# Duduzebu o Game

Deste jogo você controla um pássaro tocando em qualquer lugar da tela. Quando você toca a tela o pássaro sobe; se você não fizer isso, o pássaro começa a cair, seu objetivo é levar o pássaro para a esquerda ou para a direita sem bater em nenhum ponto. Se você conseguir levar o pássaro para a esquerda ou para a borda direita, você faz uma pontuação. Quando isso acontece, o pássaro vai para a direção oposta. Isso continua até você acertar um pico.

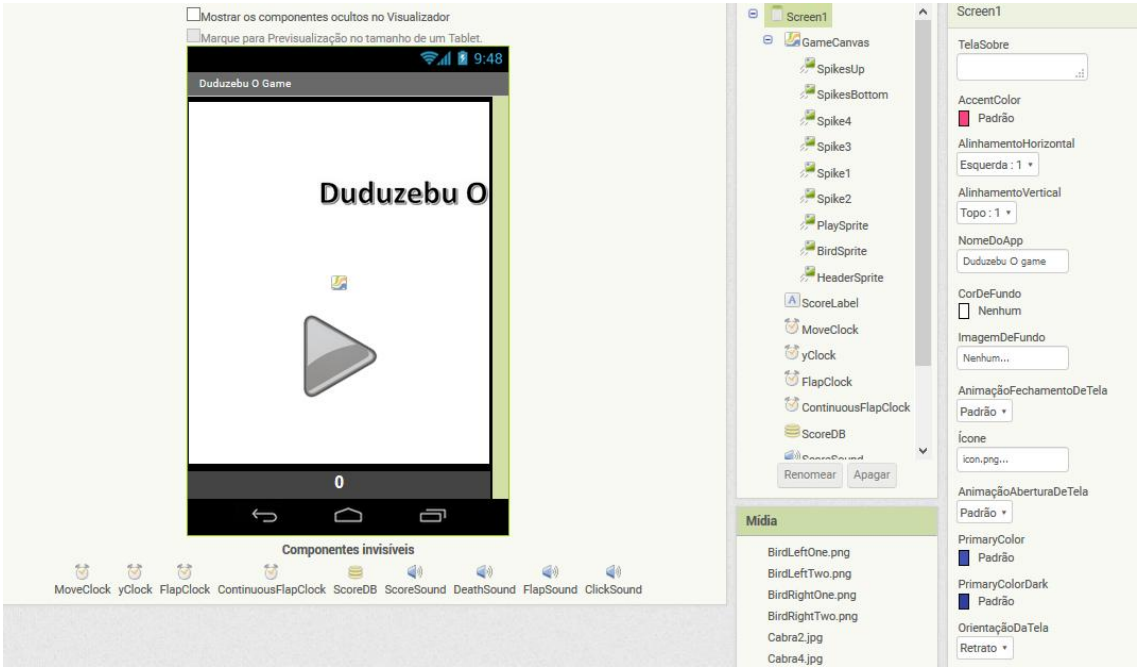
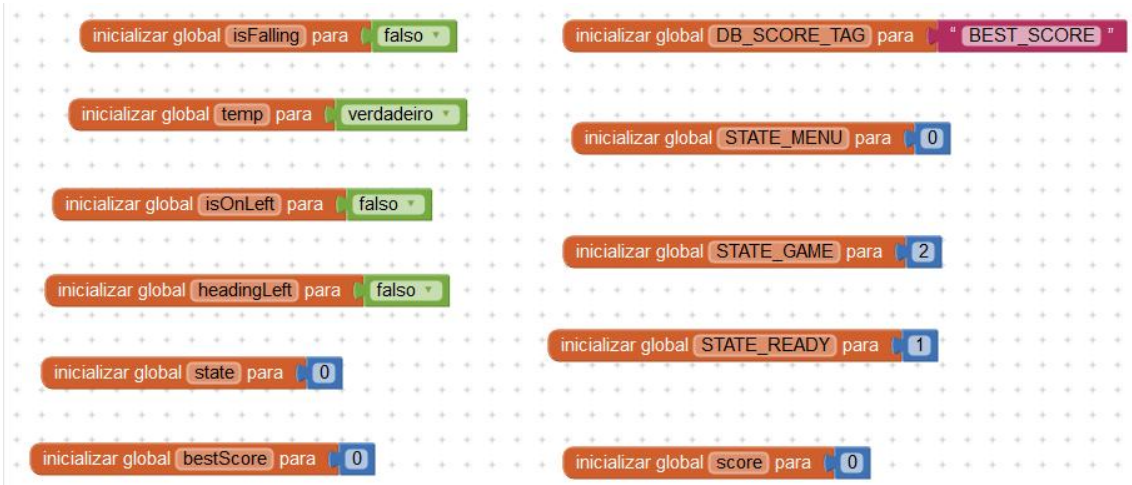



**Imagem inicial do aplicativo**




**DB_SCORE_TAG**: para salvar e ler a melhor pontuação.

**STATE_MENU**, **STATE_READY**, **STATE_GAME** - Um jogo tem estados. Quando você está na tela do menu, pronto para começar o jogo e realmente jogar o jogo. Essas variáveis são constantes, o que significa que não mudaremos seus valores. É por isso que usamos todas as letras maiúsculas para defini-las.

**state** - Isso mantém o estado atual, qualquer um dos três estados acima.

**isFalling** - Quando o jogador não está tocando, o pássaro começa a descer.

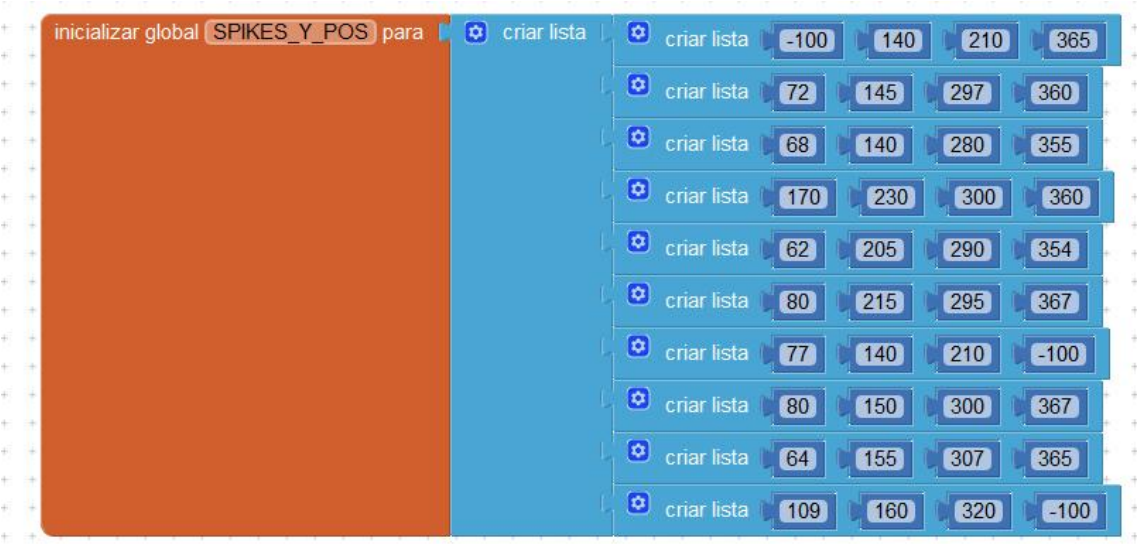
**score** - Ponto atual no jogo.

**bestScore** - Melhor pontuação do jogador.

**headingLeft** - A direção de Bird no eixo x, esquerda ou direita.

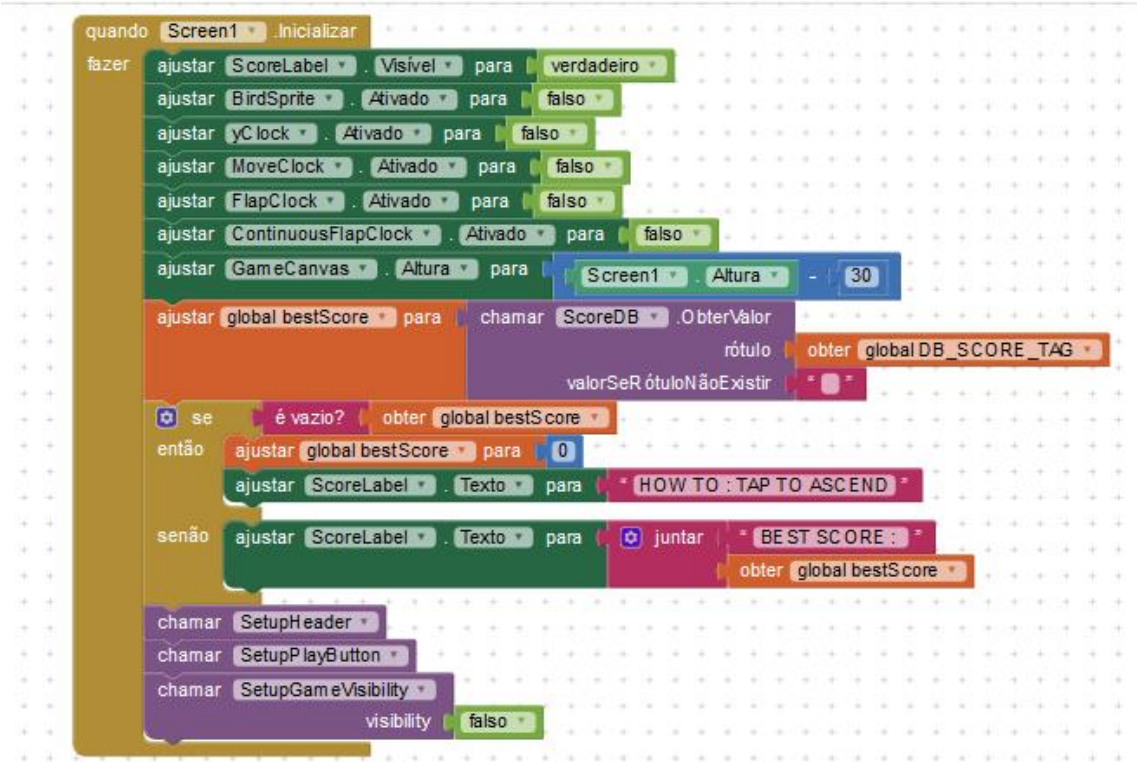
**temp** - Para usar dentro de um escopo menor. Você pode usar variáveis locais, se preferir.

**isOnLeft** - Se os obstáculos estiverem no lado esquerdo da tela.




**SPIKES_Y_POS** - Esta variável contém uma lista de listas. Temos **4** sprites de imagem (picos de obstáculos) que são colocados em ambos os lados, dependendo da direção do pássaro. Se quisermos colocar os picos à esquerda, podemos usar **0** como x, se quisermos, podemos usar a largura da tela menos a largura do ponto para **x**. Como a posição x dos nossos obstáculos é para a esquerda ou para a direita, tudo o que precisamos são valores no eixo **y**. Criamos **10** locais para esses sprites, colocando-os em várias posições na visualização do projeto, com intervalos adequados entre eles. Depois, verificamos o dispositivo se eles pareciam bons o suficiente. Em seguida, **copiámos** os valores **Y** da vista de **estrutura** para **SPIKES_Y_POS** variável. Como temos 4 sprites de obstáculos, escolhemos 4 valores no eixo y, um para cada sprite. É por isso que cada lista contém **4** itens. O primeiro é para o Sprite 1, o segundo é para o Sprite 2 e assim por diante. Você pode tentar adicionar mais ou até mesmo randomizar. Em vez de ter uma lista de listas

predefinidas, você pode criar uma lista vazia e depois adicionar itens a essa lista. Se você está se perguntando por que alguns deles têm **-100**, é porque nem sempre queremos mostrar todos os 4 sprites de obstáculo. Então nós os colocamos fora da tela. Você pode usar **-30** como cada pico é de 30 pixels de **altura**.




Vamos dividir isso. A tela é inicializada quando o aplicativo é carregado e nós fazemos o seguinte quando ele é:

**1.** Queremos que o nosso rótulo de pontuação seja visível. Então, definimos sua visibilidade como **verdadeiro**. Você também pode fazer isso na visualização do projeto.

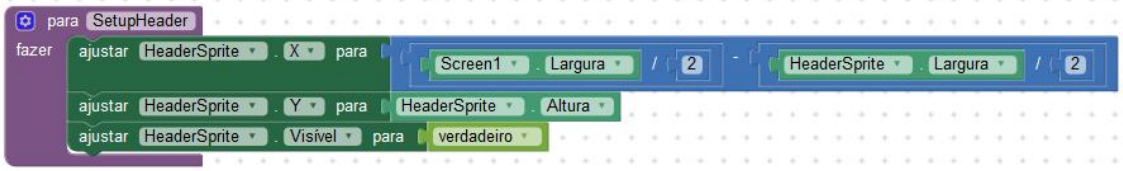
**2.** Quando mostramos o aplicativo pela primeira vez, não mostramos o pássaro. Por isso, não deve ser ativado também. Você também pode fazer isso no modo de design. Eu já poderia ter feito isso na visão de design; mas eu não gosto de ir e voltar para verificar, então refiz.

**3.** Nós não queremos que nossos relógios façam nada, então os desativamos, o que você também pode fazer no modo de design.

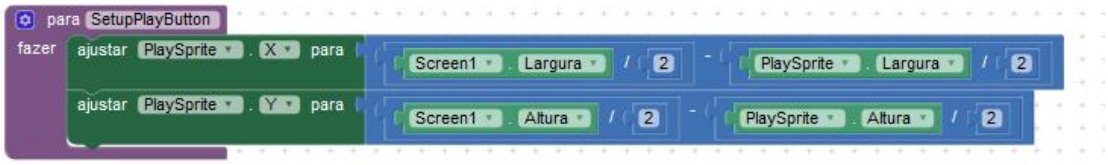
**4.** Queremos que nossa altura de tela seja a mesma que a altura do dispositivo menos a altura da etiqueta de pontuação, que é 30. Mostramos a melhor pontuação na tela do menu ou como jogar se o jogador ainda não jogou o nosso jogo.

**5.** Nós lemos a melhor forma de pontuação do TinyDB usando a tag que definimos anteriormente. Se não **obtivermos a** melhor pontuação, obteremos um texto vazio porque não colocamos nada no bloco "**valueIfTagNotThere** ". Obviamente, a tag não estará disponível se não tivermos salvado nenhuma pontuação ainda.

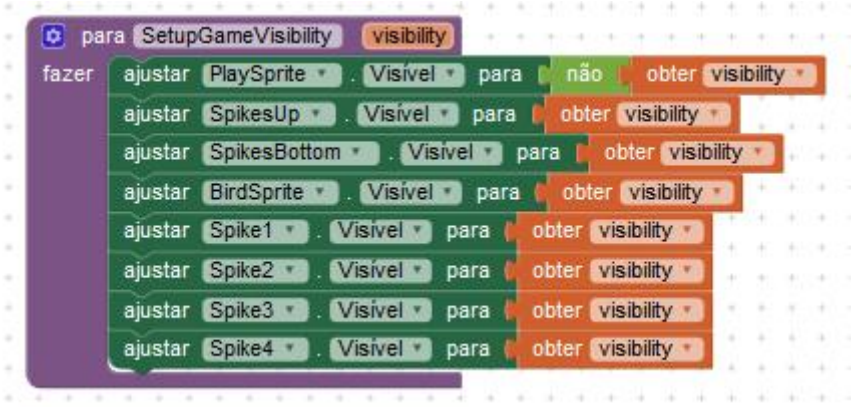
**6.** Se encontrarmos uma melhor pontuação, mostramos isso. Se não, informamos ao jogador como jogar este jogo. Você pode criar uma tela com instruções detalhadas se preferir e mostrar isso.




O procedimento **SetupHeader** define a localização do cabeçalho no centro em relação ao eixo x e um pouco abaixo da parte superior da tela usando sua altura. O cabeçalho não contém nada além do nome do jogo que é um sprite de imagem e não se move ou interage. É por isso que, na visualização de design, desativamos isso. Depois de configurarmos sua posição, nos certificamos de que esteja visível.



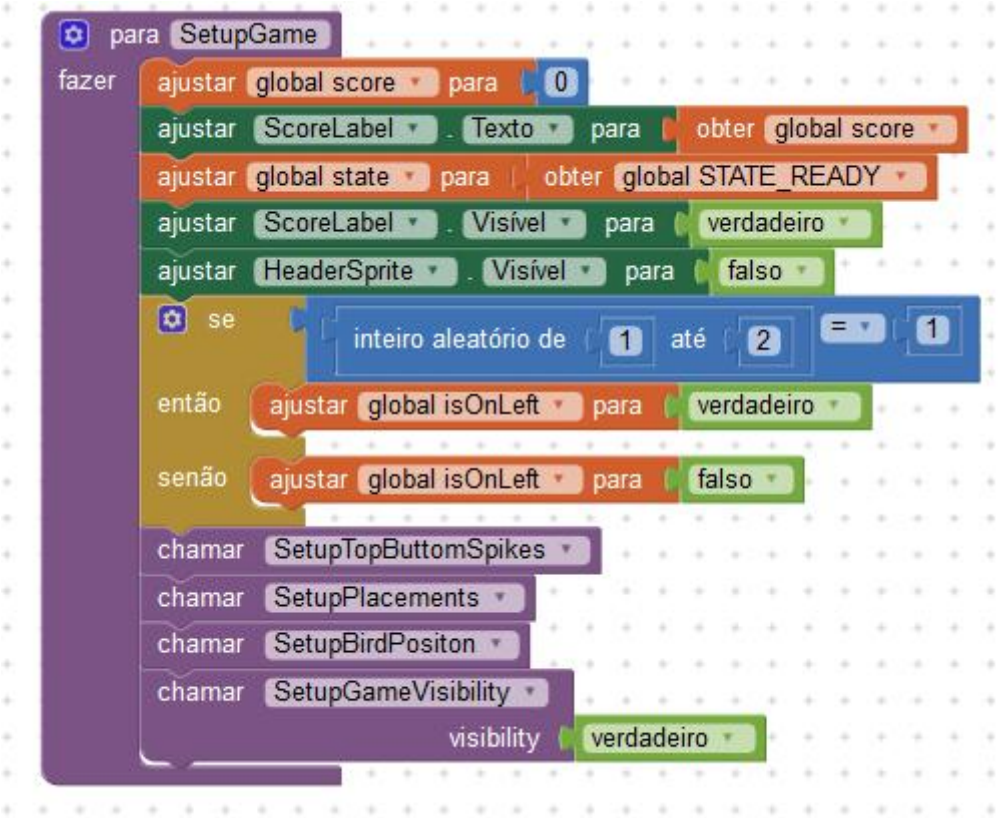

O procedimento **SetupPlayButton** coloca o botão de reprodução no centro da tela.




O procedimento **SetupGameVisibility** usa uma variável **booleana** como parâmetro. Nós nomeamos a variável **visibilidade**. Se é verdade, mostramos os componentes do jogo, caso contrário, ocultamos. A razão que usamos **não** bloquear a partir da lógica, para definir a visibilidade do botão play é porque quando o jogo começa, não quero mostrar o botão play, mas o jogo itens. Quando o jogo termina, queremos esconder todos os componentes do jogo, exceto o botão play, que também é o mesmo quando o nosso jogo é

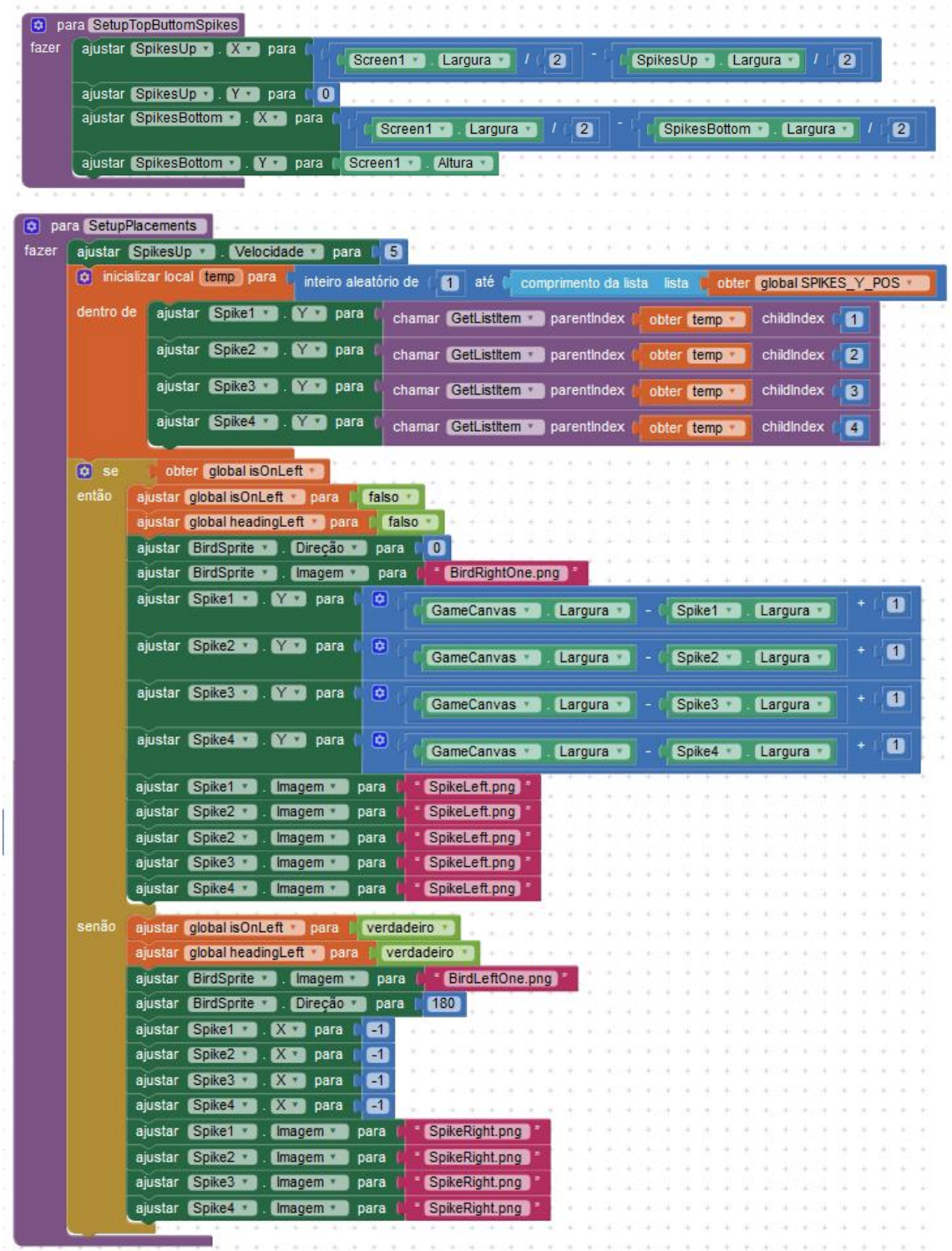
iniciado. É por isso que chamamos esse procedimento com um argumento **falso** de **Screen1.Initialize**, porque queremos mostrar apenas os itens da tela do menu, não os itens da tela do jogo.

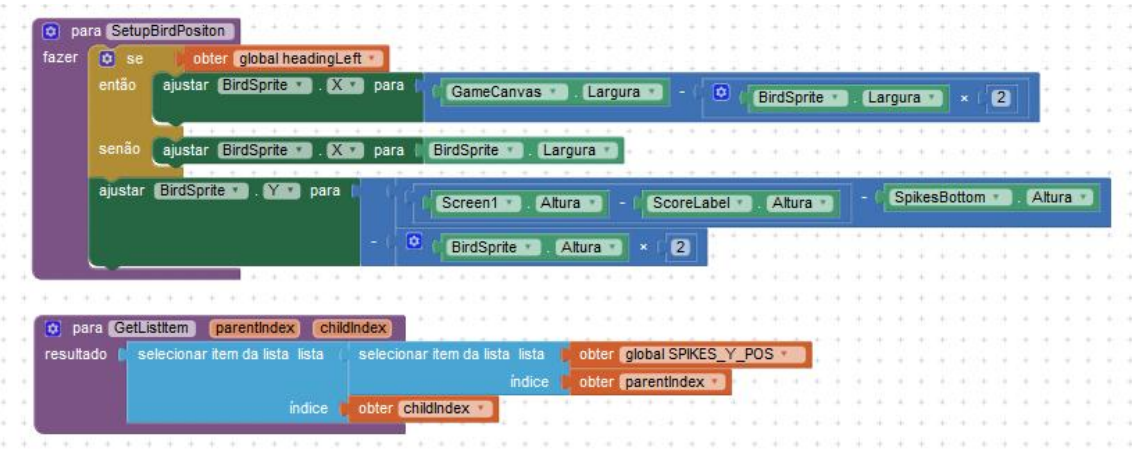

Quando o jogador toca no botão de reprodução, primeiro tocamos um som de **clique**. Então nós configuramos o jogo que eu explicarei daqui a pouco. Também começamos o nosso **ContinuousFlapClock**, pois queremos que o pássaro bata continuamente.




Vamos discutir as 4 chamadas de procedimento no **SetupGame** mais tarde. Primeiro será explicado o que acontece no começo. Então o jogo está prestes a começar. Queremos redefinir o valor da pontuação para 0. Em seguida, mostramos a pontuação que definimos como 0. Fazemos isso porque, anteriormente, a tela do menu mostrava melhor pontuação ou um tutorial. Nós mudamos o estado para estado pronto, o que significa que o jogo está esperando o primeiro toque do jogador para começar. Nós nos certificamos de que nossa etiqueta de pontuação esteja visível e que o cabeçalho esteja invisível. Queremos colocar os obstáculos aleatoriamente. Nós não queremos

começar o jogo com picos à esquerda ou direita o tempo todo, nem queremos apenas alternar. Então, escolhemos um número entre 1 e 2 inclusive. Se o sistema nos der **1**, nós setamos o **isOnLeft** para verdadeiro, caso contrário, falso. Agora vamos dar uma olhada em outros 4 procedimentos:






Em **SetupTopBottomSpikes**, definimos os valores **X** de sprites superior e inferior para o centro no eixo x usando a fórmula simples de ponto médio. Como o SpikesUp deve estar no topo, definimos o **Y** como **0;** e para o outro, colocamos na **altura** da **tela** para que apareça na parte inferior. Lembre-se na tela, o ponto **(0,0)** está no canto superior esquerdo da tela. Em **SetupPlacements**, garantimos que a velocidade do pássaro seja **5**. Então determinamos onde devemos colocar todos esses 4 picos de obstáculos em termos do eixo **y**. Como nós temos uma lista de posições pré-definidas contidas Variável **SPIKES_Y_POS**, só precisamos escolher um da lista. Então, geramos um número entre **1,** que é o índice do primeiro item da lista e o tamanho da lista, que é **10,** pois temos 10 itens / listas, o que significa que obtemos um número entre **1** e **10,** inclusive. Depois disso, obtemos a lista no índice que acabamos de receber aleatoriamente. Nós não queremos usar o mesmo conjunto de blocos várias vezes para obter um item de lista, em vez disso, criamos um procedimento chamado **GetListItem,** que podemos usar facilmente em vez de usar o mesmo conjunto de blocos e fazê-lo parecer muito ocupado.

Como **SPIKES_Y_POS** é uma lista de listas, o valor da lista também é uma lista. Agora, se queremos a lista que contém [-100, 140, 210, 365], que está no índice 1 da lista pai, devemos dar **GetListItem 1** como o **parentIndex** e, em seguida, se quisermos obter o número 210, que está em índice 3 da lista de filhos, devemos dar **3** como o **childIndex** . Então nós verificamos se devemos colocar os obstáculos à esquerda usando o valor da variável **isOnLeft** que nós determinamos no procedimento **SetupGame**.

Agora você pode estar se perguntando por que estamos mudando o valor de **isOnLeft** novamente para o oposto. É porque o **SetupGame** é chamado apenas uma vez quando o usuário inicia um jogo, mas esse **SetupPlacements** será usado sempre que o jogador rebate uma borda. Se o jogador rebate na borda esquerda, precisamos colocar os obstáculos à direita e vice-versa. Depois disso, também mudamos o rumo para o oposto da direção atual da ave no eixo **x**. A posição definida como **0** faz com que a ave se mova para a direita. Também definimos a imagem que corresponde à direção da ave. Como definimos **isOnLeft** como false, o pássaro deve estar se movendo para a direita, configuramos o **X** para a borda direita

(largura da tela) menos sua largura. Colocamos um pixel extra (**+1**) porque eu acho que a aparência é melhor assim. Você não precisa adicionar **1**.

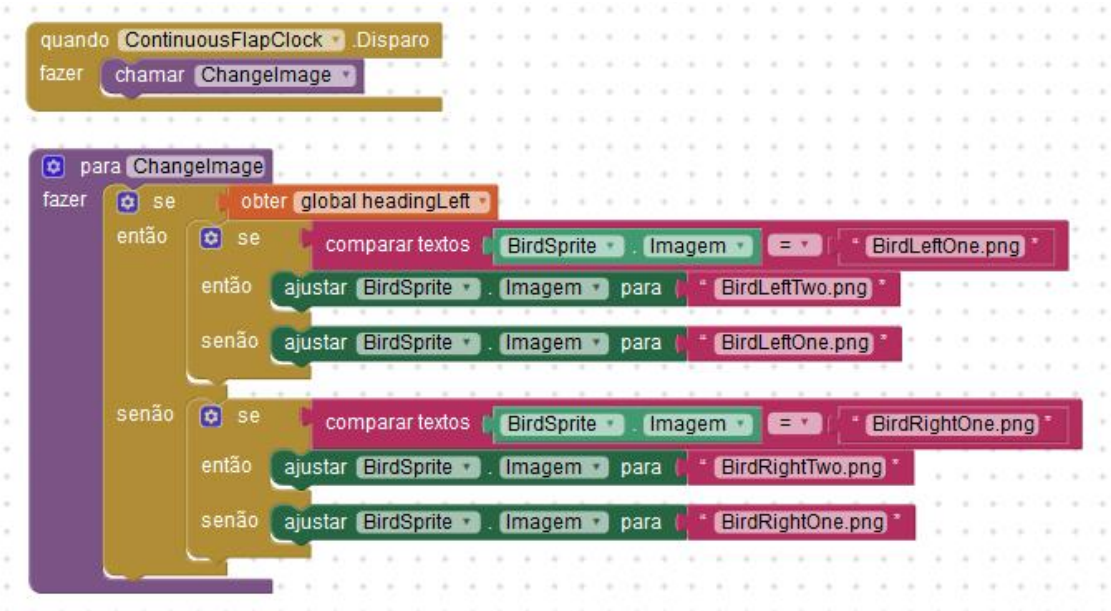
O nome da imagem "**SpikeLeft.png**" pode confundi-lo, mas pequenos picos nessa imagem estão apontando para a esquerda, por isso deve ser usado quando os obstáculos são colocados no lado direito. Na outra parte, fazemos exatamente o oposto.

A posição **180** faz com que a ave se mova para a esquerda. Nós colocamos os obstáculos **X** para **-1,** mas você pode ajustá-lo para **0**. Eu gosto assim. A mesma razão pela qual eu adicionei **1** ao colocá-los à **direita**.

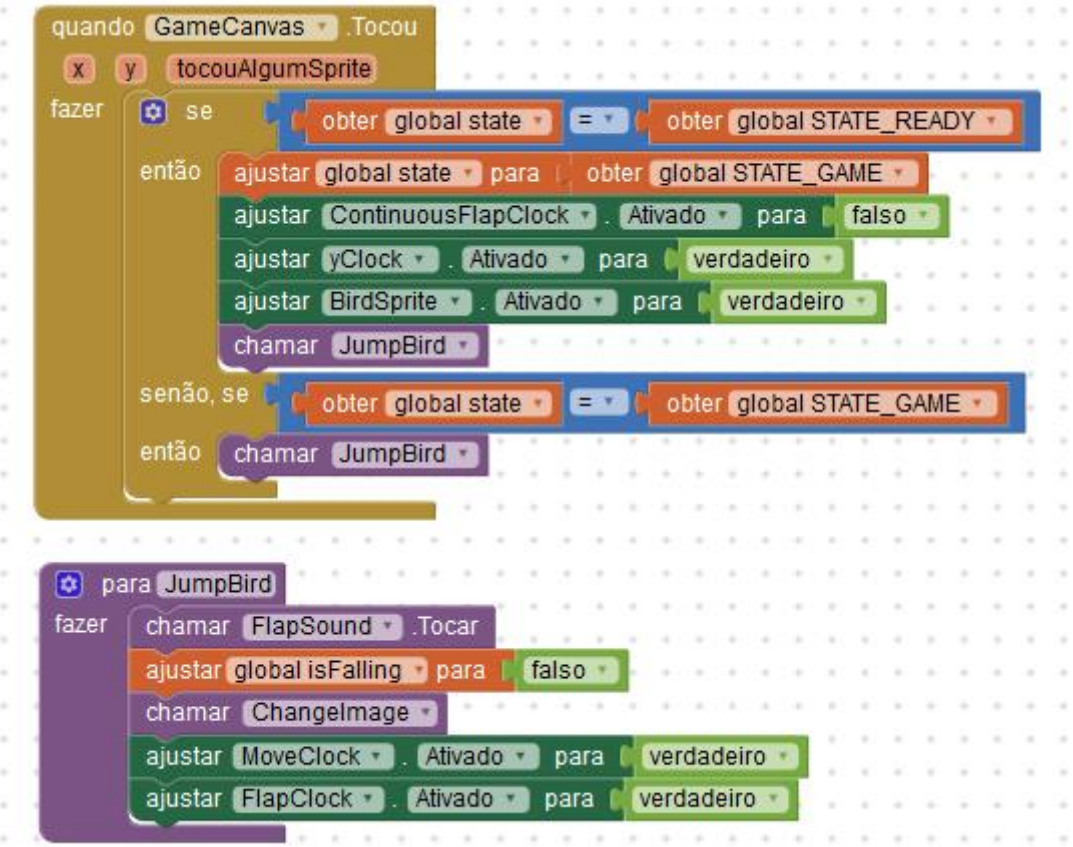
Depois de chamar **SetupPlacements**, chamamos **SetUpBirdPosition** no procedimento **SetupGame**. Em **SetupBirdPosition**, dependendo do título que definimos nos **SetupPlacements**, mudamos onde o pássaro deve ser inicialmente colocado quando o jogo está no estado pronto, que é na verdade o lado oposto de onde colocamos os picos de obstáculos. Você pode usar a propriedade de **título** do pássaro, se preferir, em vez de usar outra variável (**headingLeft**) como eu fiz. Mas você tem que fazer uma comparação e ver se é **0** (para a direita) ou **180** (para a esquerda).

O valor **Y** não depende do cabeçalho. Nós só precisamos ter certeza de que não vamos colocá-lo muito acima ou abaixo de onde ele toca o pico de baixo e morre imediatamente. No final do **SetupGame**, chamamos **SetupGameVisibility** e passamos verdadeiro como um argumento, o oposto do que fizemos quando chamamos esse procedimento de **Screen1.Initialize**. É porque desta vez queremos que as entidades / componentes do jogo fiquem visíveis e o botão play fique invisível.

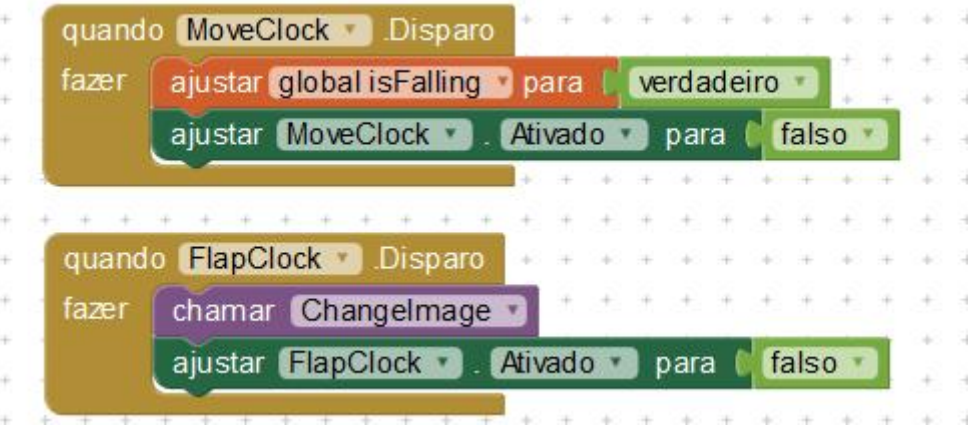
Estamos em estado pronto. O pássaro deve bater continuamente, mas o que exatamente faz bater? Se você se lembrar, no evento **PlaySprite.Touched**, **ativamos** o **ContinuousFlapClock**. No modo de design, definimos o intervalo do timer deste relógio para **300** milissegundos. Assim, uma vez ativado, a cada 300 milissegundos, o App Inventor invocará automaticamente o evento **Timer** deste relógio.




**ChangeImage:** este procedimento precisa saber a direção do pássaro. É por isso que habilitamos o **ContinuousFlapClock** depois que chamamos **SetupGame**. Se o procedimento **ChangeImage** vê que **headingLeft** está definido como **verdadeiro**, ele verifica se a imagem atual é **BirdLeftOne.png**, se for ele muda para **BirdLeftTwo.png** para fazer o pássaro parecer estar agitado. Se é a segunda imagem, ela muda para a primeira. Ele faz o oposto se o **headingLeft** for **falso**. Agora, estamos esperando que o jogador toque para começar a jogar. Como podemos começar o jogo e controlar depois de começar? Bem, nós fazemos isso sempre que há um toque na tela. O App Inventor chama **Screen1.Initialize** no início de uma inicialização de aplicativo, da mesma forma que ouve qualquer toque em uma tela por meio do evento **GameCanvas.Touched** sempre que um jogador toca a tela.

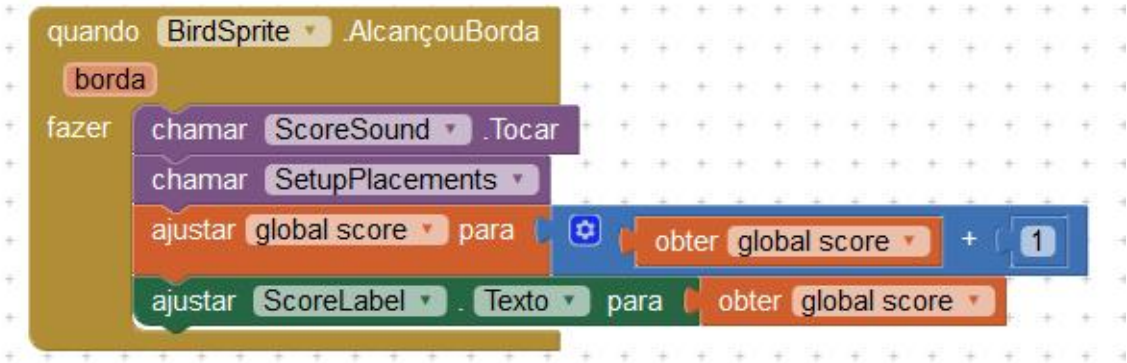



Apenas nos preocupamos com o toque do usuário quando o nosso jogo está no estado **pronto** ou no estado do **jogo**. Quando no estado pronto e o usuário toca pela primeira vez, nós mudamos o estado para o estado do jogo. Nós paramos de flapping contínuo do pássaro, desativando esse relógio. Nós habilitamos o **yClock**. Vamos ver daqui a pouco o que faz. Também habilitamos o pássaro para que ele possa ser movido para uma direção, dependendo do seu rumo atual. Sempre que há um toque, fazemos o pássaro pular, o que significa que mudamos seu valor **Y.** Para se mover ao longo do eixo x, definimos o rumo do pássaro; e na visão de design, também definimos a velocidade como sendo **5** . No procedimento **JumpBird**, tocamos o som da aba. Nós definimos **isFalling** para **falso** desde que estamos pulando, não caindo. Nós mudamos a imagem como explicamos para o **flapping** contínuo. Nós habilitamos tanto o **MoveClock** quanto o **FlapClock**.



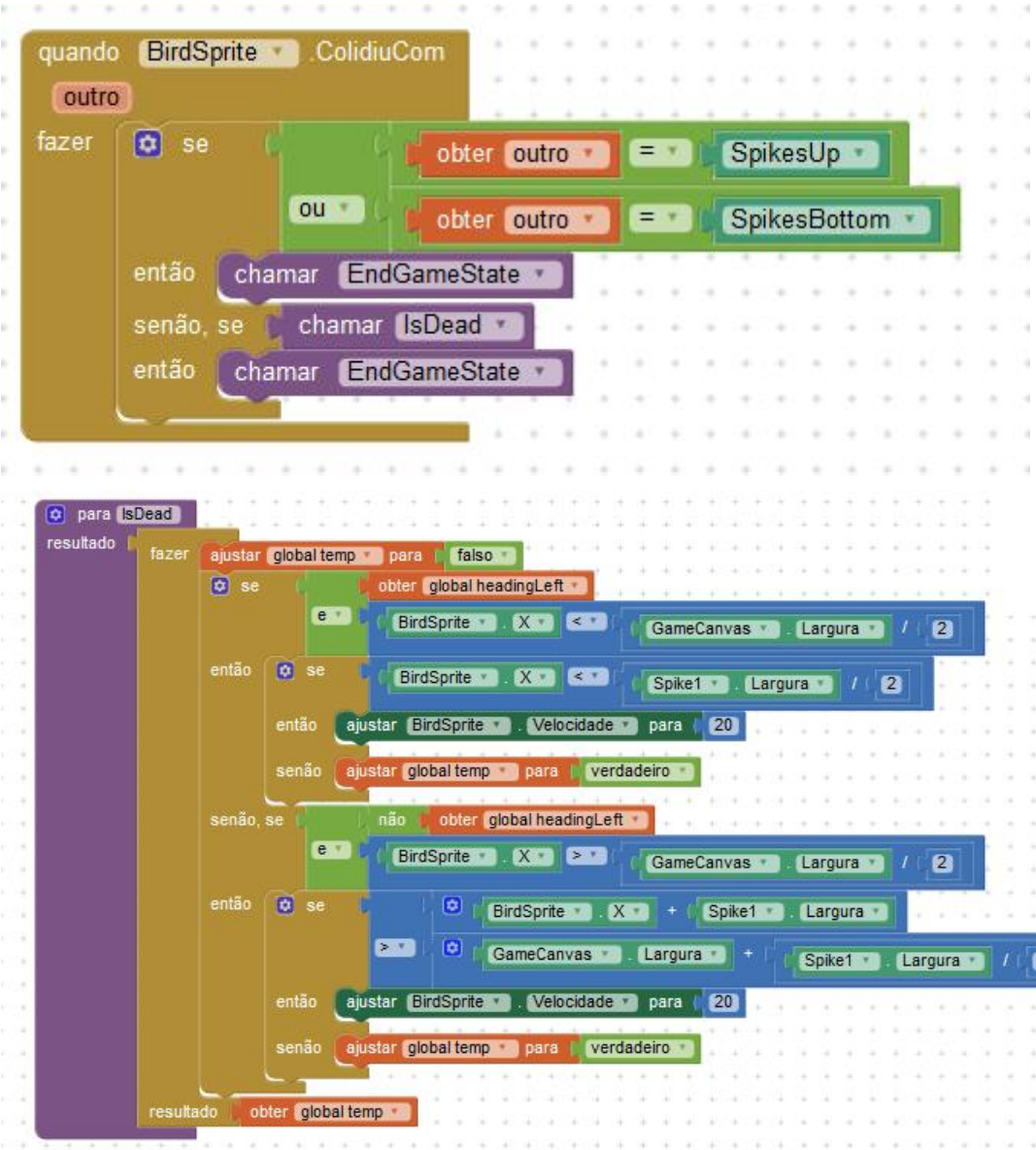
Se você se lembra do que eu mencionei sobre os relógios que temos no começo deste tutorial, você sabe que o **MoveClock** controla quanto tempo o pássaro pode voar quando o jogador toca na tela. Quando o MoveClock é disparado, paramos de voar e definimos **isFalling** como verdadeiro para que o pássaro **caia**. No evento **Timer** do **FlapClock**, mudamos a imagem do pássaro atual e desativamos o temporizador, pois não queremos que ele flua continuamente. O que faz a ave realmente subir ou descer? A resposta é **yClock**. Novamente, **yClock** é para controlar a localização da ave no eixo **y,** que tem um intervalo configurado para **0**. Então, quando é ativado, ele dispara continuamente e ele faz isso.


Se nosso pássaro cair, aumentaremos seu valor Y, já que a posição de Canvas (0, 0) está no canto superior esquerdo. Se o Y for igual à altura da tela, o pássaro estará na parte inferior. Nós fazemos o oposto se estamos subindo. Você pode estar se perguntando por que não habilitamos / desabilitamos o **yClock** no procedimento **JumpBird**. É porque quando estamos no estado do jogo, não pretendemos desativar o **yClock**. O pássaro está constantemente se movendo para cima ou para baixo. Também procedimento **JumpBird** como o próprio nome indica não deve fazer a ave cair também. Estamos perto. O pássaro pula quando o jogador bate e cai se o jogador não tocar. Agora precisamos lidar com o que acontece quando o nosso pássaro atinge a margem esquerda ou direita.



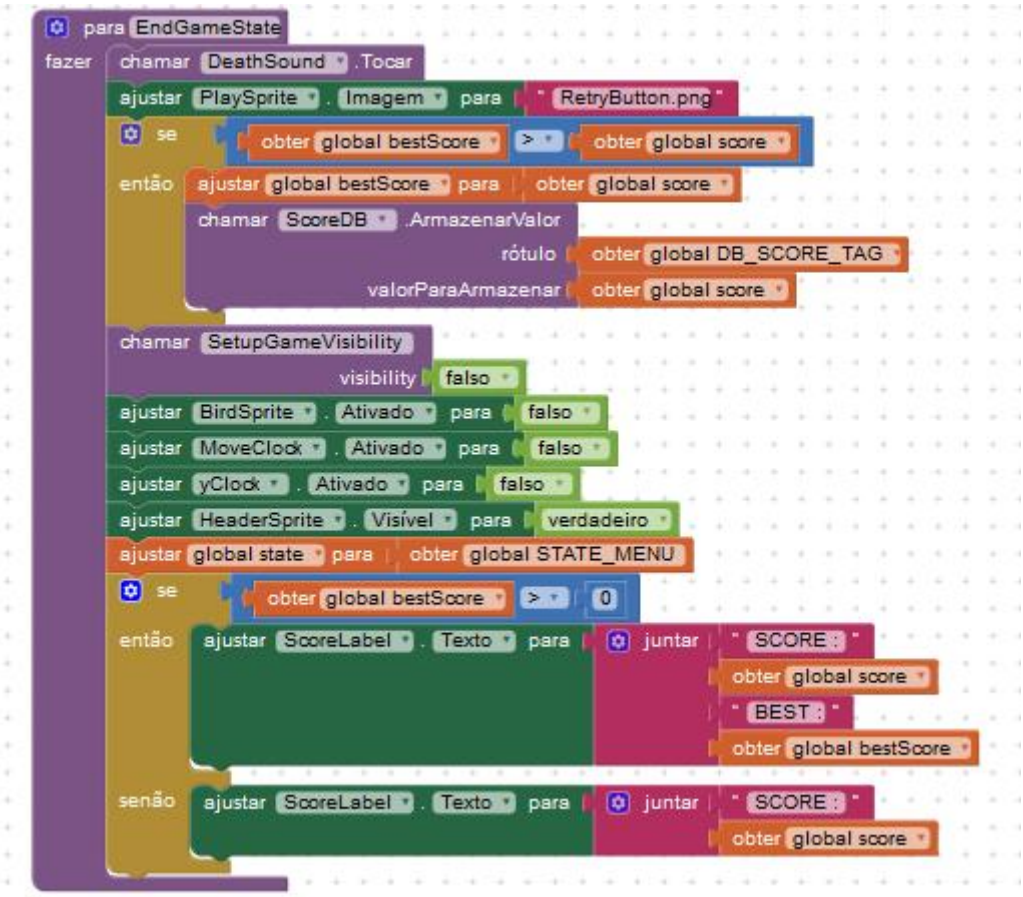

Sempre que o pássaro alcança a margem esquerda ou direita, o jogador faz uma pontuação. Então nós tocamos som de partitura. Nós chamamos **SetupPlacements** para colocar 4 obstáculos no outro lado. Nós explicamos o que este procedimento faz um tempo atrás. Adicionamos **1** à pontuação e atualizamos o marcador de pontuação para exibir a nova pontuação. Infelizmente nós temos que fazer com que a sua fofura morra se

ela bater em qualquer um dos 6 pontos (top, bottom e 4 obstáculos). Para fazer isso, precisamos verificar se está colidindo com qualquer um deles.




Vamos ver o procedimento **EndGameState** em um pouco. Por enquanto apenas pense, é o fim do jogo. Se o pássaro colidir com o pico superior ou inferior, fazemos com que ele morra imediatamente. Para os picos de obstáculos, damos um pouco de alavancagem. Nós chamamos o procedimento **isDead**. Se o pássaro colide com qualquer um dos quatro obstáculos, nós verificamos primeiro se ele está se movendo para a esquerda e não muito longe (menos que o meio da tela), e então vemos se ele cruzou o obstáculo no meio, se o fizermos, viva. Depois que é meio caminho e depois colide com qualquer outro acima ou abaixo, nós não o matamos também. Aumentamos a velocidade para atingir a borda mais rapidamente, para que não continue colidindo. É por isso que tivemos que redefinir a velocidade para **5** em **SetupPlacements** procedimento. Nós fazemos o mesmo

pela borda direita. **IsDead** retorna **verdadeiro** se precisarmos terminar o jogo, caso contrário, retorna **falso**.




Infelizmente, o pássaro morreu. Deixe o mundo saber. Jogue o som da morte. Precisamos mostrar a tela do menu. Desta vez vamos mudar a imagem do **PlaySprite** para repetir a imagem (**RetryButton.png**). Se a pontuação atual for **maior** que a melhor pontuação, se houver, definiremos o **bestScore como** a **pontuação** atual. Em seguida, salvamos no banco de dados. Nós escondemos todos os componentes do jogo. Nós desativamos o que não é visível. Nós mudamos o estado. Se não jogamos antes ou não marcamos **1**, não temos uma pontuação melhor. Nesse caso, mostramos apenas a pontuação atual. Caso contrário, mostramos os dois, lado a lado.