

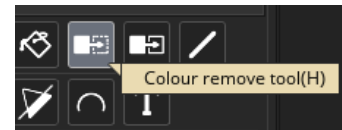
– EXERCÍCIO 2 – GameMaker Studio 2

Objetivo:

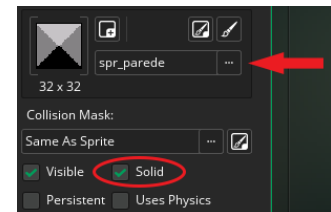
Um jogo de labirinto onde o jogador deve andar até a saída. Em algumas fases ele precisa pegar itens (diamantes) para poder abrir uma porta que tranca a saída.

Passo 1 – definindo os objetos iniciais:

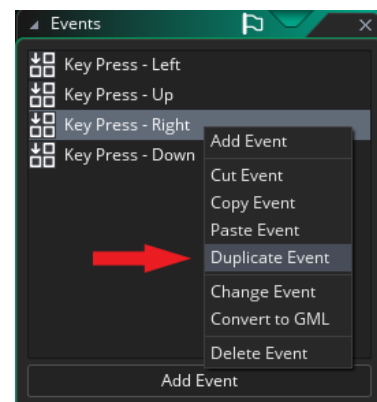
- Crie um sprite, carregue o arquivo person.gif, coloque nome: spr_jogador, remova a cor de fundo.
- Crie outro sprite, carregue o arquivo goal.gif, coloque nome: spr_bandeira, remova a cor de fundo.
- Crie mais um sprite, carregue o arquivo wall.gif, coloque nome: spr_parede



- Crie um objeto, coloque nome: obj_parede, selecione para ele o sprite da parede, ligue a opção “solid”.



- Crie outro objeto, coloque nome: obj_jogador, selecione para ele o sprite do jogador (urso).
- Para fazer ele andar com as setas do teclado, adicione o evento Key Pressed, opção direita (right), coloque a ação Set Direction Fixed, clicando na seta da direita. Após adicione a ação Set Speed, com velocidade 4.
- Repita a operação para todas as outras direções (esquerda, cima, baixo / left, up, down). A forma mais prática de fazer isso é usar o clique da direita sobre o evento e escolher duplicar, selecionando então as outras direções. Agora é só ir em cada evento e clicar na seta correta de acordo com a direção em Set Direction Fixed.

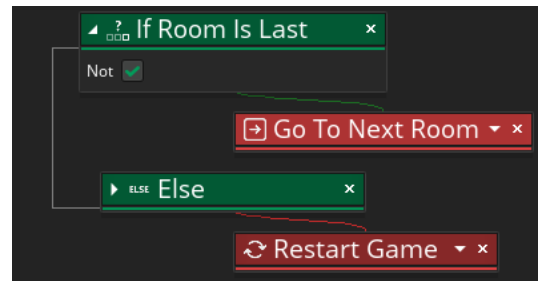


- Abra a sala de jogo já existente (room0) e adicione o objeto do jogador (urso), arrastando ele da lista Resources (na direita) para dentro da sala. Teste o jogo.

Passo 2 – refinando movimentação e criando a saída:

Bem, o urso está andando, mas ele segue eternamente... Ele deve parar quando o jogador soltar a seta do teclado.

- Para isso, no objeto do jogador crie o evento Key Down – No Key, com a ação Set Speed, velocidade 0 (padrão). Teste novamente o jogo. (obs: este evento é constantemente acionado quando nenhuma tecla está sendo pressionada)
- Crie um objeto, coloque nome: obj_bandeira (será o local onde o jogador deve ir). Escolha para ele o sprite da bandeira.
- Quando a bandeira colidir com o jogador, deve passar para próxima fase. Porém se não houver outra fase (já estiver na última) o jogo dará um erro. Então devemos usar uma ação que testa se existe uma outra fase, para só depois ir para a próxima.
 - Dentro do objeto bandeira crie o evento Collision – com o objeto do jogador, adicione a ação If Room Is Last (testa se estamos na última fase do jogo), como queremos testar se NÃO estamos na última fase, clique na opção Not.
 - Depois adicione Go To Next Room (deve ser encaixado no lado direito da ação anterior, pois ela é um teste).
 - Se não existir a próxima fase iremos reiniciar o jogo, então coloque a ação Else (encaixado abaixo do If Room Is Last, para indicar o que fazer no caso contrário do teste), após adicione a ação Restart Game (encaixado no lado direito do Else).



Verifique se tudo ficou exatamente como na imagem.

Nota: qualquer ação que é um teste condicional pode conter o Else para informar o que fazer se a condição for falsa.

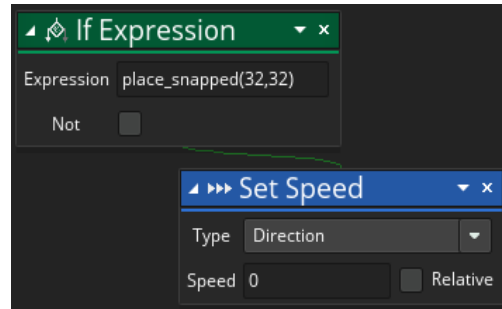
- Coloque algumas paredes na sala, formando o labirinto (com Alt pode adicionar vários objetos arrastando o mouse). **Falta colisão com as paredes!**

Passo 3 – colisão com parede:

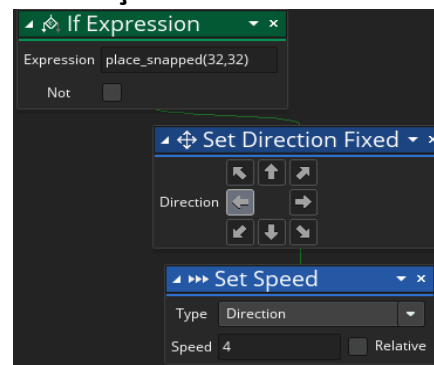
- Dentro do objeto do jogador, crie o evento Collision – com a parede, adicione ação Set Speed, velocidade 0 (padrão).

Agora ele colide com as paredes, porém está difícil de fazer “curvas”, de trocar de direção entre paredes estreitas. Isto ocorre pois o urso está com movimentação livre, seria melhor deixá-lo preso em um grid, para que não raspe em “bordas” das paredes, deixando o jogo muito melhor de jogar.

- Existe uma função (não disponível em ação drag and drop) que testa se o objeto está alinhado no grid da sala. Somente se ele estiver alinhado poderá mover e parar.
- Para isso, dentro do objeto do jogador, no evento já existente Key Down - No Key, coloque a ação If Expression (permite testar se algum código retorna positivo) e onde diz Expression digite `place_snapped(32,32)`
- Movimente a ação Set Speed que está ali para encaixar no lado direito do If Expression, desta forma o Set Speed só será executado se a expressão for positiva.



- Da mesma forma, para deixar o controle ainda melhor, coloque a mesma ação – If Expression com `place_snapped(32,32)` – como a **primeira ação** em todos os eventos de Key Press <left>,<right>,<up>,<down> (arrastando as outras ações para encaixar no lado direito dele). Isto fará com que o personagem só possa ser movimentado se estiver alinhado no grid (caso contrário ao tentar movimentar na diagonal ele poderá ficar desalinhado). Exemplo na imagem do Key Press – Left.

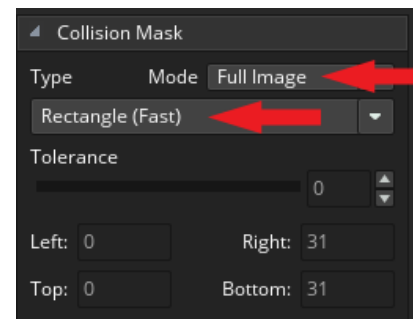


Sistema de colisão:

Há várias formas de um sprite colidir com outro: pode ser usado forma precisa (qualquer pixel que não seja transparente contará na colisão) ou outras opções. Se a forma precisa não for necessária, é melhor que o sprite colida usando um simples retângulo (conhecido como “bounding box”); isto pode evitar problemas de colisão (como o sprite trancar em quinas, de acordo com seu desenho) além de ser mais leve para processar. Então use a opção de colisão precisa somente quando for necessário.

Neste exercício o melhor é deixar o urso colidindo em toda sua área, para evitar que alguma parte do seu desenho possa trancar no cenário. Então nos atributos da máscara de colisão (seção no canto inferior esquerdo do sprite) selecione Mode - Full Image; Type – Rectangle (Fast).

Dica: sempre que houver problemas de colisões com os sprites, este é o local para ajustar o tipo de colisão desejada e resolver a situação.

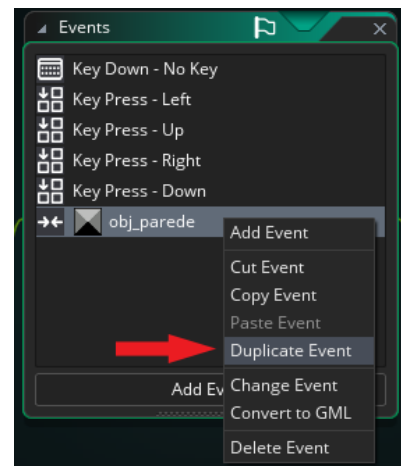


Crie algumas salas com labirintos e teste. Não esqueça de adicionar o objeto bandeira em cada sala, que trocará de fase quando o jogador encostar nele.

A primeira versão do jogo está pronta. Lembre-se de salvar o projeto.

Passo 4 – diamantes para abrir a saída:

O jogador deverá pegar alguns diamantes no cenário antes de poder sair. Podemos bloquear a saída com uma porta, que irá desaparecer quando o jogador pegar todos os diamantes.

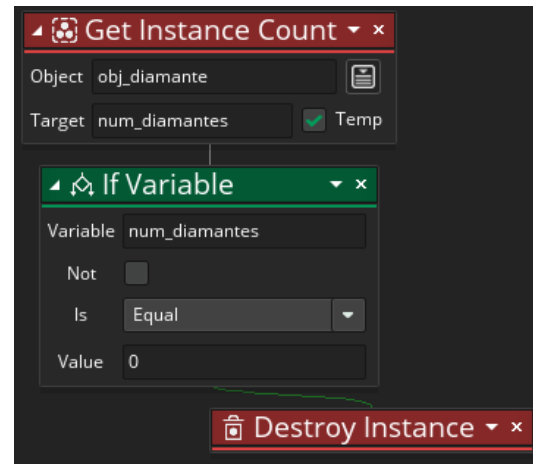
- Crie os diamantes (sprite - diamond.gif, removendo cor do fundo). O objeto deve desaparecer quando o jogador o pegar, então crie o evento de colisão (com o jogador) e coloque a ação Destroy Instance (não esqueça de sempre nomear sprite e objeto para organizar o projeto).
 - Crie a porta (sprite - door2.gif). Este objeto deve ser colocado na frente da bandeira, impedindo a saída da fase. O jogador não pode passar por ele, então devemos deixar a opção Solid ligada no objeto e **configurar que a colisão do jogador com esta porta deve impedir o movimento** (assim como foi feito com a parede). A forma mais prática de fazer isso é, no objeto do jogador, com clique da direita sobre o evento de colisão da parede, escolha duplicar evento. Após selecione colisão com a porta.
- 
- Devemos fazer a porta sumir quando todos os diamantes foram pegos. Isto precisa ser testado a cada instante, se aconteceu ou não. Neste caso usaremos o evento Step.

Explicando: Eventos do tipo *step* acontecem a cada passo (ciclo) do jogo (por padrão 30 passos = 1 segundo). Utilize-os para colocar ações que precisam ser executadas constantemente. Entretanto, é preciso ter cuidado com este tipo de evento. Principalmente, não se deve colocar muitas ações complexas para este evento em objetos com muitas instâncias (cópias). Se fizer isto, pode deixar o jogo lento.

Há 3 tipos de *step*: (geralmente, apenas será preciso utilizar o que é considerado padrão, o primeiro da lista) **step**, **begin step** e **end step**.

O *begin* é executado no começo de cada passo, antes que qualquer outro evento aconteça. O **step** (padrão) é executado antes que as instâncias sejam colocadas em suas novas posições. O *end step* é executado no fim do passo, antes do redesenho da tela.

- Então dentro do objeto porta, crie o evento Step. Coloque a ação Get Instance Count (conta quantos cópias de um objeto existem na sala), onde diz Target crie um nome de variável que irá armazenar o valor – ligue Temp pois pode ser uma variável temporária que será eliminada depois que o evento finalizar.
- Queremos saber se o número de diamantes é igual a zero, então adicione a ação If Variable, indicando a mesma variável que foi usada antes (por padrão já é testado se o valor é igual a zero).
- Por fim adicione a ação Destroy Instance (para destruir a porta), encaixe no lado direito da anterior para ela ser executada somente se o teste for positivo.

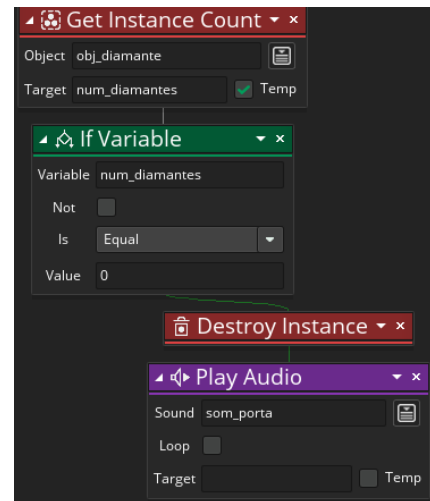


Adicione diamantes no jogo e a porta bloqueando a saída. Teste, quando pegar todos os diamantes da sala, a porta deve desaparecer.

Podemos tocar um som quando a porta sumir.

- Crie um elemento sound, carregue door.wav (coloque nome som_porta). Dentro do objeto porta, dentro do evento step já existente mande tocar o som com a ação Play Audio. Porém ele só deve tocar no momento que a porta sumir, então encaixe abaixo do Destroy Instance.

Agora a porta desaparece e um efeito sonoro toca ao mesmo tempo (somente quando a quantidade de diamantes for igual a zero).



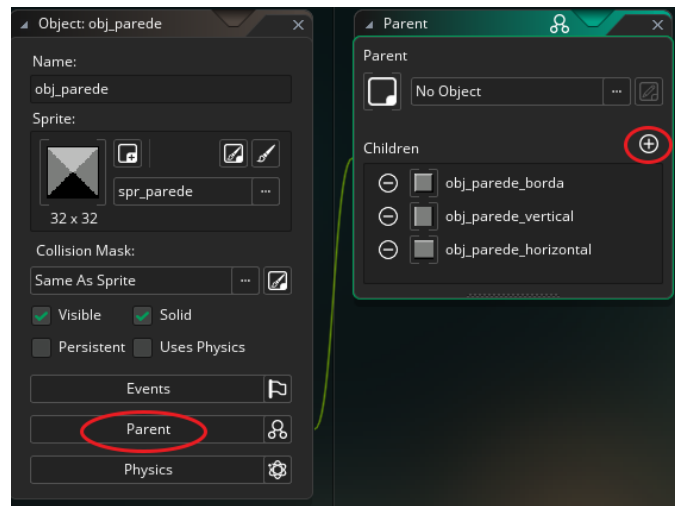
Passo 5 – visual mais interessante e parentesco (opcional):

Podemos deixar o jogo mais interessante visualmente, usando novos gráficos para as paredes; criando objeto para parede vertical, outro para horizontal, outro para as bordas, e então alterando os objetos na sala.

Podem ser usados os sprites: wall_vertical.gif, wall_horizontal.gif, wall_corner.gif (não esqueça que estes objetos devem ter a opção Solid ligada).

Novidade (parentesco):

Para não ter que programar a colisão com todos estes objetos, podemos usar o recurso de parentesco: dizer que os novos objetos são filhos do objeto parede original. (Para ativar, no objeto parede, o primeiro que foi feito, clique em Parent, e adicione os filhos com o botão +). Assim toda a programação do objeto pai é passada para os filhos (se tiver eventos iguais em ambos, a programação local, ou seja do filho, é mantida).



Opcional: podemos até dizer que a porta também é filha da parede, assim o evento de colisão com ela dentro do objeto do jogador pode ser retirado.

Passo 6 – pontuação e tela de abertura:

Vamos dar 5 pontos ao jogador a cada diamante capturado (ou seja, destruído). Quando acabar uma fase o jogador ganha mais 40 pontos.

Como queremos uma pontuação global, ou seja, qualquer objeto que gerar ponto irá afetar o total, precisamos criar uma variável global (todos os objetos podem acessar) para guardar o valor. Desta forma **não** pode ser usado a ação Set Score, pois ela é uma variável local do objeto, só iria somar consigo mesma e não levar em consideração outras cópias do objeto.

- Crie o evento Destroy no objeto diamante, coloque a ação Set Global Variable, em Name coloque: pontos, em Value coloque: 5 e “relative” ligado (relative ligado = somar com o valor já existente).
- No evento de colisão da bandeira com o jogador, use a mesma ação, com o mesmo nome de variável, com valor 40 (esta ação deve ser a primeira a ser executada!).

Nota: uma ação pode ser copiada e colada (e depois alterada) entre eventos ou objetos diferentes, bastando usar clique da direita sobre o título dela e escolhendo o comando equivalente (incluindo opção de abrir ajuda que explica sua função).

Podemos fazer um som tocar quando passar de cada fase, além de dar um aviso de parabéns ao jogador quando acabar o jogo.

- Crie um som (goal.wav) e mande ele tocar quando a bandeira colidir com o jogador (logo depois de aumentar a pontuação), usando Play Audio.
- Para o aviso de parabéns, podemos criar uma sala de fim de jogo, com esta mensagem, e quando o jogador clicar ou apertar algum botão, reinicia o jogo. Então crie um sprite (fim_de_jogo.png) e um objeto (obj_fim_jogo) que usa este sprite.

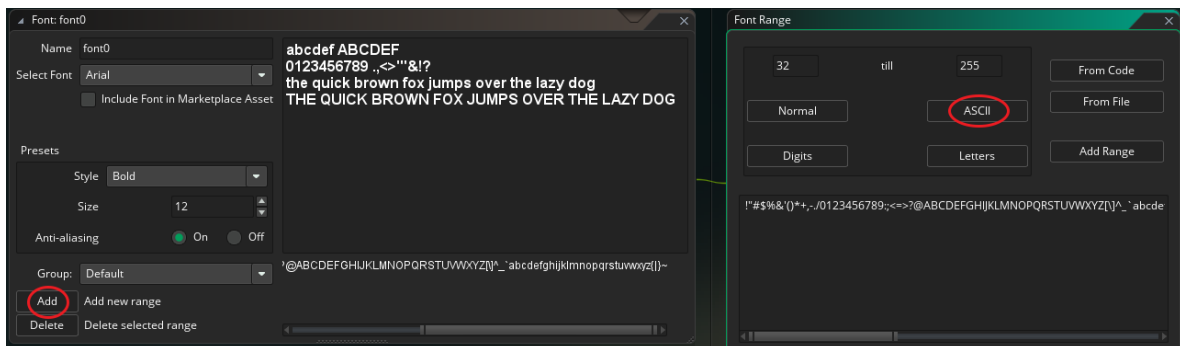
- Neste objeto crie o evento Key Pressed – Any (executado quando qualquer tecla é pressionada) e adicione a ação Restart Game. Crie também o evento Mouse - Global - Global Left Pressed (executado quando clica em qualquer lugar da tela) e inclua a ação Restart Game.
- Crie uma nova sala, troque o nome para deixar claro que é o final do jogo, e arraste para ser a última da lista (a ordem das salas na lista de rooms é a ordem que o GameMaker usa), adicione este objeto (pode colocar alguma imagem de fundo se quiser).

Note que a pontuação não está sendo exibida. Precisamos corrigir isso.

- Crie um objeto sem sprite: obj_gerenciar, que servirá como um gerenciador. Este objeto deve ser inserido em todas as salas de jogo (onde há o personagem e o labirinto, excluindo tela de abertura ou fim), porém não terá nenhum gráfico. No evento Draw deste objeto (utilize a primeira opção Draw - que é o momento quando irá desenhar algo na tela), utilize a ação Draw Value, em Caption: “Pontos:”, em Value: global.pontos (que é a variável global que usamos para armazenar os pontos), em X e Y coloque a posição que a pontuação será escrita na tela.

(Nota: quando este objeto for inserido na sala, será representado como um círculo com uma interrogação dentro, mas isto não será mostrado quando o jogo rodar. Toda variável que é global precisa ser escrita “global. seu nome”, a não ser que esteja dentro de uma ação Set Global Variable ou equivalente)

- Para definir a fonte e cor deste texto, primeiramente crie um elemento fonte no menu principal (Obs: podem ser definidos quais caracteres da fonte serão usados, no botão Add, após ASCII, seleciona todos as letras da fonte, incluindo acentos).



- Agora no evento Draw do objeto gerenciador, coloque a ação Set Draw Colour (escolha a cor que quiser clicando no quadrado branco) e a Set Font, selecionando a fonte que foi criada. (preferencialmente devem ficar no topo da lista de ações)

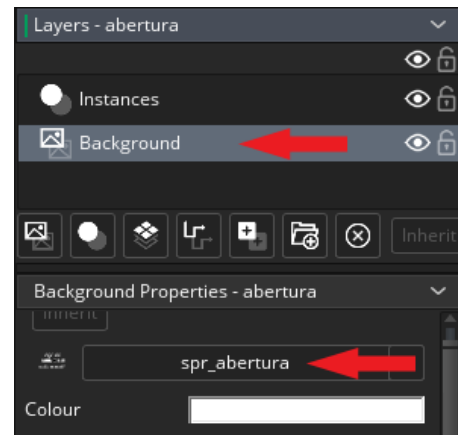
Se você testar o jogo agora, dará um erro! Pois como a ação Draw é executada constantemente, no início o jogo tentará desenhar na tela o valor de uma variável que ainda não tem nenhum valor.

Então precisamos criar na memória a variável que guarda os pontos com o valor 0, antes de poder desenhar na tela. Porém não podemos usar o evento Create do gerenciador para isso, ou a cada fase a pontuação seria zerada. O melhor é criar uma tela de abertura, e ali ter um objeto para inicializar todas as variáveis globais que o jogo irá usar.

Tela de abertura:

Para criar uma tela de abertura, precisamos de uma imagem.

- Crie um sprite e carregue abertura.png (ou crie outro gráfico), nomeie: spr_abertura
- Crie uma nova sala e use esta imagem como fundo, clicando na camada Background e selecionando o sprite abaixo. Arraste esta sala para o topo da lista de salas.



Então devemos criar outro objeto (sem sprite) que espera até que o jogador pressione alguma tecla para carregar a própria sala, além de inicializar as variáveis globais.

- Crie um objeto, nome: obj_inicio. Adicione evento Key Pressed – Any, coloque a ação Go To Next Room (se quiser pode fazer a mesma coisa com evento Global Mouse).
- Adicione evento Create, coloque a ação Set Global Variable, com o nome da variável: pontos, com o valor padrão 0 (sem relative!).
- Adicione este objeto na sala de abertura (somente na sala de abertura!).

Teste o jogo.

Passo 7 – monstros e outros elementos:

Para deixar o jogo mais interessante podemos colocar outros elementos, como alguns monstros que irão andar pelo cenário e tirar uma vida do jogador se colidirem com ele.

O primeiro monstro irá caminhar horizontalmente. Quando colidir com alguma parede irá reverter sua direção.

- Crie um novo sprite (monster1.gif – remova cor de fundo) e um objeto que o use.
- Adicione o evento de criação (Create), use a ação Set Direction Random e marque as setas esquerda e direita (assim ele irá sortear uma destas posições).
- Adicione a ação Set Speed, com valor maior que do jogador, como 6.
- Adicione evento de colisão com a parede principal, coloque a ação Reverse.

O segundo monstro irá caminhar verticalmente. Da mesma forma, quando colidir com alguma parede irá reverter sua direção.

- Crie um novo sprite (monster2.gif – remova cor de fundo) e um objeto que o use.
 - Adicione evento de criação, use a ação Set Direction Random e marque as setas cima e baixo (assim ele irá sortear uma destas posições).
 - Adicione a ação Set Speed, com valor igual ao do monstro anterior.
 - Adicione evento de colisão com a parede principal, coloque a ação Reverse.
-

Vidas:

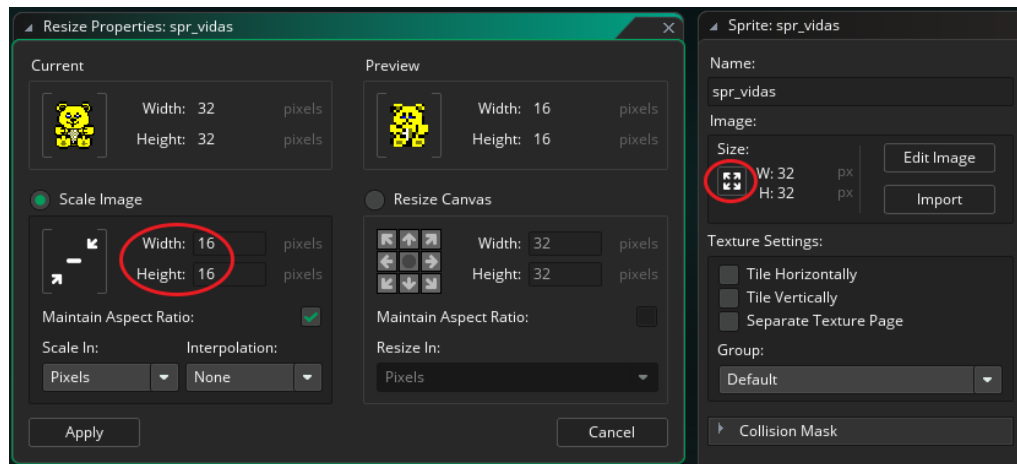
Podemos fazer o jogador perder uma vida quando colidir com algum monstro, além de exibir na tela a quantidade de vidas restantes.

Para não ser necessário colocar a programação de colisão com cada monstro, pois ambas farão a mesma coisa, podemos fazer o monstro2 ser filho do monstro1, e então programar somente a colisão com o monstro1.

- No objeto monstro2 configure o Parent como monstro1 (ou no monstro1 configure o filho como monstro2).
- Crie um novo som para tocar quando o jogador morrer (dead.wav).
- No objeto jogador, crie evento de colisão com o monstro1. Coloque as ações:
 - Tocar o som da morte – Play Audio;
 - Para diminuir a vida, precisamos usar uma viável global que irá armazenar a quantidade de vidas, então adicione Set Global Variable, com o nome da variável: vidas, valor: -1 e relative ligado;
 - Para voltar na posição de início na fase, adicione ação Jump to Start.

Podemos então exibir na tela a quantidade de vidas restantes, usando pequenas imagens que representam cada vida.

- No objeto gerenciador já existente (que foi usado antes para exibir pontuação), adicione as seguintes ações (dentro do evento Draw, depois das existentes):
 - Draw Value, [isto escreve alguma coisa na tela], configure o texto como “Vidas:” e a posição desejada em X,Y (onde diz Value, deixe em branco).
 - Draw Stacked Sprites [isto representa algum valor com imagens na tela], em Number: global.vidas, coloque a posição da tela em X, Y, e a imagem desejada => Para a imagem das vidas, crie um novo sprite, carregue a mesma imagem do urso (person.gif – remova fundo), clique no botão para editar tamanho, deixando com 16 pixels de altura e largura. Assim teremos a mesma imagem do urso, porém com metade do tamanho (veja abaixo).



- O último passo é configurar que o jogador começa com 3 vidas. Para isso, dentro do obj_inicio (aquele usado para colocar a pontuação em 0 e avançar da tela de abertura), dentro do evento “Create” já existente, coloque a ação Set Global Variable, com o nome da variável: vidas, com valor 3 (sem relative).

Mas quando acabar todas as vidas, o que acontece?

Precisamos ficar monitorando a variável que armazena as vidas, e quando ela chegar a zero, o jogo deve recomeçar.

- Dentro do objeto gerenciador (o mesmo que desenha na tela pontuação e vidas) crie o evento Step (que roda constantemente), coloque a ação:
 - If Variable, onde diz variable: global.vidas (o teste padrão já é = 0)
 - Depois encaixe no lado direito da ação anterior (para rodar somente se for positivo), a ação Restart Game.

Extra: acrescente novos efeitos sonoros, música, imagem de fundo. Crie fases interessantes, monte o jogo!