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

– EXERCÍCIO 3 –
GameMaker Studio 2

Objetivo:
Um shooter de nave clássico, vertical.

Variáveis:

O GameMaker possui várias variáveis pré-definidas que podemos utilizar. Podemos também criar qualquer variável que quisermos para algum fim específico, seja local a um objeto (só ele vê) ou global (todos podem ver).

Exemplos de variáveis de objetos (cada cópia de um objeto possui estas variáveis):

x	(coordenada x de um objeto)
y	(coordenada y de um objeto)
score	(pode ser usada para gerenciar a pontuação **local** do objeto)
lives	(pode ser usada para gerenciar a vida **local** do objeto)
health	(pode ser usada para gerenciar a energia **local** do objeto)
hspeed	(velocidade horizontal do objeto [em pixels por step])
vspeed	(velocidade vertical do objeto [em pixels por step])
direction	(direção atual de movimento em graus [0-360; 0 é horizontalmente para a direita)
speed	(a velocidade atual nesta direção)
visible	(se o objeto é visível [1] ou invisível [0])
solid	(se o objeto é sólido [1] ou não sólido [0])

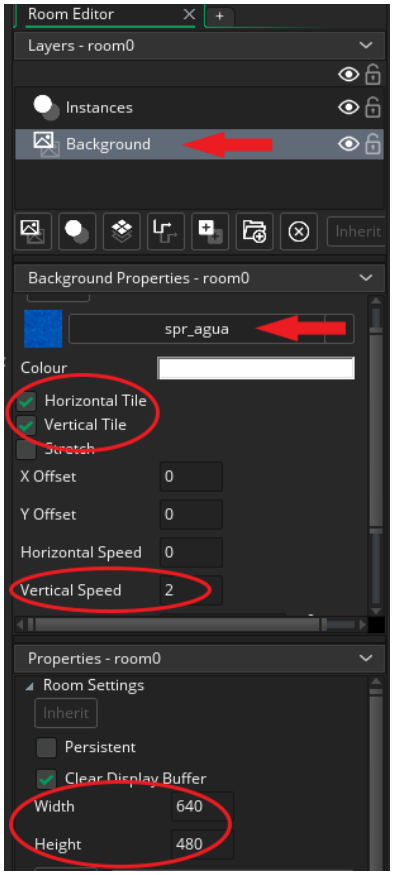
Exemplos de variáveis globais:

mouse_x	(posição x do mouse)
mouse_y	(posição y do mouse)
room_width	(largura da sala em pixels)
room_height	(altura da sala em pixels)

Há muitas outras variáveis, podem ser consultadas na documentação oficial do GameMaker - tecla F1.
Algumas ações podem manipular o valor de certas variáveis, mas também podemos manipular diretamente.

No exemplo do nosso shooter, podemos criar uma variável que irá controlar a quantidade de disparos que a nave pode fazer. (ex: atirar a cada 5 steps do jogo)
Então a nave precisa de uma variável que vai indicar se ela pode atirar ou não.
Podemos criar a variável **pode_atirar** e no evento de criação da nave colocamos o valor 1 nesta variável (indicando verdadeiro). Assim quando o jogador quiser atirar, faremos o teste do valor da variável para ver se é permitido.
Se for permitido, criamos o tiro e colocamos **pode_atirar = 0** (impedindo que atire).
Então usamos um alarme para voltar ao valor 1 depois de 5 steps.
*** Obs: este procedimento será detalhado depois.**

Passo 1 – começando com o jogo:



Ilusão de movimento do cenário:
Uma das formas de fazer o cenário parecer que "caminha" é fazendo com que a imagem de fundo da sala (background) se movimente.

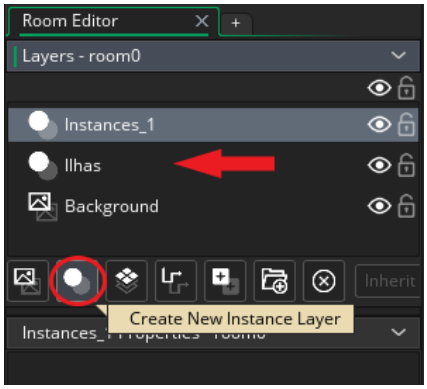
- Crie um sprite (water.gif)
- Na sala coloque este sprite como fundo, clicando na camada background e carregando ele, após clique em Horizontal Tile e Vertical Tile. Role para abaixo a janela de atributos do background se necessário para ver Vertical Speed, coloque 2. Configure a sala para ter o tamanho 640x480 (Room Settings, em Width e Height respectivamente), para ficar compatível com as imagens que usaremos.

Teste o jogo, a imagem de fundo deve estar andando.

Vamos adicionar ilhas no cenário para tornar mais interessante. Estas ilhas devem surgir em posições aleatórias para ficar mais natural a rolagem do cenário (não demonstrar repetição).

- Crie três sprites de ilhas (island1.gif, island2.gif, island3.gif) [removendo fundo].
- Crie três objetos usando estes sprites. No evento de criação (**Create**) colocaremos eles andando na mesma velocidade do background da sala, com a ação **Set Speed**, Type: Vertical, Speed: 2. (obs: depois de criar a ilha1, pode duplicar o objeto para fazer as outras, precisando só trocar o nome e sprite que usa)
- Coloque as ilhas, uma cópia de cada, em algum local da sala.

- Para ter certeza que todos os outros objetos de jogo sejam desenhados acima das ilhas, crie uma nova camada do tipo instance, que ficará sobre a anterior. De agora em diante use esta nova camada para adicionar novos objetos. (e pode renomear a anterior para deixar claro que ali estão as ilhas, com clique da direita sobre ela).



Teste o jogo.

Agora iremos testar quando as ilhas saírem da tela, e então programar para que elas voltem a aparecer no topo (em uma posição horizontal aleatória).
Para isso iremos usar algumas variáveis (já pré-definidas no GameMaker):

y = posição vertical do objeto (sendo 0 = topo)
room_height = altura da sala

Então iremos testar se o **y** do objeto é maior que **room_height** (ou seja, saiu da sala).
Se for, a ilha deve ir para o topo da sala.

- Para isso adicione o evento **Step** em cada ilha, e use a ação **If Variable**, sendo:
  variable : y
  value: **room_height** (<= copie o nome exato da variável)
  operation: greater

Agora a ação que será executada se este teste for verdadeiro (encaixe na direita):

- **Jump to Point**, para x usaremos uma função que sorteia um valor entre 0 e a largura máxima da sala (que é a variável room_width).
  - Então digite esta função: **random(room_width)**
- Para o atributo y desta ação podemos usar -65, para fazer a ilha aparecer acima da tela (pois é o tamanho da altura do sprite da ilha).

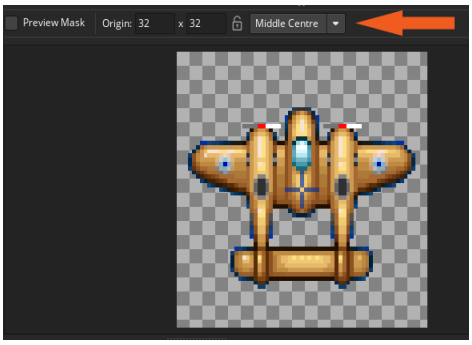
Faça isso para cada ilha (possível copiar e colar ações) e teste o jogo. Se tudo deu certo, toda vez que uma ilha sai da tela, ela volta a aparecer no topo, em uma posição aleatória horizontalmente.

Passo 2 – avião principal:

- Crie um sprite, carregue o avião do jogador (myplane.gif) Este sprite é composto por 3 sub-imagens (hélices girando) que serão animadas ao definir uma velocidade.
  [remova o fundo – selecione todos os frames para remover o fundo de uma vez só].

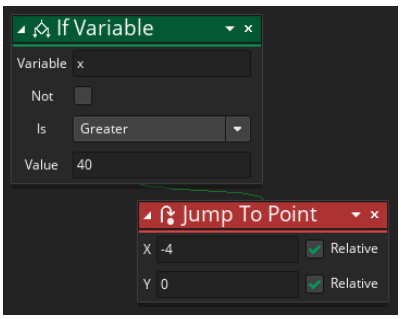
- Nesta mesma janela do sprite, coloque a origem no centro, clicando no botão “middle centre”. (alguns testes usam este ponto como referência, por isso é importante estar no centro do avião; ex: tiros devem sair do centro)

- Crie objeto “obj_jogador”, usando este sprite.



Para movimentação do avião, devemos tomar cuidado para ele não sair da tela. Então vamos testar se a variável “**x**” é maior que um valor próximo da borda (ex: 40) e permitir movimentação somente se isto for verdadeiro (neste caso para esquerda).

- Adicione evento **Key Down – Left**, coloque a ação **If Variable** sendo variável: **x ;** valor: 40 e testar se “**maior que**” **(Greater)**.
- Coloque outra ação encaixado na direita da anterior: **Jump to Point** , x: -4 ; y: 0 ; **relative ligado (ambos)**. Isto fará com que o avião pule ligeiramente para o lado esquerdo, enquanto a tecla direcional esquerda estiver sendo pressionada.



Faça o equivalente para as outras direções, adicione:

- **Key Down – Right**, testando se **x é menor (Less)** que a largura da sala menos uma pequena margem para o avião não bater na borda, então em Value coloque **600**. Depois **Jump to Point** com x: 4 ; y: 0 (relative ligado - ambos).

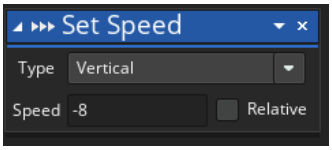
- **Key Down – Up**, testando se **y é maior que 40**. Depois **Jump to Point** com y: -2 ; x: 0 (relative ligado - ambos). Note que diminuindo o valor de y o avião sobe. Usamos valor 2 na vertical para coincidir com a velocidade de rolagem do background.

- **Key Down – Down**, testando se **y é menor** que a altura da sala menos uma margem para o avião não bater na borda, então em Value coloque 360. Usamos este valor para o avião ficar mais longe da base da tela, pois adicionaremos uma barra de interface depois. Insira ação **Jump to Point** com y: 2 ; x: 0 (relative ligado - ambos).

Adicione o objeto do jogador (avião) na fase e teste a movimentação.

Tiro:

- Crie sprite (bullet.gif) [removendo fundo], coloque a origem no centro.
- Crie objeto (obj_tiro_jogador) usando este sprite. Adicione evento de criação (**Create**) e faça o objeto andar para cima com **Set Speed**, Type: Vertical; Speed: -8.



Agora devemos destruir o tiro quando ele sair da tela, para poupar processamento. Há um evento extra que indica quando um objeto saiu completamente da sala:

- Crie o evento **Other – Outside room**, coloque a ação **Destroy Instance**.

Nota: Sempre que um objeto sair da sala e não for mais necessário, ele deve ser destruído para liberar memória, caso contrário o jogo irá ficar cada vez mais lento.

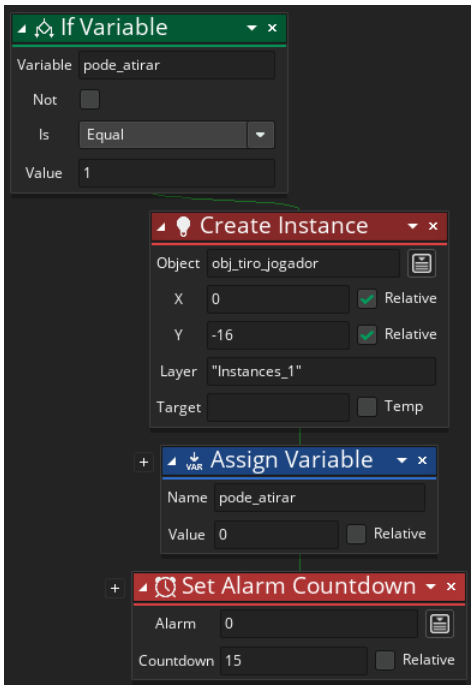
Para o avião atirar com a barra de espaço:

- Dentro do objeto jogador (avião), crie o evento **Key Down – Space**, coloque a ação **Create Instance**. Configure para criar o objeto tiro, na posição x: 0 e y: -16 **(ambos com relative ligado)**. No atributo Layer deve constar o nome exato da camada de objetos definida na room (provavelmente "Instances_1"). Isto irá criar uma cópia do objeto tiro, sobre o centro do avião (posição relativa).

Teste o jogo e veja que estão saindo muitos tiros quando pressiona a barra de espaço.
Para corrigir isso, iremos usar uma variável "**pode_atirar**". Se ela for =1 pode ser criado um tiro, caso contrário não. Assim que o primeiro tiro é criado, pode_atirar ficará com o valor 0, não permitindo que saia mais nenhum tiro. Então criamos um alarme que em meio segundo irá trocar o valor de pode_atirar para 1 novamente.

- Para implementar isso, o primeiro passo é só atirar se pode_atirar for = 1. Então no evento já existente **Key Down - Space** do jogador (avião), **antes** de criar um tiro colocamos a ação **If Variable**. Configuramos para testar se pode_atirar é igual a 1.
- Arraste a ação do tiro (que já existe) para a direita do If Variable, fazendo com que seja executada somente em caso positivo do teste.
- Depois de criar o tiro, iremos trocar o valor de pode_atirar. Utilize a ação **Assign Variable** para deixar pode_atirar = 0 **(sem relative!).**
- Agora iremos criar o alarme. Use **Set Alarm Countdown**, com Countdown: 15 (meio segundo).

Verifique se tudo ficou como na imagem ao lado.



- Então quando o alarme disparar, devemos trocar o valor de pode_atirar para 1 novamente (adicione **evento Alarm 0**, após adicione ação **Assign Variable**, colocando pode_atirar com o valor 1).
- **Não podemos esquecer que no início do jogo, quando o avião é criado, pode_atirar deve ser = 1, para poder criar o primeiro tiro e não dar erro ao testar a variável (que até o momento não existe).** Então adicione evento **Create**, e use **Assign Variable** para fazer isso.

Teste o jogo.

Passo 3 – inimigo:

O primeiro inimigo será simples, só voará para baixo da tela. Se colidir com o avião do jogador, o jogo acaba.

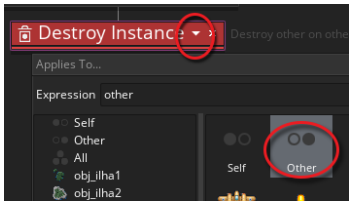
- Crie um sprite (enemy1.gif) [removendo fundo e coloque a origem no centro, defina uma velocidade para animação em Speed] e um objeto que o use. Adicione evento de criação (**Create**) coloque ação **Set Speed**, tipo vertical, speed com valor 4. Temos que fazer o avião voltar a aparecer na tela quando ele sair por baixo (semelhante ao comportamento das ilhas).
- Um meio alternativo de fazer isso é usando o evento **Other – Outside Room**, que acontece assim que o objeto sair completamente da sala. Então adicione este evento e coloque a ação **Jump to Point**; no atributo x: **random(room_width)** ; no atributo y: -16
- Lembrando que para facilitar testes enquanto se programa o jogo, em funções aleatórias, o GameMaker sempre sorteia resultados iguais. Para que o avião realmente fique em posição aleatória, volte ao evento **Create** e adicione a ação **Randomize** (obs: esta ação poderia estar em qualquer objeto).

Adicione o inimigo na sala e teste.

Agora precisamos fazer ele colidir com o tiro e morrer; além de colidir com o avião do jogador e acabar o jogo. Primeiro com o tiro:

- Crie o sprite (explosion1.gif) [removendo fundo, coloque a origem no centro, defina uma velocidade para animação em Speed] e um objeto para ele (pode chamar de obj_explosao_inimigo). Essa animação de explosão será tocada quando o inimigo for atingido.
- Para ter o som da explosão: adicione um elemento sound e carregue o arquivo (snd_explosion1.wav)
- No avião inimigo, crie o evento **colisão com o tiro do jogador**. Então adicione:
    - Para tocar o som, **Play Audio**, selecione o som carregado antes.

- Fazer desaparecer o tiro: **Destroy Instance** – opção **Other** (para destruir a **outra cópia** do objeto envolvido na colisão, no caso o tiro; impedindo que ele siga destruindo outros inimigos que encontrar)



- Mostrar animação de explosão: **Create Instance**, coloque o objeto que é a explosão e ligue relative (ambos). Deixe x,y em 0, pois queremos a explosão exatamente sobre a posição relativa do avião. Em Layer coloque o nome da camada de objetos definida na room (provavelmente "Instances_1").
- O avião inimigo não deve morrer, ele deve surgir novamente no topo da tela, em alguma posição em x; então use: **Jump to Point**; no atributo x: **random(room_width)** ; no atributo y: -16

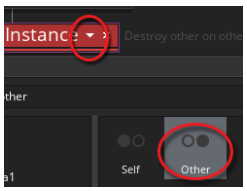
Teste o jogo. O avião inimigo deve explodir quando atingido e surgir novamente no topo da tela. Porém a explosão **não está desaparecendo**!

- No objeto explosão inimigo, use o evento **Other – Animation End** (é executado quando a animação chegar no fim); ali coloque um **Destroy Instance**, para a explosão "morrer".

Agora a colisão do avião inimigo com o avião do jogador:

- Crie um som de explosão diferente (snd_explosion2.wav) e o sprite para explosão (explosion2.gif) [removendo fundo e coloque a origem no centro, defina uma velocidade para animação em Speed]. Esta explosão é maior que a outra. Crie um objeto que a use. (pode chamar de explosao_jogador para diferenciar)
- Neste objeto explosão, no evento **Other – Animation End**, iremos fazer o jogo acabar:
  - Primeiro a explosão deve sumir no final da animação: **Destroy Instance**
  - Por fim reiniciar o jogo: **Restart Game**

- No objeto do avião inimigo, crie um evento de **colisão com o jogador** e use:
  - **Change Instance**, isto irá trocar um objeto por outro, iremos usar para o avião do jogador ser trocado pela explosão (configure para o objeto explosao_jogador em "Object"). **Importante: Marque Other ao invés de Self** (caso contrário quem irá virar explosão é o avião inimigo, e não o jogador como queremos; veja imagem ao lado)



  - **Play Audio** para tocar o som de explosão 2.
  - E o avião inimigo também deve morrer: **Destroy Instance** (opção Self).

Teste o jogo.

Ele está jogável, porém só tem um avião inimigo por vez na tela. O jogo pode ficar mais interessante com mais inimigos ao mesmo tempo. Uma forma eficiente de fazer isso é criando um objeto especial, não visível no jogo, que irá controlar a criação de inimigos.

- Crie um objeto (controlador_inimigo) e desligue a opção "visible". Ele não precisa de sprite.
    - Adicione evento **Create**, crie um inimigo no topo da tela com posição x aleatória: **Create Instance**; x: **random(room_width)** ; y: -16
      No atributo Layer deve constar o nome exato da camada de objetos definida na room (provavelmente "Instances_1")
    - Depois coloque um alarme: **Set Alam Countdown**, com 200 em countdown.

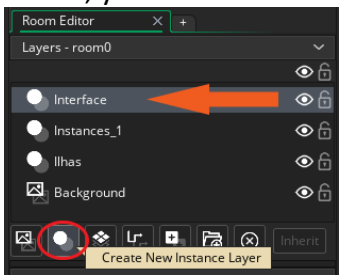
- Adicione **evento Alarm 0**, que será usado quando o alarme disparar, onde iremos acrescentar outro inimigo e configurar o alarme para disparar novamente:
    - **Create Instance** igual ao anterior, com x: **random(room_width)** ; y: -16
      Layer: "Instances_1"
    - **Set Alam Countdown**, com 500 em countdown.

Coloque este objeto na sala e teste o jogo. Agora vários inimigos podem aparecer ao mesmo tempo, pois quando o alarme dispara, cria mais um inimigo, e se configura para disparar de novo, fazendo um ciclo infinito.

Passo 4 – vidas, pontuação e dano:

Para melhorar o jogo, podemos fazer a colisão com o inimigo gerar um dano no avião do jogador. Assim teremos uma barra de energia na tela, sendo que o avião só será destruído se o dano for o máximo possível. Além disso, podemos fazer ele ter três vidas. Tudo isso será mostrado em uma barra de status (interface) na base da tela.

- Crie um objeto (controlador_vida). Usaremos o evento **Draw**, que determina o que será desenhado na tela. (obs: muitas ações que tem "draw" no nome só funcionam dentro deste evento)
    - Neste evento iremos desenhar na base da tela a imagem de interface, para isso crie um Sprite, carregue o arquivo bottom.gif, chame de spr_interface.
    - De volta ao evento **Draw** do controlador_vida, adicione **Draw Sprite** indicando o sprite da interface, com as coordenadas x: 0; y: 404.
    - Para ter certeza que será desenhado acima de todos os outros objetos, entre na sala do jogo (room), crie uma nova camada do tipo instance, chame de interface (clique da direita para renomear). Adicione o objeto controlador_vida nesta camada.



Teste o jogo, a interface deverá aparecer.

Nesta interface será mostrada a pontuação, número de vidas e a barra de energia. Então de volta ao evento **Draw** do objeto controlador_vida, use:

- **Set Draw Colour**, para determinar a cor que será usada, escolha um tom de amarelo que combine com a interface.
- Iremos usar uma variável global para pontuação, então adicione **Draw Value**, com as coordenadas x:180; y: 440. Em value coloque **global.pontos** (no atributo Caption deixe em branco, pois já está escrito "score" na imagem da interface).
- Iremos usar uma variável global para energia, então adicione **Draw Healthbar** (que desenha uma barra na tela). Em value coloque **global.energia** (em Left: 12; Top: 449; Right: 138; Bottom: 459). Escolha cores para fundo da barra (background), quando estiver cheio (max colour) e quando estiver no mínimo (min colour).
- Por último iremos desenhar as vidas como imagens (pequenos aviões). Para isso crie o sprite (life.gif) [sem fundo]. De volta ao evento **Draw** do controlador_vida, use a ação **Draw Stacked Sprites** com as coordenadas x: 16; y: 410 (escolha em Sprite a imagem das vidas). No atributo Number devemos colocar a variável global que usaremos para guardar a quantidade de vidas: **glogal.vidas**

O que falta para nossa interface funcionar é determinar no início do jogo quantas vidas o jogador tem, a quantidade máxima de energia, e deixar a pontuação zerada.

- Adicione o evento **Create** no controlador_vida, use:
    - **Set Global Variable**, com nome **pontos**, valor 0 (sem relative).
    - **Set Global Variable**, com nome **energia**, valor 100 (sem relative).
    - **Set Global Variable**, com nome **vidas**, valor 3 (sem relative).

  Nota: nesta ação não se deve colocar **global.** antes do nome da variável.

Para somar pontos ao atirar no avião:

- Vá no objeto inimigo, no evento **colisão com o tiro do jogador** (que já existe), adicione **Set Global Variable**; com nome **pontos**, valor 5 e relative ligado.

Teste o jogo. A interface funciona, mas o jogador ainda não perde energia (morre direto). Precisamos mudar isso, quando bater o jogador deve só perder energia. E irá morrer só se a energia chegar a zero.

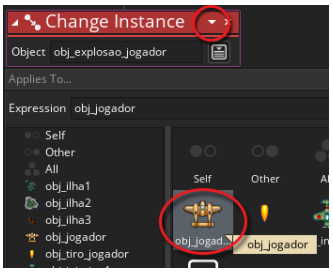
- Deve ser alterado o evento de **colisão do avião inimigo com o jogador**. Agora ele não deve matar o jogador, só tirar energia.
    - **Apague** a ação que troca o objeto do jogador para a animação de explosão (**Change Instance**)
    - **Apague** a ação que destrói o avião inimigo (**Destroy Instance**)
    - Mantenha a ação que toca o som de explosão (**Play Audio**).

- Quando o inimigo bater no avião do jogador, ele irá criar a explosão menor, ir de volta ao topo da tela e diminuir a energia do jogador. Então use:
    - **Create Instance**, configure para a explosao_inimigo e x:0, y:0 (ligue relative em ambos) Em Layer deve constar o nome da camada definida na room (provavelmente "Instances_1")
    - **Jump to Point**, x: **random(room_width)**; y: -16
    - **Set Global Variable**, com nome **energia**, valor -30, relative ligado.

Ok, tudo funciona, porém o avião do jogador não está morrendo nunca...

Para fazer isso, precisamos testar constantemente (evento Step) se a energia chegou a zero.

- Dentro do controlador_vida, adicione evento **Step**. Adicione a ação **If Variable**, onde diz variable: **global.energia** (e onde diz Is, selecione Less or Equal)
- Quando a energia acabar, vamos tocar o som de explosão, trocar o objeto do jogador para a explosão e deixar a energia de novo em 100%; para isso use: (encaixe na direita da ação de teste para rodar somente em caso positivo)
    - **Play Audio** (use o som de explosão mais forte =2)
    - **Change Instance**, em object selecione explosao_jogador. Esta troca deve ser aplicada no avião do jogador, então clique na seta e escolha obj_jogador
    - **Set Global Variable**, com nome **energia**, valor 100 (sem relative).



Agora o avião está morrendo quando acaba energia, mas o jogo já acaba. Ele deve só perder uma vida, e continuar:

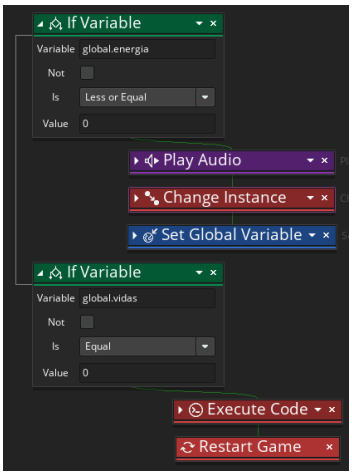
- Dento do objeto explosao_jogador, evento **Animation End**:
    - Deixe a ação **Destroy Instance**, para a explosão desaparecer.
    - **Apague** a ação Restart Game.

    - Vamos criar uma nova cópia do avião do jogador: adicione **Create Instance**, posição x:0, y:0, relative ligado ambos (ou seja, será no mesmo lugar onde o avião estava antes). Em Object selecione obj_jogador. Em Layer deve constar o nome da camada definida na room (provavelmente "Instances_1")
    - Para tirar uma vida: adicione **Set Global Variable**, com nome **vidas**, valor -1, relative ligado.

Ok, agora o avião morre quando a energia acaba, perde uma vida e continua jogando. Porém mesmo perdendo todas as vidas, o jogo não acaba nunca!

Para resolver isso:

- No objeto controlador_vida, dentro do evento **Step**:
    - Adicione a ação **If Variable**, onde diz variable: **global.vidas** (onde diz Is, deixe Equal). **Importante:** esta ação deve rodar sempre, então encaixe abaixo do teste anterior (da energia), para que não fique ligado diretamente com as últimas ações que rodam só em caso positivo **(veja na figura como deve ficar a estrutura geral depois de acabar esta seção)**.
    - Para mostrar uma mensagem de "Game Over" quando acabar o jogo, use a ação **Execute Code**, e digite: **show_message("Game Over")**
    - Por fim adicione a ação **Restart Game**.



Agora o jogo está totalmente jogável e funcional.
Para deixar mais interessante, podemos criar outros inimigos, com comportamento diferente (ex: eles atiram).

Passo 5 – mais inimigos:

Como o novo inimigo é parecido com o já existente, duplique o objeto inimigo1 (clique com botão direito sobre objeto e Duplicate) e altere a cópia colocando um novo nome e novo sprite. Para isso crie um sprite (enemy2.gif) [sem fundo e coloque origem no centro].

Este inimigo é mais perigoso, então faremos ele dar 10 pontos ao jogador quando acertado. Na **colisão dele com o tiro**, altere a ação da pontuação para dar 10 pontos ao invés de 5.

Para o inimigo atirar, precisamos de um novo sprite (enemy_bullet1.gif) [sem fundo e coloque a origem no centro] e um objeto que o use, chame de tiro_inimigo. Então adicione:

- No evento de criação (**Create**), use **Set Speed** (Type: Vertical), para o tiro já nascer em movimento (Speed 8).
- Ele deve "morrer" quando sair da sala. Então no evento **Other – Outside Room**, use **Destroy Instance**.
- Quando o tiro colidir com o avião do jogador, deve tocar som de explosão, desaparecer, diminuir a energia do avião em -5. Então adicione o evento de **colisão com o jogador** e use:
    - **Play Audio** (use o som de explosão);
    - **Destroy Instance**;
    - **Set Global Variable**, com nome **energia**, valor -5, relative ligado.

A programação do tiro do inimigo está pronta. Agora precisamos fazer o inimigo2 atirar.

Podemos fazer a decisão de atirar ou não do inimigo ser algo aleatório. Uma forma de fazer isso é usar a ação **Get Random Number**, esta ação sorteia um valor (entre um número mínimo e máximo). Podemos fazer um teste logo após, caso o número sorteado for 1 a próxima ação é executada. Ou seja, quanto maior o intervalo de valores, menor a chance de cair 1. Por exemplo, se quisermos que a ação tenha 50% de chance de ser executada, criamos um intervalo com 2 valores. Se quisermos uma chance menor, aumentamos o intervalo.

Podemos usar este teste no evento **Step**, que fará o teste a cada "passo" do jogo (lembrando que 30 passos = 1 segundo). Assim usando um intervalo de 30 valores terá em média uma condição verdadeira a cada 30 passos (ou seja, 1 tiro a cada segundo).

- Adicione evento **Step** no avião inimigo2, coloque a ação **Get Random Number** (Type: integer, pois não interessa valores quebrados), usando como valor mínimo: 1, valor máximo: 30, em Target use um nome de variável como: **teste**.
- Logo após coloque a ação **If Variable**, usando o mesmo nome de variável anterior. Configure para testar se é igual a 1.
- Então adicione **Create Instance**, configurando para criar um tiro inimigo na posição x:0 ; y:16 com relative ligado (ambos). No atributo Layer deve constar o nome da camada de objetos definida na room (provavelmente "Instances_1"). Lembre de encaixar esta ação na direita do teste anterior para rodar só em caso positivo.

Para colocar este inimigo no jogo, podemos utilizar o objeto **controlador_inimigo**:

- No evento de criação (**Create**) crie um novo alarme com **Set Alarm Countdown**, sendo Alarm 1 e 1000 em countdown.
- Adicione o **evento Alarm 1**, crie uma cópia do inimigo2 em uma posição x aleatória, de forma semelhante ao que foi usado no Alarme 0 (pode copiar e colar a ação).
- Então coloque novamente a ação **Set Alarm Countdown**, com Alarm 1 e 500 em countdown.

Desta forma o inimigo 2 deve aparecer depois de aproximadamente 30 segundos de jogo, e voltar a aparecer a cada 15 segundos.

Atirando na direção do jogador

Iremos agora criar outro tipo de inimigo, que dispara um tiro que vai na direção do jogador. Para isso duplique o objeto inimigo2 e crie um novo sprite (enemy3.gif) [sem fundo e coloque a origem no centro]. Use então este sprite para o inimigo3.

Este inimigo é mais perigoso que os outros, então daremos ao jogador 20 pontos quando destruí-lo. Ajuste a pontuação na ação respectiva, no evento de **colisão com o tiro**.

O tiro do inimigo3 será especial, então vamos usar um sprite diferente (enemy_bullet2.gif). Duplique o objeto tiro_inimigo e use este sprite. Troque o nome para tiro2_inimigo.

Vamos configurar o objeto inimigo3 para atirar este tiro novo:

- No evento **Step**, na ação que sorteia um valor (**Geat Random Number**), troque o valor máximo para 80, pois este tiro é muito difícil de desviar, se tiver vários ao mesmo tempo na tela será muito complicado para o jogador.
- Na ação que está abaixo (**Create Instance**), configure para criar o tiro2_inimigo.

Então agora o avião inimigo 3 lançará em média 1 tiro a cada 80 passos (2,6 segundos).

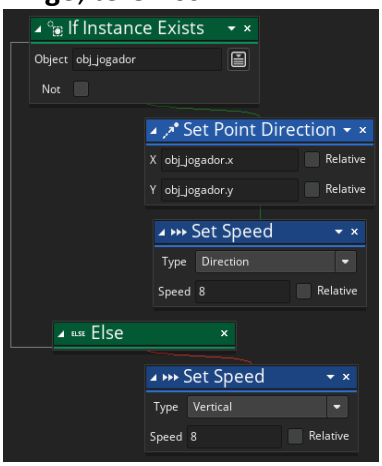
Vamos programar o comportamento do **tiro2_inimigo**:

- No evento de criação (**Create**), use: **Set Point Direction**, esta ação direciona para um ponto. Este ponto em questão será a posição do avião do jogador. Para isso podemos usar as variáveis **obj_jogador.x** e **obj_jogador.y**
- Arraste a ação **Set Speed** (que já existe) para baixo da anterior. Deixe o atributo Type como Direction, para usar a variável que guarda a direção de movimentação, definida na ação anterior. Deixe 8 em Speed.

Obs: Para acessar o valor de variáveis internas de outro objeto, podemos usar *nome_objeto*.variável. Neste caso, como queremos variáveis de posição x,y usamos *nome_objeto*.x ; *nome_objeto*.y
Quando há várias cópias do mesmo objeto na tela, é retornada a informação referente a primeira cópia criada. Quando não há nenhuma cópia, é gerado um erro.
Isto pode ser um problema, pois no momento que o avião do jogador é destruído, não há nenhuma cópia deste objeto no jogo, e se o inimigo3 tentar atirar neste mesmo instante, teremos um erro.

Podemos resolver isto testando se existe o avião do jogador na tela, e só então permitindo que o tiro seja lançado. Caso contrário, para evitarmos o erro, o tiro seguirá em vertical para baixo, sem seguir nenhum ponto.

- Retomando: no evento de criação (**Create**) do **tiro2_inimigo**, teremos:
  - **If Instace Exists** (testa se existe uma cópia do objeto), configure objeto: obj_jogador.
  - Encaixe na direita do teste as ações (já existentes)**: Set Point Direction** e **Set Speed**, que irão rodar só em caso positivo.
  - Para especificar o que será feito em caso negativo, adicione **Else**.
  - Por fim, adicione **Set Speed**, com Type: Vertical, Speed: 8. Encaixe na direita do Else para rodar só em caso negativo.



Agora só precisamos adicionar este novo inimigo ao jogo. Podemos usar novamente o **controlador_inimigo**, adicionando no evento de criação (**Create**) mais um alarme – **Set Alarm Countdown** (Alarm 2 = 2000 countdown).
Então adicione o **evento Alarm 2**, copie as ações que estão em algum outro alarme deste objeto e modifique para criar o inimigo3 e rearmar o alarme 2 em 1000 countdown.
(Desta forma, o primeiro inimigo3 deve ser criado em 66 segundos de jogo, e os próximos a cada 33 segundos).

Passo 6 – usando timelines (linha do tempo):

O jogo na versão atual está funcionando e dá algum desafio ao jogador, pois fica mais difícil com o tempo. Porém não temos controle sobre a criação dos aviões inimigos, eles podem surgir em qualquer posição, até uns por cima dos outros, sem nenhuma formação interessante. Além disso, eles não desaparecem nunca, sempre retornam a todo momento.

Seria melhor para o jogo ter um controle maior sobre a criação dos inimigos, possibilitando a construção de desafios mais elaborados. Quando a criação dos inimigos são em locais definidos, podemos criar formações dos aviões (ex: voando em V) e permitir que o jogador fique mais experiente no jogo, pois cada vez que joga os aviões aparecerão no mesmo lugar. Isso permite que ele treine a cada jogada e melhore sua performance.

Para criar eventos específicos que ocorrem com o passar do tempo podemos usar um recurso do GameMaker, chamado “timelines”. Nesta linha do tempo podemos criar ações a cada momento que quisermos.

Temos que ajustar o jogo para usar a timeline. Como ela que irá controlar a criação dos inimigos, vamos modificar o jogo para que cada inimigo destruído realmente “morra” e não volte a aparecer no topo da tela em outra posição.

- Para isso, no objeto **de cada inimigo**, no evento **colisão com o jogador**, remova a ação **Jump to Point** e coloque **Destroy Instance** no lugar.
    - No evento **colisão com o tiro**, faça a mesma coisa.
    - No evento **Outside Room**, faça a mesma coisa.

Vamos então criar uma timeline e configurar para ela criar os inimigos:

- Crie uma nova timeline (no menu geral na direita). Coloque o nome que quiser.
- Adicione um momento de tempo (botão **Add – 0**). Use **Create Instance** (inimigo1; x:320 y: -16); em Layers o nome da camada (provavelmente “Instances_1”). [ou seja, no momento 0 cria inimigo1 no meio da tela em x, e acima da tela em y)

- Adicione outro momento de tempo (**Add – 30**; clique da direita sobre o valor para alterar – change moment). Agora vamos criar 2 inimigos, um no lado do outro. Então use **Create Instance** (inimigo1; x:240 y: -16), e mais outro **Create Instance** (inimigo1; x:400 y: -16). Não esqueça dos nomes corretos das camadas em Layer.

- Adicione mais um momento de tempo (**Add – 100**) e deixe ele em branco (para que quando a timeline repetir não misture o momento de tempo final com o primeiro).

Porém uma timeline sozinha não faz nada, sempre é necessário que um objeto execute a ação que faz ela rodar.

Podemos usar o **controlador_inimigo** para isso. No seu evento de criação (**Create**) use:

- **Set Instance Timeline**; selecione a timeline que criou. Clique em Loop para ficar cíclica, infinita (sempre que acaba, começa de novo).
- **(apague todas as outras ações e eventos deste objeto; obs: clique da direta sobre nome do evento para abrir menu e remover)**

Teste o jogo. Deve surgir um inimigo no centro da tela, e 1 segundo depois mais dois inimigos, um no lado do outro (pequena formação em V).
Obs: se no início houver algum outro avião fora da formação, ele foi colocado manualmente na room, remova.

Crie novos momentos de tempo na timeline, gerando novos inimigos. Planeje o fluxo e as formações de entrada de inimigos na tela, criando uma experiência interessante para o jogador.

Atenção: se o avião inimigo for criado completamente fora da sala de jogo ex: y: -32, o evento Outside Room entrará em ação. Se lá estiver a ação Destroy Instance, o avião será destruído antes de entrar na sala.

Nesse caso não devemos usar o evento Outside Room para destruir o avião quando sair da sala, e sim usar o evento Step, testando se sua posição em y está maior que a altura da sala. Desta forma: If Variable (Variable: y ; Value: room_height ; Is: greater)

Então se isto for verdadeiro, o avião é destruído.
Se for falso, ele sorteia um valor para ver se atira; pois este seria seu comportamento normal no evento Step, no caso do inimigo 2 e 3.