Neste guião irás aprender a **criar** um **jogo simples** com **movimento** e **colisões.**

Vamos **utilizar** tudo o que **aprendemos** até **agora** como **variáveis globais**, **funções**, **sub-rotinas** ou mesmo **estruturas**.

Começa por **descarregar** o **pack multimédia** necessário para este guião aqui e **extrai** para a **pasta QB64**.

Parte 1: Definir ecrã e fazer aparecer a nave

Guarda o teu programa como **space.bas**

O primeiro passo é **definir** o **ecrã**. Neste caso **define duas constantes** de nome **larguraEcra** e **alturaEcra** com os valores **1024** e **768** respetivamente. **Cria** um **ecrã** com **Screen _NewImage** com esta **resolução** e **32 bit** de cor.

De seguida **cria** uma **estrutura** de nome **naveType** com os seguintes **atributos**:

```
posX As Integer
posY As Integer
velocidade As Integer
imagem As Long
largura As Integer
altura As Integer
```

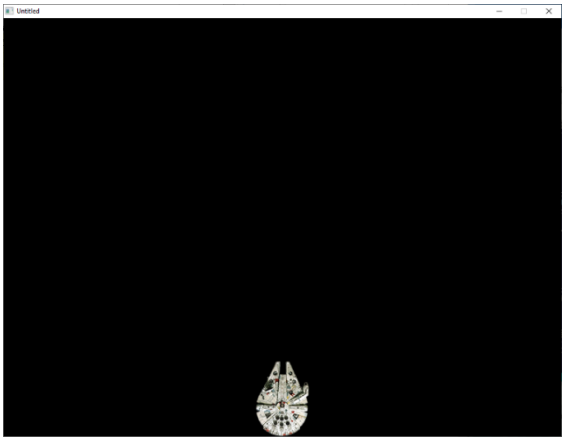
Aqui podes ver os **atributos** do nosso **player principal**, neste caso uma **nave**. Iremos **guardar informações** sobre a **posição** no **eixo** do **XX** da **nave** e também no **eixo** dos **YY**. Guardamos ainda a **velocidade**, i.e. como se **desloca** a **nave** no **ecrã**, mais **rápido** ou mais **lento**. Finalmente, a **imagem** da **nave**, que foi **fornecida** no **pack multimédia** e ainda a sua respetiva **largura** e **altura**.

Cria um **alias** desta estrutura, como **variável global**, de nome **nave**.

Vamos agora **atribuir valores** aos **componentes** da **estrutura**:

```
nave.velocidade = 10 'quanto maior mais rapido se move
nave.imagem = _LoadImage("imagens/mFalcon.png") 'nave
nave.largura = _Width(nave.imagem)
nave.altura = _Height(nave.imagem)
nave.posX = (larguraEcra - nave.largura) / 2 'posicao inical da nave centro do ecra
nave.posY = (alturaEcra - nave.altura) ' posicao inicial da nave na parte de baixo do ecra
```

Como podes verificar este **código** permite **definir** as **condições iniciais** da **nave**. Ela terá uma **velocidade** de **movimento** de **10** (mas ainda não temos código para a mexer, calma!). De seguida carregamos a **imagem** da **pasta imagens** e guardamos a **largura** e **altura** da **imagem**. Aproveitamos os dados anteriores para **centrar** a **nave** no eixo dos **XX** e posicioná-la na **parte** de **baixo** do ecrã no eixo dos **YY**.



Utiliza a função **_PutImage** para verificares se a **nave aparece** no teu **ecrã** como a do lado.

Parte 2: Movimento da nave

Nesta parte vamos fazer a **nave mover-se** para a **esquerda** e **direita** utilizando as **teclas direcionais**.

Cria uma **estrutura** de nome **teclasType** com os seguintes **atributos**:

```
esquerda As String
direita As String
```

Cria um **alias** de nome **teclas** desta **estrutura**.

Atribui os seguintes **valores** aos **componentes** da **estrutura**:

```
teclas.esquerda = Chr$(0) + Chr$(75) 'seta esquerda
teclas.direita = Chr$(0) + Chr$(77) 'seta direita
```

Lembra-te que definir as teclas assim é uma **mais valia** porque podes **alterar** as **teclas** na **estrutura** e todo o **jogo** passa a **funcionar** com as **novas teclas**.

Bem, para já as **teclas não funcionam**. Para isso **cria** uma **sub-rotina** de nome **moveNave**.

Dentro da **sub-rotina** temos de ter algo que, ao clicar na **tecla** da **esquerda** ou **direita** a nave **mude** de **lugar** no **ecrã**, neste caso **apenas** no eixo dos **XX**. Fica o **código** para **mexer** a **nave** para a **esquerda**:

```
If _KeyDown(CVI(teclas.esquerda)) Then
    nave.posX = nave.posX - nave.velocidade
```

Esta é a forma mais rápida **(_KeyDown)** de **mover** uma **imagem** no ecrã em **QB64**. O código **CVI** serve para **converter** a **string** das **teclas** em **número**.

Achas que **consegues** fazer o **código** para **mover-se** para a **direita**?

No **final** da **sub-rotina** tens de utilizar o **_PutImage** para **atualizar aposição** da **nave** no **ecrã**.

Se correres o programa ele **não** irá **funcionar** porque, já sabes que o computador tem de estar **constantemente** a **verificar** se **pressionamos** as **teclas**. Assim, antes da **sub-rotina** cria um **loop**:

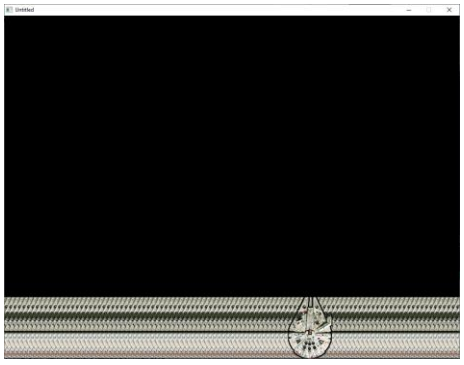
```
Do
    (código a ser repetido aqui)
loop until INKEY$="q"
```

Dentro do **loop** chama a **sub-rotina** que criaste para o **movimento** da **nave**.

Testa o **programa** e **verifica** que a **nave** se **move** para a **esquerda** e para a **direita**.

Bugou?? Pois… o **QB64** não se dá **muito bem** com **objetos** em **movimento**, mas não te **preocupes** que **há** um **fix**. **Aplica** o seguinte **código** ao **teu programa** e **verás** o problema **resolvido**:

```
Do
    _Limit 100
    PCopy _Display, 1
    moveNave
    _Display
    PCopy 1, _Display
Loop Until InKey$ = "q"
```



Explicação: Estas linhas de código **permitem manter** uma **cópia** do **ecrã visível,** entre o **anterior ecrã** e o **seguinte** evitando **arrastamento** e o **piscar** do **ecrã** (o _Limit 100 é o tempo que leva o ciclo DO a repetir. Podes alterar o valor e ver o que faz).

Testa de novo. Já funcionou?

Aproveita para **introduzir** a **imagem** de **fundo** disponibilizada no **pack**. Faz isso a **seguir** à **definição** do **ecrã**. A imagem tem a **mesma resolução** do **ecrã** logo deve ser **colocada** na coordenada **(0, 0)**

PRO: Melhoramentos: ao **atingir** os **limites** do **ecrã não deixar** ir **mais**.

Parte 3: Tiros

Nesta parte vamos possibilitar a **nave dar tiros**.

Cria uma **estrutura** de nome **tirosType** com os **seguintes atributos**:

```
posX As Integer
posY As Integer
velocidade As Integer
imagem As Long
largura As Integer
altura As Integer
estadoTiro As Integer
```

Como podes **verificar** a **estrutura** é muito **idêntica** à da **nave**. Todavia tem um **último atributo** de nome **estadoTiro** que servirá, mais à frente para **não permitir** dar **mais** que **um tiro** (até este **sair** do **ecrã** ou **colidir** com um **inimigo**).

Cria um **alias** de nome **tiros** desta **estrutura**.

Atribui os seguintes **valores** aos **componentes** da **estrutura**:

```
tiros.velocidade = 10
tiros.imagem = _LoadImage("imagens/laserredp.png")
tiros.largura = _Width(tiros.imagem)
tiros.altura = _Height(tiros.imagem)
tiros.estadoTiro = 0
```

A **posX** e **posY não** foram **definidas** porque o **tiro não aparece** no **ecrã** quando **entramos** no **jogo**, **mas** sim quando **pressionamos** uma **tecla (neste caso será a tecla espaço)**.

Ao **contrário** da **nave**, o **tiro**, só se irá **mover-se** no **eixo** dos **YY** quando pressionarmos a tecla **espaço**. Começa por **definir** uma **variável** de nome **shot** do **tipo integer** dentro da **estrutura teclas**.

Atribui o valor **32** a **teclas.shot**

Cria uma **sub-rotina** de nome **tiro** e nessa **sub-rotina** testa **se** a **tecla espaço** foi **pressionada**.

Dentro desse **IF** estabelece a **posição inicial** do **tiro**. Se pensares, vês que a posição inicial do **tiro** tem de ser a parte da **frente** da **nave**:

```
tiros.posY = (alturaEcra - nave.altura) ' posicao inicial do tiro na parte de cima da nave
```

Porém a posição **inicial** do **tiro** no **eixo** dos **XX** é uma **incógnita**, porque a **nave** está **sempre** a **mover-se** e nós **não** podemos **dizer** que o **tiro** sai **sempre** de um **sítio fixo**. De **onde** terá que **sair** o **tiro**? Da **posição X** onde a **nave** estiver naquele momento:

```
tiros.posX = nave.posX + nave.largura / 2 'o tiro tem de sair da posiao da nave quando se faz o disparo
```

Apenas foi **adicionado metade** da **largura** da **nave** para o **tiro sair** do **centro** da **nave**.

Finalmente, antes de terminar a **sub-rotina** deves fazer a **aparecer** a **imagem** com **_PutImage**

Se **correres** o **programa** o **tiro não sai**. Tens de **pensar** que **depois** de **sair** ele tem de **percorrer** o **ecrã todo** na **vertical**. Bem, para isso necessitamos de uma **flag** para saber o **estado** do **tiro**. Essa **flag** já foi **criada dentro** da **estrutura** e **inicializada** a **zero**. Para **distinguir** o **início** do **movimento** entre **quando pressionas** a **tecla** e o **movimento** do **tiro** pelo **ecrã** a variável tem de **mudar** de **valor**:

Tiros.estadoTiro=0 – é **permitido** disparar

Tiros.estadoTiro=1 – **não** é **permitido** disparar porque o tiro ainda está a percorrer o ecrã.

Altera as **condições** do teu **IF** para **estas**:

```
If _KeyDown(teclas.shot) And tiros.estadoTiro = 0 Then
```

Adiciona ao **interior** do **IF**, na **última linha** o seguinte código:

```
tiros.estadoTiro = 1
```

Esta linha de código **permite mudar** o **estado** de **inicial** de **0** para **1** indicando que agora é hora do **tiro percorrer** o **ecrã**. Para isso, deves **adicionar** um **ELSEIF** para o caso do **estado** do tiro ser **1**:

```
ElseIf tiros.estadoTiro = 1 Then
    tiros.posY = tiros.posY - tiros.velocidade ' menos porque decresce para cima
    _PutImage (tiros.posX, tiros.posY), tiros.imagem
    If (tiros.posY < 0) Then 'se bala sair do ecra   ou tocar no inimigo
        tiros.estadoTiro = 0
    End If
End If
```

No código acima encontra-se o caso do estado do tiro ser **1**. Este é o código **necessário** para o **tiro percorrer** o **ecrã**. Isso é **conseguido** com as **linhas**:

```
tiros.posY = tiros.posY - tiros.velocidade ' menos porque decresce para cima
_PutImage (tiros.posX, tiros.posY), tiros.imagem
```

Como o ecrã **decresce** nos **YY** quando o **tiro vai** para **cima**, então quando **sair** do **ecrã** (**posição** do **tiro** for **menor** que **zero**) então **voltamos** a **estabelecer** o **estado** do **tiro** a **0** para **podermos voltar** a **disparar**. Enquanto isto não acontece, o **tiros.PosY** é cada vez **menor** por **cada ciclo DO** que chama a **sub-rotina**.

Bem este código **não** irá **funcionar** até que a **sub-rotina** seja **chamada** no **loop principal**:

```
Do
    _Limit 100
    PCopy _Display, 1
    moveNave
    tiro
    _Display
    PCopy 1, _Display
Loop Until InKey$ = "q"
```

Testa o **programa** e **verifica** se **consegues disparar** nas **condições descritas anteriormente**.

PRO: **adiciona** o som **tiro.wav** ao programa para que se ouça **quando disparas**.

Parte 4: Inimigos

Nesta parte vamos fazer **aparecer inimigos** e **definir** o seu **movimento** no **ecrã**.

Cria uma **estrutura** de nome **inimigosType** com os seguintes **atributos**:

```
posX As Integer
posY As Integer
velocidade As Integer
imagem As Long
largura As Integer
altura As Integer
estadoInimigo As Integer
```

Como podes **verificar** a **estrutura** é muito **idêntica** ao da **nave**. Todavia tem um **último atributo** de nome **estadoInimigo** que **servirá**, mais à frente para **saber** se o **inimigo colidiu** com o **tiro** (ou chegou ao **fim** do **ecrã**).

Cria um **alias** de nome **inimigos** desta estrutura.

Atribui os seguintes **valores** aos componentes da **estrutura**:

```
inimigos.velocidade = 1
inimigos.imagem = _LoadImage("imagens/tiefighterp.png")
inimigos.largura = _Width(inimigos.imagem)
inimigos.altura = _Height(inimigos.imagem)
inimigos.estadoInimigo = 0 '1 vivo 0 morto
```

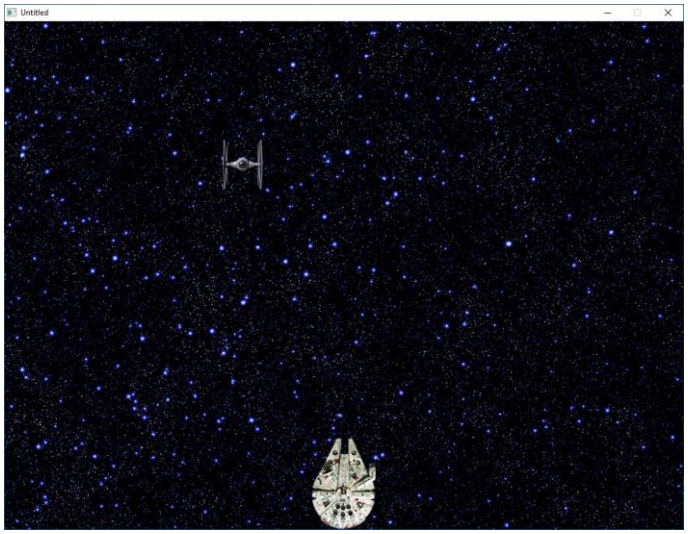
Temos de **criar** uma **animação** para o **inimigo**. Por exemplo fazê-lo **aparecer** sempre na **parte** de **cima** do **ecrã** em **locais diferentes** e **descer** até ao **fim** do **ecrã**. Para já **não** vamos pensar em **colisões** ou tiros. **Cria** uma **sub-rotina** de nome **moveInimigo**.

Nesta **sub-rotina** testa **se** o **estado** do **inimigo** é **zero** e caso isso aconteça quer dizer que está na **fase inicial** do **movimento**. **Define** a sua **posição Y** para **0 menos** a **altura** do **inimigo** para que apareça **fora** do **ecrã** (dar a sensação que vem do espaço).
De seguida **define** a sua **posição X** como um **aleatório** da **largura** do **ecrã**.
Finalmente **define** o **estado** do **inimigo** para **1** para começar o **movimento descendente**.

Cria um **ELSE**.
Neste **ELSE** vais fazer o **movimento descendente**. Faz o **mesmo** que fizeste no **tiro**, mas com a diferença que o tiro sobe o **inimigo desce**.
Dentro do **ELSE** cria um **IF** que teste **se** o **inimigo** chegou ao **fim** do **ecrã** e caso seja verdade muda o **estado** do **inimigo** de novo para **zero,** reiniciando-se assim todo o processo.

No **fim** da **sub-rotina** não te esqueças de fazer o **_PutImage** e **também** de **adicionar** ao **loop principal** esta nova **sub-rotina**.

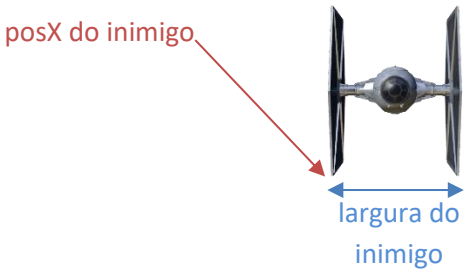
Testa o programa. O **inimigo** deve fazer um **movimento** de **cima** para **baixo** sempre de **locais diferentes** na parte superior (lembra-te de utilizar o **Randomize Timer**).



Parte 5: Colisões

Nesta parte vamos testar **colisões**. Neste caso **entre** o **tiro** e o **inimigo**.

Cria uma **sub-rotina** de nome **tiroAcertou** e nessa **sub-rotina** testa **se** o tiro **colidiu** com o **inimigo**. Parece **complexo,** não é? Principalmente porque **não podemos** tratar o **tiro** e o **inimigo** como **dois objetos** e perguntar se **colidiram**. A única **estratégia** ao nosso dispor são as **coordenadas**. Então pensa… se o **tiro estiver** no mesmo **X e Y** do **inimigo** há **colisão**.



Como podes verificar, a **posX** do **tiro** pode estar **entre** a **posX** do **inimigo** e a sua **largura**. Quanto à **posição** no eixo dos **YY** já vais **tratar** disso mais à **frente**.

Adiciona as seguintes **condições** ao **teu IF** dentro da sub-rotina **tiroAcertou**.

```
If tiros.posX >= inimigos.posX And tiros.posX <= inimigos.posX + inimigos.largura Then
```

Dentro do **IF** coloca a **zero** o **estado** do **inimigo** para que **reinicie** o seu **movimento**.

PRO: adiciona o som **hit.wav** quando **acerta** no **inimigo**.

Adiciona a sub-rotina **tiroAcertou** ao **loop principal** e testa o teu programa.

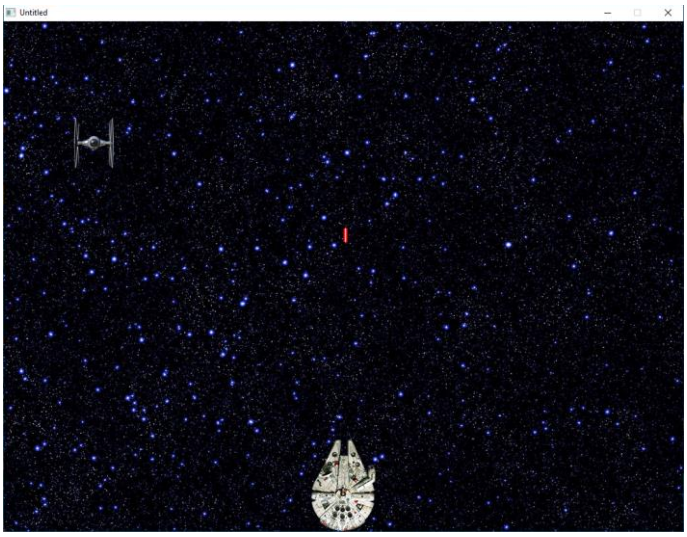
Ricardo Sérgio

O que **aconteceu** o tiro **mata** o **inimigo** logo quando é **disparado**? Pois aqui é que **entra** a parte do **eixo** dos **YY**. **Só** quando o **tiro** estiver no **local** do **inimigo** é que deve **parecer** que **lhe acertou**. Então é necessário **atribuir** mais uma **condição** ao **IF** que **verifica** se a **posição** do **Y** dos **tiros** é **menor** que a dos **inimigos** (menor porque o **Y decresce** para **cima**).

Experimenta e **verás** que já **funciona**.

Para já ainda **não consegues voltar** a **disparar** mal **matas** o **inimigo**. Para isso, **deves inicializar** o **estado** do **tiro** para que voltes a poder fazê-lo.

No **final** deverás ter algo **semelhante** à **imagem** de **baixo**:



Diverte-te a **jogar** e **desenvolve** o jogo para que fique **espetacular**. Ficam algumas ideias:

- Fazer **explodir** quando leva com o tiro.
- Fazer **Gameover** quando inimigo colide com a nossa nave ou um processo de **vidas**.
- Ter **música** ao longo de todo o jogo.
- Aumentar o **nº de inimigos**.
- Ter um **boss** level.
- Ir aumentando a **velocidade** de inimigos.
- Ter **pontuação**.
- Ter **splash** screen e **menu** screen.
- Ter **definições guardadas** num ficheiro de texto.

Guarda o teu **programa**.

Chama o teu **professor** para **avaliar.**