MANUAL DE DESENVOLVIMENTO
BATALHA DE MATEMÁTICA

Obj_Game

```
rollback_define_input(
{
    left: ord("A"),
    right: ord("D"),
    up: ord("W"),
    down: ord("S"),
    r: ord("R"),
    fire: mb_left,
    mb_x: m_axisx,
    mb_y: m_axisy,
});

rollback_define_player(obj_player, "Instances");

if (!rollback_join_game())
{
        rollback_create_game(2, false);
}
```

1. Configuração das Entradas do Usuário

Para começar, utilizamos a função `rollback_define_input` para coletar os tipos de entrada do usuário. Estes são posteriormente armazenados em variáveis específicas. Por exemplo, ao pressionar a tecla "w", a informação correspondente será armazenada na variável "up".

2. Definição do Objeto Jogável

Na função `rollback_define_player`, procedemos à definição do objeto jogador. Este objeto, central para a experiência de jogo, recebe um identificador único, neste caso, "instances".

3. Lógica Condicional para Criação de Sala

Implementamos uma estrutura condicional que pode ser interpretada como: "se a conexão não estiver no estado 'entrando', crie uma sala". Para esta verificação, utilizamos a função `rollback_join_player`, que detecta se há outros jogadores se conectando. Caso seja necessário criar um novo jogo, empregamos a função

`rollback_create_game`. Esta última requer dois parâmetros: o número de jogadores desejado e um booleano indicando se o jogo está online (false) ou offline (true).

Resumo dos Passos:

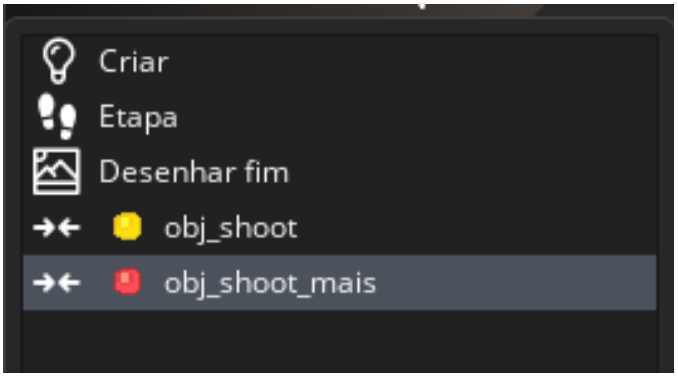
1. Configuração de Entradas do Usuário:
   - Usamos `rollback_define_input` para coletar tipos de entrada do usuário.

2. Definição do Objeto Jogável:
   - Utilizamos`rollback_define_player` para criar e identificar o objeto jogador.

3.Lógica Condicional para Criação de Sala:
   - Implementamos uma estrutura condicional usando `rollback_join_player` e `rollback_create_game` para gerenciar a criação de salas de jogo.

Obj_Player

Eventos necessários no objeto player:



. Criar;
. Etapa;
. Desenhar fim;
. Dois eventos de colisão.

Criando as variáveis para o nosso player(Evento criar)

```
shoot = 0;
direita = 0;
esquerda = 0;
cima = 0;
baixo = 0;
r = 0;


global.start = false;

spd = 3.5;
hspd = 0;
vspd = 0;
cooldown = 0;
life_mais = 2;
life_menos = -2;
dano = 1;
reiniciar = 0;
vencedor = "";


gravidade = 0.3;
```

Nascimento dos personagens:

```
if (player_id == 0)
{
        x = 50;
        y = room_height / 2;
        sprite_index = spr_idle_menos
}
else if (player_id == 1)
{
        x = room_width - 50;
        y = room_height / 2;
        image_xscale = -1;

}
```

Ao iniciar o jogo, ocorre automaticamente a criação de duas instâncias, representando nosso objeto jogador (obj_player), conforme previamente definido no objeto de jogo (obj_game). A distinção entre essas duas instâncias é determinada pelos identificadores (id's) associados a cada uma. Neste contexto, realizamos uma verificação específica dos id's para assegurar que os jogadores nasçam em locais distintos, garantindo uma distribuição espacial inicial equitativa.

Desenvolvimento da movimentação do nosso player (Evento etapa).

```
//MOVIMENTAÇÃO
var _input = rollback_get_input();
direita = _input.right;
esquerda = _input.left;
cima = _input.up;
baixo = _input.down;
shoot = _input.fire;
r = _input.r

hspd = (direita - esquerda) * spd;

vspd = (baixo - cima) * spd;
```

Recorremos à função `rollback_get_input` para capturar as entradas dos usuários, armazenando essas informações em variáveis específicas do GameMaker. A variável `hspeed` (horizontal speed) desempenha um papel fundamental ao controlar o movimento horizontal do nosso jogador, enquanto a variável `vspeed` (vertical speed) governa seu deslocamento vertical. Em essência, `hspeed` e `vspeed` são as variáveis-chave que impulsionam o movimento ao longo do eixo X e Y do nosso personagem, respectivamente.

Condições para nosso personagem andar e colidir, além de deixar a gravidade funcionando.

```
if !place_meeting(x, y + 1, obj_chao){
vspd += gravidade;
}

if place_meeting(x + hspd, y, obj_chao){
    while !place_meeting(x + sign(hspd), y, obj_chao){
         x += sign(hspd)
     }
hspd = 0;
}

x += hspd;

if place_meeting(x , y + vspd, obj_chao){
    while !place_meeting(x, y + sign(vspd), obj_chao){
         y += sign(vspd);
     }
vspd = 0;
}

y += vspd;
```

link do vídeo que explica melhor a movimentação e a gravidade:
Movimentação e Colisão | Fazendo um jogo Plataforma #1| GMS2

Controlando as sprites do player:

```
//CONTROLE DE SPRITES
if(hspd != 0){

if(player_id == 0){

sprite_index = spr_run_menos;
image_xscale = sign(hspd);

}else{
sprite_index = spr_run;
image_xscale = sign(hspd);
}
}else{
if(player_id == 0){

sprite_index = spr_idle_menos;

}else{
sprite_index = spr_idle;
}
}
```

Criação do projétil:

Para isso usamos instance_create_layer passando os parâmetros necessários.

```
if (shoot)
{

    if(cooldown <= 1){

        if(player_id == 1){
         var _proj = instance_create_layer(x+1, y+2, layer, obj_shoot_mais);
        _proj.image_angle = point_direction(x, y, _input.mb_x, _input.mb_y);
        _proj.speed = 4;
        _proj.direction = point_direction(x, y, _input.mb_x, _input.mb_y);
        _proj.player = self;
        cooldown = 40;
        }else{

        var _proj = instance_create_layer(x+1, y+2, layer, obj_shoot);
        _proj.image_angle = point_direction(x, y, _input.mb_x, _input.mb_y);
        _proj.speed = 4;
        _proj.direction = point_direction(x, y, _input.mb_x, _input.mb_y);
        _proj.player = self;
        cooldown = 40;
        }
    }
}

cooldown -= 1;
```

Restart para que não seja necessário atualizar a página:

```
if(global.start){
instance_destroy(obj_shoot);
instance_destroy(obj_shoot_mais);
reiniciar += 1;
if(reiniciar == 400){
    global.start = false;
    vencedor = "";
}
}
```

TXT(desenho fim)

Criando um texto em cima do nosso obj_player e estrutura condicional para exibir um texto com o vencedor.

```
if (global.start == true)
{
draw_set_color(c_black);
draw_text(200,(room_height/2)-50,"O operador mais forte e o " + vencedor + ".\n Espere para jogar novamente"

}

draw_set_colour (c_black);
if(player_id == 1){
draw_text(x - 10, y - 40,life_mais);
}else{
draw_text(x - 10, y - 40,life_menos);
}
```

Tiro(Colisão obj_shoot)

```
if (other.player == self) exit;

x = irandom_range(40, room_width - 40);

y = irandom_range(40, room_height - 40);

instance_destroy(other);
```

Destruímos a instância do outro objeto e quando destruída, ressurge em um X e um um Y aleatório.

Bloco de código que auxilia no restart do jogo e define o vencedor.

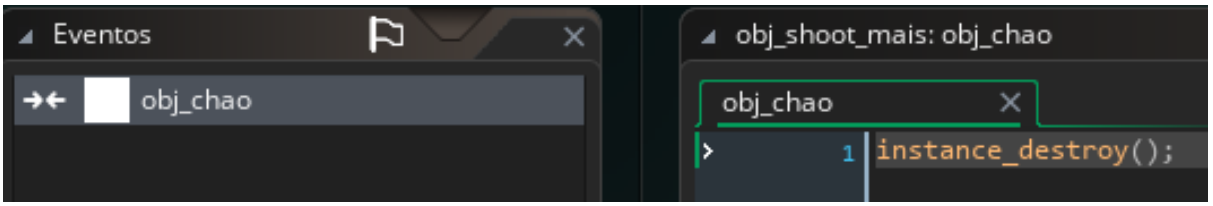
```
if(life_mais == 1){
global.start = true;
}


if(global.start){
life_mais = 11;
other.player.life_menos = -10;
vencedor = "menos";
}


if(player_id == 1){
self.life_mais = life_mais - dano;
reiniciar = 0;
other.player.reiniciar = 0;
other.player.vencedor = "";
}else{
self.life_menos = life_menos + dano;
reiniciar = 0;
other.player.reiniciar = 0;
other.player.vencedor = "";
}
```

Mesmo código só que para o tiro do outro player: “Mais”

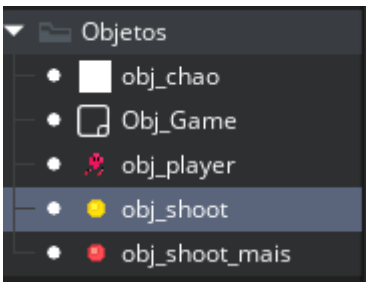
```
if(life_menos == -1){
global.start = true;
}

if(global.start){
life_menos = -11;
other.player.life_mais = 10;
vencedor = "mais";
}


if(player_id == 1){
self.life_mais = life_mais - dano;
reiniciar = 0;
other.player.reiniciar = 0;
other.player.vencedor = "";
}else{
self.life_menos = life_menos + dano;
reiniciar = 0;
other.player.reiniciar = 0;
other.player.vencedor = "";
}
```

Obj_Shoot

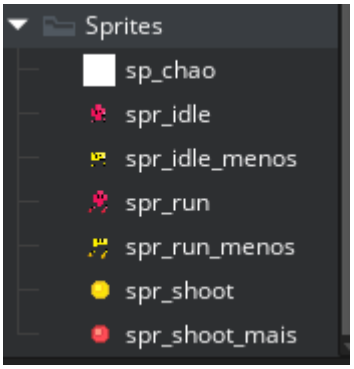
A única interação programada para nosso projétil é a detecção de colisão com as paredes. É relevante destacar que possuímos dois objetos distintos para os tiros, um designado para o menor e outro para o maior. Ambos compartilham idênticos eventos e códigos de programação, proporcionando uma abordagem consistente e eficiente para a gestão das ações associadas a esses elementos no jogo.



Objetos:



Sprites:



[Documentação rollback do gamemaker]