

GameMaker Studio


Exercício 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.

Recomendação: trocar cor do GM Studio para branco, deixando ícones coloridos, mais fáceis de identificar. Para isso vá no menu File – Preferences – na parte de baixo na direita da janela onde diz: Select Skin, marque GM8, então OK; por fim reinicie o GameMaker.

Definindo os objetos iniciais:

- Carregue os sprites do jogador (person.gif), bandeira (goal.gif, que é o ponto de chegada) e parede (wall.gif, este com opção “remove background” desligada; os anteriores com a opção ligada).
- Crie objeto parede (deve ter marcado Solid). Ele não tem nenhum evento ou ação. (obs: coloque os sprites respectivos em cada objeto que for criado)
- Crie objeto jogador. Para fazer ele andar com as setas do teclado, utilize evento keyboard, com a ação Move Fixed  (selecione direção e speed 4). Faça isso para as 4 teclas (cima, baixo, esq, dir / up, down, left, right)
- Crie uma sala de jogo (snap 32x32, que é o tamanho padrão do grid) e adicione o personagem do jogador (urso). Teste o jogo.

Bem, o urso está andando corretamente, mas ele não pára nunca... Ele deve parar quando o jogador soltar a seta do teclado.

- Para isso, no urso (objeto do jogador) crie um evento keyboard – no key, com a ação Move Fixed e marque o quadrado central (manda ele parar). Teste novamente o jogo. (obs: o keyboard – no key é constantemente acionado quando nenhuma tecla está pressionada)
- Crie então o objeto bandeira, que marca o local onde o jogador deve ir. 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 da bandeira crie o evento Collision – com objeto jogador, adicione a ação Check Next  (aba main1), depois a ação Next Room .
 - Se não existir a próxima fase iremos reiniciar o jogo, então coloque a ação Else  (aba control), depois ação Restart Game  (aba main2).



Todas as ações que são testes condicionais (como Check Next que usamos), podem conter o Else para informar o que fazer se a condição for falsa.

- Coloque algumas paredes na sala, formando o labirinto (com shift+ctrl pode adicionar vários objetos arrastando o mouse). Teste.

Falta colisão com as paredes!

- Dentro do urso, crie evento Collision – com a parede, ação Move Fixed e marque o quadrado para tirar o movimento do personagem.

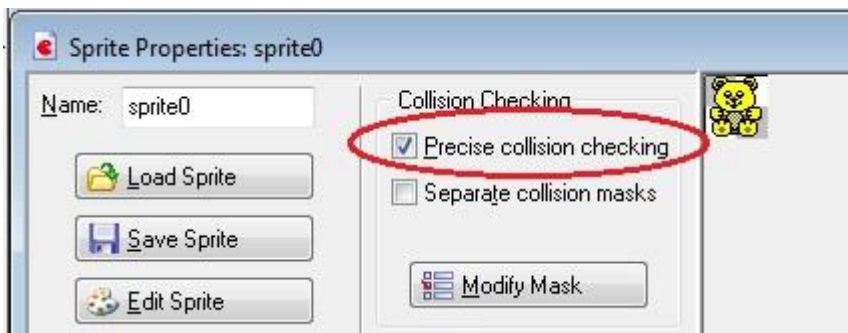
Agora ele colide com as paredes, porém está difícil de fazer “curvas”, de trocar de direção entre paredes estreitas. Isto ocorre pois o urso está com movimentação livre, seria melhor deixá-lo preso em um grid, para que não raspe em “bordas” das paredes, deixando o jogo muito melhor de jogar.

- Para isso, dentro do urso, no evento já existente <No Key>, coloque a ação Check Grid  (aba control) com snap 32x32. Dê Ok e arraste esta ação para ficar acima do “Move Fixed” que já está lá. Então primeiro ele testa se está alinhado no grid e só então faz o personagem parar.
- Da mesma forma, para deixar o controle ainda melhor, coloque a mesma ação – Check Grid  (aba control) com snap 32x32 – como a **primeira ação** em todos os eventos de keyboard <left>, <right>, <up>, <down>. 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).

Sistema de colisão:

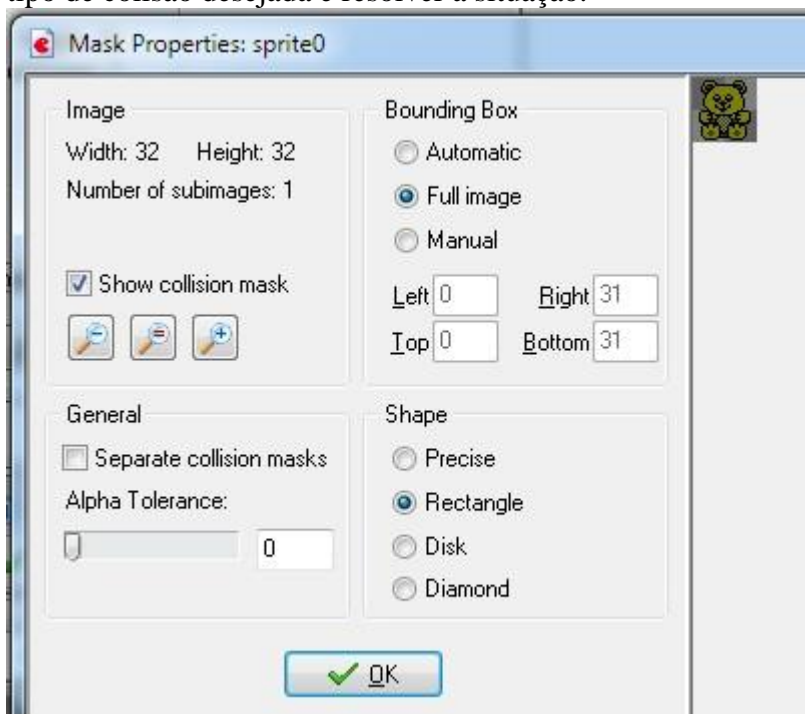
Há várias formas de um sprite colidir com outro: pode ser usado forma precisa (qualquer pixel que não seja transparente contará na colisão) ou outras opções. Se a forma precisa não for necessária, é melhor desligar esta opção para que o sprite colida usando um simples retângulo (conhecido como “bounding 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.

Ex: Com esta opção (abaixo) marcada nas propriedades do sprite, ele usará colisão precisa, ou seja, todo pixel não transparente contará na colisão. Se não usar esta opção pode-se clicar no botão “Modify Mask” para editar o “bounding box” (caixa de colisão) desejado.



Neste exercício o melhor é deixar o urso colidindo em toda sua área, para evitar que alguma parte do seu desenho possa trancar nas “quinas” do cenário. Então nos atributos que abrem ao clicar em “Modify Mask”, marque Bounding Box – Full image; Shape – Rectangle.


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 de salvar o projeto.

Mas queremos fazer um jogo em que o jogador deve 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  (aba main1).
- Crie a porta. (sprite - door2.gif, não deve ser transparente). 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 urso com esta porta deve impedir o movimento do urso** (assim como foi feito com as paredes).
- 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 são 1 segundo). Utilize-os para colocar ações que precisam ser executadas continuamente. 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 uma ação de teste: Test Instance Count  (aba control). Queremos saber se o número de diamantes é igual a zero (então no atributo object indique o objeto que é o diamante, deixe o resto como está, pois por padrão o teste é “igual a zero”). Quando o teste for verdadeiro iremos destruir a porta, então coloque após a ação Destroy Instance  (aba main1).

Teste o jogo, quando pegar todos os diamantes da sala, a porta deve desaparecer.

Podemos tocar um som quando a porta sumir.

- Crie som - door.wav. Dentro do objeto porta, dentro do evento step já existente mande tocar o som. Porém ele só deve tocar no momento que a porta sumir (explicação abaixo).

Atenção: como estamos dentro de um teste condicional, e queremos executar várias ações se este teste for verdadeiro, devemos usar os marcadores de bloco de texto, caso contrário o teste só executa o primeiro comando logo abaixo dele.

- Então devemos incluir o Start Block (aba control), deixar a ação já existente Destroy Instance logo abaixo, depois incluir a ação para tocar o som Play Sound (aba main1) e por fim fechar com o End Block (aba control). Desta forma:



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

Podemos deixar o jogo mais interessante visualmente, usando novos gráficos para as paredes; um para parede vertical, outro para horizontal, outro para as bordas (ou quinas). 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, na janela de propriedades das novas paredes, na opção Parent: escolha o objeto pai – a parede original.) 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 urso pode ser retirado.

Pontuação:

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

- Crie o evento Destroy no objeto diamante, coloque a ação Set Score  (aba score) com valor 5 e “relative” ligado. (relative ligado = somar com o valor já existente)
- No evento de colisão da bandeira com o urso, aumente a pontuação em 40 com a ação equivalente. (esta ação deve ser a primeira a ser executada!)

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 urso (logo depois de aumentar a pontuação), usando Play Sound  (aba main1).
- Para mostrar a mensagem de “Parabéns” quando acabar o jogo, use a ação Display Message  (aba main2) para isso. Coloque esta ação antes da ação que recomeça o jogo “Restart the game”.

Importante: Como agora há mais que uma ação abaixo do “Else” (referente ao teste condicional) o Start Block e End Block devem ser usados.

Resumindo: o evento de colisão da bandeira com o urso deve ter estas ações:



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



- Crie um objeto sem sprite, que servirá como um gerenciador. Este objeto será inserido na sala, 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 Score  (aba score), determinando a posição que a pontuação será escrita na tela.
(Quando este objeto for inserido na sala, será representado como uma bola azul com uma interrogação dentro, mas isto não será mostrado quando o jogo rodar)
 - Para definir a fonte e cor deste texto, primeiramente crie uma fonte no menu principal na esquerda. (obs: podem ser definidos quais caracteres da fonte serão usados, no botão [+])
 - Depois no evento Draw do objeto gerenciador coloque a ação Set Color  e a Set Font  (ambas na aba draw). (preferencialmente devem ficar no topo da lista de ações)
-

Tela de abertura:

Para criar uma tela de abertura, primeiro devemos fazer uma imagem.

- Então carregue esta imagem como background (no menu geral na esquerda).
- Crie uma nova sala e use esta imagem como fundo (na aba backgrounds). Esta sala não faz nada, é só a tela inicial. Arraste esta sala para o topo da lista de salas.



Então devemos criar outro objeto gerenciador (sem sprite) que espera até que o jogador pressione alguma tecla para carregar a própria sala.

- Crie um objeto, chame de gerencia_inicio ou algo semelhante. No evento Keyboard – any key, coloque a ação Next Room  (aba main1). Adicione este objeto na sala de abertura (somente na sala de abertura!).
 - Podemos aproveitar este objeto e fazer com que ele defina o score em 0 (pois o jogo reinicia depois do fim). Então adicione o evento Create (que roda uma vez quando o objeto nasce no jogo) e a ação Set Score  (sem relative!) com o valor padrão 0.
-


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) e um objeto que o use. Adicione o evento de criação (Create), use a ação Move Fixed  (aba move) e marque as setas esquerda e direita (assim ele irá sortear uma destas posições). Coloque um valor de “speed” maior que do jogador, como 6.
Adicione evento de colisão com a parede principal, coloque a ação Reverse Horizontal  (aba move).




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

- Crie um novo sprite (monster2.gif) e um objeto que o use. Adicione evento de criação, use a ação Move Fixed e marque as setas cima e baixo (assim ele irá sortear uma destas posições). Coloque speed igual ao do monstro anterior.
Adicione evento de colisão com a parede principal, coloque a ação Reverse Vertical  (aba move).




Vidas:

O GameMaker conta com um sistema interno de contagem de 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 (urso colide com monstro1 e urso colide com monstro2) 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.
- Crie um novo som para tocar quando o jogador (urso) morrer. (dead.wav)
- No objeto jogador, crie evento de colisão com o monstro1. Coloque as ações:
 - Tocar o som da morte  (aba main1);
 - Diminuir a vida => Set Lives  (aba score), valor -1 e relative ligado;
 - Volta para posição de início na fase => Jump to Start  (aba move)

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, **não é o gerencia_inicio!**), adicione as seguintes ações (dentro do evento Draw):
 - Draw Text  (aba draw), [isto escreve alguma coisa na tela], configure o texto como “Vidas:” e a posição desejada em X,Y.
 - Draw Life Images  (aba score), [isto representa as vidas com imagens na tela], coloque a posição e a imagem desejada => Para a imagem das vidas, crie um novo sprite, carregue a mesma imagem do urso (person.gif) porém clique no botão Edit Sprite, e no editor use o menu Transform – Stretch, 50% de altura e largura. Assim teremos a mesma imagem do urso, porém com metade do tamanho.
- O último passo é configurar que o jogador começa com 3 vidas. Para isso, dentro do gerenciador inicial (aquele usado para colocar a pontuação em 0 e avançar da tela de abertura, chamado gerencia_inicio), dentro do evento “Create” já existente, coloque a ação Set Lives  (aba score) com valor 3 (**sem relative**).

Mas quando acabar todas as vidas, o que acontece? Existe um evento “No More Lives” para tratar isso.

- Dentro do objeto gerenciador (o mesmo que desenha na tela pontuação e vidas, **não é o gerencia_inicio!**), crie o evento Other – No More Lives; coloque a ação:
 - Restart Game  (aba main2) [recomeça o jogo]

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