7 TORNANDO OS ELEMENTOS DO MVI INTERATIVOS

Bárbara Gorziza Avila – IFRGS - bapadoin@gmail.com
Érico Amaral - UFP - ericohoffamaral@gmail.com
Fabiana Santiago Sgobbi - PPGIE - UFRGS - fabianasgobbi@gmail.com
Fabrício Herpich - PPGIE - UFRGS - fabricio_herpich@hotmail.com
Liane Margarida Rockenbach Tarouco - PPGIE - UFRGS - liane@penta.ufrgs.br
Leandro Rosniak Tibola - PPGIE - UFRGS- lrtibola@gmail.com

7.1 FERRAMENTAS DE AUTORIA

O desenvolvimento da programação dos códigos, os quais vão ser associados aos objetos colocados no MVI, é que vai permitir torná-los mais interativos. À medida que os usuários interagem (tocando ou meramente se aproximando de objetos), reagem exibindo os comportamentos previstos em seu *script*. Portanto o desenvolvimento desta programação é parte essencial da implantação de um MVI. Esta programação é feita usando a linguagem que foi desenvolvida pela Linden Lab, para o ambiente *Second Life* (com pequenas alterações) e, também, complementada pela linguagem de *script* criada pelos desenvolvedores do ambiente *OpenSimulator* e que foi denominada OSSL - *OpenSim Scripting Language*. Ambas são descritas na seção seguinte.

7.1.1 Linguagem para a construção dos *scripts*

Os *scripts* podem ser inseridos em qualquer objeto 3D criado no MVI. Nota-se que tanto o avatar quanto o NPC (*Non-Player Character*) são um tipo especial de integrantes dos MVI. Assim, os NPCs não recebem programação *script* diretamente, em sua estrutura, mas são criados e controlados por um objeto com este fim específico. Os avatares, por sua vez, não aceitam programação *script* diretamente em seu corpo digital e são criados pelo servidor de Mundos Virtuais, tal como o *OpenSim* e o *Second Life*. A customização do avatar e a criação de NPCs são tratadas em mais detalhes no Capítulo 2 “As ferramentas de autoria para criar um Mundo Virtual

Imersivo". Este Capítulo apresenta a programação *script* direcionada para a manipulação dos objetos primitivos (são as formas geométricas básicas, tal como o cubo e a esfera, que permitem a construção de objetos maiores e mais complexos, como móveis e prédios).

Desta forma, os Mundos Virtuais são ambientes dinâmicos, os quais dispõem de capacidade para a interação entre avatar e objetos e permitem a comunicação entre ambos. A dinâmica, a interação e a comunicação obtidas nos MVI são possíveis pela *Linden Scripting Language* ou LSL, linguagem originalmente desenvolvida, para ser usada nos componentes do *Second Life*, mas também suportada no *OpenSim*. Em virtude da concepção e das características particulares do *OpenSim*, foram desenvolvidas funções que não existiam na LSL original, assim, foi implementada a linguagem *OpenSimulator Scripting Language*, ou OSSL, objetivando, principalmente, a comunicação HTTP, interação com avatares, manipulação de NPCs, objetos, texturas, *notecards* e recursos do ambiente virtual.

Na LSL, a maioria dos *scripts* fica inativa até receber alguma entrada (como o recebimento de uma mensagem em um canal de comunicação) ou detectar alguma alteração no ambiente (o toque no objeto, a proximidade de um avatar). Sendo assim, o *script* está em algum estado e reagirá a eventos ou entradas de acordo com os objetivos definidos pelo programador. No entanto um *script* também pode conter dois ou mais estados diferentes e reagir de maneira diferente aos eventos ou entradas, dependendo do estado em que se encontra.

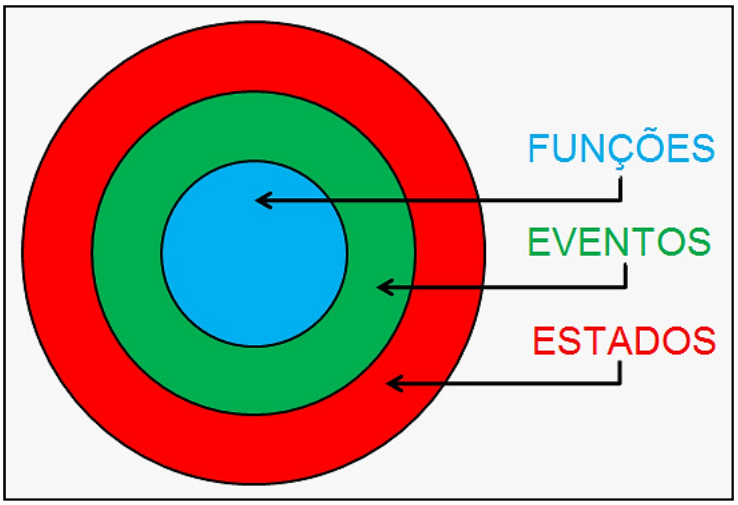
Um exemplo simples de mudança de estado é o de uma lâmpada que está no estado desligada e ignora todas as entradas, exceto a ação de ser tocada. Uma vez tocada, ela vai para o estado ligada, no qual ela aguarda a ação de ser tocada para voltar ao estado desligada.

Na LSL, um estado é uma seção especificada de código dentro da qual todos os eventos são especificados. O estado principal, exigido por todos os *scripts* LSL, é chamado de *default* (padrão). Todos os *scripts* devem ter um estado *default,* e cada estado deve ter pelo menos um evento.

Por sua vez, os eventos são formados por funções. As funções são comandos pré-definidos que permitem que o usuário combine estas funções com o seu código, para alterar os valores dos parâmetros e das variáveis do objeto que contém o próprio *script* ou de outro objeto presente no Mundo Virtual.

Assim, os *scripts* LSL são compostos de um ou mais estados e, dentro de cada estado, podem haver um ou vários eventos e cada evento é formado por uma ou muitas funções (*OPENSIM*, 2012; LINDEN RESEARCH, 2015; LSL, 2012). A estrutura hierárquica da programação *script* LSL é apresentada na Figura 7.1.

Figura 7.1 - Estrutura de codificação de um *script* LSL.



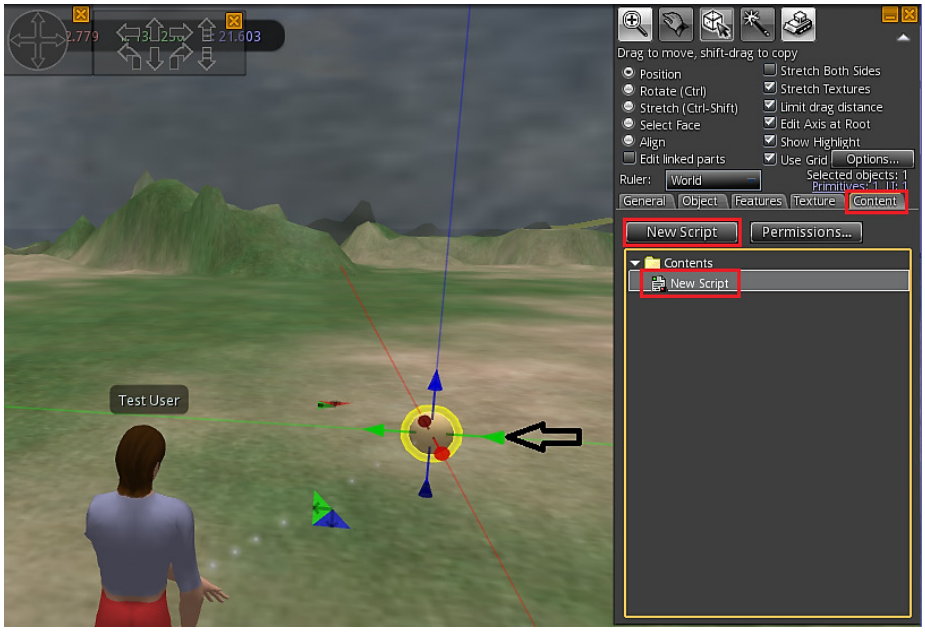
Fonte: Adaptado de GUÍA (2018).

Porém, antes de escrever qualquer trecho de um *script* LSL, é necessário criar um objeto no qual possamos inserir este código. Assim, para criar um objeto no Mundo Virtual *OpenSim*, podem ser seguidos os passos:

- clicar com o botão direito do mouse próximo ao terreno;
- no menu circular, clicar em Create;
- clicar com o botão direito do mouse no objeto criado;
- no menu circular, clicar em “Edit”, na aba “Content” e, por fim, em “New Script”.

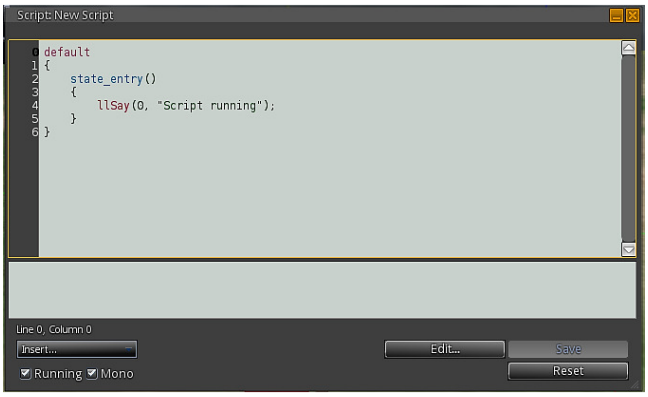
A Figura 7.2 mostra a imagem do resultado dos procedimentos acima.

Figura 7.2 - Criação de um novo *script* em um objeto.



Após estes procedimentos, é possível construir a programação LSL para aquele objeto. O *script* LSL pode ser ou não desenvolvido, no editor original do *OpenSim*, mas se forem usados editores externos ao Mundo Virtual, tal como o LSL Editor ou o Notepad++, o código deve ser copiado para o editor do próprio objeto no Mundo Virtual. O editor nativo do *OpenSim* é visto na Figura 7.3.

Figura 7.3 - editor de *script* LSL original do *OpenSim*.



7.1.1.1 Linden Scripting Language - LSL

A estrutura básica de um *script* LSL é composta pelo estado *default* e pelo evento state_entry(), como mostra o Código 1.

Código 1: Script LSL básico.

```
// o estado default é o estado padrão do LSL
default
{
// o "evento" state_entry é acionado em qualquer
// transição do estado default e na inicialização do objeto
    state_entry()
    {
            // a função llSay escreve "Alô, avatar!" no MV
            llSay(0, "Alô, avatar!");
    }
}
```

Um dos eventos mais comuns no LSL é o touch_start(), o qual é acionado quando o avatar "inicia" o click em um objeto. Este evento possui a estrutura:

- touch_start(integer total_number) { ; }

em que o total_number é o número detectado de avatares que tocam o objeto. O Código 2 mostra um exemplo do touch_start():

Código 2: Evento touch_start().

```
touch_start(integer total_number)
    {
            // executa ações
    }
```

Uma função usada para o envio de mensagens do objeto a outros objetos e/ou para o Mundo Virtual é a llSay(). Sua sintaxe é: llSay (*integer channel, string msg*), em que *channel* é um dos canais de comunicação privada, numerados de -2.147.483.648 até 2.147.483.647, enquanto as

mensagens direcionadas ao canal 0 (zero) são mostradas para todo o Mundo Virtual. O parâmetro msg é a mensagem enviada. Um exemplo desta função é apresentado no Código 3.

Código 3: Função llSay().

```
touch_start(integer total_number)
{
// mostra a mensagem "O objeto foi tocado!"
// no canal de comunicação zero
llSay(0, "O objeto foi tocado!");
}
```

Em ANEXOS, são descritos os comandos LSL necessários a uma programação básica, os quais podem ser utilizados e adaptados ao desenvolvimento de atividades educacionais em MVI.

7.1.2 *OpenSimulator Scripting Language* - OSSL

Antes de programar um *script* OSSL, é necessário verificar se estas funções estão habilitadas no *OpenSim*. É necessário editar o arquivo OpenSim.ini, situado na pasta em que o *OpenSim* foi instalado no seu computador, algo do tipo "C:\OpenSim-0.9.0.1\bin" e identificar se a opção *AllowOSFunctions* está definida como *true* e a função *OSFunctionThreatLevel* está assinalada como *High*. Alguns comandos OSSL necessitam de parâmetros específicos, para a opção *OSFunctionThreatLevel*, eles podem ser verificados na página oficial do *OpenSim* (2012).

A sintaxe do osNpcCreate é: key osNpcCreate (*string firstname, string lastname, vector position, string cloneFrom*).

Inserido em um objeto, o *script* com o comando osNpcCreate cria um NPC com o nome e sobrenome, em uma posição determinada, a partir de um modelo de aparência. O Código 4 apresenta a criação de um NPC. Este código usa variáveis que são descritas no ANEXO 1.

Código 4: Criação, movimentação e remoção do NPC com OSSL.

```
key npc;
vector toucherPos;

default
{

   state_entry()
   {
```

```
    llSay(0, "Clique neste objeto para criar, mover e apagar um NPC!");
  }

  touch_start(integer number)
  {
// recupera a posição do avatar que tocou o objeto
// e incrementa 1 metro na coordenada X para esta posição
    vector npcPos = llGetPos() + <1,0,0>;
    // captura a aparência do avatar e a guarda em "forma-roupas"
    osAgentSaveAppearance(llDetectedKey(0), "forma-roupas");
// cria o NPC chamado "NPC Clone", na posição registrada em npcPos
// e com a aparência armazenada em "forma-roupas"
    npc = osNpcCreate("NPC", "Clone", npcPos, "forma-roupas");
    // guarda a posição do avatar que tocou no objeto
    toucherPos = llDetectedPos(0);
    // transfere a execução do script para o estado existeNPC
    state existeNPC;
  }
} // fim default

state existeNPC
{
    state_entry()
    {
            // move o NPC a 3 metros do avatar na coordenada X
            osNpcMoveTo(npc, toucherPos + <3,0,0>);
            // o NPC emite uma mensagem para todo o Mundo Virtual
            osNpcSay(npc, "Olá!!! Meu nome é " + llKey2Name(npc));
    }

    touch_start(integer number)
    {
            // o NPC emite uma mensagem para todo o Mundo Virtual
            osNpcSay(npc, "Tchau!!");
            // remove o NPC
            osNpcRemove(npc);
            // torna a variável do NPC nula
            npc = NULL_KEY;
            // retorna para o estado default
            state default;
    }
} // fim do state exiteNPC
```

7.2 Ferramentas de autoria para criar o *script*

Embora a criação de *scripts* possa ser realizada usando apenas um editor de textos básico, tal como o bloco de Notas, recursos adicionais de apoio à formatação, tais como os existentes no editor LSL Editor, podem facilitar o trabalho do desenvolvedor.

O LSL *Editor Community Edition* para *Windows* é um editor, compilador e depurador de *scripts* LSL autônomo. Seu compilador e depurador já estão razoavelmente precisos.

O desenvolvimento do LSL Editor foi feito originalmente por Alphons van der Heijden, autor de o LSL Editor. Detalhe: executável e fonte podem ser encontrados em http://sourceforge.net/projects/lsleditor/

> O LSL Editor, *software* que ajuda os usuários a escrever *scripts* mais facilmente para objetos do *Second Life* e do *OpenSim*, foi atualizado com mais suporte para as funções do *OpenSim*. Com a nova versão 2.56, os usuários também podem criar *scripts* que executam outros *scripts*, o desenvolvedor do LSL Editor, Frank Rulof, disse à *Hypergrid Business*, e a funcionalidade de teste e depuração é muito melhor que a do editor LSL integrado nos visualizadores do *OpenSim* (KARIUKI, 2018).

A nova versão 2.56.0 tem mais funcionalidades de teste e depuração. Na verdade, podem-se testar seus *scripts* ou conjuntos de *scripts* e sua interação, em uma extensão muito grande fora da grade (servidores de MVI em grid); o suporte de depuração, também, é muito melhor que o editor de *scripts in-world*.

As grades do *OpenSim* podem executar os comandos LSL padrão, mas também possuem funcionalidade adicional por meio da *OpenSimulator Scripting Language* (OSSL). Por exemplo, comandos específicos do *OpenSim* podem forçar o teletransporte, o que é útil para construir portões de teleporte ou armas que imediatamente mandam alguém para um inferno virtual quando são atingidos. Também há vários comandos para criar e gerenciar caracteres não executáveis (NPCs) e para colocar texto e gráficos em um prim. As versões anteriores do LSL Editor suportavam apenas funções LSL, o que significa que não se poderia usar o editor com as funções OSSL, então, agora os *scripts* com funções OSSL irão compilar e podem ser testados e depurados.

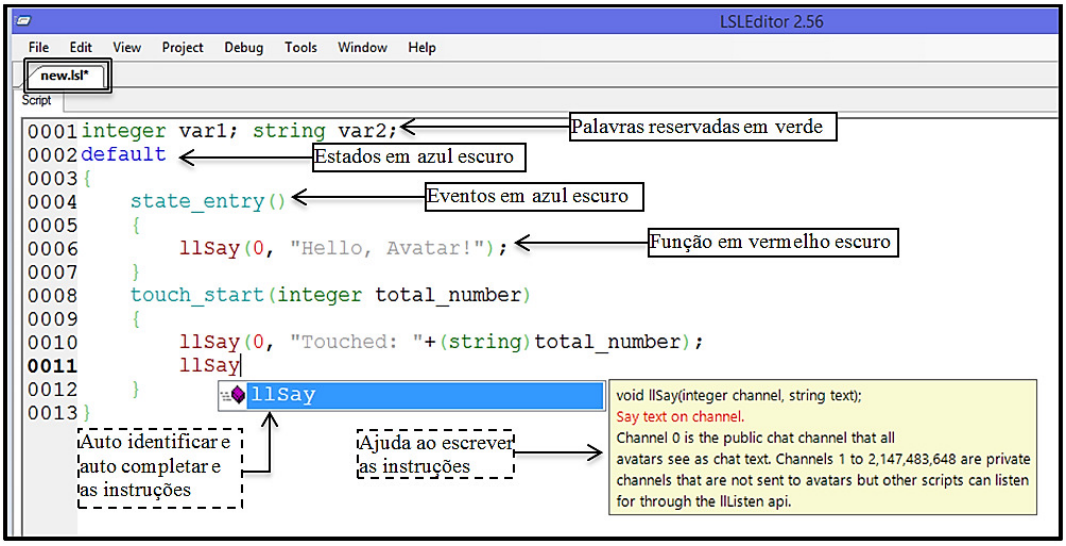
O LSL Editor atualizado, também, permite que os usuários adicionem novos objetos e itens e executem um verificador de sintaxe, bem como personalizar fontes e cores. Ele ainda permite realçar, localizar e substituir funções, saltar para um número de linha específico, bem como formatar somente a área selecionada ou o documento inteiro.

Ele funciona para plataformas *Windows*, incluindo o *Windows* 2K, 2003, XP, Vista, 7 e 8 e está disponível para *download,* na página *Source Forge* e o manual do LSL Editor está disponível na página do *Google Doc* (KARIUKI, 2018).

Dentre as facilidades apresentadas pelo LSL Editor, está a identificação do código *script,* por meio de um conjunto de cores, que identificam as palavras reservadas em verde (os

declaradores de variáveis integer e *string*), estados em azul-escuro (*default*), eventos em azul-claro (touch_start()) e as funções em vermelho-escuro (llSay()). A Figura 7.4 apresenta este esquema de cores, para o código LSL, nas caixa com borda simples.

Figura 7.4 - Tela do editor LSL Editor.



Esta disposição de cores é útil, quando se digita o código ou se procura por um erro de escrita, pois, como a LSL é sensível às letras maiúsculas e minúsculas, a função llSay() é válida, enquanto a função llsay() é inválida e gera um erro durante a compilação.

Outros recursos que auxiliam os iniciantes na programação LSL é o autoidentificar e o autocompletar o código. Assim, enquanto o programador digita as instruções, o LSL Editor verifica a sintaxe e sugere os estados, eventos e funções que se aproximam do texto digitado. Ainda, ao digitar a instrução LSL, o usuário pode verificar a sua sintaxe e descrição, na forma de um pequeno quadrado de ajuda. A Figura 7.4 mostra estes recursos nas caixa com a borda pontilhada.

Uma observação pertinente a ser feita é sobre o salvamento do *script*, já que, por padrão, o LSL Editor cria um novo arquivo com o nome de “new.lsl” e, se o usuário não alterar o nome do seu programa, ele será salvo com o mesmo nome, podendo sobrescrever programas escritos anteriormente. Então, é aconselhável que, ao escrever um *script*, ele seja salvo com um nome que indica sua função e seja fácil de lembrar em um momento posterior. Outro detalhe que pode colaborar, para manter os códigos sempre salvos, é observar o asterisco (*), ao lado do nome do arquivo; ele indica que o *script* foi modificado e ainda não foi salvo. Para manter os *scripts* sempre atualizados e evitar perda de códigos, deve-se clicar em “File - Save as”, digitar o nome e escolher

uma pasta para guardar o arquivo. O nome do arquivo-padrão “new.lsl” e o asterisco indicador de alteração podem ser visualizados, no canto superior direito da Figura 7.4, na caixa com a borda dupla.

Adicionalmente, ferramentas que permitam especificar as funções a serem implementadas de um modo mais visual, usando a estratégia de blocos visuais de programação, proposta pelo MIT para o ambiente de autoria Scratch, por exemplo, podem facilitar bastante a criação de *scripts*.

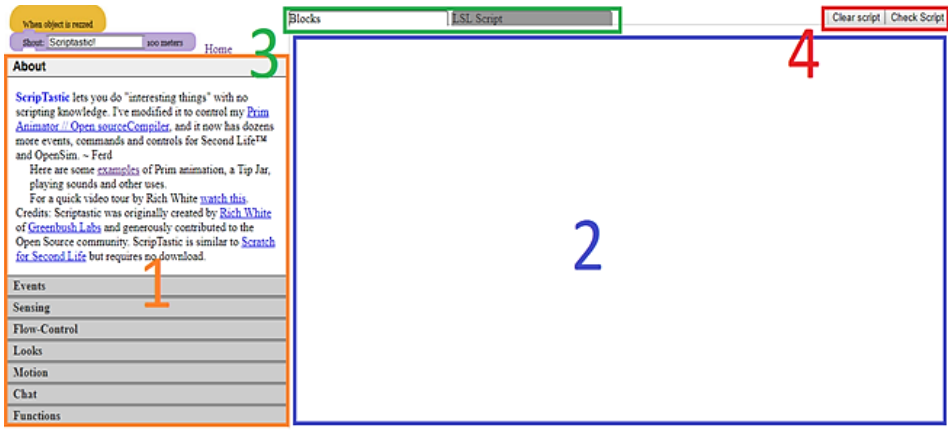
7.2.1 *ScripTastic* (versão light e simplificada para gerar LSL)

O *ScripTastic* é uma ferramenta básica de conversão de blocos visuais para LSL. O *ScripTastic* consiste em uma versão *light* e simplificada, possuindo menos funcionalidades, em relação às outras ferramentas existentes, mas traz uma interface mais simples e intuitiva ao usuário que está se iniciando no desenvolvimento de *scripts* para Mundos Virtuais. Para acessar a interface *web* do *ScripTastic*, o usuário pode abrir em seu navegador o endereço: https://www.outworldz.com/scriptastic/.

7.2.1.1 Interface básica

A interface básica, conforme a numeração e a coloração mostradas na Figura 7.5, os elementos básicos da interface do *ScripTastic* são:

Figura 7.5 - Interface inicial do *ScripTastic*.



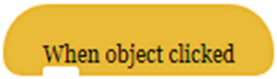
- À exceção da primeira aba, "*About*", que contém uma breve descrição do *ScripTastic*, as demais dividem os blocos existentes em categorias, para sua melhor organização, de acordo com sua funcionalidade.
- É a área de trabalho, para onde serão arrastados os blocos (no caso da visão de blocos, vide item 3) e onde será exibido o *script* LSL resultante (visão LSL).
- A primeira aba "*Blocks*" corresponde à visão de blocos e a segunda, "LSL *Script*", à visão LSL, que conterá o código do *script* gerado a partir dos blocos.
- O botão "*Clear Script*" limpa a área de trabalho, apagando tanto os blocos quanto o *script*. Já o botão "*Check Script*" varre o *script* em busca de erros e exibe os erros encontrados, na própria área de trabalho, com a visão "LSL *Script*" selecionada.

7.2.1.2 Blocos de programação

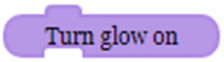
Os blocos, ao serem arrastados para área de trabalho, a partir do menu lateral esquerdo, imediatamente começam a fazer parte do programa, alterando o *script* resultante, conforme a Figura 7.6.

Figura 7.6 - Arrastando bloco para a área de trabalho.

Os blocos se encaixam formando estruturas mais complexas e são executados de forma sequencial, de cima para baixo.



Exemplo de bloco que pode iniciar um *script*. Nota-se a falta de encaixes na parte superior e a presença do encaixe na borda inferior.



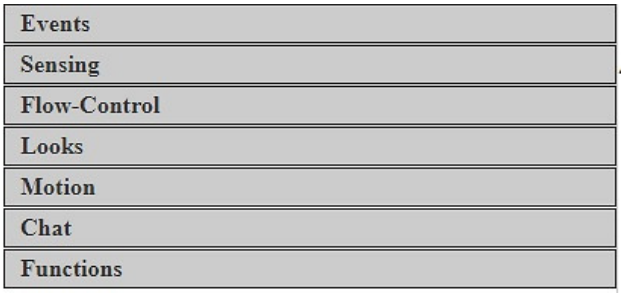
Exemplo de bloco intermediário, que deve ser encaixado em algum outro bloco superior.

Exemplo de bloco que só pode ser encaixado no final de um *script*.

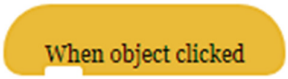
7.2.1.3 Categorias dos blocos

Todos os blocos disponíveis no *ScripTastic* estão presentes, em alguma das categorias visíveis na Figura 7.7, devidamente distribuídos, de acordo com sua funcionalidade. Os blocos de cada categoria também possuem uma coloração específica.

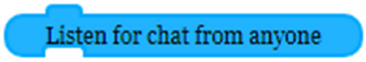
Figura 7.7 - Categorias dos blocos.



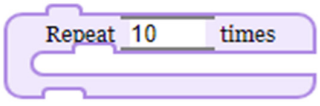
EVENTS: Blocos que definem comportamento, quando algum evento for disparado.

Exemplo: quando objeto for clicado. 

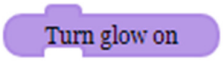
SENSING: Blocos relacionados a sensores do ambiente e do *chat*.

Exemplo: escutar mensagens no *chat*. 

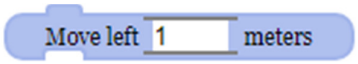
FLOW-CONTROL: Blocos com as funções básicas de controle de fluxo: condicionais e repetidores.

Exemplo: repetir ações 10 vezes. 

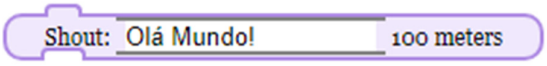
LOOKS: Blocos relacionados à aparência do objeto.

Exemplo: ligar o brilho. 

MOTION: Blocos com funcionalidade de movimentação.

Exemplo: mover 1 metro para a esquerda. 

CHAT: Blocos que enviam mensagens no *chat* do ambiente virtual ou para outros *scripts*.

Exemplo: mandar mensagem "Olá, Mundo!" para todos em um raio de 100 metros. 

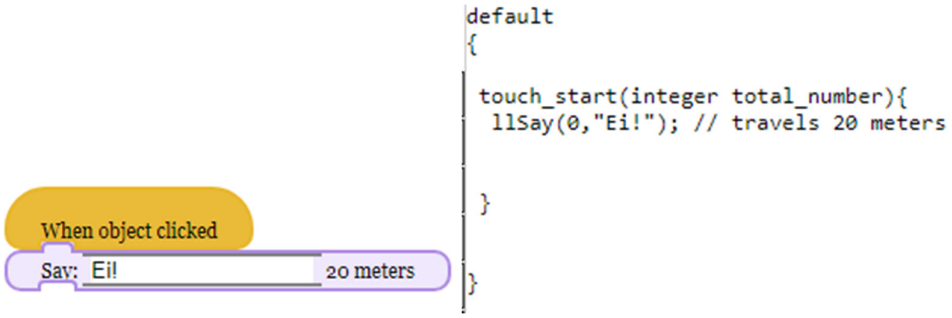
FUNCTIONS: Blocos com funções relacionadas não tão diretamente ao Mundo Virtual, mas que são essenciais dentro da programação dos *scripts*.

Exemplo: criação de uma variável inteira com valor inicial de 1. 

7.2.1.4 Exemplos de aplicações

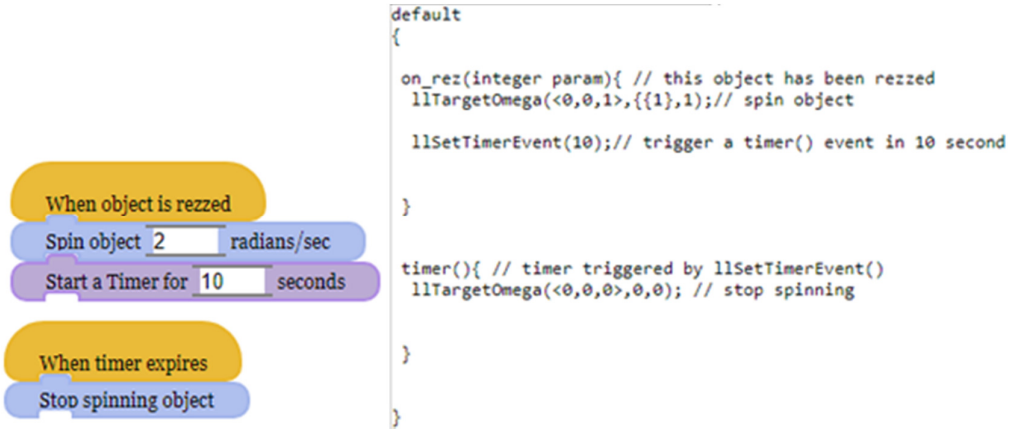
A aplicação abaixo (FIGURA 7.8), representada à esquerda pelo seu formato em blocos e na direita pelo código LSL gerado, usando o *ScripTastic*, define que, quando o objeto ligado a esse *script* for clicado, enviará a mensagem "Ei!", no *chat*, para todos num raio de 20 metros.

Figura 7.8 - Codificando um *script* em blocos visuais e sua saída na linguagem LSL.



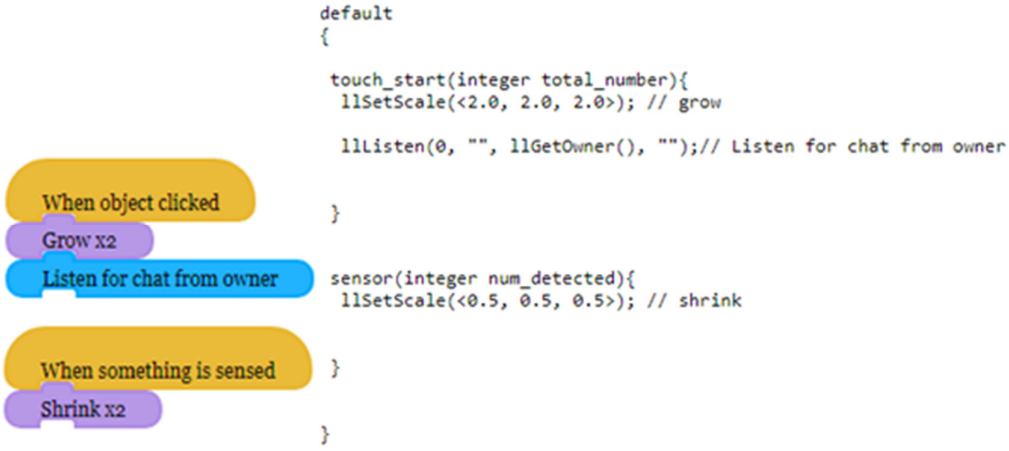
Já o próximo exemplo (FIGURA 7.9) faz com que o objeto comece a girar, assim que for criado a uma velocidade de 2 rad/seg. Após passados 10 segundos, o objeto para de girar.

Figura 7.9 - Demonstração de *scripts* desenvolvidos com blocos visuais.



Neste último exemplo (FIGURA 7.10), o objeto dobra de tamanho sempre que for clicado e é reduzido pela metade quando detecta alguma mensagem do seu dono no *chat*.

Figura 7.10 - Exemplos de aplicação dos blocos visuais para desenvolvimento de MVI.



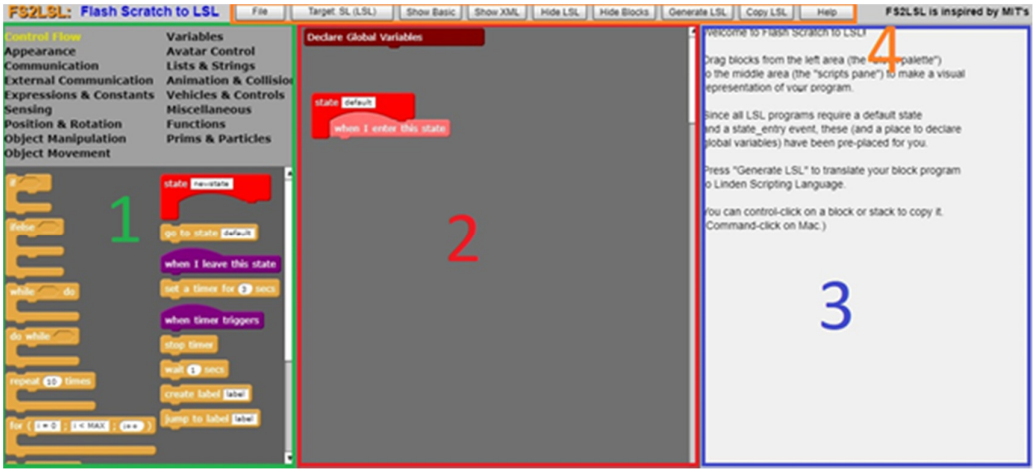
7.2.2 Ferramenta FS2LSL (Universidade de Denver)

Disponível em: http://inworks.ucdenver.edu/jkb/fs2lsl/

O FS2LSL é inspirado diretamente na aplicação de programação em blocos do MIT, o *Scratch*, utilizando blocos com *design* semelhante. Para usá-lo, é necessário o usar o *plugin Adobe Flash*.

7.2.2.1 Interface básica

Figura 7.11 - Interface inicial do FS2LSL.



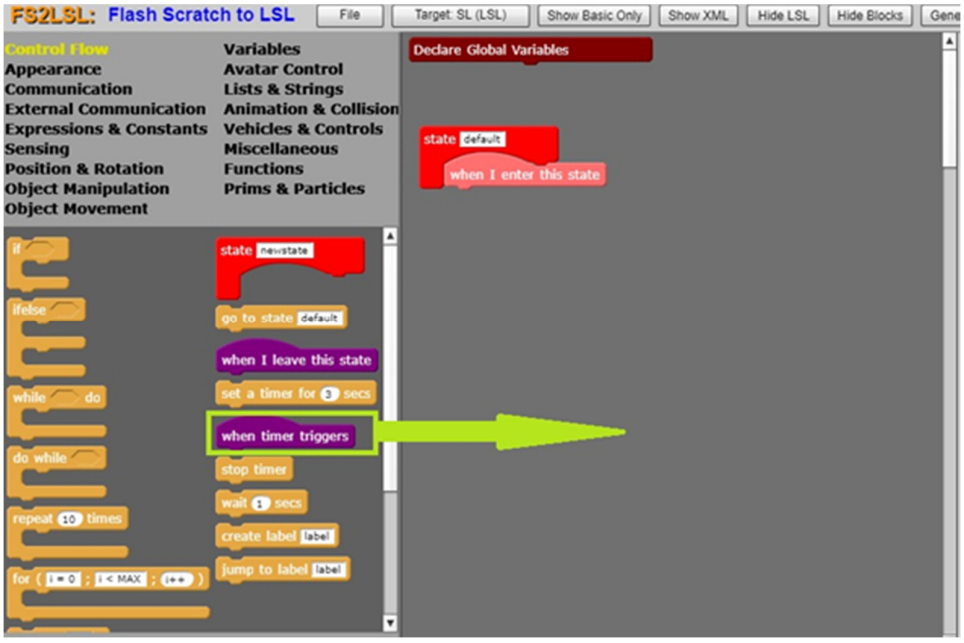
Conforme a numeração e coloração vistas na Figura 7.11, os elementos básicos da interface do FS2LSL são:

1. Menu com os todos os blocos disponíveis à aplicação, divididos em categorias.
2. Área de trabalho para onde os blocos são arrastados e onde eles são subsequentemente manipulados. Alguns blocos são criados por padrão em um projeto novo.
3. Aqui é exibido o *script* resultante dos blocos presentes na área de trabalho.
4. Barra com diversos botões e comandos:
    1. FILE abriga as funções de controle de projeto: criar, salvar, carregar.
    2. TARGET permite a troca entre ambiente *Second Life* e *OpenSim*.
    3. SHOW(...) alterna entre BASIC ONLY(só serão mostrados blocos simples) e ADVANCED(todos os blocos ficam disponíveis).
    4. SHOW/HIDE XML alterna a exibição do arquivo XML gerado, além do LSL.
    5. SHOW/HIDE LSL alterna a exibição do *script* LSL na interface.
    6. SHOW/HIDE BLOCKS alterna a exibição do área de trabalho de blocos.
    7. GENERATE LSL gera o código LSL, a partir dos blocos presentes, no momento na área de trabalho e o exibe na tela.
    8. COPY LSL copia o *script* LSL na área de transferência.
    9. HELP exibe algumas informações de ajuda sobre o FS2LSL.

7.2.2.2 Blocos de programação

Os blocos a serem usados são arrastados do menu lateral esquerdo para a área de trabalho central. O código LSL é gerado na área à direita, se habilitada, quando o botão “Generate LSL” é pressionado (FIGURA 7.12).

Figura 7.12 - Inserção dos blocos.



Os blocos se encaixam formando estruturas mais complexas e são executados de forma sequencial, de cima para baixo.

Exemplo de bloco que pode iniciar um *script*. Nota-se a falta de encaixes na parte superior e a presença do encaixe na borda inferior.

Exemplo de bloco intermediário, que deve ser encaixado em algum outro bloco superior.

Exemplo de bloco que recebe algum um parâmetro booleano.

Exemplo de expressão booleana.

7.2.2.3 Categorias dos blocos

Todos os blocos disponíveis no FS2LSL estão presentes, em alguma das categorias visíveis na Figura 7.13, devidamente distribuídos, de acordo com sua funcionalidade. Os blocos também possuem uma coloração diferenciada, dependendo de sua função.

Figura 7.13 – Categorias.

CONTROL FLOW: Blocos com as funções básicas de controle de fluxo, como condicionais e repetidores.

APPEARANCE: Blocos relacionados à aparência do objeto.

COMMUNICATION: Blocos que tratam de comunicação entre objetos e outros habitantes.

EXTERNAL COMMUNICATION: Comunicação com fatores externos.

EXPRESSIONS & CONSTANTS: Expressões e constantes matemáticas e booleanas.

SENSING: Blocos relacionados a sensores do ambiente e do *chat*.

POSITION & ROTATION: Posição e rotação de objetos.

OBJECT MANIPULATION: Blocos que modificam características de objetos.

OBJECT MANIPULATION: Blocos que movimentam objetos.

VARIABLES: Criação e manipulação de variáveis a serem utilizadas no *script*.

AVATAR CONTROL: Blocos relacionados ao controle de Avatares do Mundo Virtual.

LISTS & STRINGS: Criação e manipulação das estruturas de lista e string.

ANIMATION & COLLISION: Animação e colisão.

VEHICLE CONTROLS: Blocos usados ao controle de veículos.

MISCELLANEOUS: Blocos variados.

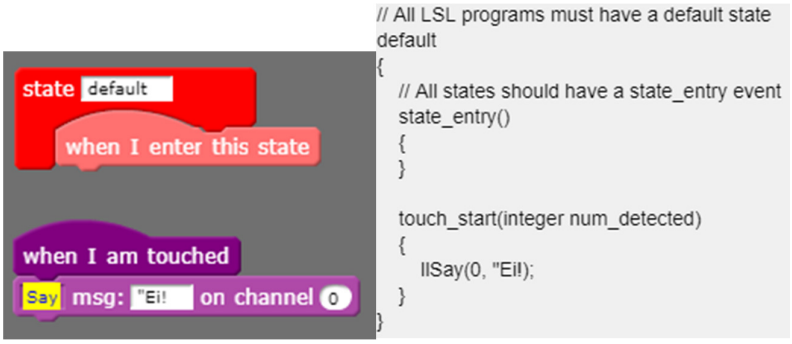
FUNCTIONS: Criação e utilização de funções customizados pelo usuário do FS2LSL.

PRIMS & PARTICLES: Blocos relacionados a primitivas e partículas.

7.2.2.4 Exemplos de aplicações

A aplicação abaixo, representada à esquerda pelo seu formato em blocos e à direita pelo código LSL gerado, usando o FS2LSL, define que, quando o objeto ligado a esse *script* for clicado, enviará a mensagem "Ei!" ao canal 0 do *chat*. Nota-se que o FS2LSL exige a existência de um estado-padrão para gerar o código (FIGURA 7.14).

Figura 7.14 - Codificando um *script* na ferramenta FS2LSL.



Fonte: dos autores.

Já o próximo exemplo faz com que o objeto comece a girar, assim que for criado a uma velocidade a uma taxa de 2π em torno do seu eixo Z. Após passados 10 segundos, o objeto cessa de girar (FIGURA 7.15).

Figura 7.15 - Exemplo de aplicação de blocos visuais para desenvolvimento de um MVI.

Neste último exemplo (FIGURA 7.16), o objeto começa a brilhar sempre que for clicado e deixa de brilhar, quando detecta alguma mensagem de seu dono, no canal 0 do *chat*.

Figura 7.16 - Demonstração de blocos visuais no desenvolvimento de *scripts* LSL.

7.3 PROCESSO DE CARGA E DEPURAÇÃO DOS *SCRIPTS*

Nesta seção, primeiro será apresentado como carregar um sprints em um *prim* (um objeto no Mundo Virtual Imersivo), utilizando soluções em *scripts* LSL. Na próxima subsessão, será apresentado o relatório de erros do depuração.

OpenSimulator, frequentemente chamado de *OpenSim*, é um servidor open source para hospedagem de Mundos Virtuais similares ao *Second Life*. *OpenSimulator* utiliza o *libsecondife,* para cuidar da comunicação entre o cliente e o servidor, portanto é possível conectar a um servidor *OpenSim* utilizando o cliente *Second Life* da Linden Lab. Outros clientes, para o *Second Life,* também, podem ser utilizados, uma vez que o *Second Life* e *OpenSim* utilizam os mesmos protocolos de comunicação.

Tudo pode ser programado nos Mundos Virtuais, por meio do LSL; podem-se fazer coisas engraçadas, como interagir um avatar com um simples objeto (cubo, por exemplo).

O LSL é uma linguagem de *scripting* que faz uso de eventos. Aqui tudo são eventos.

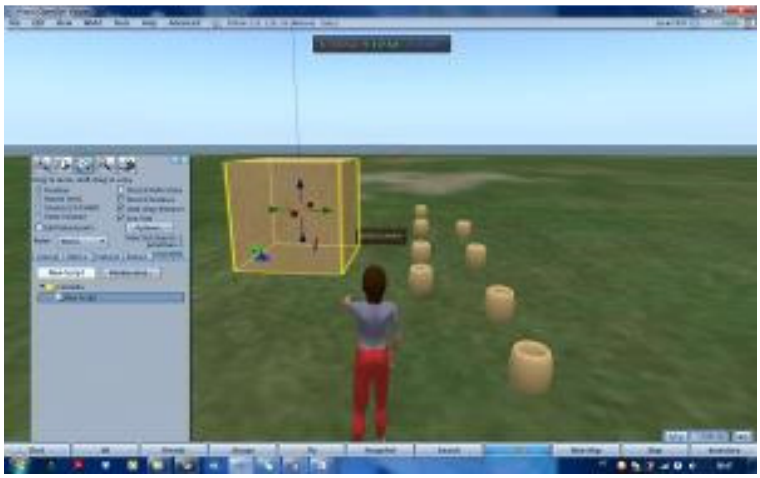
Todo o *script* segue a estrutura:

```
default
{
state_entry()
{
llSay(0, “Script running”);
}
touch_start(integer total_number)
{
llSay(0, “Hello Avatar”);
}
}
```

Um estado *default* e dentro deste estado podem-se ter outros, neste caso, será apresentado o estado *state_entry* e no *touch_start*.

Para se iniciar a escrever o *script*, é preciso colocar um cubo no chão, right-click (clique com botão direito do mouse) e escolher a opção *edit.* Em seguida, ir para o separado *content* e escolher *new script.*

Figura 7.17 – Escrita de *script*.



Abrindo o *script*, está pronto para programar.

Para começar, será apresentada uma programação que, quando o avatar formular algo, o nosso cubo responda com o típico “*Hello World*”!

Para isto, precisa-se de colocar no *state_entry* uma função que permita ouvir o que o nosso avatar diz. O state_entry é logo executado, quando o *script* é iniciado.

Essa função é o llListen(*integer channel, string name, key id, string msg*).

- channel é o canal em que vamos falar, 0 (zero), se for um canal público, em que todos ouvem o que é dito, ou outro número inteiro para comunicações privadas.
- Name é usado, quando queremos especificar o nome do avatar ou objecto, caso contrário, deixa-se em branco, usa-se ” “.
- ID é semelhante ao de cima, mas usando o UUID – identificação do avatar/Objecto.
- msg é a mensagem que queremos transmitir.

Desta forma, com este desempenho, o objeto irá receber a nossa mensagem. Esta função é complementada por:

```
listen( integer channel, string name, key id, string message )
{
llSay(0,”Hello “+message);
}
```

Desta forma, o que se escreve no *chat* (*World*, neste caso), será ouvido pelo objeto, guardado na variável *message* e concatenado à palavra *Hello*, formando o *Hello World*.

A função llSay (*channel, message*) é utilizada para comunicar entre avatares/Objetos. Existem outras funções deste tipo.

Figura 7.18 – Comunicaçao avatar/objetos.

O *touch_start* permite executar instruções, quando o avatar clica no objeto, ou seja, quando este evento é disparado. Neste caso, ao ser clicado, o objeto diz apenas “*Hello Avatar*”.

7.3.1 Relatório de erros e depuração

Pode-se usar o mesmo conceito, para rastrear problemas, depois que o código for concluído da seguinte forma:

```
notify ( string genus, string message )
 {
   criador chave = "a822ff2b-ff02-461d-b45d-dcd10a2de0c2" ;
 // linha próxima do comentário para desativar as notificações
 // genus = "off"
   if ( genus == "dbg" )
   {
   // comentou a próxima linha para desativar a depuração
      llInstantMessage ( criador, "DEBUG:" + mensagem ) ;
   }
   else if ( genus == "err" )
   {
      llOwnerSay ( "ERRO:" + mensagem ) ;
      llInstantMessage ( criador, "ERRO:" + mensagem + "no objeto" +
llGetObjectName ( ) + "que pertence a" + llKey2Name ( llGetOwner ( ) ) + "na
região" + llGetRegionName ( ) ) ;
   }
   else if ( genus == "info" )
   {
   // no prefixar porque é uma mensagem normal
      llOwnerSay ( mensagem ) ;
   }
   outro
   {
     ; // não faz nada, as notificações foram desativadas
   }
 }
```

Aqui faz-se uma série de coisas: lidar com crianças e diferentes de mensagens informativas à execução normal, relatório de erros para o proprietário e criador e mensagens de depuração. Para desativar a depuração, simplesmente comente a linha de depuração, conforme mostrado no código e descomente a linha genus = "off" para desativar todas as notificações.

Usa-se o llOwnerSay, para o proprietário atual e o llInstantMessage, para entrar em contato com o criador do *script* que continuará a funcionar, depois que a propriedade for aprovada. Basta inserir sua própria chave na parte superior.

Outros mecanismos podem ser usados e incluem llEmail ou llHTTPRequest, para postar uma mensagem, em um sistema de log externo talvez.

Definem-se os próprios tipos de mensagem personalizados e se comunica usando o método mais apropriado.

7.3.2 Mais reutilização

Pode-se estender ainda mais a reutilização, tornando-a um *script* discreto por si próprio dentro do objeto da forma:

```
notify ( string genus, string message )
 {
   criador chave = "a822ff2b-ff02-461d-b45d-dcd10a2de0c2" ;
 // desmarque esta linha para desativar as notificações
 // genus = "off"
   if ( genus == "dbg" )
   {
   // comentou esta linha para desativar a depuração
      llInstantMessage ( criador, "DEBUG:" + mensagem ) ;
   }
   else if ( genus == "err" )
   {
      llOwnerSay ( "ERRO:" + mensagem ) ;
      llInstantMessage ( criador, "ERRO:" + mensagem + "no objeto" + llGetObjectName ( ) +
"que pertence a"
                        + llKey2Name ( llGetOwner ( ) ) + "na região" + llGetRegionName ( ) ) ;
   }
   else if ( genus == "info" )
   {
   // no prefixar porque é uma mensagem normal
      llOwnerSay ( mensagem ) ;
   }
   outro
   {
     ; // não faz nada, as notificações foram desativadas
   }
 }
 padrão
 {
   state_entry ( )
   {
      notificar ( "info", "sistema de notificação pronto" ) ;
   }
   link_message ( integer sender_num, integer num, string str, id da chave )
   {
   // estamos usando a chave como um segundo parâmetro de string
      notificar ( str, id ) ;
   }
 }
```

Para enviar uma notificação, simplesmente use sua implementação-padrão:

```
{
  touch_start ( integer num_detected )
  {
     chave id = llDetectedKey ( 0 ) ;
     string message = llKey2Name ( id ) + "tentou me  roubar !" ;
  // isso pode parecer estranho, mas é válido usar o parâmetro chave de llMessageLinked
  // como um segundo recurso de passagem de string se não for realmente usado por uma chave
     llMessageLinked ( LINK_SET, 0, "info", mensagem ) ;
  }
}
```

7.4 FORMAÇÃO PARA AUTORIA EM LSL E OSSL

Nesta seção, serão abordados temas de relevância, para a formação de indivíduos, capacitados a construir soluções em ambientes Imersivos. Na primeira subseção (7.4.1), é apresentada a formação inicial em programação, a qual demonstra, por um conjunto de paradigmas distintos, como é possível capacitar programadores iniciantes. Já na seção seguinte (7.4.2), é abordada uma concepção, para a formação de docentes, a qual visa qualificá-los para a utilização dos Mundos Virtuais como ferramenta no processo de ensino e de aprendizagem.

7.4.1 A formação inicial em programação

Atualmente o uso de tecnologias é considerado um dos alicerces do novo cenário mundial, a que o indivíduo está constantemente exposto a ambientes computacionais e sua capacidade de utilizar tais recursos habilita-o a tomar decisões diárias, as quais necessitam de aptidão e domínio básico da lógica de programação. Essas são mudanças que exigem a adaptação e, desta forma, possibilitam a formação do cidadão do século XXI.

Identificando este momento de transformação, denota-se a importância da aprendizagem da Lógica de Programação, a qual já se encontra presente em diferentes meios, das escolas às universidades. A necessidade dos alunos compreenderem conceitos de raciocínio lógico é condição mínima, para que eles alcancem um desempenho desejável, em cursos de nível superior, que possuam disciplinas de programação em seus projetos pedagógicos, conforme apontado por Santos e Costa (2006). Alinhado a este conceito, observa-se a necessidade da construção do

Pensamento Computacional por estes indivíduos, que pode ser identificada como o conjunto de competências e habilidades básicas relacionadas à competência de aplicar estratégias algorítmicas para a solução de problemas (WING, 2006).

Em vários estudos, Silva (2014), Mattos (2008), Machado (2014) e Falckembach (2013) apontam que a dificuldade da construção do conhecimento pelos estudantes sobre programação é atribuída à metodologia de ensino, dita como tradicional, visto que este modelo, segundo Petry (2005), muitas vezes, não atende as expectativas do estudante, não motivando o seu interesse pelos conteúdos.

Ao aprofundar os estudos sobre programação, Borges (2000) acrescenta que o ensino de algoritmos comumente aborda um conjunto de ações comuns, conforme apresentado na Figura 7.19, iniciando-se por informações teóricas, exemplos, exercícios e execução de projetos mais avançados.

Figura 7.19 – Ações para o ensino de algoritmos e programação.

Fonte: Borges (2000).

Uma tendência, que aponta resultados pertinentes, é a adoção do ensino de algoritmos voltados à construção do Pensamento Computacional para crianças. Esta é uma estratégia adotada pelo Departamento de Educação Britânico (UK, 2013), tendo como foco disponibilizar um recurso educacional que possibilite ao aluno utilizar o Pensamento Computacional para resolver de forma criativa diferentes problemas. Desta forma, habilita-se o estudante a reconhecer, interpretar e sugerir soluções computacionais para os problemas a eles apresentados.

Além da proposta Britânica, encontram-se, na literatura, diferentes práticas pedagógicas que vislumbram alcançar um grau elevado de efetividade no ensino de programação, como o uso de realidade virtual (BARBAS; LOPES, 2013), adoção de dispositivos móveis como ferramenta de apoio (BARCELOS, 2012), prática com o uso de animações, simulações e *web* (BECK et al. 2014), intervenções complexas (BROWN, 1992), ambientes interativos – projeto Alice (CONWAY;

PAUSCH, 1997), ensino de programação por demonstração (FERREIRA et al. 2010), robótica como instrumento de aprendizagem e a utilização de redes sociais com prática em algoritmos (MIRANDA, 2014).

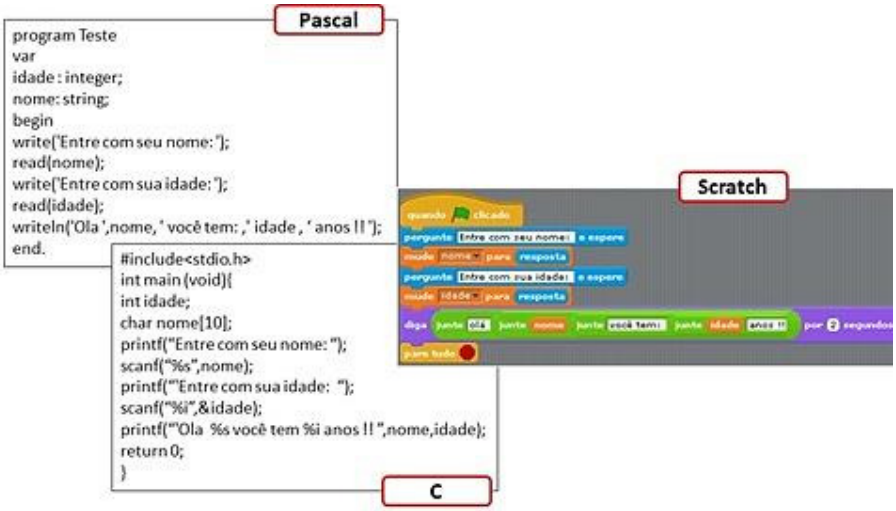
A partir da concepção de que o modelo tradicional de ensino de programação não atende a demanda dos principiantes e, reconhecendo a evolução das tecnologias, é passivel apontar a necessidade de mudanças no processo de ensino e de aprendizagem para Iniciantes. Assim sendo, as práticas atuais devem ser apoiadas por novas tecnologias (ambientes/plataformas), buscando, desta forma, disponibilizar recursos multimídia (animações, interatividade, vídeos) ao estudante.

Um ponto relevante neste contexto é o fato de que, para a construção do conhecimento sobre programação, o aluno precisa ser capaz de definir padrões e ordená-los de forma sistemática, além de reconhecer problemas e lidar com eles, a fim de resolvê-los. Porém a habilidade do estudante, ao visualizar uma solução e mapeá-la no espaço e conceber o código relacionado o auxilia na compreensão de todos os aspectos associados à programação, o que está diretamente pautado sobre a teoria de inteligência espacial. Ao entender esta concepção, é possível afirmar que a utilização de recursos visuais, como a programação em blocos visuais, beneficie o estudante no processo de aprendizagem da lógica e programação. Teoria já comprovada pelos estudos sobre a linguagem Logo de Abelson & Dissessa (1981) e Papert (1985).

Embora a LOGO seja uma linguagem simples, não teve muito êxito em sua finalidade de ser uma maneira fácil de programar por inúmeros motivos, segundo Resnik et al. (2009). Neste sentido, Papert (1980) argumentou que o entendimento de programação deva ser fácil para começar (*low floor*) e permite a construção de projetos elevados, ao longo do tempo (*high ceiling*) e apoie diferentes projetos para indivíduos com diferentes interesses (*wide walls*). Ou seja, para que uma linguagem de programação seja aceita, ela deve possuir piso baixo, teto alto e paredes largas, características inerentes ao ambiente gráfico de programação, desenvolvido pelo *Lifelong Kindergarten Group* (LKG, 2014), denominado Scratch. Constitui-se de uma interface baseada em ícones e blocos com a finalidade de facilitar o processo de aprendizagem de conceitos de computação, por meio do pensamento criativo, trabalho colaborativo e raciocínio sistemático (SCRATCH, 2014).

A fim de demonstrar a facilidade da programação com o Scratch, na Figura 7.20, são demonstrados três códigos, com a mesma funcionalidade, porém em linguagens diferentes (Pascal, C e Scratch). É visível a diferença de sintaxe e complexidade de implementação de cada aplicação.

Figura 7.20 – Comparação da programação em blocos visuais e outras linguagens.



Fonte: Adaptado de Scaico (2012).

A partir da necessidade de uma solução, para o ensino de programação para iniciantes, baseada em recursos computacionais atuais e que permita ao leitor iniciar seus primeiros passos, para o desenvolvimento de *scripts* para Mundos Virtuais, será apresentada, nesta etapa do livro, uma adaptação da metodologia proposta por Amaral, Medina e Tarouco (2016), na qual sugerem a adoção de uma abordagem didática, calcada em novos paradigmas, para o processo de ensino e de aprendizagem de algoritmos, buscando efeitos positivos. Objetiva-se, desta forma, a utilização de uma estrutura didática de ensino, baseada no entendimento sobre raciocínio lógico, construção de fluxogramas, programação com a linguagem Scratch, desenvolvimento com o Scratch4OS e construção de *scripts* para Mundos Virtuais com o LSL, conforme apresentado na Figura 7.21.

Figura 7.21 – Metodologia proposta para ensino de iniciantes em programação.

Fonte: Adaptado de Amaral, Medina e Tarouco (2016).

A aplicação desta proposta pelos autores, em sua pesquisa, visou estimular o aluno a desenvolver soluções de seu interesse, focando nos níveis motivacionais que poderiam ser alcançados com o uso destes sistemas. A motivação gerada pelo interesse desses estudantes, ao interagir com as aplicações Scratch, Scratch4OS e *OpenSim*, fomentaram a busca por um alto padrão de envolvimento. A motivação necessária ocorreu pela possibilidade de simulação aberta, inerente a estas aplicações, pois possibilitou, por meio da simulação, o desenvolvimento de hipóteses, testes e ajustes dos conceitos observados, assim como proposto por Valente (1997).

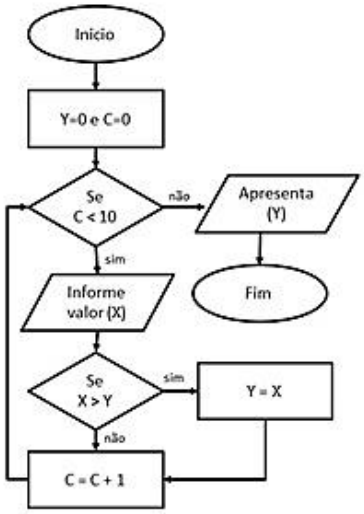
No modelo proposto, tem-se como primeira etapa a construção do raciocínio e lógica por parte dos iniciantes, visto que as principais causas dos problemas de programação, percebidas por programadores novatos, segundo Proulx (2000) e Faria & Adán Coello (2004), baseiam-se no desconhecimento de padrões de raciocínio lógico na solução de problemas. Sugere-se como recursos a utilização de problemas de raciocínio lógico, como os disponibilizados no Portal Pastelina (http://www.plastelina.net/). Estas atividades buscam desenvolver a capacidade do estudante em seguir uma sequência lógica para solucionar os desafios propostos. Como exemplo, cita-se o problema da Ovelha, Frutos e Lobo, apresentado na Figura 7.22, em que o estudante deve atravessar estes três elementos de uma margem à outra do rio, um de cada vez, respeitando a regra de que a ovelha não pode permanecer sozinha na mesma margem que os frutos e nem permanecer com o lobo.

Figura 7.22 - Prática de raciocínio lógico (ovelha, frutos e lobo).

Fonte: http://www.plastelina.net/

Como segunda etapa, no processo de aprendizagem de programação, é proposta a prática de resolução de problemas pelo uso de fluxogramas. Entendendo que os indivíduos apresentam uma maior capacidade de reconhecer algoritmos, por meio de recursos visuais, a representação gráfica por fluxogramas (BRUSILOVSKY, 1994) pode ser considerada uma solução, a fim de o aluno implemente respostas estruturadas e teste conceitos de maneira fácil e intuitiva, a partir da organização de símbolos com um fluxo e sequência lógica formalizados. Um exemplo de Fluxograma é demonstrado na Figura 7.23.

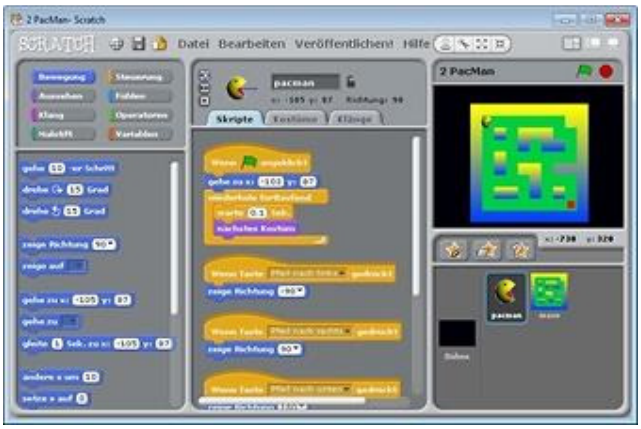
Figura 7.23 – Exemplo da resolução de problema com Fluxograma.



Fonte: Amaral, Medina e Tarouco (2016).

Com base na construção do raciocínio lógico, é possível ter os primeiros contatos com uma solução de programação. Neste sentido, a fim de proporcionar um ambiente amigável e de simples entendimento, propõe-se a utilização do Scratch (FIGURA 7.24), uma linguagem de programação em blocos visuais de fácil utilização, conforme apontado por Resnick (2009). Busca-se desenvolver a capacidade dos indivíduos em implementar programas simples, utilizando os recursos desta linguagem.

Figura 7.24 – Exemplo de aplicação (jogo) desenvolvida com Scratch.



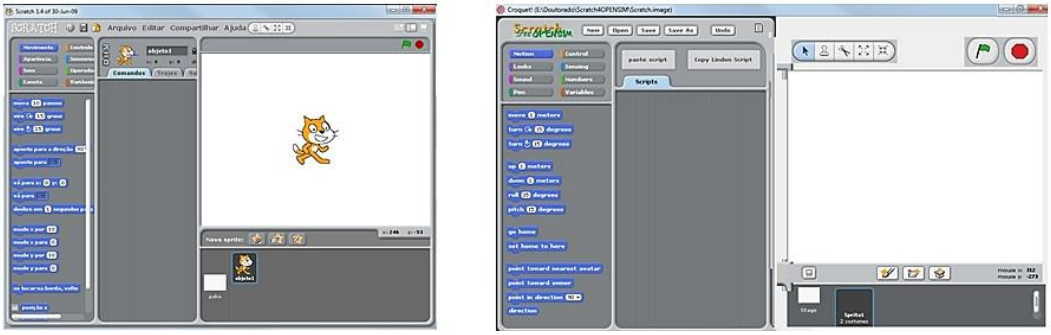
Fonte: https://scratch.mit.edu/

Nesta etapa, reconhecendo que o indivíduo tenha passado pela experiência de programar com o Scratch e, desta forma, desenvolvido habilidades necessárias à construção de soluções algorítmicas para problemas, é proposto o contato com outra ferramenta, o Scratch4OS. Um *software* baseado na linguagem Scratch, que também trabalha com o conceito de programação pela movimentação de blocos visuais (AVILA et al., 2013). O diferencial do Scratch4OS está no fato de permitir a tradução das construções visuais em *scripts* para Mundos Virtuais, funcionalidade que possibilita a programação de diferentes ações para objetos em um metaverso.

A utilização do Scratch4OS permite ao estudante ter os primeiros contatos com a sintaxe da linguagem LSL, pela construção e transporte de códigos desenvolvidos no Scratch4OS para os objetos (também denominados de *prisms*) do próprio Mundo Virtual *OpenSim*. Esta atividade possibilita ao programador reconhecer que estruturas criadas pelos blocos visuais podem ser convertidas em código (*scripts* LSL). Em suma, estas intervenções visam promover uma estrutura de ligação entre imagem e texto, a fim de que o aluno reconheça e compreenda a sintaxe dos *scripts* em LSL, relacionando o seu funcionamento com a base de conhecimento construída com os

estudos sobre programação no Scratch. A Figura 7.25 demonstra a semelhança dos ambientes de desenvolvimento Scratch e Scratch4OS.

Figura 7.25 – Ambiente de desenvolvimento Scratch e Scratch4OS.



Fonte: Amaral, Medina e Tarouco (2016).

Neste ponto, o método descrito pelos autores pressupõe que o indivíduo já tenha construído o conhecimento necessário sobre lógica de programação, algoritmos e resolução de problemas e, que, desta forma, esteja apto a construir soluções utilizando a linguagem de programação LSL. Observa-se que elementos como sintaxe e a formalidade, no desenvolvimento com a linguagem, serão obstáculos a serem ultrapassados, contudo a base sólida construída até o momento viabiliza e reduz o impacto gerado por estes elementos.

Por fim, com o intuito de comprovar a viabilidade desta metodologia, são apresentados resultados alcançados com a sua aplicação, em cursos de nível superior na Universidade Federal do Pampa, Campus Bagé. Os dados, na íntegra, podem ser obtidos em Amaral, Medina e Tarouco (2016).

A aplicação desta metodologia teve como foco o aprimoramento do processo de ensino e de aprendizagem da lógica, algoritmos e programação. Os diferentes paradigmas adotados estimularam as habilidades dos indivíduos, com os quais conseguiu-se propor soluções computacionais calcadas sobre o pensamento computacional e o raciocínio lógico efetivo. Além destas observações, analisou-se o desempenho dos estudantes, com base nas avaliações realizadas durante três semestres, com turmas distintas (TABELA 7.1). Salienta-se que, nos dois primeiros semestres (2013/2 e 2014/1), a ordem dos recursos (Lógica, Scratch e LSL) foram alternadas e, no semestre 2014/2, adotou-se exatamente a sequência descrita neste Capítulo.

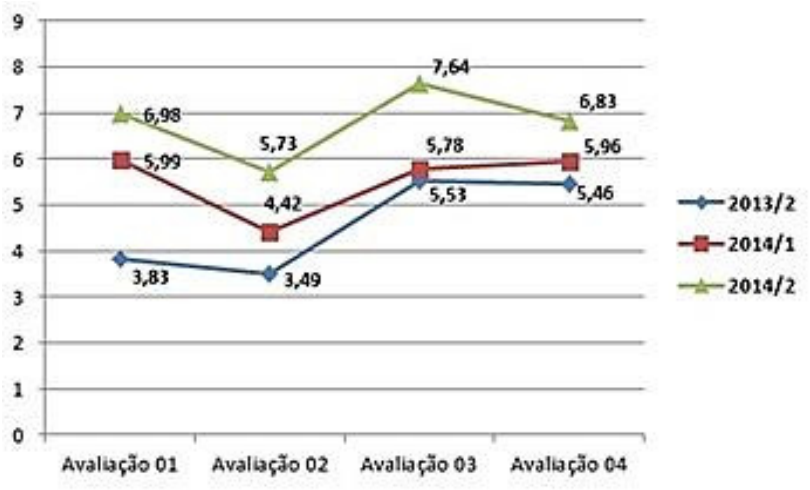
Tabela 7.1 – Participantes dos experimentos com a metodologia proposta.

Semestre x Matrículas		
Semestre	Matriculados	Frequentes
2013/2	60	21
2014/1	60	25
2014/2	60	22

Fonte: Amaral, Medina e Tarouco (2016).

A evolução dos alunos é evidenciada, na Figura 7.26, pelo gráfico que apresenta a comparação do desempenho dos estudantes, nas diferentes atividades avaliativas, realizadas durante os respectivos semestres. O progresso é percebido, em todas as quatro avaliações, com um aumento de 0,96 pontos na média de 2014/1 em relação a 2013/2, 1,26 de 2014/2 para 2014/1 e um avanço significativo de 2,22 pontos alcançados pelos alunos de 2014/2 sobre o grupo de 2013/1.

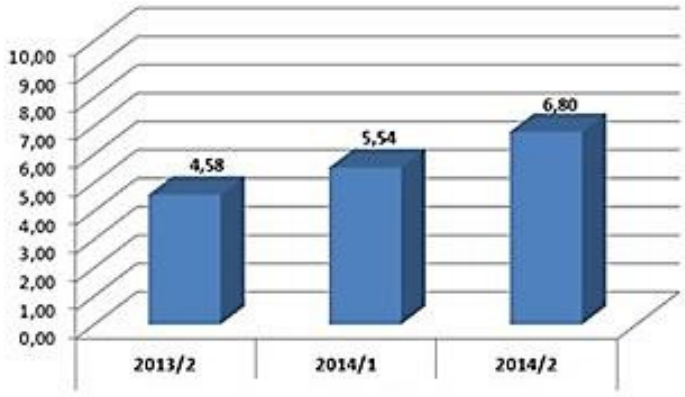
Figura 7.26 – Desempenho em atividades avaliativas por semestre.



Fonte: Amaral, Medina e Tarouco (2016).

A diferença das médias finais de todos os alunos, nos semestres observados, é mostrada na Figura 7.27. Assim, denota-se o gradual incremento dos resultados alcançados pelos estudantes nas disciplinas de algoritmos. A maior margem de intervalo de notas é percebida, no grupo geral, exibindo um acréscimo médio entre os semestres de 1,11 pontos.

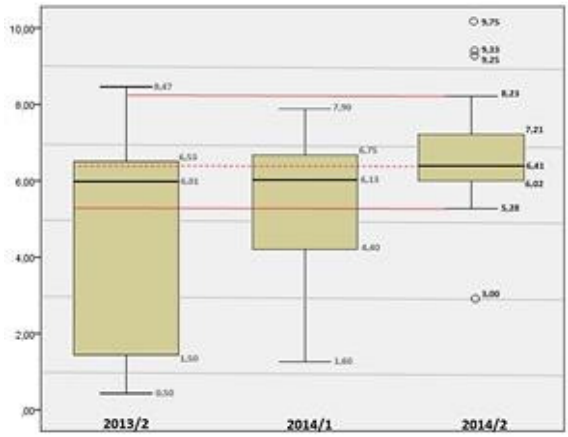
Figura 7.27 – Média final dos alunos.



Fonte: Amaral, Medina e Tarouco (2016).

A seguir, é apresentado o gráfico de comparação entre as tendências das médias dos estudantes para o período (FIGURA 7.28), considerando as notas individuais. Destaca-se, nesta representação dos *outliers* dos dados de 2014/2, referentes às notas de três acadêmicos (9,25/9,33/9,75), o bom nível de habilidade de programação desses estudantes. O limite superior no BoxPlot de 2013/1 encontra-se acima dos demais semestres, contudo a mediana, para este semestre, é a menor observada, sobressaindo novamente 2014/2, com um valor de média 7,21, para o terceiro quartil e 6,41 como mediana, sendo estes dados superiores aos demais períodos. Como informação relevante, nesta observação, distingue-se o limite inferior denotado, no segundo semestre de 2014, apontando um valor de 5,28, caracterizando que a predisposição para a melhora no desempenho ocorreu de forma linear e equânime, tanto pelos aprovados quanto pelos discentes com notas reduzidas (abaixo da média).

Figura 7.28 – Estudo comparativo entre médias.



Fonte: Amaral, Medina e Tarouco (2016).

Por fim, esta seção buscou apresentar uma proposta à formação inicial de programadores, vislumbrando que eles construam o conhecimento e desenvolvam habilidades necessárias à implementação de soluções algorítmicas em ambientes Imersivos. Reconhecendo que o ensino de algoritmos e programação não é uma tarefa simples, foi proposta a adoção de uma metodologia, que comprovadamente teve bons resultados, como estratégia para os primeiros passos do iniciante no mundo da programação.

7.4.2 A formação de docentes

A constante evolução dos dispositivos eletrônicos, em termos de memória, interface gráfica e capacidade de processamento, tem propiciado maior acesso a *softwares* que demandam um intenso uso desses componentes em função da sua alta complexidade.

Este é o caso dos Mundos Virtuais, ambientes Imersivos que simulam, com alto grau de detalhamento, aspectos do mundo real. Salienta-se que o uso dos Mundos Virtuais como ferramentas de réplica não necessariamente fica restrito à realidade. Ambientes desta natureza são comumente utilizados em jogos eletrônicos, simulando fantasias derivadas da realidade concreta. Como bem coloca Wang (2011, p. 4), "[…] computadores podem simular um cavalo, mas eles também podem facilmente simular um Pegasus".

Em função destas características, Mundos Virtuais tornam-se ambientes muito propícios para o desenvolvimento de situações de ensino e aprendizagem, calcadas na contextualização do conhecimento e na realização de tarefas autênticas. Tarefas autênticas são definidas por Jonassen (1999) como atividades que apresentam relevância ao estudante. Trata-se de desafios cognitivos com os quais o estudante deve se deparar no intuito de explorar, na prática, a aplicabilidade dos conhecimentos escolares em situações da vida real. Van Merriënboer e Kirschner (2013) esclarecem que o pleno domínio do conhecimento requer o desenvolvimento de habilidades para manipulá-lo e adequá-lo a diferentes situações.

Nos Mundos Virtuais, o estudante encontra a possibilidade de imergir em cenários contextualizados (ou até mesmo de construí-los) e, dentro destes contextos, pode realizar investigações, de acordo com as propostas pedagógicas estabelecidas, participar de simulações, interagir com objetos 3D (nem sempre facilmente construídos no ambiente concreto). Enfim, há uma diversidade de possibilidades pedagógicas alinhadas à ideia de construção de um conhecimento contextualizado que podem ser exploradas a partir de ambientes Imersivos.

O aprimoramento de atributos como a capacidade de processamento e interfaces gráficas das tecnologias disponíveis no mercado vem criando um cenário propício à profusão de ambientes Imersivos como os Mundos Virtuais. Estes ambientes, com características semelhantes aos jogos eletrônicos, simulam espaços complexos e podem ser explorados como réplicas virtuais do mundo real, ensejando o oferecimento de novas situações de ensino e aprendizagem que oportunizam a contextualização do conhecimento e o desenvolvimento de tarefas autênticas. Entretanto poucos docentes têm domínio sobre esta tecnologia, ainda considerada de alta complexidade. Com vista a promover uma aproximação do docente da Educação Básica com os ambientes Imersivos, foi elaborado um conjunto de estratégias, para compor um programa de formação, voltado ao desenvolvimento da prática docente nos Mundos Virtuais, envolvendo a produção de artefatos educacionais para tais ambientes (AVILA, 2016).

No intuito de delinear como poderia ocorrer o processo de autoria de futuros docentes, no Mundo Virtual, foi desenvolvido e testado um programa curricular à produção de artefatos educacionais para ambientes Imersivos.

O planejamento da atividade de formação envolveu constantes passagens pelo Mundo Virtual, no qual os estudantes tinham a missão de construir seus próprios laboratórios virtuais de aprendizagem, em suas respectivas áreas do conhecimento. Os laboratórios deveriam ser compostos por materiais educacionais produzidos, em ferramentas de autoria exploradas na disciplina, pela importação de conteúdos 3D e animações disponíveis em repositórios *on-line* e pela adição de *scripts* a objetos do Mundo Virtual, construídos com o apoio das ferramentas *Scratch for OpenSim* e *ScripTastic*.

Na atividade inicial da formação, solicitou-se aos alunos que explorassem o Mundo Virtual, fizessem leituras relacionadas ao uso de ambientes Imersivos no contexto educacional e, por fim, idealizassem um tema dentro de sua área de conhecimento o qual deveria nortear os artefatos que seriam produzidos. Com base no tema escolhido, seria construído, pouco a pouco, um laboratório virtual de aprendizagem dentro do Mundo Virtual. Para tanto, foi organizado o planejamento (TABELA 7.2).

Tabela 7.2 – Atividades de Planejamento.

Planejamento extensão	
Semana 1 (4h)	**Conhecendo os Mundos Virtuais** Discussão sobre Mundos Virtuais na Educação. Exploração inicial do Mundo Virtual ● Baixar e configurar visualizador para acesso. ● Cadastro no OSGRID. ● Visita a mundos no OSGRID. ● Cadastro no grid *Second Life*. ● Visita aos mundos do *Second Life*.
Semana 2 (4h)	**Desenvolvendo atividades básicas no Mundo Virtual** ● Construção e edição de objetos. ● Comunicação via bate-papo. ● Adição de imagens e páginas *web*.
Semana 3 (4h)	**Definição de um tema a ser explorado no Mundo Virtual** (discussão inicial com ideias para diferentes áreas do conhecimento). ● Importação de objetos a partir do repositório *OpenSim Creations* e do *Sketchup*. ● Animação de avatares.
Semana 4 (4h)	**Construção de *Scripts* para o Mundo Virtual** ● Apresentação da estrutura de programação da linguagem LSL. Idealização de uma animação simples a ser incorporada ao laboratório de cada professor. Utilização das ferramentas *ScripTastic* e *Scratch* for *OpenSim*. Utilização dos repositórios de *script* das linguagens LSL e OSSL: http://wiki.secondlife.com/wiki/LSL_Portal/pt, http://OpenSimulator.org/wiki/OSSL_Implemented
Semana 5 (4h)	**Instalando um servidor na máquina do usuário** ● Instalar e configurar um servidor de *OpenSim* na máquina do usuário. ● Baixar e importar bibliotecas para o Mundo Virtual. ● Salvar diferentes versões no mundo e abrí-las por meio do servidor.
Semana 6 (4h)	**Apresentação dos Laboratórios e avaliação por parte dos grupos** ● Discussão coletiva sobre possíveis melhorias a serem implementadas em cada um dos laboratórios produzidos.

A cada encontro presencial, novas ferramentas eram apresentadas e atividades de teste eram realizadas. Após a apresentação inicial, os docentes realizavam suas próprias produções, primeiramente, testando as ferramentas e, posteriormente, contribuindo para a construção de materiais que deveriam compor os laboratórios de aprendizagem de seus respectivos grupos. Ao

todo, na busca por contemplar o conteúdo descrito no planejamento do curso, seis tipos de atividades foram desenvolvidas nos encontros presenciais:

1. Modificar aparência do avatar.
2. Uso da ferramenta *chat.*
3. Criação e importação de objetos.
4. Inserção de mídias no Mundo Virtual.
5. Uso de animações.
6. Construção de *scripts*.

Com este conjunto de atividades, buscou-se oferecer aos docentes um domínio básico sobre ferramentas que compõem os Mundos Virtuais e que podem ser exploradas de forma simplificada por usuários leigos em *design* e programação. A partir da proposta de uso dos Mundos Virtuais como espaços, para a construção de laboratórios virtuais de aprendizagem, foi definido um conjunto inicial de estratégias que buscou promover a autoria dos docentes sobre o uso de tais ferramentas e buscou promover também a sua reflexão sobre o potencial pedagógico dos Mundos Virtuais no âmbito de suas respectivas áreas de conhecimento.

REFERÊNCIAS

ABELSON, H.; DISESSA, A. A. **Turtle Geometry: Computations as a Medium for Exploring Mathematics.** Cambridge, MA: MIT Press, 1981.

AMARAL, E.; MEDINA, R.; TAROUCO, L. M. R. **Processo de Ensino e Aprendizagem de Algoritmos Integrando Ambientes Imersivos e o Paradigma de Blocos de Programação Visual.** In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. 2016.

AVILA, B. G. **FORMAÇÃO DOCENTE PARA A AUTORIA NOS MUNDOS VIRTUAIS: UMA APROXIMAÇÃO DO PROFESSOR ÀS NOVAS DEMANDAS TECNOLÓGICAS.** Tese doutorado. PPGIE/UFRGS. 2016.

ÁVILA, B.; AMARAL, E.; TAROUCO, L. **Implementação de Laboratórios Virtuais no metaverso *OpenSim*.** RENOTE, v. 11, n. 1, 2013.

BARBAS, M.; LOPES, N. **Introdução à programação:(re) construção de espaços educacionais em realidade aumentada.** REVISTA DA UIIPS, p. 40, 2013.

BARCELOS, T.; SILVEIRA, I. F. **Pensamento Computacional e Educação Matemática: Relações para o Ensino de Computação na Educação Básica.** In: XX Workshop sobre Educação em Computação, Curitiba. Anais do XXXII CSBC. 2012.

BECK BRONDANI, M.; MOZZAQUATRO, P. M.; ANTONIAZZI, R. L. **Ambiente de simulação e animação para o ensino de programação.** Revista Interdisciplinar de Ensino, Pesquisa e Extensão, v. 1, n. 1, 2014.

BORGES, M. A. F. **Avaliação de uma metodologia alternativa para a aprendizagem de programação.** VIII Workshop de Educação em Computação – WEI. Curitiba, 2000.

BROWN, A. **Design experiments: theoretical and methodological challenges in creating complex interventions in classroom settings.** The Journal of the Learning Science, v.2, n.2, p.141-178, 1992.

BRUSILOVSKY, P. **Program visualization as a debugging tool for novices.** In: Proc. of INTERCHI'93 .Pp. 29-30. Amsterdam, 1994.

CONWAY M., J.; PAUSCH, R. **"Alice: Easy to Learn Interactive 3D Graphics"**, Computer & Graphics, Volume 31, Issue 3, pages 58- 59, 1997.

FALKEMBACH, G.; VIERO DE ARAÚJO, F. **Aprendizagem de Algoritmos: Dificuldades na Resolução de Problemas.** Anais SULCOMP, v. 2, n. 2, 2013.

FARIA, E. S. J.; ADÁN COELLO, J. M. **Detectando diferenças significativas entre programas como auxílio ao aprendizado colaborativo de programação.** In: WORKSHOP DE EDUCAÇÃO EM COMPUTAÇÃO, 2004.

FERREIRA, C.; GONZAGA, F.; SANTOS, R. **Um estudo sobre a aprendizagem de lógica de programação utilizando programação por demonstração.** In: Anais do XVIII Workshop sobre Educação em Computação, XXX CSBC, Belo Horizonte, MG, Brasil. 2010.

GUÍA DE INICIACIÓN LINDEN SCRIPTING LANGUAGE. Disponível em <http://wiki.secondlife.com/wiki/Main_Page>. Acesso em 23 mai. 2018.

KARIUKI, D. **New LSL Editor released. Hypergrid Business.** 2018 Disponível em https://www.hypergridbusiness.com/2018/03/new-lsl-editor-released/. Acesso em 10/11/2018.

LINDEN RESEARCH. SECOND LIFE WIKI (EN). Disponível em: <http://wiki.secondlife.com/wiki/Help:Getting_started_with_LSL. 2015>. Acesso em 13 ago. 2018.

LKG, Lifelong Kindergarten Group. Disponível em: <http://llk.media.mit.edu/>. Acessado em: 14 setembro 2014.

LSL WIKI HOMEPAGE. Disponível em <http://lslwiki.digiworldz.com/>. 2012. Acesso em: 13 ago. 2018.

MACHADO, R. F. S.; RAABE, A. L. A.. **Modelagem Cognitiva dos Problemas de Aprendizagem de Algoritmos.** Universidade do Vale do Itajaí. Disponível em: <http://200.169.53.89/download/CD%20congressos/2008/SBIE/sbie_posters/Modelagem%20cognitiva%20dos%20problemas%20de%20aprendizagem.pdf > Acesso em: 20 abril 2014.

MATTOS, M. M.; VAHLDICK, A. **Relato de uma experiência no ensino de algoritmos e programaçao utilizando um framework lúdico.** In: Anais do II Workshop de Ambientes de apoio à Aprendizagem de Algoritmos e Programação. 2008.

MIRANDA, F. N. **"Facebook e Algoritmos: Um Estudo de Caso do Uso do Facebook como Suporte Pedagógico na Disciplina de Algoritmos e Técnicas de Programação do Curso de Sistemas de Informação em IES.** Disponível em: <http://www.faminasbh.edu.br/upload/downloads/20130627151026_83686.pdf>. Acesso em: 20 junho 2014.

*OPENSIM. **OPENSIM*: SCRIPTING LANGUAGES.** 2012. Disponível: <http://*OpenSim*ulator.org/wiki/Scripting_Languages>. Acesso em: 09 ago. 2018.

PAPERT, S. **LOGO: Computadores e Educação.** São Paulo, Brasiliense, 1985.

PAPERT, S. **Mindstorms: Children, Computers, and Powerful Ideas.** Basic Books, New York, 1980.

PETRY, P. G. **Um sistema para o ensino e aprendizagem de algoritmos utilizando um companheiro de aprendizagem colaborativo.** Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, UFSC, 2005.

PROULX, V. K. **Programming patterns and design patterns in the introductory computer science course.** ACM. ACM SIGCSE Bulletin. [S.l.], 2000.

RESNICK, M. et al. **Scratch: programming for all. Commrmication of the ACM.** vo1.52. n.ll. pp.60-67, 2009.

SANTOS, R. P. dos; COSTA, H. A. X. **Análise de Metodologias e Ambientes de Ensino para Algoritmos, Estruturas de Dados e Programação aos Iniciantes em Computação e Informática.** In: INFOCOMP, Volume 5, nº.1, ISSN 1807-4545, 2006.

SCAICO, P. D.; DE LIMA, A. A.; DA SILVA, J. B. B.; AZEVEDO, S.; PAIVA, L. F.; RAPOSO, E. H. S.; MENDES, J. P. **Programação no Ensino Médio: Uma Abordagem de Ensino Orientado ao *Design* com Scratch.** In Anais do Workshop de Informática na Escola (Vol. 1, No. 1), 2012.

SCRATCH. **About Scratch (Scratch Documentation Site).** Disponível em: <http://info.scratch.mit.edu/ About_Scratch>. Acesso em: 18 julho 2014.

SGOBBI S. F.; TAROUCO R. M. L.; REATIGUI, E. **The pedagogical use of the Internet of Things in virtual worlds to encourage a behavior change in obese individuals,** Conference Ithings 2017 Exeter, Inglaterra, June 21-23, 2017 Proceedings. IEEE, 2017.

SILVA, Í. F. A. ; SILVA, I. M. M.; SANTOS, M. S. **Análise de problemas e soluções aplicadas ao ensino de disciplinas introdutórias de programação.** Universidade Fed. Rural de Pernambuco, Recife – PE. 03 pp. Disponível em: < http://www.eventosufrpe.com.br/jepex2009/cd/resumos/R1479-1.pdf>. Acesso em: 18 abril 2014.

UK. **Statutory Guidance, National Curriculum in England: computing programmes of study.** Published 11 Seteptember 2013. Disponível em: <https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study>. Acesso em: 10 setembro 2014.

VALENTE, J. A. **Informática na educação: instrucionismo x construcionismo.** Manuscrito não publicado, NIED: UNICAMP, 1997.

WING, J. M. **Computational thinking. Communications of the ACM,** v. 49, n. 3, p. 33–35, 2006.

ANEXOS

ANEXO 1 - VARIÁVEIS

Os tipos de variáveis são usadas para os diferentes objetivos que se pretendem alcançar:

- integer: armazena números inteiros (1, 2, 5 e 7);
- float: guarda números reais (1.235 ou 10.0001);
- string: armazena texto (“Olá Avatar!”);
- key: variável especial usada para identificar características próprias de elementos do Mundo Virtual, como por exemplo: um avatar, um objeto ou uma textura;
- vector: uma variável composta por três números de tipo float (<1.0,1.0,1.0>);
- rotation: variável composta por 4 números de tipo float, usada para dar movimento aos objetos.

ANEXO 2 - OPERADORES

Para a realizar cálculos e operações lógicas, o LSL utiliza estes operadores:

X=Y : X **recebe** o valor de Y;

X==Y : a comparação é verdadeira se o valor de X é **igual** ao valor de Y;

X!=Y : a comparação é verdadeira se o valor de X é **diferente** do valor de Y;

X < Y : a comparação é verdadeira se o valor de X é **menor** do que valor de Y;

X > Y : a comparação é verdadeira se o valor de X é **maior** do que valor de Y;

X<=Y : a comparação é verdadeira se o valor de X é **menor ou igual** ao valor de Y;

X>=Y : a comparação é verdadeira se o valor de X é **menor ou igual** ao valor de Y;

+ : mais (soma);

- : menos (subtração);

* : multiplicação;

/ : divisão;

++ : incremento de 1 em 1;

-- : decremento de 1 em 1;

% : módulo (resto de divisão);

&& : operador lógico e (and)

|| : operador lógico ou (or)

ANEXO 3 - COMANDOS LSL

As características do texto podem ser alteradas com a função llSetText, que possui a sintaxe [llSetText(string text, vector color, float alpha);] . O primeiro parâmetro text define o texto que vai ser apresentado acima do objeto. O parâmetro color indica a cor do texto, ele é formado por 3 valores que representam a formatação RGB (Red, Green, Blue) e a intensidade da cor é definida pelo número que varia de 0.0 a 1.0, desta forma, <1.0, 0.0, 0.0> = vermelho, <0.0, 1.0, 0.0> = verde, <0.0, 0.0, 1.0> = azul, <0.0, 0.0, 0.0> = preto e <1.0, 1.0, 1.0> = branco. O parâmetro alpha indica a transparência do texto, variando de 0 (completamente transparente) a 1 (completamente opaco). O Código 4 apresenta uma aplicação do llSetText.

Código Anexo 4: Função llSetText().

```
touch_start(integer total_number)
{
// mostra o texto “NOME” em vermelho e sólido
llSetText(“NOME”, <1.0,0.0,0.0>, 1.0);
}
```

De forma semelhante, a função llSetColor muda a cor do objeto nas respectivas faces do objeto. A função possui esta composição [llSetColor(vector color, integer face);]. O parâmetro color é semelhante ao da função llSetText. Já o parâmetro face é o “lado” da figura geométrica que será alterado. O Código 5 mostra a função llSetColor.

Código Anexo 5: Função llSetColor().

```
touch_start(integer total_number)
{
// altera todas as faces do objeto para verde
llSetColor(<0.0,1.0,0.0>, ALL_SIDES);
}
```

Além da cor, a posição do objeto pode ser verificada e alterada pelas funções llGetPos e llSetPos, respectivamente. A função llGetPos() apresenta o vetor com as coordenadas do objeto na região em que ele se encontra. Um retorno para a função llGetPos() poderia ser <123.5, 25.3, 86.2>, em que X=123.5, Y=25.3 e Z=86.2.

Já a função llSetPos(vector pos) coloca o objeto em uma nova posição pelo parâmetro o qual é um vetor com as coordenadas <X,YZ>. O Código 6 mostra estas funções.

Código Anexo 6: Funções llGetPos() e llSetPos().

```
touch_start(integer total_number)
{
// escreve a posição do objeto capturada com llGetPos()
llSay(0,"A posição do objeto é "+(string)llGetPos());
// a partir da captura da posição do objeto,
// altera sua posição em 1 metro com llSetPos()
llSetPos (llGetPos ()+<0, 0, 1>);
llSay (0,"A nova posição é "+(string)
llGetPos());
}
```

Muitas vezes, precisa-se de que uma determinada ação seja realizada de tempos em tempos. A linguagem LSL permite fazer isto com a função llSetTimerEvent.Esta função faz com que o evento do temporizador seja acionado, no máximo, uma vez a cada 'X' segundos, assim:

llSetTimerEvent(float sec);

O parâmetro sec é a quantidade de segundos que o evento deve esperar. Por sua vez, a função llFrand(float mag)um valor do tipo float entre 0,0 e o número informado. Neste exemplo, ele irá varia entre 0 e 1. O Código 7 mostra um exemplo destas funções.

Código Anexo 7: Função llSetTimerEvent.

```
default
{
    state_entry ()
    {
            // define em 5 segundos o tempo de espera
            llSetTimerEvent(5);
    }

    timer ()
    {
    float x= llFrand(1); // gera um nro aleatorio para x
    float y= llFrand(1); // gera um nro aleatorio para y
    float z= llFrand(1); // gera um nro aleatorio para x
    llSay(0, "x= "+ (string)x +" y= "+ (string)y + " z= "+ (string)z);
    llSetColor (<x, y, z>, ALL_SIDES);
    }
} // fim de default
```

Como descrito anteriormente, o *OpenSim* possui um grande número de canais para comunicação. Este canais são úteis, quando se deseja que um objeto execute certas instruções,

sem que seja necessário tocá-lo. A função llListen faz com que um objeto “escute” os canais de mensagens. Sua sintaxe é: llListen(integer channel, string name, key id, string msg), em que channel é o canal que o objeto deve escutar, name identifica um avatar ou objeto específico pelo nome, id identifica um avatar ou objeto específico pela identificação numérica e msg é a mensagem recebida. A função llListen executa o evento Listen, cujas instruções LSL são codificadas.

Para enviar uma mensagem por um canal no *viewer*, é necessário digitar o canal para o qual se quer enviar a mensagem, seguido da mensagem, assim [/33 vermelho] por exemplo. A Figura 7.29 mostra o *chat* local, que fica no canto esquerdo inferior dos *viewers*. O Código 8 apresenta um exemplo de uso dos canais de comunicação.

Figura 7.29 - Local chat para comunicação no Mundo Virtual.

Código Anexo 8: Função llListen().

```
default
{
    state_entry()
    {
llSay(0, "Escutando a porta 33.");
// escuta o canal 33 para qualquer emissor
// com qualquer mensagem
llListen(33, "", "", "");
}

listen(integer channel, string name, key id, string message)
    {
            llSay(0, "Cor recebida: " + message);
            if ( message == "vermelho" )
                    {
                    llSetColor( < 1.0, 0.0, 0.0 >, ALL_SIDES );
                    }
            else if ( message == "verde" )
                    {
                    llSetColor( <0, 1, 0>, ALL_SIDES);
                    }
            else if (message == "azul" )
                    {
                    llSetColor( <0, 0, 1>, ALL_SIDES);
                    }
            else
                    {
```

```
                    llSay (0, "Informe vermelho, verde ou azul");
                    }
    } // fim listen
} // fim default
```

Além da comunicação por canais, o OpenSim possui a função llDialog(), a qual possibilita a criação de menus, que são caixas de diálogo apresentadas no canto superior esquerdo da tela ou inferior direito, os quais podem ser usados como recursos de seleção e fluxo de controle das ações do usuário no ambiente virtual. A função llDialog() possui a estrutura: llDialog(key avatar, string message, list buttons, integer channel), em que avatar é o usuário que verá o menu, message é o texto apresentado na caixa de diálogo, buttons são os textos para cada um dos botões e channel é o canal de comunicação usado por este menu. O Código Anexo 9 mostra a função llDialog().

Código Anexo 9: Função llDialog().

```
// define 100 como o canal de comunicação
integer canal=100;
default
{
   state_entry ()
   {
      // escuta o canal de comunicação definido
      llListen(canal, "", "", "");
      // ao iniciar, altera a cor do objeto para preta
      llSetColor(<0,0,0>, ALL_SIDES);
   }

   touch_start (integer total_number)
   {
    // mostra o menu para o avatar que tocar no
// objeto através da llDetectedKey(0)
llDialog(llDetectedKey(0), "Escolha uma cor", ["Vermelho", "Verde",    "Azul"],
canal);
   }

listen (integer channel, string name, key id, string message )
   {
      // escreve a cor escolhida na tela
      llSay(0, "Você escolheu: " + message);
      // muda a cor do objeto de acordo com a escolha
      if ( message == "Vermelho")
      {
         // altera a cor para Vermelho
         llSetColor(<1,0,0>, ALL_SIDES);
      }
      else if ( message == "Verde" )
      {
```

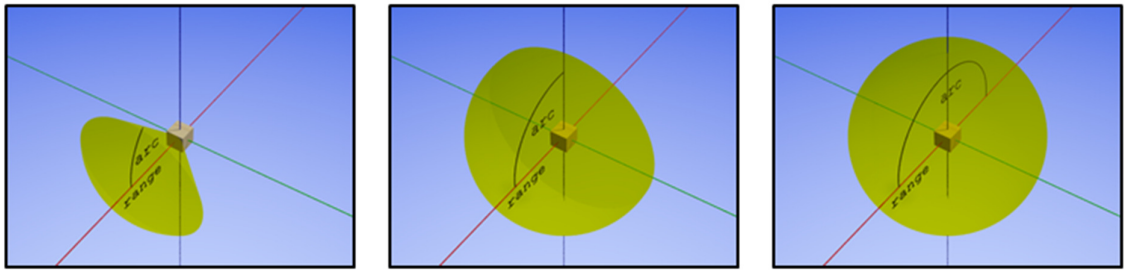
```
        // altera a cor para Verde
        llSetColor(<0,1,0>, ALL_SIDES);
    }
    else if ( message == "Azul" )
    {
        // altera a cor para Azul
        llSetColor(<0,0,1>, ALL_SIDES);
    }
  } // fim listen
} // fim default
```

Por vezes, almeja-se que determinado *script* LSL seja executado sem que o usuário toque em um objeto. Isto pode ser feito por meio da função llSensor(). Sua sintaxe é: llSensor(string name, key id, integer type, float radius, float arc).

O parâmetro ***name*** é o nome do avatar ou do objeto procurado, ***id*** é a identificação única do grupo, avatar ou ***object*** na região, ***type*** é a máscara de identificação para o grupo, avatar ou objeto, ***radius*** é o raio do centro do objeto até o limite definido (de 0.0 até 96.0 metros do Mundo Virtual), ***arc*** é o ângulo máximo entre o eixo X dos objetos e o eixo X do avatar ou objeto detectáveis, variando de 0.0 até PI. A Figura 7.30 mostra algumas opções de construção destes ângulos.

Além da detecção dentro do *radius*, é possível atribuir um *script* LSL para quando o avatar ou objeto ultrapassarem o limite da distância. O evento no_sensor() é responsável por esta condição. Na execução do Código Anexo 10, quando o avatar se aproxima a menos de 10 metros e toca no objeto, uma mensagem é apresentada, mas se a distância for maior do que 10 metros, outra mensagem é apresentada.

Figura 7.30 - representação dos ângulos para o parâmetro arc.



Cone do sensor arc=PI/4 Cone do sensor arc=PI/2 Cone do sensor arc=PI

Código Anexo 10: Função llSensor().

```
default
{
 touch_start (integer total_number)
{
// varre o ambiente buscando qualquer nome e id de avatar
// para o tipo AGENT, na distância de 10 mt, com ângulo PI
    llSensor("", "", AGENT, 10, PI);
}

 sensor (integer num)
{
    // escreve o nome do avatar detectado a partir
//da função llDetectedName()
   llSay(0, "Avatar "+ llDetectedName(0) + " detectado!");
    }

 no_sensor()
    {
    llSay(0, " Nenhum avatar detectado!");
    }
} // fim default
```

Como descrito anteriormente, o LSL permite realizar a mudança entre estados, como foi relatado no exemplo da lâmpada. O Código 11 mostra a transição entre estados.

Código Anexo 11: Mudança de estados no LSL.

```
// no estado default, a lâmpada inicia "desligada"
default
{
    state_entry()
    {
            llSay(0, "Lâmpada desligada.");
    }

    touch_end(integer num_detected)
    {
            llSay(0,"Ligando .....");
            state ligada;
    }
} // fim estado default

state ligada
{
    state_entry()
    {
            llSay(0, "Lâmpada ligada.");
    }
```

```
    touch_end(integer num_detected)
    {
            llSay(0,"Desligando .....");
            state default;
    }
} // fim state ligada
```