Criado no Game Maker utilizando o GML (game maker language)

Create é usado principalmente para inicializar variáveis e definir configurações iniciais do objeto. Ele é executado uma vez quando o objeto é criado na room, garantindo que todas as variáveis tenham um valor inicial antes de qualquer outro evento ser executado.

O evento Step no GameMaker é usado para executar lógica continuamente a cada frame do jogo. É o evento mais comum para atualizar estados e comportamentos de objetos, já que ele é chamado em cada ciclo de atualização, permitindo que você controle a lógica de movimentação, física, IA, e interações em tempo real.

Botão iniciar :

create:

```
som_ja_tocado = false;
```

left pressed (botão do mouse) :

```
o_fade_control.fading_out = true;
o_fade_control.target_room = paper_slash;
if (mouse_check_button_pressed(mb_left)) {
  if (!som_ja_tocado) {
    audio_play_sound(s_button_press, 1, false);
                audio_stop_sound(s_menu_music);
    som_ja_tocado = true; // Marca o som como tocado
  }
}
```

botão sair :

```
game_end();
```

o player:

create:

```
event_inherited();


s_idle = s_player;
s_move = s_player;
s_dead = s_player_dead;
//---------------------------------------
hp_max = 20;                    //hp do jogador
hp = hp_max;
//---------------------------------------
stamina = 30;                   //stamina do jogador
stamina_max = stamina;
stamina_recover_rate = 0.2;
stamina_recover_delay = 45;
stamina_recover_timer = 0;
//---------------------------------------
hearth_width = 750;             //imagem do coração para o hud
hearth_height = 50;
hearth_x = 12;
hearth_y = 25;
//---------------------------------------
healthbar_width = 315;          //imagem da barra de vida para o hud
healthbar_height = 16;
healthbar_x = 70
healthbar_y = 30
//---------------------------------------
staminabar_width = 315;         //imagem da barra de stamina para o hud
```

```
staminabar_hight = 16;

staminabar_x = 70

staminabar_y = 55

//---------------------------------------

dead = false;

//---------------------------------------

knockback_time = 0;

//---------------------------------------

// Velocidade normal e velocidade de dash

walk_speed = 0.6;

run_speed = 1.5;

dash_speed = 3;

dash_duration = 0.2 * room_speed; // Duração do dash em frames (0.5 segundos)

smoke_timer = 0;

//---------------------------------------

// Variáveis de controle do dash

is_dashing = false;

//---------------------------------------

afterimage_count = 4; // Número de afterimages a serem criados

afterimage_delay = 2; // Tempo de atraso entre cada afterimage (em frames)

afterimage_remaining = 0;    // Contador para afterimages restantes a serem
criadas

//---------------------------------------

attack_cooldown = 0;

//---------------------------------------

combo_state = 0;

combo_timer = 0;

combo_time_limit = 50; // ajuste o tempo limite para o próximo ataque em frames
```

```
//---------------------------------------
score_p = 0;
best_score = 0;
//---------------------------------------
timer = 5 * 60 * 60; // tempo do jogo
```

step:

```
switch(state){
	default:
		olhando_mouse();
		reset_variabels();
		get_input()
		calc_player_movement()
		dash();
		player_anim();
	break;
	case states.DEAD:
		instance_create_layer(o_camera.x, o_camera.y, "dead", o_morreu);

		reset_variabels();
		calc_player_movement();
		sprite_index = s_dead;
	break;
	case states.KNOCKBACK:
		player_anim();
		if knockback_time-- <= 0 state = states.IDLE;
	break;
```

```
}
```

```
//------------------------------
// Verifica se a tecla de espaço foi pressionada e se há stamina suficiente
if (keyboard_check_pressed(vk_space) && stamina >= 5) {
   stamina -= 5;          // Reduz a stamina em 5
   stamina_recover_timer = 0; // Reinicia o temporizador quando o jogador usa stamina
}

// Inicia a recuperação de stamina após o atraso definido
if (stamina < stamina_max) {
   stamina_recover_timer++; // Incrementa o temporizador

   // Verifica se o temporizador atingiu o atraso necessário para iniciar a recuperação
   if (stamina_recover_timer >= stamina_recover_delay) {
      stamina += stamina_recover_rate; // Recupera stamina

      // Garante que a stamina não ultrapasse o máximo
      if (stamina > stamina_max) {
         stamina = stamina_max;
      }
   }
}
```

```
//------------------------------------------------------------------------



if stamina >= 5 && (keyboard_check_pressed(vk_space)){
   // Inicia a sequência de afterimages
   afterimage_remaining = afterimage_count;  // Configura a quantidade de afterimages a serem criadas
   alarm[0] = afterimage_delay;           // Inicia o primeiro alarm para o delay
}


if (hp <= 0) {
   mask_index = -1;
}


//------------------------------------------------------------------------
// Reduz o cooldown, se estiver ativo
if (attack_cooldown > 0) {
   attack_cooldown--;
}


// Verificar se o botão do mouse 1 (esquerdo) foi pressionado
if (mouse_check_button_pressed(mb_left) && attack_cooldown == 0 && hp > 0 && stamina >= 1 ) {
   if (combo_state == 0 || (combo_state > 0 && combo_timer < combo_time_limit)) {
      combo_state += 1;
      combo_timer = 0; // Reseta o timer após cada ataque


                  if (combo_state == 1){
```

```
                instance_create_layer(x, y, "Instances", oSwordAttack);
                stamina -= 2;
                stamina_recover_timer = 0;
                audio_play_sound(sound_attack1, 1, false);
                attack_cooldown = 21;

                }
                else if (combo_state == 2){
                instance_create_layer(x, y, "Instances", oSwordAttack2);
                stamina -= 3;
                stamina_recover_timer = 0;
                audio_play_sound(sound_attack2, 1, false);
                attack_cooldown = 21;
                }
                else if (combo_state == 3){
                instance_create_layer(x, y, "Instances", oSwordAttack3);
                stamina -= 4;
                stamina_recover_timer = 0;
                combo_state = 0;
                audio_play_sound(sound_attack1, 1, false);
                attack_cooldown = 21;
                }
    }
}

// Incrementa o combo_timer a cada frame
if (combo_state > 0) {
    combo_timer += 1;
```

```
    if (combo_timer > combo_time_limit) {
        combo_state = 0; // Reseta o combo se o tempo limite for excedido
        combo_timer = 0;
    }
}

if (timer > 0) {
    timer--;
} else {
            ini_open("temp_data.ini");
            ini_write_real("score_p", "best_score", global.best_score);
            ini_close();
            gmda_submit(1, global.best_score);
    game_restart(); // Reinicia o jogo quando o tempo acaba
}



function collision(){
            //set target values
            var _tx = x;
            var _ty = y;

            //move back to last step position, out of the collisions
            x = xprevious;
            y = yprevious;

            //get distance we want to move
            var _disx = ceil(abs(_tx - x));
```

```
        var _disy = ceil(abs(_ty - y));
        var _collision = [o_parede, o_parede2];

        if place_meeting(x + _disx * sign(_tx - x), y, _collision) x = round(x);
        if place_meeting(x, y + _disy * sign(_ty - y), _collision) y = round(y);

        //move as far as in x and y before hitting the solid
        repeat(_disx){
                if !place_meeting(x + sign(_tx - x), y, _collision) x += sign(_tx - x);
        }
        repeat(_disy){
  if !place_meeting(x, y + sign(_ty - y), _collision) y += sign(_ty - y);
        }

}

if (score_p > global.best_score) {
  global.best_score = score_p; // Atualiza a melhor pontuação
}
```

SCRIP DAS FUNÇÕES DO JOGADOR QUE ESTÃO NO SWITCH(STATE)

```
function reset_variabels(){
        up = 0;
        down = 0;
        left = 0;
        right = 0;
```

```
        vmove = 0;
        hmove = 0;
}

function get_input(){

        if keyboard_check(ord("A")) left  = 1;
        if keyboard_check(ord("D")) right = 1;
        if keyboard_check(ord("W")) up    = 1;
        if keyboard_check(ord("S")) down  = 1;



}

function olhando_mouse(){
        image_angle = point_direction(x, y, mouse_x, mouse_y);
}



function dash(){



        // Inicia o dash se espaço for pressionado e o personagem não está dashing
        if (keyboard_check_pressed(vk_space) && !is_dashing && stamina >= 5) {
           is_dashing = true;
           dash_timer = dash_duration;
        }
```

```
// Checa se o dash está ativo
if (is_dashing) {
    // Define a velocidade para o dash
    var current_speed = dash_speed;
    dash_timer -= 1;

    // Verifica se o dash terminou
    if (dash_timer <= 0) {
        is_dashing = false;
    }
} else {
    // Define a velocidade normal quando não está dashing
    var current_speed = walk_speed;
}

// Movimento usando W, A, S, D com a velocidade atual (normal ou dash)
if (keyboard_check(ord("W"))) {
    y -= current_speed;
}
if (keyboard_check(ord("S"))) {
    y += current_speed;
}
if (keyboard_check(ord("A"))) {
    x -= current_speed;
}
if (keyboard_check(ord("D"))) {
    x += current_speed;
}
```

```
}

function calc_player_movement(){

        // Define a velocidade padrão de caminhada e corrida
  var _speed = walk_speed;
        var pitch = 1; // Pitch normal
        // Verifica se a tecla de corrida está pressionada
  if (keyboard_check(vk_shift)) && stamina >= 1 {
    _speed = run_speed;
                pitch = 2; // Pitch mais alto para corrida
                stamina -= 0.02;
                stamina_recover_timer = 0;
                if (smoke_timer <= 0) {
      instance_create_layer(x, y, "Instances", o_run);
      smoke_timer = 20; // Tempo para reaparecer a fumaça (em frames)
    }
  }
        hmove = right - left;
        vmove = down - up;

        if (hmove != 0 || vmove != 0) {
    // Toca o som de andar/correr, caso não esteja tocando
    if (!audio_is_playing(walk_sound)) {
      audio_play_sound(walk_sound, 1, true);
    }
    // Ajusta o pitch do som de acordo com a velocidade
```

```
        audio_sound_pitch(walk_sound, pitch);
        // Obtém a direção e distância para o movimento
        var _dir = point_direction(0, 0, hmove, vmove);
        hmove = lengthdir_x(_speed, _dir);
        vmove = lengthdir_y(_speed, _dir);
        // Adiciona movimento à posição do jogador
        x += hmove;
        y += vmove;
    } else {
        // Para o som se o jogador não está se movendo
        if (audio_is_playing(walk_sound)) {
            audio_stop_sound(walk_sound);
        }
    }
    x += hsp;
    y += vsp;
    switch(state) {
        default: var _drag = 0.1; break;
        case states.DEAD: var _drag = 0.06; break;
    }
    hsp = lerp(hsp, 0, _drag);
    vsp = lerp(vsp, 0, _drag);
            if (smoke_timer > 0) {
        smoke_timer--;
    }
}
function player_anim(){
                    if hmove != 0 or vmove != 0 {
```

```
                sprite_index = s_move;
        } else {
                sprite_index = s_idle;
        }
}
```

draw gui do personagem;

```
draw_sprite_stretched(s_hearth, 0, hearth_x, hearth_y, hearth_width,
hearth_height);

draw_sprite(s_border_bg, 0, healthbar_x, healthbar_y);

draw_sprite_stretched(s_health, 0, healthbar_x, healthbar_y, (hp / hp_max) *
healthbar_width, healthbar_height);

draw_sprite(s_border, 0, healthbar_x, healthbar_y);

draw_sprite(s_border_bg, 0, staminabar_x, staminabar_y);

draw_sprite_stretched(s_stamina, 0, staminabar_x, staminabar_y, (stamina /
stamina_max) * staminabar_width, staminabar_hight);
```

```
draw_sprite(s_border, 0, staminabar_x, staminabar_y);



draw_text(70, 85, "Pontos: " + string(score_p));
draw_text(70, 105, "Melhor Score: " + string(global.best_score));
var seconds = timer div 60; // Converte o tempo para segundos
var minutes = seconds div 60; // Converte para minutos
seconds = seconds mod 60; // Resto da divisão para segundos
draw_text(70, 125, "Tempo: " + string(minutes) + ":" + string(seconds));
```

desenha na tela o que acontece com vida, stamina, seus pontos e tempo