

Capítulo 1 – Entendendo a Interface	3
1. Conceitos da Interface do Blender	4
1.1. Teclado e mouse	4
1.2. O sistema de janelas	5
1.3. Tipos de Janelas	6
1.4. Contextos, Painéis e Botões	7
1.5. Toolbox	10
1.6. Telas	10
1.7. Cenas	11
2. Navegando no Espaço 3D	11
2.1. A direção de visão (rotação)	11
2.2. Movendo e ampliando a visão	12
2.3. Perspectiva e Projeção Ortográfica	12
2.4. Modo de desenho	14
2.5. Vista local	14
2.6. O sistema de layers	15
Capítulo 2 – Abóbora Saltitante	16
1. Modelando um ambiente	17
2. Anexando um objeto de outra cena	20
3. Liguem os motores!	23
4. Interatividade	24
4.1. Mais Controle	27
5. Controle da câmera	27
6. Luz em tempo real	28
7. Animação de objetos	29
8. Refinando a cena	30
9. Adicionando som à nossa cena	31
Capítulo 3 – Lógica e LogicBricks	33
1. Mas é lógico!	34
1.1. Entra em cena a lógica booleana	34
1.2. LogicBrick: no pasa de un elemento booleano	35
2. Tá bom, não é tão lógico assim	36
2.1. Complicando um pouco a situação	38
3. Algumas tabelas-verdade	43
4. Conclusão	43
Capítulo 4 – Criando o Esqueleto de um Jogo	44
1. Finite State Machine: outro nome para liquidificador	45
2. A estrutura global de um jogo é uma FSM!	46
3. E liguem o liquidificador! ...quer dizer, o Blender!	47
3.1. Preparando as superfícies para receber texto	48
3.2. Mapeando o texto nas superfícies	48
3.3. Propriedades: bananas são amarelas e marcianos são verdes	51
3.4. A propriedade Text	52
3.5. Ligando os estados da FSM	53
3.6. Fazendo os itens do menu funcionarem	54
4. Conclusão	55
Capítulo 5 – Um Joguinho de Nave Básico	56
1. Preparativos	57
1.1. Premissa e Objetivo do Jogo	57
1.2. Um Jogador versus Jogo	57

1.3. O Campo	57
1.4. Recursos.....	57
1.5. Conflitos	57
1.6. Processos	57
1.7. Resultado	58
1.8. Regras.....	58
2. Preparando o Ambiente	59
2.1. Texturizando os cilindros.....	59
2.2. Acertando a extensão do mapeamento	62
2.3. Ei, não aponta esse dedo pra mim!.....	66
2.4. Mas... e o outro cilindro?.....	67
3. Naves, Emissor de Naves e Tiros	69
3.1. As naves e os tiros.....	69
3.2. O emissor de naves	69
4. Posicionando os elementos de overlay do jogo	71
4.1. Ícones.....	71
4.2. Marcadores de vida, de cidades e de pontuação	71
5. Senhoras e senhores, dêem partida nos motores!	72
5.1. Gira gira gira... Rotacionando os cilindros	72
5.2. Movimentando a nave	73
5.3. Disparando as naves inimigas.....	79
5.4. Chumbo neles!	84
5.5. Mayday, mayday! Dez maneiras de se destruir a sua própria nave.....	86
5.6. Você é a última fronteira entre eles e a cidade.....	88
5.7. Alterando os marcadores	90
5.8. Game Over!.....	93
5.9. E não é que temos um menu de inicialização?	94
5.10. Ah, antes que eu me esqueça.....	94
6. Conclusão.....	96
Capítulo 6 – Um Exemplo de Aplicação Walkthrough.....	97
1. Anexando o template na sua cena.....	98
2. Importando uma cena de outro aplicativo	102
3. Exportando um arquivo auto-executável.....	102
4. Conclusão.....	103

Capítulo 1

Entendendo a Interface

Se você está começando a usar o Blender agora, seria importante adquirir um bom conhecimento da filosofia de trabalho da sua interface antes de começar a trabalhar com ele. Os conceitos por trás da sua interface não são conceitos-padrão, sendo, portanto, diferentes de outros pacotes 3D. Usuários de Windows, principalmente, precisarão se acostumar à forma que o Blender manuseia controles, tais como escolhas de botões e movimentos do mouse. Mas, na verdade, essa diferença é uma de suas grandes forças: uma vez entendido como trabalhar no estilo Blender, você descobrirá que pode trabalhar de forma extremamente rápida e produtiva.

Além disso, a sua interface foi consideravelmente modificada na transição da versão 2.28 para a versão 2.3 – então, mesmo usuários experientes do Blender poderão se beneficiar deste capítulo.

1. Conceitos da Interface do Blender

A interface de usuário é o veículo de interação entre o usuário e o programa. O usuário se comunica com o programa via teclado e mouse, o programa emite respostas via tela e seu sistema de janelas.

1.1. Teclado e mouse

A interface do Blender utiliza os três botões do mouse mais uma miríade de teclas de atalho (para uma discussão completa e aprofundada deste tópico, consulte a seção XXX). Se o seu mouse tem somente dois botões, você pode emular o botão do meio (a seção XXX descreve como). O mouse wheel pode ser usado, mas não é necessário, já que existem atalhos apropriados às suas funções.

Este livro usa as seguintes convenções para descrever as ações de usuário:

- Os botões do mouse são chamados **BEM** (botão esquerdo do mouse), **BMM** (Botão do Meio do Mouse) e **BDM** (botão direito do mouse).
- Se o seu mouse tem um wheel, **BMM** refere-se a clicar o wheel como se ele fosse um botão, enquanto **MW** significa rolá-lo.
- Letras das teclas de atalho são nomeadas adicionando-se o sufixo *KEY* à letra, p.ex. **GKEY** refere-se à tecla *g* do teclado. Teclas podem ser combinadas com os modificadores **SHIFT**, **CTRL** e/ou **ALT**. Para teclas modificadas o sufixo *KEY* geralmente será descartado, p.ex. **CTRL-W** ou **SHIFT-ALT-A**.
- De **NUM0** a **NUM9**, **NUM+** e assim por diante referem-se às teclas do teclado numérico, localizado no lado direito do teclado (o NumLock geralmente deverá estar ativado).
- Outras teclas são referenciadas pelos seus nomes, tais como **ESC**, **TAB** e de **F1** a **F12**.
- Outras teclas especiais dignas de nota são as setas direcionais **SETACIMA**, **SETABAIXO**, **SETAESQUERDA** e **SETADIREITA**.

Como o Blender faz uso extensivo do mouse e do teclado, uma “regra de ouro” emergiu dentre os usuários do Blender: mantenha uma mão no mouse e a outra no teclado! As teclas utilizadas mais frequentemente estão agrupadas de uma forma tal que podem ser alcançadas pela mão esquerda em posição padrão (ou seja, com o dedo indicador na **FKEY**) no layout de teclado padrão. Essa regra assume que você utiliza o mouse com a sua mão direita.

1.2. O sistema de janelas

Agora é hora de iniciar o Blender e começar a brincar com ele...

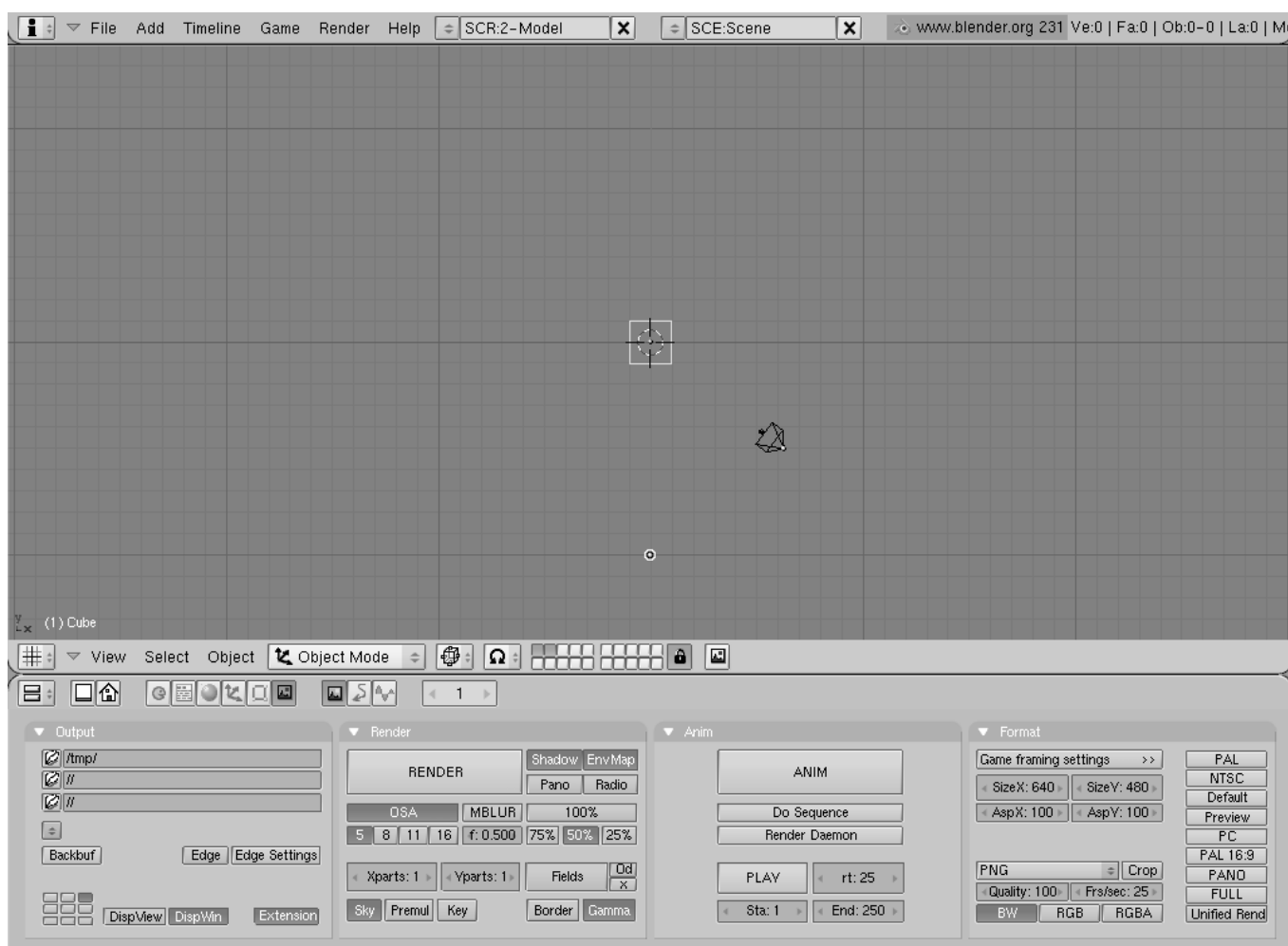


Figura 1. A cena padrão do Blender.

A **Figura 1** mostra a tela que você deve ver após iniciar o Blender. Por padrão ela é separada em três janelas: o menu principal no topo, a grande *3DView* (Vista 3D) no centro e a *ButtonsWindow* (Janela de Botões) na base. A maioria das janelas possui um cabeçalho (a faixa de cor cinza-claro contendo ícones de botões – por essa razão, também iremos nos referir ao cabeçalho como a *Toolbar* [Barra de Ferramentas] da janela); se presente, o cabeçalho pode estar no topo (tal como na *ButtonsWindow*) ou na base (como na *3DView*) da área de uma janela.

Se você mover o mouse sobre uma janela, notará que o seu cabeçalho muda para um tom mais claro de cinza. Isso significa que a janela está “focada”; todas as teclas de atalho que você pressionar agora irão afetar o conteúdo desta janela.

Você pode customizar facilmente o sistema de janelas do Blender para se ajustar às suas necessidades e aos seus desejos. Você pode criar uma nova janela dividindo uma janela existente; para tanto, focalize a janela que você quer dividir (mova o mouse sobre ela), clique na borda com **BMM** ou **BDM** e selecione *Split Area* (**Figura 2**). Agora, você pode confirmar a posição da nova borda clicando com **BEM** ou cancelar a sua ação pressionando **ESC**. A nova janela surgirá como um clone daquela que você dividiu, mas ela pode ser configurada para mostrar um tipo diferente de janela ou para mostrar a cena 3D de um ponto de vista diferente.

Dica – Itens da interface

Rótulos nos botões de interface, itens de menus e, no geral, qualquer texto mostrado na tela será referenciado neste livro numa fonte diferente, desta forma.

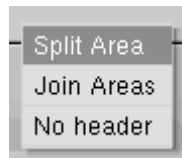


Figura 2. O menu **Split** de criação de novas janelas.

Crie uma nova borda vertical escolhendo **Split Area** em uma borda horizontal e vice-versa. Você pode redimensionar uma janela arrastando uma de suas bordas com **BEM**. Para reduzir o número de janelas, clique numa borda entre duas janelas com **BMM** ou **BDM** e escolha **Join Areas**. A janela resultante irá receber as propriedades da janela que estava previamente focalizada.

Para configurar a posição de um cabeçalho clique **BDM** nele e escolha **Top** ou **Bottom**. Você também pode esconder o cabeçalho escolhendo **No Header**, mas isso só é recomendado se você já conhece todas as teclas de atalho relevantes. Você pode mostrar um cabeçalho novamente clicando em uma das bordas da janela com **BMM** ou **BDM** e escolhendo **Add Header**.

1.3. Tipos de Janelas

A área de cada janela poderá conter diferentes tipos e conjuntos de informação, dependendo no que você estiver trabalhando. Tais informações podem incluir modelos 3D, animação, materiais de superfície, scripts em Python e daí por diante. Você pode selecionar o tipo de cada janela clicando no botão mais à esquerda do seu cabeçalho com **BEM** (**Figura 3**).



Figura 3. O menu de seleção do tipo de janela.

Nós iremos explicar as funções e a utilização dos respectivos tipos de janela posteriormente neste livro. Por enquanto nós só precisaremos nos preocupar com os três tipos de janelas que são apresentados na cena padrão do Blender:

3DView. Fornece uma visão gráfica da cena 3D em que você está trabalhando. Você pode visualizar sua cena de qualquer ângulo com uma variedade de opções; veja a seção XXX para mais detalhes. Ter várias 3DViews na mesma tela pode ser útil se você quiser observar suas modificações de diferentes perspectivas ao mesmo tempo.

ButtonsWindow. Contém a maioria das ferramentas para edição de objetos, superfícies, texturas, luzes e muito mais. Você irá precisar desta janela constantemente, caso ainda não conheça todas as teclas de atalho de cor. De fato, você poderá querer mais de uma dessas janelas abertas ao mesmo tempo, cada uma mostrando um conjunto de botões diferente.

UserPreferences (Preferências do Usuário). Essa janela normalmente fica escondida, mostrando apenas o seu menu – veja a seção XXX para mais detalhes. Ela é usada raramente, já que contém apenas opções de configuração global.

Nota

Como este livro está voltado à criação de conteúdo em tempo real no Blender, somente as janelas relevantes ao nosso trabalho serão explicadas.

Há várias novidades no Blender 2.30 e nas suas versões posteriores. Em primeiro lugar, todos os cabeçalhos ficaram mais legíveis e menos poluídos de botões.

A maioria dos cabeçalhos, imediatamente à direita do botão de seleção do tipo de janela, agora exibe uma série de menus; essa é uma das principais características da interface das versões 2.30 e posteriores. Agora, menus permitem não somente o acesso direto a várias funções e comandos que antes só eram acessíveis via teclas de atalho ou botões secretos. Menus podem ser ocultos e mostrados via o botão triangular à esquerda deles.

Menus não são somente sensíveis à janela (eles mudam conforme o tipo de janela), mas também sensíveis ao contexto (mudando conforme o objeto selecionado), de forma que ficam sempre compactos, apresentando somente as ações que podem ser realizadas num determinado momento.

Todos os itens dos menus mostram as suas respectivas teclas de atalho, caso haja alguma. O trabalho no Blender flui de maneira ideal quando as teclas de atalho são utilizadas – portanto, na sua maior parte, o resto deste livro irá usar as teclas de atalho (os respectivos itens de menus serão apenas apresentados na primeira vez em que o comando surgir). De qualquer forma, os menus não deixam de ser preciosos, pois dão uma visão completa de todas as ferramentas e comandos que o Blender oferece.

Um atributo das janelas, que normalmente é útil para se conseguir uma edição precisa, é a maximização da sua área. Se você utilizar o item **Maximize Window** do menu **View** ou a tecla de atalho **CTRL-SETABAIXO**, a janela focalizada estenderá a sua área de forma a preencher toda a tela. Para retornar ao tamanho normal, acesse o item **Tile Window** do menu **View** ou **CTRL-SETACIMA**.

Nota

Daqui pra frente, para mostrar qual item de qual menu deve ser acessado, será utilizado o formato **Menu>>Item**, p.ex., **View>>Maximize Window**.





1.4. Contextos, Painéis e Botões

Os botões do Blender são mais instigantes do que aqueles existentes na maioria das outras interfaces de usuário – e eles ficaram ainda melhores na versão 2.30, devido principalmente ao fato deles serem vetoriais e desenhados em OpenGL, o que os torna elegantes e ampliáveis.

Os botões, na sua maioria, estão agrupados na **ButtonsWindow**. A partir do Blender 2.3, a **ButtonsWindow** passou a apresentar seis contextos principais, que podem ser escolhidos a partir da primeira fileira de ícones do cabeçalho (**Figura 4**, conjunto de botões à esquerda), cada qual podendo ser subdividido num número variável de subcontextos, selecionáveis a partir da segunda fileira de ícones (**Figura 4**, conjunto de botões à direita):



Figura 4. Contextos e Subcontextos.

-  **Logic (F4)**
-  **Script (sem atalho)**
-  **Shading (F5)**
 -  **Lamp buttons (sem atalho)**

-  Material buttons (sem atalho)
-  Texture buttons (**F6**)
-  Radiosity buttons (sem atalho)
-  World buttons (**F8**)
-  Object (**F7**)
-  Editing (**F9**)
-  Scene (**F10**)
 -  Render buttons (sem atalho)
 -  Anim/playback buttons (sem atalho)
 -  Sound block buttons (sem atalho)

Uma vez que o contexto é selecionado pelo usuário, o subcontexto normalmente é determinado pelo Blender baseado no objeto ativo. Por exemplo, dentro do contexto Shading, se um ObjetoLamp é selecionado, o subcontexto Lamp é apresentado; se um ObjetoMesh ou outro objeto renderizável é selecionado, então Material Buttons se torna o subcontexto ativo; e, se um ObjetoCâmera é selecionado, o subcontexto ativo será World.

Nota

Mesh (malha, em português) é o conjunto de vértices e de ligações entre vértices que forma os polígonos de um modelo 3D.

A mais notável das novidades na interface provavelmente é a presença de *Painéis* para agrupar os botões logicamente. Cada painel possui o mesmo tamanho e pode ter a sua posição mudada dentro da área da ButtonsWindow clicando e arrastando o seu cabeçalho com **BEM**. Os painéis podem ser alinhados clicando **BDM** na JanelaBotões e escolhendo o layout desejado a partir do menu que aparecerá (**Figura 5**).



Figura 5. Menu Align Buttons.

MW rola os painéis na direção do seu alinhamento, **CTRL-MW** e **CTRL-BMM** ampliam ou diminuem os painéis. Os painéis podem ser ocultos/expandidos individualmente, clicando com **BEM** no triângulo à esquerda de seu cabeçalho.

Painéis mais complexos são organizados em guias. Clicando **BEM** em uma guia no cabeçalho do painel muda os botões apresentados (**Figura 6**). Guias podem ser arrancadas de um painel para formar painéis independentes clicando **BEM** no seu cabeçalho e arrastando-a para fora. De forma semelhante, painéis independentes podem ser transformados em um painel único com guias, arrastando e soltando o cabeçalho de um painel sobre outro cabeçalho.

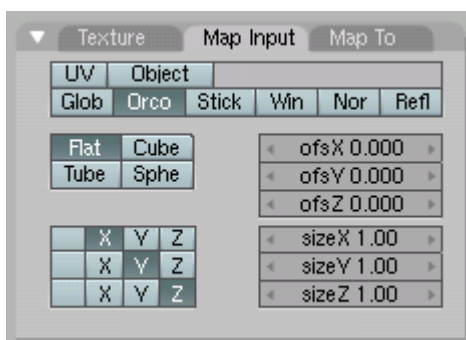


Figura 6. Painel com Guias.

Por último, existem vários tipos de botões dispostos dentro das guias dos painéis:

Botões de Operação. São botões que realizam uma operação quando são clicados (com **BEM**, igual a todos os botões). Eles podem ser identificados pela sua cor marrom no esquema de cores padrão do Blender (**Figura 7**).



Figura 7. Um botão de operação.

Botões de Alternância. Botões de alternância vêm em várias cores e formatos (**Figura 8**). As cores verde, violeta e cinza não alteram a funcionalidade – elas são apenas auxiliares visuais para reconhecer agrupamentos de botões e conteúdos da interface mais rapidamente. Clicar neste tipo de botão não realiza nenhuma operação, apenas alternando um estado entre “liga” e “desliga”.

Alguns botões possuem um terceiro estado que é identificado pelo texto do botão que fica amarelo (o botão **Emit** da **Figura 8**). Normalmente o terceiro estado significa “negativo”, e o estado que seria o “liga” passa a significar “positivo”.



Figura 8. Botões de alternância.

Botões de Seleção. Botões de seleção são grupos de botões de alternância mutuamente exclusivos, ou seja, não mais do que um botão de seleção em um determinado grupo pode estar “ligado” em um dado momento.

Botões Numéricos. Botões numéricos (**Figura 9**) podem ser identificados pelo rótulo: a sua descrição é seguida de dois pontos e um número. Botões numéricos podem ser manuseados de várias formas:

- Para aumentar seu valor, clique com **BEM** no canto direito do botão, onde existe um pequeno triângulo; para diminuí-lo, clique no canto esquerdo do botão, onde outro triângulo é mostrado.
- Para alterar o valor de forma mais abrangente, segure **BEM** e arraste o mouse para a esquerda ou para a direita. Se você segurar **CTRL** enquanto fizer isso, o valor será alterado em passos discretos, ou seja, espaçados; se você segurar **SHIFT**, você terá um controle mais preciso sobre os valores. **ENTER** pode ser usado no lugar de **BEM** aqui (somente enquanto o cursor do mouse estiver sobre o botão).

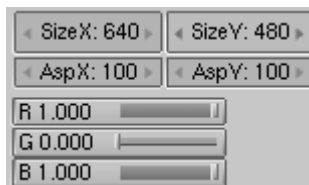


Figura 9. Botões numéricos.

- Você pode inserir um valor a partir do teclado segurando **SHIFT** e clicando **BEM**. Pressione **SHIFT-BACKSPACE** para limpar o valor atual; **SHIFT-SETAESQUERDA** para mover o cursor para o início do valor; **SHIFT-SETADIREITA** para mover o cursor para o fim; **ENTER** para confirmar o valor. Pressione **ESC** para restaurar o valor original.
- Alguns botões numéricos possuem uma barra de rolagem, ao invés de apresentarem apenas um valor com triângulos nas laterais. O mesmo método de operação se aplica a eles, exceto que cliques simples com **BEM** devem ser realizados no canto esquerdo ou direito da barra, enquanto que clicar no rótulo ou no valor do botão automaticamente o levará ao modo de inserção por teclado.

Botões de Menu. Use os botões de menu para fazer seleções dentro de listas criadas dinamicamente. Botões de menu são usados principalmente para conectar BlocosDados entre si (BlocosDados são estruturas tais como Meshes, Materiais, Texturas e assim por diante; ao conectar um ObjetoMaterial a um ObjetoMesh, você está aplicando o ObjetoMaterial ao ObjetoMesh). Você verá um exemplo para tal bloco de botões na **Figura 10**. O primeiro botão (com dois pequenos triângulos, um apontando para cima e outro para baixo) abre um menu que lhe permite selecionar o BlocoDados com o qual se deseja conectar segurando **BEM** e soltando-o sobre o item requerido. O segundo botão mostra o tipo e o nome do BlocoDados conectado e lhe permite editar o seu nome após clicar **BEM**. O botão “X” desfaz a conexão, o botão com o desenho de um carro gera um nome automático para o BlocoDados e o botão “F” especifica se o BlocoDados deve ser salvo no arquivo mesmo se ele não estiver sendo utilizado (desconectado).

Dica – Objetos Desconectados

Dados desconectados *não são perdidos* enquanto você não sair do Blender. Isso é um poderoso atributo de correção: se você excluir um objeto, o material atribuído a ele se torna desconectado, mas ainda está lá! Você terá apenas que reconectá-lo a um outro objeto ou ativar o botão “F”.



Figura 10. Botões de conexão de BlocosDados.

1.5. Toolbox

Pressionando **BARRAESPAÇO** em uma 3DView ou segurando **BEM** ou **BDM** com o mouse parado por mais de meio segundo abrirá a *Toolbox* (Caixa de ferramentas). Ela contém seis contextos principais, dispostos em duas linhas, cada qual abrindo menus e submenus.

Três destes contextos abrem os mesmos três menus apresentados no cabeçalho da 3DView; dos outros três, Add permite adicionar novos Objetos à cena enquanto Edit e Transform mostram todas as operações permitidas no(s) Objeto(s) selecionado(s) (**Figura 11**).

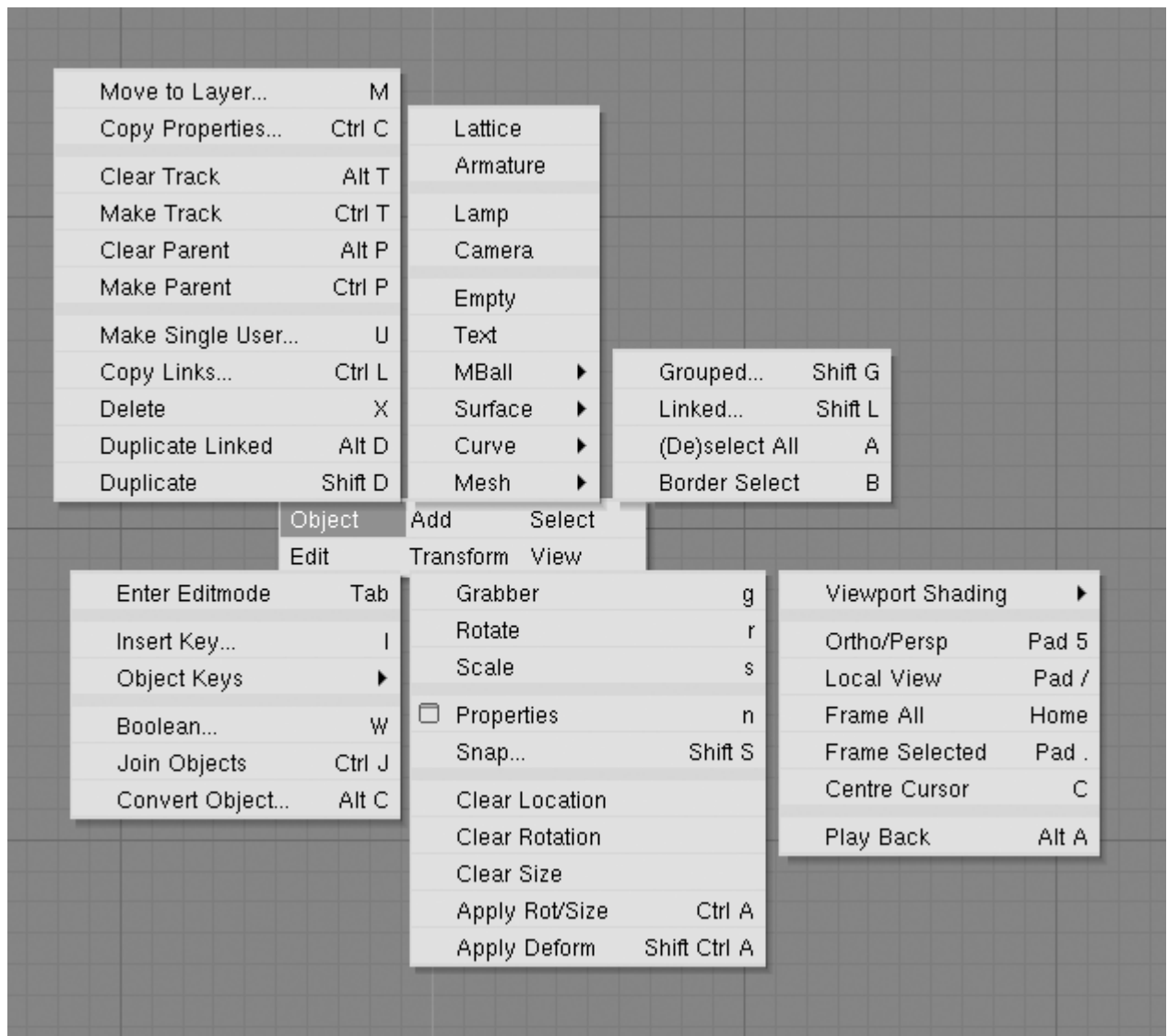


Figura 11. A Toolbox.

1.6. Telas

A flexibilidade do Blender com relação a janelas permite a criação de ambientes de trabalho personalizados para diferentes tarefas, tais como modelagem, animação e criação de scripts. Normalmente, é útil poder

alternar rapidamente entre diferentes ambientes dentro de um mesmo arquivo. Isso é possível através da criação de várias telas: todas as mudanças de e nas janelas, descritas nas seções XXX e XXX, são salvas dentro de uma tela, de forma que se você modificar suas janela em uma tela, as outras telas não serão afetadas, mas a cena em que você está trabalhando permanecerá a mesma em todas as telas.

O Blender é iniciado com três telas-padrão diferentes; elas estão disponíveis através do botão de menu SCR que se encontra no cabeçalho da UserPreferences, mostrado na **Figura 12**. Para alternar para a próxima tela (em ordem alfabética), pressione **CTRL-SETADIREITA**; para mudar para a tela anterior, pressione **CTRL-SETAESQUERDA**.

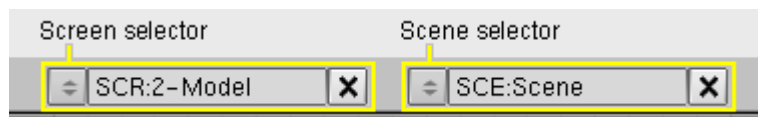


Figura 12. Seletores de cena e de tela

1.7. Cenas

Também é possível ter várias cenas dentro de um mesmo arquivo Blender. As cenas podem usar os objetos umas das outras, ou serem completamente separadas umas das outras. Você pode selecionar e criar cenas com o botão de menu SCE localizado no cabeçalho da UserPreferences (**Figura 12**).

Quando você cria uma nova cena, você pode escolher entre quatro opções para controlar o seu conteúdo:

- Empty cria uma cena vazia;
- Link Objects cria uma nova cena com o mesmo conteúdo da cena atualmente selecionada. Mudanças em uma cena irão modificar a outra;
- Link ObData cria a cena nova baseada na cena atualmente selecionada, com conexões às mesmas meshes, materiais e assim por diante. Isso significa que você pode modificar a posição dos objetos e propriedades relacionadas, mas modificações às meshes, materiais, etc. irão modificar outras cenas, a não ser que você faça manualmente cópias únicas dos objetos modificados;
- Full Copy cria uma cena completamente independente com uma cópia de todo conteúdo da cena selecionada.

2. Navegando no Espaço 3D

O Blender permite a você trabalhar em espaços tridimensionais, mas as telas de nossos monitores são apenas bidimensionais. Para conseguir trabalhar em três dimensões, você tem que ser capaz de mudar o seu ponto de vista bem como a direção de visão da cena. Ambos são possíveis em todas as 3DViews.

Mesmo que nós estejamos descrevendo a 3DView, a maioria dos outros tipos de janelas usam uma série de funções equivalentes, p.ex. é possível movimentar e ampliar uma ButtonsWindow e seus painéis.

2.1. A direção de visão (rotação)

O Blender fornece três direções de visão padrão: Lateral (*Side*), Frontal (*Front*) e Topo (*Top*). Como o programa utiliza um sistema de coordenadas de mão direita, com o eixo Z apontando para cima, “lateral” corresponde a olhar ao longo do eixo X; “frontal” ao longo do eixo Y, na direção negativa; e “topo” ao longo do eixo Z. Você pode selecionar a direção de visão para uma 3DView com os itens do menu View (**Figura 13**) ou pressionando as teclas de atalho **NUM3** para “lateral”, **NUM1** para “frontal” e **NUM7** para “topo”.

Dica – Teclas de atalho

Lembre-se que a maioria das teclas de atalho afeta a janela que está focalizada. Portanto, certifique-se de que o cursor do mouse está sobre a área da janela na qual você quer trabalhar antes de usar os atalhos!

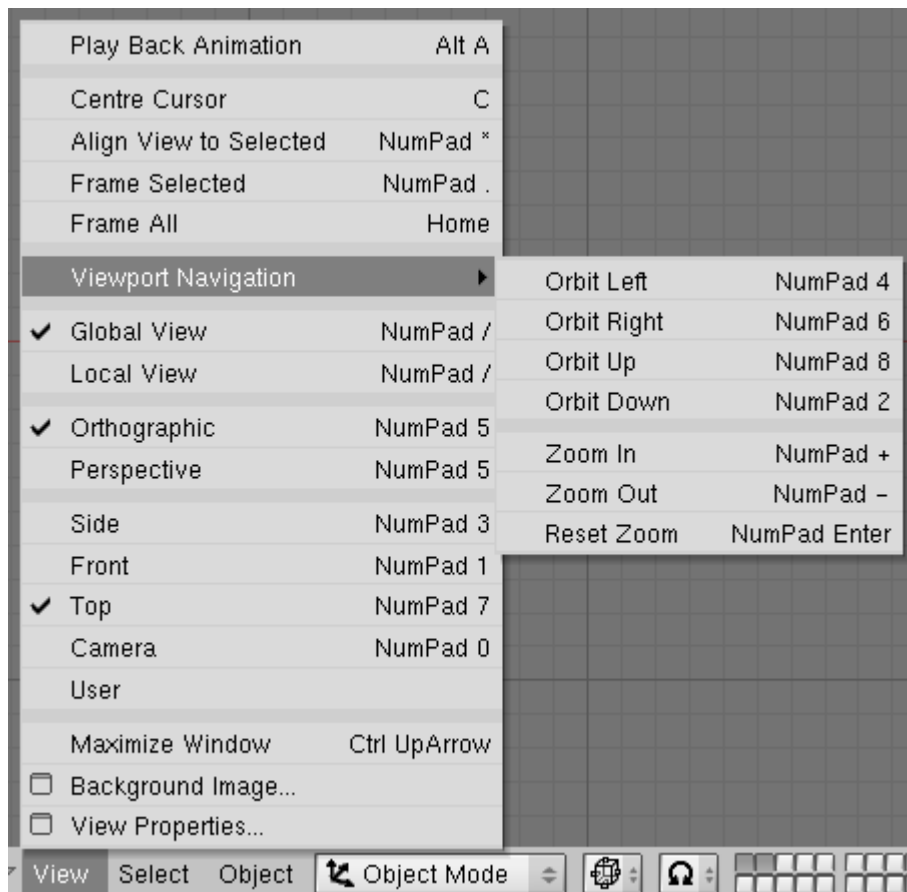


Figura 13. O menu View de uma 3DView.

Além dessas três direções-padrão, a visão pode ser rotacionada em qualquer ângulo que você desejar. Clique e arraste **BMM** na área da visão: se você começar no meio da janela para cima e para baixo ou para a esquerda e para a direita, a visão será rotacionada em torno do centro da janela. Se você começar em um canto e não mover em direção ao centro, você poderá rotacionar em torno de um dos eixos da visão. Brinque com esta função até você sentir como ela funciona.

Para mudar o ângulo de visão em passos discretos, use **NUM8** e **NUM2**, que correspondem ao arrasto vertical com **BMM**, ou então **NUM4** e **NUM6**, que correspondem ao arrasto horizontal com **BMM**.

2.2. Movendo e ampliando a visão

Para mover a visão, segure **SHIFT** e arraste **BMM** na 3DView. Para passos discretos, use **CTRL-NUM8**, **CTRL-NUM2**, **CTRL-NUM4** e **CTRL-NUM6**, tal como com a rotação.

Você pode ampliar ou reduzir a visão segurando **CTRL** enquanto arrasta **BMM**. Os atalhos são **NUM+** e **NUM-**. Tais funções também podem ser encontradas em View>>Viewport Navigation.

Dica 1 – Mouse Wheel

Se você tiver um mouse wheel, as mesmas ações de **NUM+** e **NUM-** podem ser realizadas com **MW**. A direção da rotação selecionará a ação.

Dica 2 – Se você se perder...

Se você se perder dentro do espaço 3D, o que não é incomum, dois atalhos poderão lhe auxiliar: **HOME** muda a visão de forma que todos os objetos fiquem visíveis (também acessível através de View>>Frame All), enquanto **NUM.** altera a visão de forma que os objetos selecionados fiquem visíveis (também acessível através de View>>Frame Selected).

2.3. Perspectiva e Projeção Ortográfica

Cada 3Dview suporta dois tipos diferentes de projeção. Eles estão demonstrados na **Figura 14**: ortográfica (esquerda) e perspectiva (direita).

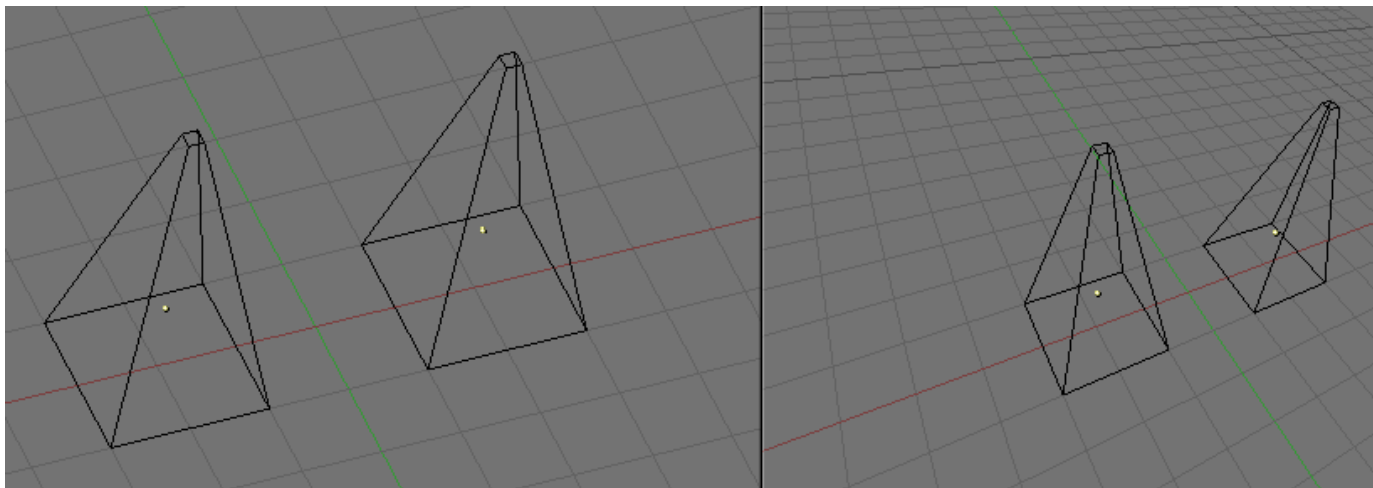


Figura 14. Projeções ortográfica (esquerda) e perspectiva (direita).

Nossos olhos estão acostumados à vista em perspectiva, onde objetos distantes parecem menores. A projeção ortográfica pode parecer estranha num primeiro momento, pois os objetos permanecem do mesmo tamanho, independente de suas distâncias: isso corresponde a ver a cena de um ponto infinitamente distante. No entanto, a vista ortográfica é muito útil (e é a vista padrão do Blender e da maioria dos programas 3D), porque ela possibilita uma percepção mais “técnica” da cena, facilitando os atos de desenhar e avaliar proporções.

Dica – Projeção em Perspectiva e Ortográfica

A vista em perspectiva é construída geometricamente da seguinte forma: existe uma cena em 3D e você é um observador colocado num ponto O . A cena 2D em perspectiva é construída colocando-se um plano, uma folha de papel, onde a cena 2D será desenhada, à frente do ponto O , perpendicular à direção da vista. Para cada ponto P na cena 3D, uma linha é desenhada, passando por O e P . O ponto de intersecção S entre essa linha e o plano é a projeção em perspectiva daquele ponto. Projetando-se todos os pontos P da cena você tem uma vista em perspectiva.

Por sua vez, numa projeção ortográfica, também chamada de “ortonormal”, você tem a direção da vista mas não tem um ponto de vista O . A linha é então desenhada através do ponto P , de forma a ficar paralela à direção da vista. A intersecção S da linha com o plano forma a projeção ortográfica. E, projetando-se todos os pontos P da cena, você tem uma vista ortográfica.

Para mudar a projeção de uma 3DView, acesse View>>Orthographic ou View>>Perspective (mostrados na **Figura 13**). O atalho **NUM5** alterna entre os dois modos.

Dica – Projeção de Câmera

Perceba que a mudança da projeção de uma 3DView não afeta a forma como a cena será renderizada. A renderização, por padrão, está em perspectiva. Se você precisar criar uma renderização ortográfica, selecione a câmera e pressione Ortho, localizado no painel Camera, dentro do contexto EditButtons (**F9**) da ButtonsWindow.

A entrada de menu View>>Camera coloca a 3DView no modo câmera (atalho: **NUM0**). A cena é então mostrada como será eventualmente renderizada (ver **Figura 15**): a imagem renderizada irá conter tudo que estiver dentro das linhas tracejadas externas. Ampliações e reduções dessa visão são possíveis, mas, para mudar o ponto de vista, você terá que mover ou rotacionar a câmera.

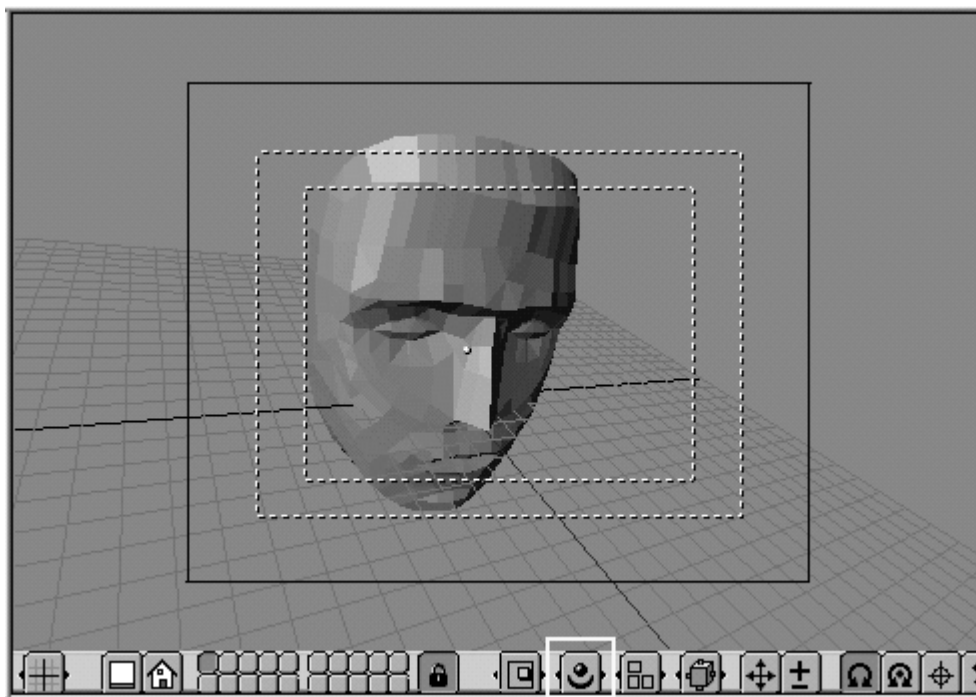


Figura 15. Demonstração da vista da câmera.

2.4. Modo de desenho

Dependendo da velocidade do seu computador, da complexidade da sua cena e do tipo de trabalho que você estiver fazendo, você vai querer alternar entre os vários modos de desenho:

- Textured (Texturizado) – Tenta desenhar tudo da forma mais completa possível, apesar de ainda não ser uma alternativa à altura da renderização. Perceba que, se você não tiver iluminação na sua cena, tudo ficará preto.
- Shaded (Sombreado) – Desenha superfícies sólidas, incluindo o cálculo de iluminação, sem as texturas aplicadas. Da mesma forma que o desenho texturizado, você não verá nada sem luzes.
- Solid (Sólido) – Semelhante ao sombreado, mas funciona mesmo que a cena não contenha iluminação.
- Wireframe (Estrutura em arame) – Os objetos consistem apenas de linhas que tornam suas formas reconhecíveis. Essa é a forma de desenho padrão.
- Bounding Box (Caixa Limítrofe) – Os objetos não são desenhados; em seus lugares são mostradas apenas caixas retangulares, correspondentes ao tamanho e à proporção de cada um deles.

O modo de desenho de uma 3DView pode ser selecionado a partir do botão Viewport Shading, localizado no cabeçalho (**Figura 16**), ou a partir dos seguintes atalhos: **ZKEY** alterna entre Wireframe e Solid; **SHIFT-Z** alterna entre Wireframe e Shaded; e **ALT-Z** alterna entre Textured e Wireframe.



Figura 16. O botão Viewport Shading de uma 3DView.

2.5. Vista local

No modo de vista local, somente os objetos selecionados são mostrados, simplificando a edição de cenas complexas. Para entrar no modo local, primeiro selecione os objetos que você for usar (veja a seção *Selecionando Objetos* do capítulo XXX, *Modo Objeto*) e então acesse o menu View>>Local View; use o menu View>>Global View para retornar à visão global (o menu é mostrado na **Figura 13**). O atalho para alternar entre as visões local e global é **NUM/**.

2.6. O sistema de layers

Cenas 3D costumam se tornar exponencialmente mais confusas com o aumento de sua complexidade. Para controlar essa situação, os objetos de uma cena podem ser agrupados em “*layers*” (camadas), de forma que somente as layers selecionadas por você sejam mostradas em um determinado momento. As layers 3D se diferenciam das layers de aplicativos gráficos 2D: elas não têm influência na ordem de desenho e existem (exceto por algumas funções especiais) meramente para fornecer ao modelador uma visão geral mais clara da cena.

O Blender oferece 20 layers; você pode escolher quais serão mostradas a partir dos botõezinhos sem identificação agrupados no cabeçalho da 3DView (Figura 17). Para selecionar apenas uma layer, clique no botão apropriado com **BEM**; para selecionar mais de uma, segure **SHIFT** enquanto clica com **BEM**.

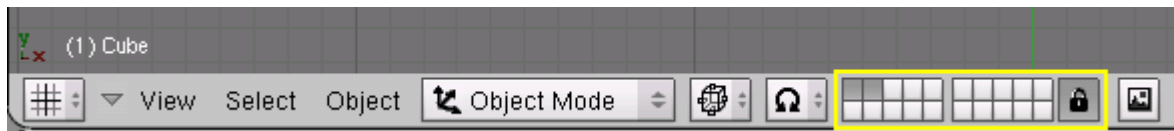


Figura 17. Botões de layers de uma 3DView (destacados pelo retângulo amarelo).

Para selecionar as layers via teclado, pressione de **1KEY** a **0KEY** (na área principal do teclado) para as layers de 1 a 10 (a fileira superior de botões), e de **ALT-1** a **ALT-0** para as layers de 11 a 20 (a fileira inferior). A tecla **SHIFT** permite a seleção de múltiplas layers.

Por padrão, o botão de trava, localizado à direita dos botões de layers, está pressionado; isso significa que mudanças feitas na visibilidade das layers em uma 3DView afetarão todas as 3DViews. Para selecionar determinadas layers em uma 3DView, sem afetar a visibilidade das demais, você deve primeiro desmarcar a trava da 3DView em questão.

Para mover objetos selecionados para uma layer diferente, pressione **MKEY**, selecione a layer que você deseja a partir do diálogo pop-up e pressione o botão Ok.

Capítulo 2

Abóbora Saltitante

Usando o arquivo de exemplo “Pumpkin-Run”, a maioria das principais técnicas de construção de um jogo 3D serão explicados e ilustrados. No entanto, não pense que você irá se tornar um desenvolvedor de jogos em algumas páginas – mas você pode aprender o básico aqui e se divertir ao mesmo tempo! Você também será encorajado a experimentar e brincar com o Blender.

O que será ensinado nesse capítulo:

- Carregar e salvar cenas no Blender;
- Manipular objetos e navegar em uma cena;
- Mapeamento de texturas básico;
- Executar a parte interativa da engine 3D integrada do Blender;
- Adicionar interatividade para controlar elementos do jogo;
- Controle de câmera e luzes;
- Animação básica de objetos;
- Adicionar e usar sons.

1. Modelando um ambiente

Inicie o Blender clicando no seu ícone. Ele será aberto na sua forma padrão, como mostra a **Figura 1-1**.

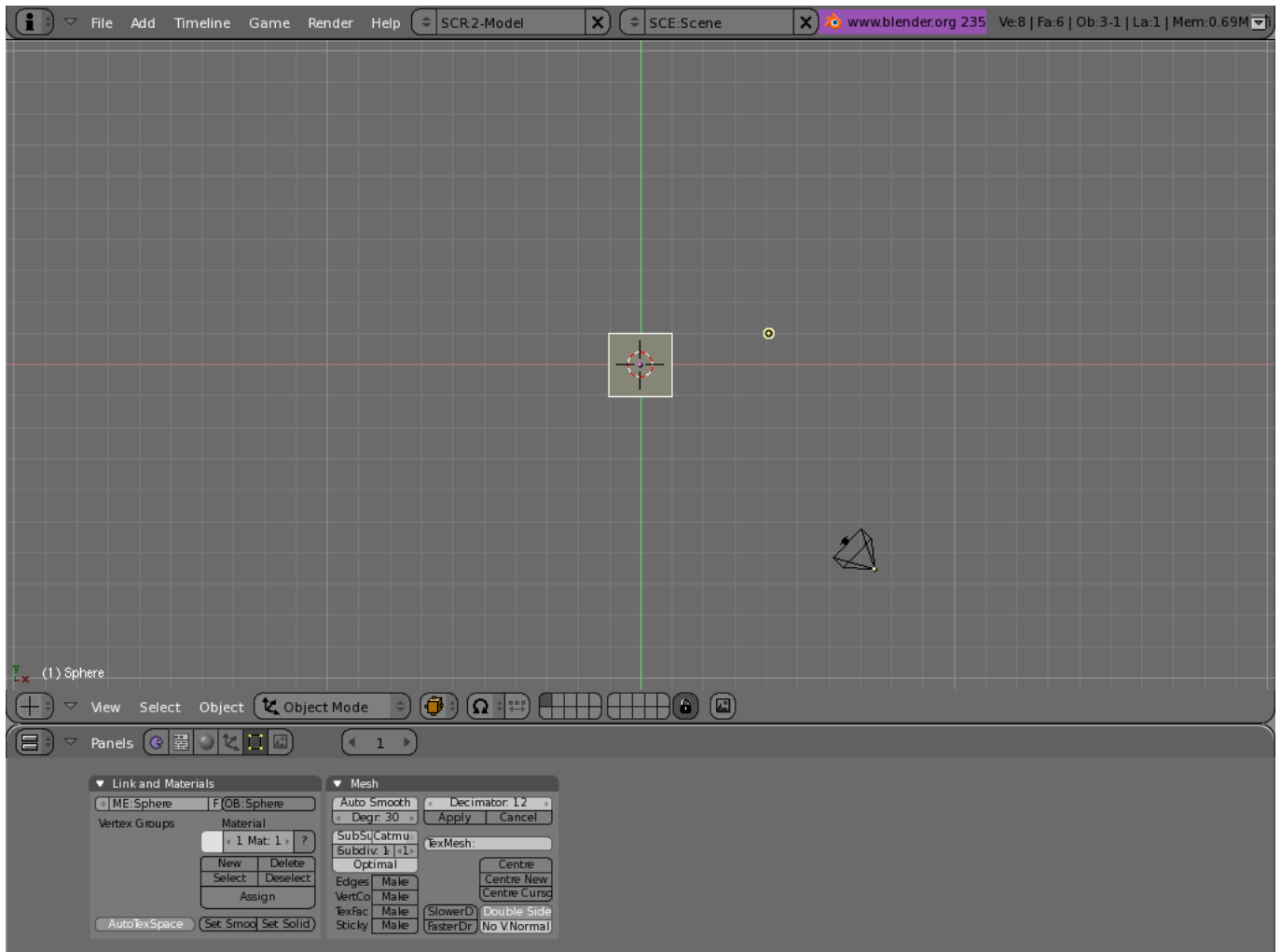


Figura 1-1. O Blender logo após a sua inicialização.

A grande janela ao centro é a 3DView, nossa janela para o mundo 3D dentro das cenas do Blender. O quadrado rosa é um cubo simples, desenhado no modo wireframe. Nós estamos atualmente olhando a cena de cima, a chamada *TopView*. O triângulo é a representação de uma câmera; o círculo amarelo é a representação de um ponto de luz.

Mova o cursor do mouse sobre a câmera e pressione **BDM**; isso selecionará a câmera.

Nota

Blender usa **BDM** para selecionar objetos!

Agora nós iremos mudar a vista da cena. Mova o cursor do mouse sobre a 3DWindow, pressione e segure **BMM** e mova o mouse para rotacionar a vista.

Dica

O Blender foi projetado para trabalhar de forma ótima usando-se um mouse de três botões. Entretanto, se você tiver apenas um mouse de dois botões, você pode substituir o botão do meio do mouse segurando **ALT-BEM**.

Agora, retorne à TopView da cena pressionando **NUM7**. As ações até aqui realizadas devem lhe dar uma idéia básica de como navegar no espaço 3D através de um ambiente de trabalho bidimensional (a tela do monitor, no caso). Todos os detalhes estão descritos no capítulo 1.

Selecione o cubo pressionando **BDM** com o mouse sobre ele. O cubo será desenhado em rosa quando você selecioná-lo efetivamente. Agora, pressione **DEL**; surgirá uma caixa de diálogo sob o ponteiro do mouse, perguntando se você quer apagar os objetos selecionados. Clique sobre ela para confirmar a ação. Se você movimentar o ponteiro do mouse para muito longe da caixa, ela irá desaparecer e você terá que pressionar novamente a tecla **DEL** e repetir o processo para confirmar a eliminação do cubo.

Após apagar o cubo, pressione **BARRAESPAÇO** para abrir a Toolbox. Selecione Add>>Mesh>>Plane. Um plano será desenhado na 3DView, centralizado no 3Dcursor.

Nota

O *3DCursor* (Cursor 3D) é um elemento especial do Blender. Ele tem no *posicionamento* a sua principal função; como exemplos, podemos citar que todos os objetos inseridos na cena 3D surgem centralizados no 3DCursor e que ele também pode ser usado como centro de rotação de qualquer objeto.

Uma vez inserido o plano, a cena será automaticamente transferida para o *EditMode* (Modo Edição). É nesse modo que você manipula os vértices dos objetos; por enquanto queremos deixar as coisas como estão e retornar ao modo padrão da 3DView, o *ObjectMode* (Modo Objeto). Para tanto, pressione **TAB** e estaremos de volta ao ObjectMode. Repare que o plano permanece selecionado.

Agora iremos modificar o plano, redimensionando-o. Certifique-se de que o plano continua selecionado (com uma borda rosa à sua volta), posicione o cursor do mouse sobre a 3DView, pressione **SKEY** e mova o mouse; o plano irá mudar de tamanho de acordo com o movimento do mouse.

Dica

O efeito do redimensionamento é diretamente proporcional à distância que o cursor do mouse se encontra do objeto selecionado, no momento em que **SKEY** é pressionada. Ou seja, se o cursor do mouse estiver próximo ao objeto, ele será redimensionado mais rapidamente; se o cursor estiver afastado, o redimensionamento terá um efeito mais reduzido. Portanto, tenha em mente o grau de redimensionamento que você deseja ao iniciá-lo e posicione o cursor de mouse apropriadamente.

Se você segurar **CTRL** enquanto movimenta o mouse, o redimensionamento será feito de 0,1 em 0,1. Redimensione o plano até que ele fique com tamanho 10,0 em todos os eixos. Para tanto, observe as informações de redimensionamento no cabeçalho da 3DView (**Figura 1-2**), então pressione **BEM** para finalizar a operação de redimensionamento.

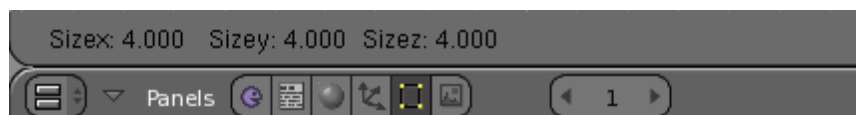


Figura 1-2. Informações de redimensionamento no cabeçalho da 3DView.

Nota

Se você quiser cancelar a operação de redimensionamento, pressione **BDM** ou **ESC**. Além do mais, **ESC** irá cancelar qualquer procedimento do Blender, sem fazer quaisquer mudanças ao seu objeto.

Agora irei mostrar como personalizar a tela do Blender, em especial a disposição das janelas. Mova o cursor do mouse lentamente sobre o extremo inferior da 3DView (**Figura 1-3**) até ele mudar para uma seta dupla. Agora pressione **BMM** ou **BDM** e um menu aparecerá.

Clique em *Split Area*. Mova a linha que surgir até o meio da 3DView e pressione **BEM**; o Blender divide a 3DView em duas vistas idênticas da cena 3D.

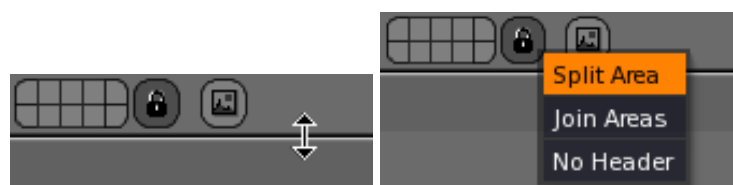


Figura 1-3. Dividindo uma janela.

Mova o mouse sobre a janela da direita e pressione **SHIFT-F10**. A janela deverá mudar para uma janela *UV/ImageEditor* (Editor UV/de imagens); esse é o local no Blender para se trabalhar com imagens e texturas que irão colorir nossos modelos em tempo-real.

Nota

Todas as teclas pressionadas serão executadas pela janela ativa (ou seja, a janela sobre a qual o mouse estiver posicionado). Não há necessidade de se clicar em uma janela para ativá-la.

Mova novamente o mouse por sobre o plano na 3DView à esquerda e selecione-a caso não esteja mais selecionada. Agora pressione **ALT-Z** – o plano é desenhado como um sólido negro. Pressione **FKEY** e o plano ficará branco, com as bordas desenhadas como linhas pontilhadas. Com a **FKEY**, nós acabamos de entrar no famoso *UVFaceSelectMode* (Modo de Seleção de Faces/UV), utilizado para selecionar faces e aplicar texturas aos modelos.

Mova o mouse para a janela da direita e selecione o menu Image>>Open...

Uma janela *FileBrowser* (Navegador de Arquivos, **Figura 1-4**) abrirá.

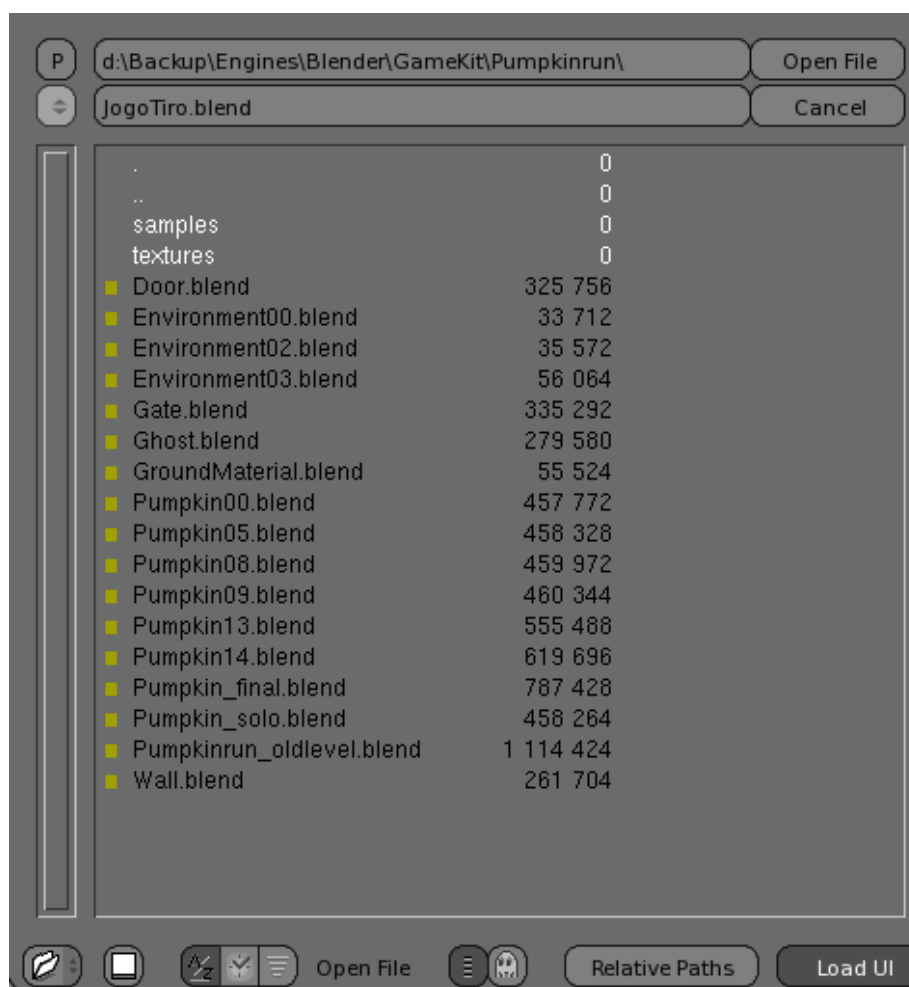


Figura 1-4. Uma janela *FileBrowser*.

Pressionando e segurando o *MenuButton* (Botão de Menu) (colocar imagem) com o **BEM** você poderá selecionar caminhos recém-navegados e, em sistemas operacionais Windows, uma lista dos seus drives.

O diretório em que você está atualmente é mostrado no campo de texto mais ao topo. O botão *ParentDir* (Diretório-Pai) (colocar imagem) permite a você subir ao diretório-pai do diretório atual.

Usando esses métodos, vá até o local onde está gravada a pasta *Pumpkinrun*, abra a pasta *textures* e localize o arquivo *conccray_q.jpg*. Clique sobre ele com **BEM** e carregue-o, pressionando a seguir o botão *Load Image*, localizado no canto superior direito da janela.

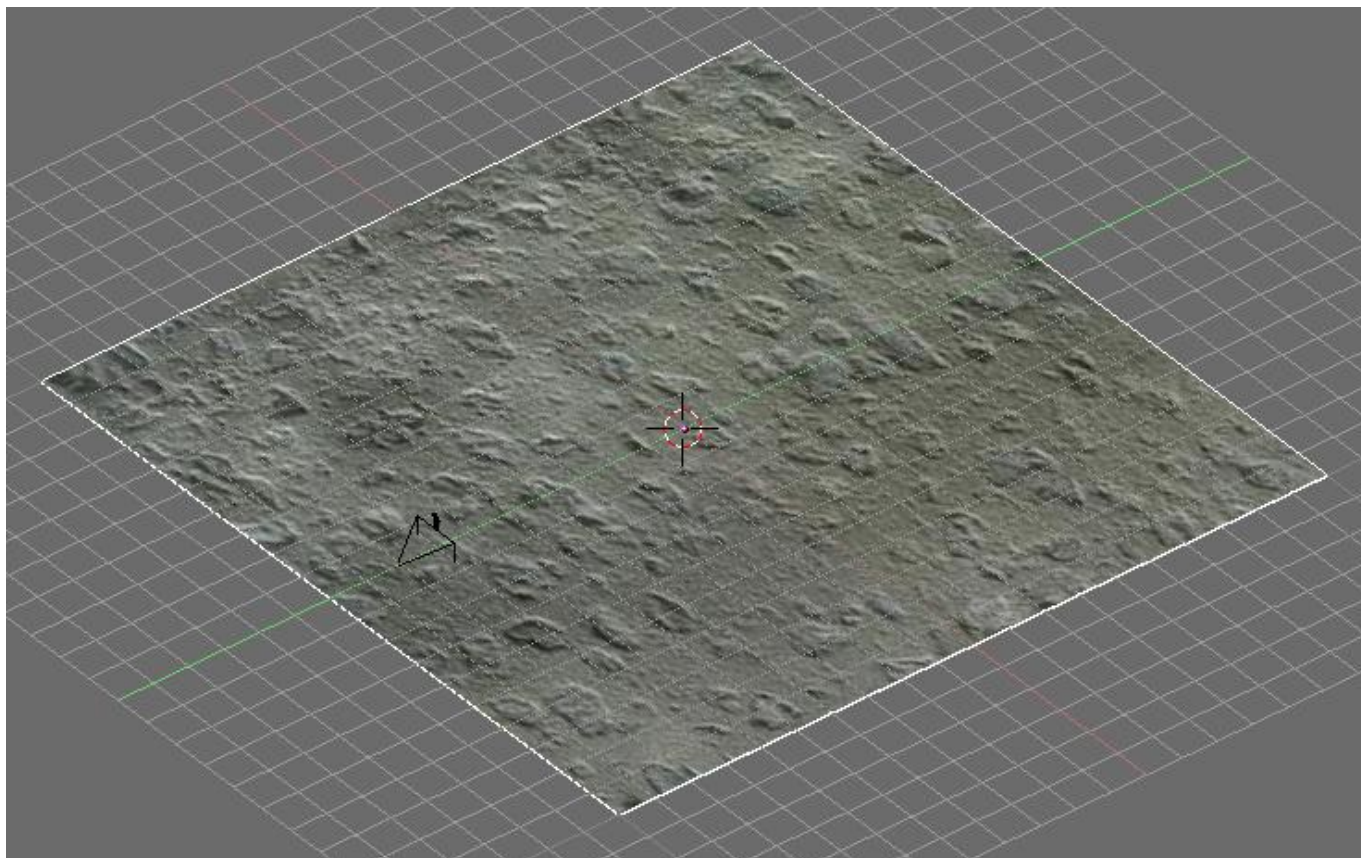


Figura 1-5. O plano texturizado na 3DView.

A textura agora é mostrada tanto na 3DView como no UV/Image Editor. Se você avistar alguma cor estranha na textura, pressione **SHIFT-K** sobre a 3DView. Agora saia do FaceSelectMode pressionando **FKEY**.

Nós acabamos de criar um ambiente bem simples, mas para tanto usamos a maioria dos passos necessários à criação de níveis mais complexos para jogos.

Agora é hora de salvar a cena. Para facilitar o processo, nós iremos incluir a textura na cena a ser salva. Portanto, selecione **File>>Pack Data** no menu no alto da tela. Um ícone de um embrulho aparecerá na barra de menu, indicando que a cena está “empacotada”. Agora, selecione **File>>Save As**, navegue pela janela da mesma forma que foi descrito anteriormente para selecionar a textura, digite um nome para o arquivo (atualmente chamado de `untitled.blend`) e clique no botão **Save File**. Os detalhes sobre como salvar arquivos foram apresentados no capítulo 1.

2. Anexando um objeto de outra cena

Como não temos tempo para cobrir os tópicos de modelagem e uso geral do Blender como uma ferramenta para criar mundos completos, nós iremos carregar objetos pré-fabricados. Na verdade, não há um formato de arquivo específico dentro do Blender para se armazenar objetos; portanto, qualquer cena pode ser usada como um arquivo de onde se retiram objetos. Ou seja, você pode navegar e reutilizar qualquer cena de qualquer arquivo que você desejar!



Figura 2-1. O herói!

Mova o cursor do mouse sobre o UV/ImageEditor (a janela da direita) e pressione **SHIFT-F5** para mudar para uma 3DView. Agora nós iremos começar a ação e para isso precisaremos de mais vistos dentro do espaço 3D.

Pressione **SHIFT-F1** com o mouse sobre uma das 3DViews – um FileBrowser aparecerá no modo *Append* (Anexo, **Figura 2-2**), permitindo-nos explorar arquivos do Blender e carregar qualquer objeto de qualquer cena dentro da cena atual.

Nota

Você também pode usar o menu File>>Append para acessar a função Append, mas, como já foi dito anteriormente, o uso de atalhos é mais rápido.



Figura 2-2. FileBrowser no modo Append.

Usando os métodos de navegação já mencionados, clique com **BEM** sobre o arquivo PumpkinSolo.blend, localizado na pasta PumpkinRun. O arquivo será aberto, mostrando todas as partes da cena como se estivéssemos em um navegador de arquivos.

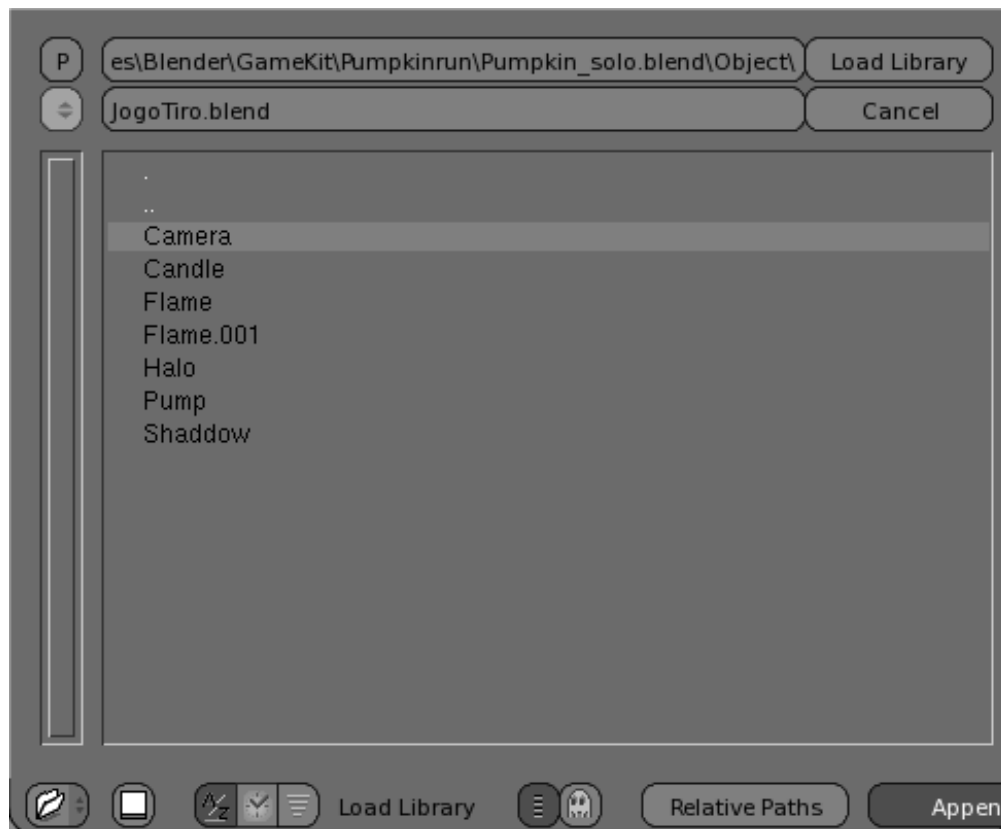


Figura 2-3. Navegando dentro de um arquivo *.blend*.

Clique com **BEM** sobre a pasta Objects. Você verá os objetos contidos na cena (**Figura 2-3**). Selecione todos os objetos pressionando **AKEY**. Confirme pressionando **ENTER** ou clicando com **BEM** no botão Load Library.

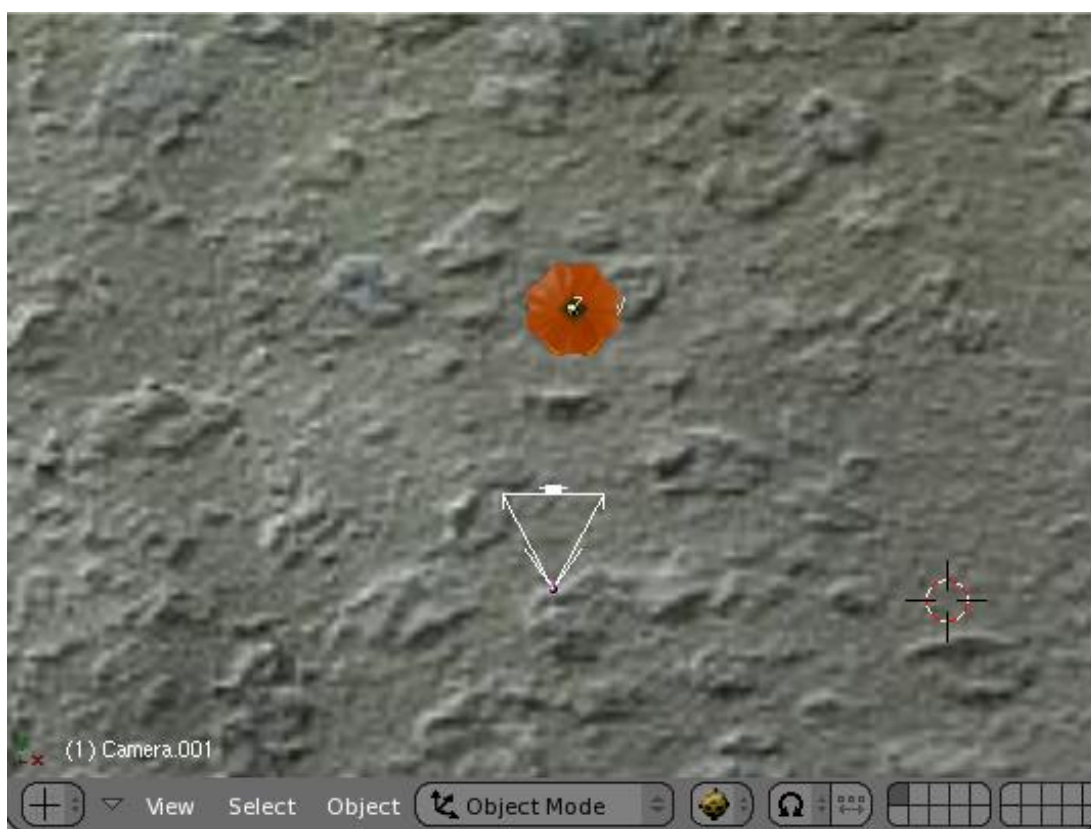


Figura 2-4. A abóbora na TopView após ser carregada no ambiente.

Agora você deve estar vendo a abóbora como um ponto laranja, descansando no meio do plano na 3DView da esquerda.

Mude a janela da direita para uma segunda 3DView, pressionando **SHIFT-F5** com o mouse sobre ela. Você verá a mesma TopView mostrada na 3DView da esquerda, mas desenhada em wireframe.

Nós anexamos uma câmera junto com os outros objetos anexados. Agora, selecione a câmera que está mais próxima da abóbora com **BDM**. Essa operação é melhor realizada numa vista em wireframe. Move o cursor do mouse de volta à 3DView da esquerda (texturizada) e pressione **CTRL-NUM0**. Isso muda a vista da cena para a câmera selecionada, dando uma bela visão do personagem.

3. Liguem os motores!

Agora já podemos iniciar o game engine do Blender! Com o mouse sobre a 3DView, pressione **PKEY** e você verá a abóbora no nosso chão texturizado. O personagem tem uma vela animada dentro dele e você pode vê-la tremeluzindo. Para parar o game engine e retornar ao Blender pressione **ESC**.

Nota

Game engine é o programa responsável pela execução das ações relativas a um jogo – por exemplo, a física, a detecção de entrada de dados pelo jogador, a inteligência artificial, etc.

Eu ouço você dizendo: “Maravilha, mas cadê a animação?”. Bem, dê-me um minuto.

Mova o mouse sobre a 3DView da direita e pressione **NUM3** para mudar para uma SideView. Amplie a vista pressionando **NUM+** algumas vezes ou segure **CTRL-BMM** e move o mouse para cima, o que lhe dará uma ampliação suave. Você também pode a vista segurando **SHIFT-BMM** e movimentando o mouse. Dessa forma nós preparamos a vista para mover a abóbora para cima.

Clique sobre o personagem com **BDM** (clique em algum lugar sobre o wireframe da abóbora) e ele ficará rosa, indicando que foi selecionado.

Antes de usar a física do game engine, existe uma opção que deve ser ativada. Com o mouse sobre a ButtonsWindow, pressione **F5** para acessar o contexto *Shading* (sombreamento) – surgem cinco sub-contextos no cabeçalho da janela. Clique no quinto ícone na barra de sub-contextos (com o desenho de um planeta) para abrir os *World buttons* (botões globais). Na guia *Mist/Stars/Physics* (no centro da janela), no topo, existe um botão de menu marcado com a opção *None*. Clique e segure **BEM** sobre ele e mude a opção para *Sumo* – agora a física do game engine foi ativada, possibilitando controle total sobre os objetos da cena.

Agora nós iremos acessar o centro de comando principal para a parte 3D interativa do Blender. Para tanto, pressione **F4** ou clique no ícone *Logic* na barra de contextos.



Figura 3-1. O contexto *Logic*.

Localize o botão *Actor* à esquerda da ButtonsWindow e clique nele com **BEM** (repare que se você não ativou a opção *Sumo*, como mostrado anteriormente, esse botão não aparecerá). Isso torna nosso personagem em essência um ator. Surgem mais dois botões; clique no botão *Dynamic*. Essa opção muda o objeto de uma maneira tal que ele passa a reagir a propriedades físicas, como gravidade, colisões elásticas ou forças aplicadas a ele. Por enquanto, nós iremos ignorar a pletora de botões que surgiram logo abaixo, quando você clicou no botão *Dynamics*.



Se você iniciar o game engine agora, não haverá muita diferença, mas nós mudaremos isso em um minuto.

Dê um zoom out na 3DView da direita (Lembra? Use **CTRL-BMM** or **NUM+/NUM-** para dar zoom). Certifique-se de que a abóbora está selecionada (rosa, senão clique sobre ela com **BDM**), pressione **GKEY** sobre a 3DView da direita e mova o mouse. O personagem irá seguir o movimento do mouse dentro da 3DView. **GKEY** inicia o *GrabMode* (modo agarrar), permitindo a você movimentar objetos dentro do espaço 3D.

Dica

Pressionando **XKEY**, **YKEY** ou **ZKEY** você restringe a ação do GrabMode ao eixo relacionado à tecla de atalho.

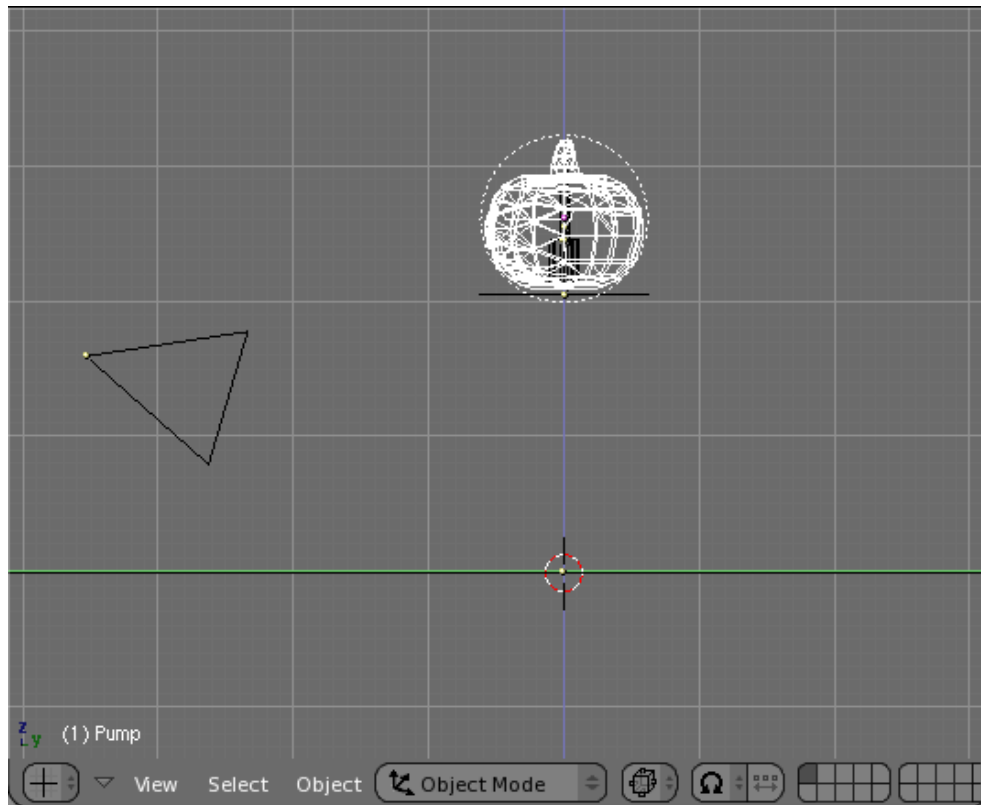


Figura 3-2. A SideView após mover a abóbora para cima.

Mova o objeto em linha reta para cima (pressione **YKEY** para auxiliar no movimento) até ele desaparecer da 3DView da esquerda e confirme a nova posição com **BEM**. Se você não tiver certeza da sua ação, você pode cancelar a operação com **ESC** ou **BDM** e tentar de novo.

Agora mova o mouse sobre a 3DWindow da direita (a *CameraView* ou vista da câmera) e pressione **PKEY** para iniciar o game engine. A abóbora cai e quica gentilmente até repousar no chão. Pressione **ESC** para sair do game engine.

4. Interatividade

O contexto *Logic* (**F4**) é dividido logicamente em quatro colunas. Nós já utilizamos a coluna mais à esquerda para configurar os parâmetros do objeto para fazer a abóbora cair. As três colunas da direita são usadas para construir a interatividade do nosso jogo.

Então vamos movimentar a abóbora ao nosso comando.

As colunas são rotuladas como *Sensors* (sensores), *Controllers* (controladores) e *Actuators* (ativadores). Você pode imaginar os *Sensors* como se fossem os sentidos de um ser vivo, os *Controllers* como o cérebro e os *Actuators* como os músculos do mesmo.

Pressione o botão Add uma vez em cada coluna com **BEM** para construir um *LogicBrick* (bloco lógico) de cada categoria – um Sensor, um Controller e um Actuator. (Figura 4-1).



Figura 4-1. LogicBricks recém-criados.

Os tipos de LogicBricks adicionados estão quase certos; para nossa primeira tarefa, apenas o primeiro precisa ser modificado. Pressione e segure o botão de menu rotulado **Always** (sempre) e escolha **Keyboard** do menu que surge (Figura 4-2).



Figura 4-2. Modificando o tipo de LogicBrick.

Agora clique **BEM** no campo **Key** do **Keyboard Sensor** (sensor de teclado) – o texto **Press any key** (pressione qualquer tecla) aparece. Pressione a tecla que você quer utilizar para mover o jogador para frente (eu sugiro **SETACIMA**).

Agora dê uma olhada mais de perto no **Motion Acuator** (ativador de movimento). Nós iremos definir como o jogador deverá mover. A primeira linha de números rotulada **Force** (força) define a quantidade de força que será aplicada quando o Motion Actuator estiver ativo. Os três números definem as forças nas direções X, Y e Z, respectivamente.

Se você olhar mais atentamente à vista wireframe do jogador você verá que o eixo local X está apontando para a frente dele. Portanto, para mover o personagem para frente, precisamos aplicar uma força positiva na direção do eixo X. Para isso, clique e segure **BEM** sobre o primeiro número na linha **Force**. Arraste o mouse para a direita para incrementar o valor para 80.00. Você pode segurar **CTRL** para restringir o valor a múltiplos de dez. Outra forma de entrar um valor exato é segurar **SHIFT** enquanto clica com **BEM** no campo; isso permitirá a você entrar um valor usando o teclado.

Tendo quase criado a configuração mostrada na Figura 4-3, agora podemos conectar os LogicBricks. Os fios passarão a informação de LogicBrick para LogicBrick, ou seja, de um Sensor para um Controller e de um Controller para um Actuator.

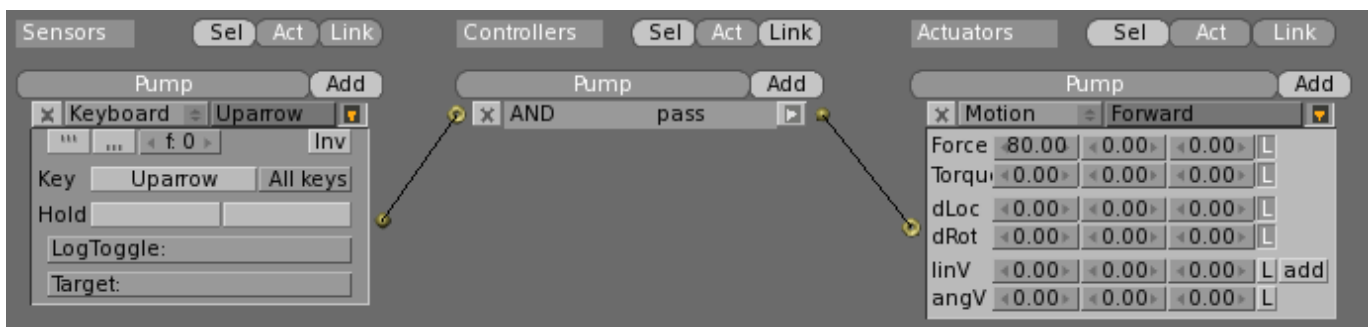


Figura 4-3. Os LogicBricks usados para mover o jogador para frente.

Clique e segure **BEM** sobre a bolinha amarela à direita do Keyboard Sensor e arraste a linha que surgir até o anel amarelo à esquerda do *AND Controller* (controlador E). Solte o mouse e os LogicBricks estarão conectados! Agora, conecte a bolinha amarela à direita do AND Controller com o anel do Motion Controller.

Para apagar uma conexão, mova o mouse sobre a conexão. A linha ficará destacada e poderá ser apagada com **XKEY** ou **DEL**.

Dica

Sempre nomeie seus objetos e LogicBricks; isso irá ajudá-lo a se encontrar através das cenas e se referir a LogicBricks específicos mais tarde. Para nomear um LogicBrick, clique no campo do nome (à direita do botão de seleção do tipo de LogicBrick) com BEM e entre o nome usando o teclado. Inicialmente, o Blender irá nomear os LogicBricks automaticamente com nomes únicos, tais como “sensor1”, “sensor2” ou “act”, “act1”, etc.; portanto, você não tem que se preocupar com colisão de nomes. Um detalhe importante nos nomes dos LogicBricks é que eles são sensíveis ao caso – portanto, “MoveEsquerda” e “moveesquerda” são considerados pelo Blender como dois nomes diferentes.

Agora pressione **PKEY** para iniciar o game engine e quando você pressionar **SETACIMA** brevemente, o jogador irá se mover em direção à câmera.

Para tornar o movimento mais interessante, nós podemos adicionar um pulo. Para tanto, entre com “20.00” no terceiro campo (eixo Z) da linha *Force* do Motion Controller. Se você testar o movimento no game engine agora, você verá que existe um problema: ao segurar a tecla pressionada, a abóbora é lançada ao espaço! Isso acontece porque a força continua sendo aplicada mesmo estando no ar.

Para resolver isso, temos que nos certificar que as forças somente serão aplicadas quando a abóbora tocar o chão. É aqui que o *Touch Sensor* (sensor de toque) entra em ação. Adicione um novo Sensor clicando em Add na coluna Sensors. Mude o tipo para Touch, da mesma forma como você fez com o Keyboard Sensor (**Figura 4-2**).

Conecte o Touch Sensor ao AND Controller. Agora o Keyboard e o Touch Sensors estão conectados ao Controller. O tipo AND do Controller ativará o Motion Actuator somente se a tecla estiver pressionada E o jogador estiver tocando o chão. Essa é uma maneira fácil de adicionar lógica às suas cenas interativas. Além do AND Controller, existem os Controller OR (OU), Expression (Expressão) e Python (a linguagem de script do Blender), que oferecem mais flexibilidade na construção da lógica do seu jogo.

Dica

A essa altura, o espaço na ButtonsWindow deve estar ficando escasso. No entanto, ao invés de modificar o layout de janelas, você pode minimizar os LogicBricks. Para isso, pressione a setinha laranja logo abaixo do nome do bloco – assim, você ainda é capaz de visualizar as conexões, apesar do conteúdo do bloco estar escondido.

Para tornar o movimento mais dinâmico, nós iremos adicionar agora LogicBricks para fazer a abóbora pular sem parar. Adicione um novo Controller e um novo Actuator clicando em Add nas colunas apropriadas. Nomeie o novo Actuator “PularSempre”. Conecte o Touch Sensor com o novo AND Controller e este ao novo Actuator “PularSempre”.

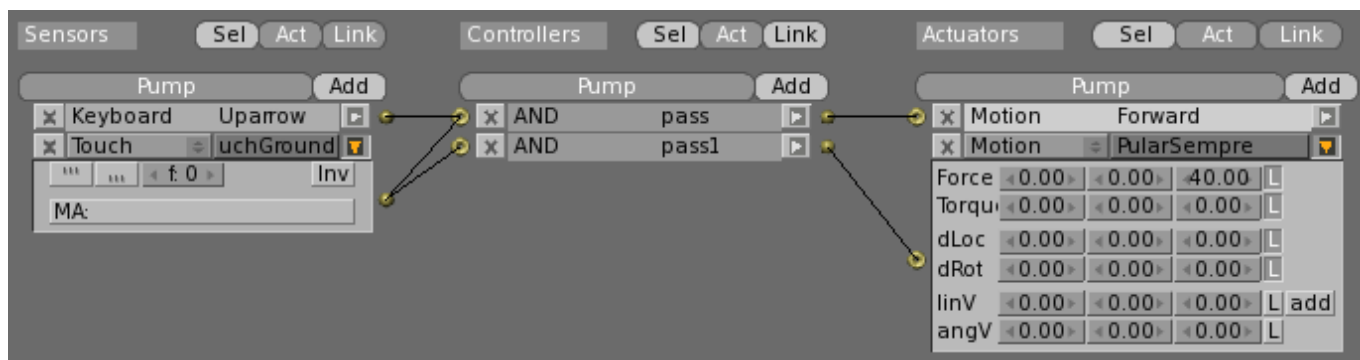


Figura 4-4. LogicBricks para adicionar pulso constante.

Sim, não só um Controller pode estar conectado a dois Sensors, como um Sensor pode alimentar mais de um Controller. Inicie o jogo novamente com **PKEY** – a abóbora pula e **SETACIMA** a move para frente.

4.1. Mais Controle

Agora iremos adicionar mais LogicBricks para pilotar o jogador com as setas direcionais.

Adicione um Sensor, um Controller e um Actuator clicando no botão Add de cada coluna. Mude o tipo do Sensor para Keyboard com o botão de menu. Não se esqueça de nomear os LogicBricks clicando no campo do nome de cada bloco. Conecte o Sensor (“SetaEsquerda”) com o Controller (“passa2”) e o Controller com o Actuator (“Esquerda”). Use **SETAESQUERDA** como a tecla ativadora do Sensor “SetaEsquerda”.

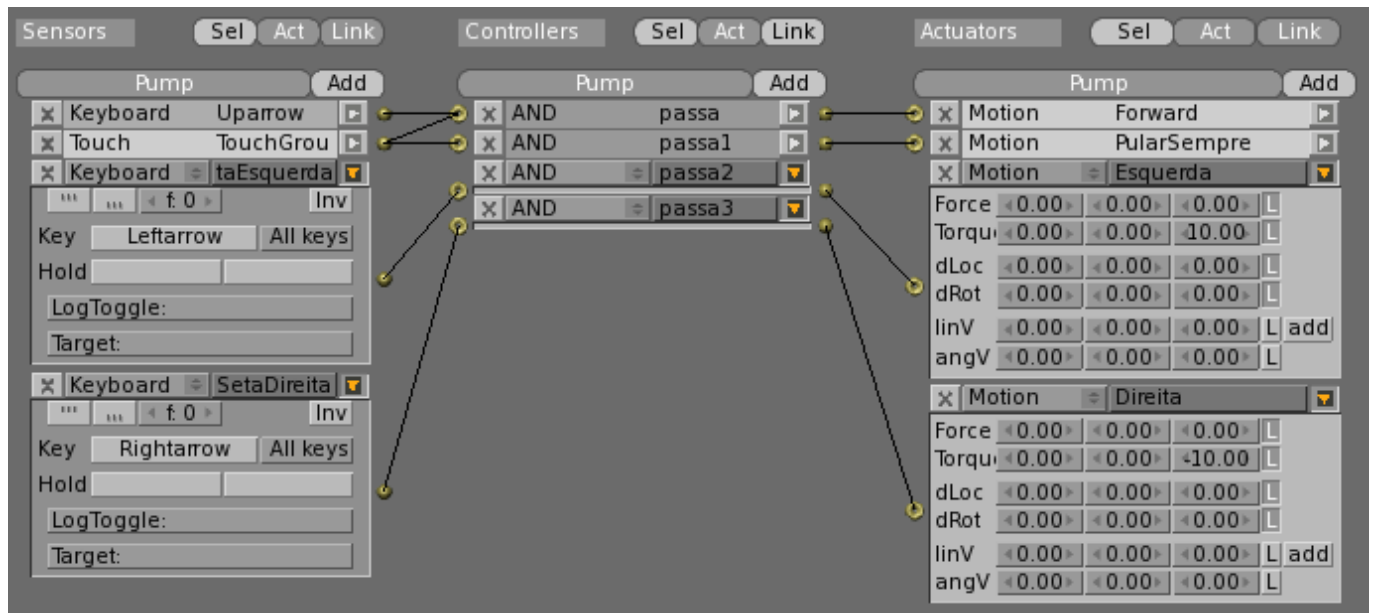


Figura 4-5. LogicBricks para pilotar o jogador.

Entre “10.0” no terceiro campo (eixo Z) da linha rotulada como Torque. Torque é a força que gira o objeto; nesse caso, o objeto irá girar em torno do seu eixo longitudinal. Tente a mudança no game engine – a abóbora irá virar para a esquerda quando você pressionar **SETAESQUERDA**. Repita os passos, mas modifique-os para girar o objeto para a direita. Para tanto, use **SETADIREITA** e entre com um Torque no eixo Z de “-10.0” (Figura 4-5).

5. Controle da câmera

Nessa seção, eu mostrarei a você como configurar uma câmera que irá seguir o ator, tentando imitar um *cameraman* de verdade.

Mova o mouse sobre a 3DView da direita (a vista em wireframe) e dê um zoom out com **NUM-** ou **CTRL-BMM** e movimentos do mouse. Localiza a segunda câmera (a que está mais distante do jogador) e selecione-a com **BDM**. Com o mouse sobre a 3DView esquerda (texturizada), pressione **CTRL-NUM0** – a vista é modificada para a câmera selecionada.

A vista agora está um pouco estranha porque a câmera está posicionada exatamente no plano do chão. Mova o mouse sobre a 3DView da direita e pressione **GKEY** para entrar no GrabMode. Mova o mouse para cima até a abóbora ficar aproximadamente no meio da CameraView (3DView da esquerda).

Certifique-se de que o contexto Logic (**F4**) da ButtonsWindow está aberto. Agora adicione um Sensor, um Controller e um Actuator como você aprendeu anteriormente. Conecte os LogicBricks e mude o tipo do Actuator para um *Camera Actuator* (ativador de câmera). O Camera Actuator seguirá o objeto de uma maneira flexível, resultando em movimentos suaves.

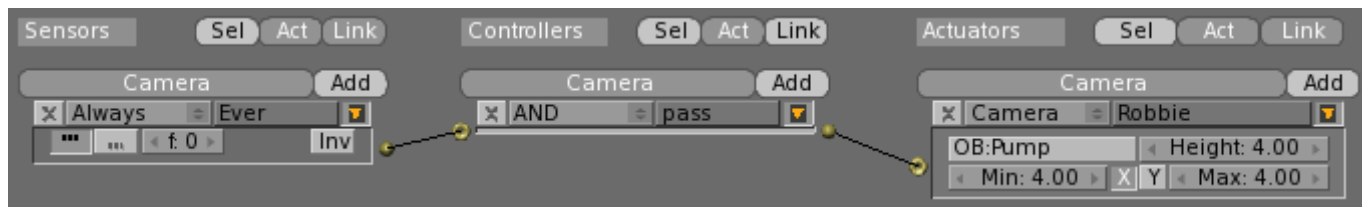


Figura 5-1. LogicBricks para a câmera perseguidora.

Clique com **BEM** no campo OB: no Camera Actuator e entre o nome da do objeto abóbora, “Pump”. A câmera irá seguir esse objeto. Clique e segure **BEM** sobre o campo Height e mova o mouse para a direita para aumentar o valor para cerca de 4.0. Essa é a altura em que a câmera permanecerá com relação ao objeto.

Dica

Segurar CTRL enquanto se ajusta um botão numérico irá modificar o seu valor em estágios, facilitando o ajuste do valor. SHIFT-LMB em um botão numérico permitirá a você entrar o valor manualmente.

Os campos Min: e Max: determinam as distâncias máxima e mínima que a câmera poderá ficar do objeto. Eu escolhi “Min: 4.0” e “Max: 6.0”. Inicie o game engine para testar o Camera Actuator. Experimente um pouco com os valores.

6. Luz em tempo real

A luz em tempo real no game engine do Blender é executada pelo subsistema OpenGL e tira proveito da transformação e iluminação (Transform and Lighting ou, simplesmente, T&L) acelerada por hardware, se a sua placa gráfica fornecê-la.

Coloque o 3DCursor com **BEM** na 3DWindow da direita, aproximadamente três unidades da grade acima das câmeras. Use o Toolbox (**BARRAESPAÇO**), Add>>Lamp.

Observe o efeito na abóbora, na 3DWindow texturizada da esquerda. Como referência, a abóbora da esquerda na figura 6-1 está acesa e a da direita não. Tente mover a luz em torno da abóbora na 3DWindow (certifique-se que a luz está selecionada [rosa, use **BDM** para selecionar] e pressione **GKEY**), para que você consiga perceber que a vista é atualizada em tempo real. Movendo a luz sob a abóbora dá um visual mais assustador, por exemplo.

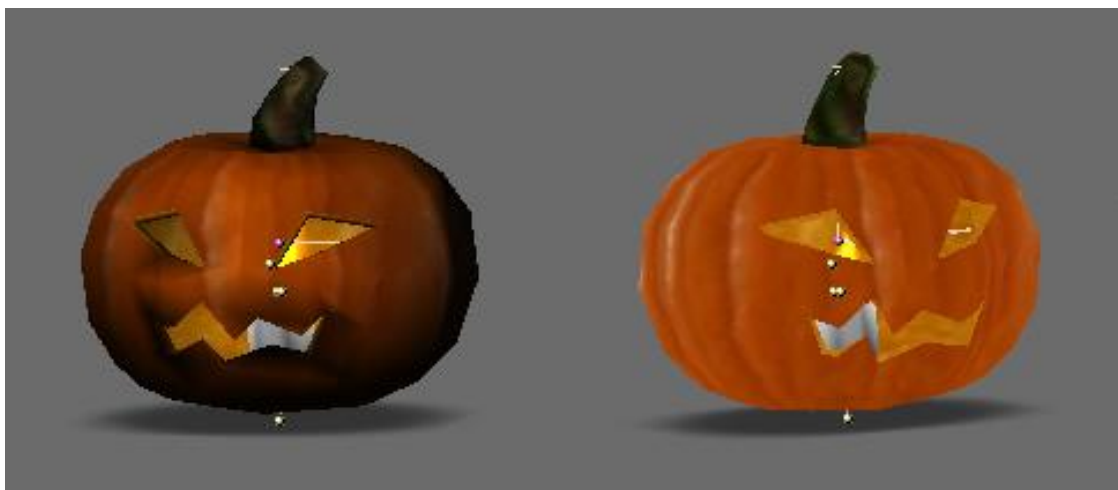


Figura 6-1. Adicionando uma luz à cena.

Nota

Por enquanto, a luz em tempo real do Blender não projeta sombras. A sombra da abóbora é criada de outra forma. Também tenha em mente que luzes em tempo real causam uma diminuição no rendimento dos seus jogos. Portanto, procure manter o mais baixo possível o número de objetos com luzes em tempo real ativadas.

7. Animação de objetos

Aqui eu irei cobrir o básico sobre a combinação do sistema de animação do Blender com o game engine. As curvas de animação (Ipos, redução de *interpolations*) do Blender estão totalmente integradas no programa, dando-lhe controle total tanto sobre as animações convencionais quanto sobre as interativas.

Use **SHIFT-F1** ou Menu>>Append. Navegue até a pasta Pumpkinrun e abra o arquivo Door.blend, clique em Object, selecione todos os objetos com **AKEY**, confirme com **ENTER**. Isso irá anexar uma parede com uma porta de madeira à cena; a abóbora irá colidir com a parede e a porta. A detecção de colisão é manipulada automaticamente pelo game engine do Blender.

Mude a 3DWindow da direita para uma TopView (**NUM7**) e dê zoom (**NUM+** ou **NUM-**), conforme for necessário, para ver a porta por completo. A porta tem o nome e o eixo ativados, portanto eles devem estar visíveis. Selecione a porta com **BDM** (ela ficará rosa).

Agora nós iremos fazer uma simples animação por *keyframe* (quadro-chave):

1. Certifique-se que o FrameSlider (a barra de seleção de frames, ver figura ao lado) está posicionado no frame 1 pressionando **SHIFT-SETAESQUERDA**.
2. Pressione **IKEY**, selecione Rot do menu que irá aparecer.
3. Agora, avance o tempo da animação pressionando SETACIMA cinco vezes para posicionar o FrameSlider no frame 51. Com o game engine rodando a 50 quadros por segundo, a nossa animação irá rodar em um segundo.
4. Pressione **RKEY** (mantenha o cursor do mouse sobre a TopView) e rotacione a porta 150° sentido horário. Você pode ver o grau de rotação no cabeçalho da 3DWindow. Para facilitar a rotação exata, segure **CTRL** enquanto rotaciona a porta.
5. Agora, insira uma segunda um segundo keyframe pressionando **IKEY** e novamente escolha Rot.
6. Mova o FrameSlider novamente ao frame 1 (**SHIFT-LEFTARROW**) e pressione **SHIFT-ALT-A**. Você verá a animação da porta sendo executada. Após 51 frames a animação irá correr até o frame 250 e então repetirá.
7. Pressione **ESC** para parar a animação.

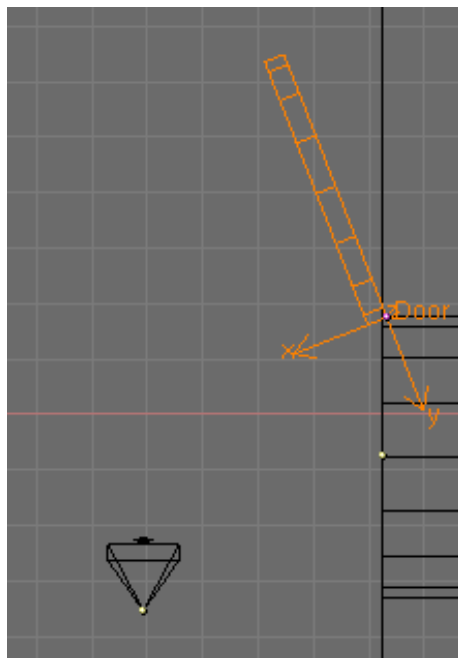


Figura 7-1. Rotacionando a porta.

Com a porta ainda selecionada, mude a ButtonsWindow para o contexto Logic (**F4**). Adicione um Sensor, um Controller e um Actuator, ligue-os, nomeie-os, mude o Sensor para um Keyboard Sensor, escolha BARRAESPACO como sua tecla ativadora e mude o Actuator para o tipo Ipo.

Mude o modo do Ipo Actuator para Ping-Pong, usando o botão de menu. Clique com **SHIFT-BEM** no campo Sta e mude o valor para “1”; mude também o valor de End para 51 (Figura 7-2). Dessa forma, o Ipo Actuator roda a animação da porta do frame 1 ao 51, abrindo a porta. Uma nova invocação do Ipo Actuator irá fechar a porta (por causa do modo Ping Pong).



Figura 7-2. LogicBricks para rodar um Ipo no modo Ping-Pong.

Execute a cena (**PKEY**) na vista texturizada e a porta irá abrir e fechar quando você pressionar **BARRAESPAÇO**; você poderá mover o ator se ele for atingido pela porta. Para visualizar as curvas de animação (Ipos) mude uma das janelas para uma *IpoWindow* (Janela Ipo), pressionando **SHIFT-F6**.

8. Refinando a cena

Você deve ter notado que existe um problema na cena: o ator pode escalar a parede, porque ele pode pular a cada toque em qualquer objeto, mesmo as paredes.

Selecione a abóbora e observe o Touch Sensor – o campo **MA:** está vazio. **MA:** referencia o nome de um material. Se você preencher esse campo com o nome de um material, o Touch Sensor só irá reagir a objetos com esse material. No nosso caso, queremos que a abóbora reaja somente ao chão que foi criado no início. Mas qual é o nome do material? Na verdade, nós não definimos um material. Para tornas as coisas mais simples, nós iremos mais uma vez utilizar a habilidade de anexar elementos pré-fabricados de uma cena diferente.

Pressione **SHIFT-F1** com o cursor do mouse posicionado sobre uma das 3DWindows, o que irá abrir uma FileWindow no modo Append. Vá até a pasta Pumpkinrun e abra o arquivo GroundMaterial.blend, clicando com **BMM** sobre ele – o arquivo será aberto imediatamente. Como alternativa, você pode clicar no arquivo com **BEM** e pressionar **ENTER** para abri-lo.

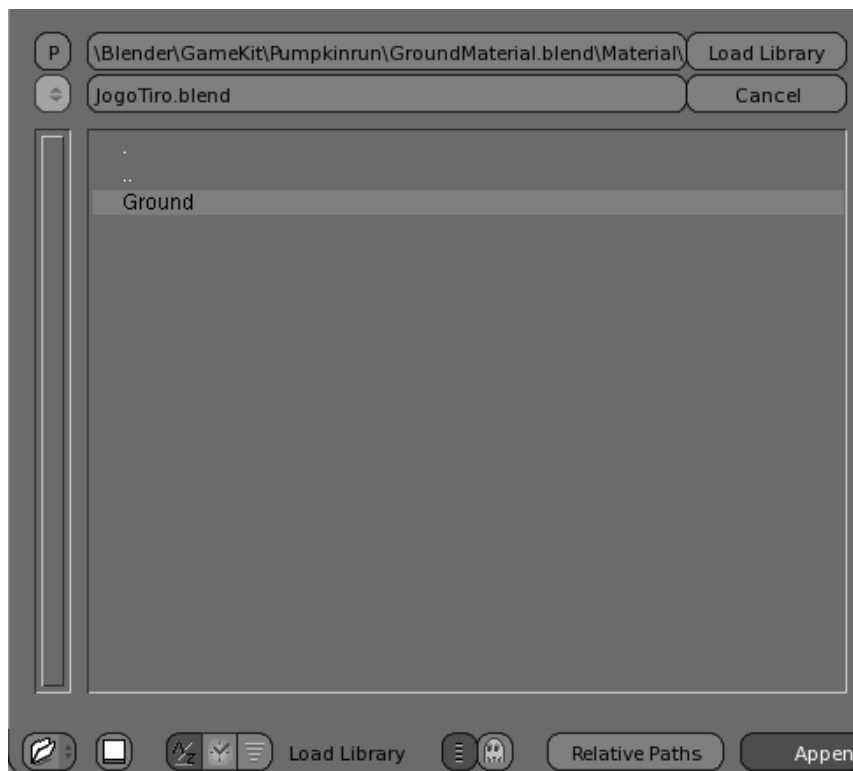


Figura 8-1. Navegando dentro do arquivo GroundMaterial.

Clique com **BEM** na pasta Material – você verá os materiais contidos na cena (**Figura 8-1**). Selecione o material Ground com **BDM** e clique no botão Load Library.

De volta à 3DWindow, selecione o plano do chão e pressione **F5** para chamar o contexto Shading da ButtonsWindow. Localize o Botão de Menu (figura) no cabeçalho da ButtonsWindow, clique nele e segure-o com **BEM**. Escolha “0 Ground” a partir do menu. O zero no nome nos indica que ainda não há objetos utilizando esse material.

Agora, selecione novamente a abóbora, retorne ao contexto Logic (**F4**) e escreva “Ground” no campo MA: do Touch Sensor.

Nota

Lembre-se que o Blender diferencia maiúsculas de minúsculas! Portanto, “Ground” e “ground” são vistos como duas coisas completamente diferentes. O Blender irá apagar um campo quando você tentar colocar o nome de um objeto que não existe. Isso pode soar frustrante, mas realmente ajuda no momento de depurar o jogo, porque previne que você negligencie um detalhe fácil de passar despercebido.

Tente sair pulando ao longo da cena, agora você não consegue mais escalar a parede.

Uma última coisa: seria legal se a porta abrisse apenas quando o ator estiver próximo a ela. O *Near Sensor* (sensor de proximidade) irá nos ajudar aqui.

Selecione a porta e adicione um novo, mudando-o para o tipo Near. Conecte-o ao AND Controller que já existe (**Figura 8-2**). O campo Dist: dá a distância na qual o Near Sensor começará a reagir ao ator. Aliás, ele reagirá a qualquer ator se deixarmos o campo Property: vazio. O campo Reset: marca a distância entre o Near Sensor e o ator, na qual o Near Sensor “esquece” o ator. Tente mudar o campo Dist: para “4.0”; agora você precisa se aproximar da porta primeiro, mas você pode recuar antes de pressionar **BARRAESPAÇO** para abrir a porta sem correr o risco de ser atingido.

Agora a porta só abrirá quando você apertar **BARRAESPAÇO** e a abóbora estiver próxima da porta.

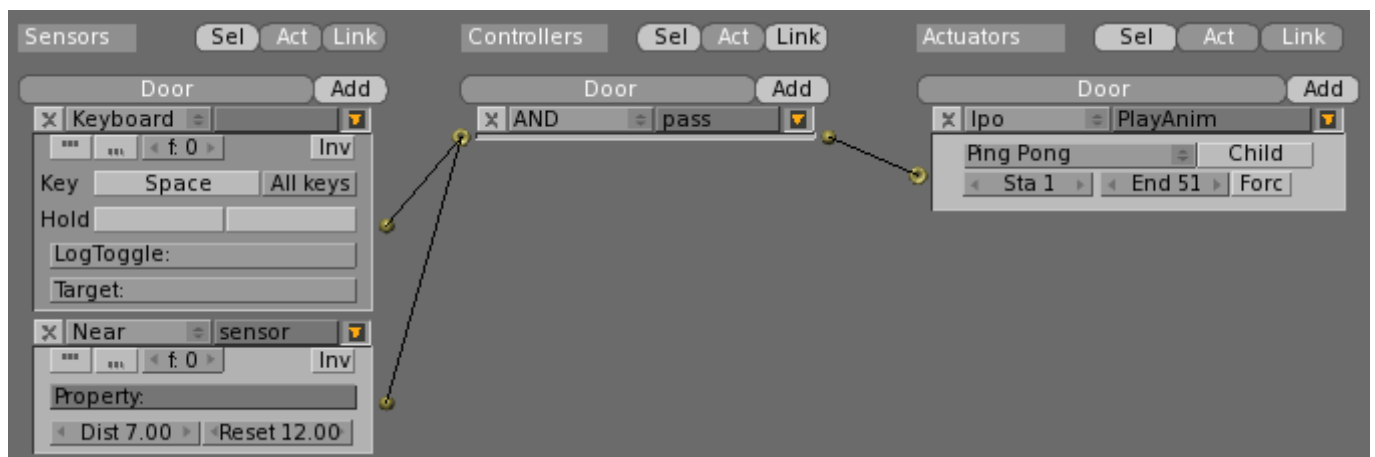


Figura 8-2. Near Sensor.

9. Adicionando som à nossa cena

Aqui eu mostrarei como adicionar um som a um evento no Blender.

No lugar da 3DWindow abra a janela Audio Timeline. Clique no botão de menu (**Figura 9-1**) no cabeçalho da janela e escolha a opção OPEN NEW. Localize a pasta Pumpkinrun, abra a subpasta samples e selecione o arquivo DoorOpen.wav, clicando nele com **BMM** ou clicando com **BEM** e, em seguida, pressionando o botão SELECT WAV FILE.

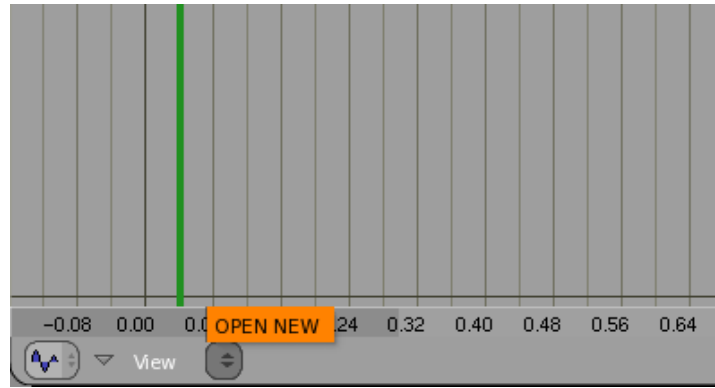


Figura 9-1. Carregando um novo arquivo de som.

Retorne à 3DWindow (**SHIFT+F5**), selecione a porta com **BDM** e mude para o contexto Logic (**F4**). Adicione um novo Actuator e mude o seu tipo para Sound. Conecte-o ao controlador (**Figura 9-2**). Clique e segure o botão de menu solitário que aparece no Actuator e selecione o arquivo de som DoorOpen.wav do menu. Por último, mude o modo Play Stop para Play End – isso significa que o som será executado por completo, sem ser interrompido prematuramente.



Figura 9-2. O Sound Actuator da porta.

E aqui concluímos o capítulo. Agora você deve ter uma idéia de como funciona o processo de criação de um jogo com o Blender – nós percorremos alguns dos passos básicos e a essa altura você já está preparado para seguir outros tutoriais, começar a estudar cenas prontas ou desenvolver as suas próprias idéias.

Capítulo 3

Lógica e LogicBricks

Agora que você matou a vontade de fazer alguma coisa no Blender, está na hora de aprofundar um pouco mais na área de lógica. Não tenho pretensões de fazer um ensaio matemático sobre o assunto, mas algumas questões precisam estar bem resolvidas para que possamos continuar adiante.

Nesse capítulo você vai aprender:

- O que é lógica booleana;
- O que são Logicricks;
- Quais são as três categorias de LogicBricks;
- Como a lógica booleana se aplica aos LogicBricks do Blender.

Eu sei que é meio chato ter que tocar nesses assuntos matemáticos, mas, infelizmente, sem entender de lógica vai ficar um tanto complicado utilizar os LogicBricks de forma adequada. Portanto, respire fundo e vamos em frente!

1. Mas é lógico!

Você já se pegou falando que alguma coisa é lógica?

Mas você já parou pra pensar o que seria lógica?

O bom e velho dicionário Aurélio define lógica como “coerência de raciocínio, de idéias”. Para que haja coerência, devem existir princípios regendo um raciocínio; e, levando-se em conta que qualquer coisa pode ser um princípio, concluímos que existem tantas lógicas quanto existem seres humanos!

(Bom, se levarmos em conta que cada pessoa acaba seguindo mais de uma lógica, que no final significa que ela segue lógica nenhuma, podemos igualar o número de lógicas à população de baratas do mundo...)

Quer um exemplo? Se uma pessoa tem como princípio “dinheiro não é tudo, mas é 100%”, como você acha que essa pessoa deve viver? Se ela for coerente ao seu princípio, ela deverá gastar todo o seu tempo perseguindo as verdinhas. Afinal, dinheiro não é tudo, mas é 100%!

Agora, se adicionarmos o princípio “dinheiro não traz felicidade” ao conjunto de princípios dessa pessoa, como ela deverá reagir para se manter coerente? Tendo uma síncope nervosa, provavelmente: não é preciso ser um físico nuclear para perceber que os dois princípios são contraditórios entre si.

Qual dos dois está certo? Sei lá eu! Só sei que, se você quer viver com um pouco de lógica na sua vida, você deve ser coerente aos princípios que definem a *razão* daquela lógica. Razão é avaliação de idéias; é comparar um conjunto de idéias com princípios bem-definidos e conseguir chegar a uma conclusão coerente, ou seja, uma conclusão lógica.

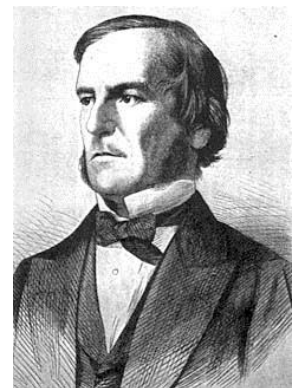
1.1. Entra em cena a lógica booleana

No dia 02 de janeiro de 1852, George Boole, quando estava se preparando para escrever seu livro “*Uma investigação das leis do pensamento, sobre as quais fundam-se as teorias matemáticas de lógica e probabilidade*”, escreveu para um amigo seu o seguinte:

...[Eu olho para a minha teoria de lógica e probabilidade] no seu estado atual como a mais valiosa, senão a única contribuição valiosa que eu fiz ou farei à Ciência e o fato pela qual eu desejaria ser lembrado daqui por diante.

E o que George Boole fez? Nada de mais: apenas definiu uma teoria de lógica que seria a pedra fundamental da eletrônica e que, por sua vez, viria a definir a base de construção dos chips de computadores. Pouca coisa, não? :-)

Essa teoria viria a ser conhecida como *lógica booleana*, em homenagem ao seu criador. A lógica booleana é uma espécie de álgebra que reduz todas as suas operações a dois resultados possíveis: verdadeiro e falso.



George Boole (1815-1864), matemático inglês

A forma mais fácil de se entender esse tipo de lógica é utilizando exemplos do cotidiano. Se alguém lhe disser “eu entrei debaixo da chuva **E** me molhei” você não terá muitos problemas pra concluir que a pessoa está dizendo a verdade. Portanto, se alguém lhe disser “eu entrei debaixo da chuva **E NÃO** me molhei”, a sua primeira reação será de dúvida quanto àquilo ser verdade ou não.

Agora, vamos supor que a pessoa lhe diga: “abri meu guarda-chuva **E** entrei debaixo da chuva **E NÃO** me molhei”. As coisas mudaram um pouco de figura: um elemento a mais (o guarda-chuva) entrou na sentença e, a partir daí, ela se tornou verdadeira.

Percebeu que eu destaquei os **Es** e os **NÃOs** das sentenças? **E** e **NÃO** são *operadores* dentro da lógica booleana. As sentenças conectadas pelos operadores seriam os *operandos* da expressão.

Na álgebra convencional você teria algo assim:

$$2 + 5 = 7$$

O sinal **+** é o operador e **2** e **5** são os operandos da expressão que tem **7** como resultado. Agora observe a seguinte expressão booleana:

Eu entrei debaixo da chuva **E NÃO** me molhei = **FALSO**

E e **NÃO** são os operadores da expressão, enquanto “Eu entrei debaixo da chuva” e “me molhei” seriam os operandos. Como resultado, a expressão apresenta-se **FALSA**.

Perceba que **E** e **NÃO** são dois operadores independentes, tal como acontece no exemplo abaixo de álgebra comum:

$$2 \times (-5) = -10$$

O sinal negativo colocado à frente do cinco é um *operador unário* que altera seu valor. Já o sinal de multiplicação é um *operador binário* que está atuando sobre o número dois e o número cinco modificado pelo sinal negativo, dando um resultado de dez negativo.

Operadores unários, binários e ternários

Um operador é *unário* quando ele atua sobre apenas um operando, como aconteceu com o sinal negativo no exemplo anterior. O operador *binário* opera sobre dois operandos, como no sinal de soma em $2 + 2$. Já o operador *ternário* precisa de três operadores para funcionar; o único exemplo de operador ternário é o operador **SE – ENTÃO – SENÃO**, do qual creio que não falaremos tão cedo.

Para facilitar a vida na hora de escrever as sentenças, nós vamos trocá-las por letras. Então, se pegarmos a expressão:

Eu entrei debaixo da chuva **E NÃO** me molhei = **FALSO**

E trocamos “Eu entrei debaixo da chuva” pela letra A e “me molhei” pela letra B, teremos:

A E NÃO B = FALSO

Usando letras dessa forma, nós podemos começar a nos concentrar na estrutura das expressões, independentemente do conteúdo de seus operandos, podendo criar generalizações importantes.

Como saber se o resultado de uma expressão será verdadeiro ou falso? A primeira coisa que tem que se ter em mente é que os operandos por si só expressam verdades ou mentiras. Se eu considerar como verdades a pessoa ter entrado debaixo da chuva e ela ter se molhado, a expressão terá um resultado; mas, se eu afirmar que uma das afirmações (ou ambas) é falsa, o resultado será modificado. Portanto, lembre-se que os operadores booleanos manipulam apenas VERDADEIROS e FALSOs, resultando também apenas em VERDADEIROS e FALSOs.

Agora, antes de fritar um pouco mais o cérebro falando de operadores, vamos recapitular e aprofundar o conceito de LogicBrick.

1.2. LogicBrick: no pasa de un elemento booleano

Já tocamos no assunto dos LogicBricks no capítulo anterior, quando fomos criar a interatividade da cena da abóbora, mas agora chegou o momento de uma definição mais formal. LogicBrick, ou bloco lógico, é um elemento de uma expressão booleana que, colocado em série ou em paralelo com outros LogicBricks, dispara uma ou mais ações de um objeto específico.

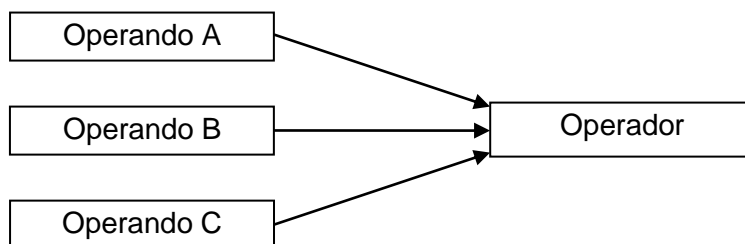
Hein?!

Vamos por partes; dizer que um LogicBrick é um elemento de uma expressão booleana significa dizer que ele é, na verdade, um operando ou um operador componente da expressão.

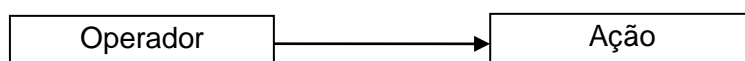
Todo LogicBrick possui um canal de entrada ou saída (ou ambos) por onde fluem os resultados booleanos, ou seja, VERDADEIRO e FALSO. Como todo operando booleano é um resultado VERDADEIRO ou FALSO por si só, os LogicBricks que representam operandos possuem canais de saída de informação; e, como todo LogicBrick que representa um operador deve receber as informações dos LogicBricks operandos e mandá-los adiante para que as ações possam ser disparadas, eles possuem canais de entrada e saída de informações; já os LogicBricks que representam as ações a serem realizadas pelo objeto possuem um canal de entrada de informações, para que possam ser avisados de quando devem ser disparados.

Os LogicBricks que representam operandos são os Sensors; os LogicBricks que representam operadores são os Controllers; e, por último, os LogicBricks que representam as ações dos objetos são os Actuators.

Os operandos de uma expressão booleana, dentro da filosofia dos LogicBricks, serão sempre colocados em paralelo, ligados em série com os operadores, como mostra o esquema a seguir:



Por sua vez, os operadores também serão ligados em série com as ações a serem realizadas:



Acho melhor dar um exemplo prático para garantir um pouco de sustância à conversa...

2. Tá bom, não é tão lógico assim...

Vamos colocar tudo o que falamos até agora em prática. Inicie o Blender e selecione com **BDM** o cubo que aparece na configuração inicial da 3DView (**Figura 2-1**). Ele será a nossa cobaia... :-)

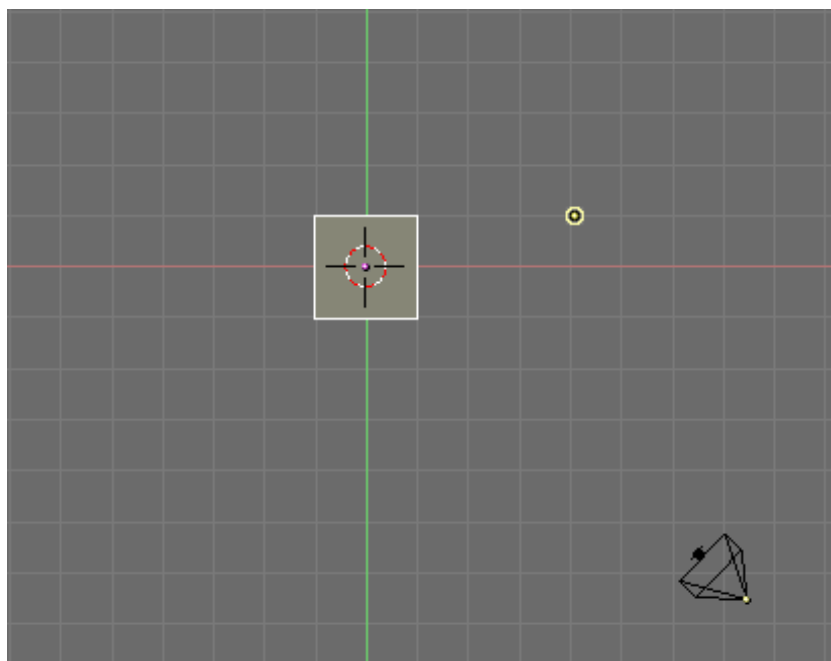


Figura 2-1. Mãos ao alto, cobaia!

Pressione **F4** para abrir o contexto Logic da ButtonsWindow (ou clique no Pac-Man roxo para abrir o contexto. Agora, cá entre nós, Pac-Man roxo? Não pega bem...) e clique no botão Add de cada coluna uma vez (**Figura 2-2**; não vou explicar de novo a disposição dos elementos dessa janela – ela já foi explicada no capítulo 2); a nossa primeira missão será fazer o cubinho se mover quando as setas direcionais forem pressionadas.

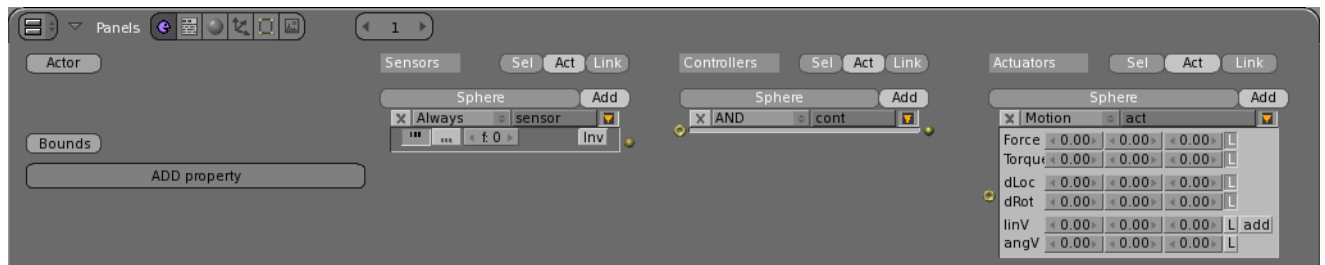


Figura 2-2. Um Sensor, um Controller e um Actuador adicionados.

Mude o Sensor para um Keyboard Sensor e deixe o Controller e o Actuador como estão; conecte o Sensor com o Controller e o Controller com o Actuador (**Figura 2-3**). Dentro do Keyboard Sensor, clique com **BEM** no botão Key e pressione **SETAESQUERDA** – a seta esquerda ficará mapeada no Keyboard Sensor (com o nome Leftarrow). Isso significa que, toda vez que o usuário pressionar **SETAESQUERDA**, esse Keyboard Sensor irá disparar um sinal VERDADEIRO, que será tratado pelos Controllers ao quais ele estiver conectado – no nosso caso, apenas o AND Controller irá receber o sinal. No Motion Actuator, clique com **BEM** no primeiro campo da linha dLoc (o campo do eixo X) e insira o valor -0.3. Posicione o ponteiro do mouse sobre a 3DView e pressione **PKEY** – a reprodução em tempo-real é iniciada. Se você pressionar **SETAESQUERDA**, o cubo deverá se mover para a esquerda.



Figura 2-3. Configuração do movimento para a esquerda.

Pressione ESC para sair do modo de reprodução. Insira mais um Sensor, um Controller e um Actuador e ligue-os entre si. Mude o Sensor para um Keyboard Sensor, da mesma forma que foi feito anteriormente, mas desta vez marque **SETADIREITA** como a tecla ativadora do Sensor e, no Motion Actuator, coloque o valor 0.3 no primeiro campo da linha dLoc (**Figura 2-4**).

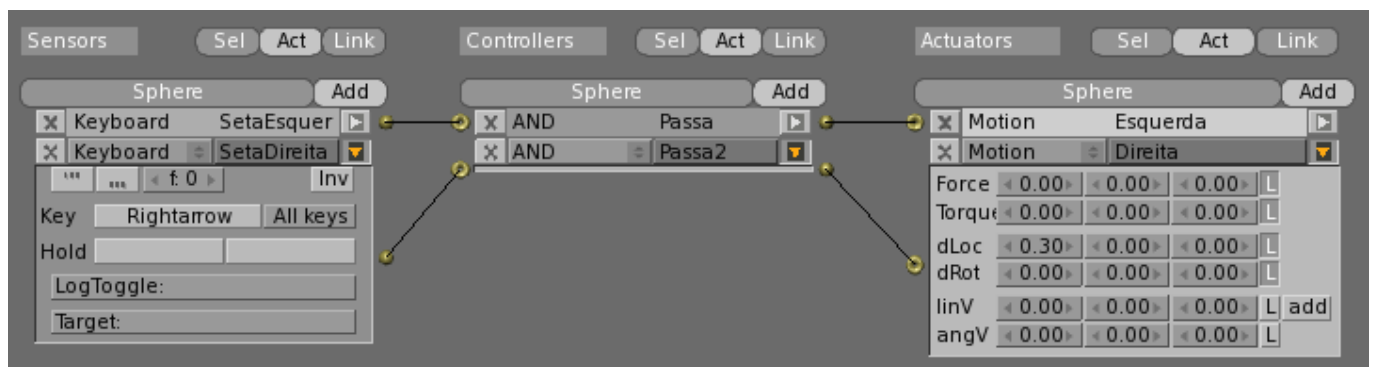
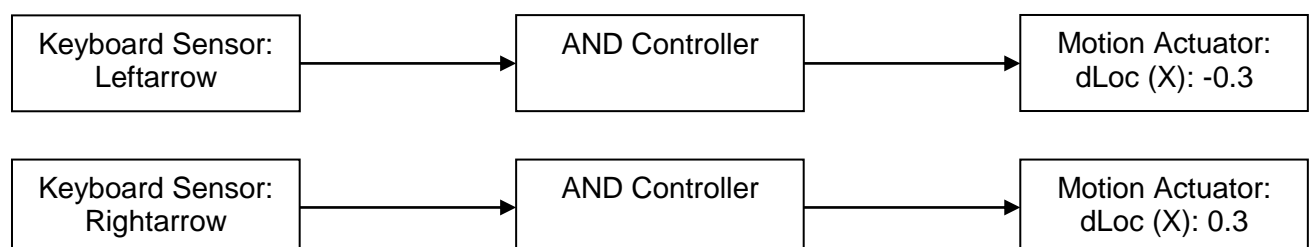
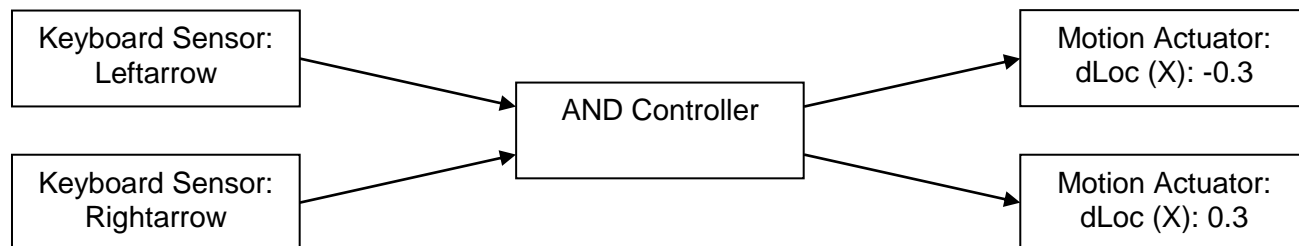


Figura 2-4. Configuração do movimento para a direita.

Se você reproduzir o seu projeto agora, as setas direita e esquerda deverão estar funcionando corretamente. Esquemáticamente temos o seguinte:



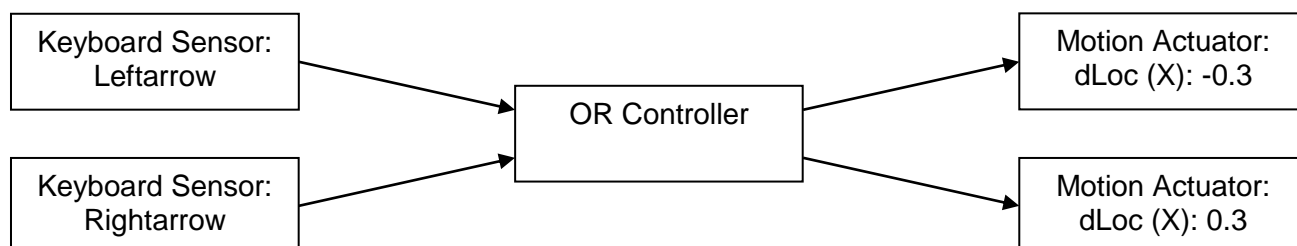
Agora alguém poderia sugerir: “Por que não fazer dessa forma:”



“Assim economizariamos um Controller!”

Sinto dizer, mas assim não vai funcionar. Como foi mostrado anteriormente, o operador AND (E, em português) só dispara o seu sinal se todos os seus operandos (os dois Keyboard Sensors, no caso) forem verdadeiros; portanto, para que os Actuators sejam disparados, as setas esquerda E direita devem ser pressionadas ao mesmo tempo!

“Sem problemas!”, alguém diria, “Então façamos da seguinte maneira:”



“Trocando o AND Controller por um OR Controller nós resolvemos o problema!”

Realmente, dessa forma o problema do pressionamento das teclas é resolvido, já que o operador OR (OU, em inglês) dispara seu sinal quando um ou mais de seus operandos forem verdadeiros, mas ainda resta outro problema: o Controller irá disparar ambas as ações sempre que uma das teclas for pressionada! Sabe o que isso significa? Que o cubo não sairá do lugar, pois

$$0.3 + (-0.3) = 0!$$

Mais uma vez, o grande erro aqui foi querer economizar em LogicBricks, sem levar em conta a lógica do objeto como um todo.

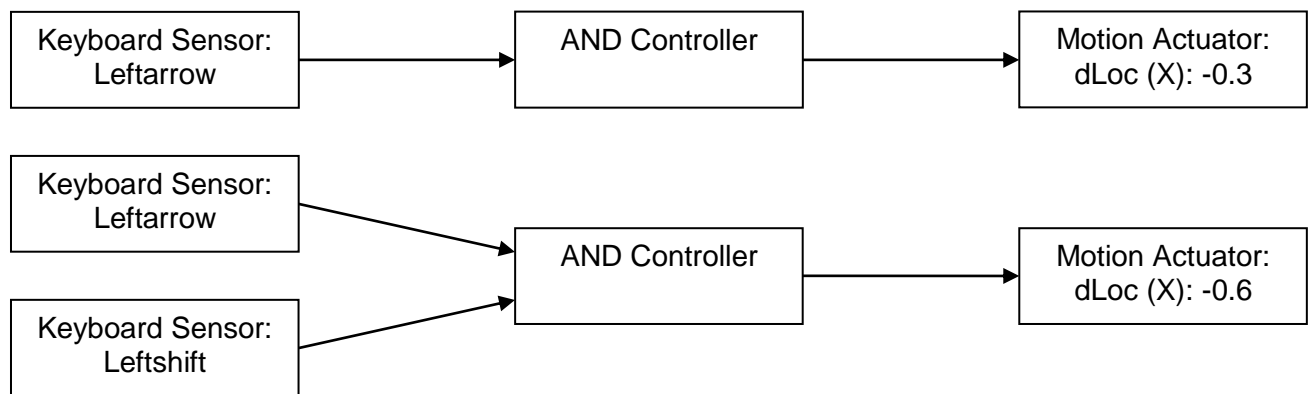
Como regra geral, procure não misturar os LogicBricks de ações independentes, mesmo que isso gere um pouco de redundância. Só pense em economizar LogicBricks quando você começar a ter certeza do que cada elemento faz!

Poderíamos adicionar LogicBricks para o pressionamento de **SETAACIMA** e **SETAABAIXO**, mas a lógica é a mesma; portanto, vamos seguir em frente.

2.1. Complicando um pouco a situação

Vamos adicionar mais um elemento à lógica de movimentação do cubo: consideremos que, quando o usuário pressionar **SETAESQUERDA** ou **SETADIREITA**, o cubo deverá *andar*; e, quando o usuário pressionar **SHIFT+SETAESQUERDA** ou **SHIFT+SETADIREITA**, o cubo deverá *correr*.

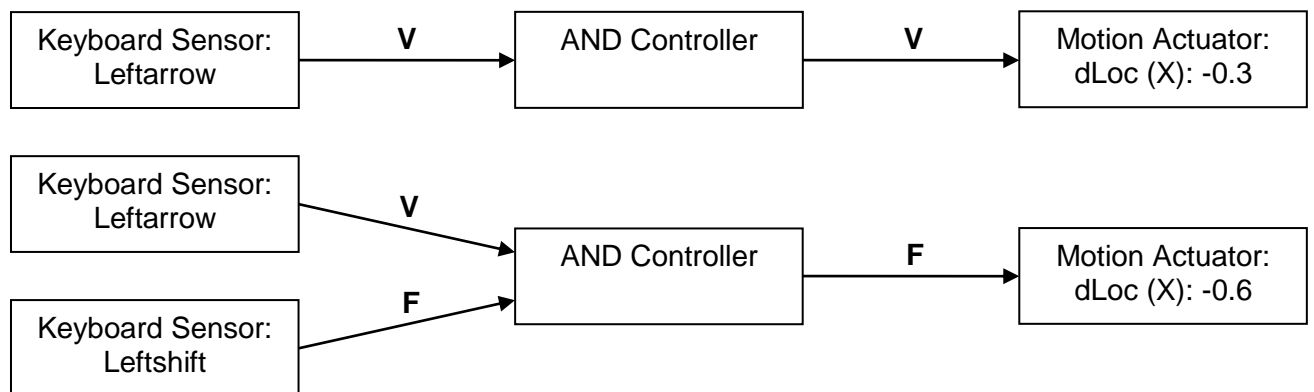
A diferença básica entre andar e correr está na velocidade; portanto, aí temos a primeira dica: o valor inserido no Motion Actuator da ação “correr” deverá ser maior do que aquele inserido na respectiva ação “andar”. Esboçemos, então, um primeiro esquema apenas para as ações “andar” e “correr” para a esquerda:



Aparentemente está tudo certo: as ações são independentes uma da outra e a ação “correr” realmente só será disparada quando **SETAESQUERDA** e **SHIFTESQUERDO** forem pressionados ao mesmo tempo (perceba que o Blender faz distinção entre **SHIFTESQUERDO** e **SHIFTDIREITO**). Os valores dos movimentos também parecem estar corretos: a ação “correr”, quando disparada, faz o cubo mover-se com o dobro de velocidade.

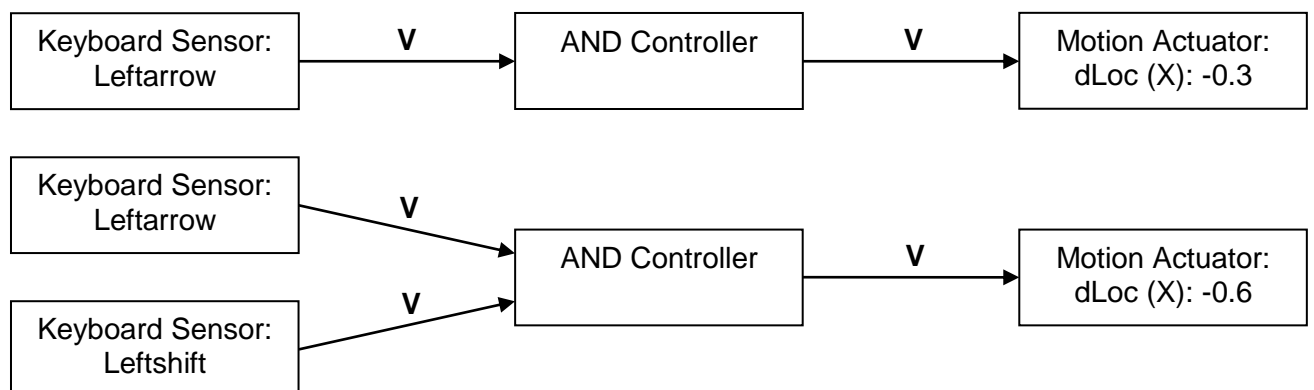
Mas existe um problema nesse esquema... você saberia dizer qual é?

Vamos fazer alguns testes com essa lógica. Se pressionarmos **SETAESQUERDA**, os valores transmitidos pelos LogicBricks serão os seguintes:



Observando o esquema, fica fácil perceber que a ação “andar” será disparada ao pressionarmos **SETAESQUERDA** (o único Sensor envolvido é verdadeiro, resultando em um sinal verdadeiro do Controller) e que a ação “correr” não será disparada (pois verdadeiro E falso resulta em falso).

Mas... o que acontecerá se pressionarmos **SHIFTESQUERDO+SETAESQUERDA**?

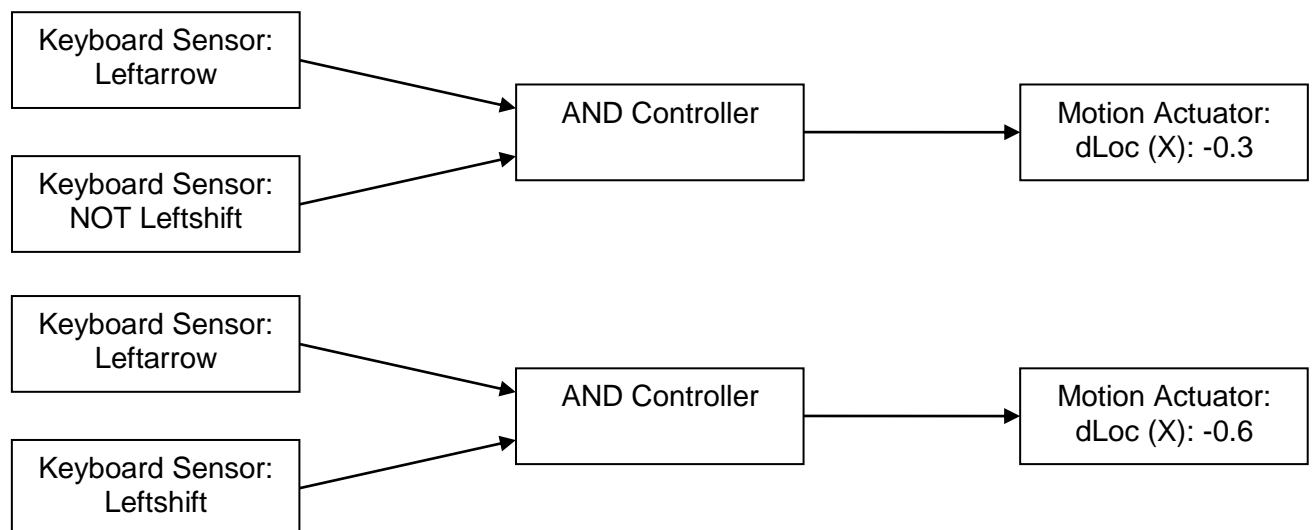


Repare que a ação “correr” é disparada, como esperado, mas a ação “andar” também foi disparada! Isso acontece porque a ação “andar” é disparada quando **SETAESQUERDA** está pressionado, *independente do estado das outras teclas*. Portanto, não interessa se **SHIFTESQUERDO** está pressionado ou não – ele não influi de maneira alguma sobre o estado da ação “andar”.

A principal consequência disso tudo é que as ações serão disparadas simultaneamente, resultando num movimento de -0.9 no eixo X (-0.3 da ação “andar” mais -0.6 da ação “correr”). “Espera um pouco”, diria o nosso amigo que está além de seu tempo, “Qual é o problema que existe nisso? No final das contas, não era isso que queríamos – que a ação ‘correr’ gerasse um movimento mais rápido? Se o problema é o valor final, basta reduzir o valor em dLoc de -0.6 para -0.3, que você terá -0.6 como resultado final!”

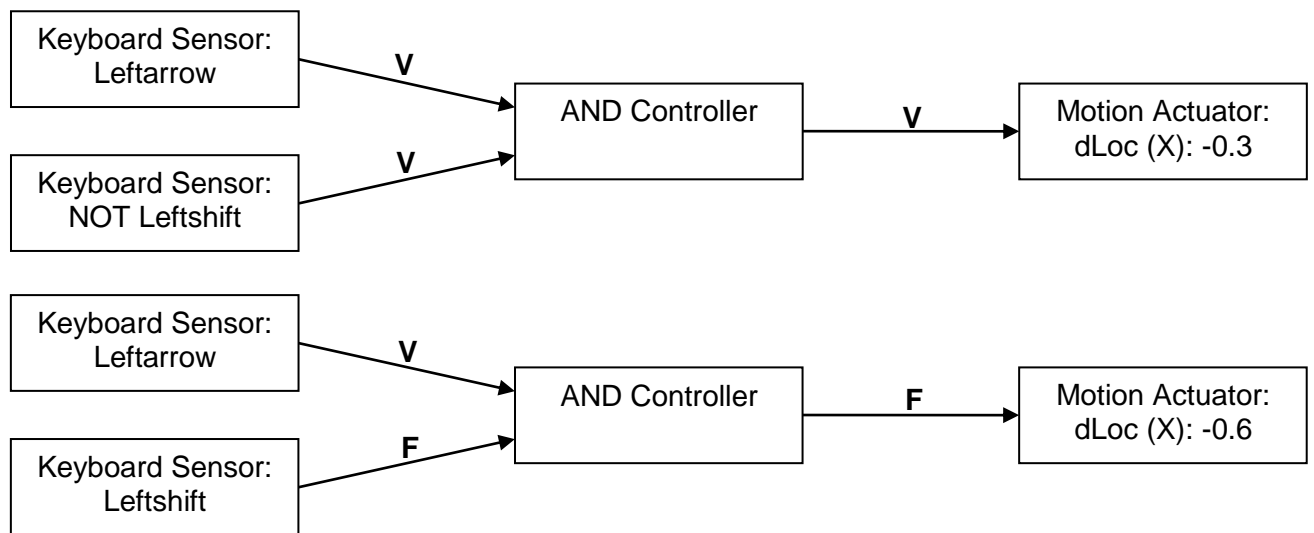
Olha, por um momento esse argumento até parece coerente, mas ele não leva em conta um princípio que eu já citei anteriormente: NUNCA, JAMAIS misture LogicBricks de ações independentes! A mistura, nesse caso, não é feita conectando-se fisicamente os LogicBricks, mas ela acontece na mistura dos resultados. Eu vou falar sobre isso no próximo capítulo, quando for definir Finite State Machines, mas posso adiantar o seguinte fato: faz sentido dizer que uma pessoa está andando E correndo ao mesmo tempo? Ou ela está fazendo um, ou está fazendo outro, não é verdade? Isso é bem diferente de dizer que a pessoa está andando e acenando ao mesmo tempo – nesse caso as duas ações podem acontecer ao mesmo tempo, mas deixemos essa discussão para adiante...

Aqui está uma forma de resolver a situação:

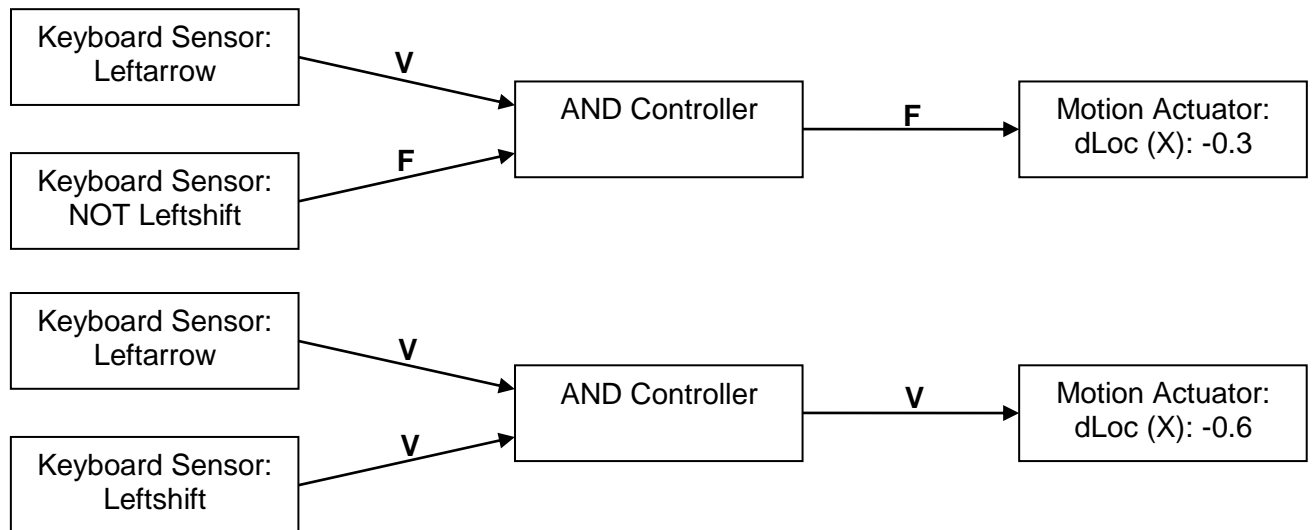


Aqui, um Keyboard Sensor foi adicionado à ação “andar”: NOT Leftshift. O NOT (NÃO, em inglês) irá inverter os valores disparados por esse Sensor; portanto, quando **SHIFTESQUERDO** estiver pressionado, ele irá disparar FALSO, e quando não estiver pressionado irá disparar VERDADEIRO.

Dessa forma, ao pressionarmos **SETAESQUERDA**, a situação dos sinais disparados será a seguinte:



E, ao pressionarmos **SHIFTESQUERDO+SETAESQUERDA**, o esquema ficará assim:



A figura a seguir mostra os LogicBricks inseridos no Blender de forma a refletir essa situação, mostrando as ações “andar” e “correr” para a esquerda; as ações para a direita são análogas às apresentadas aqui.

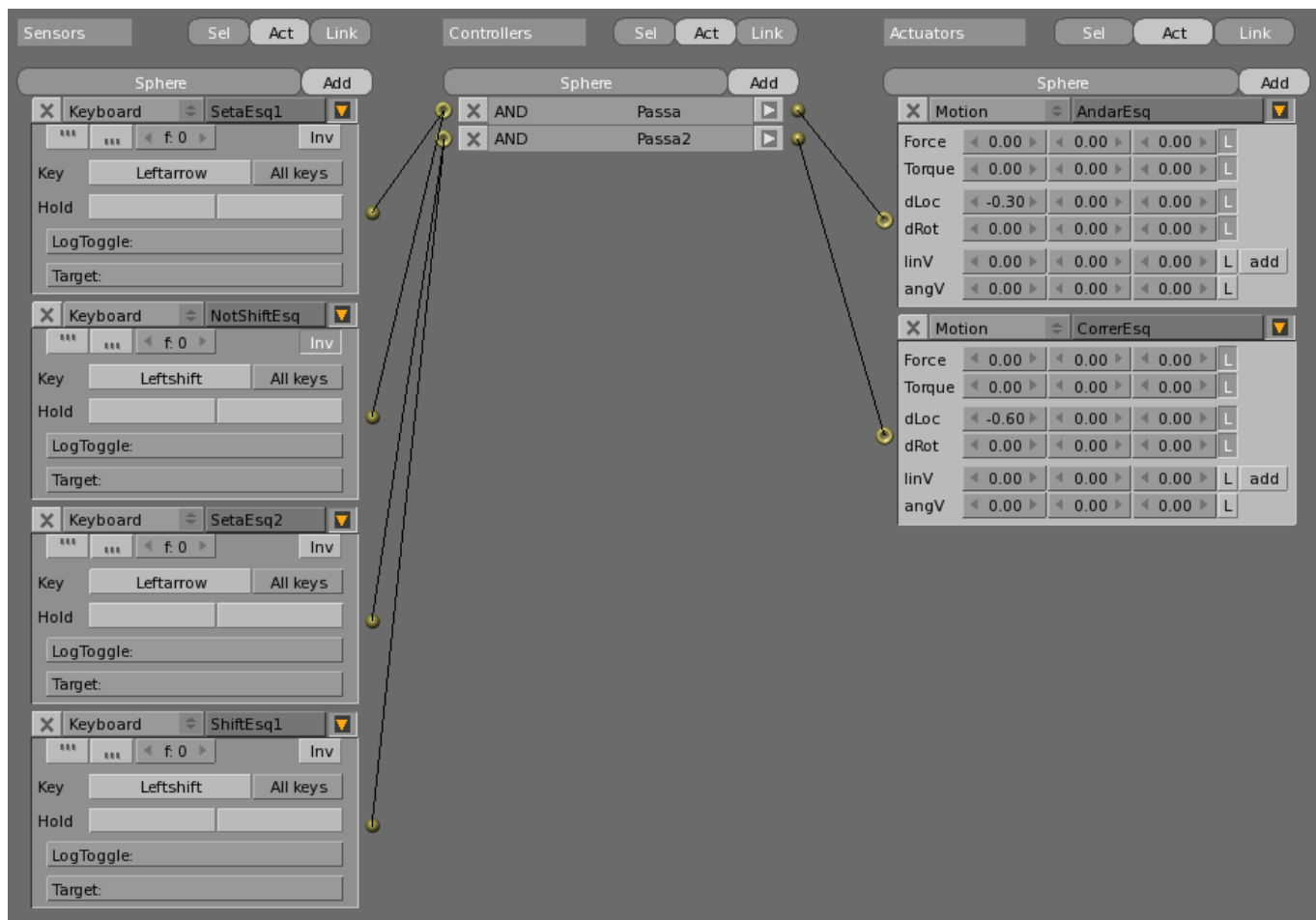


Figura 2-5. Configuração dos movimentos “andar” e “correr” para a esquerda.

Perceba que o botão Inv do Sensor NotShiftEsq está pressionado; isso significa que todos os sinais transmitidos por ele serão invertidos, ou seja, quando **SHIFTESQUERDO** estiver pressionado ele transmitirá um sinal FALSO e quando não estiver pressionado um sinal VERDADEIRO será transmitido.

3. Algumas tabelas-verdade

A seguir eu apresento tabelas que refletem os resultados de algumas operações booleanas simples. Essas tabelas costumam ser chamadas de *tabelas-verdade* e são apresentadas aqui apenas para que você tenha um resumo do funcionamento das operações básicas; não há a menor necessidade de decorá-las – é preferível que você entenda os conceitos que as regem.

Tabelas E												
Tabela 1					Tabela 2					Tabela 3		
V	E	V	=	V	V	E	NÃO V	=	F	NÃO V	E	NÃO V = F
V	E	F	=	F	V	E	NÃO F	=	V	NÃO V	E	NÃO F = F
F	E	V	=	F	F	E	NÃO V	=	F	NÃO F	E	NÃO V = F
F	E	F	=	F	F	E	NÃO F	=	F	NÃO F	E	NÃO F = V

Tabelas OU														
Tabela 1					Tabela 2					Tabela 3				
V	OU	V	=	V	V	OU	NÃO V	=	V	NÃO V	OU	NÃO V	=	F
V	OU	F	=	V	V	OU	NÃO F	=	V	NÃO V	OU	NÃO F	=	V
F	OU	V	=	V	F	OU	NÃO V	=	F	NÃO F	OU	NÃO V	=	V
F	OU	F	=	F	F	OU	NÃO F	=	V	NÃO F	OU	NÃO F	=	V

4. Conclusão

O segredo de se criar a interatividade de um jogo, seja no Blender ou em qualquer outro ambiente de desenvolvimento, é dominar a capacidade de raciocínio lógico. A lógica é a base das regras de qualquer jogo; por mais que os GDEs (Game Development Environments, ou Ambientes de Desenvolvimento de Jogos) evoluam e facilitem a vida dos designers e desenvolvedores, nunca, em momento nenhum, eles conseguirão suprimir a necessidade de se pensar logicamente. Portanto, não se deixe intimidar pelo assunto – se você realmente pretende ingressar nessa área, respire fundo e vá à luta! Não se deixe desestimular por uma ou duas batalhas perdidas – no final, o que conta não é vencer batalhas isoladas, mas a guerra como um todo!

Capítulo 4

Criando o Esqueleto de um Jogo

Antes de entrar na criação de um jogo propriamente dito, vamos construir um esqueleto onde o jogo será inserido. Esse esqueleto será formado por um menu inicial mais a cena onde será inserido o jogo. Parece pouca coisa? Então dê uma olhada no que você vai aprender nesse capítulo:

- O que é Finite State Machine (FSM);
- Aplicação do conceito de FSM na criação dos estados do jogo;
- Aplicação de textos como texturas;
- O que são propriedades e quais os tipos de propriedades existentes;
- Como acessar as propriedades de um objeto.

Está pronto? Então continue lendo!

1. Finite State Machine: outro nome para liquidificador

Finite State Machine, ou máquina de estados finitos, é um conceito importantíssimo dentro da área de inteligência artificial. Uma FSM é um objeto que possui um número limitado (ou seja, finito) de ações; tais ações recebem o nome de *estados*. Uma FSM está sempre realizando uma e apenas uma ação, ou melhor, sempre se encontra em um dos estados definidos.

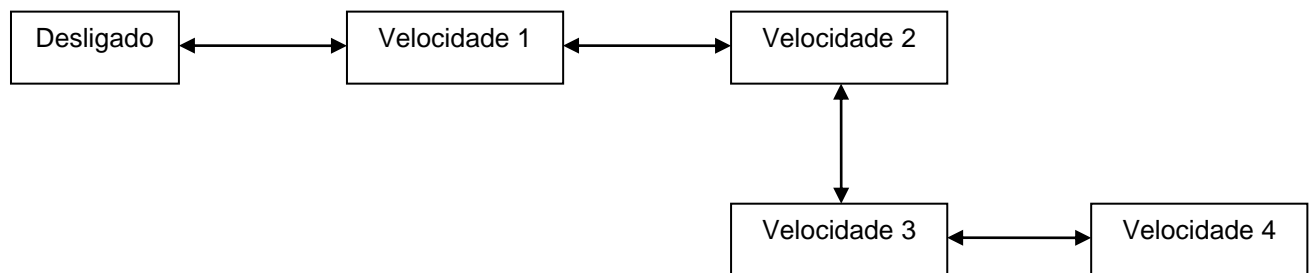
Para tornar o conceito mais claro, vamos usar um exemplo: o liquidificador. Sim, para surpresa geral da nação, o liquidificador é uma FSM! Ela possui um número finito de estados:

- Desligado;
- Velocidade 1;
- Velocidade 2;
- Velocidade 3;
- Velocidade 4.

Supondo um liquidificador simples de 4 velocidades, ele pode estar em algum estado além dos citados? Não dá pra ele estar meio ligado, assim como não existem as velocidades 1.35, 2.5, 3.97... Trocando em miúdos, uma FSM só pode fazer aquilo que foi projetada para fazer e nada mais.

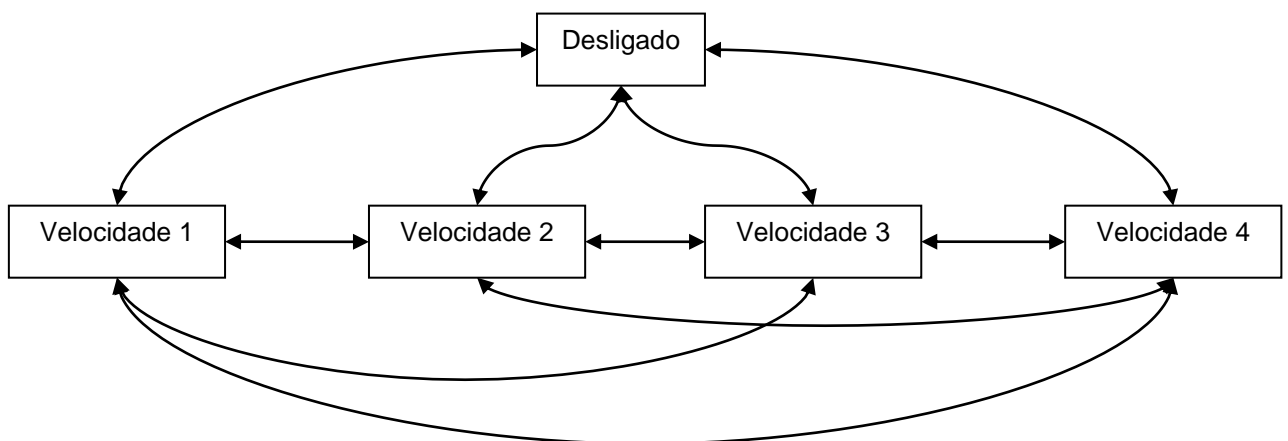
Agora vamos imaginar dois modelos de liquidificadores baseadas nessa FSM. O primeiro possui apenas uma alavanca, que deverá ser deslizada para a esquerda e para a direita para se alterar o seu estado; já o segundo liquidificador possui um botão para cada estado.

No primeiro caso, a troca de estados é linear: você passa do estado “Desligado” para o estado “Velocidade 1”; daí, você pode voltar para o estado anterior ou mudar para o estado “Velocidade 2” e assim por diante. O esquema dessa FSM ficaria mais ou menos assim:



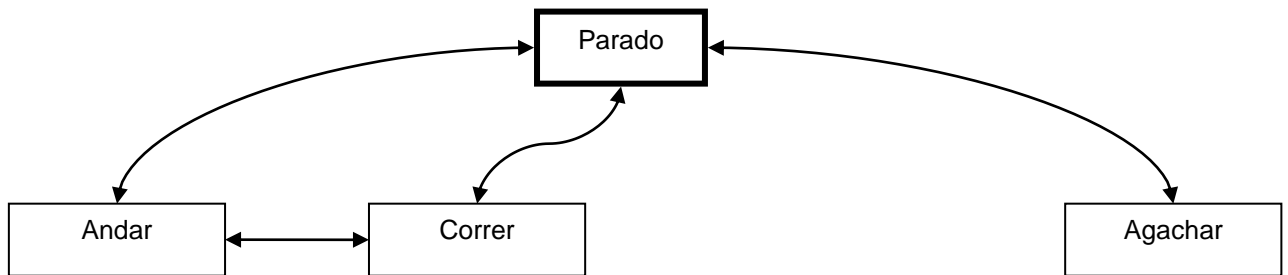
As setas indicam a direção em que é permitida uma mudança de estado. Nesse caso, todas as setas são bidirecionais, indicando que a mudança de estado pode ocorrer em ambos os sentidos.

Imaginemos agora o segundo caso. Assim que o liquidificador for ligado, ele poderá passar para qualquer velocidade imediatamente, sem ter que passar pelos outros estados. De forma inversa, ele pode ser desligado independentemente da velocidade em que estiver trabalhando. O esquema desse liquidificador é mais ou menos esse:



Entendeu? Não basta definir os estados para se construir uma FSM – você também precisa definir as *regras de transição* de um estado a outro. Com o mesmo grupo de estados, criamos duas FSMs completamente diferentes!

Uma FSM simpleszinha de tudo, representando um personagem, poderia ser esquematizada assim:



Definimos quatro estados para a FSM: “Parado”, “Andar”, “Correr” e “Agachar”. “Parado” é o chamado “estado padrão”: é o estado em que a FSM sempre iniciará. O estado padrão é uma espécie de estado neutro: normalmente ele poderá ser traduzido como “fazer nada”, ou seja, sempre que a FSM estiver ociosa, ela muda para esse estado.

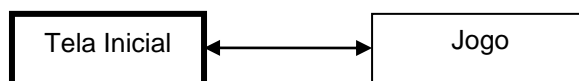
Os estados “Parado”, “Andar” e “Correr” são intercambiáveis entre si, ou seja, de qualquer um dos três você pode mudar para os outros dois. No entanto, “Agachar” deve ser acessado exclusivamente a partir de “Parado”. Por que isso? Decisões de implementação. Quem vai decidir qual estado pula para qual é você, o designer. Não existe resposta certa: o que existe é implementação adequada.

Essa FSM pode ser ampliada, remodelada, fatiada e processada; aqui, o que realmente importa é você entender a idéia básica de criação de uma FSM. Vamos, então, partir para um exemplo prático.

2. A estrutura global de um jogo é uma FSM!

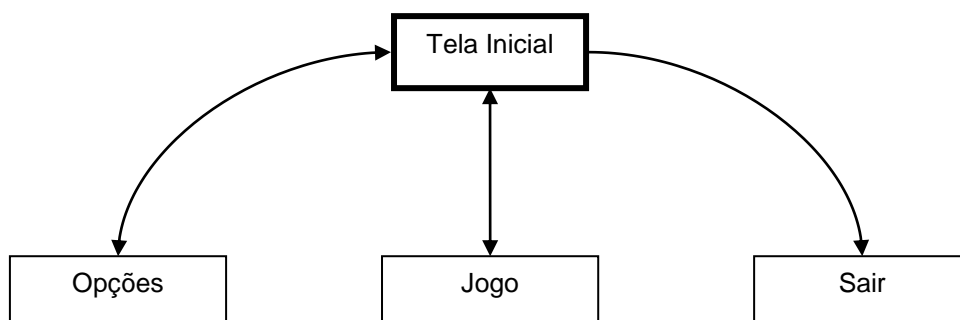
Nenhum jogo entra direto na ação, certo? Por mais que tenha apenas uma tela no começo com os dizeres “Press Start”, existe sempre um estado inicial global que direciona o jogador para determinadas opções.

Percebeu o uso da palavra “estado”? É isso aí, aquele menu inicial que qualquer jogo tem é uma FSM! O menu simples que foi descrito acima define a seguinte FSM:



Você pode estar se perguntando porque a seta que liga os estados é bidirecional. Bem, se o jogador perder ou desistir, ele precisa voltar para algum lugar, não é verdade? A resposta mais óbvia seria ele voltar para a tela inicial.

Vamos definir, agora, uma FSM mais complexa:



Essa FSM será o esqueleto do menu que construiremos no Blender. “Tela inicial” é o estado padrão; a partir dele três outros estados ficam acessíveis: “Opções”, “Jogo” e “Sair”. “Opções” é aquela tela que a maioria dos jogos possui, onde muda-se o número de vidas, a dificuldade, etc.; “Jogo” é o jogo em si; e “Sair”, quando acessado, fecha o jogo. Do estado “Jogo” você pode voltar para “Tela Inicial”, o mesmo ocorrendo com “Opções”.

Repare que a seta de “Tela Inicial” para “Sair” é unidirecional – afinal, não dá mais para voltar para nenhum estado uma vez que o jogo estiver fechado, certo?

Definida a FSM que servirá de modelo, vamos ver como construí-la dentro do Blender.

3. E liguem o liquidificador! ...quer dizer, o Blender!

Abra o arquivo Cap04-Esqueleto/Menu01.blend. O Blender deverá estar dessa maneira:

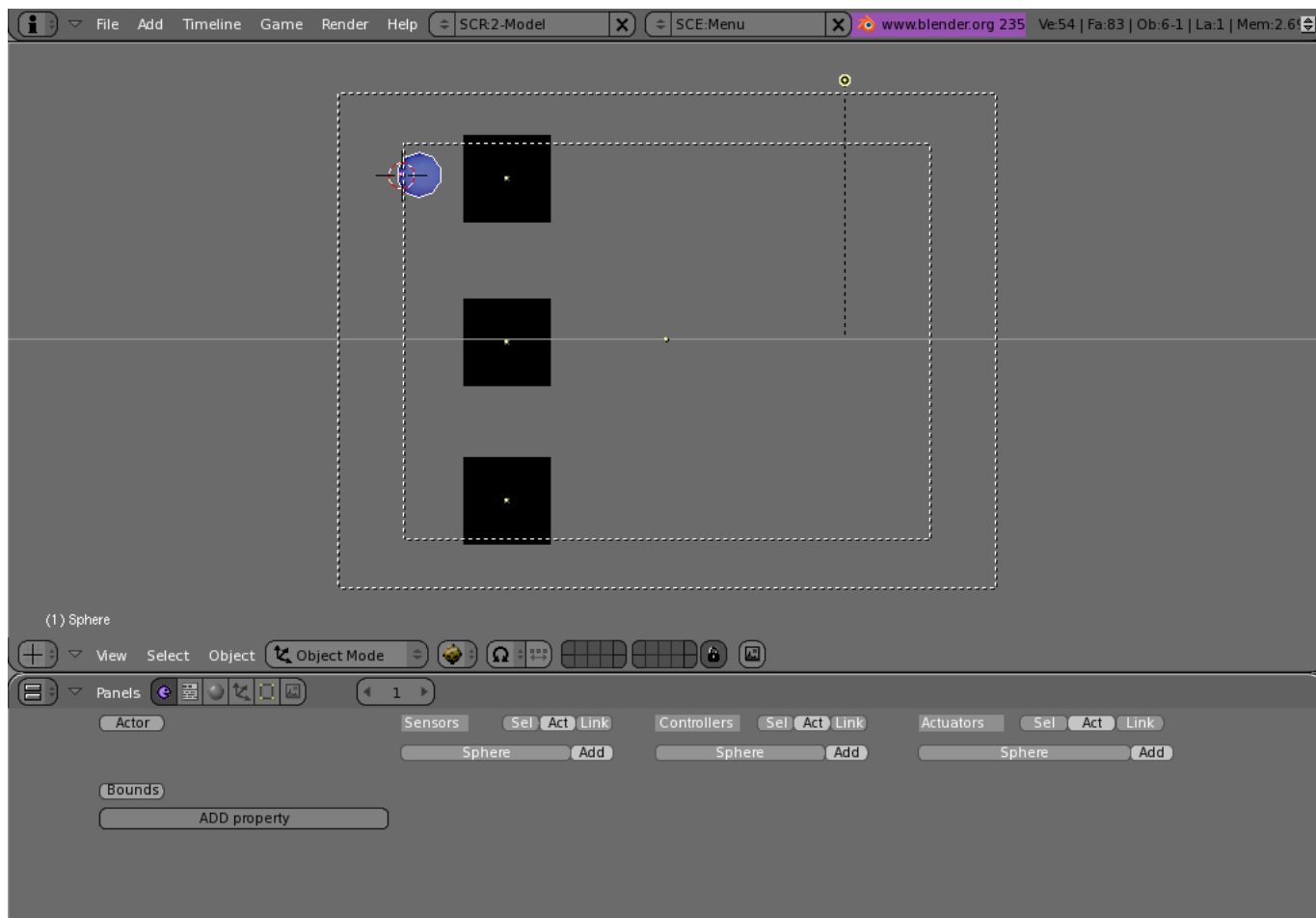


Figura 4-1. A tela inicial do arquivo Menu01.blend.

Cada plano representa um item do menu; a esfera à esquerda serve para indicar qual dos itens está selecionado num determinado momento. Nós iremos, agora, mapear o texto de cada item do menu.

Isso é uma coisa importante a se saber sobre o Blender. Ele não trabalha com texto de verdade; você tem de criar uma textura especial, onde estarão representados todos os caracteres que você irá utilizar. Essa textura é criada com um utilitário chamado *ftblender* – no entanto, explicar como utilizá-lo está completamente fora de escopo. Ao invés disso, eu já criei algumas fontes pra você utilizar quando precisar; elas estão dentro da pasta Cap04-Esqueleto/Fontes.

As texturas de fontes não passam de arquivos .TGA com tamanho de 512x512 pixels (atenção, isso é muito importante! Qualquer outro tamanho de textura vai causar problemas na formatação do texto!). Portanto, elas devem ser mapeadas nas faces de um objeto... *quase* da mesma forma que você mapearia uma textura normal. Existem dois detalhes a mais que você deve prestar atenção.

3.1. Preparando as superfícies para receber texto

Selecione um dos planos com **BDM** e pressione **FKEY**, alternando a 3DView para o modo UV/Face Select – por padrão, o plano é selecionado automaticamente. Pressione **F9** para mudar a ButtonsWindow para o contexto Editing. À direita na ButtonsWindow existe uma guia chamada *Texture face*; dentro dessa guia existem vários botões que ativam propriedades especiais das faces – permitir a aplicação de texturas, reagir à iluminação da cena, etc. Na terceira linha à direita existe um botão chamado *Text*; ao clicar nele, o Blender irá ativar algumas funções especiais para reconhecer a textura aplicada como texto.

Além de ativar esse botão, ative também o botão *Alpha*. Assim, o texto aparecerá com fundo transparente.

Saia do modo UV/Face Select pressionando **FKEY** novamente e repita esse procedimento para os outros dois planos.

3.2. Mapeando o texto nas superfícies

Divida a 3DView ao meio e mude a janela da direita para UV/Image Editor (**Figura 4-2**).

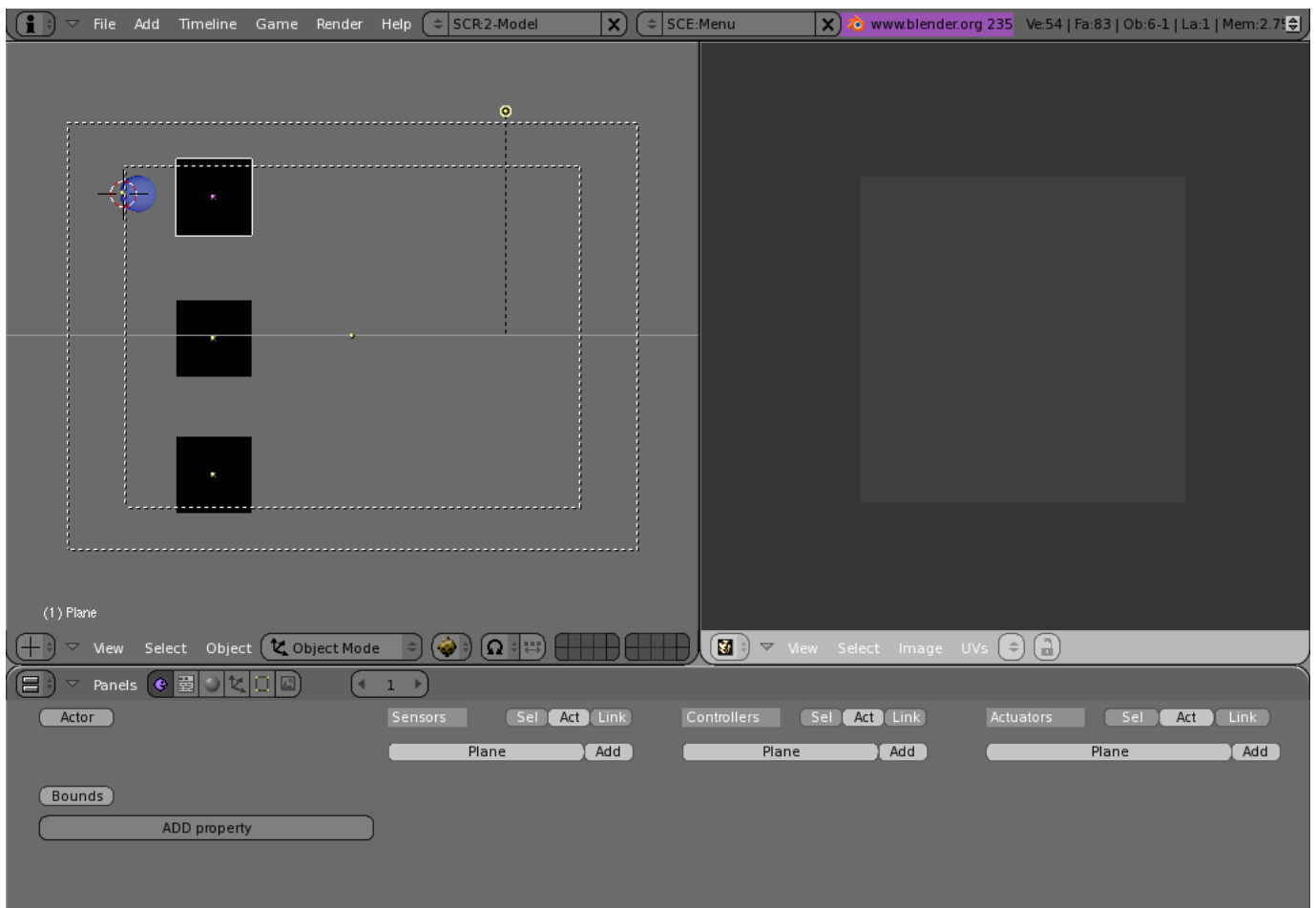


Figura 4-2. A tela dividida e devidamente modificada.

Selecione qualquer um dos planos e entre no modo UV/Face Select (**FKEY**). Certifique-se de que o plano está selecionado (seus vértices deverão estar aparecendo na UV/Image Editor) e, no cabeçalho da UV/Image Editor, abra o menu Image>>Open... (**Figura 4-3**).

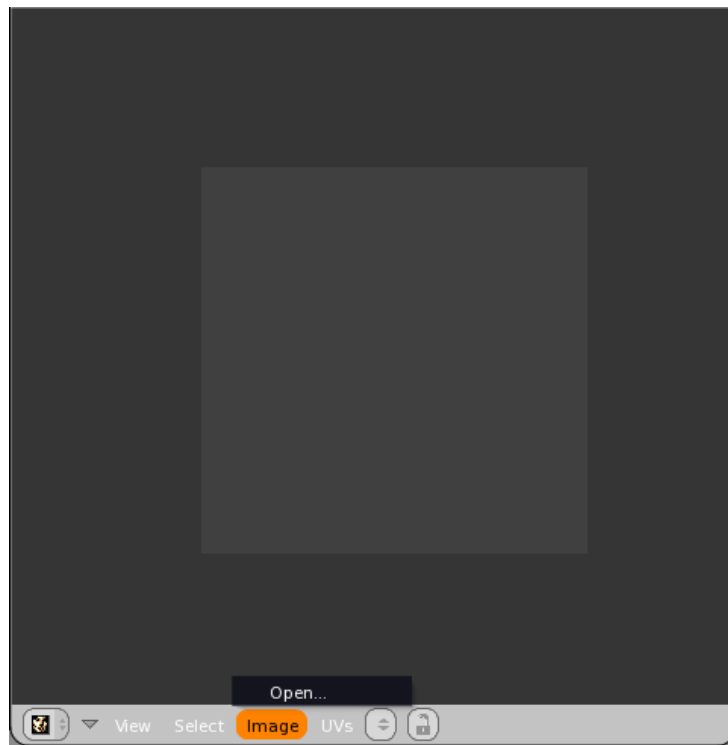


Figura 4-3. Localização do menu Image.

Vá até a pasta Cap04-Esqueleto/Fontes e selecione o arquivo verdana.tga. Repare em duas coisas: os caracteres foram dispostos em linhas e colunas dentro da imagem e o primeiro caractere é o arroba (@, **Figura 4-4**).

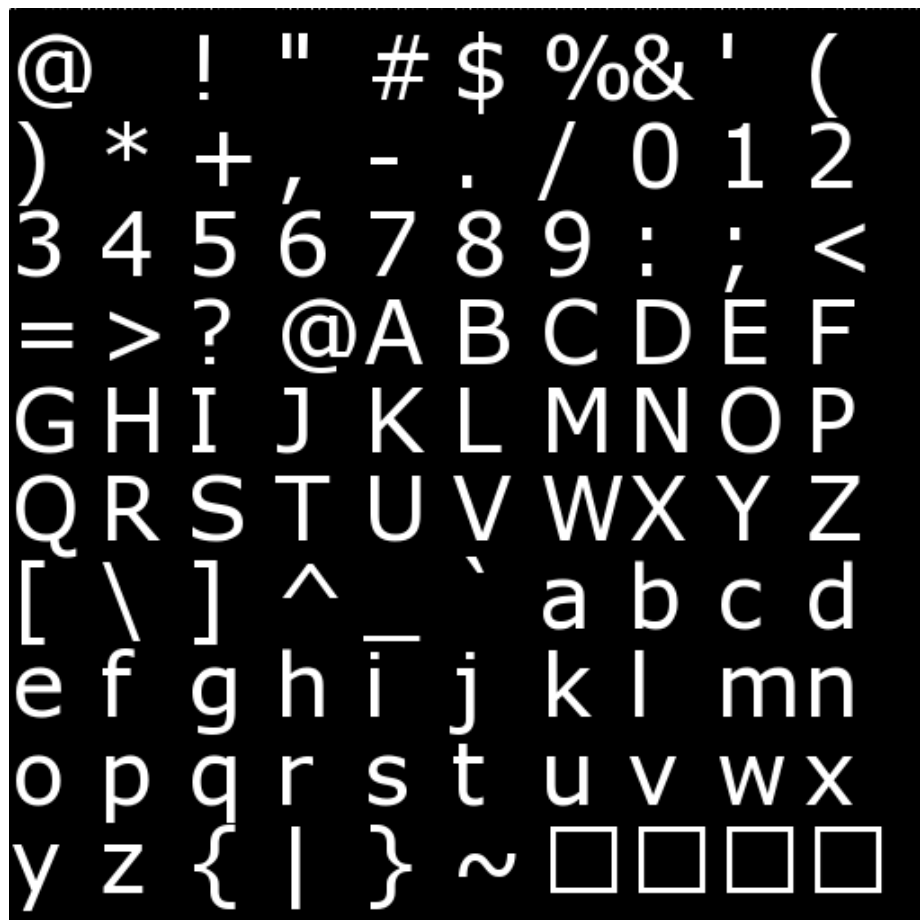


Figura 4-4. O arquivo verdana.tga.

Para mapear o texto em uma face você *deverá mapear o arroba* nessa face. Sim, é exatamente isso que você entendeu: não se deve mapear a textura inteira, apenas o arroba. Para isso, certifique-se de que o ponteiro do mouse está posicionado sobre a janela UV/Image Editor e pressione **AKEY** – todos os vértices devem ficar amarelos, indicando que estão selecionados. Usando **SKEY** (redimensionamento) e **GKEY** (movimentação), posicione os vértices exatamente em torno do arroba, como mostra a figura abaixo.

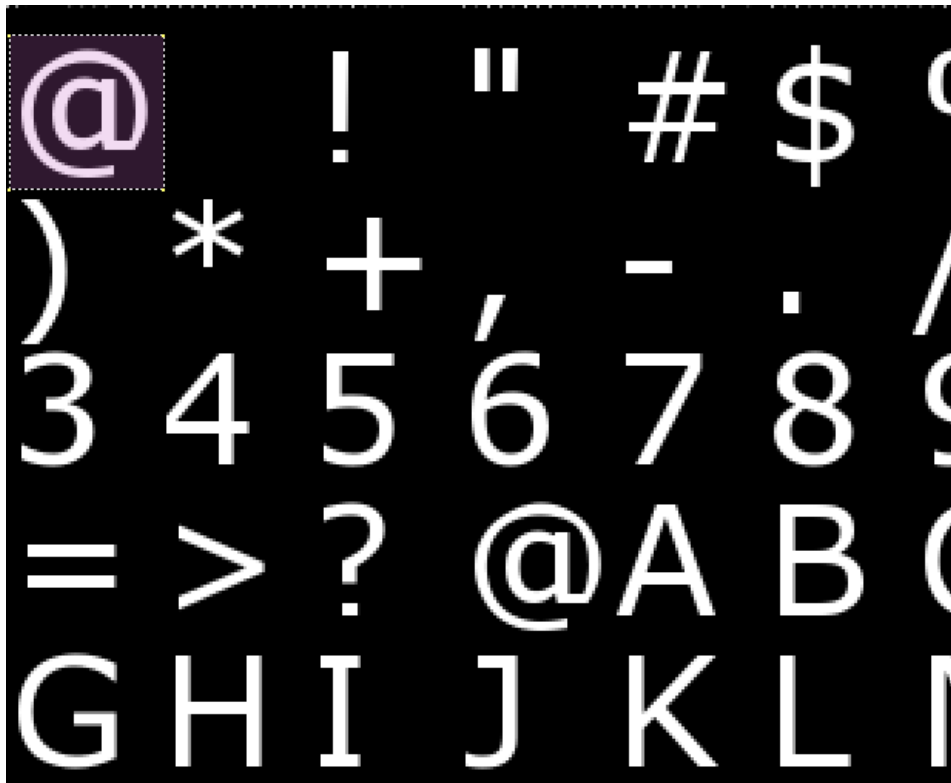


Figura 4-5. O arroba mapeado.

Se você pressionar **ALT+Z** na 3DView, verá que o arroba está mapeado errado (**Figura 4-6**). Para arrumar isso, certifique-se de ainda estar no modo UV/Face Select e pressione **RKEY**, selecionando a opção UV Co-ordinates (valeu pela dica, Evandro!). Faça isso três vezes para que o mapeamento seja corrigido.

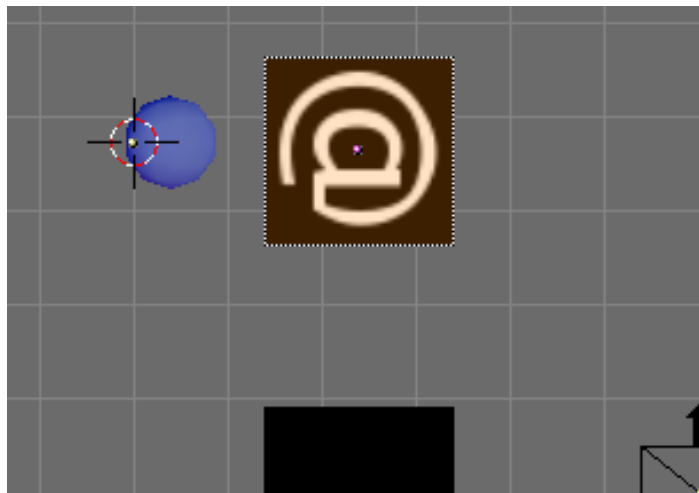


Figura 4-6. Tá bêbado, arroba?

Pronto? Podemos escrever algum texto agora? Bem, ainda não... Eu preciso explicar mais um conceito antes de chegarmos lá; portanto, vamos passar para o assunto *propriedades*.

Obs. 1: O mapeamento especial de texto só funciona em planos – portanto, nem pense em mapear texto em objetos 3D!

Obs. 2: Você deve estar se perguntando o porquê do arroba ser o primeiro caractere. Bom, eu também fico me perguntando isso até hoje... :) Provavelmente, ao mapear o arroba, o Blender fica conhecendo a largura e a altura máximas de um caractere, já que o arroba é o caractere mais gordinho de todos...

3.3. Propriedades: bananas são amarelas e marcianos são verdes

Se você abrir o contexto Logic na ButtonsWindow (F4), irá perceber uma coisa: falamos sobre Sensors, Controllers e Actuators, até brincamos com o botão Actor no capítulo 2, mas e quanto àquele botão discreto, pequenininho, chamado ADD property (Figura 4-7)?

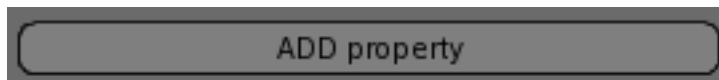


Figura 4-7. O botão ADD property, em toda sua glória!

Pois bem, chegou o momento de falar sobre ele. Enquanto os Sensors, os Controllers e os Actuators servem para definir as ações de um objeto, as propriedades ou *properties* servem para definir as características desse mesmo objeto.

Correr, pular, dançar, etc. são ações. Cor de pele, cor de cabelo, altura, idade, etc. são propriedades. Às vezes, uma ação pode usar uma propriedade para definir como ela será realizada. Poderíamos dizer, por exemplo, que a ação “andar” utiliza a propriedade “idade” para definir os seus detalhes de atuação.

Agora, perceba outra coisa: as ações, bem como as propriedades de um indivíduo, são parte constituinte daquele indivíduo e de mais ninguém; mesmo que vários indivíduos tenham as mesmas ações e as mesmas propriedades, a forma e os valores dessas ações e propriedades poderão variar bastante.

Se quisermos falar um pouco em programês (a língua dos programadores), as propriedades não passam de variáveis que um objeto possui. Essas variáveis podem conter vários tipos de informações: números, textos, valores booleanos, etc (Figura 4-8). No entanto, você deverá escolher qual desses tipos uma variável irá assumir. Uma vez definido o tipo de valor de uma variável (números, por exemplo) você não poderá inserir outro tipo de valor (textos, por exemplo), a não ser que você modifique o tipo de valor aceito pela propriedade.



Figura 4-8. Os tipos de valores que uma propriedade pode assumir.

Enfim, os tipos de valores aceitos pelas propriedades no Blender são:

- **Bool:** Aceita apenas dois tipos de valores, **True** e **False**.
- **Int:** Aceita números inteiros – 15, 314, 65535, etc.
- **Float:** Aceita números inteiros e decimais. Todo valor inserido deve conter a casa decimal, mesmo que seja um número inteiro; outra coisa que você deve lembrar é que o separador decimal é o ponto, e não a vírgula. Exemplos de valores Float: 15.5, 3.14, 5.0, 317.142567, etc.
- **String:** Aceita uma sequência de quaisquer caracteres. Exemplos de strings seriam “Teste”, “Olá mundo!”, “65536” (aqui o número será tratado como texto), etc. Na verdade, o Blender tem uma pequena limitação: ele só entende os caracteres comumente aceitos no padrão americano – ou seja, nada de acentos...
- **Timer:** O timer é um valor float especial, que é atualizado automaticamente para simular um cronômetro. Ele começa a funcionar assim que a cena em que está inserido é iniciada.

Agora que você já sabe o que são propriedades, vamos a um exemplo prático.

3.4. A propriedade Text

Muito bem, o último detalhe para se criar o mapeamento de texto é o seguinte: o plano mapeado obrigatoriamente deverá possuir uma propriedade chamada `Text`, e ela deverá ser escrita exatamente assim, com `T` maiúsculo e tudo o mais (Figura 4-9). O valor dessa propriedade é quem definirá o texto a ser escrito no plano!

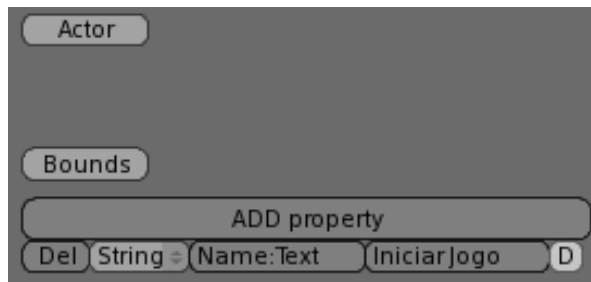


Figura 4-9. A propriedade `Text` já ajustada e com o texto inserido.

Vamos, então, concluir o assunto. Selecione qualquer um dos três planos (supondo que você já fez tudo o que foi dito até aqui com os três) e pressione **F4**, abrindo o contexto `Logic`. Clique no botão `ADD property` – uma nova propriedade é adicionada, com o tipo `Float` automaticamente selecionado. Renomeie-a para `Text` (terceiro campo da esquerda para a direita) e mude o tipo de valor de `Float` para `String`. Escreva um texto para a propriedade (quarto campo da esquerda para a direita) – “Iniciar Jogo”, por exemplo (não escreva as aspas; eu as coloquei só para deixar claro que estou falando de uma string). Se a `3DView` já estiver no modo de desenho `Textured` (**ALT+Z**) e no modo `Object`, você verá o texto na tela assim que confirmar a sua digitação.

Repita esses mesmos passos para os outros dois planos, escrevendo “Opcoes” em um deles (não podemos usar acentos, lembra?) e “Sair” no outro. No final, a sua tela deverá estar mais ou menos assim:

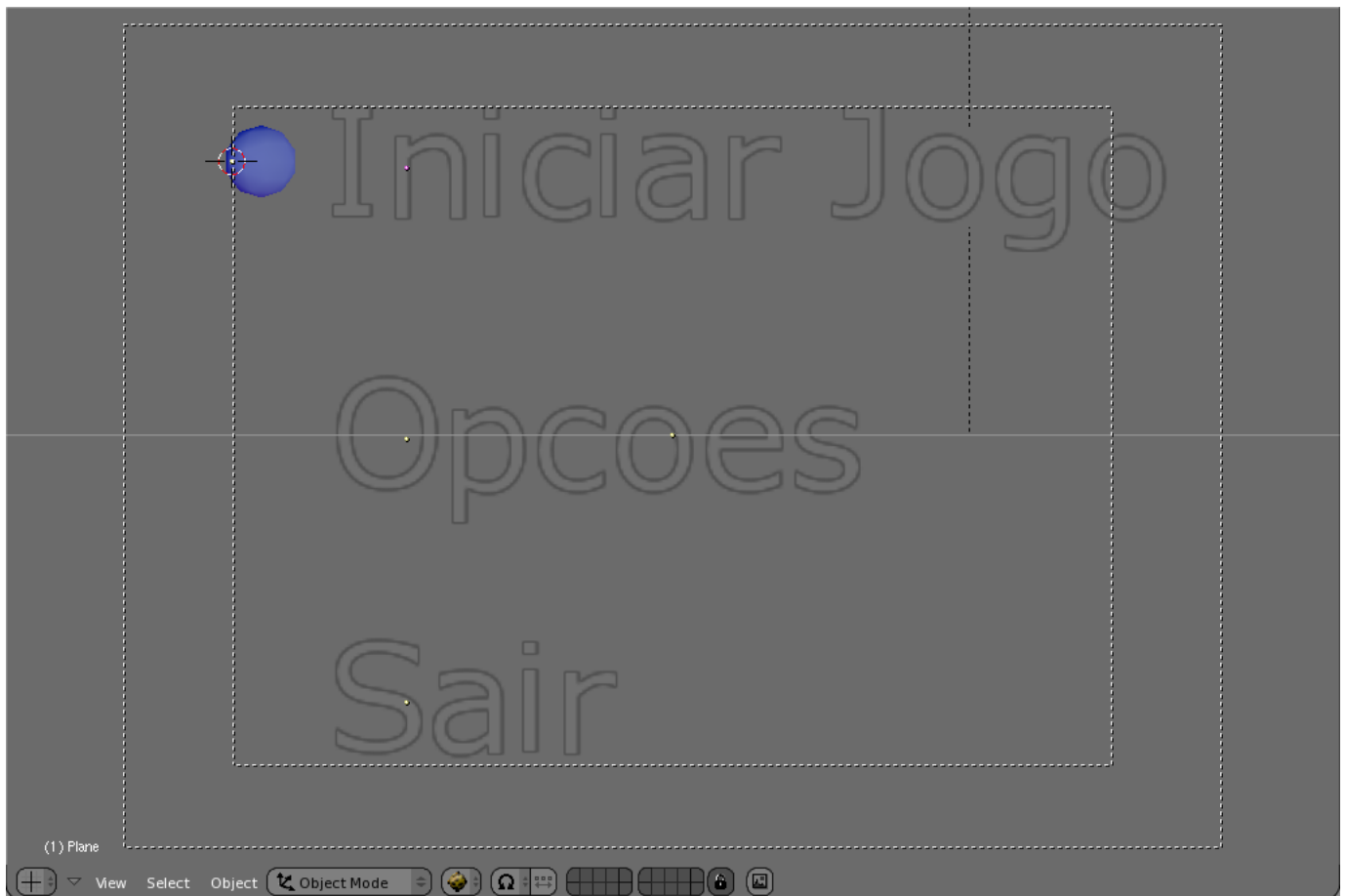


Figura 4-10. O menu completo.

Tendo tirado esse assunto do caminho, vamos prosseguir para o próximo tópico: a lógica em si.

3.5. Ligando os estados da FSM

Se você pressionar **SETAESQUERDA** e **SETADIREITA** dentro do Blender, verá que a esfera se move para cima e para baixo. Isso acontece porque ela já tem três quadros de animação pré-definidos – em cada quadro ela está ao lado de um dos itens do menu. Nós iremos fazer a esfera se mover para cima e para baixo dentro do jogo, controlando o quadro de animação em que ela se encontra.

Selecione a esfera com **BDM**, pressione **F4** para surgir o contexto Logic, adicione uma propriedade, cinco Sensors, três Controllers e três Actuators. Modifique-os e conecte-os conforme a figura abaixo:

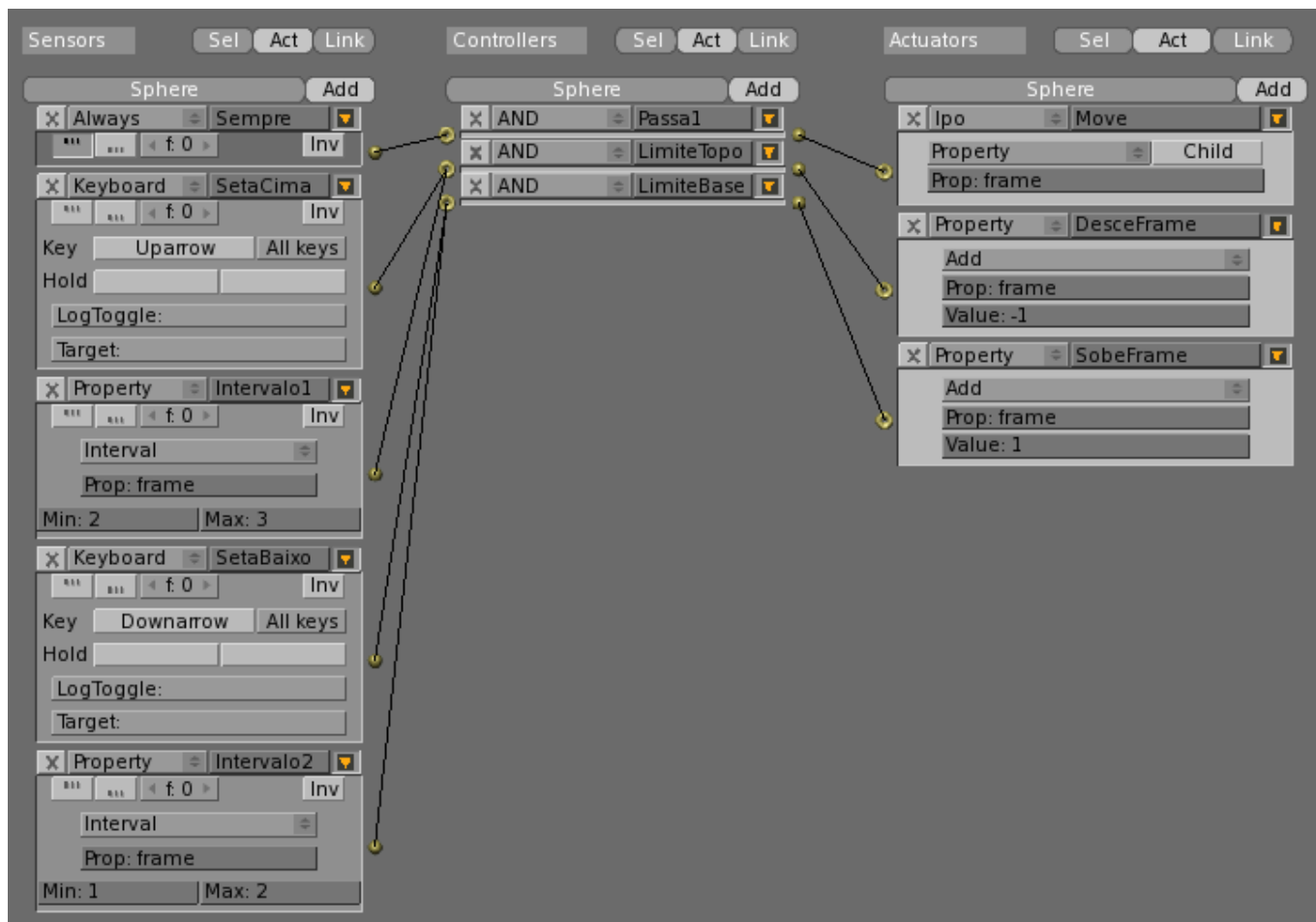


Figura 4-11. LogicBricks para posicionar a esfera no item de menu desejado.

A propriedade `frame` é quem irá modificar o quadro de animação atual da esfera: assim que o seu valor for modificado, o quadro de animação da esfera será modificado automaticamente.



Figura 4-12. A propriedade `frame` também deve ser inserida na esfera.

Vamos às explicações: o *Property Sensor* (Sensor de propriedade) dispara um sinal quando a propriedade especificada nele se encontra dentro dos valores especificados por ele. O Sensor `Intervalo1`, por exemplo, irá disparar um sinal VERDADEIRO quando o valor da propriedade `frame` estiver entre 2 e 3. O Sensor `Intervalo2` funciona de forma similar e os demais Sensors você já conhece.

O *IPO Controller* inserido define o quadro de animação a ser mostrado baseado na propriedade `frame`. Ele está ligado a um *Always Sensor*, o que significa que será executado uma vez a cada quadro do jogo.

Os dois *Property Controllers* (Controladores de Propriedade) manipulam o valor da propriedade especificada sempre que forem ativados. No nosso caso, ambos estão configurados para adicionar um determinado valor à propriedade (um deles adiciona -1, que seria o mesmo que subtrair 1).

O Controller *DesceFrame* só será ativado quando os Sensors *SetaCima* e *Intervalo1* forem verdadeiros, ou seja, **SETACIMA** deverá estar pressionado e o valor da propriedade *frame* deverá estar entre 2 e 3. Por que isso? Se não limitarmos as vezes que *DesceFrame* atua, o valor de *frame* continuará decrescendo indefinidamente, alcançando valores negativos e podendo resultar em situações indesejadas. O mesmo raciocínio vale para *SobeFrame* e os Sensors relacionados a ele.

3.6. Fazendo os itens do menu funcionarem

Eu só vou mostrar como fazer “Iniciar Jogo” e “Sair” funcionarem – fazer “Opcoes” funcionar é uma questão de repetir o que foi mostrado aqui.

Em primeiro, lugar, crie uma nova cena (escolha a opção *Empty*) e nomeie-a “Jogo” (lembre-se, o Blender faz distinção entre maiúsculas e minúsculas – muito cuidado!). Retorne à cena do menu e selecione a esfera novamente (se ela não estiver selecionada, claro). Adicione mais três Sensors, dois Controllers e dois Actuators e modifique-os conforme a figura abaixo:

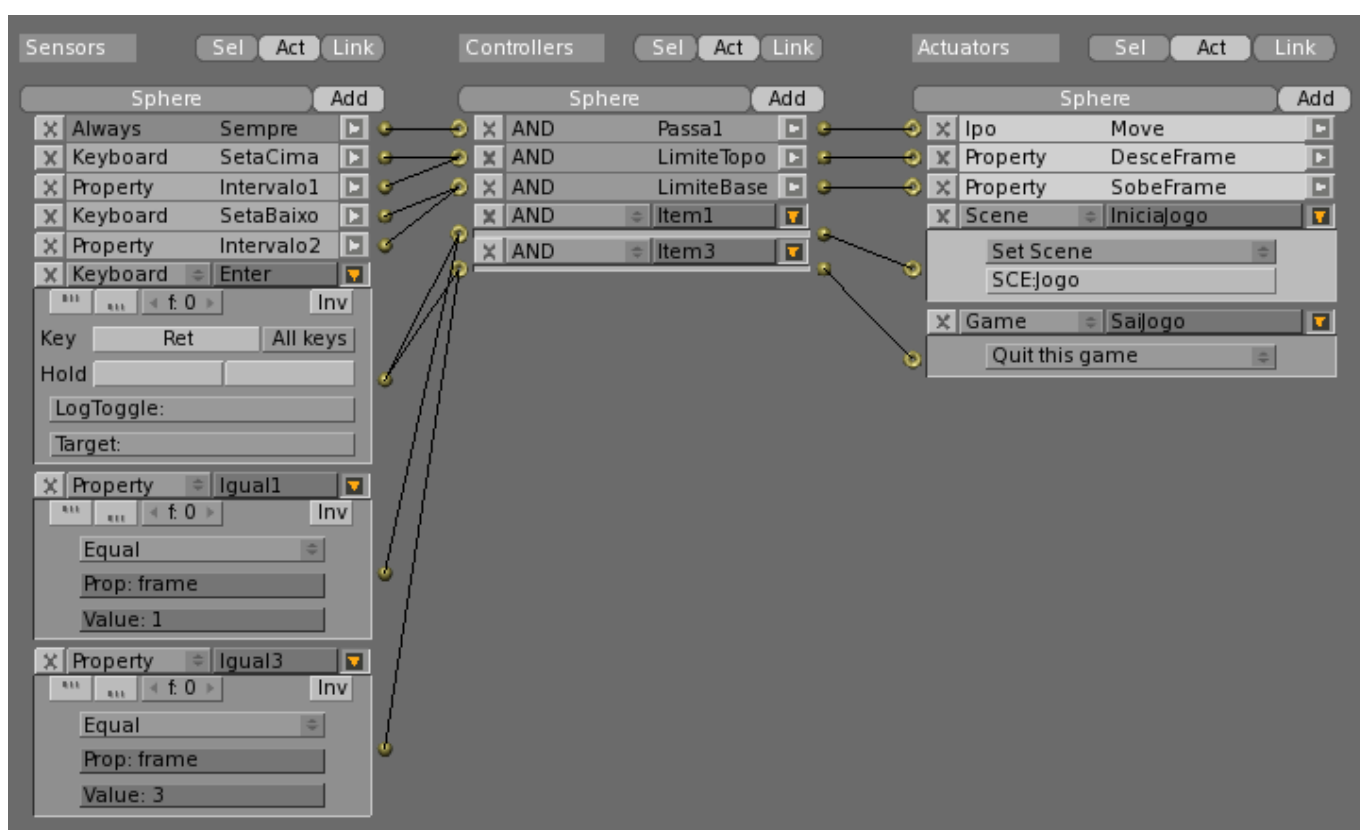


Figura 4-13. LogicBricks para acionar os itens de menu.

Repare que o Sensor *Enter* está ligado aos dois Controllers e cada um dos Property Sensors está ligado a um Controller. Dessa forma, o pressionamento da tecla **ENTER** será verificado junto com o valor de *frame*: se o seu valor for 1, significa que a esfera está posicionada à frente de “Iniciar Jogo” e, portanto, o Actuator *IniciaJogo* deverá ser executado. Por outro lado, se o valor de *frame* for 3, a esfera está posicionada à frente de “Sair” e será a vez do Actuator *SaiJogo* ser executado.

O Actuator *IniciaJogo* é um Scene Actuator, ou seja, manipula aspectos de ou relacionados a cenas. No nosso caso, utilizamos a subfunção *Set Scene*, que carrega a cena especificada. Já *SaiJogo* é um Game Actuator, que executa alguns comandos relacionados à execução do projeto como um todo; aqui nós executamos o comando *Quit this game*, que fecha o jogo e retorna ao sistema operacional (ou ao Blender, se você executar esse comando de dentro dele).

4. Conclusão

O esqueleto que foi criado aqui poderá servir como base de qualquer jogo que você for criar no Blender. O arquivo `MenuFinal.blend`, localizado na pasta `Cap04-Esqueleto`, representa o projeto final desse capítulo. Basta abrir a cena `Jogo` e começar a criar o seu jogo lá ou, se preferir, anexe objetos de outros arquivos `.blend` (janela `Append`, **SHIFT+F1**). No capítulo seguinte você vai aprender a anexar um *template* (modelo) para criar visualizações de arquitetura ou, enxergando mais além, para criar a base de jogos em primeira pessoa...

Capítulo 4

Um Joguinho de Nave Básico

Agora sim! Estamos progredindo a olhos vistos e agora chegamos em um dos pontos culminantes: o jogo! Claro que não vai ser o próximo grande hit do verão, não vai vender milhões de cópias (pra falar a verdade, eu acho que nem a sua avó compraria uma cópia dele por desencargo de consciência :-)) nem vai fazer de você uma estrela da noite pro dia, mas temos que começar por algum lugar, certo?

Aqui está o que você vai aprender neste capítulo cheio de pompas e circunstâncias:

- Alguns conceitos básicos de animação;
- Como coordenar animações pré-definidas com os comandos do jogador;
- Como atirar;
- Como marcar e mostrar a pontuação do jogador.

Ok, já comecei mentindo, que coisa feia... Na verdade, o capítulo mostra dez vezes mais coisas do que acabei de citar, mas é bom não assustar de cara... :-)

Vamos lá, meu jovem padawan! Estamos entrando na reta final!

1. Preparativos

Vamos começar o nosso jogo tentando traçar um plano de ação teórico. É importante não se esquecer desse passo – ele será a nossa linha-guia quando estivermos desenvolvendo o jogo.

Para tanto, vamos usar um pouco do material das aulas teóricas para criar uma documentação bem simples.

1.1. Premissa e Objetivo do Jogo

A premissa do jogo é um tanto clichê: hordas de alienígenas invadiram o planeta Terra; você é a última linha de defesa – e a última esperança – dos terráqueos!

O objetivo do jogo é abater as naves alienígenas, não deixando-as passar por você. Para cada nave que passar, uma grande cidade será destruída; são cinco cidades no total: se as cinco forem destruídas, o planeta terá sido irreversivelmente conquistado e o jogo termina.

1.2. Um Jogador versus Jogo

O jogo é um típico *shooter* horizontal. E, como na maioria dos shooters, você deverá enfrentar sozinho a invasão hostil. O jogo poderia ser estendido para abarcar também uma versão para dois jogadores, mas vamos tentar manter as coisas da forma mais simples possível.

O jogador será representado por uma navezinha de qualidade duvidosa, mas fazer o quê? Um dia eu ainda melhoro as minhas habilidades de modelagem... :-)

1.3. O Campo

O campo do jogo é o céu do planeta Terra. A direção a ser tomada pelo jogador é definida pelo próprio jogo.

Como estamos fazendo um jogo de baixo orçamento (orçamento zero, na verdade :-), iremos lançar mão de alguns truques para conseguir um campo visualmente atrativo – mas falaremos disso daqui a pouco.

1.4. Recursos

O jogo, nessa versão preliminar, terá os seguintes recursos:

- Os tiros da nave do jogador
- A energia do jogador;
- As cidades a serem protegidas;
- A pontuação do jogador.

1.5. Conflitos

O jogo não tem obstáculos, apenas oponentes. Nessa versão simplezinha, haverá apenas um tipo de inimigo, que fará uma coreografia bem definida.

Numa versão mais sofisticada, poderiam ser colocados vários inimigos diferentes, com diferentes tipos de coreografias. Isso acabaria gerando um dilema: quais naves destruir antes, para ter certeza de que nenhuma irá passar por você?

1.6. Processos

As ações do jogador se resumem a:

- Mover-se pela tela;
- Atirar.

Os oponentes terão apenas uma ação:

- Mover-se pela tela.

Já os triggers do jogo são:

- Quando um tiro do jogador colidir com uma nave inimiga, ela deverá explodir;
- Quando o jogador colidir com uma nave inimiga, ambos deverão explodir e uma vida será subtraída;

- Quando o jogador se aproximar de uma das bordas da tela, o seu movimento naquela direção deverá parar.
- Se uma nave ultrapassar a borda esquerda da tela, uma cidade será destruída;
- Se o jogador perder todas as vidas ou se todas as cidades forem destruídas, o jogo termina;
- Quando uma nave for destruída, deve-se aumentar o placar do jogador em 10 pontos.

1.7. Resultado

Como o jogo assume o padrão “um jogador *versus* jogo”, o jogo acaba quando o jogador perde para o próprio jogo; e, como o jogo não tem um final definido, a pontuação alcançada pelo jogador é a única forma de medir o resultado de uma partida.

1.8. Regras

As regras formam um compêndio das inter-relações dos elementos formais do jogo. Portanto, já temos boa parte das regras – só falta formalizá-las:

- O jogador movimenta a sua nave com as setas direcionais;
- A barra de espaço dispara tiros;
- O jogador deverá atirar nas naves inimigas, não permitindo que elas ultrapassem o canto esquerdo da tela;
- Para cada nave que ultrapassar o canto esquerdo, uma cidade será destruída;
- As cidades estão indicadas no canto inferior esquerdo da tela;
- Se o jogador colidir com uma nave inimiga, ambas as naves serão destruídas; o jogador perderá uma vida.
- As vidas estão indicadas no canto inferior esquerdo da tela, ao lado do número de cidades;
- Se as vidas do jogador acabarem ou todas as cidades forem destruídas, o jogo termina.

2. Preparando o Ambiente

Agora eu pretendo falar rapidamente de uma área importante na criação de um jogo: a texturização. Para tanto, abra o arquivo `Cap05-JogoNave\JogoNave2-0.blend`.

Observação

A numeração dos arquivos na pasta `Cap05-JogoNave` irão seguir a estrutura do capítulo. O arquivo acima é o `JogoNave2-0.blend` porque estamos no tópico 2 do capítulo. Se estivéssemos no tópico 3.1, por exemplo, o arquivo se chamaria `JogoNave3-1.blend`. Na verdade, talvez esta seja a única vez que irei pedir a você que abra um arquivo. Se você perder o seu arquivo ou quiser consultar uma parte específica do jogo, observe o tópico que está falando o que você procura e abra o arquivo com a numeração anterior, já que a numeração indica *até onde o projeto está pronto*.

A partir da visão do topo, você deverá estar visualizando a nave do jogador, um cubo, dois cilindros concêntricos, um ponto de luz, um objeto `Empty` (Vazio) e uma câmera. A nossa missão agora será texturizar os dois cilindros.

Qual a função dos dois cilindros? Eles servirão como cenário de fundo para o jogo. A idéia é mais ou menos a seguinte: as texturas aplicadas em ambos os cilindros deverão ter suas bordas laterais imperceptíveis, ou seja, a borda esquerda deverá casar perfeitamente com a borda direita e vice-versa. Assim, podemos criar um cenário infinito sem muitos recursos.

Usar um cilindro como cenário de fundo não é a técnica mais utilizada, mas, por uma questão de imprecisões existentes no Blender, ela passa a ser a nossa melhor opção.

2.1. Texturizando os cilindros

Muito bem. Selecione o cilindro externo (**BDM**) e pressione **FKEY** – a 3DView deverá ter mudado do *Object Mode* para o *UV/Face Selection Mode*. Nesse modo você poderá selecionar as faces do cilindro, individualmente ou em grupo, e atribuí-las uma imagem como textura. Você pode ter várias imagens atribuídas a um mesmo objeto, mas apenas uma imagem por face.

Antes, divida a 3DView (aproxime lentamente o ponteiro do mouse da borda inferior ou superior da 3DView, clique com **BDM** e selecione a opção `Split View`; selecione o ponto onde você quer que a tela seja dividida e confirme com **BEM**); agora, mude a 3DView da direita para uma janela UV/Image Editor (clique com **BEM** no primeiro botão à esquerda do cabeçalho da vista, **Figura 5-1**).

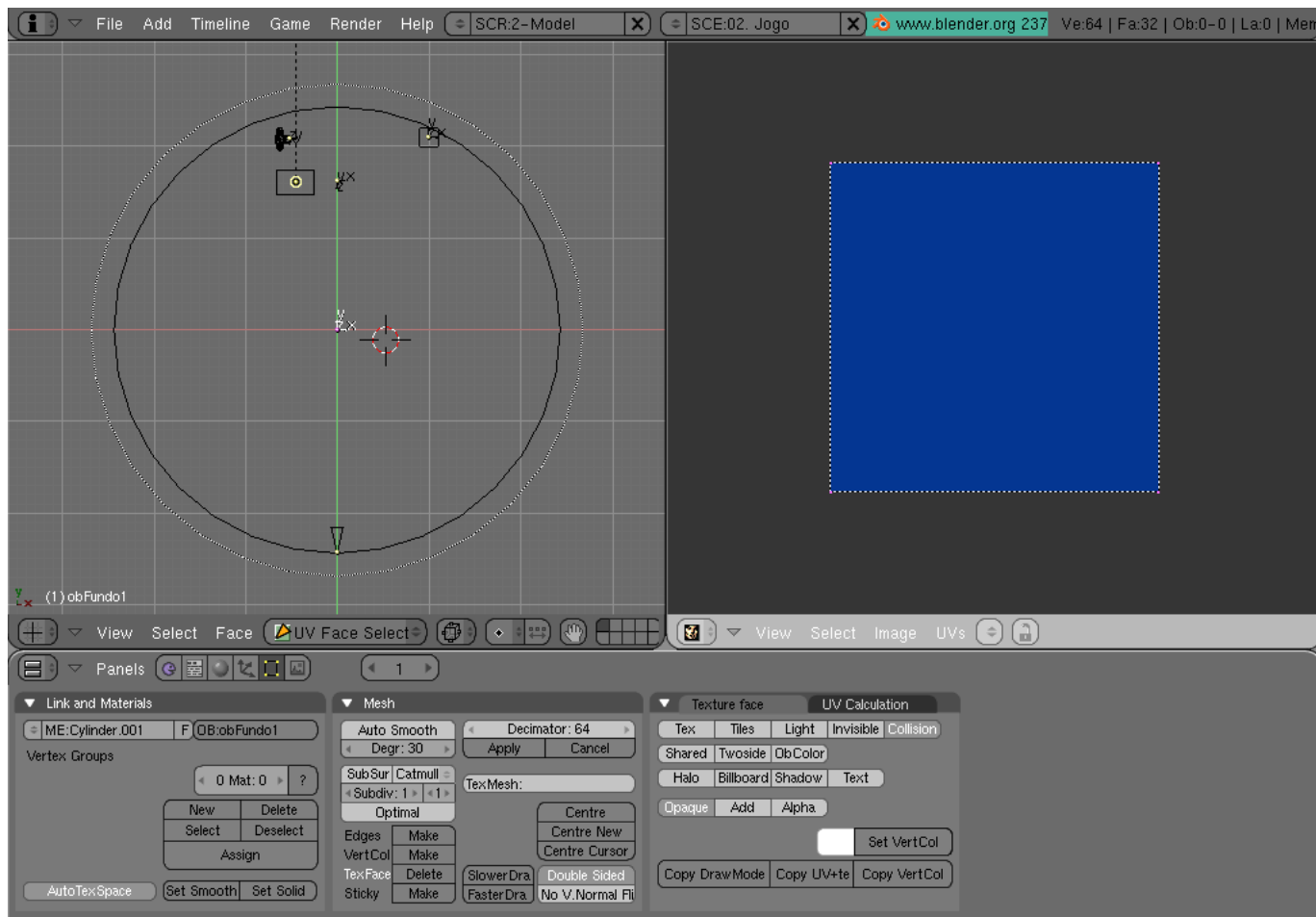


Figura 5-1. Divisão inicial da tela.

A nossa intenção é mapear apenas as faces laterais – aliás as faces do topo e da base do cilindro foram apagadas, já que nunca aparecerão no nosso jogo. Mas, antes de selecionarmos as faces para o mapeamento, existem alguns detalhes que precisam ser colocados em pauta.

Se a face selecionada for quadrada, aparecerão quatro vértices (indicado pelos pontos rosas em torno do quadrado azul mostrado na **Figura 5-1**), indicando onde cada vértice da face está mapeado; se for triangular, obviamente aparecerão apenas três vértices. Aliás, outra coisa que você deve ter percebido é que, ao selecionar mais de uma face do cilindro, aparentemente nada muda na janela UV/Image Editor. Afinal, se estamos selecionando mais de uma face, não deveriam aparecer as coordenadas de todas as faces?

Na verdade, todas as faces estão aparecendo na UV/Image Editor, só que estão sobrepostas. As faces, de início, recebem coordenadas padrão de mapeamento, de forma a usarem toda a área da imagem a ser mapeada, no caso de faces com quatro vértices, ou metade dela, no caso de faces com três vértices. Nós queremos mudar isso: cada face lateral do cilindro deverá receber um pedaço da imagem que será usada como fundo.

Uma das formas de se fazer isso é selecionando face por face e ajustando os pontos na raça – alguém se habilita?

Claro que existe um jeito melhor de se fazer isso; o Blender possui algumas ferramentas que facilitam a vida nesse momento. Essas ferramentas são chamadas de *UV Unwrapping* – algo como Desdobramento UV. A idéia é pegar um objeto 3D e desmontá-lo até se tornar um objeto 2D. O melhor exemplo desse tipo de técnica é aquela cruz que todo mundo já deve ter recortado de alguma revista infantil. Era só dobrar e colar onde mandavam e pronto – você tinha um cubo formado!

(Só que, no nosso caso, nós partimos do cubo pronto e queremos descobrir um jeito de transformá-lo na cruz.)

Dos mapeamentos UV que o Blender oferece, o mais interessante é o LSCM ou *Least Square Conformal Map* (difícil traduzir isso; é algo como “mapeamento ajustado ao menor quadrado possível”). Para o LSCM

funcionar direitinho, você precisa dizer ao Blender quais são as arestas das faces selecionadas que serão abertas no momento de desdobrar o objeto – tomemos o nosso cilindro como exemplo.

Imagine um cilindro feito com uma folha de sulfite, aberto em cima e embaixo. Se você tentar transformá-lo num objeto 2D do jeito em que ele se encontra, o máximo que você conseguirá é uma versão “amassada” dele...

Qual é a solução? Para criar o cilindro, você não teve que colar as bordas da folha? Pois bem, para abri-la, não seria mais lógico cortá-la no ponto onde as bordas foram unidas? Esse é o princípio de criação de fendas: o LSCM usará essas fendas como dicas de onde deverá abrir as faces selecionadas.

Perceba que eu disse “faces selecionadas”, e não “objeto”. O LSCM calcula o desdobramento das faces selecionadas e não do objeto como um todo. Portanto, para conseguirmos abrir as faces laterais do cilindro de forma coerente, basta marcar uma de suas arestas verticais como fenda ou *seam*, em inglês.

Para criar uma fenda, pressione **TAB** – o cilindro entrará no Edit Mode. Clique na ferramenta de seleção de arestas no cabeçalho da 3DView (**Figura 5-2**; se você não estiver vendo-a, clique com **BMM** no cabeçalho e arraste-o para a esquerda até a ferramenta aparecer). Selecione qualquer uma das arestas laterais com **BDM** (você vai ter que rotacionar um pouco a câmera com **BMM**) e pressione **CTRL+E**. Surgirá uma caixa de diálogo – selecione a primeira opção, Mark Seam.



Figura 5-2. A ferramenta de seleção de arestas está entre os pontinhos (seleção de vértices) e o triângulo (seleção de faces), à direita.

A aresta ficará laranja (**Figura 5-3**), indicando que, a partir de agora, ela funcionará como uma dica de corte para o LSCM. Pressione **TAB** para retornar ao UV/Face Selection Mode.

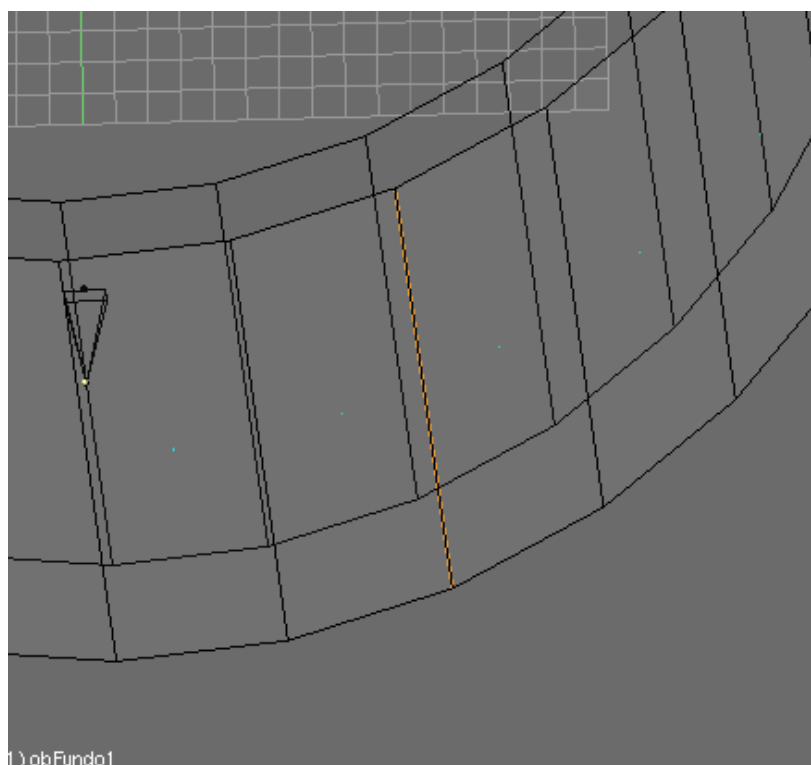


Figura 5-3. As arestas marcadas como seams aparecem em laranja na 3DView.

Agora sim, selecione todas as faces laterais (na 3DView) pressionando **AKEY** –se alguma face já estiver selecionada, **AKEY** irá desselecioná-la; portanto você terá que pressionar **AKEY** duas vezes para que as faces sejam selecionadas. Repare que, ao selecionar uma face na 3DView, a UV/Image Editor mostra as coordenadas de mapeamento de textura daquela face: a linha vermelha representa o eixo U, enquanto a linha verde representa o eixo V.

Após ter selecionado todas as faces, pressione **UKEY** com o ponteiro do mouse sobre a 3DView – surgirá a caixa de diálogo UV Calculation. Selecione a quarta opção, LSCM. Repare no que aconteceu na janela UV/Image Editor: o mapeamento de todas as faces surgiu lado a lado. Agora, abra o menu Image >>

Open... da janela UV/Image Editor e selecione a imagem `ceu1.jpg` que deve estar na pasta `Cap05-JogoNave\Texturas`. Clique em `Open Image` para usá-la como textura das faces selecionadas.

Acabou? Que nada! Estamos apenas começando... Existem alguns detalhes que requerem a nossa atenção:

1. O mapeamento das faces não abrangeu toda a extensão da figura. Como resultado, as bordas esquerda e direita da imagem, quando grudadas uma na outra, aparecerão truncadas, ou seja, se você prestar atenção, perceberá que a imagem quebra naquele ponto, destruindo a ilusão de um cenário de fundo contínuo e infinito;
2. Não sei se você percebeu, mas a textura foi aplicada do lado de fora das faces (**Figura 5-4**; pressione **FKEY** para sair do modo UV Face Select e pressione **ALT-Z** para enxergar a textura; retorne pressionando **FKEY** novamente). Claro, isso é o que normalmente se faz quando se mapeia um modelo. Mas lembre-se: não estamos fazendo algo normal, estamos fazendo um jogo! Reparou que a câmera está posicionada dentro do cilindro? Isso porque o cilindro representa o *cenário externo*, devendo *englobar* os demais elementos do jogo. Se você olhar o cilindro do ponto de vista da câmera (selecione a câmera e pressione **CTRL+0**), perceberá que, *do ponto de vista da câmera*, as texturas foram mapeadas na direção errada.

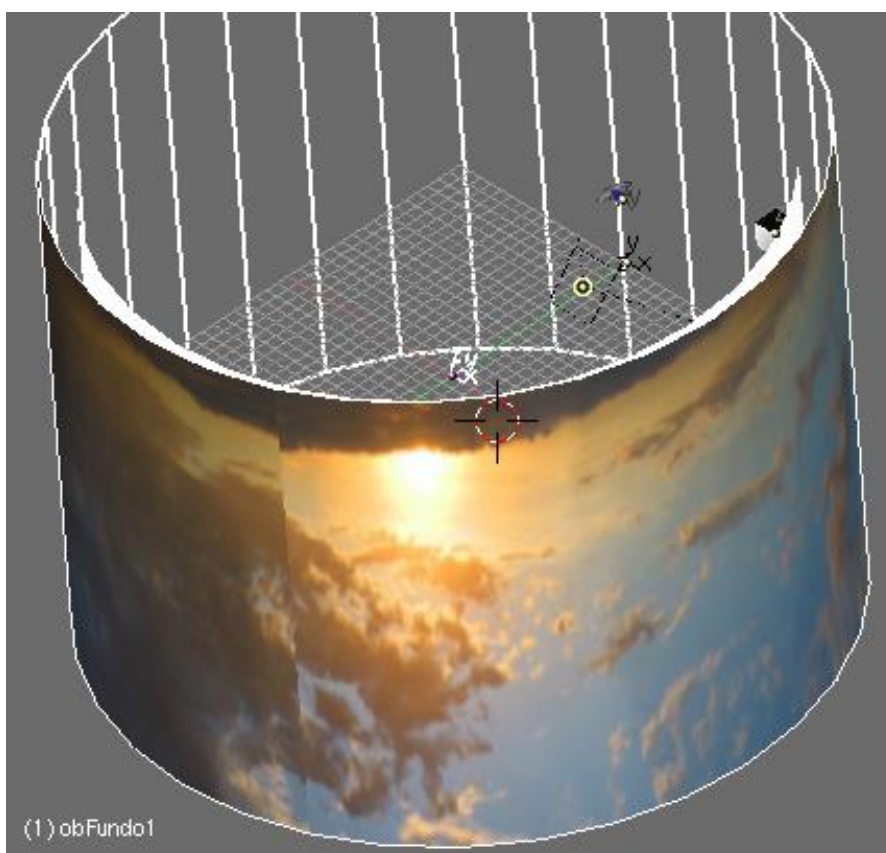


Figura 5-4. Tem alguma coisa errada aqui...

Como diria Jack, o Estripador, vamos por partes.

2.2. Acertando a extensão do mapeamento

A manipulação de vértices na UV/Image Editor segue o mesmo padrão dos vários modos da 3DView: **GKEY** movimenta os vértices, **SKEY** redimensiona um grupo de vértices e **RKEY** os rotaciona. Mas, além desses comandos nós vamos usar também a caixa de diálogo Transform Properties (acessível pressionando-se **NKEY**) para inserir valores precisos pelo teclado.

Muito bem. Posicione o mouse sobre a UV/Image Editor e pressione **AKEY** para selecionar todos os vértices – vértices selecionados são mostrados em amarelo, enquanto vértices desselecionados aparecem em rosa. Se, ao pressionar **AKEY**, todos os vértices aparecerem desselecionados (ou seja, rosa), significa que já existia um ou mais vértices previamente selecionados (por padrão, **AKEY** primeiramente desseleciona todos os vértices selecionados). Se isso acontecer, basta pressionar **AKEY** novamente que todos os vértices serão selecionados.

Agora, pressione **NKEY** para mostrar a caixa de diálogo. Quando você tem apenas um vértice selecionado, surgirão dois campos: **Vertex X** e **Vertex Y** (**Figura 5-5**). Através deles você pode posicionar de forma cirúrgica um vértice dentro de uma textura.

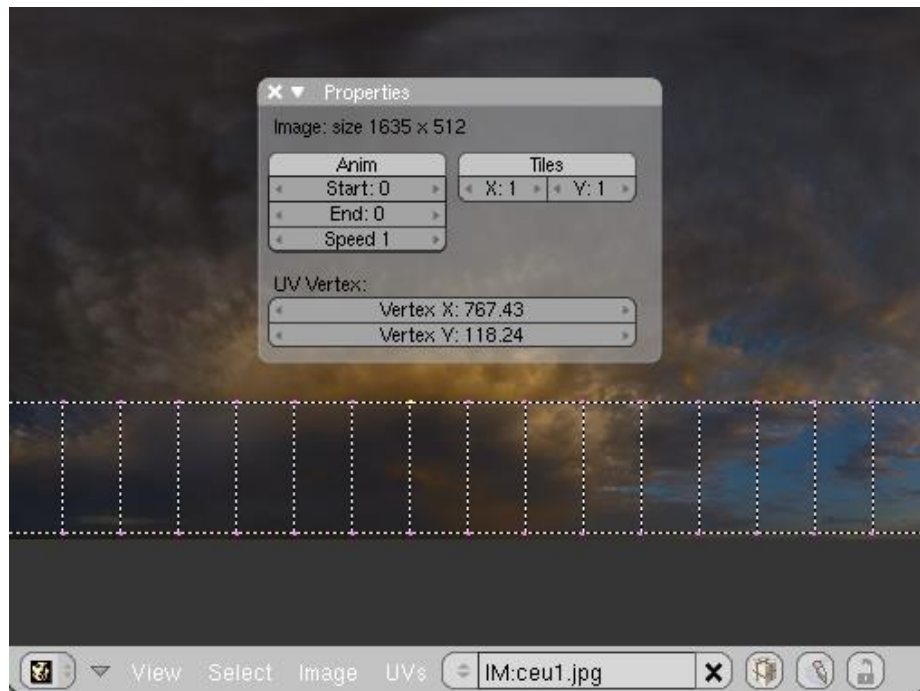


Figura 5-5. Transform Properties com um vértice selecionado.

Mas, como estamos com vários vértices selecionados, os campos foram renomeados para **Median X** e **Median Y** (**Figura 5-6**). Agora, todos os valores que forem inseridos nesses campos dirão respeito ao ponto médio dos vértices – se, por exemplo, você tem dois vértices selecionados e eles possuem coordenadas 0 e 10 no eixo X, respectivamente, então o Median X dessa seleção será 5. Se outros vértices forem adicionados à seleção, os pontos médios (X e Y) serão automaticamente recalculados.



Figura 5-6. Transform Properties com vários vértices selecionados.

Prosseguindo, aqui está o nosso plano de ação:

1. Posicionar os pontos médios da nossa seleção exatamente no centro da figura.

Para tanto, basta pegar as dimensões da imagem (mostradas no alto da caixa de diálogo, no formato *largura X altura*), dividi-las por 2 e inseri-las nos campos **Median X** e **Median Y**, lembrando que no primeiro você deverá colocar a metade da largura (resultando em 512), enquanto no segundo será colocada a metade da altura (256, no caso). O grupo de vértices agora deve estar centralizado com a imagem (**Figura 5-7**).

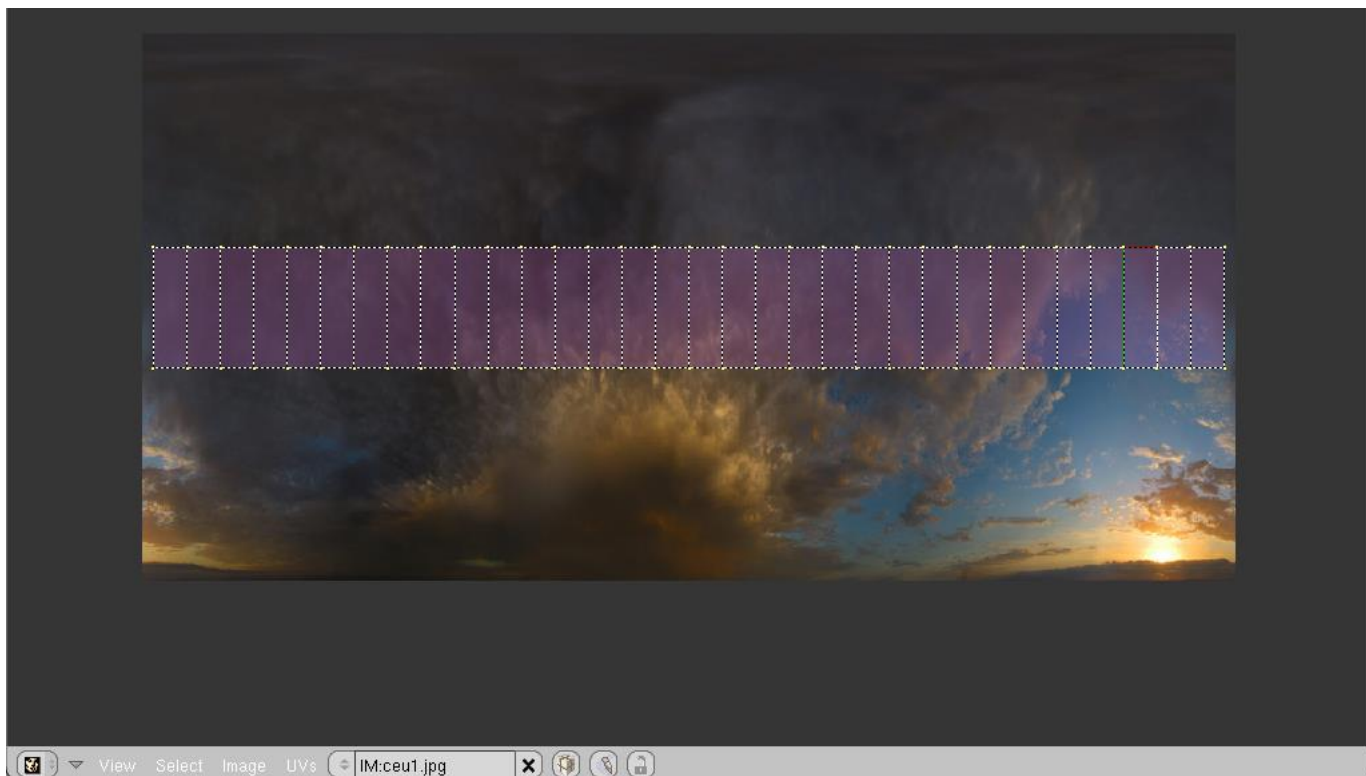


Figura 5-7. Os vértices centralizados na imagem.

2. Acertar a coordenada Y dos vértices.

Nós queremos que as faces selecionadas ocupem toda a extensão da textura; para conseguir isso, nós vamos acertar uma coordenada por vez, começando por Y. Pressione **AKEY** para desselecionar todos os vértices; agora, pressione **BKEY** e arraste uma caixa de seleção em torno dos *vértices superiores apenas* (e não se esqueça que tudo que está sendo falado aqui deve ser feito na UV/Image Editor, e não na 3DView!) – isso deverá selecionar os vértices envolvidos pela caixa de seleção. Esse comando é aditivo, ou seja, se você criar uma nova caixa de seleção envolvendo um outro grupo de vértices, os novos vértices serão adicionados à seleção anterior.

Pois bem, com os vértices superiores selecionados, digite o valor da altura no campo Median Y – 0, no nosso caso. Os vértices superiores devem ter se deslocado para o extremo inferior da textura. O porquê de ter deslocado os vértices superiores para baixo? É que, se você reparar na 3DView, *a imagem está invertida*; devemos, portanto, inverter a coordenada V da texturização.

Repita todo o processo descrito nesse passo, mas, desta vez, com os vértices inferiores – e lembrando de inserir, dessa vez, o valor 512 no campo Median Y (**Figura 5-8**).

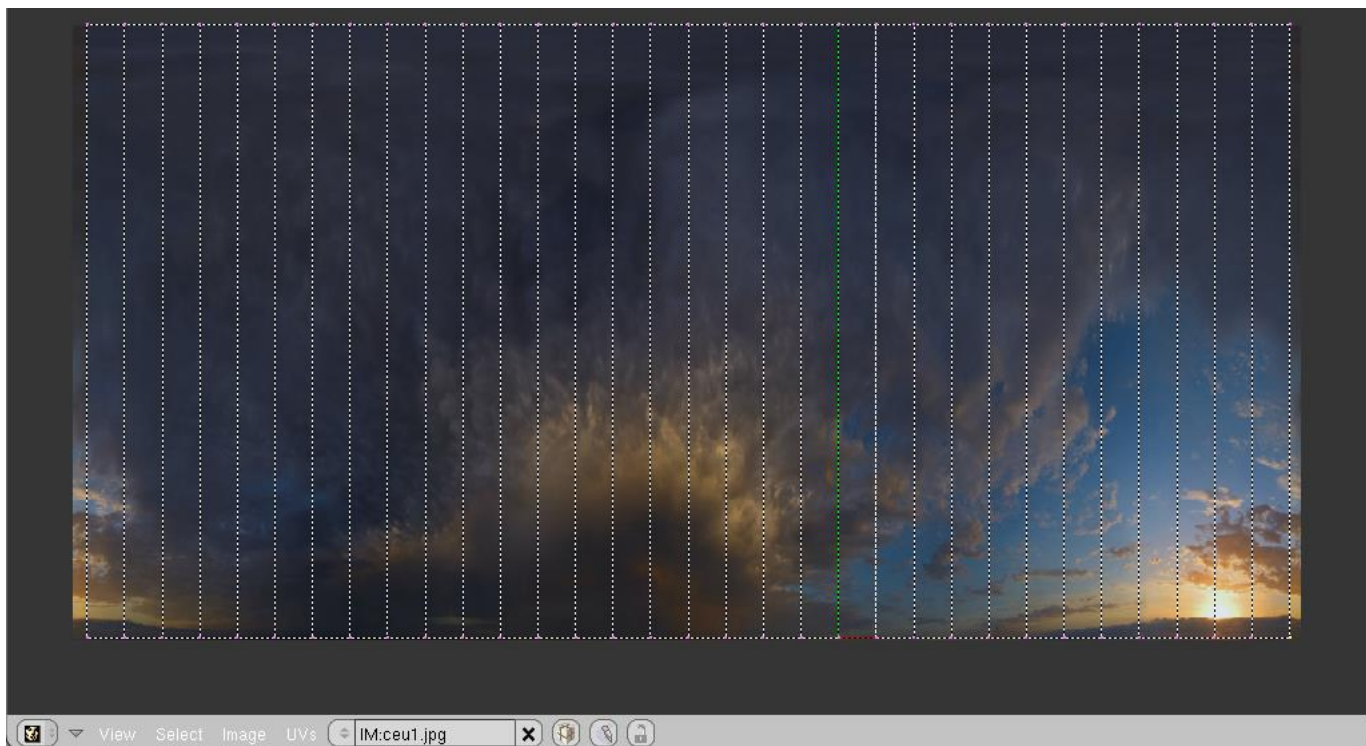


Figura 5-8. Os vértices corretamente posicionados no eixo Y.

3. Redimensionar os vértices no eixo X.

Agora que os vértices estão corretamente posicionados no eixo Y, vamos tratar de acertá-los no eixo X. No entanto, agora temos um problema a mais: não dá pra corrigir a coordenada X da mesma forma que corrigimos a coordenada Y, por um simples motivo: agora, não se trata apenas de deslocar os vértices, mas também de aumentar a distância entre eles proporcionalmente. Esse é um tipo de ação que não dá pra fazer numericamente, então teremos que apelar para o nosso sentido aranha nesse momento...

Pressione **AKEY** duas vezes – uma para desselecionar os vértices selecionados e outra para selecionar todos os vértices. Pressione **SKEY** para iniciar o redimensionamento e, logo em seguida, pressione **XKEY** para restringi-lo ao eixo X – repare que os vértices vão se distanciando proporcionalmente uns dos outros; dessa forma, evitamos que ocorram distorções na texturização. Vá esticando a seleção de vértices até que os vértices mais extremos estejam mais ou menos posicionados nos extremos da textura. Clique com **BEM** para confirmar a nova configuração do grupo (**Figura 5-9**).

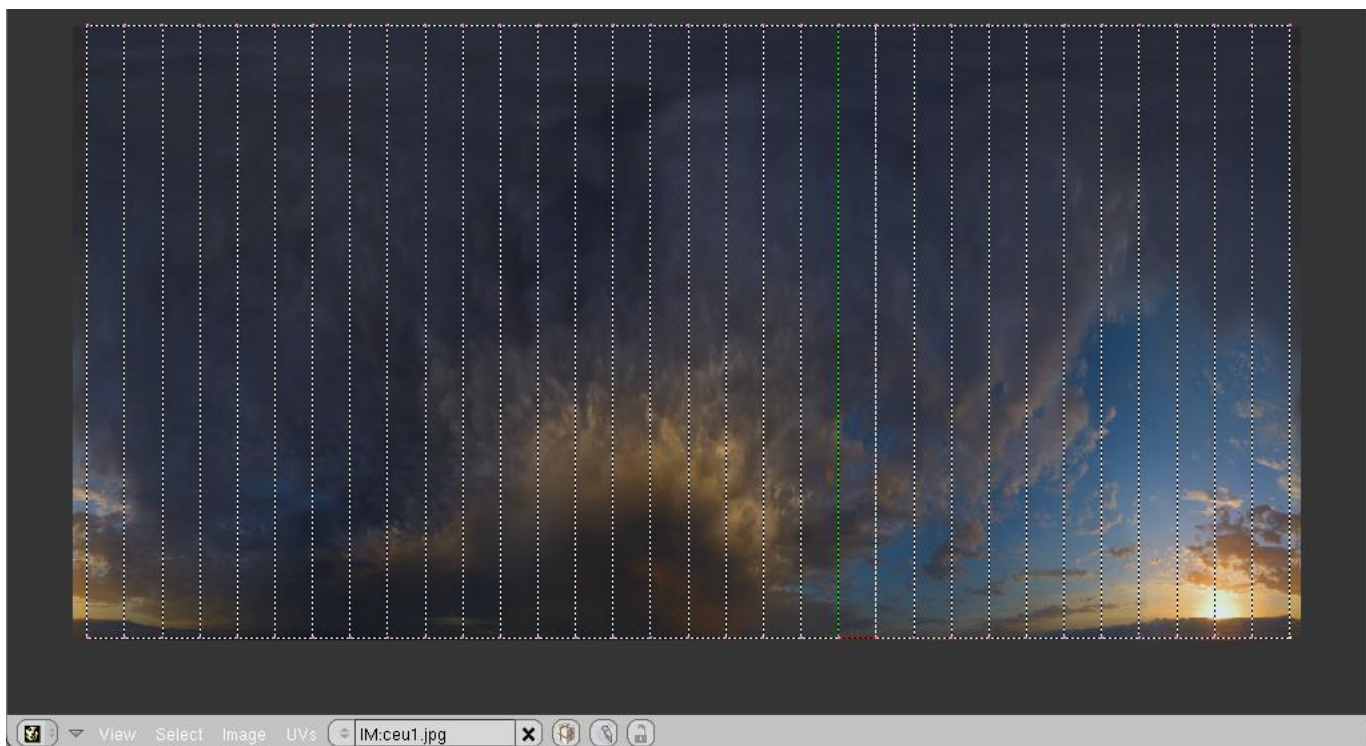


Figura 5-9. Os vértices mais ou menos posicionados no eixo X.

4. Acertar o posicionamento dos vértices extremos.

Acabou? Nananina! Tem mais um detalhe a ser acertado antes de batermos o martelo: como o redimensionamento no eixo X foi feito no olhômetro, a textura ainda continua truncada! Então agora chegou o momento de acertar numericamente os vértices extremos.

E agora eu peço muita atenção para um detalhe: nós vamos mexer nos vértices extremos, independentemente dos vértices intermediários. Isso significa que iremos, de uma certa forma, inserir uma distorção no mapeamento, mas, como tomamos o cuidado de posicionar o melhor possível os vértices antes de cometer essa heresia, a distorção será imperceptível. Afinal, pequenas distorções no mapeamento interno são aceitáveis, mas nos extremos são imperdoáveis!

Muito bem. O procedimento que eu vou mostrar aqui deverá ser feito uma vez para cada um dos quatro vértices extremos. Em primeiro lugar, selecione um por vez com **BDM**; em seguida, na caixa de diálogo Transform Properties, digite os valores relacionado a ele:

- Para o vértice inferior esquerdo – Vertex X = 0; Vertex Y = 0;
- Para o vértice superior esquerdo – Vertex X = 0; Vertex Y = 512;
- Para o vértice inferior direito – Vertex X = 1024; Vertex Y = 0;
- Para o vértice superior direito – Vertex X = 1024; Vertex Y = 512;

Agora que o mapeamento está concluído, posicione o ponteiro do mouse sobre a 3DView e pressione **FKEY** para sair do modo UV/Face Select e retornar ao modo Object. Pressione **ALT+Z** (se você não o pressionou anteriormente) e voilá: eis que surge a textura aplicada no cilindro!

Muito bem, resolvemos o primeiro problema – o de criar uma textura infinita. Passemos, então, à segunda questão.

2.3. *Ei, não aponta esse dedo pra mim!*

Com o cilindro devidamente texturizado, precisamos inverter a direção do mapeamento para que a textura aponte para dentro. Para tanto, faça o seguinte:

- Selecione o cilindro com **BDM**;
- Pressione **TAB** para entrar no modo Edit;
- Pressione **F9** para mudar os botões para o contexto Editing;
- Lá na direita da ButtonsWindow, existe uma guia chamada Mesh Tools 1. No centro dessa guia existe um botão chamado Draw Normals (Desenhar Normais); clique nele se ele não estiver selecionado. Deverão surgir as retas normais das faces num tom verde-água. Se você não estiver enxergando direito as normais por estarem muito pequenas, basta ajustar o tamanho delas no botão NSize, localizado logo acima de Draw Normals;
- Percebo que as retas normais estão apontando para fora. Essas retas servem principalmente para se fazer dois tipos de cálculos: de iluminação e de texturização. No caso da texturização, ela será mostrada apenas na direção em que a reta normal do polígono aponta. Para inverter a direção das normais, pressione **AKEY** para selecionar todos os vértices (ou arestas, ou faces, dependendo do modo de seleção que estiver ativo) do objeto; em seguida, pressione o botão Flip Normals, localizado na guia Mesh Tools (esse botão pode ser complicadinho de encontrar numa primeira vez; no topo da guia Mesh Tools existem 12 botões dispostos em quatro fileiras e três colunas – o botão Flip Normals é o quarto botão da primeira coluna, de cima para baixo).

Pronto! Agora, se você ativar a visão da câmera, você conseguirá enxergar a textura mapeada no cilindro (**Figura 5-10**). Até que não foi tão difícil assim, certo?

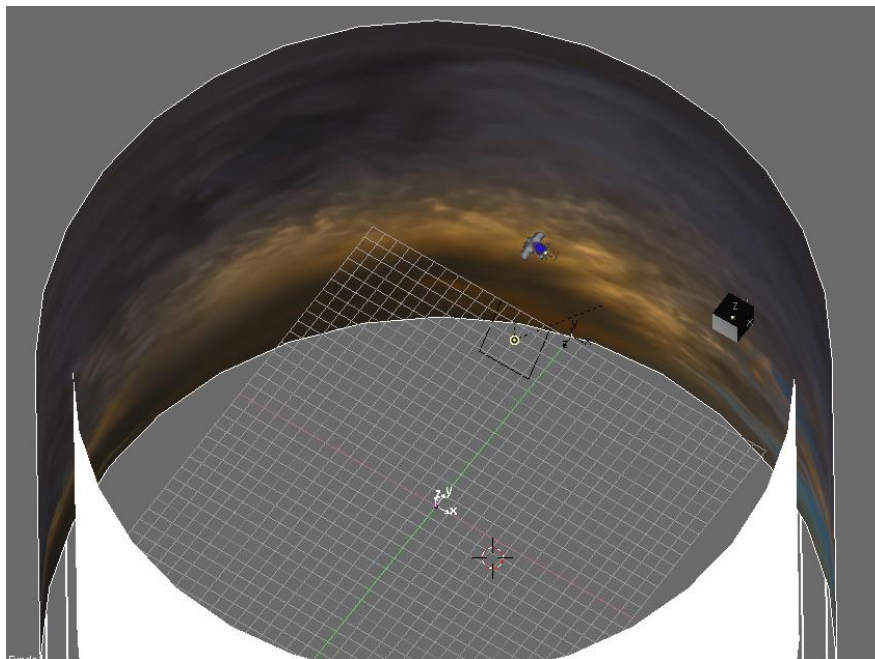


Figura 5-10. A textura do cilindro agora está para dentro.

(Já vou avisando que não tenho plano de saúde e que não adianta me mandar conta de cardiologista, psicólogo, terapeuta alternativo, ou qualquer outro lugar que você venha a frequentar depois desse capítulo...)

2.4. Mas... e o outro cilindro?

Então, eu tenho uma má e uma boa notícia pra você. A má notícia é que você vai ter que repetir todo o procedimento de mapeamento com esse cilindro, só que agora você vai usar o arquivo `ceu2.png` como textura. A boa notícia é que fazer pela segunda vez é sempre mais fácil... :-)

Tá bom, não é uma notícia tão boa assim...

Você deve estar se perguntando por que diabos eu coloquei dois cilindros como cenário de fundo, certo? É que mais pra frente nós vamos criar um efeito interessante com eles...

Voltando. Uma coisa a mais que você terá que fazer no segundo cilindro é ativar o uso do canal alfa nas suas faces. Para isso, entre no modo UV/Face Select (**FKEY**; se você estiver no modo Edit, então pressione **TAB** para retornar ao modo Object e só então pressione **FKEY**) e selecione tudo (**AKEY**, uma ou duas vezes, dependendo de já existirem elementos selecionados ou não). Com a ButtonsWindows no contexto Editing (**F9**), localize a guia Texture Face, à direita da janela, e clique no botão Alpha. Com isso, todas as faces selecionadas passam a reconhecer o canal alfa da textura, certo? Errado! Ao clicar no botão Alpha, você mudou apenas a configuração da face ativa (aquela que mostra os eixos U e V à sua volta na 3DView); para mudar as demais, clique no botão Copy DrawMode, localizado logo abaixo do botão Alpha. Com isso, todas as características de desenho da face ativa serão copiadas para as demais faces selecionadas (**Figura 5-11**).

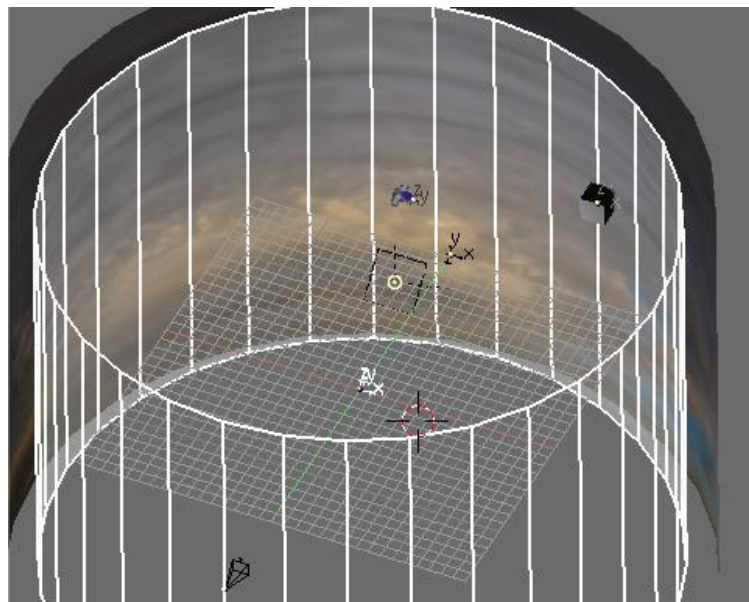


Figura 5-11. O segundo cilindro está com o canal alpha ativado.

E é isso aí! Agora podemos prosseguir para a descrição dos outros elementos do jogo.

3. Naves, Emissor de Naves e Tiros

O jogo, nessa versão simplificada, irá possuir apenas um tipo de nave para o jogador usar, um tipo de tiro e um tipo de nave inimiga. A ideia aqui é mostrar o conceito básico por trás de cada elemento – dominando o conceito básico, fica fácil criar variações a partir dele.

3.1. As naves e os tiros

Como está completamente fora do escopo desse capítulo ensinar a modelar uma nave, elas já foram inseridas na cena. Aliás, se você prestar atenção, irá perceber que as naves, tanto do jogador quanto inimigas, possuem a mesma forma – é que eu usei o mesmo modelo para ambos, variando apenas a pintura delas. A nave do jogador está pintada de azul, enquanto a nave inimiga é predominantemente laranja.

Já o tiro foi bem mais fácil de modelar: é um paralelepípedo criado no próprio Blender e pintado de branco. Ao dar um zoom no tiro (ele está bem à direita do “G” do Game Over), você irá perceber que ele possui um eixo de coordenadas local; esse eixo será de extrema importância mais pra frente, quando formos definir as ações de criação de instâncias do tiro e da nave inimiga.

Outra coisa que você deve reparar é que os modelos da nave inimiga e do tiro estão na layer 11, e não na layer 1 – para comprovar isso, pressione alternadamente **1KEY** e **ALT-1** (**Figura 5-12**); assim, o conteúdo das duas camadas é mostrado alternadamente. Para a visualização de ambas, segure **SHIFT** enquanto pressiona **1KEY** e **ALT-1**, um de cada vez. Mais pra frente você vai entender o porquê disso.



Figura 5-12. O conteúdo da camada escondida.

3.2. O emissor de naves

Reparou na existência de um cubinho no lado direito da 3DView (**Figura 5-13**)? Ele será o disparador das naves que forem criadas – por isso eu o chamei de emissor de naves. Conceitualmente falando, você pode pensar nele como se fosse a nave-mãe, de onde decolam as naves; tecnicamente falando, o cubo tem a mesma função de uma fonte emissora de partículas, mas, ao invés de emitir faíscas, fagulhas, etc., ele emite naves.

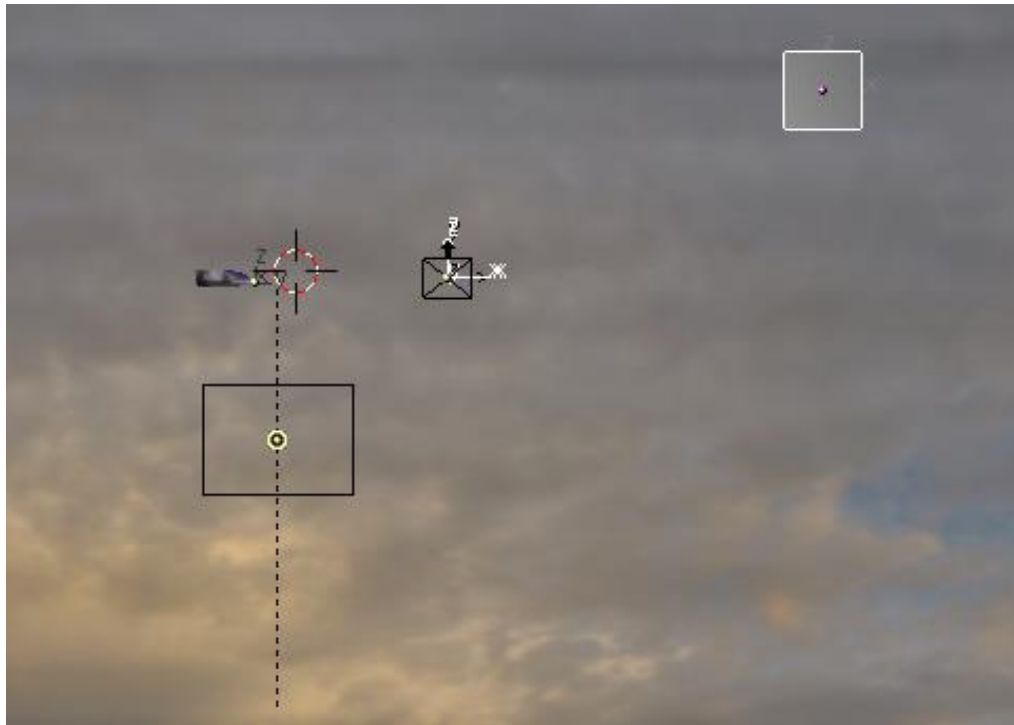


Figura 5-13. Olá, cubinho...

Esse emissor ficará fora da tela, movendo-se para cima e para baixo. Assim, as naves sempre serão disparadas a partir de pontos diferentes do espaço. Poderíamos ter usado outro tipo de estratégia – por exemplo, colocar dois ou três emissores fixos em pontos distintos – mas a intenção é tentar manter as coisas o mais simples possível.

4. Posicionando os elementos de overlay do jogo

Calma que eu explico: *overlay* é o nome que se dá à camada de interface do jogo. No nosso caso, a interface é composta pelo placar, pela quantidade de cidades restantes e pelo número de vidas do jogador.

Os elementos do overlay podem ser compostos por dois itens: um ícone, representando a sua função, e uma área numérica, que varia conforme o desenrolar do jogo. Vamos, portanto, analisá-las separadamente.

4.1. Ícones

Há dois ícones no nosso overlay, um relativo à cidade e o outro relativo às vidas do jogador (**Figura 5-14**):

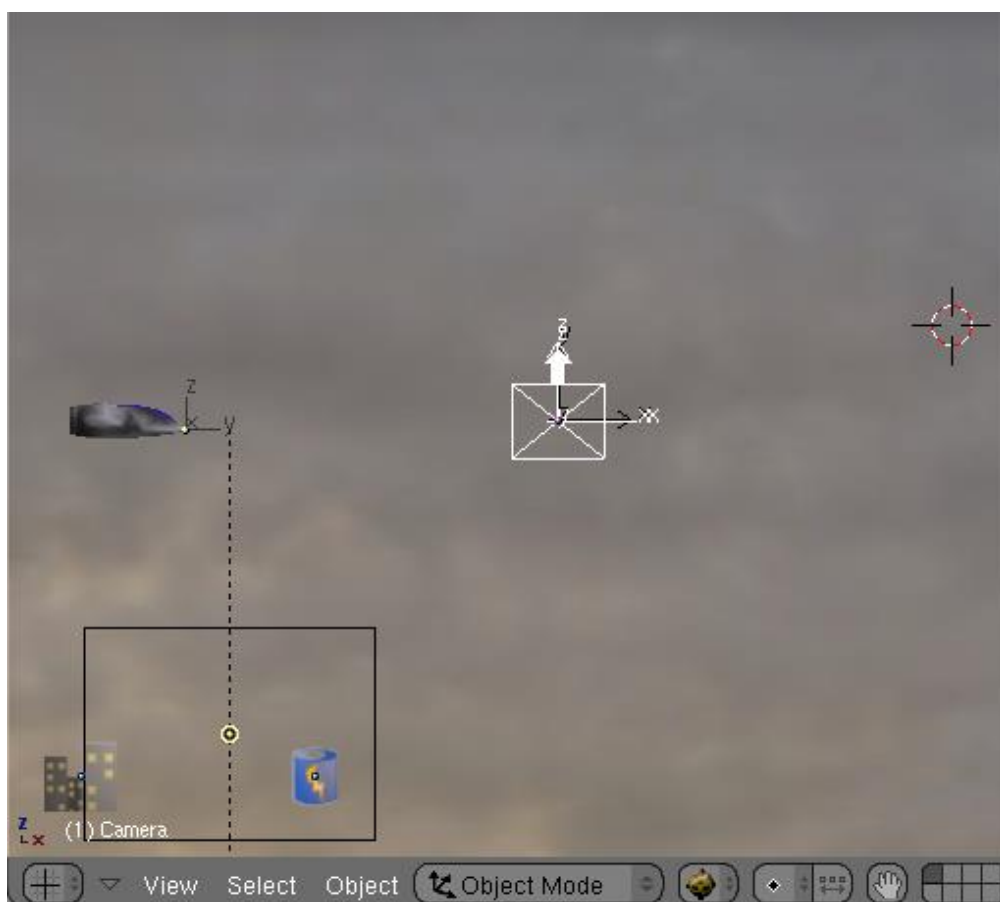


Figura 5-14. Os ícones da cidade e da energia.

Para criá-los, basta mapeá-los em um plano e posicioná-los corretamente (coloque-os no canto inferior direito da tela, como mostrado na **Figura 5-14**; para posicionar melhor, selecione a câmera e pressione **0NUM** para entrar na visão dela. O tracejado externo é o limite da visão). Na pasta Cap05-JogoNave\Texturas você encontrará as imagens cidade.png e bateria.png, que devem ser usadas, respectivamente, como texturas dos ícones da cidade e da energia.

4.2. Marcadores de vida, de cidades e de pontuação

A parte dinâmica do overlay, relacionada à contagem de pontuação, de vidas, etc., tem a ver com o que você aprendeu no capítulo anterior. Primeiro, você deve ter aquela imagem contendo todos os caracteres alfanuméricos, que serão usados na construção dos placares. Para sua comodidade, o arquivo JogoNave4-2.blend já vem com os marcadores posicionados.

Por fim, para fazer os placares funcionarem, teremos que adicionar lógica a eles, o que será mostrado mais pro final do capítulo.

5. Senhoras e senhores, dêem partida nos motores!

E agora começa a diversão! Até então, tudo o que fizemos foi preparar o terreno para podermos alcançar o verdadeiro objetivo desse capítulo: fazer o jogo funcionar. Pois bem: senhoras e senhores, é bom apertarem os cintos, porque estamos prestes a decolar!

(Ok, ok, foi meio clichê; tudo bem que é um jogo de nave e que o jogo não é a grande sensação do verão, mas eu tinha que valorizar o produto, né? :-)

5.1. Gira gira gira... Rotacionando os cilindros

Um minuto da sua atenção, por favor

A partir deste ponto serão mostradas algumas dezenas de Sensors, Controllers e Actuators. No entanto, ao invés de mostrar tudo de um objeto e fazer malabarismos absurdos para explicar como cada coisa encaixa com as outras que nem foram apresentadas, eu decidi tomar um outro caminho: conforme formos entrando em determinados assuntos dentro da lógica do jogo, irei apresentar todos os LogicBricks relacionados ao assunto – com isso, acabarei indo e voltando várias vezes nos mesmos objetos. A única coisa que você deve fazer, ao se deparar com um conjunto novo de LogicBricks, é reparar a qual objeto o conjunto pertence e *adicioná-lo* a este objeto – ou seja, nada de apagar o que já foi inserido! O processo é aditivo, não destrutivo.

Para saber a qual objeto a lógica apresentada nas figuras pertence, basta reparar que, na parte superior dos LogicBricks, aparece o nome do objeto que os contém.

Outro detalhe: mais pro final do tópico aparecerão figuras com vários LogicBricks minimizados. Isso significa que você já inseriu estes LogicBricks em algum momento e que eles não precisam ser reinseridos.

A primeira parte que iremos botar pra funcionar é o movimento do plano de fundo. Lembra-se dos dois cilindros que texturizamos agora há pouco? Pois bem, iremos fazê-los rotacionarem, criando a sensação de que a nave está voando, quando na verdade ela está quietinha no seu canto!

E qual a razão de termos colocado dois cilindros de cenário de fundo. É que iremos criar um efeito chamado *paralaxe*.

A paralaxe diz respeito à sensação de profundidade que temos quando estamos nos movimentando e observamos os objetos à nossa volta: objetos mais próximos parecem se movimentar mais rápido, enquanto que objetos mais distantes nos dão a impressão de estarem se mexendo mais devagar. Pois bem, o que iremos fazer para simular a paralaxe é determinar velocidades diferentes para os cilindros: o cilindro mais ao fundo irá se movimentar mais devagar do que o cilindro da frente.

Além de melhorar a sensação de profundidade, o efeito de paralaxe também serve para disfarçar a repetição contínua do fundo – isso acontece porque o cruzamento dos dois planos sempre gera imagens diferentes das anteriores.

Muito papo, pouca ação. É melhor começar a trabalhar antes que você durma em cima da apostila! :-)

Selecione o cilindro externo com BDM, abra o contexto `Logics (F4)`; crie um Sensor, um Controller e um Actuator, como mostrado na **Figura 5-15**, e conecte-os. Não modifique os LogicBricks – as opções padrão são exatamente o que precisamos.



Figura 5-15. Os LogicBricks do cilindro externo.

No Actuator Motion, localize a fileira denominada **dRot**. Clique com **SHIFT+BEM** no terceiro campo numérico da fileira – ele abrirá para edição via teclado. Digite -0.04 e pressione **ENTER** – mas atenção! A tecla **NUMPONTO** (o ponto do teclado numérico) não funciona dentro desses campos numéricos – o porquê disso? Boa pergunta... O jeito é usar o ponto do teclado normal – aquele que fica duas teclas à direita de **MKEY**. Ah, e outra coisa – não adianta colocar vírgula como separador decimal, porque não vai funcionar.

Com esse valor, o cilindro irá rotacionar 0,1 unidades de medida no eixo Z, no sentido negativo. Se você pressionar **PKEY** sobre a 3DView para testar o jogo, verá que o cilindro gira de uma maneira tal que dá a impressão da nave estar voando nos céus do planeta, quando está acontecendo exatamente o contrário...

Selecione o cilindro interno e faça o mesmo procedimento anterior com ele. A única diferença será na velocidade: como esse cilindro está mais próximo, iremos colocar um valor de rotação de -0.10 no terceiro campo numérico da fileira **dRot**.

O que é dLoc e dRot?

dLoc e **dRot** são abreviações de *deltaLocation* e *deltaRotation*. O delta, na física, diz respeito à variação de algum atributo de um objeto que está sendo estudado, podendo ser a variação de velocidade, espaço, tempo, etc. No caso do Blender, **dLoc** ou *deltaLocation* define o quanto um objeto irá se deslocar num determinado eixo a cada quadro de animação do jogo; já **dRot** ou *deltaRotation* define o quanto um objeto irá rotacionar em um determinado eixo (medido em graus) a cada quadro de animação do jogo.

E o que define os campos numéricos do Motion Actuator?

Já mencionei que os três campos de cada fileira do Motion Actuator dizem respeito, respectivamente, aos eixos X, Y e Z?

Se você testar o jogo agora (**PKEY** na 3DView) o cilindro interno estará girando mais rápido que o externo, dando a sensação de paralaxe que buscávamos criar.

Vamos, então, ao próximo passo: movimentar a nave na tela.

5.2. Movimentando a nave

A movimentação da nave deve levar dois fatores em conta: a captura do pressionamento das setas direcionais e a limitação do movimento dela (afinal, não queremos que ela saia da tela – quem mais iria aceitar uma missão kamikaze caso ela vá embora?).

A captura das setas direcionais é a parte trivial do problema. Basta adicionar um Keyboard Sensor para cada uma das setas (o processo foi descrito no capítulo 3), ligar cada uma delas a um Controller distinto e conectar cada um dos Controllers a um Motion Actuator (também distinto), como é mostrado na figura abaixo:

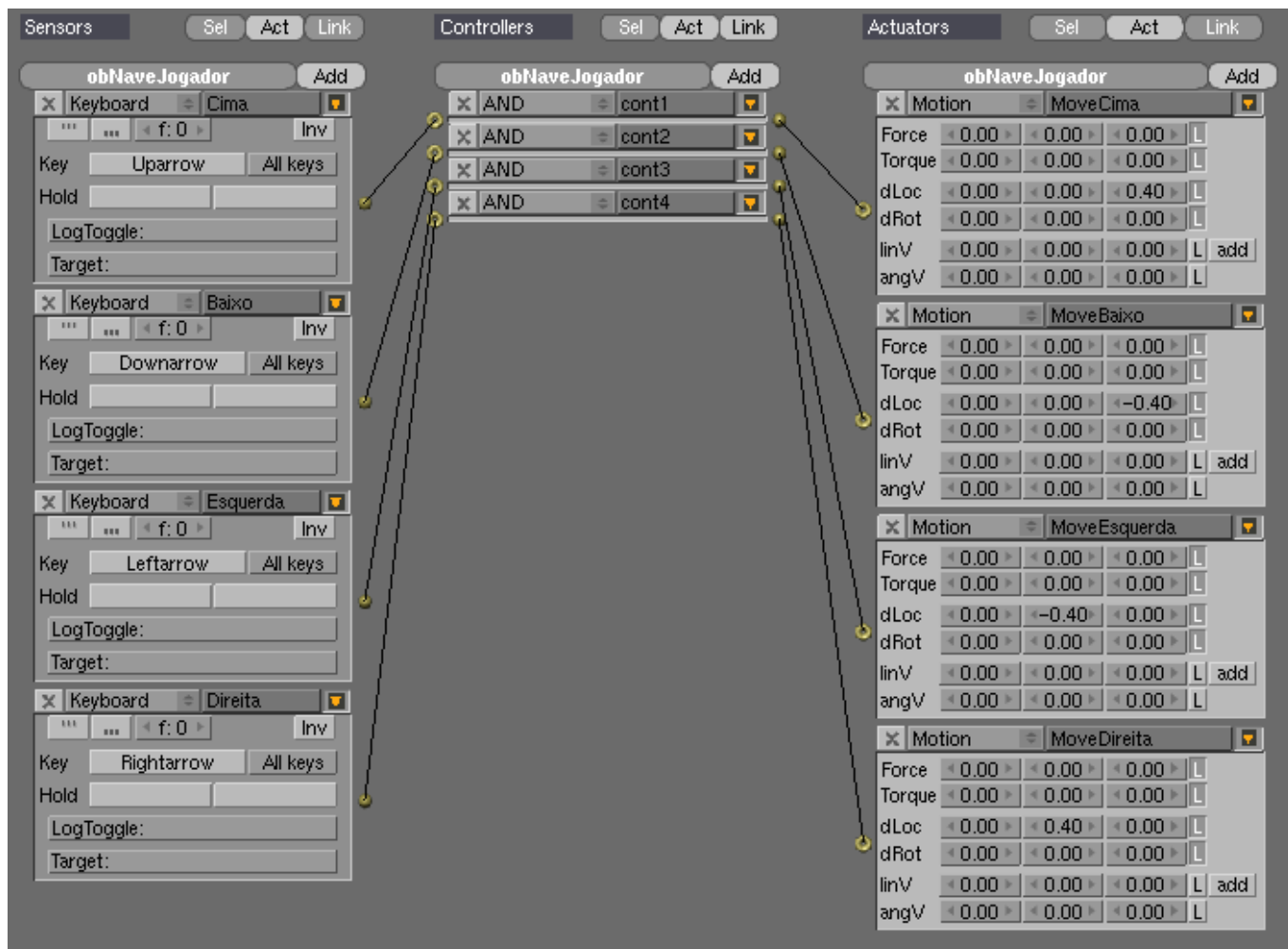


Figura 5-16. Os LogicBricks de movimentação da nave.

Renomeie os Sensors e os Actuators da maneira que aparecem na figura.

Os valores a serem colocados em cada um dos Actuators são os seguintes:

- Para o movimento para cima, coloque o valor 0.4 no campo Z da fileira dLoc;
- Para o movimento para baixo, coloque o valor -0.4 no campo Z da fileira dLoc;
- Para o movimento para a esquerda, coloque o valor -0.4 no campo Y da fileira dLoc;
- Para o movimento para a direita, coloque o valor 0.4 no campo Y da fileira dLoc.

Repare que o botãozinho verde à direita da fileira dLoc (com um “L” escrito) está pressionado, o que significa que o movimento da nave será feito relativo às suas coordenadas locais, e não às coordenadas globais da cena.

Essa foi a parte fácil; agora, como vamos definir os limites de movimentação da nave, de forma a mantê-la dentro da tela?

O jeito mais fácil é colocar objetos em volta da cena que servirão apenas de *detectores de colisão*: assim que a nave colidir com um desses objetos, ela deverá parar o movimento que estava fazendo naquela direção.

Para evitar que esses objetos apareçam na tela, iremos aplicar uma textura simples com 100% de transparência em todos os pixels; assim conseguiremos esconder o objeto como um todo.

Selecione a visão lateral (**NUM1**), pressione **BARRAESPAÇO** selecione o menu Add>>Mesh>>Cube. Um cubo irá aparecer no centro do 3DCursor. Movimente-o nas três visões ortográficas com **GKEY** até que ele esteja exatamente à esquerda da nave (relativo à vista da câmera, **Figura 5.17**).



Figura 5-17. O cubo à esquerda da nave será esticado para criar o limite esquerdo.

Selecione o cubo com **BDM** e pressione **NKEY** na 3DView para abrir a janela Transform Properties (propriedades de transformação). No campo OB, renomeie o objeto para paredeEsquerda; no campo SizeY, digite o valor 10 – isso fará com que o cubo estique 10 vezes o seu tamanho no eixo Y. No campo LocX (posicionamento no eixo X), coloque o valor -8. Além desses campos, os demais devem possuir valores semelhantes aos mostrados na figura abaixo.

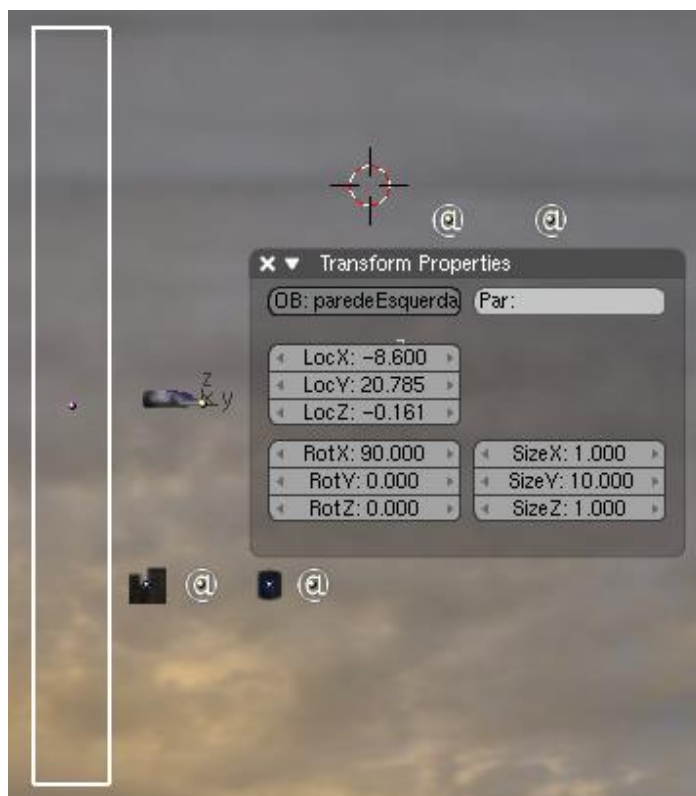


Figura 5-18. Valores de redimensionamento e posicionamento do cubo.

Agora, com o cubo ainda selecionado, pressione **FKEY**. – você irá mudar do modo Object para o modo UV Face Select. Divida a 3DView ao meio e mude a janela direita para UV/Image Editor. Abra o menu Image>>Open... e selecione com **BMM** o arquivo transp.png da pasta Cap06-JogoTiro\Texturas. Pressione **F9** para abrir o contexto Editing na ButtonsWindow. No painel Texture face, selecione a opção Alpha – isso irá ativar o Alpha Key que existe na imagem, tornando o objeto completamente transparente. Será?

Por mais que todas as faces do objeto estejam selecionadas, na verdade você está alterando apenas as propriedades da *face ativa* (como saber qual face está ativa? Na primeira vez que você entra no modo UV Face Select não dá pra saber, já que o Blender não dá nenhuma dica sobre isso...). Para transmitir a modificação para as demais faces, pressione o botão **Copy Draw Mode**, localizado na parte inferior do painel **Texture face**. Pronto, o cubo está completamente transparente e agora temos certeza que ele não irá aparecer na cena!

Precisamos fazer mais uma coisa: criar uma propriedade dentro do objeto `paredeEsquerda`. Pressione **FKEY** para sair do modo UV Face Select (só por garantia, não há a necessidade de sair dele para fazer esse passo) e, no contexto **Logics da ButtonsWindow (F4)**, clique uma vez no botão **ADD property**, localizado à esquerda da janela – irá surgir uma nova propriedade; renomeie-a para `limiteEsquerdo`.

Qual é a função dessa propriedade? Nós iremos dizer à nave que, quando for se movimentar para a esquerda, que faça um teste de colisão com qualquer objeto que contenha uma propriedade denominada `limiteEsquerdo` – mais sobre isso daqui a pouco.

Outra coisa que você deve reparar é que nomeamos o objeto como `paredeEsquerda` e a propriedade como `limiteEsquerdo`. Isso foi feito apenas para ilustrar que o objeto e as propriedades que ele contém são coisas diferentes – poderíamos ter colocado o mesmo nome, se quiséssemos, que o Blender não reclamaria.

Um terceiro detalhe: não importa o tipo de valor da propriedade – tudo que nós iremos utilizar será o nome.

Com o objeto `paredeEsquerda` selecionado e no modo **Object**, duplique-o pressionando **SHIFT-D** – o novo objeto entrará automaticamente no modo de movimentação. Pressione **BDM** para deixá-lo na mesma posição do objeto original – nós iremos fazer a modificação numericamente.

Se a janela **Transform properties** não estiver aberta, abra-a pressionando **NKEY**. Renomeie o objeto para `paredeDireita` e, no campo **LocX**, insira o valor 8. No contexto **Logics da ButtonsWindow**, renomeie a propriedade para `limiteDireito`.

Pressione **SHIFT-D** para duplicar o objeto `paredeDireita`, renomeie a cópia para `paredeSuperior` e coloque os seguintes valores:

- **LocX:** 0;
- **LocZ:** 7.3;
- **RotY:** 90 (o objeto irá rotacionar noventa graus no eixo Y).

Os demais campos devem permanecer como estão. No final, renomeie a propriedade deste objeto para `limiteSuperior`.

E, por fim, Pressione **SHIFT-D** para duplicar o objeto `paredeSuperior`, renomeie a cópia para `paredeInferior` e modifique **LocZ** para -7.3. Renomeie a propriedade deste objeto para `limiteInferior` e pronto – estamos preparados para começar a brincadeira (**Figura 5-19**)!



Figura 5-19. As paredes devidamente posicionadas e texturizadas.

Selecione a nave do jogador e crie mais quatro Sensors, modificando-os para Collision Sensors. Além da primeira linha de botões que é idêntica para todos os Sensors, o Collision Sensor possui uma segunda fileira contendo dois botões:

M/P: quando este botão não está pressionado, o dono do Collision Sensor verifica se ele colidiu com algum objeto na cena que contenha uma propriedade específica. Se o botão estiver pressionado, será feita uma verificação de colisão com algum outro objeto contendo um material específico.

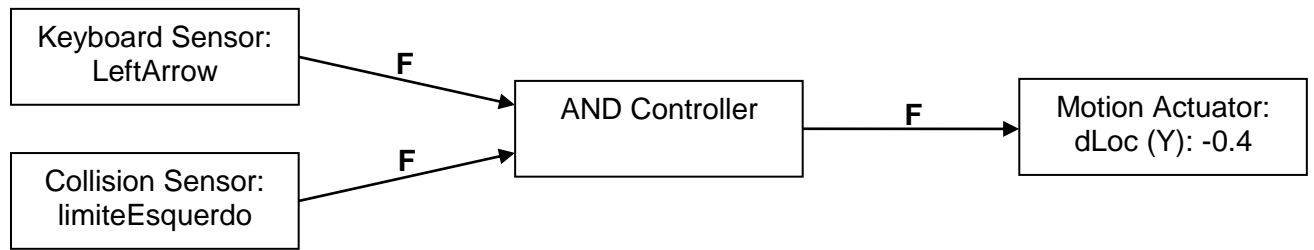
Property: indica qual a propriedade que será verificada nos objetos para testar a colisão – apenas objetos que contenham tal propriedade serão testados. Se o botão M/P estiver pressionado, este campo mudará para Material.

Renomeie os Sensors para limiteSuperior, limiteInferior, limiteEsquerdo e limiteDireito. No campo Property, repita o nome do Sensor – se, por exemplo, o Sensor foi nomeado limiteSuperior, o seu campo Property também deverá ser nomeado limiteSuperior. Lembre-se que o nome do Sensor não irá influir no seu funcionamento – estamos renomeando-os apenas por uma questão de organização (se você for usar Python, aí sim o nome do LogicBrick é importante, mas não é o nosso caso).

Agora, verifique se os quatro Controllers inseridos lá no começo do tópico são AND Controllers – como a opção AND é a opção padrão, tudo já deve estar como queremos. Agora, faça os seguintes passos:

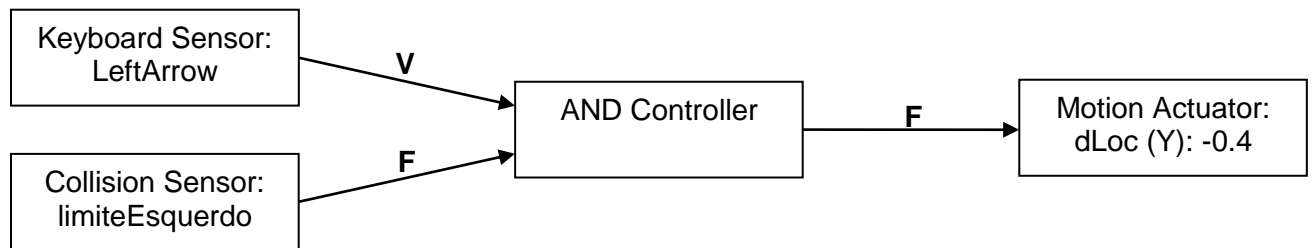
- Conecte o Sensor limiteSuperior ao mesmo Controller em que o Sensor Cima está conectado;
- Conecte o Sensor limiteInferior ao mesmo Controller em que o Sensor Baixo está conectado;
- Conecte o Sensor limiteEsquerdo ao mesmo Controller em que o Sensor Esquerda está conectado;
- Conecte o Sensor limiteDireito ao mesmo Controller em que o Sensor Direita está conectado.

Um último detalhe importante antes de toda essa bagunça funcionar: você deve pressionar o botão Inv dos quatro Collision Sensors. O porquê disso? Acompanhe o esquema abaixo:



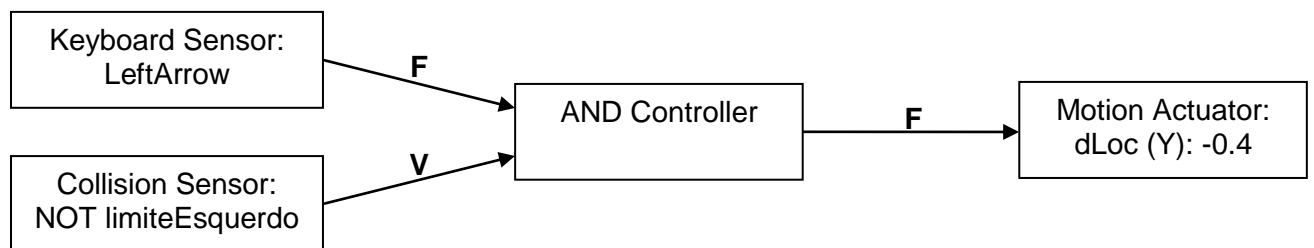
Este é o estado inicial do esquema, ou seja, o jogador não está pressionando **SETAESQUERDA** e a nave não colidiu com o objeto *paredeEsquerda*, dono da propriedade *limiteEsquerdo*. Se você se recorda do capítulo 3, FALSO **E** FALSO resulta em FALSO.

No entanto, veja o que acontece quando o jogador pressiona **SETAESQUERDA**:



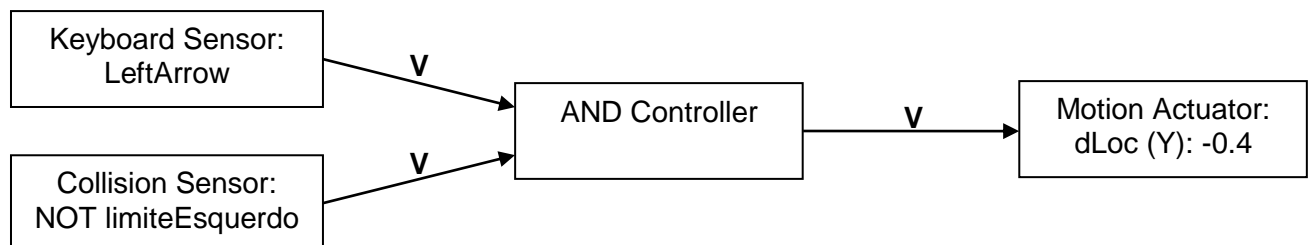
O Keyboard Sensor muda para VERDADEIRO; no entanto, como a nave *não* está colidindo com o objeto *paredeEsquerda*, o Collision Sensor permanece FALSO. Portanto, VERDADEIRO **E** FALSO continua sendo FALSO – dessa forma a nave *nunca* irá sair do lugar!

Vamos, agora, inverter o resultado do Collision Sensor para ver o que acontece:

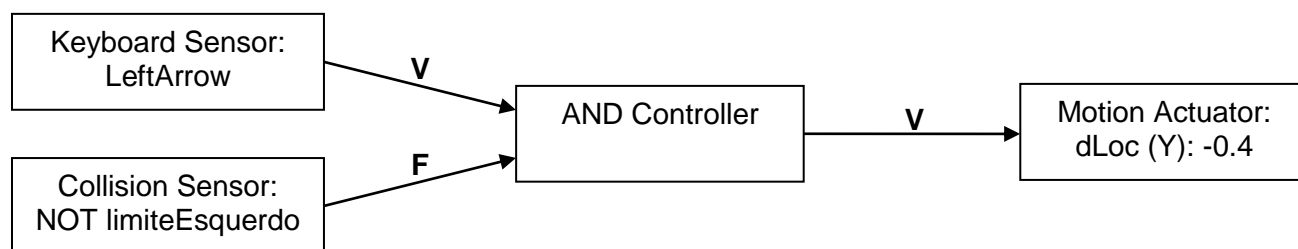


O esquema mostra o novo estado inicial: como o teste de colisão inicia negativo, o resultado é invertido, mandando um sinal VERDADEIRO ao invés de FALSO. Como FALSO **E** VERDADEIRO resulta em FALSO, a nave não sai do lugar.

Vamos pressionar **SETAESQUERDA** para ver o que acontece:



Enquanto **SETAESQUERDA** estiver pressionado e a nave não estiver colidindo com o objeto *paredeEsquerda*, a nave irá se movimentar! Veja agora o que acontece quando a nave colide com *paredeEsquerda*:



Como o sinal do Collision Sensor está invertido, assim que a nave colidir com algum objeto que possua a propriedade `limiteEsquerdo` (paredeEsquerda, no nosso caso), o Sensor irá disparar um sinal de FALSO, tornando FALSO o resultado do Controller – o que é exatamente aquilo que queríamos que acontecesse!

Simples, não? Confira nas figuras abaixo como fica o esquema final de movimentação da nave:

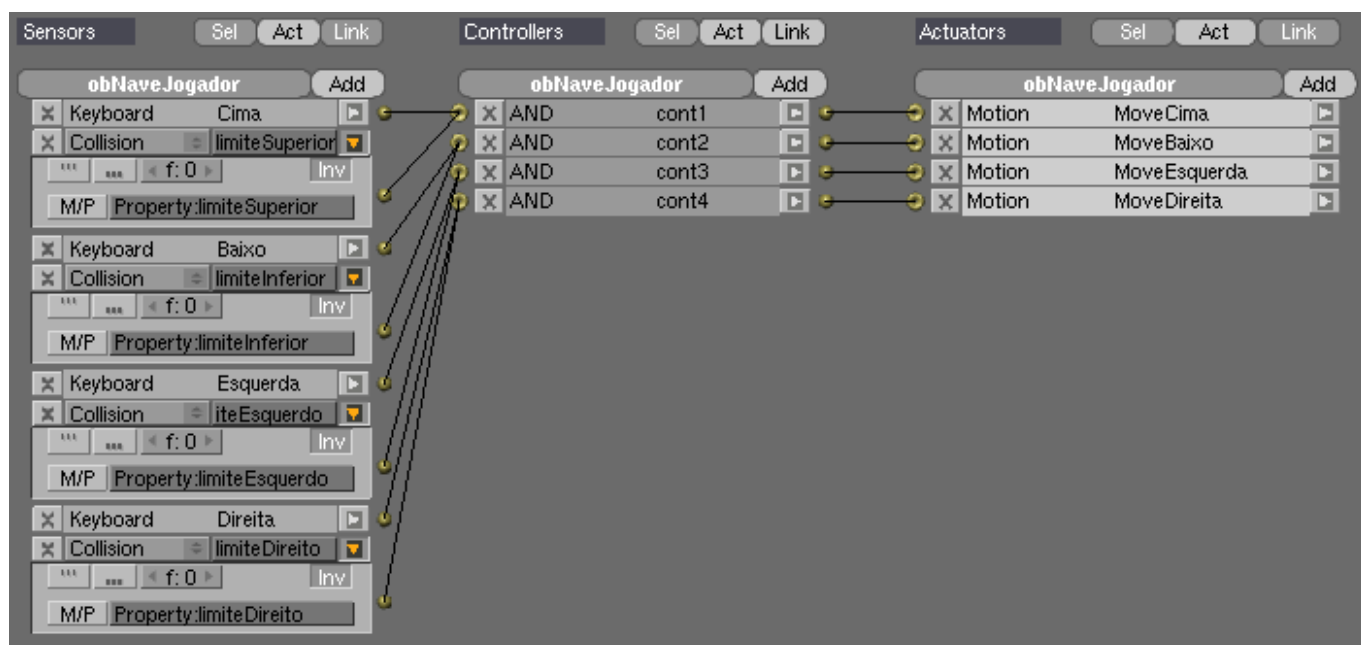


Figura 5-20. Lógica final de movimentação da nave.

5.3. Disparando as naves inimigas

Lembra do emissor de naves? Iremos fazê-lo funcionar agora, tendo em mente duas coisas:

1. O emissor deve criar uma sequência de naves;
2. Deve existir um intervalo entre uma sequência e outra.

Essa é uma das partes mais complexas do jogo, então trate de providenciar um pouco do seu energizador de neurônios preferido – guaraná com catuaba, café polonês, taco de baseball, enfim – para conseguir prestar atenção!

Pronto? Então vamos começar. :-)

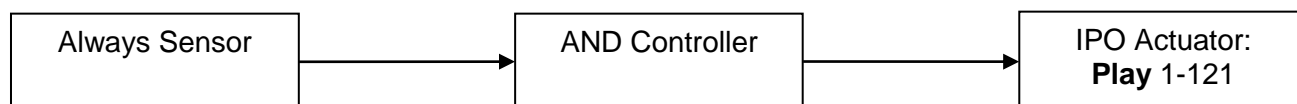
A lógica que nós iremos criar irá envolver 7 Sensors, 6 Controllers e 10 Actuators – portanto, o melhor a fazer é dividir os processos internos do emissor de naves.

A regra de funcionamento do emissor será a seguinte:

1. O emissor ficará fazendo um movimento ininterrupto de sobe-e-desce;
2. De tempos em tempos o emissor irá lançar um determinado número de naves. Após ter lançado esse número de naves, ele deverá entrar num período de intervalo;
3. Terminado o período de intervalo, ele irá lançar uma nova bateria de naves.

O item 1 acontece sem parar e independente dos outros itens; os itens 2 e 3 acontecem de forma intercalada, também sem parar – ou seja, primeiro o item 2, depois o 3, depois o 2, depois o 3...

Vamos, então, analisar como transformar tudo isso em LogicBricks. O item 1 é o mais fácil de todos:



- O Always Sensor é um Sensor especial que dispara um sinal de VERDADEIRO a cada quadro de animação do jogo;
- Como só temos um Sensor, tanto faz o tipo de Controller;
- Já o IPO Actuator roda parte da ou toda a animação do objeto, criada na janela Ipo Curve Editor. Existem várias maneiras de rodar a animação, mas, no nosso caso, vamos usar o método Play, que roda a animação em loop a partir do frame especificado como inicial ate o frame especificado como final, voltando para o frame inicial e rodando novamente a animação desde o começo, *ad eternum*. (Figura 5.21).

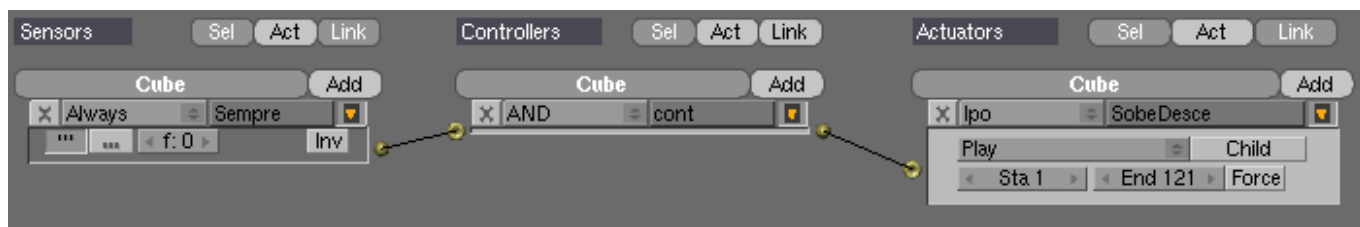


Figura 5-21. Lógica da animação do emissor de naves.

Os itens 2 e 3 necessitam de um pouco de preparativos. Para tanto, vamos criar quatro propriedades:

- count [Int; valor inicial: 0] – faz a contagem de tempo entre uma nave e outra, ambas da mesma bateria;
- intervalo [Bool; valor inicial: false] – indica se o intervalo entre duas baterias está acontecendo (true) ou não (false);
- naveN [Int; valor inicial: 1] – marca a posição de uma nave dentro de uma bateria;
- countIntervalo [Int; valor inicial: 0] – faz a contagem de tempo entre uma bateria de naves e outra.

A figura a seguir mostra as quatro propriedades no Blender:

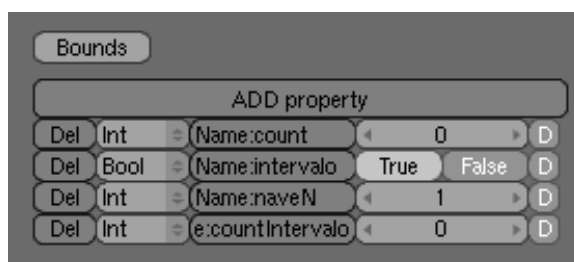


Figura 5-22. As propriedades do emissor de naves.

Só mais um detalhe: clique no botão “D” ao lado de cada propriedade para que elas façam parte das informações de debug da cena (o “D” do botão faz referência à palavra *Debug*). Agora, no menu superior do Blender, acesse o menu Game>>Show Debug Properties, para que elas apareçam na tela e você possa saber quais valores as propriedades estão armazenando num determinado momento.

Vamos, então, ao destrinche do item 2. Para melhor entendê-lo, vamos dividi-lo em três sub-itens:

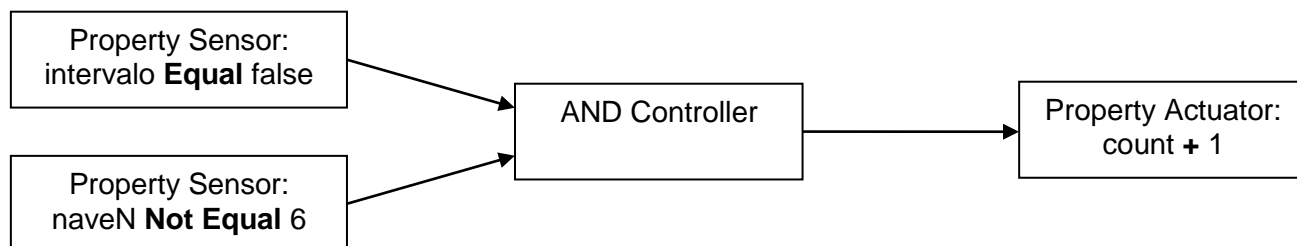
- 2.1. Se o intervalo não estiver acontecendo **E** as naves de uma bateria ainda não tiverem acabado, então adicione 1 ao contador de tempo entre uma nave e outra (count, no nosso caso);
- 2.2. Se o contador atingir o valor limite, retorne-o para o valor 0, dispare uma nave e adicione 1 ao contador de naves (naveN, no nosso caso);

2.3. Se o contador de naves atingir o valor limite, retorne-o para 1 e avise ao emissor de naves que o período de intervalo começou.

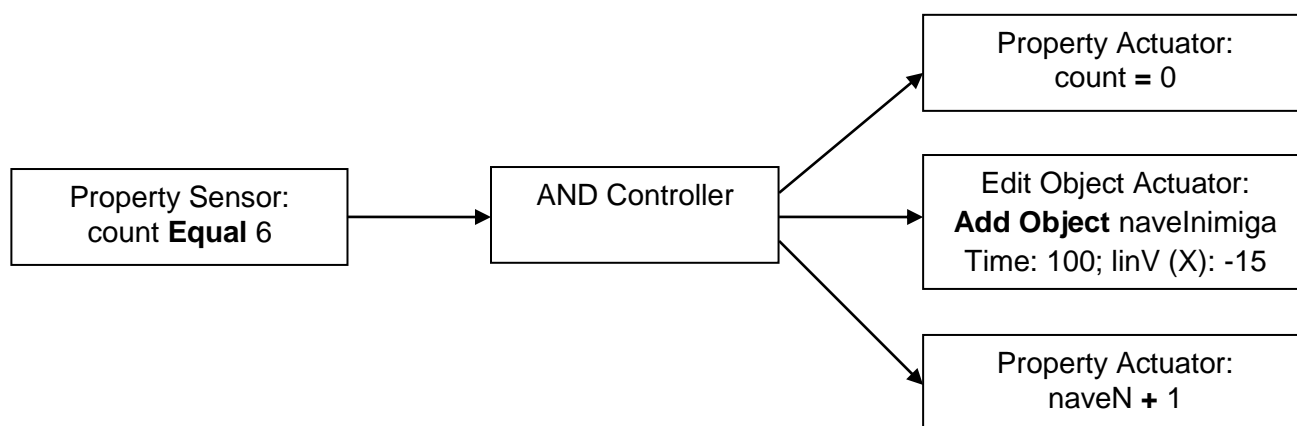
Por que no item **2.1** temos que fazer a verificação da ocorrência ou não do intervalo? Os itens **2** e **3** definem estados do emissor de partículas – portanto, não podem estar acontecendo simultaneamente. Dessa forma, podemos manter uma propriedade como chaveador de estados, alterando o seu valor para definir qual estado está ocorrendo num dado momento.

Os esquemas dos itens **2.1**, **2.2** e **2.3** são mostrados a seguir, levando em conta que `count` terá um valor máximo de 6 e `naveN` terá um valor máximo de 5:

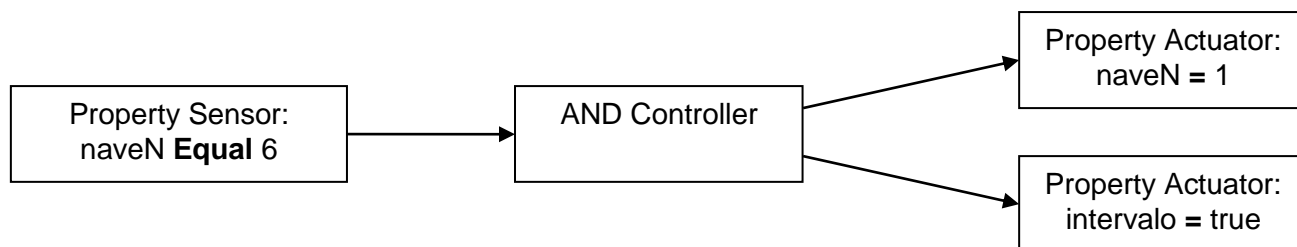
2.1.



2.2.



2.3.



Um *Property Sensor* faz testes com as propriedades do objeto, disparando `true` caso o teste seja verdadeiro, ou `false`, em caso contrário. Em **2.1**, por exemplo, o primeiro *Property Sensor* está verificando se a propriedade `intervalo` está armazenando o valor `false` naquele momento (comando `Equal`); já o segundo *Property Sensor* verifica se `naveN` não é igual a 6 naquele momento (comando `Not Equal`).

Mas espera um pouco: não havíamos dito que `naveN` não passaria de 5? Então porque testá-lo contra o valor 6?

Fica mais fácil de entender a lógica observando o sub-item **2.3**: se naveN for igual a 6, então ele será reconfigurado para 1; dessa forma, naveN recebe o valor 6 por uma fração infinitesimal de tempo – é como se a propriedade nunca tivesse recebido esse valor!

Vamos tentar entender isso de outro jeito. Acompanhemos a evolução da propriedade naveN, passo a passo:

- **naveN = 1**: primeira nave é disparada;
- **naveN = 2**: segunda nave é disparada;
- **naveN = 3**: terceira nave é disparada;
- **naveN = 4**: quarta nave é disparada;
- **naveN = 5**: quinta nave é disparada;
- **naveN = 6**: naveN volta para 1; começa o intervalo.

Resumindo: os testes devem ser feitos em cima do número de naves da bateria mais um, para que toda a lógica de transição do estado de emissão de naves para o estado de intervalo possa acontecer. Entendeu? ;-)

As três figuras seguintes mostram como os sub-itens **2.1**, **2.2** e **2.3** ficam ao serem inseridos no Blender:

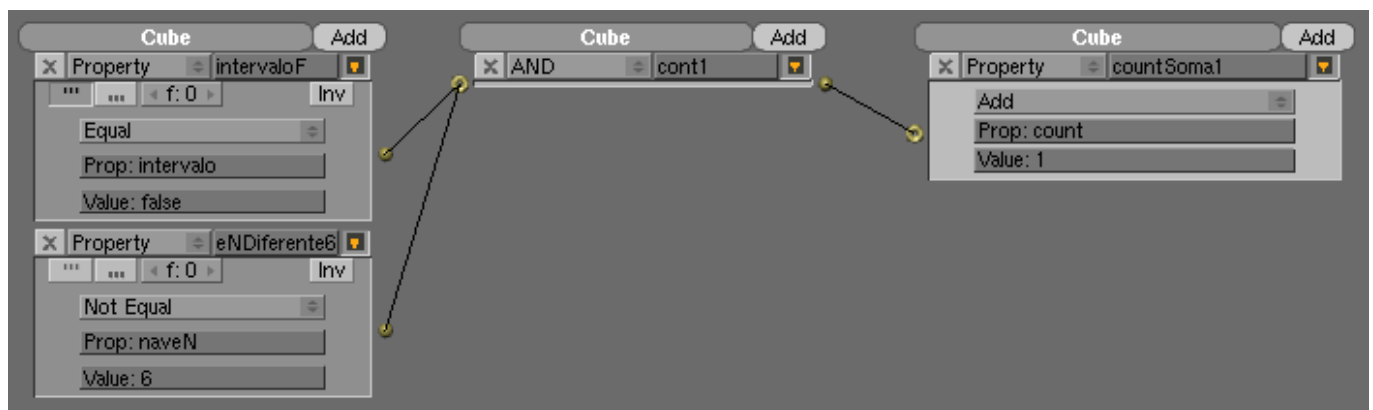


Figura 5-23. Sub-item 2.1.

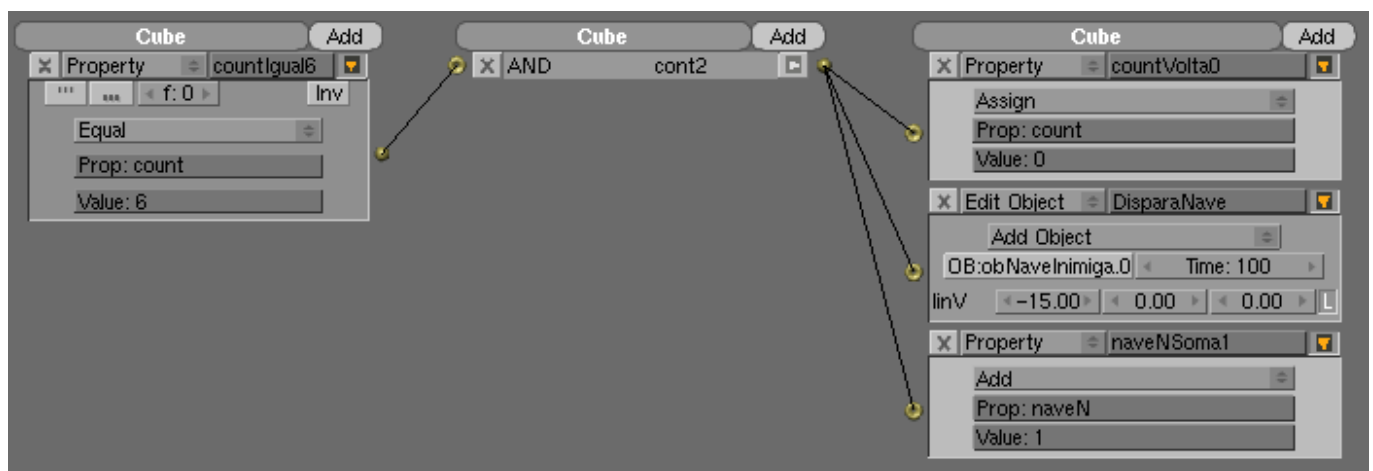


Figura 5-24. Sub-item 2.2.

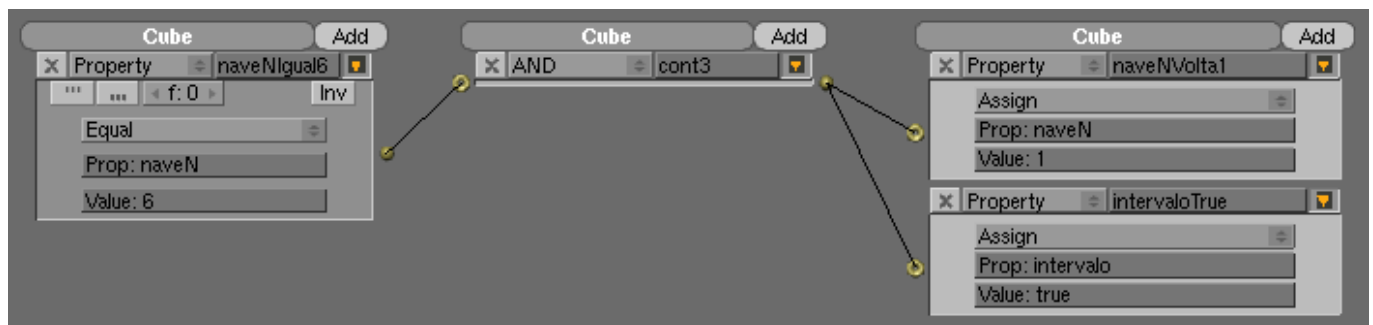


Figura 5-25. Sub-item 2.3.

Hein?! Vitrola...

Existe um detalhe importante na Figura xxx: se você reparar bem no Property Sensor denominado `intervaloF`, verá que o botão `Activate TRUE Pulse Mode` (botão com os três pontinhos na parte superior) está pressionado. Eu fui obrigado a ativá-lo, senão as coisas não funcionariam. Isso é um detalhe muito mal documentado no Blender; então, para tentar explicá-lo, vou ter que usar todo o meu lado detetive e disparar deduções.

Isso acontece normalmente quando vai se verificar propriedades do tipo `Bool`, e o motivo provavelmente é o seguinte: um Property Sensor só é disparado quando o valor da propriedade que ela está testando é modificado. Por isso que tudo ocorre normalmente quando se testa as propriedades `name` e `count` – elas são constantemente modificadas, disparando constantemente seus Property Sensors; no entanto, a propriedade `intervalo` muda de vez em quando, mas nós continuamos precisando que o Property Sensor seja disparado todo quadro! É aí que entra o botão `Activate TRUE Pulse Mode` (Ativar Modo de Pulso VERDADEIRO): dessa forma, o Sensor dispara um sinal todo quadro de animação de jogo, independente do valor da propriedade ter sido modificado ou não.

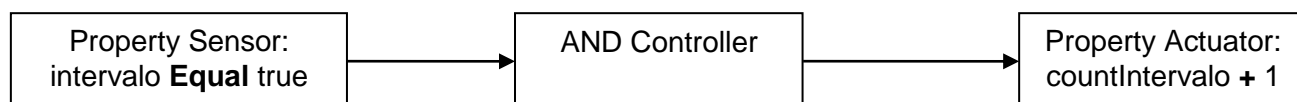
O Blender provavelmente mantém variáveis de estado para cada propriedade de cada objeto inserido na cena e os Property Sensors devem estar vinculados à modificação dessas variáveis. Entendeu? Pois é, eu também não. Mas, enfim, são coisas da vida... E de repente eu só falei abobrinha aqui... ;-)

Chega de papo furado e sigamos em frente! Agora, iremos dividir o item **3** em dois sub-itens:

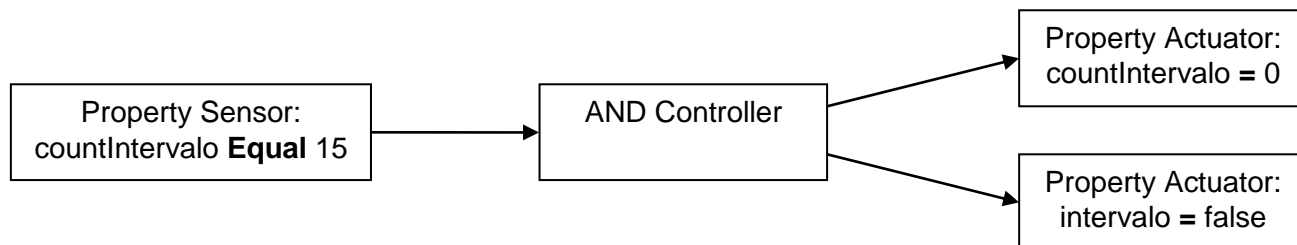
- 3.1.** Se o intervalo estiver acontecendo, então adicione 1 ao contador de tempo do intervalo (`countIntervalo`, no nosso caso);
- 3.2.** Se o contador atingir o valor limite, retorne-o para o valor 0 e diga que o intervalo acabou.

Aqui estão os esquemas de **3.1** e **3.2** em toda sua intimidade, levando em conta que `countIntervalo` terá um valor máximo de 15:

3.1.



3.2.



Não há muito que comentar aqui: todos os fatos relevantes já foram esclarecidos. Vejamos, então, como tudo isso fica no Blender:



Figura 5-26. Sub-item 3.1.

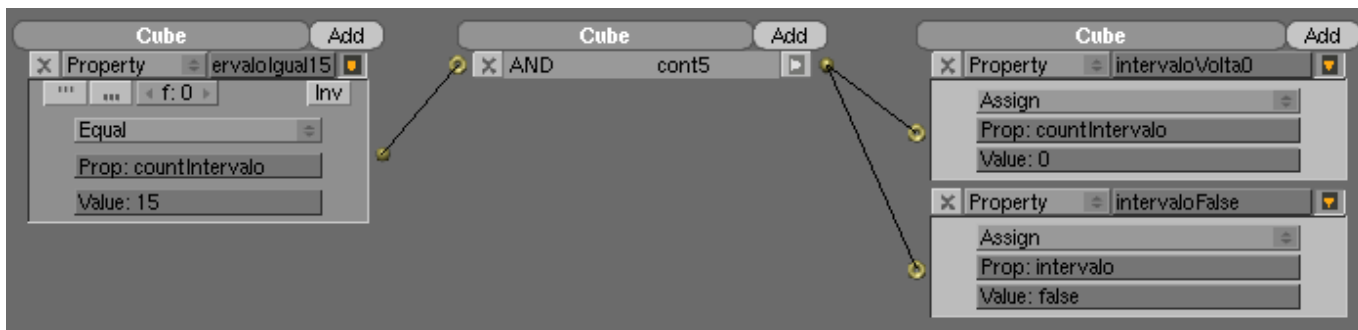


Figura 5-27. Sub-item 3.2.

O único fato a ser comentado aqui é que o Property Sensor denominado `intervaloV`, da mesma forma que o Property Sensor `IntervaloF`, está com o botão `Activate TRUE Pulse Mode` pressionado. Fora isso, todo o resto deve estar claro pra você a esta altura.

O quê?! Ainda não? Então trate de ler e praticar este tópico antes de passarmos para o próximo!

5.4. Chumbo neles!

A lógica do tiro é basicamente a mesma do emissor de naves. No entanto, vamos fazer a mesma coisa de outra maneira.

Selecione a nave do jogador com **BDM**, ative o contexto **Logic (F4)** da ButtonsWindow e, fora o que já existe de lógica, adicione mais um Sensor, um Controller, um Actuator e conecte-os. Mude o Sensor para um Keyboard Sensor e o Actuator para um Edit Object Actuator. Na opção **Key** do Keyboard Sensor, pressione **BARRAESPAÇO** para que ela seja a tecla ativadora do tiro; na opção **OB** do Edit Object Actuator, digite o nome do objeto referente ao tiro (`obTiro`, no caso), mude **Time** para 100 e **linV (Y)** para 20. A figura seguinte mostra como fica a lógica no Blender:

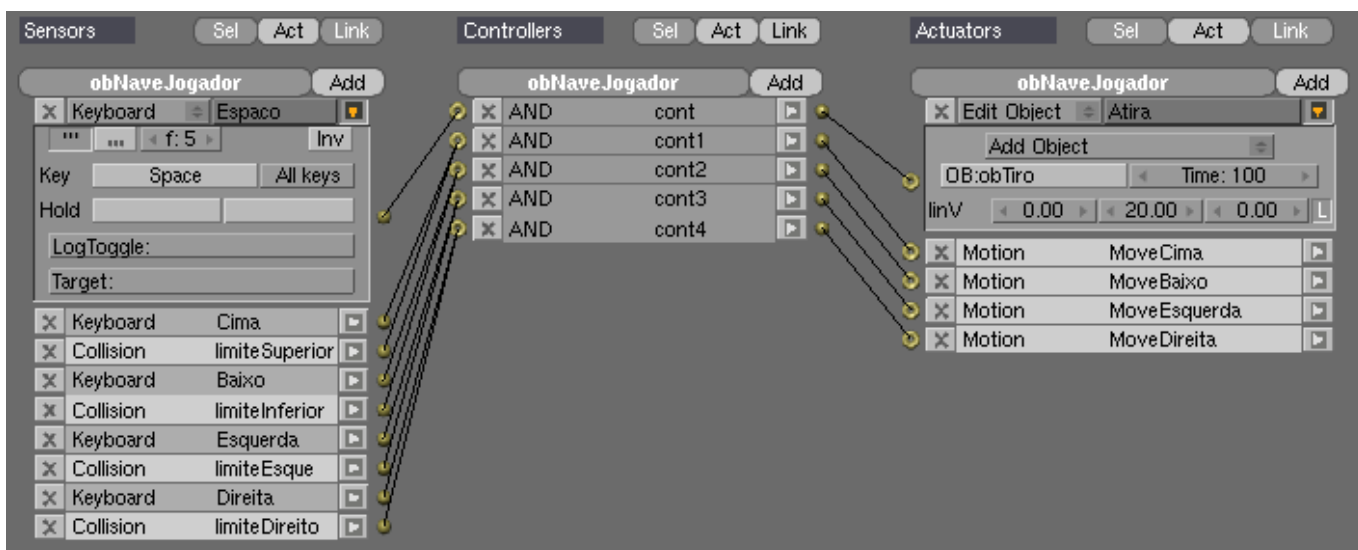


Figura 5-28. Lógica de disparo do tiro.

Se você testar o jogo agora (**PKEY**) verá que, de fato, quando se pressiona **BARRAESPAÇO**, um tiro sai da nave. No entanto, o único jeito de se disparar tiros seguidos é pressionando repetidamente **BARRAESPAÇO**. Não seria legal dar a possibilidade ao jogador de segurar **BARRAESPAÇO** e os tiros saírem continuamente?

Pois bem, então façamos isso. Pressione o botão **Activate TRUE Pulse Mode** do **Keyboard Sensor** e altere o valor do botão f , mais à direita, de 0 para 5 (Figura xxx). Esse botão define a frequência com que o Sensor será ativado, em 1/50 de segundo. Traduzindo: se o valor de f for 5, por exemplo, o Sensor será ativado dez vezes por segundo, já que 5/50 é igual a 1/10 ou 0,1 segundo. Se o valor de f for 10, então o Sensor será ativado cinco vezes por segundo, pois 10/50 é igual a 1/5 ou 0,2 segundo. E por aí vai.

Agora, se você testar mais uma vez o jogo, verá que tanto segurando quanto pressionando repetidamente **BARRAESPAÇO** os tiros saem em sequência. Claro que poderíamos complicar um pouco mais a situação, definindo um número máximo de tiros que podem permanecer na tela; mas, como já complicamos bastante no tópico anterior, não custa nada dar uma folguinha agora...

Mais uma coisinha... Que tal colocarmos um som de tiro toda vez que o jogador pressionar **BARRAESPAÇO**? Pois bem, pressione **F10** para acessar o contexto **Scene (cena)** da **ButtonsWindow**; em seguida, selecione o subcontexto **Sound Block Buttons** (**Figura 5-29**).

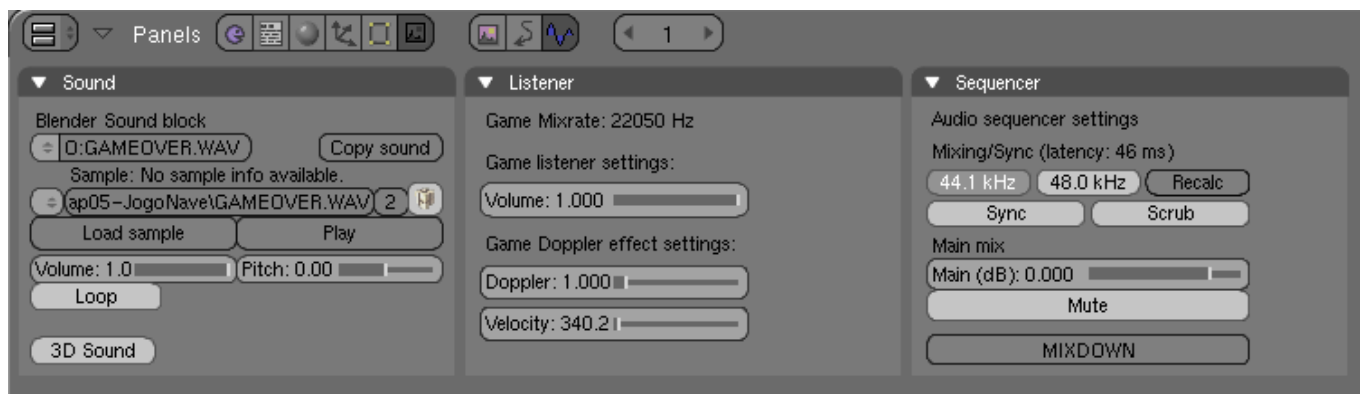


Figura 5-29. O subcontexto Sound Block Buttons.

No painel **Sound**, à esquerda, clique na seta dupla abaixo de **Blender Sound Block** e selecione a opção **Open New**. Na janela **FileBrowser**, procure pelo arquivo **Tiro.wav** na pasta **Cap05-JogoNave\Sons**, selecione-o com **BEM** e clique em **SELECT WAV FILE**. Para escutar o som, clique no botão **Load Sample**, bem no meio do painel **Sound** (localize-o na **Figura 5-29**).

Agora que carregamos o som dentro do projeto, basta criar a lógica que irá dispará-lo. Pressione **F4** para retornar ao contexto **Logic** e selecione a nave do jogador. Adicione um novo **Actuator** e mude-o para um **Sound Actuator**; clique no único botão que aparece no **Actuator** e selecione o arquivo **Tiro.wav**. Mude o comando **Play Stop** para **Play End** – assim, o som será reproduzido até o fim, independente da tecla que o disparou não estar mais pressionada. Por fim, conecte-o ao mesmo **Controller** que o **Edit Object Actuator** responsável pela criação do tiro está conectado, como mostra a figura abaixo:

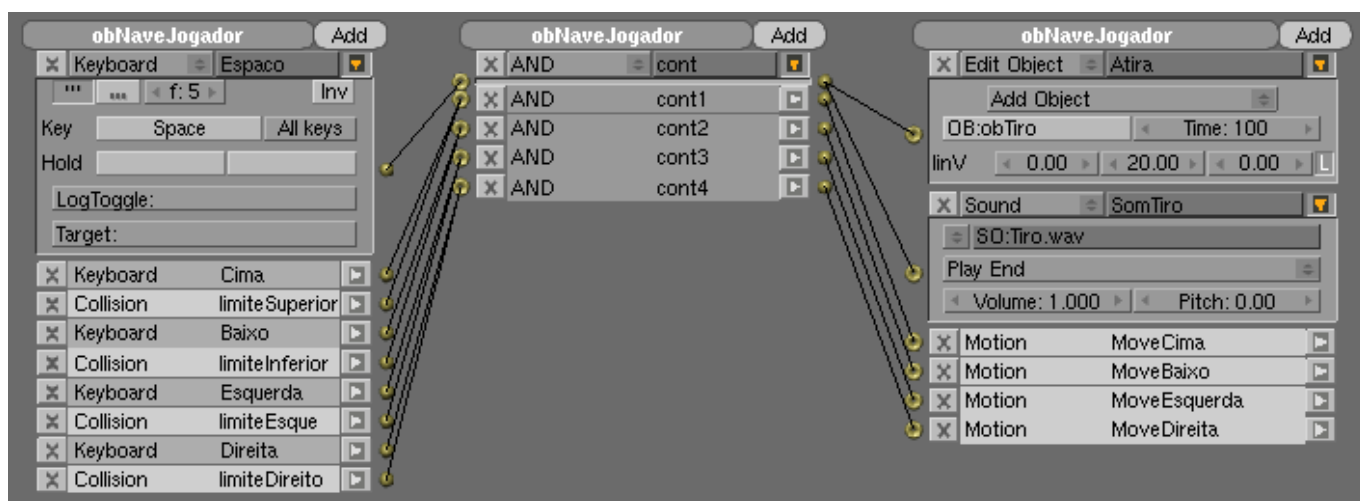


Figura 5-30. O Sound Actuator foi conectado ao mesmo Controller do Edit Object Actuator responsável pelo tiro.

5.5. Mayday, mayday! Dez maneiras de se destruir a sua própria nave...

Tudo bem que serão mostradas apenas duas maneiras de se destruir as naves, mas enfim... Existem mil maneiras de se destruir uma nave... Invente uma! ;-)

Não só a sua nave, mas também as naves inimigas devem explodir quando atingidas. Se você achou que teríamos efeitos pirotécnicos Spielbergianos de última geração, vai com calma: tudo que iremos fazer será ativar a animação de uma esfera quando uma nave explodir – a animação já está pronta na camada escondida (alterne para ela com **ALT-1** e volte para a camada correta pressionando **1KEY**).

Vamos começar com a colisão entre os tiros e as naves inimigas. Mas, antes de testar as colisões, precisamos inserir uma propriedade em cada um deles para que o tiro encontre a nave e vice-versa. Vá para a camada escondida, selecione o modelo do tiro com **BDM** (ele é pequenininho e está do lado esquerdo da esfera que será a explosão) e, no contexto Logic (**F4**), adicione uma propriedade chamada **tiro**. Agora, selecione a nave inimiga (é a nave virada para a direita, à direita da explosão) e adicione a ela uma propriedade chamada **naveInimiga**.

Selecione novamente o tiro e insira a seguinte lógica nele:



Figura 5-31. Teste de colisão do tiro com as naves inimigas.

A única coisa que isso faz é autodestruir o tiro que colidiu com uma nave. Agora, com a nave inimiga selecionada, adicione a seguinte lógica:



Figura 5-32. Teste de colisão da nave inimiga com o tiro.

Além de se autodestruir, a nave cria uma instância da explosão no local onde ela colidiu. O tempo de vida deve ser de 24 frames, o dobro dos 12 frames de animação da explosão. O porquê disso? Também gostaria de saber...

Selecione a explosão (a esfera entre o tiro e a nave inimiga) e insira o seguinte nela:



Figura 5-33. Lógica de animação da explosão.

O único fato que ocorre aqui é a reprodução da explosão. No entanto, a reprodução só é iniciada quando se cria uma instância da explosão – ou seja, a explosão só ocorre no momento em que a nave é atingida. Simples, não?

Resolvido o problema entre os tiros e as naves inimigas, vamos nos concentrar agora na colisão entre a nave do jogador e as naves inimigas. A lógica da colisão em si é simples, mas existe uma coisinha que vai complicar um pouco: é importante que o jogador tenha uma dica de que sua nave foi atingida, seja visual ou auditiva. No nosso caso, vamos mostrar ao jogador que a nave dele foi atingida com uma modificação na textura da nave – você já deve ter reparado que existe mais uma nave na camada escondida, vermelha e virada para a direita. Pois bem, no momento em que a nave do jogador for atingida, nós iremos substituí-la temporariamente por esse modelo.

Precisaremos adicionar duas propriedades à nave do jogador para que as coisas funcionem (**Figura 5-34**):

- `count` [Int; valor inicial: 0] – conta o tempo que a nave ficará vermelha;
- `naveJogador` [tipo de valor indiferente] – usado pela nave inimiga no teste de colisão;

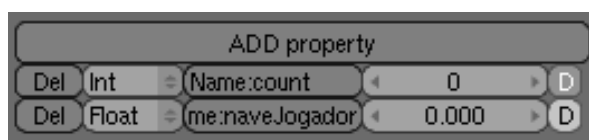


Figura 5-34. Propriedades do objeto `obNaveJogador`.

Agora, iremos dividir a lógica de colisão da nave do jogador em três partes:

1. Se a nave do jogador colidir com uma nave inimiga, atribua à propriedade `count` o valor 1 e troque a nave normal pela nave;
2. Se `count` for diferente de 0, então adicione 1 à propriedade `count` (ou seja, quando `count` for diferente de 0, a contagem de tempo se inicia);
3. Se a propriedade `count` atingir um valor limite (10, no nosso caso), então troque a nave vermelha pela nave normal e volte a propriedade `count` para 0;

As figuras seguintes mostram cada uma das etapas implementadas no Blender:

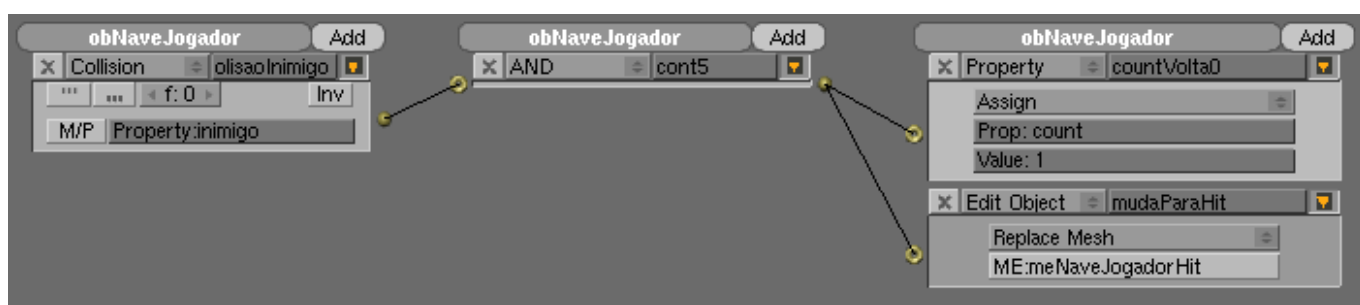


Figura 5-35. Etapa 1.

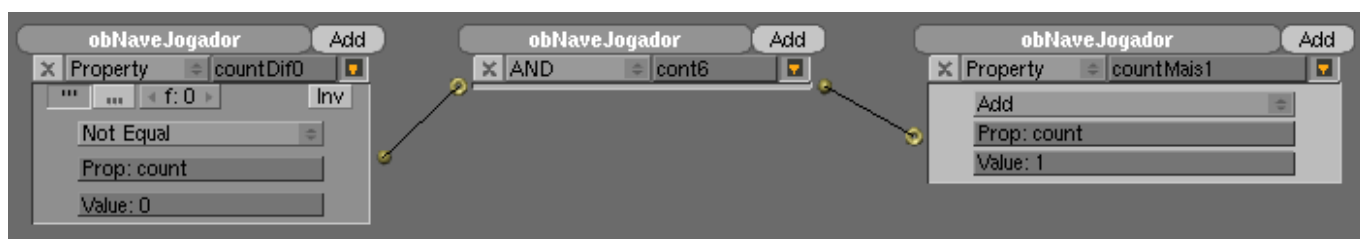


Figura 5-36. Etapa 2.



Figura 5-37. Etapa 3.

O único Logicbrick que precisa ser comentado é o Edit Object Actuador com o comando `Replace Mesh` ativado. Com a nave do jogador selecionada (volte para a camada correta pressionando **1KEY**), pressione **F9** para mudar a ButtonsWindow para o contexto Editing. À esquerda, no painel Link and Materials, você verá logo na parte de cima um campo chamado **ME** (**Figura 5-38**). No Blender, o objeto em si e a malha utilizada para representá-lo são considerados elementos diferentes. Isso permite que a malha de um objeto seja trocada, sem que se alterem outras propriedades suas (como a lógica, por exemplo).

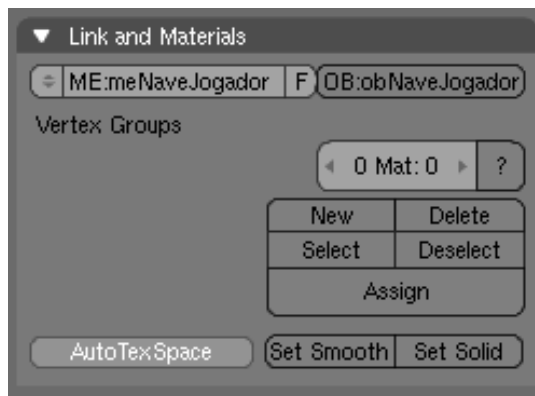


Figura 5-38. A propriedade ME no painel Link and Materials.

O comando Replace Mesh possui apenas um campo, ME, onde deve ser colocado o nome da malha que irá substituir a malha atual. Lembre-se, para saber o nome da malha de um objeto, selecione-o e pressione **F9** para abrir o contexto Editing. Eu sugiro que você renomeie a malha da nave do jogador para meNaveJogador e a malha da nave vermelha para meNaveJogadorHit, que são os nomes utilizados nas figuras acima (para renomear o nome da malha, basta clicar sobre ele no painel Link and Materials e modificá-lo).

A lógica da nave inimiga não tem segredos, como mostra a figura abaixo:

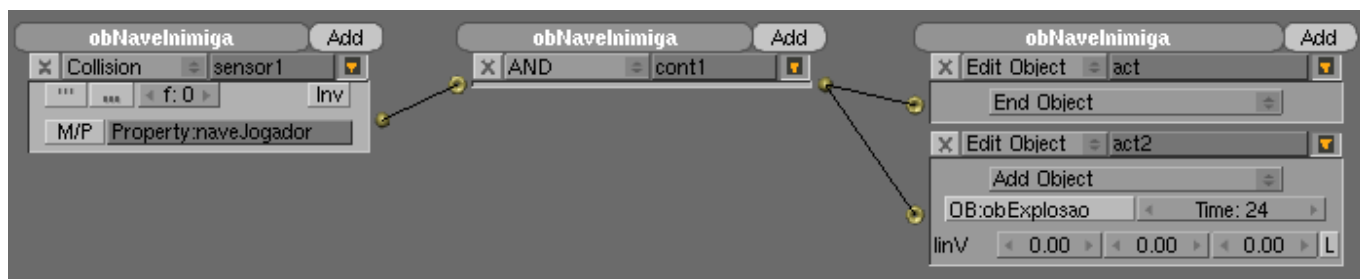


Figura 5-39. A lógica de colisão da nave inimiga com a nave do jogador.

Simplesmente repetimos o processo relativo ao tiro. E chega de colisão que eu já estou ficando com dor de cabeça...

5.6. Você é a última fronteira entre eles e a cidade...

Quando uma nave inimiga consegue passar por você e alcançar o lado esquerdo da tela, uma cidade deverá ser destruída. Essa parte será fácil: não tem nada além do que já fizemos até agora.

A lógica de redução do número de cidades será colocado no marcador do número de cidades, circunscrito em vermelho na figura abaixo:



Figura 5-40. Oi, eu sou o marcador de cidades...

Antes, precisamos fazer duas coisas. Primeiro, selecione o objeto `paredeEsquerda` e duplique-o com **SHIFT-D**. Pressione **XKEY** para restringir o deslocamento do objeto ao eixo X e movimente-o para a esquerda do objeto original (mas não muito; deixe apenas um pequeno espaço entre eles). No contexto Logic, altere o nome da propriedade `limiteEsquerdo` para `limiteCidades`.

Segunda coisa: mude para a camada escondida (**ALT-1**), selecione a nave inimiga e adicione um Sensor, um Controller e um Actuator; modifique o Sensor para um Collision Sensor, insira `limiteCidades` no campo Property e conecte-o com o Controller recém-criado; modifique o Actuator para um Message Actuator, insira `CidadeMenos1` no campo Subject e, por fim, conecte o Controller recém-criado aos três Actuators, como mostra a figura abaixo:

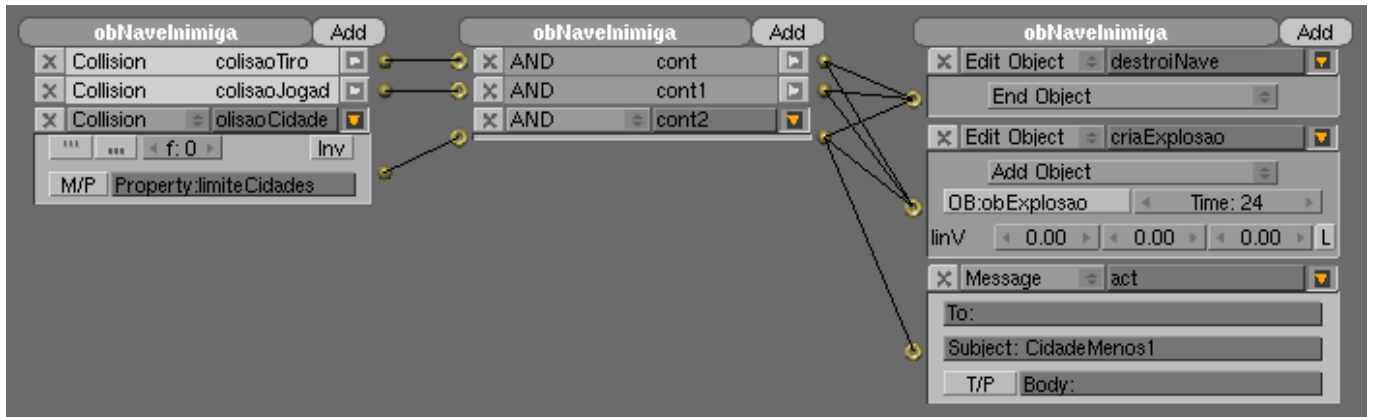


Figura 5-41. Finalizando a lógica da nave inimiga.

Dessa forma, a nave irá explodir quando houver a colisão, além de enviar uma mensagem avisando que destruiu uma cidade.

Do Message Actuator normalmente iremos utilizar apenas o campo Subject (assunto), que transmite a mensagem para todos os objetos da cena. Se você quiser ser mais específico, pode indicar qual objeto deverá receber a mensagem inserindo seu nome no campo To (para). Para receber mensagens, os objetos devem ter Message Sensors especificando quais mensagens pretendem capturar.

Agora podemos ir ao que interessa. Selecione o marcador de cidades e, no contexto Logics, adicione duas propriedades: uma do tipo Int chamada `Text`, que irá marcar a quantidade de cidades restantes, e outra do tipo Bool chamada `continuaSubtraindo`, que irá verificar se o número de cidades chegou a 0. Modifique o valor inicial de `Text` de 0 para 5 (o número inicial de cidades) e de `continuaSubtraindo` para true, como mostra a figura abaixo:

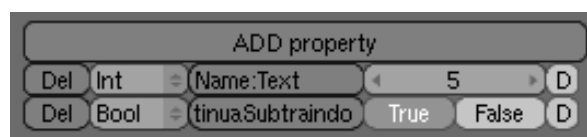


Figura 5-42. Propriedades do marcador de cidades.

Com tudo preparado, podemos inserir a seguinte lógica no objeto:



Figura 5-43. Lógica de redução do número de cidades.

Repare que no campo Subject do Message Sensor está escrito *exatamente* o assunto da mensagem que será recebida por ele. E, quando eu digo exatamente, é exatamente mesmo – inclusive maiúsculas e minúsculas. No Property Actuator, é adicionado -1 à propriedade, que seria o mesmo que subtrair 1 dela (o Property Actuator não possui um comando Subtract).

Ufa, que parecia que isso não ia acabar nunca! Agora que toda a lógica do jogo está no lugar só falta criar as mensagens que irão modificar os marcadores e definir o estado de Game Over.

Ainda está comigo? Então vamos, que estamos chegando ao fim!

5.7. Alterando os marcadores

Um dos últimos detalhes a ser tratado é ligar os marcadores com as ações que ocorrem no jogo. Iremos fazer quatro coisas agora:

1. Adicionar 10 pontos ao placar sempre que uma nave inimiga for destruída;
2. Subtrair um de energia quando a nave do jogador colidir com uma nave inimiga;
3. Subtrair uma cidade quando uma nave inimiga alcançar o lado esquerdo da tela;
4. Destruir a nave do jogador quando a sua energia ou o número de cidades chegar a 0.

A alteração dos marcadores ocorrerá principalmente através de mensagens. As *mensagens* são a forma que os objetos de uma cena têm de trocar informações entre si. Vamos, então, criar os Messages Actuators que irão disparar as mensagens ao longo do jogo. Primeiro, selecione o tiro e adicione a seguinte lógica nele (lembre-se de adicionar apenas os LogicBricks que não estão minimizados):



Figura 5-44. Lógica de redução do número de cidades.

Este Message Actuator indica que uma nave foi destruída e que o placar deve ser alterado.

A mensagem que a nave inimiga manda para o placar das cidades já foi feita no tópico anterior, então prossigamos.

Agora, selecione a nave do jogador, insira um Message Actuator, modifique-o e conecte-o de acordo com a figura abaixo:

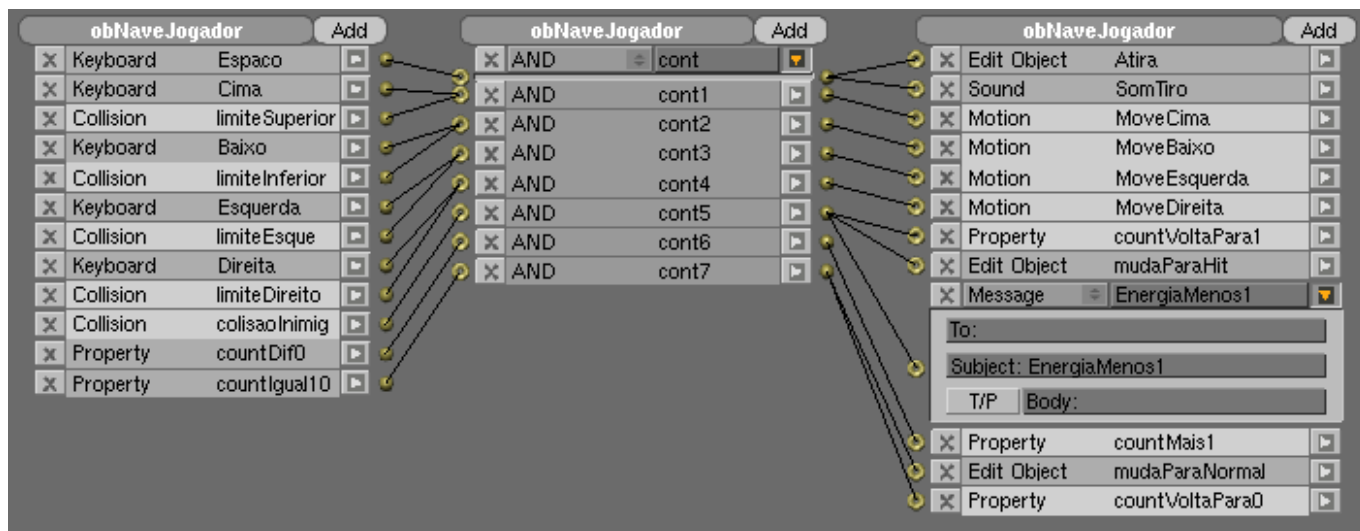


Figura 5-45. Mensagem que a nave do jogador envia quando perde energia.

Essa é a mensagem que indica que a nave perdeu energia.

Agora que as mensagens estão sendo devidamente disparadas, vamos configurar os receptores. Primeiro, selecione o marcador de energia (atenção, é o marcador numérico, e não o ícone!) e adicione a seguinte lógica:

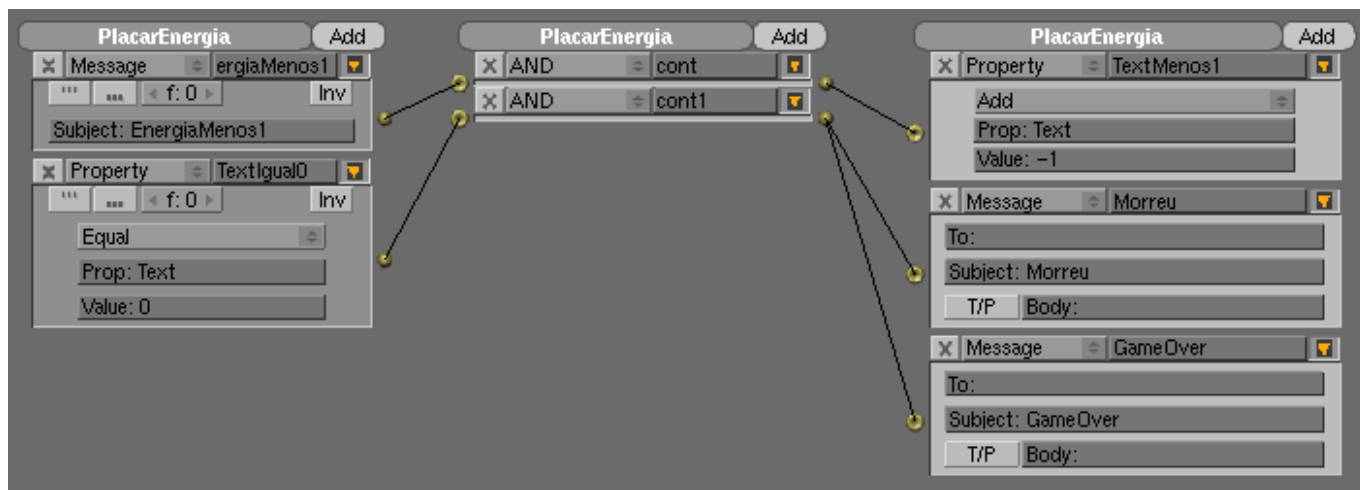


Figura 5-46. Lógica do placar de energia.

Assim que o marcador recebe uma mensagem, automaticamente ele subtrai 1 da quantidade de energia. Se a energia chegar a 0, o marcador dispara duas mensagens: *Morreu*, que será capturada pela nave do jogador, e *GameOver*, que será capturada pelo objeto *obGeraGameOver*, colocado bem no centro da cena, como mostra a figura abaixo:

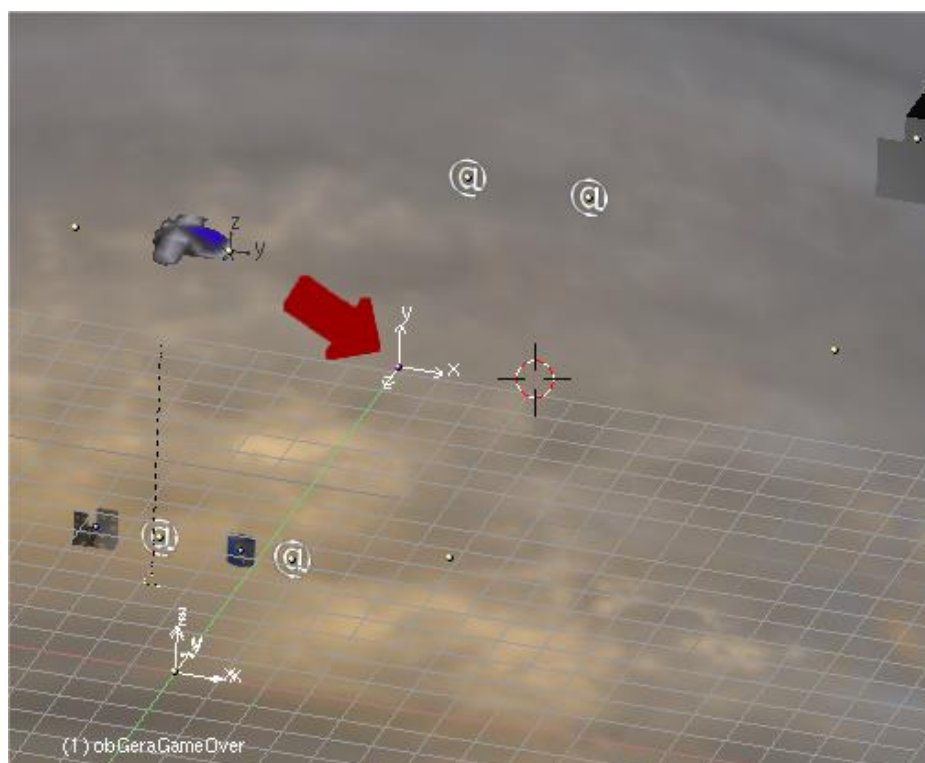


Figura 5-47. O objeto *obGeraGameOver*.

Iremos voltar nele daqui a pouco. Selecione a nave do jogador e insira a seguinte lógica:

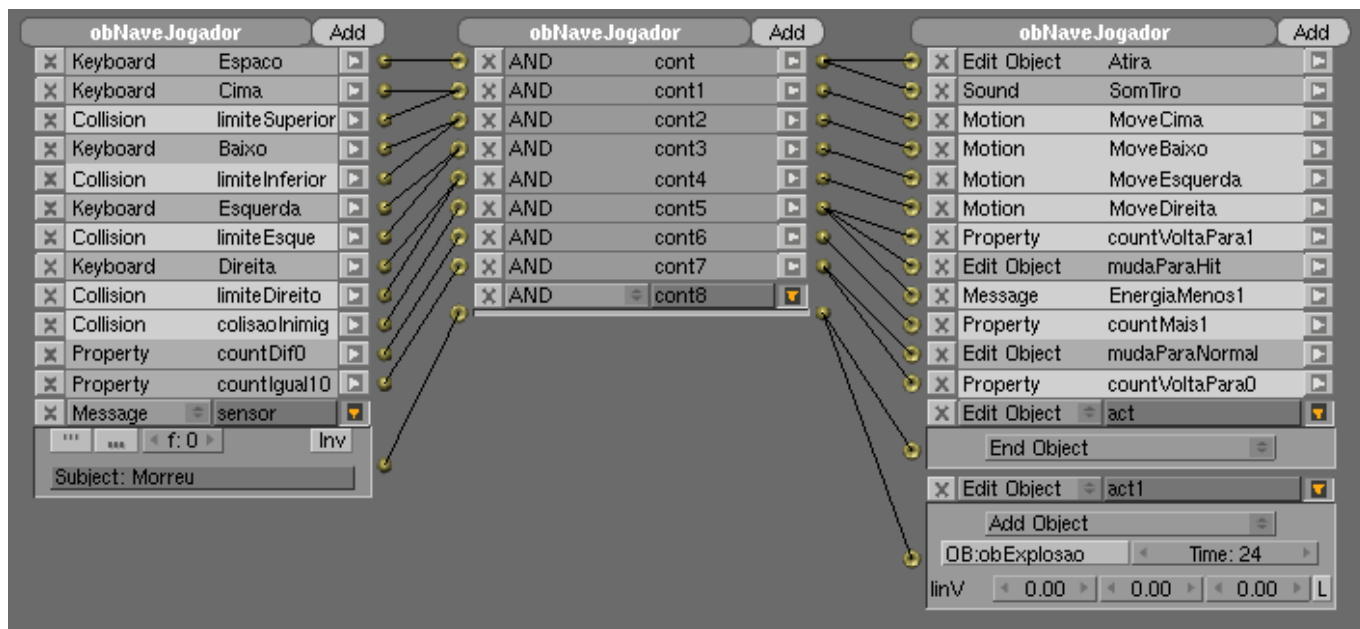


Figura 5-48. Ei, nave, é aqui que você explode...

Isso irá garantir que a nave saberá quando deve ser destruída dentro da cena.

O raciocínio usado para o marcador de energia é o mesmo para o marcador de cidades – selecione-o e faça como mostra a figura abaixo:

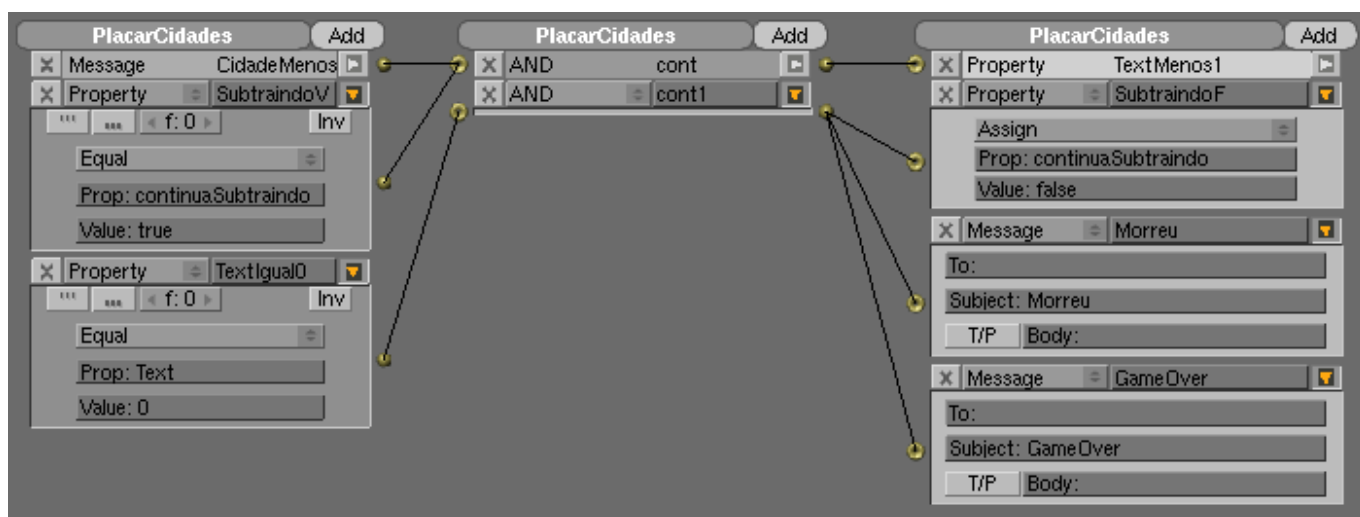


Figura 5-49. A lógica do marcador de cidades finalizada.

Como você deve ter reparado, existem alguns detalhes a mais aqui. Como as naves não param de ser disparadas, devemos evitar que o número de cidades fique negativo. Estude com atenção a lógica e você irá entender o que ela faz. Dica: `continuaSubtraindo` está sendo usado como chaveador da mensagem `CidadeMenos1`.

Por último, selecione o marcador do placar (o arroba no canto superior direito da tela) e insira o seguinte:

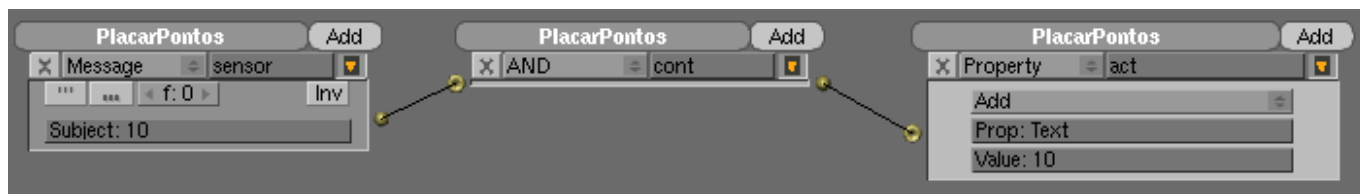


Figura 5-49. A lógica do marcador de cidades finalizada.

Agora o placar irá somar 10 ao seu valor sempre que uma nave for destruída – pronto para marcar o hi-score?

5.8. Game Over!

E, pra finalizar, vamos dar uma simpática mensagem de Game Over para o jogador quando ele perder todas as vidas ou todas as cidades forem destruídas.

obGeraGameOver é um objeto do tipo Empty (Vazio), ou seja, não possui nem pode adquirir uma malha – por isso ele não aparece na cena. Ele possui duas funções principais: guardar propriedades globais, ou seja, propriedades que serão usadas coletivamente em uma cena, e ajudar no posicionamento de objetos a serem adicionados na cena, que é o nosso caso.

Primeiro, selecione-o e insira uma propriedade do tipo Bool chamada GameOver; após isso, insira a seguinte lógica:



Figura 5-50. A lógica do marcador de cidades finalizada.

Qual é a necessidade de se ter uma propriedade chamada GameOver que verifica se o jogo já acabou? O problema é o seguinte: se o jogador morrer porque acabou a sua energia, as naves continuam se mexendo, acabando com as cidades e disparando um segundo Game Over, podendo vir a gerar resultados indesejados. Para evitar isso, obGeraGameOver muda a propriedade GameOver para true na primeira vez em que é ativado, evitando que a lógica aconteça uma segunda vez.

Pois bem, a função principal de obGeraGameOver é fazer aparecer na tela um outro objeto, obGameOver, que está na camada escondida (ALT-1) e você verá que ele é o plano onde está texturizado o texto “Game Over”, como mostra a figura abaixo:



Figura 5-51. obGameOver está selecionado e envolto em uma moldura branca.

Selecione-o, adicione uma propriedade do tipo Int chamada count e insira a seguinte lógica:

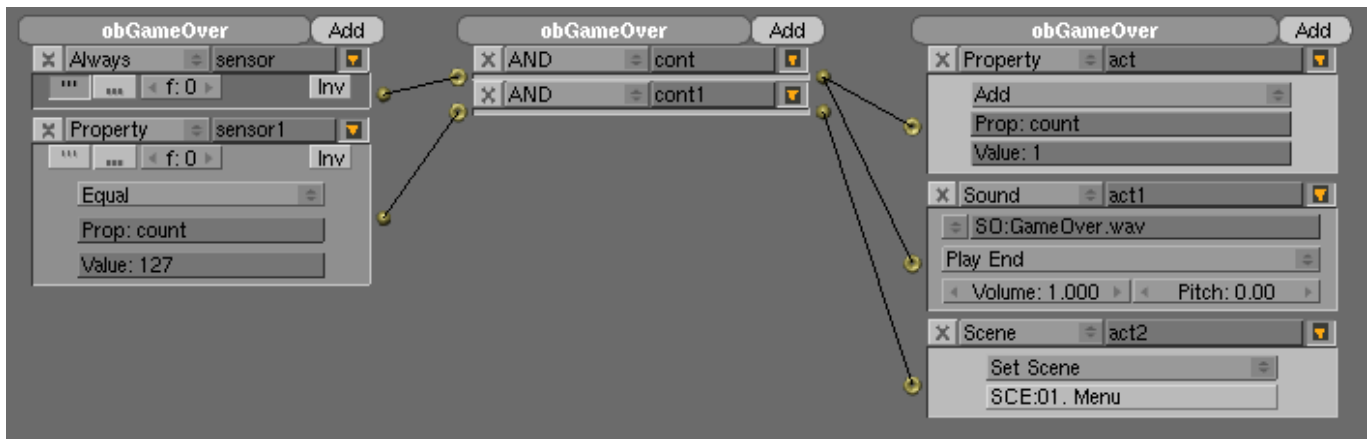


Figura 5-52. A lógica do objeto obGameOver.

É este objeto quem dá um basta no jogo. No entanto, ele possui um contador interno, que irá atrasar o término do jogo. Além disso, ele possui um Sound Actuator que... Bem, rode o jogo e verifique você mesmo. :-)

O Scene Actuator está configurado para o comando Set Scene, que irá alternar para a cena “01. Menu”.

... Ué, eu não disse que tinha outra cena não jogo? Que maldade minha... :P

5.9. E não é que temos um menu de inicialização?

Olha só que beleza – por essa você não esperava, hein? Já temos um menu configuradinho para o jogo! Claro que não preciso falar sobre ele – já tratamos do assunto em outras épocas, em outras paragens. Mas, se você quiser estudá-lo, vale a pena – algumas coisinhas estão diferentes... boa sorte!

O capítulo seguinte, entre outras coisas, irá mostrar o processo de geração de um auto-executável, que virá bem a calhar com esse projeto. Você só tem que lembrar de um detalhe: ao gerar um auto-executável, a cena que estiver atualmente selecionada será considerada a cena inicial do jogo. Portanto, antes de qualquer coisa, verifique se a cena “01. Menu” está aparecendo na tela, ok?

5.10. Ah, antes que eu me esqueça...

Então, tem uma pequena gambiarra que precisamos fazer... Se você tentasse gerar um auto-executável agora, iria perceber que o jogo não está jogável...

O problema é que o Blender demora um pouquinho para configurar todos os elementos da cena – mas, enquanto ele está configurando, o jogo não fica parado! E, quando você ganha controle da situação, as naves já estão quase relando o canto esquerdo da tela...

Vamos, então, remediar a situação. Selecione o emissor de naves, adicione uma propriedade do tipo Int chamada delay (atraso, em português) e insira a seguinte lógica nele:

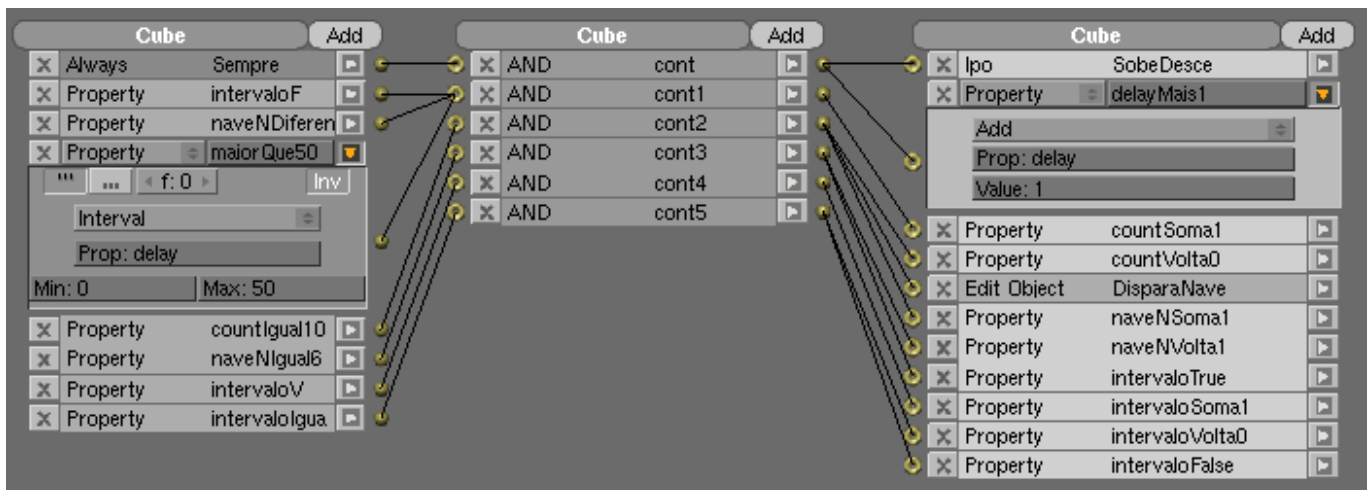


Figura 5-53. Atrasando a emissão de naves.

Conecte o novo Property Sensor ao mesmo Controller dos Property Sensors intervaloF e naveNDiferente6; já o Property Actuator deve ser ligado ao mesmo Controller em que está conectado o Ipo Actuator SobeDesce. Isso irá atrasar a emissão de naves por um tempo suficiente para que tudo esteja no lugar quando o jogo começar de fato.

Deixa eu explicar melhor o Property Sensor que acabamos de colocar. Normalmente ele funcionaria da seguinte maneira: enquanto a propriedade marcada estivesse entre os valores mínimo máximo, definidos na parte inferior do Sensor, ele enviaria uma mensagem de true adiante. No entanto, como o botão Inv está pressionado, a mensagem true será enviada quando a propriedade *não estiver entre os valores mínimo e máximo*. Isso é uma forma de verificar se a propriedade é maior que algum valor (ou menor, se for o caso).

6. Conclusão

Este é um momento sublime. Vale um ritual pagão inteirinho dedicado só a ele – afinal, você acabou de concluir o seu primeiro jogo, de cabo a rabo! Ok, ok, foi meio cansativo, foi meio maluco, provavelmente o seu estoque de neurônios caiu pela metade, mas fala a verdade – valeu a pena, não?

Se você achou que não valeu a pena, tudo bem – lembra do ritual pagão? Então, faça-o de qualquer jeito, só que não se esqueça de jogar a apostila no fogo – assim, você descarrega as vibrações do capítulo e, de quebra, a sua raiva! :-)

Capítulo 6

Um Exemplo de Aplicação Walkthrough

Nesse capítulo eu pretendo tocar no assunto da criação de aplicações walkthrough, ou seja, aquelas aplicações em que você caminha dentro de um ambiente virtual a partir de uma visão em primeira pessoa. Esse tipo de aplicação pode ser usado tanto para criar visualizações de arquitetura como pode ser usado de esqueleto para a criação de jogos de tiro, RPGs, etc.

Como a maior parte do trabalho sujo já está feito, este capítulo será curtinho. Aqui você vai aprender:

- Como inserir o *template* (modelo) da visão em primeira pessoa;
- Como manipular os controles disponibilizados pelo template;
- Como criar um arquivo auto-executável a partir de um arquivo .blend.

Simples, mas eficiente! Vamos começar?

1. Anexando o template na sua cena

Abra o arquivo Cap05-Walkthrough/CenaInicial.blend. A cena inicial é simples de tudo: apenas um cubo grande o suficiente para servir de ambiente (**Figura 6-1**).

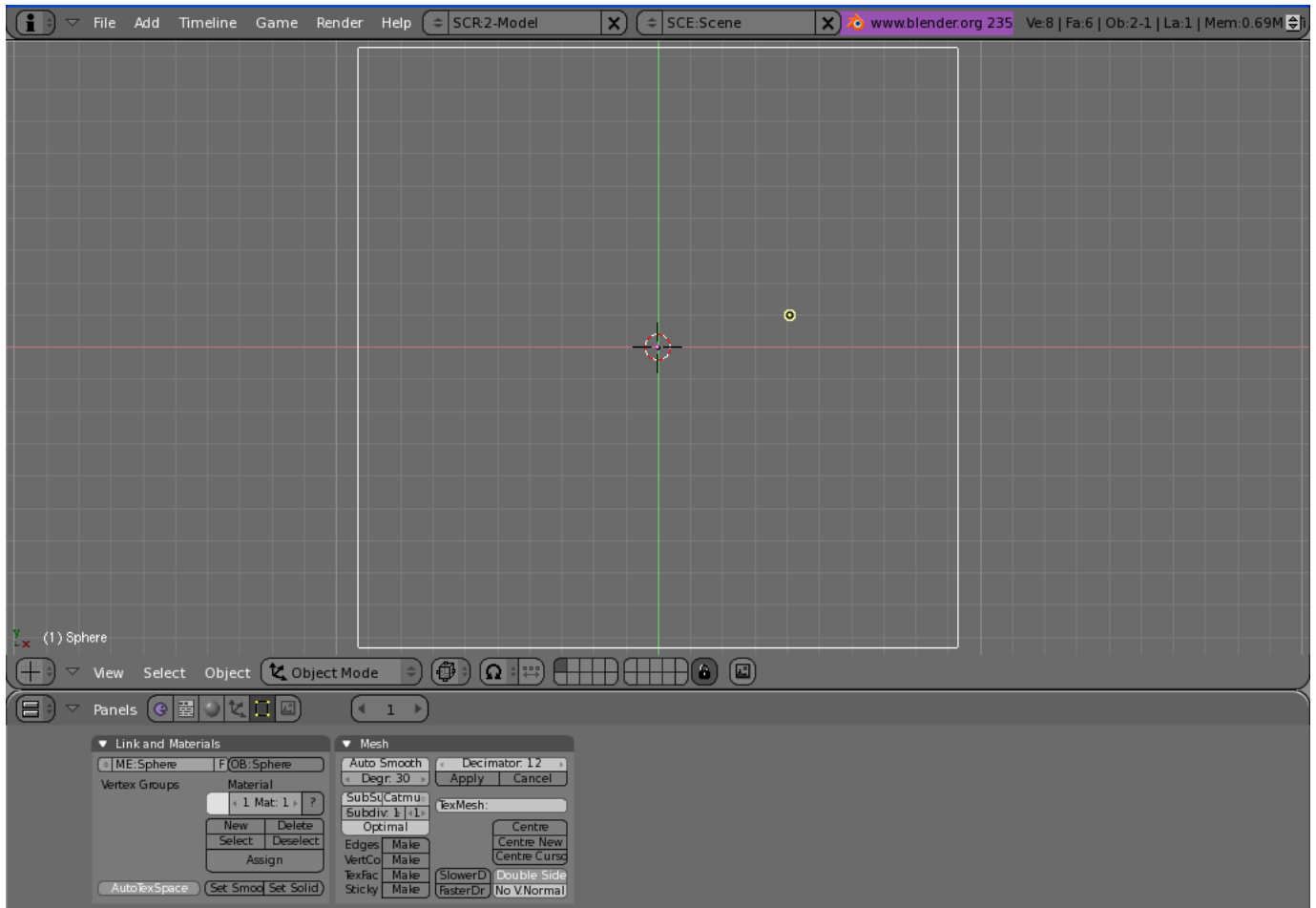


Figura 6-1. A cena inicial.

Pressione **SHIFT+F1** para abrir uma janela FileBrowser no modo Append (lembra disso? Fizemos a mesma coisa no capítulo 2, quando anexamos a abóbora de outro arquivo .blend). Localize o arquivo walkthrough_template.blend dentro da pasta Cap06-Walkthrough; selecione-o com **BEM**, abra a pasta Object, selecione com **BDM** os objetos camera0, shoulder e viewer e confirme a seleção pressionando o botão Load Library (**Figura 6-2**).

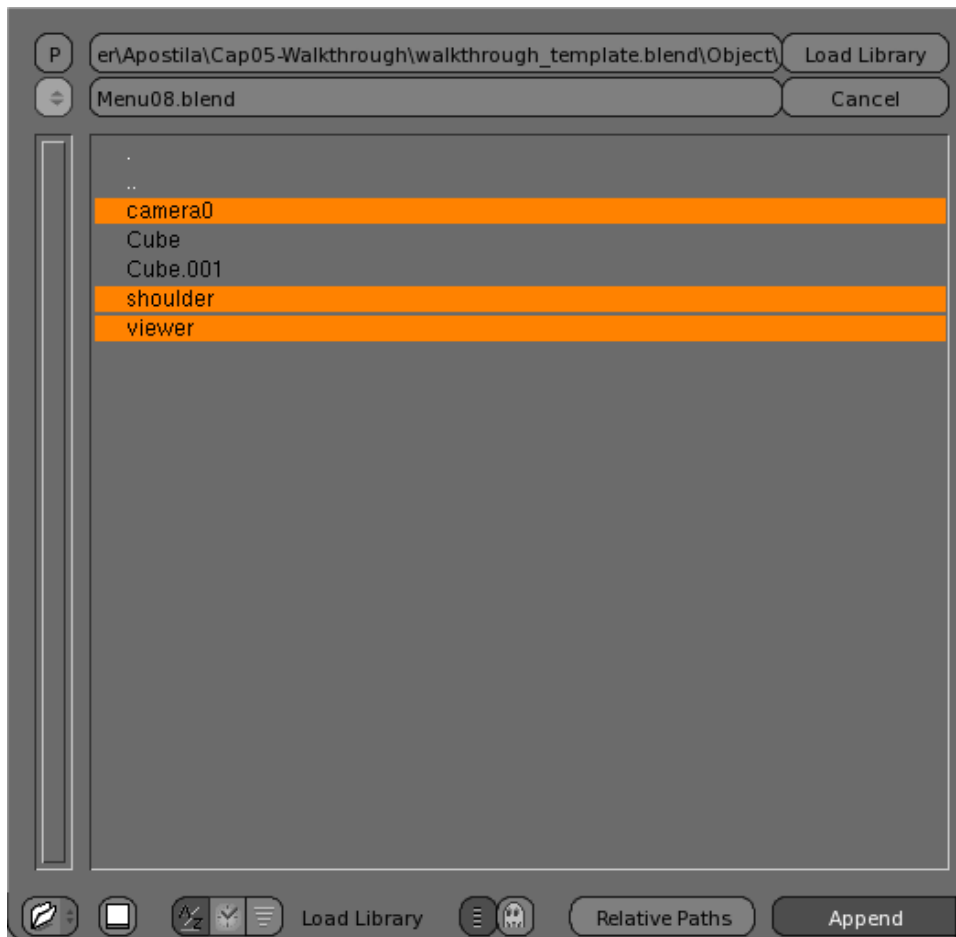


Figura 6-2. Os objetos devidamente selecionados na janela FileBrowser.

A sua cena deve estar como mostrado na figura abaixo:

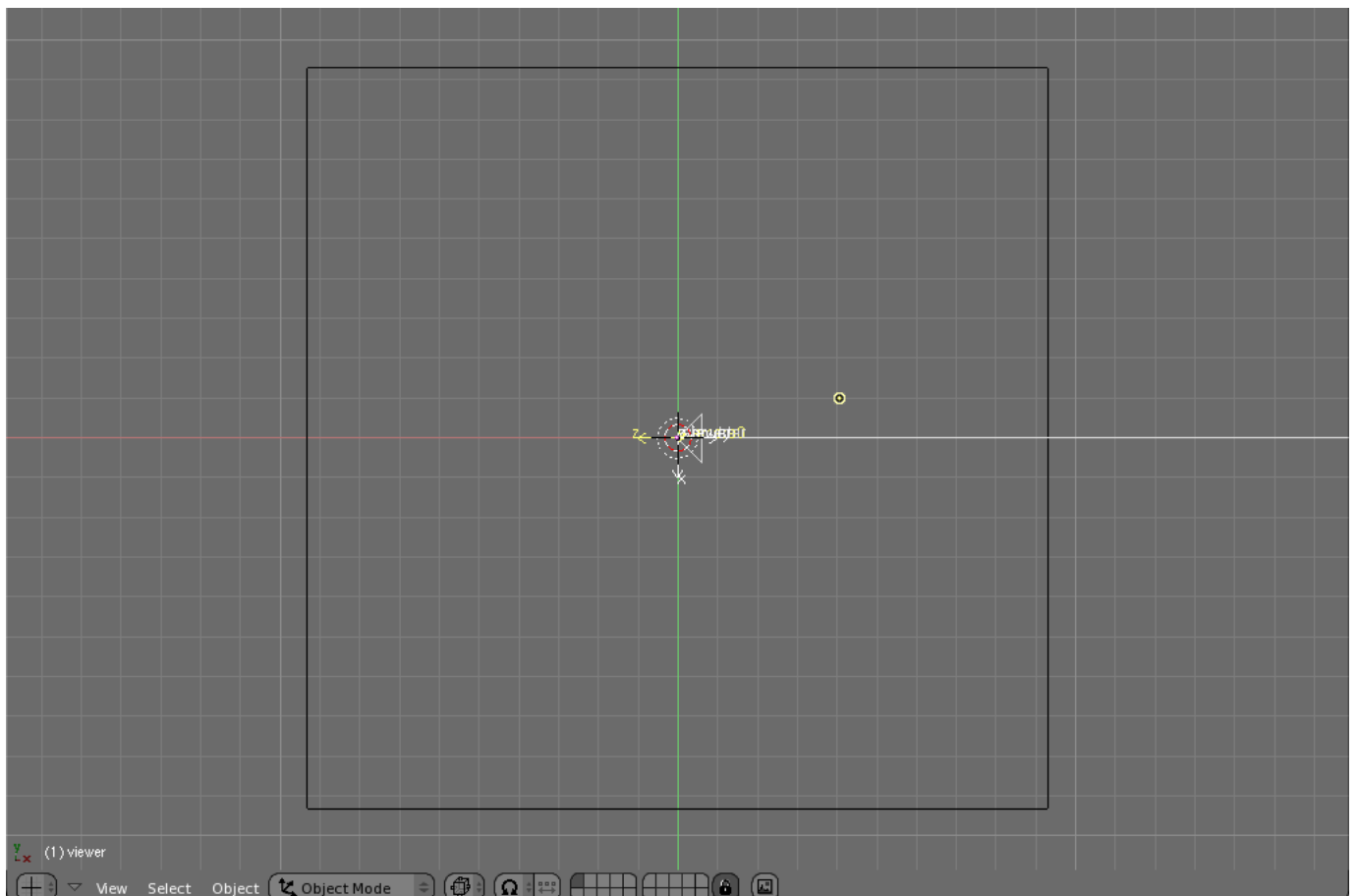


Figura 6-3. Eu, hein? Que bagunça é essa no meio da tela?

shoulder e viewer são os dois objetos que irão representar o personagem virtual; camera0 será a câmera utilizada para visualizar a cena (a **Figura 6-4** mostra mais de perto os três objetos). Selecione camera0 com **BDM** (se você não conseguir selecioná-la de primeira, continue pressionando **BDM** até o nome dela aparecer no canto inferior esquerdo da 3DView). Pressione **CTRL+0** para mudar a 3DView para a visão da câmera e pressione **PKEY** para iniciar a aplicação.

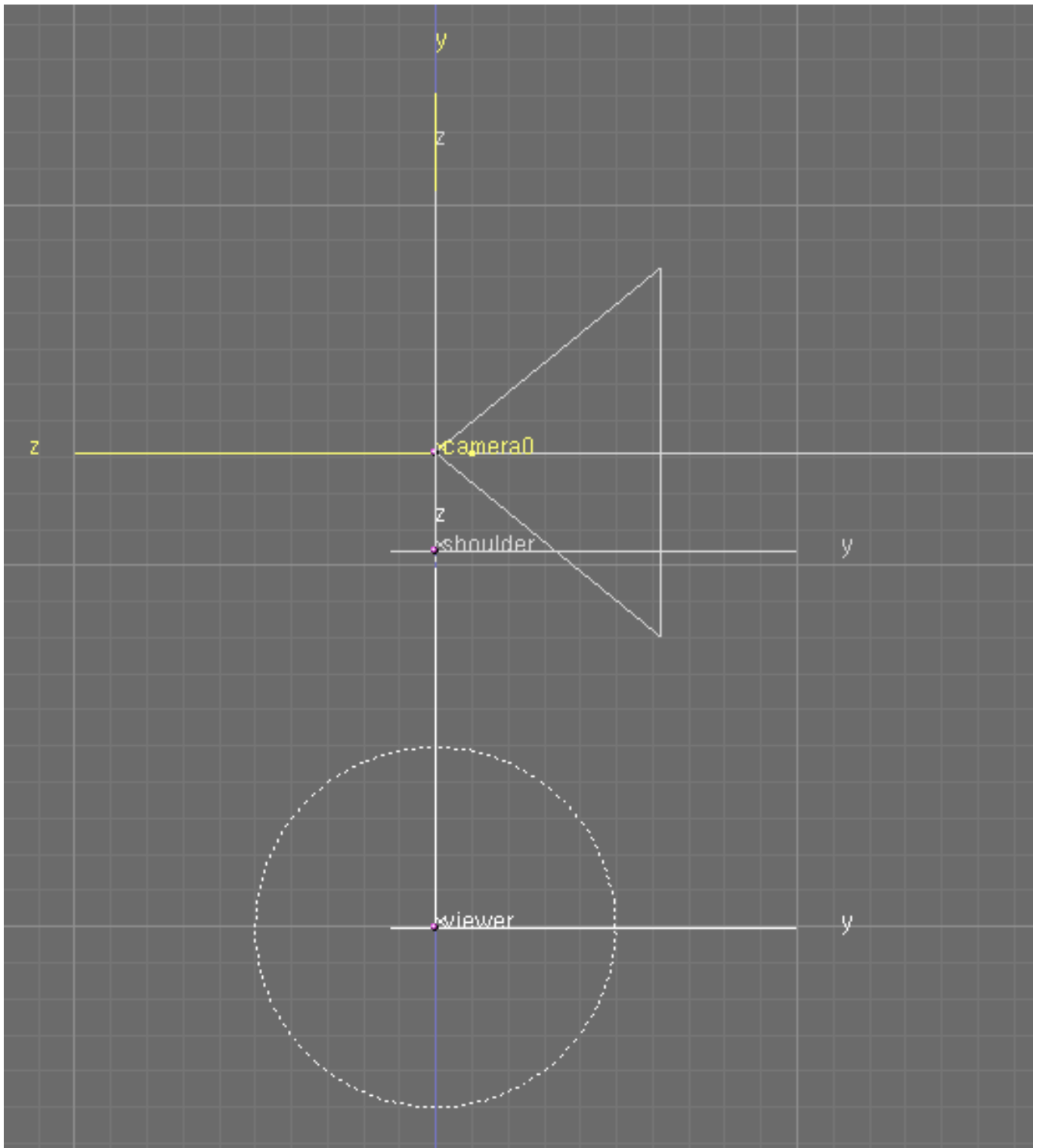


Figura 6-4. Toda a intimidade do nosso personagem.

A primeira coisa que irá acontecer é o personagem cair até colidir com o cubo – isso é normal, já que a gravidade está atuando sobre o modelo. Uma vez que ele tenha tocado o chão, você pode testar os seguintes comandos: **SETACIMA** e **SETABAIXO** o movimentam para frente e para trás, respectivamente; **SETAESQUERDA** e **SETADIREITA** o movimentam lateralmente para a esquerda e para a direita; segurar **BEM** e movimentar o mouse irá girar o seu ponto de vista. Se você pressionar **BARRAESPAÇO** o personagem irá entrar em *flymode*, ou seja, modo vôo. Nesse modo você pode se movimentar livremente no ambiente, anulando a força da gravidade; para voltar ao modo normal (*walkmode* ou modo caminhar), pressione **BARRAESPAÇO** novamente.

Talvez você queira trazer o personagem mais próximo do chão, de forma que ele não comece caindo toda vez que iniciarmos a aplicação. Pressione **ESC** para sair do modo de execução e mude a vista para uma visão frontal (**1PAD**). Selecione o objeto viewer com **BDM**, pressione **GKEY** para iniciar o modo de movimentação seguido de **ZKEY** para restringir o movimento ao eixo Z. Traga o objeto para próximo da base do cubo (mas não encoste, senão ele corre o risco de entalar!) e confirme com **BEM** (Figura 6-5).

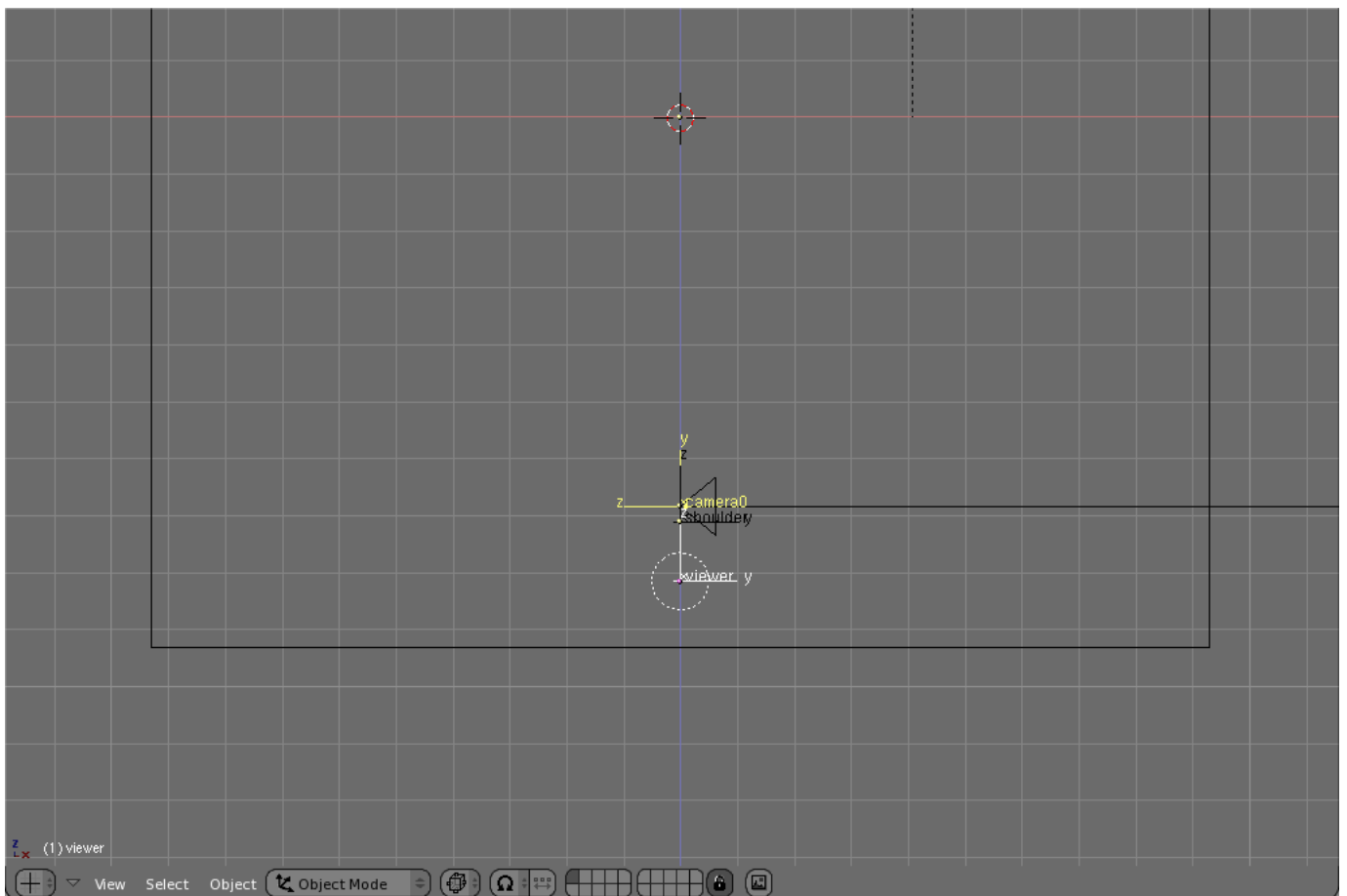


Figura 6-5. O personagem posicionado mais próximo da base do cubo.

Se você quiser arriscar dar uma espiada nos LogicBricks envolvidos na movimentação do personagem, mantenha o objeto viewer selecionado e pressione **F4** para ativar o contexto Logic. Não vou nem tentar começar uma explicação do que está acontecendo ali – o núcleo da lógica envolve muita programação em Python – e Python está completamente fora de escopo dessa apostila...

2. Importando uma cena de outro aplicativo

O Blender importa modelos 3D nos formatos VRML (*Virtual Reality Modeling Language* ou Linguagem de Modelagem de Realidade Virtual) ou DXF (*Drawing Exchange Format* ou Formato de Troca de Desenhos) – isto significa que o seu ambiente pode ser modelado, por exemplo, no AutoCad ou no 3D Studio MAX e importado diretamente na sua cena.

O próprio Blender é bem apropriado a esta finalidade, com as suas ferramentas de modelagem e seu solucionador de radiação. No entanto, modelar um ambiente também está fora do escopo deste capítulo.

Enquanto a importação dos formatos DXF e VRML 1.0 é suportada no Blender, o formato recomendado para importação é o VRML 2.0. Modelos DXF e VRML 1.0 importados não terão algumas informações importantes para serem apresentados de forma satisfatória, tais como informações de texturas – normalmente, antes de se mostrarem úteis, eles acabarão requerendo um pouco de processamento de sua parte.

Portanto, se possível, salve o modelo a ser importado no formato VRML 2.0.

Para importar uma cena, acesse o menu File>>Import e selecione o formato desejado.

3. Exportando um arquivo auto-executável

A exportação em si do arquivo auto-executável não é difícil; o que complica a situação é que você vai precisar de alguns arquivos extra para fazê-lo funcionar corretamente.

Acesse o menu File>>Save Runtime, dê um nome para o arquivo, selecione a pasta onde será salvo, confirme no botão Save Runtime e pronto! O seu auto-executável foi criado! Agora vamos à parte complicada: você vai precisar de mais dois arquivos para fazê-lo funcionar: python23.dll e sdl.dll – ambos estão na pasta de instalação do Blender. Copie-os para a pasta em que você criou o auto-executável e tudo deverá funcionar perfeitamente...

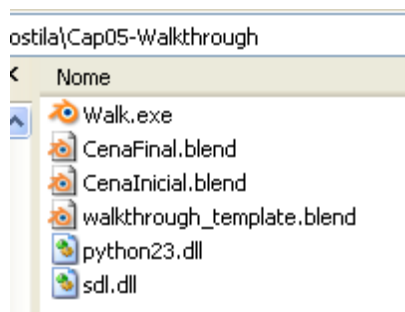


Figura 6-6. Walk.exe é o auto-executável que eu criei; repare na presença dos dois arquivos na mesma pasta.

4. Conclusão

Eu prometi que esse capítulo seria curto, não prometi? Pois bem, e com ele eu termino o nosso tour básico pelo mundo interativo do Blender. Claro que existem milhões de outras coisas que poderiam ser mostradas: modelagem, renderização, programação em Python, etc., mas a idéia era tentar manter as coisas o mais simples possível.

Espero que a jornada tenha sido de algum proveito para você, porque pra mim foi. E também espero que possamos nos reencontrar futuramente em alguma outra ocasião, seja em algum outro curso ou apenas para tomarmos um café (ou suco de laranja, se preferir, ou quem sabe uma vodka, ou um energético, ou água lavadeira... você escolhe :-)) e jogar conversa fora. De qualquer forma, ficam aqui os meus sinceros votos de sucesso no universo da computação gráfica.

Arigato Gozaimasu! E até a próxima!