel_z = 1 + random(1);
p.alpha = 25 + random(50);
p.bmap = fire_tga;
p.size = 25; // gives the size of the fire particles
p.flags |= (BRIGHT | MOVE);
p.event = fade_fire;
}
action my_jet() // attach this action to your plane
{
VECTOR jet_offset;
// put your own code here
// .......................
while (1)
{
// put your own jet flight code here
// the example below simply makes the plane fly in a circle
c_move (my, vector(20 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
my.pan += 2 * time_step; // 2 sets the radius of the circle
// end of the simply flying example code
// the jet particle effect code starts below
// place the particle jet at the proper position, play with these values
vec_set(jet_offset.x, vector(-100, -10, -20));
vec_rotate(jet_offset.x, my.pan);
vec_add(jet_offset.x, my.x);
effect(fire_effect, 5, jet_offset.x, nullvector); // generate 5 fire particles each frame
wait (1);
}
}
```

display a string just like a typewriting machine does, so that each letter or character is displayed after a little break

```
STRING* temp_str = "#100";
STRING* backup_str = "#100";
STRING* typewriter_str = "Hello! This is a typewriter test!";
SOUND* typewriter_wav = "typewriter.wav";
TEXT* typewriter_txt =
{
pos_x = 20;
pos_y = 20;
string(temp_str);
flags = VISIBLE;
}
function init_startup() // call this function whenever you need to
{
wait (-1); // wait until the level is loaded
var i = str_len(typewriter_str);
while(i > 0)
{
i -= 1;
str_cpy(backup_str, typewriter_str);
str_trunc(backup_str, i);
str_cpy(temp_str, backup_str);
snd_play(typewriter_wav, 60, 0);
wait (-0.3);
}
}
```

a lite-C camera that always shows the player from its side

```
action player_and_cam() // simple player and camera code
{
var walk_percentage;
var stand_percentage;
player = my; // I'm the player
while (1)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed,
c_move (my, vector(10 * (key_w - key_s) * time_step, 0, 0), nullvector, GLIDE);
my.pan += 6 * (key_a - key_d) * time_step; // "6" = rotating speed
// place the camera 0 quants in front of the player, 200 quants sideways and 25 quants above player's origin
vec_set (camera.x, vector(0, 200, 25));
vec_rotate (camera.x, player.pan);
vec_add (camera.x, player.x);
camera.pan = player.pan - 90; // make the camera look at the player from a sideview perspective
if (key_w + key_s) // the player is moving?
{
walk_percentage += 5 * time_step;
ent_animate(my, "walk", walk_percentage, ANM_CYCLE);
stand_percentage = 0;
}
else // the player is standing still (or rotating)
{
stand_percentage += 1.5 * time_step;
ent_animate(my, "stand", stand_percentage, ANM_CYCLE);
walk_percentage = 0;
}
wait (1);
}
}
```

When a player gets a high score

```
var high_score = 0;
var current_score = 0;
STRING* name_str = "#50"; // allow the name to have up to 50 characters
STRING* input_str = "#50";
TEXT* name_txt =
{
pos_x = 10;
pos_y = 10;
string(name_str);
flags = VISIBLE;
}
TEXT* input_txt =
{
pos_x = 10;
pos_y = 550;
string(input_str);
flags = VISIBLE;
}
PANEL* score_txt =
{
pos_x = 10;
pos_y = 10;
digits(300, 10, 4, *, 1, high_score);
digits(300, 300, 4, *, 1, current_score);
flags = VISIBLE;
}
function random_number()
{
var filehandle;
current_score = integer(random(100));
if (current_score > high_score) // a new high score was achieved?
{
high_score = current_score;
str_cpy(input_str, "New high score! Please type a name!");
wait (-3); // display the message for 3 seconds
str_cpy(input_str, "#50"); // reset the string
inkey(input_str); // now let's input player's name
str_cpy(name_str, input_str); // let's update the high score name on the screen as well
filehandle = file_open_write("highscore.txt"); // overwrite the existing highscore.txt file (if any)
file_str_write(filehandle, input_str);
file_asc_write (filehandle, 13); // write the following value on a separate line
file_asc_write (filehandle, 10); // using asc(13) = carriage return + asc(10) = line feed
file_var_write(filehandle, high_score);
file_close(filehandle);
}
}
function score_startup()
{
var filehandle;
filehandle = file_open_read("highscore.txt");
if (filehandle) // the file exists?
{
file_str_read(filehandle, name_str);
high_score = file_var_read(filehandle);
}
else // the highscore.txt file doesn't exist
{
str_cpy(name_str, "None");
high_score = 0;
}
on_r = random_number; // generates a random number every time when the player presses the "R" key
}
```

have a looping sound fade in / out as a player gets closer / farther to it

```
action looping_sound() // place a small entity in Wed and attach it this action
{
var sound_handle;
var sound_volume;
set (my, PASSABLE | INVISIBLE);
// make sure that the player action includes a line of code that looks like this:
// "player = my;" (without the quotes)
while (!player) {wait (1);}
var dist_to_player;
sound_handle = media_loop("mysound.wav", NULL, 1); // start with a small volume
while (1)
{
dist_to_player = vec_dist(player.x, my.x);
sound_volume = maxv(1, 5000 / (dist_to_player + 1)); // feel free to use your own formula here
media_tune(sound_handle, sound_volume, 0, 0); // tune the sound volume each and every frame
wait (1);
}
}
```

a subway and I want the doors to move along with the subway

```
// create the subway doors as mdl entities and name them subdoor1.mdl and subdoor2.mdl
function subway_door1()
{
while (1)
{
vec_set(my.x, vector(-100, -40, 10));
vec_rotate(my.x, you.pan);
vec_add(my.x, you.x);
wait (1);
}
}
function subway_door2()
{
while (1)
{
vec_set(my.x, vector(100, -40, 10));
vec_rotate(my.x, you.pan);
vec_add(my.x, you.x);
wait (1);
}
}
action my_subway() // attach this action to the subway
{
ent_create("subdoor1.mdl", nullvector, subway_door1);
ent_create("subdoor2.mdl", nullvector, subway_door2);
while (1)
{
// primitive movement code; move the "real" subway using c_move, etc
// the following loops simply move the subway back and forth along the x axis (0... 1000 quants)
while (my.x < 1000)
{
my.x += 20 * time_step;
wait (1);
}
wait (-2);
while (my.x > 0)
{
my.x -= 20 * time_step;
wait (1);
}
wait (-2);
}
}
```

create bullets for a gun

```
ENTITY* weapon1;
function attach_weapon1()
{
weapon1 = my; // I'm the gun
set(my, PASSABLE);
while (1)
{
vec_set (my.x, vector (20, -10, 35)); // set the proper gun offset in relation to the player
vec_rotate (my.x, you.pan);
vec_add (my.x, you.x);
my.pan = you.pan;
my.tilt = camera.tilt;
wait (1);
}
}
action players_code() // simple player and 1st person camera code
{
player = my; // I'm the player
ent_create ("weapon1.mdl", nullvector, attach_weapon1);
while (1)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, GLIDE);
vec_set (camera.x, player.x); // use player's x and y for the camera as well
camera.z += 30; // place the camera 30 quants above the player on the z axis (approximate eye level)
camera.pan -= 5 * mouse_force.x * time_step; // rotate the camera around by moving the mouse
camera.tilt += 3 * mouse_force.y * time_step; // on its x and y axis
player.pan = camera.pan; // the camera and the player have the same pan angle
wait (1);
}
}
function remove_bullets() // this function runs when the bullet collides with something
{
wait (1); // wait a frame to be sure (don't trigger engine warnings)
ent_remove (my); // and then remove the bullet
}
function move_bullets()
{
VECTOR bullet_speed[3]; // stores the speed of the bullet
// the bullet is sensitive to impacts with entities and level blocks
my.emask = ENABLE_IMPACT | ENABLE_ENTITY | ENABLE_BLOCK;
my.event = remove_bullets; // when it collides with something, its event function (remove_bullets) will run
my.pan = weapon1.pan;
my.tilt = weapon1.tilt;
bullet_speed.x = 35 * time_step; // adjust the speed of the bullet here
bullet_speed.y = 0; // the bullet doesn't move sideways
bullet_speed.z = 0; // or up / down on the z axis
while (my) // this loop will run for as long as the bullet exists (it isn't "NULL")
{
// move the bullet ignoring its creator (weapon1)
c_move (my, bullet_speed, nullvector, IGNORE_PASSABLE | IGNORE_YOU);
wait (1);
}
}
function fire_bullets()
{
VECTOR bullet_pos[3];
// generate bullets from the 6th vertex of the weapon (get the vertex number from Med)
vec_for_vertex(bullet_pos, weapon1, 6);
ent_create("bullet.mdl", bullet_pos, move_bullets);
}
function bullets_startup()
{
on_mouse_left = fire_bullets;
}
```

bullet hits the target, the target dies or an effect triggers

```
BMAP* sparks_tga = "sparks.tga";
function fade_sparks(PARTICLE *p)
{
p.alpha -= 2 * time_step; // fade out the sparks
if (p.alpha < 0)
p.lifespan = 0;
}
function sparks_effect(PARTICLE *p)
{
set (my, PASSABLE);
p->vel_x = 5 - random(10);
p->vel_y = 5 - random(10);
p->vel_z = 5 - random(10);
p.alpha = 50 + random(50);
p.bmap = sparks_tga;
p.size = 5; // gives the size of the sparks
p.flags |= (BRIGHT | MOVE);
p.event = fade_sparks;
}
function got_shot()
```

```
{
// the event was triggered by player's body (and not by its bullets)? Then nothing should happen!
if (you == player) {return;}
effect(sparks_effect, 10, my.x, nullvector); // generate 10 particles at the origin of the target
var anim_percentage = 0;
my.event = NULL; // the enemy is dead here, so it shouldn't react to player's bullets from now on
while (anim_percentage < 100) // this loop will run until the "death" animation percentage reaches 100%
{
ent_animate(my, "death", anim_percentage, NULL); // play the "death" animation only once
anim_percentage += 3 * time_step; // "3" controls the animation speed
wait (1);
}
set (my, PASSABLE); // allow the player to pass through the corpse now
}
action target_entity()
{
my.emask = ENABLE_IMPACT | ENABLE_ENTITY; // the target is sensitive to impact with player's bullets
my.event = got_shot; // and runs its event function when it is hit
}
```

a rabbit that can detect obstacles (walls) and can jump over them automatically

```
action wall_jumper() // attach this action to your rabbit model
{
VECTOR wall_ahead[3];
VECTOR temp[3];
var jumper_height = 0;
var dist_to_ground = 0;
var anim_percentage = 0;
while (my.x < 1200) // jump all the walls until my.x reaches 1200 (play with 1200)
{
// create a vector that is placed 100 quants in front of the rabbit and close to its feet
vec_set (wall_ahead.x, vector (70, 0, -20)); // play with 100 and -20
vec_rotate (wall_ahead.x, my.pan);
vec_add (wall_ahead.x, my.x);
c_move (my, vector(10 * time_step, 0, jumper_height * time_step), nullvector, GLIDE); // 10 = movement speed
anim_percentage += 10 * time_step;
ent_animate(my, "rockrun", anim_percentage, ANM_CYCLE);
if (c_content(wall_ahead.x, 0) == 3) // detected a wall in front of the rabbit?
{
// then move upwards for as long as there's a wall in front of the rabbit
jumper_height = 90; // 90 gives the height of the jump, play with it
}
else // no wall is detected? The let's descend (or continue to move as before)
{
vec_set(temp.x, my.x);
temp.z -= 10000;
dist_to_ground = c_trace(my.x, temp.x, IGNORE_PASSABLE);
// keep the jumper with its feet above the ground and allow a bit of jumping on regular surfaces as well
if (dist_to_ground > 70) // play with 70
{
jumper_height = -dist_to_ground * 0.15 * time_step; // play with 0.15 (sets the descending speed)
}
}
wait (1);
}
}
```

attach a "real", working headlight to one of my models

```
function my_headlight()
{
set(my, PASSABLE);
vec_set(d3d_spotlightcone, vector(25, 80, 4)); // play with these values
vec_set(my.blue, vector(255, 255, 255)); // the headlight will generate white light
my.lightrange = 500; // on a range of up to 500 quants
my.flags2 |= SPOTLIGHT;
while (1)
{
// play with the numerical values, they set the position of the headlight (x, y, z)
vec_set (my.x, vector (10, 0, 50));
vec_rotate (my.x, you.pan);
vec_add (my.x, you.x);
my.pan = you.pan;
wait (1);
}
}
action my_model() // simple entity movement code using the cursor keys
{
ent_create("headlight.mdl", nullvector, my_headlight); // create a tiny headlight model
while (1)
{
c_move (my, vector(10 * (key_cuu - key_cud) * time_step, 0, 0), nullvector, GLIDE | IGNORE_PASSABLE);
my.pan += 6 * (key_cul - key_cur) * time_step;
wait (1);
}
}
```

blow up a barrel with flames and smoke

```
BMAP* fire_tga = "fire.tga";
BMAP* smoke_tga = "smoke.tga";
function fade_fire(PARTICLE *p)
{
p.alpha -= 4 * time_step; // fade out the fire particles
if (p.alpha < 0)
p.lifespan = 0;
}
function fade_smoke(PARTICLE *p)
{
p.alpha -= 0.5 * time_step; // fade out the fire particles
if (p.alpha < 0)
p.lifespan = 0;
}
function fire_effect(PARTICLE *p)
{
set (my, PASSABLE);
p->vel_x = 5 - random(10);
p->vel_y = 5 - random(10);
p->vel_z = 10 - random(20);
p.alpha = 50 + random(50);
p.bmap = fire_tga;
p.size = 15; // gives the size of the flame particles
p.flags |= (BRIGHT | MOVE);
p.event = fade_fire;
}
function smoke_effect(PARTICLE *p)
{
set (my, PASSABLE);
p->vel_x = 1 - random(2);
p->vel_y = 1 - random(2);
p->vel_z = 1 + random(2);
p.alpha = 10 + random(20);
p.bmap = smoke_tga;
p.size = 25; // gives the size of the smoke particles
p.flags |= MOVE;
p.event = fade_smoke;
}
function explo_sprite()
{
set(my, BRIGHT);
my.alpha = 100;
my.scale_x = 3; // this value gives the scale of the explosion sprite
my.scale_y = my.scale_x;
my.frame = 1;
while (my.frame < 5)
{
my.frame += 0.5 * time_step;
wait (1);
}
while (my.alpha > 0)
{
my.alpha -= 3 * time_step;
}
ent_remove (my);
}
function got_shot()
{
my.event = NULL; // don't react to other events from now on
set(my, PASSABLE); // the barrel is now passable
ent_create("explo+5.tga", my.x, explo_sprite);
effect(fire_effect, 1000, my.x, nullvector);
effect(smoke_effect, 100, my.x, nullvector);
my.alpha = 100;
set(my, TRANSLUCENT);
while (my.alpha > 0)
{
my.alpha -= 5 * time_step;
wait (1);
}
ent_remove(my);
}
action explo_barrel() // attach this action to your barrels
{
// make the barrel entity sensitive to c_trace and impact with other entities
my.emask |= (ENABLE_SHOOT | ENABLE_IMPACT | ENABLE_ENTITY);
my.event = got_shot;
}
```

blow it up with smoke, flames and have it roll over on its side at the same time

```
BMAP* fire_tga = "fire.tga";
BMAP* smoke_tga = "smoke.tga";
function fade_fire(PARTICLE *p)
{
p.alpha -= 4 * time_step; // fade out the fire particles
if (p.alpha < 0)
p.lifespan = 0;
}
function fade_smoke(PARTICLE *p)
{
p.alpha -= 1 * time_step; // fade out the fire particles
if (p.alpha < 0)
p.lifespan = 0;
}
function fire_effect(PARTICLE *p)
{
set (my, PASSABLE);
p->vel_x = 4 - random(8);
p->vel_y = 4 - random(8);
p->vel_z = 2 + random(5);
```

```
p.alpha = 20 + random(30);
p.bmap = fire_tga;
p.size = 25; // gives the size of the flame particles
p.flags |= (BRIGHT | MOVE);
p.event = fade_fire;
}
function smoke_effect(PARTICLE *p)
{
set (my, PASSABLE);
p->vel_x = 3 - random(6);
p->vel_y = 3 - random(6);
p->vel_z = 3 + random(5);
p.alpha = 5 + random(10);
p.bmap = smoke_tga;
p.size = 50; // gives the size of the smoke particles
p.flags |= MOVE;
p.event = fade_smoke;
}
function explo_sprite()
{
set(my, BRIGHT);
my.alpha = 100;
my.scale_x = 3; // this value gives the scale of the explosion sprite
my.scale_y = my.scale_x;
my.frame = 1;
while (my.frame < 5)
{
my.frame += 0.5 * time_step;
wait (1);
}
while (my.alpha > 0)
{
my.alpha -= 3 * time_step;
}
ent_remove (my);
}
function car_hit()
{
my.event = NULL; // don't react to other events from now on
ent_create("explo+5.tga", my.x, explo_sprite);
wait (-0.1);
my.skill1 = my.z; // store the initial height of the car
while (my.z < (my.skill1 + 500)) // allow the car to jump up to 500 quants
{
my.z += 50 * time_step;
if (my.roll < 270)
my.roll += 10 * time_step;
wait (1);
}
my.skill10 = 0;
while (my.skill10 < 0.5) // keep the car at its maximum height for 0.5 seconds (for increased realism)
{
my.skill10 += time_step / 16;
if (my.roll < 270)
my.roll += 10 * time_step;
wait (1);
}
while (my.z > (my.skill1 + 10)) // allow the car to return to the ground now, 10 = experimental value
{
my.z -= 40 * time_step;
if (my.roll < 270)
my.roll += 10 * time_step;
wait (1);
}
while (my.roll < 270) // make sure that the car has rotated for a full 270 degrees angle
{
my.roll += 10 * time_step;
wait (1);
}
while (1)
{
```

```
effect(fire_effect, 200, my.x, nullvector);
effect(smoke_effect, 50, my.x, nullvector);
wait (1);
}
}
action explo_car() // attach this action to your car
{
// make the car entity sensitive to c_trace and impact with other entities
my.emask |= (ENABLE_SHOOT | ENABLE_IMPACT | ENABLE_ENTITY);
my.event = car_hit;
}
```

a flash (smashing like) appear in random places on my model

```
function smashed_effect()
{
while (my.scale_x < 2)
{
my.scale_x += 1 * time_step;
my.scale_y = my.scale_x;
wait (1);
}
while (my.scale_x > 0.1)
{
my.scale_x -= 3 * time_step;
my.scale_y = my.scale_x;
wait (1);
}
ent_remove (my);
}
action smashed_entity()
{
VECTOR temp[3];
var random_vertex;
while (1)
{
random_vertex = integer(random(ent_vertices (my))) + 1;
vec_for_vertex (temp, my, random_vertex);
ent_create("smashed.pcx", temp.x, smashed_effect);
wait (-3);
}
```

a camera that can zoom in and out, rotates around the player but always faces it. The left and right arrow key should do that; the up and down arrow should move above and below the player

```
// use WSAD to walk, keys left / right to rotate the camera around the player
// use the mouse wheel to zoom in / out and the up / down keys to set the height of the camera
action player_cam() // player / camera code
{
VECTOR temp;
VECTOR temp2;
var movement_speed = 5; // movement speed
var anim_percentage;
var cam_angle = 90; // set the initial camera angle here
var cam_dist = 250; // set the default zoom in factor here
var cam_height = 150; // set the default camera height here
player = my; // I'm the player
while (1)
{
if((!key_w) && (!key_s)) // the player isn't moving at all?
{
ent_animate(my, "stand", anim_percentage, ANM_CYCLE); // and play the "stand" animation
}
else // the player is moving?
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE); // and play the "walk" animation
}
// zoom in / out using the mouse wheel
cam_dist += 0.2 * mickey.z * time_step;
// limit the distance between the camera and the player to 20...500
cam_dist = clamp(cam_dist, 20, 500);
// set the height of the camera using the up / down cursor keys
cam_height += 3 * (key_cuu - key_cud) * time_step;
// limit the height of the camera to 50...300
cam_height = clamp(cam_height, 50, 300);
// rotate the player using the A / D keys
my.pan += 4 * (key_a - key_d) * time_step;
anim_percentage += 5 * time_step; // 5 = animation speed
camera.x = player.x - cam_dist * cos(cam_angle);
camera.y = player.y - cam_dist * sin(cam_angle);
cam_angle += 5 * (key_cur - key_cul) * time_step; // 5 gives the camera rotation speed
camera.z = player.z + cam_height;
vec_set(temp2.x, my.x);
vec_sub(temp2.x, camera.x);
vec_to_angle(camera.pan, temp2); // rotate the camera towards the player at all times
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = 0;
temp.z = 0;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}

}
```

code which allows a player to ride upon a moving train on a path, but while the player is on the train he can move around also

```
var entity_speed = 3;
var dist_to_node;
var current_node = 1;
var angle_difference = 0;
VECTOR temp_angle;
VECTOR pos_node[3]; // stores the position of the node
ENTITY* train;
function move_target()
{
while(1)
{
entity_speed = minv(5, entity_speed + 0.5 * time_step);
c_move(my, vector(entity_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x));
wait(1);
}
}
action move_on_path() // attach this action to your model
{
```

```
train = my;
set(my, POLYGON);
move_target();
path_scan(me, my.x, my.pan, vector(360, 180, 1000));
path_getnode (my, 1, pos_node, NULL);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate towards the node
while(1)
{
dist_to_node = vec_dist(my.x, pos_node);
if(dist_to_node < 50) // close to the node?
{
current_node = path_nextnode(my, current_node, 1);
if (!current_node) {current_node = 1;} // reached the end of the path? Then start over!
path_getnode (my, current_node, pos_node, NULL);
}
wait(1);
}
}
action player_over_train()
{
while (!train) {wait (1);}
var player_offset_x = 0;
var player_offset_y = 0;
var anim_percentage;
set(my, PASSABLE);
while (1)
{
player_offset_x += 5 * (key_w - key_s) * time_step;
player_offset_y += 5 * (key_a - key_d) * time_step;
my.x = train.x + player_offset_x;
my.y = train.y + player_offset_y;
my.z = train.z + 50; // place the player model on top of the train model, play with this value
if((!key_w) && (!key_s)) // the player isn't moving at all?
{
ent_animate(my, "stand", anim_percentage, ANM_CYCLE); // and play the "stand" animation
}
else // the player is moving?
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE); // and play the "walk" animation
}
anim_percentage += 5 * time_step; // 5 = animation speed
wait (1);
}
}
```

a machine gun that creates hit hole panels with random orientation on the walls.

```
VECTOR trace_coords;
STRING* hithole_tga = "hithole.tga"; // that's your hit hole bitmap
STRING* target_mdl = "target.mdl"; // weapon target model
SOUND* bullet_wav = "bullet.wav"; // bullet sound
function fire_bullets(); // creates the bullets
function show_target(); // displays the (red) target model
function display_hithole(); // shows the hit hole bitmap
ENTITY* myweapon_ent =
{
type = "myweapon.mdl"; // weapon model
pan = 0; // weapon angle
x = 55; // 55 quants ahead of the view, play with this value
y = -20; // 20 quants towards the right side of the screen, play with this value
z = -20; // 20 quants below, play with this value
pan = 2; // weapon's pan angle (you can also use tilt and roll)
flags2 = VISIBLE;
}
```

```
action my_player() // attach this action to your player
{
var movement_speed = 10; // movement speed
VECTOR temp;
player = my; // I'm the player
set (my, INVISIBLE);
while (1)
{
player.pan -= 7 * mouse_force.x * time_step;
camera.x = player.x;
camera.y = player.y;
camera.z = player.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = player.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX);
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
function weapon_startup()
{
on_mouse_left = fire_bullets; // call this function when the left mouse button is pressed
proc_mode = PROC_LATE; // run this function at the end of the function scheduler list (eliminates jerkiness)
while (1)
{
vec_set(trace_coords.x, vector(10000, 0, 0)); // the weapon has a firing range of up to 10,000 quants
vec_rotate(trace_coords.x, camera.pan);
vec_add(trace_coords.x, camera.x);
if (c_trace(camera.x, trace_coords.x, IGNORE_ME | IGNORE_PASSABLE) > 0) // hit something?
{
ent_create (target_mdl, target.x, show_target); // then show the target model
}
wait (1);
}
}
function show_target()
{
set (my, PASSABLE); // the target model is passable
my.ambient = 100; // and should look bright enough
my.scale_x = minv (6, vec_dist (my.x, camera.x) / 500); // play with 6 and with 500
my.scale_y = my.scale_x;
my.scale_z = my.scale_x;
wait (1);
ent_remove (my);
}
function fire_bullets()
{
while (mouse_left)
{
c_trace(camera.x, trace_coords.x, IGNORE_ME | IGNORE_PASSABLE | ACTIVATE_SHOOT);
if (!you) // hit a wall?
{
ent_create (hithole_tga, target.x, display_hithole); // then create a bullet hit hole
}
snd_play (bullet_wav, 100, 0); // play the bullet sound at a volume of 100
wait (-0.16); // fire 6 bullets per second (6 * 0.16 ~= 1 second)
}
}
function display_hithole()
{
vec_to_angle (my.pan, normal); // orient the hit hole sprite correctly
vec_add(my.x, normal.x); // move the sprite a bit away from the wall
set (my, PASSABLE); // the hit hole bitmap is passable
set (my, TRANSLUCENT); // and transparent
my.ambient = 50;
my.roll = random(360); // and has a random roll angle
my.scale_x = 0.5; // we scale it down
my.scale_y = my.scale_x; // on the x and y axis
wait (-20); // show the hit hole bitmap for 20 seconds
ent_remove (my); // and then remove it
}
```

two gun entities. Setting their positions works fine, but I want to be able to change the gun model as well, depending on the value of a variable named currentGun

```
var currentGun = 1; // default weapon = no.1
ENTITY* weapon1_ent =
{
type = "weapon1.mdl"; // weapon model
pan = 0; // weapon angle
x = 55; // 55 quants ahead of the view, play with this value
y = -20; // 20 quants towards the right side of the screen, play with this value
z = -20; // 20 quants below, play with this value
pan = 2; // weapon's pan angle (you can also use tilt and roll)
}
ENTITY* weapon2_ent =
{
type = "weapon2.mdl"; // weapon model
pan = 0; // weapon angle
x = 60; // 60 quants ahead of the view, play with this value
y = 15; // 15 quants towards the left side of the screen, play with this value
z = -18; // 18 quants below, play with this value
pan = -3; // weapon's pan angle (you can also use tilt and roll)
}
function toggle_guns_startup()
{
while (1)
{
if (key_1)
{
weapon1_ent.flags2 = VISIBLE;
weapon2_ent.flags2 = ~VISIBLE;
currentGun = 1;
}
if (key_2)
{
weapon1_ent.flags2 = ~VISIBLE;
weapon2_ent.flags2 = VISIBLE;
currentGun = 2;
}
if (currentGun == 1)
{
weapon1_ent.flags2 = VISIBLE;
weapon2_ent.flags2 = ~VISIBLE;
}
if (currentGun == 2)
{
weapon1_ent.flags2 = ~VISIBLE;
weapon2_ent.flags2 = VISIBLE;
}
wait (1);
}
}
```

fades in a music track, plays it for 1 minute and then fades it out.

```
function music_startup()
{
var tune_handle;
var tune_factor = 1; // play the sound at an initial volume of 1
tune_handle = media_play("mymusic.wav", NULL, tune_factor);
while (tune_factor < 80) // increase the volume until it reaches 80
{
media_tune(tune_handle, tune_factor, 100, 0);
tune_factor += 1 * time_step; // 1 gives the fade-in speed
wait (1);
}
wait (-60); // play the music loud for 1 minute
while (tune_factor > 0) // now change the volume - decrease it gently
{
media_tune(tune_handle, tune_factor, 100, 0);
tune_factor -= 0.6 * time_step; // 0.6 gives the fade-out speed
wait (1);
}
media_stop (tune_handle); // now we can stop the sound
}
```

an aimlessly moving AI that changes direction at a random interval between 4 and 10 seconds

```
action random_guy()
{
var anim_percentage;
var time_passed = 0;
var random_interval;
while (1)
{
if (time_passed == 0)
{
random_interval = 4 + random(6);
}
c_move (my, vector(5 * time_step, 0, 0), nullvector, GLIDE); // "5" controls the walking speed
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 6 * time_step; // "6" controls the "walk" animation speed
time_passed += time_step / 16;
if (time_passed > random_interval)
{
time_passed = 0; // reset time_passed
my.pan += 90 - random(180); // and then add -90...+90 degrees to the pan angle
}
wait (1);
}
}
```

an entity that is controlled using the arrow keys and acts like a magnet for some other entities

```
function got_scanned()
{
my.event = NULL; // the entities will follow the magnet from now on
VECTOR temp_pos;
while (1)
{
if (vec_dist (player.x, my.x) > 50) // the object isn't close enough to the magnet?
{
vec_set(temp_pos, player.x);
vec_sub(temp_pos, my.x);
vec_to_angle(my.pan, temp_pos);
my.tilt = 0;
// the entity is oriented towards the magnet here
c_move (my, vector(3 * time_step, 0, 0), nullvector, GLIDE);
}
wait (1);
}
}
```

```
// the entities that have this action attached to them are sensitive to the magnet entity
action sensitive_ents()
{
my.emask |= ENABLE_SCAN; // make this entity sensitive to scanning
my.event = got_scanned;
}
action my_magnet() // attach this action to your entity
{
player = my; // I'm the player
while (1)
{
// move the player using the cursor keys
c_move (my, vector(10 * (key_cuu - key_cud) * time_step, 6 * (key_cul - key_cur) * time_step, 0), nullvector, GLIDE);
vec_set (camera.x, player.x); // use player's x and y for the camera as well
camera.z += 500; // place the camera 500 quants above the player on the z axis
camera.tilt = -90;
// make the magnet active on a radius of up to 200 quants around it
c_scan(my.x, my.pan, vector(360, 180, 200), IGNORE_ME);
wait (1);
}
}
```

two entities but I'd like to switch the control from one to the other, using the same player action instead of creating several player actions

```
BMAP* arrow_pcx = "arrow.pcx";
function mouse_startup()
{
// allow the player to switch the characters even if they are 10,000 quants away from each other
mouse_range = 10000;
mouse_mode = 1;
mouse_map = arrow_pcx;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function move_player()
{
var anim_percentage;
while (player == my)
{
camera.x = my.x - 200 * cos(my.pan); // place the camera 200 quants behind the player
camera.y = my.y - 200 * sin(my.pan);
camera.z = my.z + 300; // and 300 quants above its origin
camera.pan = my.pan;
camera.tilt = -35; // look down at the player
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = rotation speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 0, 0), nullvector, GLIDE);
my.pan += 6 * (key_a - key_d) * time_step;
if (key_w || key_s) // the player is walking?
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 8 * time_step; // "8" controls the "walk" animation speed
}
else // the player is standing still?
{
ent_animate(my, "stand", anim_percentage, ANM_CYCLE);
anim_percentage += 0.5 * time_step; // "0.5" controls the "stand" animation speed
}
wait (1);
}
}
function switch_players()
{
if (player == my) {return;} // don't make the switch if the actual player is clicked again
player = my;
camera.pan = my.pan; // start with a proper camera orientation
move_player();
}
action set_player()
{
player = my;
move_player();
my.emask |= ENABLE_CLICK;
my.event = switch_players;
}
```

add movement to my player and put the camera behind it, so that when I press W,A,S,D the player will move and the camera will be kept always behind the player

```
action my_player()
{
var movement_speed = 10; // movement speed
var walk_percentage;
var stand_percentage;
VECTOR temp;
player = my; // I'm the player
while (1)
{
player.pan -= 5 * mouse_force.x * time_step; // rotate the player using the mouse
if(!key_w && !key_s) // the player isn't moving at all?
{
ent_animate(my, "stand", stand_percentage, ANM_CYCLE); // then play its "stand" animation
stand_percentage += 5 * time_step; // 5 = animation speed
}
else // the player is moving?
{
ent_animate(my, "walk", walk_percentage, ANM_CYCLE); // then play its "walk" animation
walk_percentage += 6 * time_step; // 6 = animation speed
}
camera.x = player.x - 250 * cos(player.pan);
camera.y = player.y - 250 * sin(player.pan);
camera.pan = player.pan; // the camera and the player have the same pan angle
camera.z = player.z + 150; // place the camera above the player, play with this value
camera.tilt = -25;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
temp.z = 0;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

bullet holes on surfaces

```
VECTOR trace_coords;
var weapon_height;
var current_ammo = 20;
var players_ammo;
STRING* hithole_tga = "hithole.tga";
STRING* target_mdl = "target.mdl";
SOUND* bullet_wav = "bullet.wav";
SOUND* gotweapon2_wav = "gotweapon2.wav";
SOUND* nobullets_wav = "nobullets.wav";
SOUND* gotammo_wav = "gotammo.wav";
function fire_bullets(); // creates the bullets
function show_target(); // displays the (red) target model
function display_hithole(); // shows the hit hole bitmap
ENTITY* weapon2_ent =
{
type = "ar18.mdl"; // weapon model
pan = 0; // weapon angle
```

```
x = 55; // 55 quants ahead of the view, play with this value
y = -22; // 22 quants towards the right side of the screen, play with this value
z = -22; // 22 quants below, play with this value
pan = 2; // weapon's pan angle (you can also use tilt and roll)
}
action player1() // attach this action to your player
{
var movement_speed = 10; // movement speed
VECTOR temp;
player = my; // I'm the player
set (my, INVISIBLE);
while (1)
{
player.pan -= 7 * mouse_force.x * time_step;
camera.x = player.x;
camera.y = player.y;
camera.z = player.z + 50 + 1.1 * sin(my.skill44); // play with 50 and 1.1
camera.pan = player.pan;
camera.tilt += 5 * mouse_force.y * time_step;
vec_set (temp.x, my.x); // trace 10,000 quants below the player
temp.z -= 10000;
temp.z = -c_trace (my.x, temp.x, IGNORE_ME | IGNORE_PASSABLE | USE_BOX);
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
action players_weapon() // attach this action to your player model
{
set (my, PASSABLE); // the weapon is made passable
while (!player) {wait (1);} // we wait until the player is created
while (vec_dist (player.x, my.x) > 50) // we wait until the player comes closer than 50 quants
{
my.pan += 3 * time_step;
wait (1);
}
players_ammo = 300;
snd_play (gotweapon2_wav, 80, 0);
set (my, INVISIBLE);
set (weapon2_ent, VISIBLE);
wait (-1);
ent_remove (my);
}
function use_weapon_startup()
{
on_mouse_left = fire_bullets; // call this function when the left mouse button is pressed
proc_mode = PROC_LATE; // run this function at the end of the function scheduler list (elliminates jerkiness)
VECTOR player1_pos; // stores the initial position of the player
VECTOR player2_pos; // stores the position of the player after a frame
while (!player) {wait (1);}
while (1)
{
vec_set (player1_pos.x, player.x); // store the initial player position
if (is (weapon2_ent, VISIBLE)) // the weapon is visible?
{
vec_set(trace_coords.x, vector(10000, 0, 0)); // the weapon has a firing range of up to 10,000 quants
vec_rotate(trace_coords.x, camera.pan);
vec_add(trace_coords.x, camera.x);
if (c_trace(camera.x, trace_coords.x, IGNORE_ME | IGNORE_PASSABLE) > 0) // hit something?
{
ent_create (target_mdl, target.x, show_target); // then show the target model
}
}
wait (1);
vec_set (player2_pos.x, player.x); // store player's position after 1 frame
if (vec_dist (player1_pos.x, player2_pos.x) != 0) // the player has moved during the last frame?
{
weapon_height += 30 * time_step; // then offset weapon_height (30 = weapon waving speed)
weapon2_ent.z += 0.03 * sin(weapon_height); // (0.03 = weapon waving amplitude)
}
}
}
function show_target()
{
set (my, PASSABLE); // the target model is passable
my.ambient = 100; // and should look bright enough
my.scale_x = minv (6, vec_dist (my.x, camera.x) / 500); // play with 6 and with 500
my.scale_y = my.scale_x;
my.scale_z = my.scale_x;
wait (1);
ent_remove (my);
}
function fire_bullets()
{
if (is (weapon2_ent, INVISIBLE)) {return;} // don't do anything if the player hasn't picked up the weapon yet
if (current_ammo > 0) // got ammo?
{
c_trace(camera.x, trace_coords.x, IGNORE_ME | IGNORE_PASSABLE | ACTIVATE_SHOOT);
if (you == NULL) // hit a wall?
{
ent_create (hithole_tga, target.x, display_hithole); // then create a bullet hit hole
}
snd_play (bullet_wav, 100, 0); // play the bullet sound at a volume of 100
current_ammo -= 1; // decrease the number of bullets
}
else // no ammo left?
{
snd_play (nobullets_wav, 100, 0); // then play the noammo.wav sound
}
}
function display_hithole()
{
vec_to_angle (my.pan, normal); // orient the hit hole sprite correctly
vec_add(my.x, normal.x); // move the sprite a bit away from the wall
set (my, PASSABLE); // the hit hole bitmap is passable
set (my, TRANSLUCENT); // and transparent
my.ambient = 50;
my.roll = random(360); // and has a random roll angle (looks nicer this way)
my.scale_x = 0.5; // we scale it down
my.scale_y = my.scale_x; // on the x and y axis
wait (-20); // show the hit hole for 20 seconds
ent_remove (my); // and then remove it
}
action ammo_pack() // attach this action to your ammo pack models
{
set (my, PASSABLE);
while (!player) {wait (1);}
while (vec_dist (player.x, my.x) > 50)
{
my.pan += 3 * time_step;
wait (1);
}
snd_play (gotammo_wav, 80, 0);
set (my, INVISIBLE);
current_ammo += 20; // use your own value here
wait (-1);
ent_remove (my);
}
```

program a character that changes its size every time when it is shot

```
function got_shot()
{
if (you == player) {return;} // don't react if the player touches the entity
my.scale_x += 0.1;
my.scale_y = my.scale_x;
my.scale_z = my.scale_x;
}
action my_entity()
```

```
{
set (my, POLYGON);
my.emask |= (ENABLE_IMPACT | ENABLE_SHOOT); // shoot using real bullets or c_trace
my.event = got_shot; // the event function runs every time when the entity is hit
}
```

a 3rd person camera that stays behind the player and rotates around the character if the player doesn't move for a while

```
action player1()
{
var movement_speed = 10; // movement speed
var anim_percentage;
var standing_still = 0;
var camangle;
VECTOR temp;
VECTOR temp2;
player = my; // I'm the player
while (1)
{
if((!key_w) && (!key_s) && (!key_a) && (!key_d)) // the player isn't moving at all?
{
standing_still += time_step / 8; // play with 8 until you get the proper pausing interval
ent_animate(my, "stand", anim_percentage, ANM_CYCLE); // and play the "stand" animation
}
else // the player is moving?
{
camangle = 0; // reset the camera rotation angle
standing_still = 0; // reset standing_still
ent_animate(my, "walk", anim_percentage, ANM_CYCLE); // and play the "walk" animation
}
anim_percentage += 5 * time_step; // 5 = animation speed
if (standing_still > 10) // the player didn't move at all lately? Then rotate the camera around it
{
camera.x = player.x - 250 * cos(camangle);
camera.y = player.y - 250 * sin(camangle);
camangle += 0.5 * time_step;
vec_set(temp2.x, my.x);
vec_sub(temp2.x, camera.x);
vec_to_angle(camera.pan, temp2);
}
else // the player has changed its position recently?
{
camera.x = player.x - 250 * cos(player.pan);
camera.y = player.y - 250 * sin(player.pan);
camera.pan = player.pan; // the camera and the player have the same pan angle
}
camera.z = player.z + 150; // place the camera above the player, play with this value
camera.tilt = -25;
temp.x = movement_speed * (key_w - key_s) * time_step;
temp.y = movement_speed * (key_a - key_d) * 0.6 * time_step;
temp.z = 0;
c_move (my, temp.x, nullvector, IGNORE_PASSABLE | GLIDE);
wait (1);
}
}
```

npc is supposed to rotate towards the player and say its line once, but only when it is looking directly at the player.

```
action my_npc() // attach this action to your npc
{
// make sure that the player action includes this line of code: "player = my;"
VECTOR temp;
var rot_speed = 20; // 20 gives the rotation speed, play with it
var npc_angle;
var new_line = 0;
while (1)
{
```

```
if (vec_dist (my.x, player.x) < 300)
{
vec_set(temp.x, player.x);
vec_sub(temp.x, my.x);
vec_to_angle(npc_angle, temp.x);
if (ang(npc_angle - my.pan) < -3)
{
my.pan -= rot_speed * time_step;
}
else
{
if (abs(ang(npc_angle - my.pan)) > 3)
{
my.pan += rot_speed * time_step;
}
else // the npc faces the player here, so let's play its line
{
if (!new_line) // the line hasn't been played yet?
{
new_line = 1;
beep();
// plays your line here only once
}
}
}
}
wait (1);
}
}
```

text appear when you touch something and makes it disappear when you move away

```
TEXT* my_text =
{
pos_x = 50;
pos_y = 30;
layer = 10;
string ("Please don't kill me!");
}
function i_am_touched()
{
set(my_text, VISIBLE);
}
action coward_entity()
{
while (!player) {wait (1);}
my.emask |= (ENABLE_IMPACT | ENABLE_ENTITY);
my.event = i_am_touched;
while (1)
{
if (vec_dist (player.x, my.x) > 200)
{
reset(my_text, VISIBLE);
}
wait (1);
}
}
```

camera follow the mouse pointer gently, but stops the camera when the pointer is close to the center of the screen

```
BMAP* arrow_pcx = "arrow.pcx";
function mouse_startup()
{
mouse_mode = 1;
mouse_map = arrow_pcx;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function camera_startup()
{
VECTOR* offset;
while (1)
{
offset.x = mouse_pos.x - screen_size.x / 2;
offset.y = mouse_pos.y - screen_size.y / 2;
// allow a zone of 60 pixels in the center of the screen where the camera doesn't follow the mouse pointer
if (abs(offset.x) > 30)
{
camera.pan -= 0.01 * offset.x * time_step; // 0.01 gives the rotation speed
}
if (abs(offset.y) > 30)
{
camera.tilt -= 0.01 * offset.y * time_step;
}
wait (1);
}

}
```

play footstep sounds in time with my walking animation

```
SOUND* step_wav = "step.wav";
// the action doesn't include a c_move line of code in order to allow you to see the footsteps clearly
action footstep_guy()
{
var anim_percentage;
var sounded_once;
while (1)
{
ent_animate(my, "walk", anim_percentage, ANM_CYCLE);
anim_percentage += 3 * time_step; // "3" controls the animation speed
anim_percentage %= 100;
if (anim_percentage < 10)
{
sounded_once = 1;
}
if ((anim_percentage > 50) && (sounded_once == 1)) // play the first footstep sound at 50%
{
```

```
sounded_once = 2;
ent_playsound(my, step_wav, 200);
}
if ((anim_percentage > 95) && (sounded_once == 2)) // play the second footstep sound at 95%
{
sounded_once = 3;
ent_playsound(my, step_wav, 200);
}
wait (1);
}
}
```

creating a crosshair that is controlled with the arrow keys, while the player can be moved using the WSAD keys

```
BMAP* cross_tga = "cross.tga";
PANEL* crosshair_pan =
{
bmap = cross_tga;
pos_x = 400;
pos_y = 300;
flags = VISIBLE;
}
function crosshair_startup() // centers the crosshair on the screen regardless of the video resolution
{
while (1)
{
crosshair_pan.pos_x = (screen_size.x - bmap_width(cross_tga)) / 2;
crosshair_pan.pos_y = (screen_size.y - bmap_height(cross_tga)) / 2;
wait (1);
}
}
action players_code()
{
player = my; // I'm the player
set (my, INVISIBLE); // no need to see player's model in 1st person mode
while (1)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, GLIDE);
if (key_space) // if the player presses the space key fire a bullet
{
// create your own function that fires bullets or get it from one of the Aum workshops
}
vec_set (camera.x, player.x); // use player's x and y for the camera
camera.z += 30; // place the camera 30 quants above the player on the z axis (approximate eye level)
camera.pan -= 5 * key_force.x * time_step; // rotate the camera around using the arrow keys
camera.tilt += 3 * key_force.y * time_step; // do this on the x and y axis
player.pan = camera.pan; // the camera and the player have the same pan angle
wait (1);
}
}
```

a 2D game that uses 4 different car sprites (for the 4 movement directions) and allows you to control the car using the arrow keys.

```
BMAP* carleft_tga = "carleft.tga";
BMAP* carright_tga = "carright.tga";
BMAP* carup_tga = "carup.tga";
BMAP* cardown_tga = "cardown.tga";
var direction = 0; // 1 = up, 2 = down, 3 = left, 4 = right
PANEL* carleft_pan =
{
bmap = carleft_tga;
pos_x = 400;
pos_y = 300;
}
PANEL* carright_pan =
{
bmap = carright_tga;
pos_x = 400;
pos_y = 300;
}
PANEL* carup_pan =
{
bmap = carup_tga;
pos_x = 400;
pos_y = 300;
}
PANEL* cardown_pan =
{
bmap = cardown_tga;
pos_x = 400;
pos_y = 300;
}
function control_car_startup()
{
set(carleft_pan, VISIBLE); // this will set the initial car orientation
direction = 3;
while (1)
{
if (key_cuu)
{
if (direction == 2)
{
carup_pan.pos_x = cardown_pan.pos_x;
carup_pan.pos_y = cardown_pan.pos_y;
}
if (direction == 3)
{
carup_pan.pos_x = carleft_pan.pos_x;
carup_pan.pos_y = carleft_pan.pos_y;
}
if (direction == 4)
{
carup_pan.pos_x = carright_pan.pos_x;
carup_pan.pos_y = carright_pan.pos_y;
}
while (key_cuu)
{
clamp(carup_pan.pos_y, 0, 570);
clamp(carup_pan.pos_x, 0, 770);
set(carup_pan, VISIBLE);
reset(cardown_pan, VISIBLE);
reset(carleft_pan, VISIBLE);
reset(carright_pan, VISIBLE);
carup_pan.pos_y -= 5 * time_step;
wait (1);
}
direction = 1;
}
if (key_cud)
{
if (direction == 1)
{
cardown_pan.pos_x = carup_pan.pos_x;
cardown_pan.pos_y = carup_pan.pos_y;
}
if (direction == 3)
{
cardown_pan.pos_x = carleft_pan.pos_x;
cardown_pan.pos_y = carleft_pan.pos_y;
}
if (direction == 4)
{
cardown_pan.pos_x = carright_pan.pos_x;
cardown_pan.pos_y = carright_pan.pos_y;
}
while (key_cud)
```

```
{
clamp(cardown_pan.pos_y, 0, 570);
clamp(cardown_pan.pos_x, 0, 770);
reset(carup_pan, VISIBLE);
set(cardown_pan, VISIBLE);
reset(carleft_pan, VISIBLE);
reset(carright_pan, VISIBLE);
cardown_pan.pos_y += 5 * time_step;
wait (1);
}
direction = 2;
}
if (key_cul)
{
if (direction == 1)
{
carleft_pan.pos_x = carup_pan.pos_x;
carleft_pan.pos_y = carup_pan.pos_y;
}
if (direction == 2)
{
carleft_pan.pos_x = cardown_pan.pos_x;
carleft_pan.pos_y = cardown_pan.pos_y;
}
if (direction == 4)
{
carleft_pan.pos_x = carright_pan.pos_x;
carleft_pan.pos_y = carright_pan.pos_y;
}
while (key_cul)
{
clamp(carleft_pan.pos_y, 0, 570);
clamp(carleft_pan.pos_x, 0, 770);
reset(carup_pan, VISIBLE);
reset(cardown_pan, VISIBLE);
set(carleft_pan, VISIBLE);
reset(carright_pan, VISIBLE);
carleft_pan.pos_x -= 5 * time_step;
wait (1);
}
direction = 3;
}
if (key_cur)
{
if (direction == 1)
{
carright_pan.pos_x = carup_pan.pos_x;
carright_pan.pos_y = carup_pan.pos_y;
}
if (direction == 2)
{
carright_pan.pos_x = cardown_pan.pos_x;
carright_pan.pos_y = cardown_pan.pos_y;
}
if (direction == 3)
{
carright_pan.pos_x = carleft_pan.pos_x;
carright_pan.pos_y = carleft_pan.pos_y;
}
while (key_cur)
{
clamp(carright_pan.pos_y, 0, 570);
clamp(carright_pan.pos_x, 0, 770);
reset(carup_pan, VISIBLE);
reset(cardown_pan, VISIBLE);
reset(carleft_pan, VISIBLE);
set(carright_pan, VISIBLE);
carright_pan.pos_x += 5 * time_step;
wait (1);
}
direction = 4;
```

```
}
wait (1);
}
}
```

create a radar that scans around it and highlights all the entities that are within its range

```
action entity_radar() // attach this action to the radar entity
{
while (1)
{
// scan all the entities that are closer than 1000 quants to this entity
c_scan(my.x, my.pan, vector(360, 180, 1000), IGNORE_ME);
wait (1);
}
}
function i_am_scanned()
{
while (event_type == EVENT_SCAN)
{
my.ambient = 100;
wait (1);
}
my.ambient = 100;
}
action scanned_entities() // attach this action to the scanned entities
{
my.emask |= ENABLE_SCAN; // make the entity sensitive to scanning
my.event = i_am_scanned;
}
```

player pick up the diary pages by colliding with them, like walking up to them and they go into the inventory

```
BMAP* diary1_pcx = "diary1.pcx";
BMAP* diary2_pcx = "diary2.pcx";
BMAP* diary3_pcx = "diary3.pcx";
PANEL* diary1_pan =
{
bmap = diary1_pcx;
pos_x = 0;
pos_y = 0;
}
PANEL* diary2_pan =
{
bmap = diary2_pcx;
pos_x = 0;
pos_y = 50;
}
PANEL* diary3_pan =
{
bmap = diary3_pcx;
pos_x = 0;
pos_y = 100;
}
action players_code() // simple player and 1st person camera code
{
player = my; // I'm the player
set (my, INVISIBLE); // no need to see player's model in 1st person mode
while (1)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, GLIDE);
vec_set (camera.x, player.x); // use player's x and y for the camera as well
camera.z += 30; // place the camera 30 quants above the player on the z axis (approximate eye level)
camera.pan -= 5 * mouse_force.x * time_step; // rotate the camera around by moving the mouse
camera.tilt += 3 * mouse_force.y * time_step; // on its x and y axis
player.pan = camera.pan; // the camera and the player have the same pan angle
wait (1);
}
}
action diary1() // attach this to your diary entity (model, sprite, etc)
{
set(my, PASSABLE);
while (!player) {wait (1);} // wait until the player is loaded
while (vec_dist (player.x, my.x) > 70) {wait (1);}
set(diary1_pan, VISIBLE);
ent_remove(my);
}
action diary2() // attach this to your diary entity (model, sprite, etc)
{
set(my, PASSABLE);
while (!player) {wait (1);} // wait until the player is loaded
while (vec_dist (player.x, my.x) > 70) {wait (1);}
set(diary2_pan, VISIBLE);
ent_remove(my);
}
action diary3() // attach this to your diary entity (model, sprite, etc)
{
set(my, PASSABLE);
while (!player) {wait (1);} // wait until the player is loaded
while (vec_dist (player.x, my.x) > 70) {wait (1);}
set(diary3_pan, VISIBLE);
ent_remove(my);
}
```

character selection

```
var character_type = 1; // the first model is the default one
BMAP* arrow_pcx = "arrow.pcx";
STRING* level1_wmb = "test.wmb";
function start_game()
{
level_load (level1_wmb); // now load the "real" level
while (!player) {wait (1);} // wait until the player is loaded
// load the proper player model, depending on the "character_type" value
if(character_type == 1)
{
ent_morph(player, "character1.mdl");
}
if(character_type == 2)
{
ent_morph(player, "character2.mdl");
}
if(character_type == 3)
{
ent_morph(player, "character3.mdl");
}
// add as many characters as you want here
}
function mouse_startup()
{
on_mouse_right = start_game; // click the right mouse button to start the game using the selected character
mouse_map = arrow_pcx; // use an arrow bitmap for the mouse pointer
mouse_mode = 2;
while (mouse_mode > 0)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function set_character()
{
character_type %= 3;
character_type += 1;
if(character_type == 1)
{
```

```
ent_morph(my, "character1.mdl");
}
if(character_type == 2)
{
ent_morph(my, "character2.mdl");
}
if(character_type == 3)
{
ent_morph(my, "character3.mdl");
}
// add as many characters as you want here
}
// attach this action to character1.mdl (your default character model)
action my_character()
{
my.emask |= ENABLE_CLICK;
my.event = set_character;
}
```

bird that would fly randomly in a big outdoor level

```
// place the birds at slightly different heights and pan angles if you plan to use several birds
action pretty_bird()
{
set (my, PASSABLE);
var bird_distance;
var anim_percentage;
while (1)
{
bird_distance = 200 + random(1000); // move 200... 1200 quants in a random direction
my.skill1 = 0;
while (my.skill1 < bird_distance)
{
my.skill1 += c_move (my, vector(10 * time_step, 0, 0), nullvector, IGNORE_PASSABLE);
ent_animate(my, "fly", anim_percentage, NULL);
anim_percentage += 5 * time_step;
anim_percentage %= 100;
wait (1);
}
// now add 10... 60 degrees to the current angle of the bird slowly while it keeps flying
my.skill2 = my.pan + (10 + random(50)); // my.skill2 will store the new angle of the bird
while (my.pan < my.skill2)
{
my.pan += 1 * time_step; // change the rotation speed by playing with 1
// the bird will fly a little slower while it is changing its pan angle
c_move (my, vector(8 * time_step, 0, 0), nullvector, IGNORE_PASSABLE);
ent_animate(my, "fly", anim_percentage, NULL);
anim_percentage += 4 * time_step;
anim_percentage %= 100;
wait (1);
}
// the bird is now oriented in the direction of its new pan angle here and can move another 200...1000 quants
wait (1);
}
}
```

a ball toggle its 2 skins when it collides with the wall

```
var ball_speed = 25;
function ball_event()
{
if (event_type == EVENT_BLOCK) // collided with the level geometry?
{
my.skin %= 2;
my.skin += 1;
}
vec_to_angle (my.pan, bounce);
// don't allow movement in 3D, comment this line if you plan to create a 3D breakout game
my.tilt = 0;
// add some randomness to the ball (we wouldn't want to see it bouncing back and forth between 2 obstacles)
my.pan += 1 - random(2);
}
action my_ball()
{
randomize();
my.pan = 70 - random(140);
my.emask = ENABLE_BLOCK;
my.event = ball_event;
while(1)
{
c_move (my, vector(ball_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE);
wait(1);
}
}
```

a model to follow a predefined path

```
var entity_speed = 3;
var movement_enabled = 0;
var dist_to_node;
var current_node = 1;
var angle_difference = 0;
VECTOR temp_angle;
VECTOR pos_node[3]; // stores the position of the node
function move_target()
{
while(1)
{
if(movement_enabled)
{
entity_speed = minv(5, entity_speed + 0.5 * time_step);
c_move(my, vector(entity_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x));
}
wait(1);
}
}
action move_on_path() // attach this action to your model
{
move_target();
result = path_scan(me, my.x, my.pan, vector(360, 180, 1000));
if (result) {movement_enabled = 1;}
path_getnode (my, 1, pos_node, NULL);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate towards the node
while(1)
{
dist_to_node = vec_dist(my.x, pos_node);
if(dist_to_node < 50) // close to the node?
{
current_node = path_nextnode(my, current_node, 1);
if (!current_node) {current_node = 1;} // reached the end of the path? Then start over!
path_getnode (my, current_node, pos_node, NULL);
}
wait(1);
}
}
```

spawn particles in a circle around my character

```
BMAP px_tga = "px.tga";
function fade_p()
{
my.alpha -= 10 * time_step;
if (my.alpha < 0) {my.lifespan = 0;}
}
function effect_x()
{
```

```
my.alpha = 30 + random(65);
my.flare = on;
my.bmap = px_tga;
my.size = 12;
my.bright = on;
my.move = on;
my.function = fade_p;
}
function create_px()
{
var temp_pos;
var temp_var;
while (1)
{
temp_var += 35 * time_step; // 35 = particle rotation speed
vec_set (temp_pos.x, my.x);
temp_pos.x += 20 * cos(temp_var); // 20 = particle radius
temp_pos.y += 20 * sin(temp_var); // use the same value here
temp_pos.z += 10; // play with 10
effect(effect_x, 1, temp_pos.x, nullvector);
wait (1);
}
}
action players_code() // simple player code
{
create_px();
player = my; // I'm the player
while (1)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, glide);
wait (1);
}
}
```

fade out a sound that is played by snd_loop

```
var loop_handle;
var loop_volume = 100;
sound atmosphere_wav = "atmosphere.wav";
function loop_startup()
{
loop_handle = snd_loop(atmosphere_wav, 80, 0);
while (!key_f) {wait (1);}
while (loop_volume > 1)
{
snd_tune(loop_handle, loop_volume, 0, 0);
loop_volume -= 0.5 * time_step; // 0.5 = fading speed
wait (1);
}
snd_stop (loop_handle);
}
```

model includes an animation named "dance". How can I call this animation when the player hits something, then restore it to "walk" when the "dance" frames end

```
var dancing = 0;
function do_the_dance()
{
var dance_percentage = 0;
dancing = 1;
while (dance_percentage < 100)
{
ent_animate(my, "dance", dance_percentage, anm_cycle);
dance_percentage += 2 * time_step; // "2" controls the "dance" animation speed
wait (1);
}
dancing = 0;
}
action players_code()
{
var anim_percentage;
player = my; // I'm the player
camera.pan = 90;
my.enable_impact = on;
my.enable_entity = on;
my.enable_block = on;
my.event = do_the_dance;
while (1)
{
if (dancing == 0)
{
// move the player using the "W", "S", "A" and "D" keys; "10" = movement speed, "6" = strafing speed
c_move (my, vector(10 * (key_w - key_s) * time_step, 6 * (key_a - key_d) * time_step, 0), nullvector, glide);
if (key_w || key_s || key_a || key_d)
{
ent_animate(my, "walk", anim_percentage, anm_cycle);
anim_percentage += 8 * time_step; // "8" controls the "walk" animation speed
}
else // the player is standing still?
{
ent_animate(my, "stand", anim_percentage, anm_cycle);
anim_percentage += 1 * time_step; // "1" controls the "stand" animation speed
}
}
wait (1);
}
}
```

object orbit around a specified xyz point from my level

```
action object_orbits
{
var orbit_radius = 200; // use your own value here
var orbit_speed = 4; // and here
var temp_angle;
var orbit_center;
vec_set (orbit_center.x, my.x); // store the orbit center position
while(1)
{
my.x = orbit_center.x + sin(temp_angle) * orbit_radius;
my.y = orbit_center.y + cos(temp_angle) * orbit_radius;
my.z = orbit_center.z;
temp_angle += orbit_speed * time_step;
wait(1);
}
}
```

attach particles to the mouse pointer

```
BMAP arrow_pcx = "arrow.pcx";
BMAP snow_tga = "snow.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = arrow_pcx;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function fade_part()
{
my.ALPHA -= 15 * time_step;
if (my.ALPHA < 0) {my.LIFESPAN = 0;}
}
function mouse_particles()
{
var particle_direction;
particle_direction.x = 0.2 * (random(1) - 2);
particle_direction.y = 0.2 * (random(1) - 2);
particle_direction.z = 0.1 * random(1);
vec_add(my.VEL_X, particle_direction);
my.ALPHA = 30 + random(65);
my.FLARE = ON;
my.BMAP = snow_tga;
my.SIZE = 0.5;
my.BRIGHT = ON;
my.MOVE = ON;
my.FUNCTION = fade_part;
}
function particles_startup()
{
var particle_origin;
while (1)
{
particle_origin.x = mouse_pos.x;
particle_origin.y = mouse_pos.y;
particle_origin.z = 40;
vec_for_screen (particle_origin, camera);
effect(mouse_particles, 1, particle_origin.x, normal);
wait (1);
}
}
```

camera follow a path in a loop

```
var cam_speed = 1; // initial speed
var dist_to_node;
var current_node = 1;
var angle_difference = 0;
var temp_angle;
var pos_node; // stores the position of the node
ENTITY* dummy_ent;
function move_dummy()
{
my.PASSABLE = ON;
my.INVISIBLE = ON;
while (1)
{
vec_set (my.x, you.x);
wait (1);
}
}
```

```
function move_target()
{
while(1)
{
cam_speed = minv(15, cam_speed + 5 * time_step); // 15 gives the movement speed
c_move(my, vector(cam_speed * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
vec_to_angle (dummy_ent.pan, vec_diff (temp_angle, pos_node, my.x));
if(my.pan != dummy_ent.pan)
{
angle_difference = ang(dummy_ent.pan - my.pan);
my.pan += angle_difference * 0.2 * time_step;
}
vec_set (camera.x, my.x);
camera.pan = my.pan;
camera.tilt = my.tilt;
wait(1);
}
}
action dummy_target()
{
dummy_ent = ent_create ("dummy.mdl", nullvector, move_dummy);
move_target();
result = path_scan(me, my.x, my.pan, vector(360, 180, 1000));
if (!result) {return;}
path_getnode (my, 1, pos_node, NULL);
vec_to_angle (my.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate towards the node
vec_to_angle (dummy_ent.pan, vec_diff (temp_angle, pos_node, my.x)); // rotate dummy_ent towards the node as well
while (1)
{
dist_to_node = vec_dist(my.x, pos_node);
if(dist_to_node < 300) // close to the node?
{
current_node = path_nextnode(my, current_node, 1);
if (!current_node) {current_node = 1;} // reached the end of the path? Then start over!
path_getnode (my, current_node, pos_node, NULL);
}
wait(1);
}
}
```

a model animate using "walk" for 10 seconds, then "jump" for 10 seconds, "walk" again for 10 seconds, and so on

```
action two_animations()
{
var anim_speed;
while (1)
{
my.skill88 = 0;
while (my.skill88 < 10) // play this animation for 10 seconds
{
my.skill88 += time_step / 16;
anim_speed += 2 * time_step; // 2 = walk animation speed
ent_animate(my, "walk", anim_speed, ANM_CYCLE);
wait (1);
}
my.skill88 = 0;
while (my.skill88 < 10) // play this animation for 10 seconds
{
my.skill88 += time_step / 16;
anim_speed += 3 * time_step; // 3 = jump animation speed
ent_animate(my, "jump", anim_speed, ANM_CYCLE);
wait (1);
}
wait (1);
}
}
```

a random soundtrack in my game

```
var track_number;
var track_handle;
function random_tracks_startup()
{
while (1)
{
track_number = 1 + total_frames % 5; // use 5 sound tracks in this example
if (track_number == 1)
{
track_handle = media_play("track1.wav", NULL, 50);
}
if (track_number == 2)
{
track_handle = media_play("track2.wav", NULL, 50);
}
if (track_number == 3)
{
track_handle = media_play("track3.wav", NULL, 50);
}
if (track_number == 4)
{
track_handle = media_play("track4.wav", NULL, 50);
}
if (track_number == 5)
{
track_handle = media_play("track5.wav", NULL, 50);
}
while (media_playing (track_handle)) {wait (1);} // wait until the current track has ended
}
}
```

trying to make my own button with 3 different images of the same size,

```
// main panel bitmap
BMAP* mypanel_pcx = "mypanel.pcx";
// bitmaps used for the buttons
BMAP* pictureon_pcx = "pictureon.pcx";
BMAP* pictureoff_pcx = "pictureoff.pcx";
BMAP* pictureover_pcx = "pictureover.pcx";
function my_function(); // function prototype
PANEL* my_pan =
{
layer = 15;
button(40, 10, pictureon_pcx, pictureoff_pcx, pictureover_pcx, my_function, NULL, NULL);
flags = SHOW;
}
// just a simple function that runs when the player clicks the button
function my_function()
{
beep();
}
```

get the coordinates of the model which was hit by a c_trace ray

```
VECTOR trace_start, trace_end, hit_coords;
function fire_weapon()
{
while (mouse_left)
{
vec_set (trace_end.x, vector(10000, 0, 0)); // the c_trace ray has a lenght of 10,000 quants
vec_rotate (trace_end.x, camera.pan);
vec_add (trace_end.x, camera.x);
vec_set (trace_start.x, vector(0, 6, 0)); // the c_trace ray starts a bit offset from the
center of the camera - this makes the red ray visible for the player
vec_rotate (trace_start.x, camera.pan);
vec_add (trace_start.x, camera.x);
// trace from the center of the camera 10,000 quants in front of it
c_trace (trace_start.x, trace_end.x, IGNORE_ME);
draw_line3d(trace_start.x, NULL, 10000);
draw_line3d(trace_end.x, vector(0, 0, 255), 100);
if(you) // got an entity?
{
vec_set(hit_coords, target.x); // then get the coordinates of the hit point!
}
else
{
vec_set(hit_coords, nullvector); // display zero if the ray didn't hit an entity
}
wait (1);
}
}
function init_startup()
{
on_mouse_left = fire_weapon;
}
PANEL* hit_pan = // displays the hit point coordinates
{
layer = 15;
digits(10, 30, "Hit coordinate x: %.f", *, 1, hit_coords.x);
digits(10, 50, "Hit coordinate y: %.f", *, 1, hit_coords.y);
flags = SHOW;
}
```

a panel that pops up with a button on it. The button closes the panel correctly, but it doesn't disappear together with the panel.

```
BMAP* mypanel_pcx = "mypanel.pcx";
BMAP* pictureon_pcx = "pictureon.pcx";
BMAP* pictureoff_pcx = "pictureoff.pcx";
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function close_panel(); // function prototype
PANEL* my_pan =
{
bmap = mypanel_pcx;
layer = 15;
button(40, 10, pictureon_pcx, pictureoff_pcx, pictureon_pcx, my_function, NULL, NULL);
flags = SHOW;
}
// hides the panel as soon as the player click the button
function my_function()
{
reset(my_pan, SHOW);
}
```

a circular mask using two different panels? The first panel would be the one of the front and the other one would be the background

```
BMAP* front_pcx = "front.pcx";
BMAP* background_pcx = "background.pcx";
BMAP* pointer_tga = "pointer.tga";
function mouse_startup()
{
mouse_mode = 2;
mouse_map = pointer_tga;
while (1)
{
vec_set(mouse_pos, mouse_cursor);
wait(1);
}
}
function front_clicked(); // function prototype
PANEL* front_pan =
{
bmap = front_pcx;
layer = 20; // appears over background_pcx;
on_click = front_clicked; // runs this function when it is clicked
flags = SHOW;
}
PANEL* background_pan =
{
bmap = background_pcx;
layer = 10; // appears below front_pcx;
flags = SHOW;
}
// a simple function that runs when the front panel is clicked
function front_clicked()
{
beep();
}
```

an object that rotates. On this object another object should be placed, acting as a flickering light

```
VECTOR light_pos;
function blinking_lights()
{
while (1)
{
// get the coordinates of the 10th vertex on the sphere model and keep the light at that
position
vec_for_vertex(my.x, you, 10);
vec_set(my.blue, vector(0, 0, 255)); // our light will have a blue color
my.lightrange = 30; // and a range of 30 quants
wait (-0.1); // keep the light on for 0.1 seconds
my.lightrange = 0;
wait (-0.1); // then turn it off for 0.1 seconds
}
}
action my_rotator()
{
// create a NULL entity - you can also use something like "light.mdl" if you need visible
light generating models
ent_create(NULL, nullvector, blinking_lights);
while (1)
{
my.pan += 3 * time_step; // rotate the sphere around its pan, tilt, roll angles
my.tilt += 2 * time_step;
my.roll += 1 * time_step;
wait (1);
}
}
```

a 3d model and want it coming out of the ground little by little, like a line of grass growing up on the ground.

```
action growing_grass()
{
var grass_height; // stores the vertical size of the fully grown grass model
var temp_z; // stores the initial grass height value
set(my, INVISIBLE);
c_setminmax(me); // set my bounding box of the grass model to its real size
grass_height = my.max_z - my.min_z; // now compute the real height of the grass model
temp_z = my.z;
my.z -= grass_height; // move the grass downwards until it's prepared to grow (provided that
it was placed on the ground properly, of course)
wait (-5); // wait for 5 seconds before having the grass grow
reset(my, INVISIBLE);
while (my.z < temp_z) // move the grass model upwards until it returns to its initial height
{
my.z += 0.5 * time_step; // 0.5 sets the growing speed
wait (1);
}
}
```

How do I make my models have shadows on terrain entities? I've tried everything...

shadow_stencil = 2; // choose different shadow_stencil values here, depending on your needs - check the manual for more info
Then, set the "shadow" flag for the entities that are supposed to produce shadows from within Wed. Check the "cast" flag as well if you'd like the entities to also project shadows on themselves.

an entity turn and run towards the player

```
VECTOR chaser_dest;
ANGLE chaser_angle;
action player_chaser()
{
var stand_percentage, run_percentage;
// wait until the player model is loaded
while (!player) {wait (1);}
while (1)
{
if (vec_dist(player.x, my.x) > 500) // the player is away from the chaser?
{
ent_animate(my, "stand", stand_percentage, ANM_CYCLE); // then play the "stand" animation
stand_percentage += 2 * time_step; // 2 = "stand" animation speed
}
else // the player has come closer than 500 quants to the chaser here
{
if (vec_dist(player.x, my.x) > 150) // the chaser didn't approach the player close enough yet?
{
vec_set(chaser_dest, player.x);
vec_sub(chaser_dest, my.x);
vec_to_angle(chaser_angle, chaser_dest);
my.pan += ang(chaser_angle.pan - my.pan) * 0.1 * time_step;
ent_animate(my, "run", run_percentage, ANM_CYCLE); // then play the "run" animation
c_move (my, vector(10 * time_step, 0, 0), nullvector, IGNORE_PASSABLE | GLIDE);
run_percentage += 6 * time_step; // 6 = "run" animation speed
}
else
{
// the chaser is close enough to the player here, so it stops for now
// I chose to play the same stand animation, but feel free to do your own thing
ent_animate(my, "stand", stand_percentage, ANM_CYCLE); // then play the "stand" animation
stand_percentage += 2 * time_step; // 2 = "stand" animation speed
}
}
wait (1);
}
}
```