Jogos Digitais - UNISINOS
Maurício B. Gehling – mbg3dmind@gmail.com

– EXERCÍCIO 4 –
GameMaker Studio 2

Objetivo:
Um jogo de plataforma, com movimentação de tela.

Passo 1 – primeira versão simplificada:

Iremos criar um personagem e alguns blocos que servirão como chão e paredes.

- Para o bloco crie um objeto usando o sprite (block.png). O objeto será sólido (marque "solid").
- Crie o personagem, que por enquanto será uma bola (ball.gif) [remova o fundo]. Vamos fazer sua movimentação horizontal primeiro:
    - Adicione o evento **Key Down – Left**, vamos testar se ele pode ir para esquerda, ou seja, se a posição ao seu lado esquerdo está livre de colisão com o bloco (que representa chão, parede).
    - Para isso coloque a ação: **If Object At**; object: selecione o objeto do bloco; x: -4 ; y: 0 (relative ligado – ambos). Como esta ação testa se há o objeto definido neste ponto, mas queremos saber se não há o objeto, marque a opção "**Not**" para reverter o teste.
    - Se estiver livre executará a próxima ação, que é mover para esquerda. Coloque então **Jump to Point**; x: -4 ; y: 0 (relative ligado – ambos). Encaixe esta ação na direita do teste anterior para rodar só em caso positivo.

    - Adicione o evento **Key Down – Right**, faça a mesma coisa, porém testando o lado direito, ou seja, x: 4 ; y: 0 (relative ligado). Obs: pode duplicar o evento da esquerda e ajustar.

Agora iremos fazer a movimentação vertical (pulo). Para isso podemos usar gravidade. Ela irá acelerar o objeto para baixo, até chegar em uma velocidade máxima (pois o personagem pode atravessar o chão se cair rápido demais).
Então no evento Step iremos testar se a posição abaixo está livre de colisão com o bloco. Se estiver, o personagem está no ar e iremos colocar um valor positivo para a gravidade. Caso contrário o valor será 0.

- Adicione evento **Step**, use **If Object At**; object: selecione objeto do bloco; x: 0 ; y: 1 (relative ligado – ambos). Marque a opção "**Not**".
- Adicione **Set Gravity Force**; force: 0.5 (não use vírgula). Arraste na direita do teste anterior para rodar só em caso positivo.
- Adicione **Else**; após**: Set Gravity Force**; force: 0. Arraste na direita do Else.

- Para o personagem não cair muito rápido, iremos testar se sua velocidade vertical (representada pela variável vspeed) é maior que 12. Se for, a velocidade vertical deve permanecer fixa em 12.
    - Use **If Variable**; (arraste **embaixo** do else anterior para rodar sempre)
        - em variable coloque **vspeed**
        - Is: Greater
        - Value: 12
    - Então adicione **Set Speed**; type: vertical; Speed: 12. Arraste na direita do teste anterior para rodar só em caso positivo.

(nota: na imagem, todo evento step; as ações foram minimizadas com a seta no início do nome)

- Para o personagem pular: Adicione evento **Key Down – Up**; use ação: **Set Speed**; type: vertical; Speed: -10 (**sem relative**)
    - Porém ele só deve pular se estiver no chão! Então temos que testar se abaixo do personagem há uma colisão com o bloco, para permitir o pulo. Adicione **If Object At**; object: selecione objeto do bloco; x: 0 ; y: 1 para testar logo abaixo do personagem (relative ligado - ambos).
        - **Coloque esta ação acima do Set Speed anterior; e deixe o Set Speed encaixado na direita do teste** (para rodar só em caso verdadeiro).

Por fim quando colidir com o chão o personagem deve parar de cair. Então:

- Adicione evento de **colisão com o bloco**, use **Set Speed**; type:vertical; speed: 0 (sem relative)

Crie uma sala com chão e plataformas (usando os blocos) e o personagem (bola).
Obs: para os blocos encaixarem corretamente na sala, troque o valor de Snap (no topo direito da janela da sala) para 16 em X e Y.

Teste o jogo.

Problema: A bola dá uma pequena "freada" estranha às vezes, antes de colidir com o chão em uma queda. Isto acontece porque o evento de colisão retorna o objeto a sua posição anterior quando detecta uma colisão (para evitar que um objeto fique dentro do outro).
No nosso caso queremos que a bola pare somente no ponto exato da colisão. Para isso podemos usar a função **move_to_contact_solid** (só disponível por código).

Com esta função quando acontece uma colisão, o objeto não é movido para a posição anterior, ao invés disso é movimentado na direção e distância máxima determinada pela função, até encontrar o ponto de colisão. Ela só testa com objetos sólidos (como o bloco).

- No evento de **colisão com o bloco**, coloque como **primeira ação** - **Execute code**; digite: **move_contact_solid(direction,12)**
  - Explicando: direction é a variável que guarda a direção atual do objeto (todo objeto possui esta variável automaticamente), 12 é a valor máximo que a função irá mover o objeto até a colisão.

A primeira versão simples do jogo está pronta.

Passo 2 – melhorando gráficos:

Agora vamos melhorar os gráficos. Para o personagem principal iremos usar o sprite de uma menina. Ou melhor, iremos usar dois sprites, um quando estiver olhando para esquerda e outro para direita.

- Crie dois sprites (character_left.gif; character_right.gif) [remova o fundo].

Importante: para evitar problemas de colisão, a área usada para colisão – o bounding box – deve ser exatamente igual nos dois sprites. A forma mais fácil de fazer isso é usando uma **máscara de colisão** – um sprite que será escolhido como referência para colisão.
Desta forma, mesmo que um objeto altere seu sprite (durante animação, por exemplo), toda a colisão será feita de acordo com a área de colisão deste único sprite que foi usado como máscara. Este recurso é um atributo do objeto.

Então no objeto do personagem, altere o sprite padrão (que era a bola) para o novo sprite (olhando para direita). Também escolha como máscara o sprite olhando para direita, para que somente a colisão dele seja usada.

Então agora no evento de **Key Down – Left** (do objeto personagem), iremos trocar o sprite para aparecer o correto (olhando para esquerda):

- Use: **Set Sprite**. Coloque-o acima das outras ações. Escolha o sprite correto para ser exibido, as outras configurações não precisam ser alteradas.
- Da mesma forma, no evento **Key Down – Right**, use a mesma ação, escolhendo o sprite em que o personagem olha para direita.

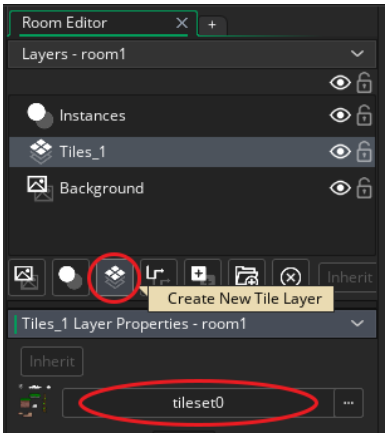
Teste o jogo.

Cenário de fundo:

Para criar um cenário de fundo (background) mais elaborado, podemos usar “tiles”.

Este sistema funciona da seguinte forma: carregamos uma imagem única que contém vários pedaços de desenhos (chão, paredes, plataformas, nuvens), todos alinhados de forma regular (do mesmo tamanho). Observe tiles.gif.
Então usando estes vários pedaços de desenhos, criamos o cenário desenhando na sala.

- Crie um novo sprite (carregue tiles.gif, remova o fundo).
- Crie um novo Tile Set (é um tipo de elemento assim como sprite ou objeto), e carregue o sprite anterior nele. Configure no menu da direita:
  - Tile Properties: tile width e tile height: 16 [largura e altura de cada bloco]
  - Tile Separation X e Tile Separation Y: 1 [quantidade de pixel que separa os blocos]
  - Outras configurações não precisam ser alteradas.



Entre na sala (room0) e crie uma nova camada para tile (Tile Layer), no botão logo abaixo, selecione o tile que foi criado.

Então no menu da direita, onde o tile aparece, selecione alguma parte da imagem clicando sobre ela e pinte no cenário (clique esquerda: adiciona; clique direita: apaga).
Se quiser pintar um grupo maior de tiles, pode definir na parte de cima do menu da direita, onde diz Size.

Por fim crie um novo sprite de nuvens (sky.bmp) e coloque como fundo de todo cenário (configure na camada background da sala – selecione as opções Horizontal Tile e Vertical Tile para repetir o fundo e preencher toda a sala).

Porém estamos só editando o cenário de fundo, o personagem não tem colisão com isso. Então devemos adicionar os blocos no lugar do chão, plataformas, etc. (qualquer lugar que queira colidir) e deixá-los invisíveis (propriedade do objeto, desligue visible).
Assim teremos colisão, mas sem o visual dos blocos cinzas no meio do jogo.

Nota: *para voltar a adicionar objetos, precisa selecionar a camada Instances; se não estiver aparecendo no menu da direita a lista de objetos, clique na aba Resources no topo.*

Se os blocos forem muito grandes para combinar com certas partes do cenário, crie outros blocos menores, deixando-os como filhos do bloco principal (desta forma irá compartilhar todo o código de colisão).

Se quiser que o tile apareça sobre os blocos de colisão, arraste a camada Tiles_1 para cima de Instances. Mas isto só importa durante a edição, no jogo os blocos são invisíveis.
(obs: é possível criar diversas camadas diferentes de tile, background, etc., com diferentes níveis de profundidade – definido pela ordem em que as camadas estão. Ex: se o tile estiver abaixo de instances, poderia desenhar algo atrás do personagem, como uma montanha. Se quiser desenhar algo na frente - como uma árvore ou planta, é só criar uma nova camada tile acima de instances e desenhar nesta camada o que quiser)

Passo 3 – adicionando inimigos:

O jogo está funcionando, mas está muito sem graça, precisa de alguns inimigos para gerar desafio ao jogador. Vamos fazer um monstro que caminha nas plataformas de um lado a outro.
Precisamos de dois sprites, uma para o mostro virado para esquerda e outro para direita.

- Crie dois sprites (monsterl.gif; monsterr.gif) [remova o fundo].
- Crie um objeto usando o sprite virado para direita, adicione o evento **Create**, faça ele caminhar para direita com a ação **Set Speed** (Type: Horizontal; speed: 4)
- Se colidir com uma parede, deve reverter sua direção. Então crie evento de **colisão com o bloco cinza** do chão, use a ação **Reverse** (Direction: Horizontal)

O mostro retorna, porém o sprite continua o mesmo. Devemos trocar pelo sprite correto quando ele vier pela esquerda ou direita. Usaremos o evento End Step, que é executado depois dos cálculos do Step, porém antes do objeto ser desenhado na tela.
Testando a variável “hspeed”, que indica a velocidade horizontal, poderemos saber se o mostro está indo para esquerda ou direita. Se hspeed < 0 está indo para esquerda, caso contrário direita.

- Adicione evento **End Step** (fica no menu do Step) e use:
  - **If Variable** (Variable: hspeed ; ls: less ; Value: 0)
  - **Set Sprite** (escolha o sprite monstro virado para esquerda), encaixe esta ação na direita do teste anterior para rodar só em caso positivo;
  - Adicione **Else**;
  - **Set Sprite** (escolha sprite monstro virado para direita), encaixe esta ação na direita do Else para rodar só em caso negativo.

Importante: para evitar problemas de colisão, o bounding box (área de colisão) deve ser exatamente igual nos dois sprites. A forma mais fácil de fazer isso é escolher o sprite inicial (monstro olhando para direita) como máscara no objeto.

Inclua um monstro na sala e teste o jogo.

Para evitar que os monstros saiam andando no ar quando acabar as plataformas, iremos criar um novo objeto chamado marcador. Ele será um bloco azul invisível, e serve para reverter a direção do monstro quando colidir com ele. Usar marcadores invisíveis é uma boa técnica para fazer objetos realizarem certas ações em partes do cenário.

- Crie um sprite simples de um bloco azul (marker.gif) e um objeto que o use. Este objeto será invisível (desligue visible). Adicione cópias deste objeto nas bordas das plataformas para o mostro colidir (na sala).
- Crie o evento **colisão do monstro com o marcador**, adicionando ação **Reverse** (Direction: Horizontal)

O personagem deve morrer quando colidir com o monstro. Porém se ele pular na sua cabeça, pode "achatar" e matar o monstro (como em jogos do Mario e outros de plataforma).
Então no evento de colisão do personagem com o monstro, devemos testar se estamos atingindo o monstro por cima para fazer ele achatar. Para isso usaremos o seguinte teste:

vspeed > 0 && y < other.y + 8

Explicando passo a passo:

vspeed > 0

Se a velocidade vertical (ou vspeed) for maior que 0, então o personagem está movendo para baixo (caindo).

&&

Significa "**E"** lógico. Ou seja, este teste só será verdadeiro se ambas as expressões forem verdadeiras.

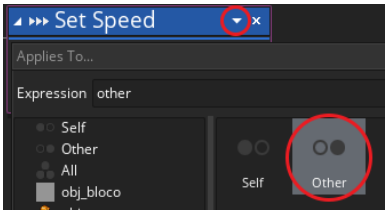
y < other.y + 8

Se a coordenada em y do personagem (equivalente variável y) for menor que a coordenada y do outro objeto envolvido na colisão (neste caso o monstro) + 8, significa que ele colidiu próximo da parte superior.

Então vamos programar isso no evento de **colisão do personagem com o monstro**:

- **If Expression;** em expression digite: **vspeed > 0 && y < other.y + 8**
- Se esta expressão for verdadeira o monstro deve morrer. Podemos fazer o sprite do monstro trocar por um "monstro achatado", quando ele for atingido.
    - Primeiramente ele deve parar de caminhar para morrer parado. Use então a ação **Set Speed**, speed: 0

      (encaixe esta ação na direita do teste anterior para rodar só em caso positivo)

        - Importante: use opção **Other** (na seta antes do X), para que seja executado no outro objeto da colisão e não no próprio jogador.

    - Para trocar para o monstro achatado, crie o sprite (monster_flat.gif) [remova fundo] e um objeto que use este sprite;
    - Então de volta ao personagem, na colisão com o monstro, use a ação **Change Instance**; (escolha o objeto monstro achatado, e a opção **Other**)

- Adicione **Else**; pois agora iremos programar a morte do jogador, quando ele encostar no monstro sem estar pulando na sua cabeça.
    - Adicione **Restart Room**, pois quando o jogador morre deve recomeçar a fase (encaixe esta ação na direita do Else para rodar só em caso negativo)

O que ficou faltando fazer é desaparecer o monstro achatado depois de algum tempo. Então adicione o evento de criação (**Create**) do objeto "monstro achatado", adicione um alarme com a ação **Set Alarm Countdown,** em Countdown coloque 10.
Quando o alarme disparar o objeto deve morrer, então adicione o **evento Alarm 0**, e lá dentro coloque a ação **Destroy Instance**.

Teste o jogo.

Passo 4 – adicionando armadilhas:

Podemos colocar algumas armadilhas no cenário que devem ser puladas (pontas no chão). Então um outro marcador invisível pode ser utilizado no cenário em locais onde o jogador morrerá se pisar.

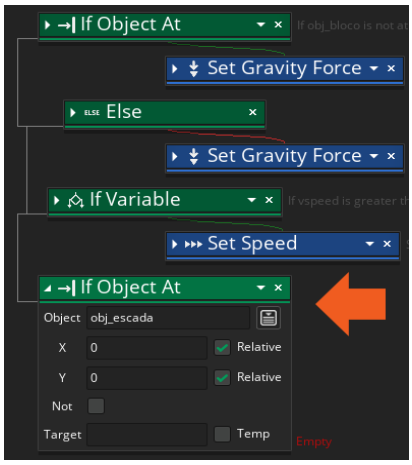
- Dentro da sala, usando os "tiles" do cenário (precisa estar selecionado a camada Tiles), selecione o desenho das pontas e acrescente em alguns lugares sobre o chão.

- Crie um objeto marcador **invisível**, usando sprite (dead.gif). Coloque ele na sala sobre as pontas (precisa estar selecionado a camada Instances; se não estiver aparecendo no menu da direita a lista de objetos, clique na aba Resources no topo)
- No objeto do jogador, adicione o evento de **colisão com o bloco vermelho (morte)**, e para o jogo recomeçar adicione a ação **Restart Room**.

Passo 5 – adicionando escadas:

Podemos colocar uma forma do personagem subir em "escadas" para alcançar pontos mais altos do cenário. Por exemplo, uma planta alta pode ser usada para escalar. Neste caso selecionamos a planta nos "tiles" do cenário de fundo e pintamos na tela. Para programar as escadas, vamos utilizar outro marcador invisível. Se o personagem estiver colidindo com este marcador (que ficará sobre a planta), ele poderá subir por ela. Caso contrário terá o mesmo comportamento que tem atualmente. (botão para cima = pulo)

- Crie sprite (ladder.gif – deve ser transparente no fundo, pois é um retângulo fino) e um objeto que o use (**invisível**). Posicione cópias deste objeto na fase, em locais onde será permitido escalar (sobre planta alta do cenário de fundo).
- Quando ele estiver escalando, não deve cair, ou seja a gravidade deve ser "desativada". Para isso, no evento **Step** do personagem vamos testar se ele está colidindo com o marcador da escada: (estas ações irão depois das já existentes)
    - Adicione **If Object At**, escolha o marcador da escada, x: 0 ; y: 0 e relative ligado em ambos. (testa se há colisão com um objeto específico na posição determinada)
        - Importante: esta ação deve rodar sempre, então não encaixe com a última ação que faz parte de um teste anterior (Set Speed); arraste um pouco para direita; verifique na imagem.
          (obs: ações anteriores foram minimizadas com a seta no início do nome)



- Se isto for verdadeiro, estamos colidindo com a escada, então iremos entrar com várias ações (encaixe na direita do teste para rodar só em caso positivo):
    - **Set Gravity Force**; Force: 0 ("desliga" gravidade para não acelerar o jogador)
    - **Set Speed**; Type: Vertical; Speed: 0 (faz velocidade vertical atual ser nula)
    - Agora podemos trocar o sprite do personagem para ele "virar de costas" e parecer que está "escalando". Crie sprite (character_climbing.gif) [remova fundo]

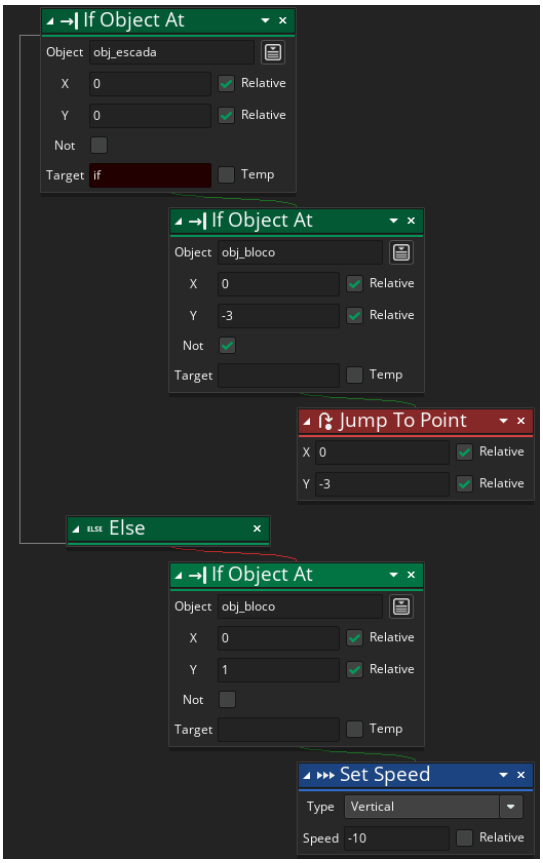
- De volta ao evento **Step** do personagem, adicione (encaixado depois da última ação) **Set Sprite** ; escolha o sprite da escalada;

Agora falta programar a tecla para cima, que era usada como pulo. Se o personagem estiver colidindo com o marcador da escada, conseguirá subir. Caso contrário executa o pulo normalmente.

No evento **Key Down - Up** do personagem, iremos testar se ele está colidindo com o marcador da escada:

- Coloque como **primeira ação**: **If Object At**, escolha o marcador da escada, x: 0 ; y: 0 e relative ligado (ambos). As ações pré-existentes neste evento ficarão abaixo do **Else,** pois serão executadas se o teste for negativo. Então já aproveite e inclua um **Else** logo abaixo deste primeiro teste.
- As próximas ações ficarão logo abaixo deste primeiro teste, acima do Else (arraste na direita para rodar só em caso positivo). São elas:
  - **If Object At**; object: selecione objeto do bloco; x: 0 ; y: -3 (relative ligado – ambos). **Marque a opção "Not"** (testa se não há colisão acima do personagem)
  - **Jump to Point** ; x: 0 ; y: -3 ; relative ligado (isso fará o personagem subir para um ponto um pouco acima). Arraste esta ação na direita do teste anterior.
- Abaixo do Else existe o antigo código que testa se o jogador está colidindo com o chão para permitir pular. Arraste este bloco para direita do Else, para que rode só em caso negativo do teste anterior (ou seja, só vai permitir pular se o jogador não está encostando com o marcador da escada).

Confira na imagem todas as ações na ordem correta, de todo evento Key Down - Up:



Por fim falta a possibilidade do personagem descer enquanto estiver escalando, então crie o evento **Key Down - Down** e adicione:

- **If Object At**, escolha o marcador da escada; x: 0 ; y: 0 (relative ligado - ambos).
- **If Object At**; object: selecione objeto do bloco; x: 0 ; y: 3 (relative ligado – ambos). **Marque a opção "Not"** (testa se não há colisão abaixo do personagem). Arraste na direita do teste anterior para só rodar em caso positivo.
- **Jump to Point** ; x: 0 ; y: 3 ; relative ligado - ambos (isso fará o personagem descer para um ponto um pouco abaixo). Arraste esta ação na direita do teste anterior.

Teste o jogo e veja se o personagem está subindo e descendo a "escada" corretamente.

Passo 6 – próxima fase:

Para seguir para a próxima fase, podemos criar um objeto de saída, ou seja, o objetivo do jogador é chegar até este objeto. Quando ele colidir com a saída, o jogo testará se existe mais uma fase e o levará até lá, caso contrário acaba o jogo.

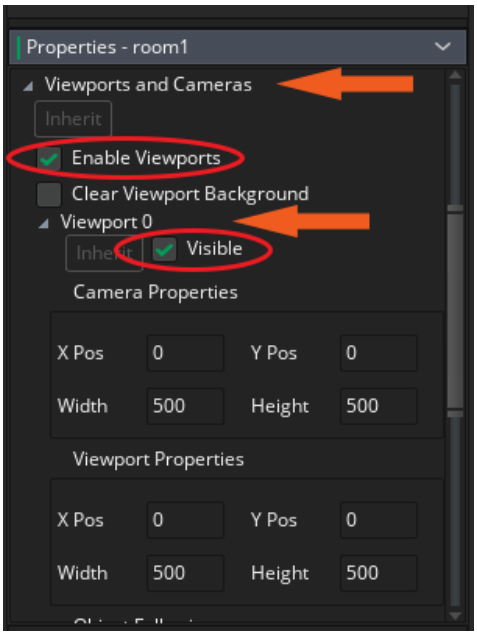
- Crie sprite (level.gif) [remova o fundo] e um objeto que o use (este será o objeto de saída da fase).
- Adicione evento de **colisão do personagem com este objeto**, então iremos testar se a fase atual não é a última (ou seja, existe outra). Adicione **If Room Is Last, marque Not**.
    - Após adicione **Go To Next Room**, para ir para próxima fase. Arraste na direita do teste anterior.
    - Se o teste der negativo iremos encerrar o jogo com uma tela de aviso e reiniciar. Adicione **Else**, e então **Execute Code**, e digite o código: **show_message("Game Over")**. Arraste o Execute Code na direita do Else.
    - Por fim inclua a ação **Restart Game**.

Posicione o objeto de saída no final da fase. Crie outra fase nova. **Teste o jogo.**

Passo 7 – janelas de visão - views (câmera):

A maioria dos jogos de plataforma tem uma fase bem extensa horizontalmente e a câmera vai acompanhando o personagem, mostrando só parte do cenário por vez. Podemos fazer isso utilizando “views” da sala.

Dentro das configurações da sala (room), no canto inferior esquerdo, abra a seção “Viewports and Cameras”, clique em “Enable Viewports” (liga o sistema de views).



Clique na seção “Viewport 0”, clique em Visible (deixa visível esta primeira view, que será a única que usaremos). *Nota: podem ser usadas várias views ao mesmo tempo, como em jogos cooperativos com 2 câmeras dividindo a tela no meio.*

Abaixo em “Camera Properties” configuramos o ponto inicial esquerdo da view dentro da sala (X,Y), assim como sua largura e altura (W,H). Esta será o tamanho da área visível da sala dentro do jogo (ou seja, deve ser menor que o tamanho total da fase, caso contrário a view não terá para onde andar). Pode fazer um teste usando X:0 ; Y:0; Width: 500 ; Height: 500 (depois ajuste o tamanho que achar melhor, de acordo com seu jogo).

Em “Viewport Properties” configuramos o tamanho que esta janela terá na tela quando o jogo rodar. Para não esticar ou encolher o cenário, deve ser o mesmo tamanho de “Camera Properties”.

Em “Object Following”, escolha um objeto para a view seguir (geralmente o jogador). Em “Horizontal Border” e “Vertical Border” temos o tamanho em pixels de “borda” da view; ela só irá se mover quando o objeto chegar nesta borda. Então se quiser que o personagem fique no centro da tela e a view acompanhe a todo momento, utilize a metade da largura da view em “Horizontal Border”, e metade da altura em “Vertical Border” (ou seja, 250 no nosso teste).

Em “Horizontal Speed” e “Vertical Speed”, temos a velocidade máxima de movimento da view por step. Um valor de -1 é movimentação instantânea. Qualquer outro valor determina a quantidade de pixels por step que a view anda. Se usarmos um valor menor que a velocidade do personagem que está sendo seguido, a tela irá andar suavemente e lentamente, porém o personagem pode sair da tela, pois anda mais rápido. O ideal é usar a mesma velocidade que ele (ou deixar no padrão -1).

Podemos refazer toda a sala para criar um cenário bem maior horizontalmente, permitindo que a view acompanhe a ação até chegar no final. Para trocar o tamanho da sala, basta ir em “Room Settings” (na mesma região da configuração da view, porém bem mais acima, role a barra lateral). Aumente o valor de Width para a sala crescer na largura.

Crie fases interessantes e finalize o jogo! Não esqueça que a configuração da view deve ser feita em cada sala (ou duplique a sala atual para ter uma cópia e modifique).