FACULDADES INTEGRADAS DE TAQUARA
CURSO DE SISTEMAS DE INFORMAÇÃO

DESENVOLVIMENTO DE UM JOGO PARA COMPUTADOR UTILIZANDO BLENDER E PYTHON

CELIO PAULO FERREIRA DE ALMEIDA KIRSCH

TAQUARA

2010

CELIO PAULO FERREIRA DE ALMEIDA KIRSCH

DESENVOLVIMENTO DE UM JOGO PARA COMPUTADOR UTILIZANDO BLENDER E PYTHON

Trabalho de conclusão de curso apresentado ao Curso de Sistemas de Informação das Faculdades Integradas de Taquara, sob orientação do Me. Marcelo Azambuja.

Taquara

2010

AGRADECIMENTOS

Ao orientador professor Marcelo Azambuja pela auxilio no desenvolvimento do projeto.

A minha esposa, pela paciência e ao meu filho, pelo incentivo.

RESUMO

Resumo: *Esta monografia demonstra a construção de um jogo para computador. Relata, de forma sucinta, a história dos jogos para computador, a metodologia, a construção e as ferramentas utilizadas. A importância da inteligência artificial no jogo está baseada no algoritmo de busca, pois, seu objetivo é deixar o jogo interessante e imprevisível para o jogador.*

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento da tecnologia de jogos para computador sempre sofreu influência dos movimentos sociais, políticos e culturais durante sua evolução. Segundo (CHANNEL, 2007), com o impacto da guerra fria, nos anos 50, e a possibilidade das potências Estados Unidos e União Soviética destruírem o mundo com suas bombas nucleares levou os cientistas a desenvolver tecnologias computacionais para simular testes balísticos de mísseis e criar cenários para prever os resultados de uma guerra nuclear. A mesma tecnologia foi utilizada pelo cientista William Higinbothan para desenvolver o primeiro jogo para computador em 1958 conhecido como *Tennis for Two*, em um rústico osciloscópio. Na era da Corrida Espacial e Guerra do Vietnã, jogos como *Space War,* de Steve Russell, surgiram junto com as primeiras empresas do ramo de videogames. O sucesso foi estrondoso durante anos, pois o conceito de coloca o jogador no controle da diversão é superior ao fato de assistir televisão de forma passiva. Entre os anos de 1970 e 1980 uma revolução acontece: Jogos com espaçonaves ou imagens que representam raquetes de tênis ou carros dão lugar a uma nova tecnologia de videogame onde os personagens são reconhecíveis pelo seu rosto e abre a possibilidades de se criar histórias complexas como *Mario*, *Donkey Kong* e *Zelda*. A passagem para a adolescência dos adoradores de videogames na década de 90 já não estavam sendo supridas pela indústria e para acompanhar a cultura pop da época, empresas correm para lançar personagens carismáticos e heróicos como *Sonic*. Por outro lado, anti-heróis durões e mais realistas também eram bem aceitos pelo público como no jogo *Grand Theft Auto III*.

Outra revolução aconteceu com a tecnologia 3D. Esta tecnologia rica em detalhes coloca o jogador literalmente “dentro do jogo”. Conforme (CHANNEL, 2007) e suas características marcantes são perspectiva e profundidade nos cenários que seduziam a todos tanto pelo ambiente como pelo tema polêmico abordado nos primeiros jogos. *Doom* e *Castle Wolfenstein 3-D,* considerados jogos revolucionários, simulavam com precisão o mundo real e criavam o paradoxo ente o que é real e o que é virtual. Apesar da violência destes dois jogos com atiradores em primeira pessoa, eles foram um gênero muito popular em sua época. Outro acontecimento revolucionou o mundo dos videogames: Após os ataques terroristas de 11 de setembro, o governo norte americano lança seu próprio jogo com o intuito de recrutar jovens para o exército. Surgia um novo jogo, *American’s Army,* e uma nova polêmica também. Estaria certo recrutar jovens para guerras reais através da brincadeira de um videogame?

Jogos com temas étnicos que simula conflitos entre nações estariam gerando uma guerra virtual?

Outro evento marca os jogos de videogames: A internet revolucionou o mundo e os videogames também. Conforme (CHANNEL, 2007), quando a *ARPANET* apareceu, quase que imediatamente usuários começaram a usar esta nova tecnologia para jogar. Ela foi criada em 1960, interligava computadores militares e foi precursora da internet. Jogar pela internet criou a cultura onde as pessoas podem criar seus personagem e nele acrescentar traços pessoais para serem reconhecidos neste mundo. Esta possibilidade de ter uma vida virtual paralela atraiu muitas pessoas e levou a outra polêmica: é natural pessoas passarem mais tempo neste mundo virtual do que no mundo real? Por outro lado, experiências virtuais são na verdade experiências humanas e são proporcionadas pelas modernas tecnologias de jogos eletrônicos.

Apesar do pouco desenvolvimento na área, há empresas internacionais interessadas no mercado de desenvolvimento de jogos no Brasil. Elas acreditam na capacidade de desenvolvimento dos programadores nacionais. Conforme (GLOBO, 2009), empresas de destaque internacional no desenvolvimento de jogos eletrônico, como *Sony*, estão observando os desenvolvedores nacionais através de eventos como o Simpósio Brasileiro de Jogos e Entretenimento Digitais (SBGames), bem como em contatos com a Associação Brasileira das Desenvolvedoras de Jogos Eletrônico (Abragames) para conhecer a tendência dos desenvolvedores nacionais.

2 JUSTIFICATIVA

Para alcançar os objetivos das disciplinas de Estágio e Trabalho de Conclusão de Curso (TCC), bem como solidificar os conhecimentos adquiridos ao longo do curso de Sistemas de Informação, este trabalho apresenta o processo de desenvolvimento de software na área de entretenimento.

Entre os tipos de jogos eletrônicos atualmente conhecidos, o foco desse estudo é direcionado para jogos de computador, e entre os vários tipos de jogos, o gênero tiro em primeira pessoa (FPS).

Do inglês, FPS (*First Person Shooter)* ou jogo de tiro em primeira pessoa significa o jogador assumir o lugar do personagem principal. A visão do personagem é substituída pela visão do jogador, como se ele estivesse, de fato, dentro do jogo e interagindo diretamente com o ambiente.

O projeto de um jogo se mostra promissor devido ao fato de abranger várias áreas da ciência da computação e proporcionar o desenvolvimento dos conceitos de gerenciamento de projeto, desenvolvimento e análise de software. Ao escolher um projeto de desenvolvimento de jogos, o acadêmico desenvolve uma atividade multidisciplinar atuando em várias áreas do conhecimento humano e de Sistemas de Informação. Segundo (CLUA e BITTENCOURT, 2005), com esta diversidade, todos os módulos devem funcionar em perfeita harmonia e explorar ao máximo o hardware e placas gráficas. Enquanto a maioria dos softwares segue apenas uma série de requisitos e atende somente para o que foi programado, os jogos, além disso, devem ser divertidos e agradáveis, pois devem proporcionar entretenimento para as pessoas. A imersão é sua característica mais forte e obtida através da arte e da tecnologia, unindo Educação, Psicologia, Artes Plásticas e aspectos computacionais como a compreensão de varias tecnologias como Computação Gráfica, Inteligência Artificial, Redes de Computadores e Multimídia.

Tratando-se esse trabalho de um projeto didático, seu objetivo maior será a demonstração das etapas de construção, os resultados alcançados, as metodologias, tecnologias e algoritmos utilizados.

3 OBJETIVO

As seções a seguir apresentam o objetivo geral do jogo e os vários objetivos específicos desse software.

3.1 Objetivo geral

Este trabalho tem côo objetivo principal desenvolver um jogo de tiro em primeira pessoa, (**FPS – *First Person Shooter)***, para computador, bem como demonstrar a construção gráfica desse jogo utilizando a ferramenta de software *Blender*. O controle lógico do jogo foi implementado com a linguagem de programação *Python*. O trabalho visa também analisar as melhores técnicas a fim de tornar o jogo adequado em termos de interação como o usuário.

3.2 Objetivos específicos

a) pesquisar softwares e tecnologias que atendam as necessidades do programa levando em consideração a filosofia de software livre;

b) especificar o gênero do jogo, estruturar sua história e seus objetivos;

c) modelar os personagens e criar animações, desenvolver a inteligência artificial e testes de forma apropriados.

4 TECNOLOGIAS

Um aspecto importante do projeto é a adoção de tecnologias com a filosofia de software livre, portanto, os softwares utilizados para o desenvolvimento são todos gratuitos. O uso da linguagem de programação Python se justifica por ser ela a base para as programações do aplicativo Blender. Esta linguagem foi utilizada para desenvolver a inteligência artificial do jogo, na construção de scripts que controlam principalmente o personagem não jogador, aquele controlado pelo computador. Segundo (LUTZ e ASCHER, 2007), Python é uma linguagem orientada a objetos, de estruturação e reutilização de código. Pode-se ainda construir scripts para sistemas orientados a objetos como *C++* e *Java* e, assim como *Perl* e *Linux*, seu código-fonte é aberto e com uma grande rede de suporte pela internet.

O Gimp – GNU (*Image Manipulation Program*) é um editor de imagens utilizado para criar e manipular imagens e fotografias. Sua utilização no projeto foi para manipulação de imagens, desenvolver texturas, cor dos objetos e aparência dos objetos. Segundo (MILANI, 2005), o Gimp é um programa de edição de imagens sob a licença GPL - *General Public License* que se refere aos softwares com liberdade de execução, estudo, modificação e repasse sem que o desenvolvedor necessite pedir permissão ao autor do programa.

Blender foi o principal software para desenvolver o projeto. Nesta ferramenta os testes são feitos simultaneamente, ou seja, a modelagem, a texturização e a ação de executar o jogo na ferramenta podem ser feitos a qualquer momento, sem a necessidade dele estar pronto. Conforme (BRITO, 2008), é uma ferramenta completa para criação de objetos 3D, para modelação, texturização e visualização de conteúdo 3D interativo. O Blender possibilita a criação de espaços tridimensionais, imagens, vídeos e com seu motor 3D, em tempo real permite o desenvolvimento de jogos com sua própria *game engine*. Originado da empresa "*Not a Number*" (NaN), ele é hoje "Software Livre" e disponível sobre licença GNU GPL.

Arbaro é um software para criação de imagens de árvores. Com comandos simples é possível desenhar troncos, folhas e galhos. Feito o desenho, basta transferir o arquivo par o Blender e o desenho é transformado em um objeto 3D.

Jude foi utilizado para documentar o projeto com diagramas UML. Conforme (PIMENTA, 2006), o Jude é um software para modelagem que suporta desenhos de sistemas de orientação a objetos e diagramas UML.

A Inteligência Artificial do jogo teve um aspecto relevante no projeto, pois para atender as expectativas do usuário o software deve simular o ambiente com maior realismo

possível. Foi adotado o algoritmo de busca A*, (pronuncia-se “A - estrela”), para controlar o inimigo do usuário (jogador). Sua função no projeto é escolher a melhor rota para o inimigo chegar até o jogador. Com os algoritmos desenvolvidos, o inimigo sabe onde o jogador se encontra, analisa todas as rotas e, então, toma a decisão: “se eu for por este caminho vou demorar mais, ele é o mais longo. Este outro caminho é mais curto, porém, tem muitas árvores e não vou conseguir passar. Este outro tem uma casa na frente, se eu contornar a casa pela direita o caminho é mais curto, porém, como tem algumas pedras posso demorar mais do que o caminho anterior, o mais longo.”. É este tipo de decisão que o algoritmo proporciona. Segundo (RUSSELL e NORVIG, 2004), este tipo de algoritmo se chama Algoritmo de Busca com Informação e utiliza o conhecimento específico do problema. A abordagem geral é chamada de Busca Pela Melhor Escolha, ou *Best-First*. Busca A* significa minimizar o custo total estimado da solução. Pode-se representar o custo em uma busca como o gasto que o elemento tem ao deslocar-se de um nó ate o seu objetivo.

5 METODOLOGIA

As metodologias utilizadas para este projeto de desenvolvimento de software foram o U*nified Modeling Language* (UML) e o *Rational Unified Process* (RUP), pois, trata-se de um processos de especificação e implementação popular, com etapas claras e com vasto material de apoio. Segundo (MARTINS, 2004), o UML foi criado para modelar sistemas baseados em software e o gerenciamento do projeto é regulamentado pelo RUP.

Sua popularidade está baseada na versatilidade, e sua estrutura proporciona ao desenvolvedor uma visão clara da integração das várias facetas do processo de desenvolvimento. A premissa do RUP é que o software está baseado em componentes interconectados entre si via interface e a ferramenta utilizada para especificar estes componentes é o UML. Suas características fundamentais são: ser dirigidos por caso de uso, ser centrado na arquitetura, ser interativo e ser incremental. O RUP é um processo robusto, voltado para grandes empresas e grandes projetos comerciais; contudo, sua flexibilidade proporciona construir projetos de menor porte como para este estudo. Conforme (DEITEL, 2005), o RUP é concebido para projetar aplicativos comerciais de grande porte, como define o autor: aplicativos de "poder industrial", porém, é possível criar processos simplificados de projeto concebido por alunos de curso de programação. O RUP possui as seguintes fases: Abertura (ou Concepção, ou Iniciação), Elaboração, Construção e Transição.
As fases também são chamadas de marcos, conforme mostrado na Figura 1.

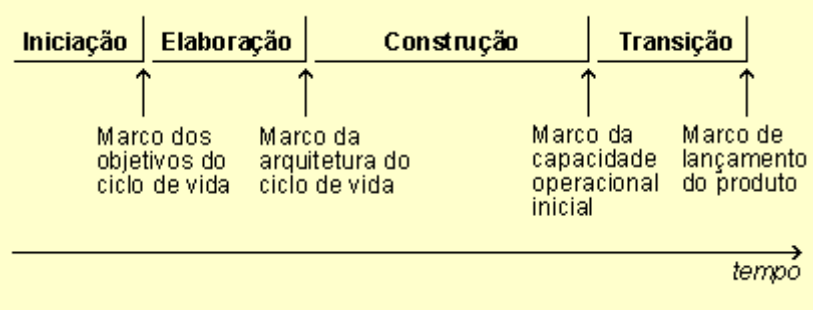


Figura 1 - Fases ou marcos em uma estrutura RUP.
Fonte: WTHREEX (2010).

A Tabela de Desenvolvimento serve para visualizar o projeto como um todo. Demonstra os itens das fases, o inicio e o desenvolvimento das disciplinas. A letra i significa

“inicio” e a letra r “refinamento”. Segundo (LARMAN, 2007) as disciplinas descrevem atividades de trabalho específicas. Estas atividades são chamadas de artefatos e pode ser definidos como qualquer produto do trabalho: código, gráficos, diagramas, texto, etc. Quadro 1.

Disciplina	Artefato	Iniciação/ Concepção	Elaboração	Construção	Transição
Modelagem de Negócio	Modelo de domínio		i	r	r
Requisitos	Modelo de caso de uso	i	r	r	r
	Visão	i	r	r	r
	Especificações suplementares	i	r	r	r
Projeto (Analise e Design)	Modelo de projeto		i	r	r
	Documento de Arquitetura de software		i	r	r
Desenvolvimento/ Implementação	Protótipos de tela	i	r	r	r
	Modelos de objetos		i	r	r
	Desenvolvimento dos componentes			i	r
	Integração dos componentes			i	r
	Modificações e acerto				i
Teste	Caso de teste		i	r	r
	Script de teste			i	
	Teste da versão final				i
Desdobramento ou Implantação	Conclusão do empacotamento do sistema				i
Codificação e Mudanças	Ferramentas	i	r	r	

Quadro 1 - Tabela de Desenvolvimento

6 DESENVOLVIMENTO DO PROJETO

Este capítulo relata as atividades do projeto. A seqüência de desenvolvimento descreve as disciplinas em função das fases da Tabela de Desenvolvimento.

6.1 Fase de Concepção

Nesta fase, deve-se deixar clara a visão do sistema do ponto de vista do negócio, concluir se o projeto é viável ou não. Segundo (MARTINS, 2004), o objetivo principal na fase de abertura do projeto é: especificar a visão do produto final, garantir que as funcionalidades estejam mapeadas e a descrição do comportamento do cenário, ou seja, analisar o caso de uso.

Ao se desenvolver um jogo o estudo de caso não se enquadra em um estudo tradicional. Conforme (LARMAN, 2007), o estudo de caso de um software para jogos, apesar dos objetos e a aplicação da UML serem iguais a outros projetos mais tradicionais como banco de dados os requisitos não são os mesmos.

6.1.1 Desenvolvimento da disciplina requisitos na fase de concepção.

Primeira disciplina a ser desenvolvida na fase de concepção apresentando os documentos Visão, Modelo de caso de uso e Especificações suplementares.

6.1.1.1 Visão

A visão serve para comunicar, de forma simples, as principais idéias do projeto. Conforme (WTHREEX, 2001), a visão é definida pelo entendimento dos envolvidos sobre o produto a ser desenvolvido em relação às necessidades e características mais importantes.

Serve de base contratual para requisitos técnicos mais detalhados, pois, trata das descrições dos requisitos.

Este projeto, ao ser desenvolvido, observou os atrativos de interação e história do jogo. Os atrativos segundo (SCHUYTEMA, 2008) são elencados como:

a) o jogador deve conhecer o objetivo logo no início e não ficar vagando pelo cenário;
b) as vitórias devem ser aninhadas, referindo-se ao jogador conquistar vitórias parciais ao decorrer do jogo, ou decifrar enigmas, até alcançar o objetivo maior e não descarregar toda sua adrenalina para desafios mais simples;
c) ter um contexto compreensível para o jogador satisfazer sua expectativa em relação ao jogo, geralmente estes jogos são familiares ao usuário e ele não deve ter dificuldade de entender o contexto, pois, prejudicaria a experiência vivida;
d) os ícones devem dispensar traduções. A parte de interface com o usuário considera as funções obvias de agir da interface;
e) a maioria dos jogadores é usuário Windows, e será interessante basear a representação visual neste estilo para criar uma atmosfera familiar ao jogador. Outra observação de interface está na sua coerência. Como o ser humano tem habilidade para tarefas repetitivas, ao descobrir como uma coisa funciona, ele tem a expectativa que esta funcione da mesma maneira nas próximas vezes. Por exemplo: se no jogo a tecla "Enter" executa a mesma tarefa que um clique do mouse, então, deve-se manter a coerência deste padrão durante todo o jogo;
f) para causar um impacto visual mais realístico as texturas são utilizadas com fotos ou desenhos;

6.1.1.2 Modelo de caso de uso

Os casos de usos são narrativas utilizadas para descobrir e registrar requisitos. Segundo (LARMAN, 2007), os casos de uso são narrativas em textos bem detalhadas. Seu objetivo principal é descobrir e registrar os requisitos funcionais, escrevendo narrativas de uso do sistema. Engloba o caso de uso, o ator e o cenário. O ator representa algo com comportamento, identificado pelo seu papel. O cenário representando uma seqüência específica de ações e interação entre ator e sistema. Resumindo, o caso de uso retrata uma

coleção de cenários relacionados de sucesso e fracasso descrito por um ator utilizando um sistema para atingir seu objetivo.

O caso de uso mais significativo no projeto é o “Jogar Selva” (visto na Figura 2 e Quadro 2). Como ele é uma simulação de computador e observado por apenas uma pessoa, pode-se concluir que o jogador é um observador. Conforme (DEITEL, 2005), o caso de uso modela a interação entre uma entidade externa, chamado de ator, ou melhor: “Observador”, e usando um sistema para atingir um objetivo. A tentativa de captar todas as regras do jogo em um formato de caso de uso pode deixar o diagrama confuso e não natural. Estas regras podem ser enquadradas como regras do negócio e serão enquadradas em outras especificações.

“Jogar Selva” foi o nome provisório do jogo.

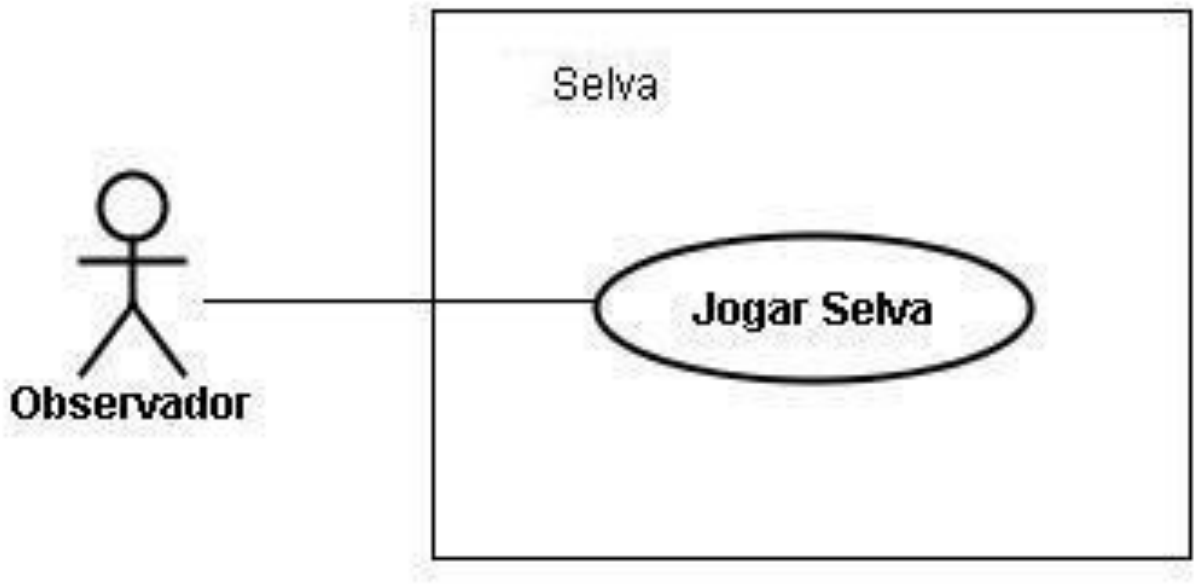


Figura 2 - Caso de uso mais significativo

O texto de caso de uso do estudo segue o modelo de Larman (2007). O autor adota um formato de gabarito de caso de uso desenvolvido por Alistair Cockburn, e suas seções são descritas como:

Caso de uso: Descreve a ação, geralmente começa com um verbo.

Escopo: Define o sistema em projeto.

Nível: “Objetivo do usuário”.

Interessados e interesses: Quem interessa o caso de uso e seu objetivo.

Principal cenário de sucesso: Um caminho típico, incondicional e otimista do cenário de sucesso.

Requisitos especiais: São requisitos não funcionais relacionados.

Escopo: aplicação Jogo Selva. **Nível:** objetivo do usuário. **Interessados e Interesses:** Jogador, também chamado de Observador. Deseja observar as saídas da simulação do jogo. **Principal cenário de sucesso:** 1 Observador solicita um jogo novo. 2 Observador começa a jogar, caminhando e atirando no cenário. 3 O sistema ativa ação conforme o tipo de interação do jogador, tanto do observador como do inimigo. As ações serão definidas posteriormente. Repetir passo três até encontrar vencedor ou Observador cancelar. **Requisitos Especiais:** Disponibilizar modos de rastreamento gráfico e textura.

Quadro 2 - Tabela de caso de uso

6.1.1.3 Especificações suplementares

Especificação suplementar significa centralizar informações sobre regras especificas de uma aplicação. Segundo (LARMAN, 2007), a especificação suplementar trata de requisitos, informações e restrições que não ficam facilmente documentados no caso de uso. Especificações não funcionais:

a) A máquina virtual Python é essencial para o funcionamento do jogo, porém, deverá ser transparente para o usuário ao instalar o jogo.

b) O idioma nos menus será escritos em inglês, para maior abrangência de usuário.

6.1.2 Desenvolvimento da disciplina implementação

Segunda disciplina a ser desenvolvida na fase de concepção.

6.1.2.1 Criação de protótipos de tela

A visualização da interface do jogo utilizando desenhos das telas principais deixa claro o que é esperado do software.

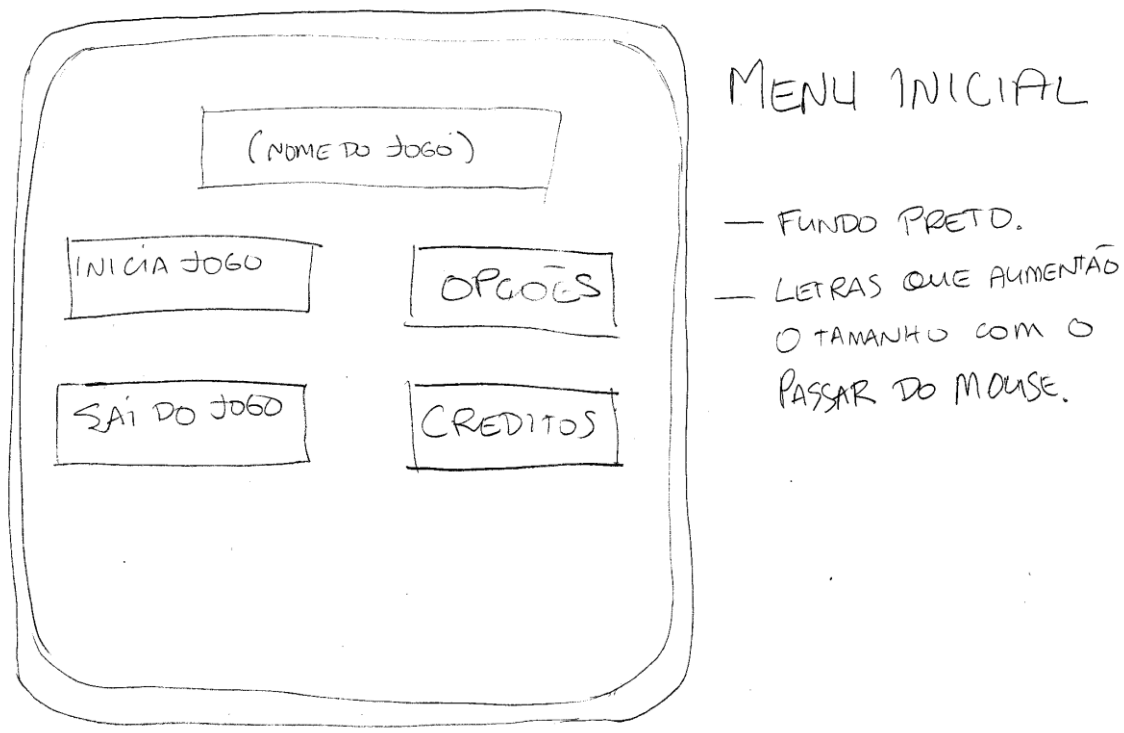


Figura 3 - Desenhos do menu.

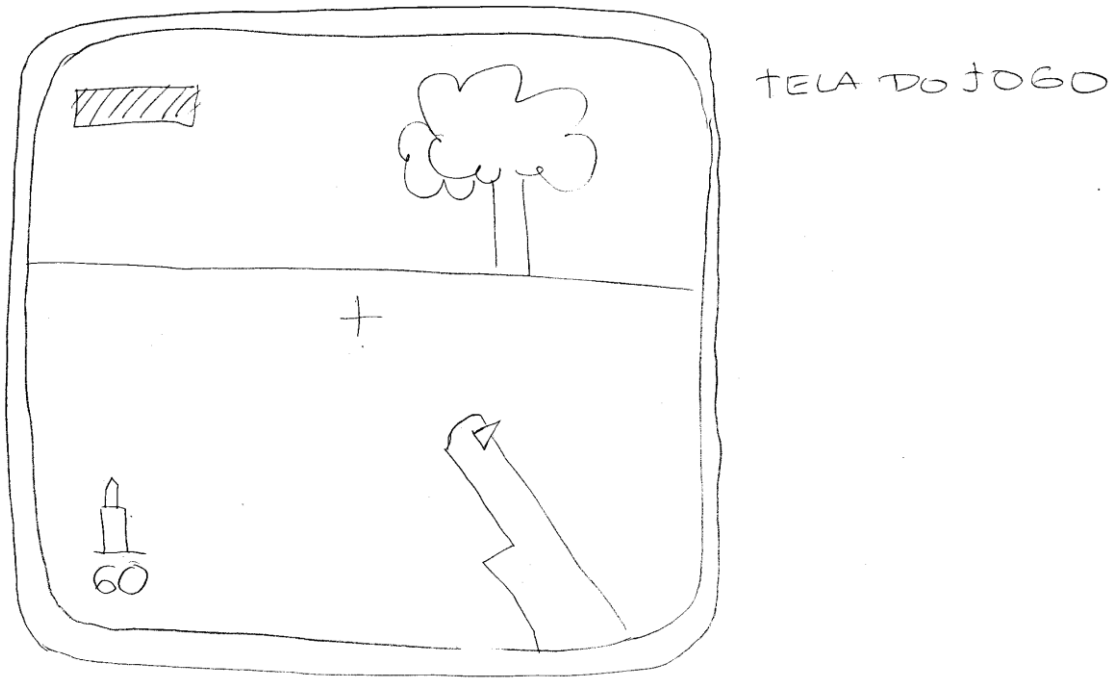


Figura 4 - Tela do jogo

6.1.3 Desenvolvimento da codificação e mudanças

Nesta fase foi feito o levantamento dos requisitos para estruturar o projeto.

6.1.3.1 Ferramentas utilizadas

a) Python 2.5.4.
b) Gimp 2.6.7.
c) Blender versão 2.48a.
d) Arbaro 1.8.9.
e) Jude Comunity 5.5 (Model Version: 30).

6.2 Fase elaboração

Nesta fase, além de elencar os requisitos faltantes é necessário definir a arquitetura e monitorar os riscos para as próximas fases. Segundo (MARTINS, 2004), a elaboração da fase é um processo de engenharia, primeiro especificando a arquitetura e depois planejando o software para o restante do projeto. Seu objetivo principal é: ter uma visão completa do sistema, como escopo, requisitos, funcionalidades, etc. Deve também analisar a situação e desenvolver uma arquitetura conforme os requisitos do sistema e ter uma visão do produto definida. Se necessário mudar a visão do produto ela deve acontecer com facilidade. Eliminar os riscos mais significativos e estar com a arquitetura, os requisitos e os planos sólidos ao final da fase é imprescindível para alcançar os objetivos.

Como o software é uma simulação, as ações do jogador podem ser enquadradas nas regras de domínio descritas ao decorrer da metodologia. Segundo (LARMAN, 2007), em uma linguagem orientada a objeto uma regra de domínio é uma representação visual de classes conceituais, ou objetos do mundo real. O autor descreve vários diagramas UML:

a) Diagrama de seqüência para descrever como os atores externos interagem com o sistema;

b) Arquitetura lógica para organizar as classes de software em pacotes, subsistemas e camadas;
c) Diagrama de comunicação para facilita a interação entre objetos em forma de grafo ou rede;
d) Diagrama de interação para generalização dos tipos seqüência e comunicação;
e) Diagrama de classes para ilustrar classes, interfaces e suas associações. Para uma modelagem estática de objeto;
f) Diagrama de atividades para mostra uma seqüência de ação que pode ser em paralelo;

6.2.1 Disciplina modelagem do negócio

Primeira disciplina na fase de elaboração.

6.2.1.1 Desenvolvimento da regra de domínio

O modelo de domínio ilustra conceitos em um domínio. Segundo (LARMAN, 2007), modelo de domínio é uma representação visual do mundo real. Os objetos deste mundo podem ser chamados de classes conceituais do mundo real. Deve ser observado que não são classes de software.

Lista informal de requisitos para elaboração do modelo de domínio do jogo proposto:

a) Apenas um jogador pode jogar;
b) Um jogo inicia quando o jogado assume o personagem principal e ele está no cenário com liberdade para se locomover e atira;
c) A entrada do usuário consiste nos disparos de sua arma e comandos para o deslocamento no jogo;
d) A arma possui dois cartuchos com 30 cápsulas;
e) O objetivo do inimigo é procurar e eliminar o jogador;
f) O jogo termina quando o jogado ou inimigo é atingido por alguns disparos.

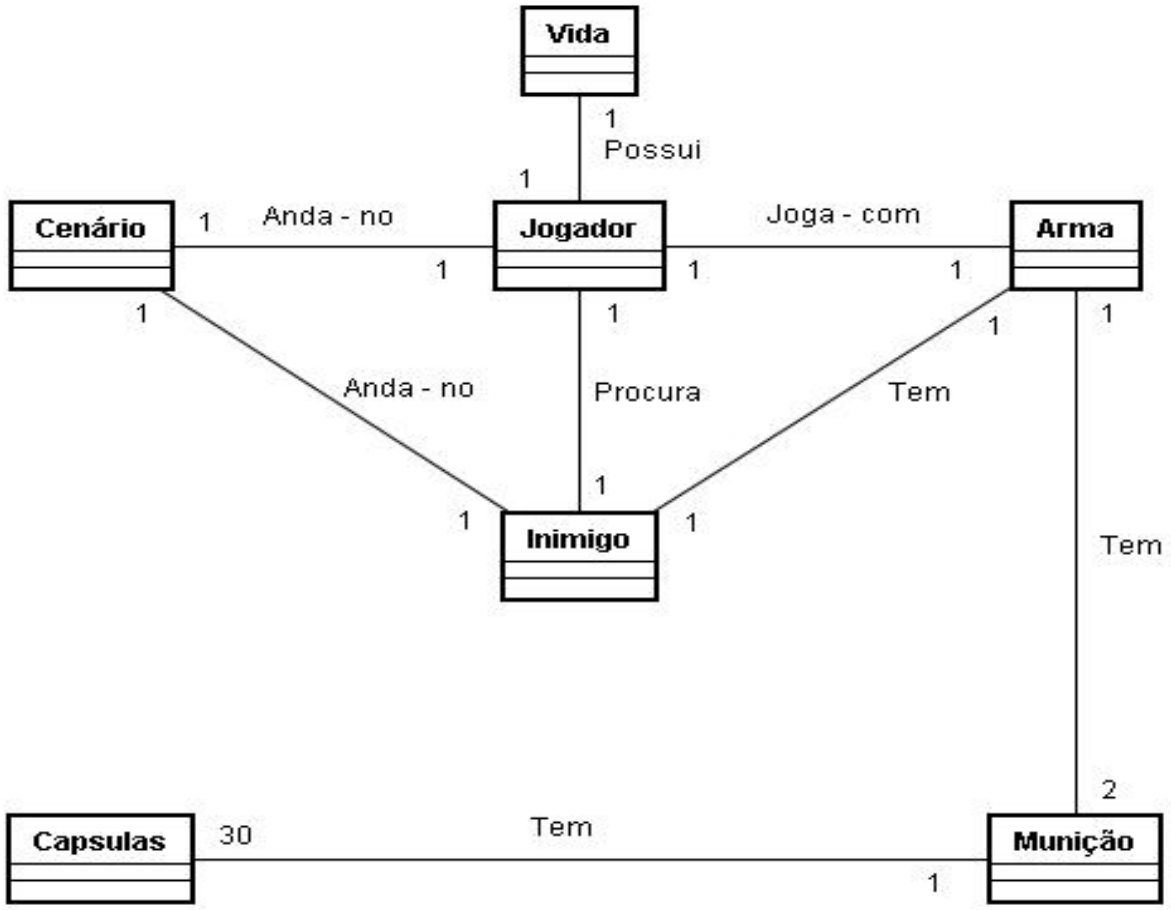


Figura 5 - Modelo de Domínio

6.2.2. Disciplina requisitos (continuação da fase iniciação)

Geralmente alguns artefatos da disciplina requisitos são inicializados na fase de Iniciação, outros apenas na fase de Elaboração. Os artefatos inicializados na Iniciação que não sofreram alteração não serão descritos nesta fase. Continuação dos casos de uso, Quadro 3.

6.2.2.1 Desenvolvimento do modelo de caso de uso

Caso de uso referenciando o jogador e o jogador inimigo. Descrição simples do cenário do jogo.

Ator: Jogador **Processo Escolher Menu:** Ao iniciar o jogo, o jogador (observador) encontra o menu principal com as funções de iniciar o jogo, verificar créditos ou então sair do jogo. Optando por iniciar o jogo a próxima tela o observador estará dentro do jogo interagindo com o cenário. Pressionando uma tecla pré-determinada ele volta ao menu principal. Optando por ver créditos é visualizado dado sobre o autor e o software, com uma tecla pré-definida volta ao menu principal. Optando por sair, o jogo é encerrado.
Ator: Jogador Inimigo (sistema) **Processo Jogar:** Durante a partida o inimigo sempre caminha pelo cenário e verifica se o jogador está em sua frente; se estiver, ele dispara a arma. O inimigo caminha para a posição que o jogador se encontrava no inicio do jogo. Caso ocorra um disparo da arma do jogador o inimigo caminha para esta posição. Se o jogador não for encontrado nesta última posição o inimigo se desloca aleatoriamente pelo cenário procurando ou aguardando um tiro do jogador. Se o inimigo estiver a uma distância mínima do jogador automaticamente ele efetuará a mira e dispara tiros. Saindo desta distância mínima o inimigo volta a seguir o seu caminho, independente da localização do jogador. Se alguns tiro do inimigo acertar o jogador o jogo termina.
Ator: Jogador **Processo Jogar:** Durante a partida o jogador pode se deslocar para qualquer lugar do cenário. Ele não consegue atravessar árvores ou paredes. Os disparos da arma ocorrem um por vez. Ao acertar alguns tiros no inimigo ele morre e o jogo termina.

Quadro 3 - Alguns casos de uso: Processo Escolher menu e Jogar

6.2.3 Disciplina projeto

Seqüência das disciplinas na fase de elaboração.

6.2.3.1 Desenvolvimento do documento modelo de projeto

O modelo de projeto é utilizado como base para as atividades de implementação e testes, podendo ser visto como uma abstração da implementação do sistema, sendo utilizado para conceber e documentar o projeto do sistema de software, Figura 6. É um artefato composto e abrangente que envolve todas as classes de projeto, subsistemas, pacotes, colaborações e os relacionamentos entre eles. O modelo de projeto define principalmente a arquitetura, mas também é usado como um veículo para a análise durante a fase de elaboração. Em seguida, ele é refinado por decisões de projeto detalhadas durante a fase de construção. Figura 7 e 8.

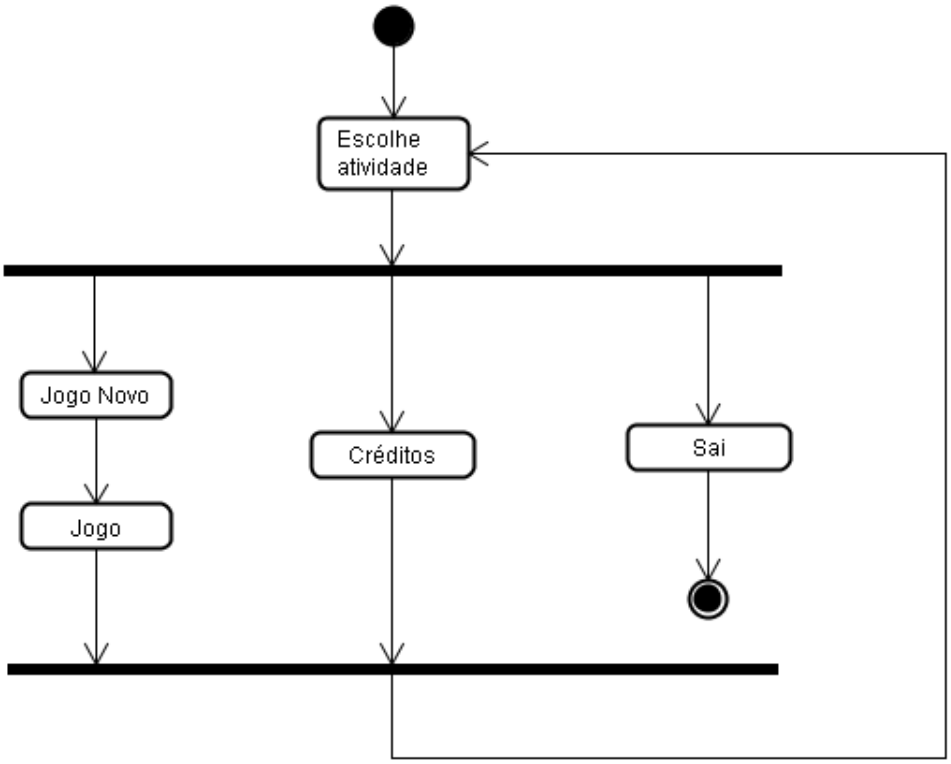


Figura 6 - Fluxograma do menu

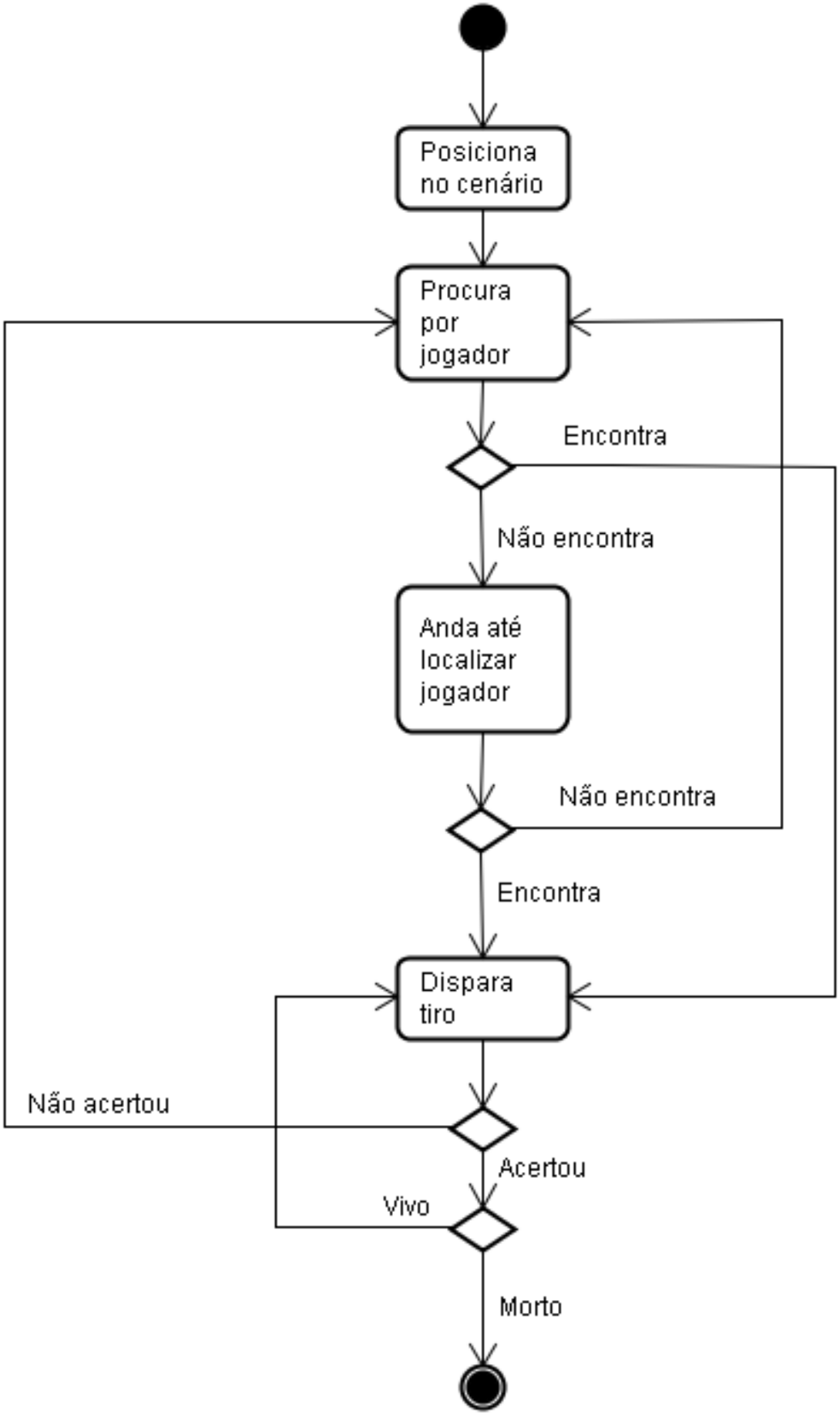


Figura 7 - Fluxograma do inimigo

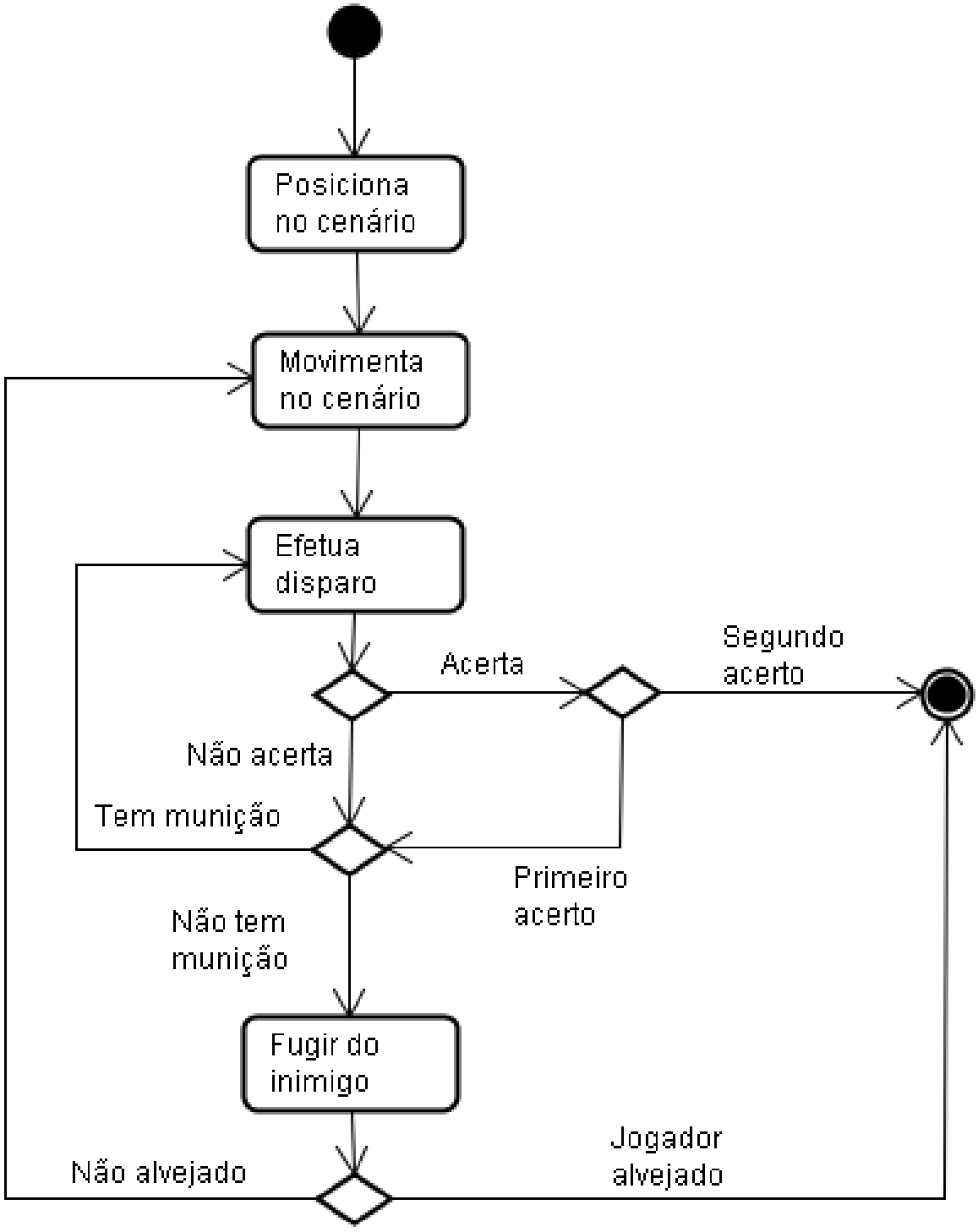


Figura 8 - Fluxograma do jogador

6.2.3.2 Desenvolvimento da arquitetura de software na fase de elaboração.

Nesta fase do projeto será definida a visão lógica do sistema. Os documentos gerados foram a descrição da *Game Engine*, da inteligência artificial, do algoritmo A* e da hierarquia dos objetos e funções.

a) Descrição da hierarquia da *Game Engine*:

A programação foi estruturada entre o aplicativo Blender e a linguagem de programação *Python*. Entre os dois existe a API Blender para ligar as duas estruturas como um ponto em comum. Os códigos *Python* são chamados de *script's* e são usados em tempo de execução, ou seja, à medida que o software é utilizado o código é acionado. No próprio aplicativo Blender existe a *Game Engine* responsável pela execução do jogo.

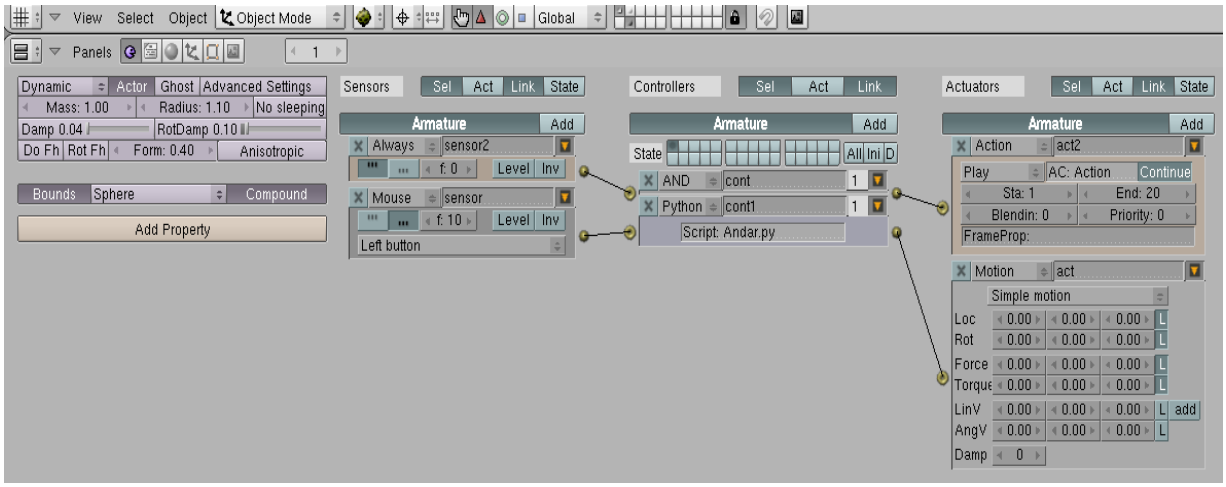


Figura 9 - Painel "*Logic*" no aplicativo Blender

Os objetos modelados no aplicativo Blender, como personagens, árvores e chão, serão controlados durante o jogo pela especificação do *script Python* criado pelo programador e ativados através da *Game Engine* do aplicativo Blender. A Figura 9 demonstra a ligação de dois *Sensors "Armature"*, dois *Controllers "Armature"* e dois *Actuadors "Armadure"*. *Armadure* é o nome dado ao objeto responsável pelo movimento de braços e pernas do inimigo no jogo. O resultado destas ligações é a movimentação do inimigo em direção ao jogador como resposta ao acionamento do botão esquerdo do mouse. Cada sensor "*Sensors*" é chamado de bloco lógico "*Logic Brick*" e cada bloco recebe um nome. O primeiro se chama

“sensor2” e sempre “*Always*” está ativado para executar uma ação pré-determinada. Este sensor está ligado a um bloco lógico controlador “*Controllers*”, chamado de “cont” e está ativado na função lógica *AND* que avalia se o bloco sensor está ativado. Se ativado (verdadeiro) o controlador executa o atuador “*Actuadors*” correspondente. O atuador se chama “act2” e executa uma ação “*Action*” que significa pegar uma animação pronta do objeto armadura e executá-la. No bloco act2 está definida uma animação chamada de “*Action*” e será executada do quadro um até o vinte.

Uma animação no blender é como um desenho animado, o conjunto de vários desenhos passados em um determinado espaço de tempo cria a impressão do movimento. Geralmente para um desenho animado são necessários no mínimo de vinte e quatro quadros ou desenhos em seqüencia para criar uma animação de um segundo. No Blender esta condição também deve ser considerada, porém, ela ocorre de forma automática. Os objetos animados no aplicativo devem estar registrados em uma posição inicial e também deve ser colocado na posição final para ser registrado onde eles devem parar. Para chegar da posição inicial até a final o próprio aplicativo preenche os espaços. Por exemplo: para animar o deslocamento de uma esfera é necessário marcar a posição inicial do objeto e em seguida marca a posição onde ela deve parar. Em seguida é executada a animação e o Blender, em uma seqüência de quadros predefinida, completa a animação do início ao fim. Em cada quadro o aplicativo apaga o objeto do quadro anterior e recria no quadro seguinte, em seqüência, e tem-se a impressão do movimento.

O segundo sensor chamado de “sensor” é ativado através do mouse, especificamente quando o botão esquerdo é acionado. O segundo quadrado com os três pontinhos pressionados significa que a cada dez ciclos do relógio do computador será verificado se o botão esquerdo do mouse foi acionado. A especificação do ciclo está descrita em “f10”, Figura 9. O sensor está ligado a um controlador chamado “cont1” e está configurado para executar um código *Python*, O código está em um arquivo chamado de “Andar.py” que define como o objeto *armature* deve se comportar em relação ao andar. O controlador “act” define os tipos de movimento do *armature*, como deslocamento e rotação representados por “*loc*”, “*rot*”, “*force*”, “torque”, etc. A vantagem de utilizar o código *Python* está na possibilidade de usar os dados do bloco lógico controlador dentro do código, potencializando sua utilização através de consultas e atribuição de valores. Os dados impostos direto no bloco controlador são estáticos e não manipuláveis.

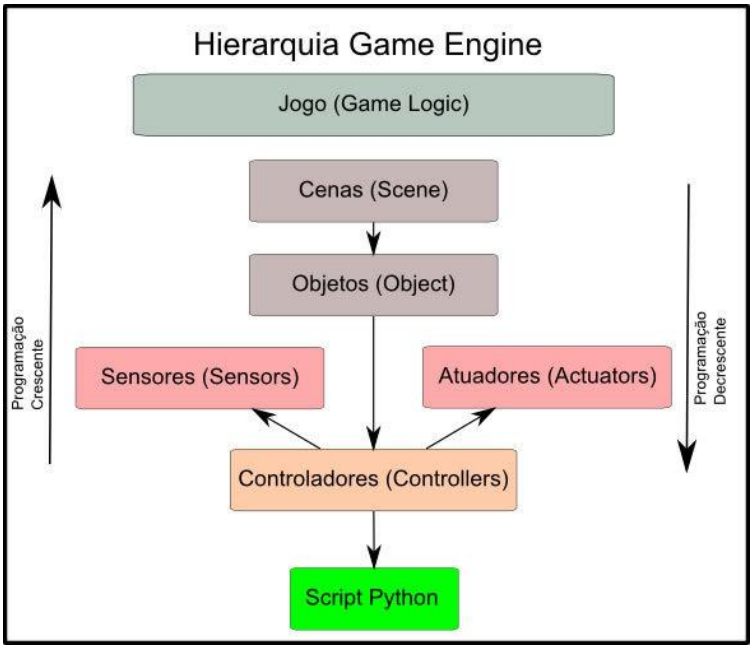


Figura 10 - Hierarquia da *Game Engine*.
Fonte: PROCEDURALBASE (2007).

A hierarquia dentro da *Game Engine* no jogo desenvolvido é crescente, Figura 10. Sua base é o *scipt Python* que está disponibilizado através do controlador. O controlador recebe informações dos sensores através de vários componentes como teclado e mouse e repassa para o código *Python*. O controlador também avalia as funções disponíveis no atuador, como "*act* e *rot*", (ver Figura 9), para ser processada no *script python*. Com estas duas informações o controlador avalia o código *Python* e repassa o resultado para o objeto ser executado. A cena é formada por todos os objetos no jogo. Para chegar ao atuador a estrutura do código será: *Script* – controlador – atuador.

b) Descrição da inteligência artificial:

Para o inimigo ter a capacidade de procurar e encontrar o jogador durante a partida foi adotado o método baseado no algoritmo de busca A*.

Este algoritmo pode ser representado por uma estrutura de dados árvore, chamada de árvore de busca. Consiste em compara nós no algoritmo como se fossem pontos de ramificações em uma árvore real. Quando um nó, ou ramificação, é selecionado no algoritmo para ser comparado com outro nó, ou ramificação da mesma árvore, a busca ocorre nos nós a frente sempre dando preferência para o que proporcionar a menor distância *g(n)*. Outra característica deste algoritmo é a **função heurística** *h(n)*, que representa o custo estimado em linha reta para ir de um nó até o outro nó da busca. Ela pode ser representada como a distância em linha reta entre o nó até o objetivo, independente se existe outros nós no caminho. A

escolha do melhor caminho é feita com base em uma **função de avaliação** *f(n)*, que escolhe o nó com valor mais baixo, no caso da árvore a ramificação com o nó mais próximo e o valor menor para chegar até o objetivo. Neste algoritmo a fórmula *f(n) = g(n) + h(n)* representa **o custo estimado da solução de custo mais baixo passando por *n* = custo para alcançar cada nó + custo para ir do nó até o objetivo**. Para que se encontre a solução de custo mais baixo será verificar o nó com valor mais baixo *g(n) + h(n)*. Segundo (RUSSELL e NORVIG, 2004), a estratégia é efetuar uma busca com informações especificas do problema e definir o próprio problema. O custo para alcançar cada nó é *g(n)* e pode ser exemplificada como a distância entre as cidades da Romênia, Figura 11. O custo para chegar até ao objetivo é *h(n)* e pode ser representada pela distância em linha reta de cada cidade até a cidade de Bucareste, Tabela 4. A estimativa da solução de custo mais baixo passando por *n é f(n)*, através da avaliação *g(n) + h(n)*, Figura 12 e a observação mais importante será sobre a heurística admissível, onde, *h(n)* nunca superestime o custo para alcançar o objetivo, ou seja, o custo para alcançar o nó não pode ser maior para alcançar o seu objetivo. A heurística admissível sempre considera que o custo da resolução do problema seja menor do que ele é na realidade. O custo *g(n)* é o custo exato para se alcançar *n*, tem-se como conseqüência imediata que *f(n)* nunca irá superestimar o custo verdadeiro de uma solução passando por *n*.

Pode-se considerar um mapa rodoviário para sair de Hrand e chegar a Bucareste usando-se uma BUSCA-EM-ÁRVORE.

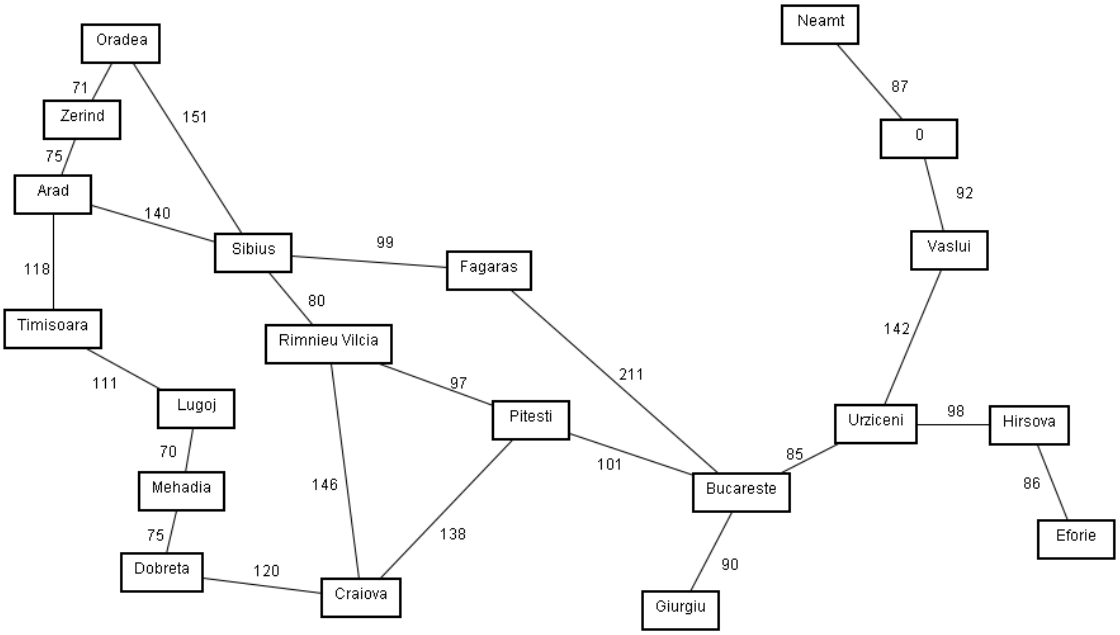


Figura 11 - Mapa rodoviário da Romênia. Distância em linha reta de uma cidade a outra *g(n)*
Fonte: Adaptado de Russell e Peter (2004, p. 65).

Arad	366	Mehadia	241
Bucareste	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitestia	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Tabela 4 - Distância em linha reta até Bucareste
Fonte: Adaptado de Russell e Peter (2004, p. 96).

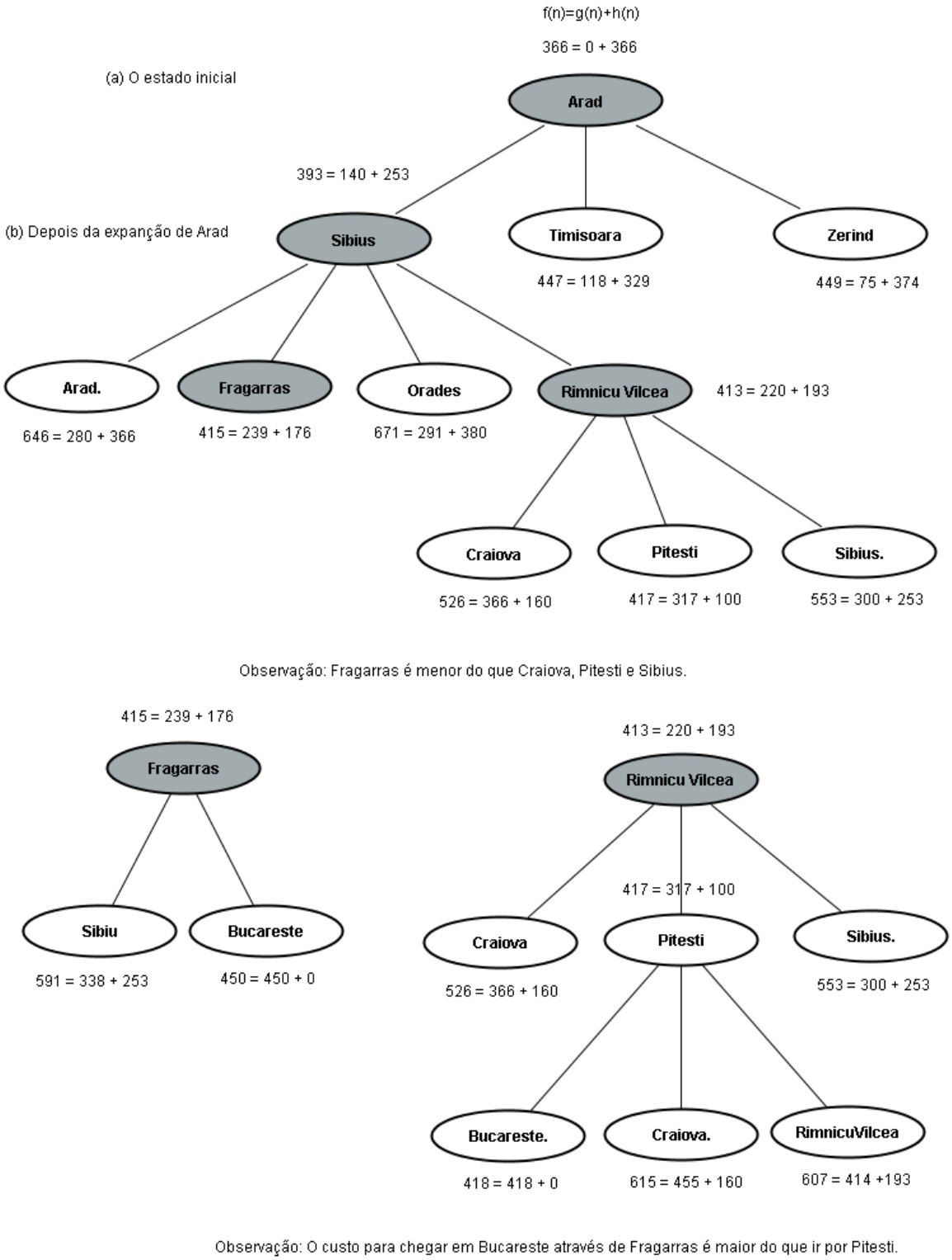


Figura 12 - Deslocamento de Arad até Bucareste utilizando árvore para escolher o menor caminho.
Fonte: Adaptado de Russell e Peter (2004, p. 98).

Pode-se observar que na cidade de Fagaras há um caminho direto para Bucareste, porém, seu custo está em 450, sendo mais alto que Pitesti, 418. Este é um efeito da heurística admissível.

c) Descrição do algoritmo A*

A área onde ocorrerá a busca deve ser divida por linhas e colunas, transformando o local em uma grade quadrada. Cada espaço quadrado representa uma posição. Portanto, se um objeto em um quadrado necessita deslocar-se até outro quadrado ele deve escolher os quadrados que proporcionem o caminho mais curto. Caso haja no caminho do objeto algum quadrado que ele não possa acessar ele deve contorná-lo para chegar ao seu destino. A Figura 13 demonstra esta situação: o quadrado verde deve chegar até o quadrado azul e os quadrados pintados de preto é a parede entre eles.

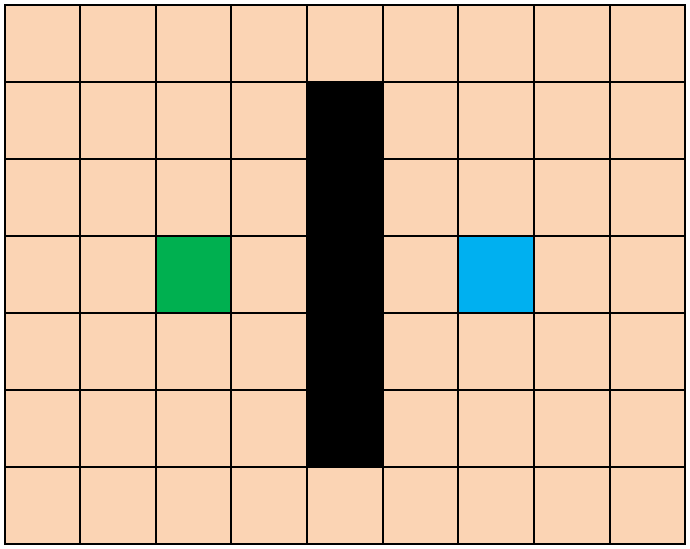


Figura 13 - Mapa de deslocamento, vista superior.
Fonte: Adaptado de LESTER (2005).

Passo 1:

O quadrado verde para começar sua caminhada deve selecionar todos os quadrados em sua periferia, excluindo quadrado que contenha paredes, Figura 14. Os quadrados selecionados representam possíveis caminhos e devem ser guardados na memória. Outro dado que deve ser guardar em memória é a posição anterior e atual do quadrado verde. Segundo (LESTER, 2005), é criada uma “lista aberta” para verificar os possíveis caminhos e outra

“lista fechada” para e excluir a posição do quadrado verde das pesquisas futuras. Nesta situação o quadrado verde será o quadrado pai e todos os amarelos convergem para ele.

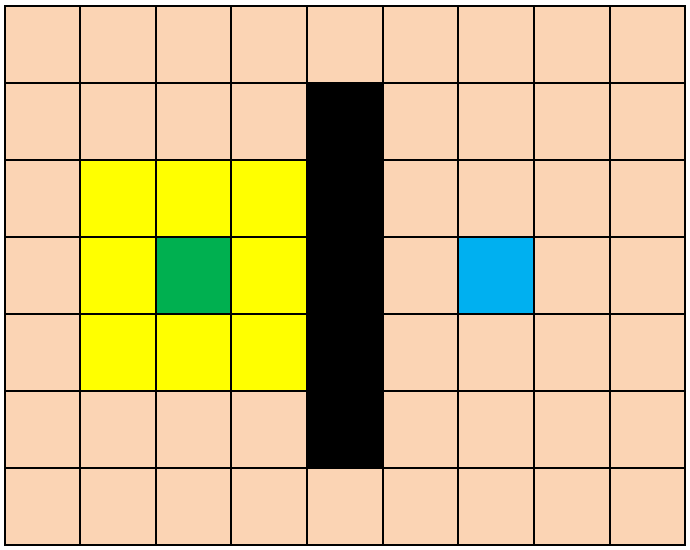


Figura 14 - Mapa de deslocamento, pesquisa de caminhos válidos
Fonte: Adaptado de LESTER (2005).

Passo 2:

Para verificar a melhor posição cada quadrado amarelo deverá conter o custo de deslocamento do verde para o amarelo *g(n)*, o custo de cada amarelo até o azul *h(n)* e o custo estimado da solução *f(n)*, figura 15.

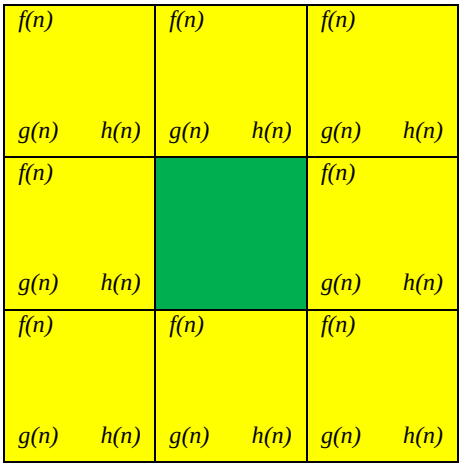


Figura 15 - Definindo variáveis para deslocamento
Fonte: Adaptado de LESTER (2005).

Os valores de deslocamento entre verde e amarelo tem o custo na vertical e horizontal de 1, para deslocamentos em diagonal o custo será 1.4. Para calcular a distância de cada quadrado amarelo até o azul será adotado o método de Manhathan. Segundo (LESTER, 2005), este método consiste em contar os quadrados apenas na vertical e horizontal entre o amarelo desejado e o azul, Figura 16. Sua adoção não é a melhor porque superestima a distância restante, porém, será adotado na explicação do algoritmo.

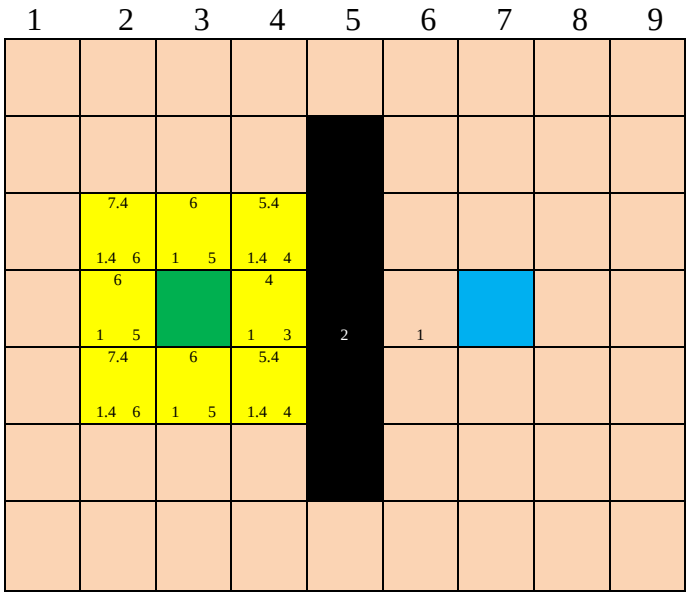


Figura 16 - Verificação das distâncias entre verde e azul
Fonte: Adaptado de LESTER (2005).

Passo 3:

A verificação do melhor local leva em consideração a soma *f(n) = g(n) + h(n)* dos quadrados amarelos (lista aberta). É eleito o menor *f(n)* e desconsidera-se a lista fechada com seus elementos.

No exemplo a menor distância é *f(n)* = 4, Figura 17.

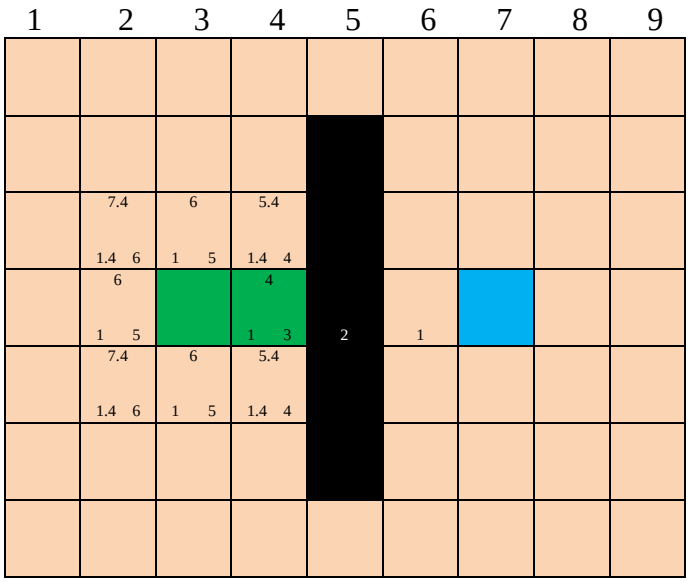


Figura 17 - Primeiro deslocamento
Fonte: Adaptado de LESTER (2005).

Passo 4:

Nesta etapa *f(n)*=4 é colocado na lista fechada e é eleita uma nova lista amarela tendo em vista o quadrado verde eleito como base. Os elementos da lista fechada e quadrados representando paredes devem ser desconsiderados. No exemplo, o caminho mais próximo do quadrado azul são os dois quadrados com a função: *f(n)* = 5.4 (acima do último quadrado verde eleito) e *f(n)*=5.4 (abaixo do último quadrado verde eleito). Neste caso, a escolha do quadrado depende da construção do código-fonte, pode ser o primeiro a entra na lista aberta ou o último. Para o desenvolvimento do algoritmo a consulta dos quadrados amarelos se dá de cima para baixo e da esquerda para a direita. Caso haja alguma função *f(n)* igual será

considerado a última encontrado. Portanto o quadrado eleito é *f(n)* = 5.4 abaixo do último quadrado verde eleito, Figura 18.

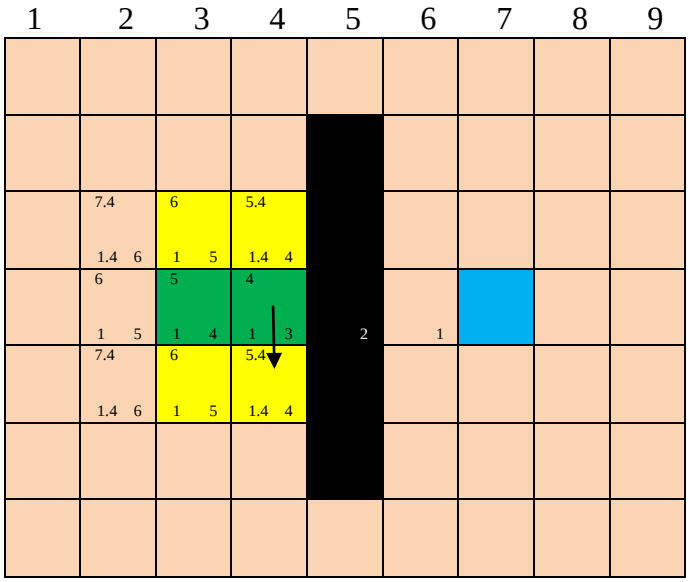


Figura 18 - Escolha do melhor quadrado
Fonte: Adaptado de LESTER (2005).

Passo 5:

Considerando a pesquisa no quadrado *f(n)* = 5.4, Figura 19, o melhor caminho encontrado foi *f(n)* = 6, último quadrado com valor igual encontrado na pesquisa da na lista aberta.

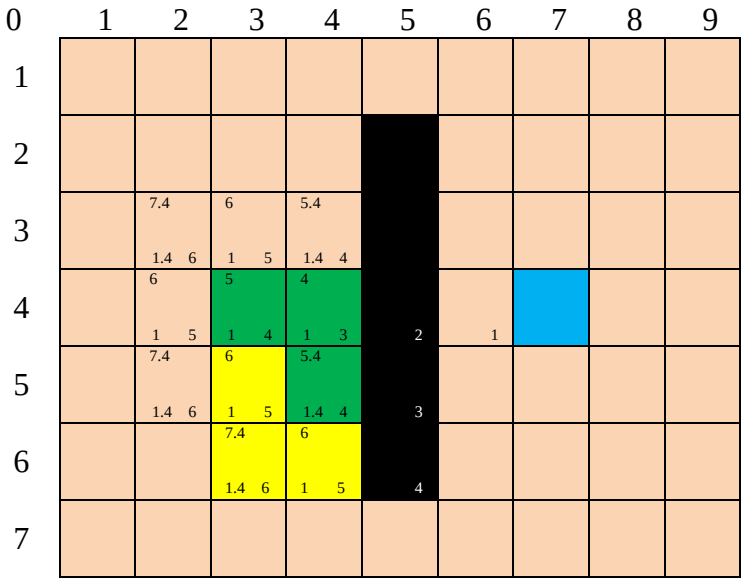


Figura 19 - Definindo quadrado.
Fonte: Adaptado de LESTER (2005).

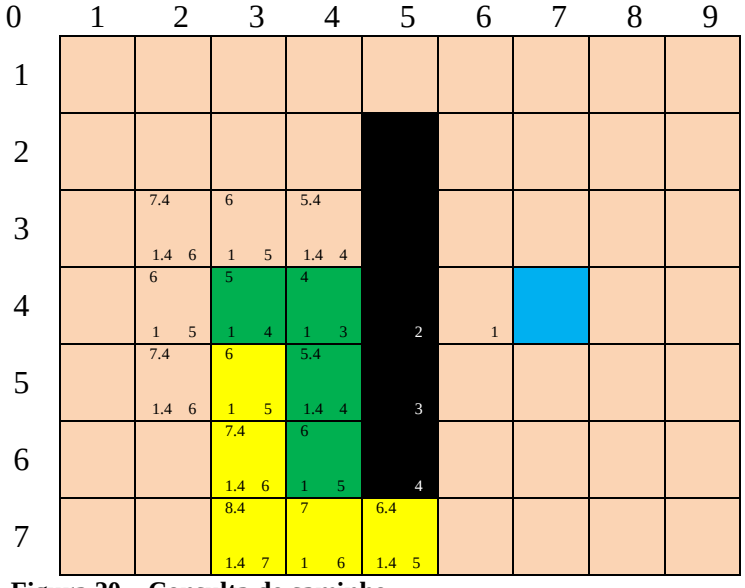


Figura 20 – Consulta de caminho.
Fonte: Adaptado de LESTER (2005).

Pode-se observar o revés na marcha para chegar ao quadrado azul, pois, o menor custo para chegar ao quadrado azul não é a função *f(n) = 6.4*, na coluna 5, linha 7, Figura 20, Tecnicamente o menor custo é a função *f(n) = 6*, na coluna 3, linha 5. A lista fechada será consultada e o algoritmo bloqueará qualquer deslocamento para uma função *f(n)* com o mesmo valor existente nesta lista.

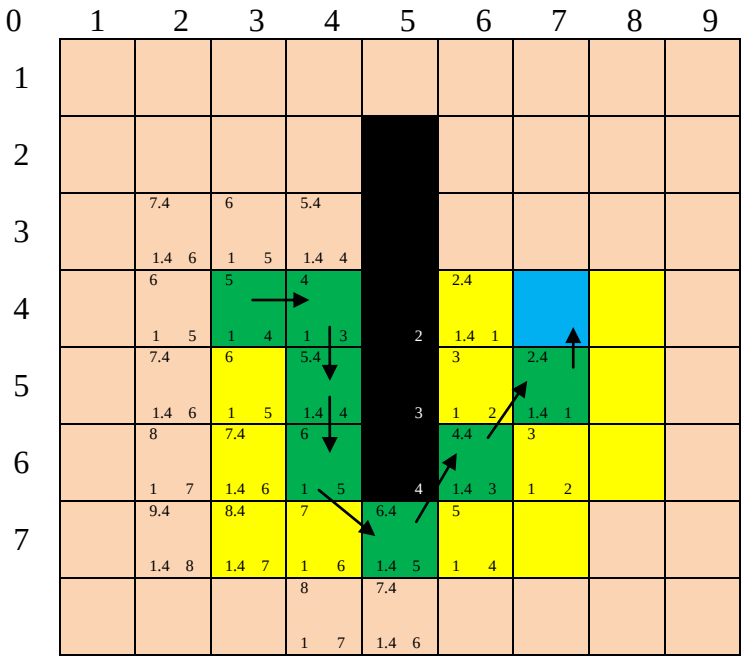


Figura 21 - Caminho definido.
Fonte: Adaptado de LESTER (2005).

As consultas seguintes serão repetições dos passos um a cinco, Figura 21.

O caminho ideal está demonstrado pelas setas. Foi criada uma lista fechada com sete posições: [(3,4) (4,4) (4,5) (4,6) (5,7) (6,6) (7,5)] que será o caminho para chegar ao quadrado azul.

d) Descrição da hierarquia dos objetos:

A compreensão da atuação dos objetos está relacionada com qual outro objeto ele mantém uma relação de dependência ou imposição. A hierarquia está definindo qual objeto pode ser influenciado. Por exemplo: Todos os outros objetos dependem do objeto Selva. O objeto Inimigo é influenciado pelo objeto Selva, porém, o objeto Corpo, Ação e Pose estão todos relacionados com Inimigo.

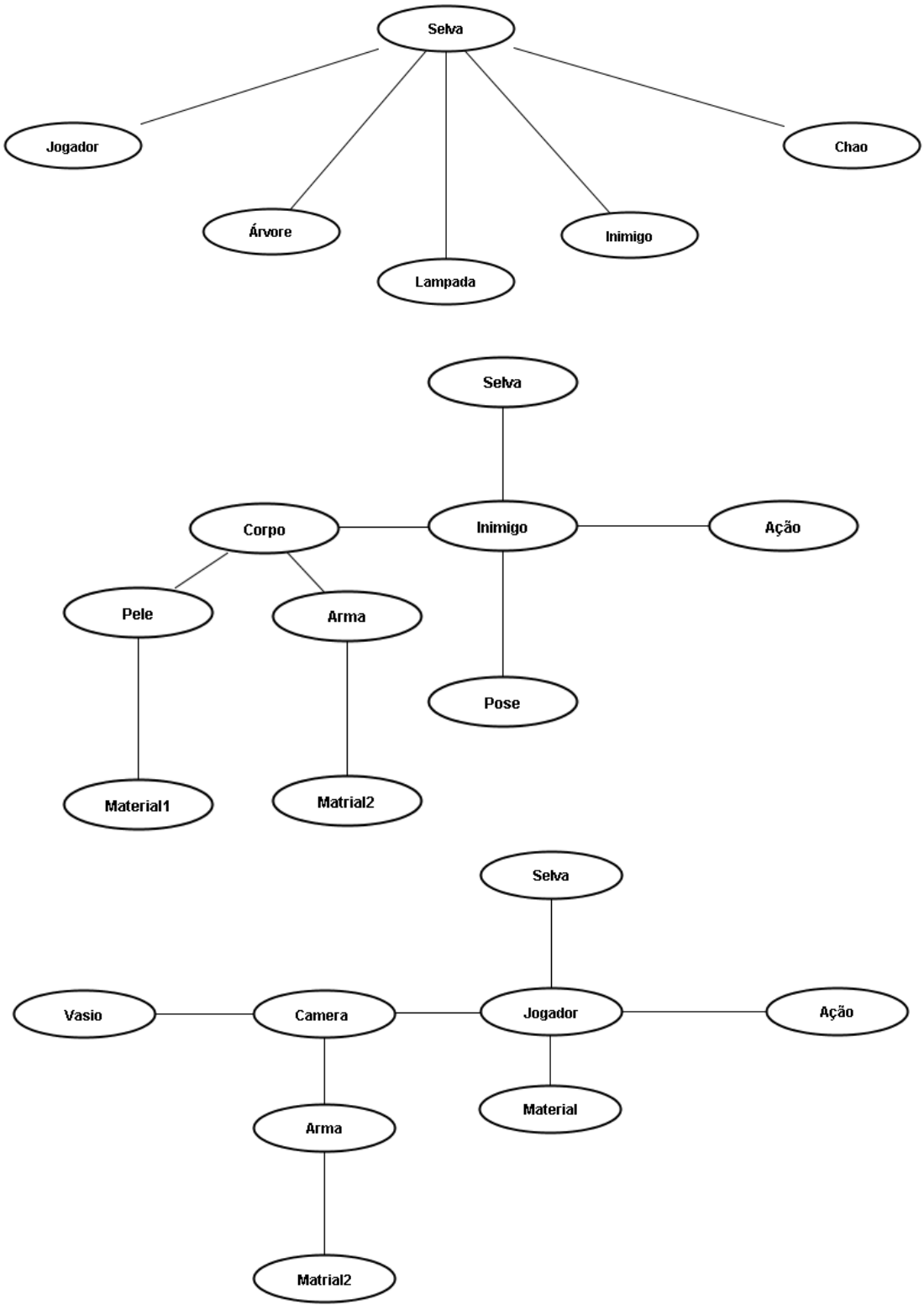


Figura 22 - Hierarquia dos objetos. De cima para baixo. A palavra material refere-se ao tipo de textura e cor do objeto.

6.2.4 Descrição da disciplina implementação

Desenvolvimento da interface com o usuário na fase de elaboração.

6.2.4.1 Estudo de modelos de objetos

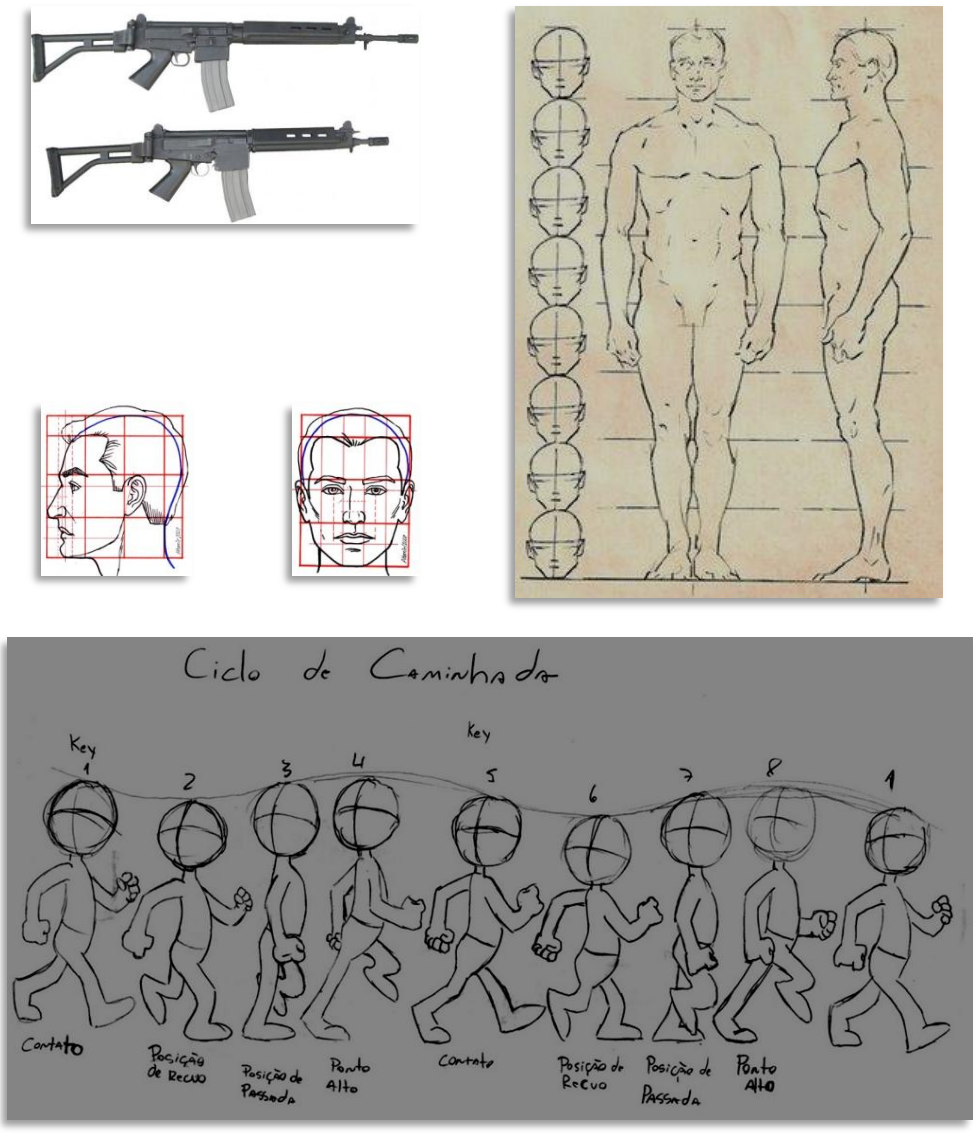


Figura 23 - Modelos para construção dos objetos.
Fonte: DESENHEPINTE (2008), MOREIRA (2008), SOUSMAN (2008), TERRESTRE(2009)

A Figura 23 serve como modelo para arma do jogador e inimigo. O desenho do corpo humano será usado como modelo para corpo do inimigo. As figuras com o rosto serão utilizadas para modelar a face do inimigo. O último desenho estuda o movimento do andar.

6.2.5 Descrição da disciplina teste na fase de elaboração

Os testes são parte fundamental no sucesso do projeto. Segundo (MARTINS, 2007), a correção de problemas de software é de 100 a 1000 vezes mais cara de realizar após a implantação do sistema. Conforme o autor, os testes podem ser implantados para os cenários mais relevantes que representam o comportamento esperado do sistema. O objetivo é testar e avaliar a qualidade do produto, corrigindo problemas e má interpretação dos requisitos.

A seqüência de teste em um projeto pode ser: Testes de unidade: os menores elementos do projeto. Testes integração: entre componentes e subsistemas. Teste de sistema: todo o sistema é testado e Teste de aceitação: testado pelo usuário.

Os tipos de teste são o Teste de *benchmark*: compara o desempenho do elemento testado contra os requisitos e o Teste de funcionamento: verifica se o elemento testado funciona corretamente.

O modelo de testes indica os elementos que vão ser testados e o modo como serão testados. Os utilizados neste projeto serão os de Caso de Teste: verificam os objetivos, dados dos testes, condições de execução e resultado esperado. *Scripts* de Teste: serão utilizados para testar instruções de programas para execução automatizada de um procedimento de teste ou parte dele.

6.2.5.1 Desenvolvimento de casos de testes

Caso de teste: CT1
Cenário/Condição: Distancia em linha reta entre inimigo e jogador com obstáculo.
Entradas: • Parede em forma de “U”. • Parede muito comprida. • Jogador posicionado dentro de um quadrado representado por uma parede.
Resultado: • Em todos estes casos o inimigo anda de forma desorientada. • Como ele não deixa sobrescrever *f(n)* com o mesmo valor. A função para o inimigo sair de uma parede em “U” seria passar ele por um quadrado que tenha o mesmo valor de *f(n).* Mesmo este não esteja na lista fechada ele não poderia se deslocar para esta saída, pois, o algoritmo não permite sobrescrever funções na lista fechada. O fato de o inimigo entrar no “U” em busca do melhor caminho impossibilita sua saída porque estando dentro do “U”, para sair necessitaria voltar e esta volta passa pela mesma distância de entrada, apesar de quadrados diferentes. • Caminhar perpendicularmente em uma parede muito grande traz problema de *h(n)* porque em uma determinada parte da parede se torna mais perto chegar ao jogador se o inimigo se afastar um quadrado da parede e voltar para o centro da parede caminhando de forma perpendicular a ela. Nesta situação ele percorre todos os quadrados atrás da parede com a distância menor que a distância do centro da parede até o final dela. • Se o jogador está posicionado dentro de um quadrado que representa parede o algoritmo não vai encontra a posição do jogador. Os quadrados que representam a parede são descartados para efeito de cálculos. O algoritmo não prevê o controle do jogador sobre a matriz, ele apenas captura sua posição para ser calculada.

Tabela 5 - Caso de teste na fase de elaboração.

6.3 Fase construção

Na fase de construção, o sistema e documentação serão desenvolvidos. Uma versão de teste e o manual do usuário deverão ser esboçados. Segundo (MARTINS, 2004), a construção é considerada uma fase de processo de manufatura, focando o gerenciamento dos recursos, otimização de tempo, custos e qualidade. Os objetivos serão alcançados se o sistema for construído, testado e alcançado a aprovação dos investidores. Se os componentes estiverem completos e testados. Se o software estiver de acordo com a visão do projeto e pronto para implantação. Se tiver versões funcionais para testes e com o manual do usuário pronto a fase

estará concluída. É característica da fase o mapeamento de projetos para código, desenvolvimento dirigido por testes e refatoração: para aplicar pequenas mudanças de ajuste preservando o comportamento do software.

6.3.1 Disciplina de implementação na fase de construção

Nesta disciplina, o desenvolvimento dos componentes segue a descrição das partes do software:

a) Modelagem dos personagens;
b) Texturização dos personagens;
c) Animação do jogador inimigo;
d) Implantação do código-fonte *Python*;
e) Lógica do Blender.

6.3.1.1 Desenvolvimento dos componentes na fase de construção

a) Criação dos personagens e cenários:

Todos os objetos do jogo foram criados pelo aplicativo Blender. No Modo de Edição, Edit Mode, é possível manipular objetos através de suas faces. Para modelar uma pessoa parte-se de um objeto simples como um cubo. O cubo possui seis faces, oito vértices e doze arestas. Ao aplicarmos uma divisão neste objeto todos estes itens se duplicam, ou então, pode-se aplicar uma divisão apenas em uma face, ou uma aresta para duplicar uma parte do cubo em particular. Criando componentes e movendo-os é possível modelar qualquer objeto. Para suavizar a aparência do objeto é aplicado um modificador *Subesurf* para suavizar as arestas do objeto. Figura 24.

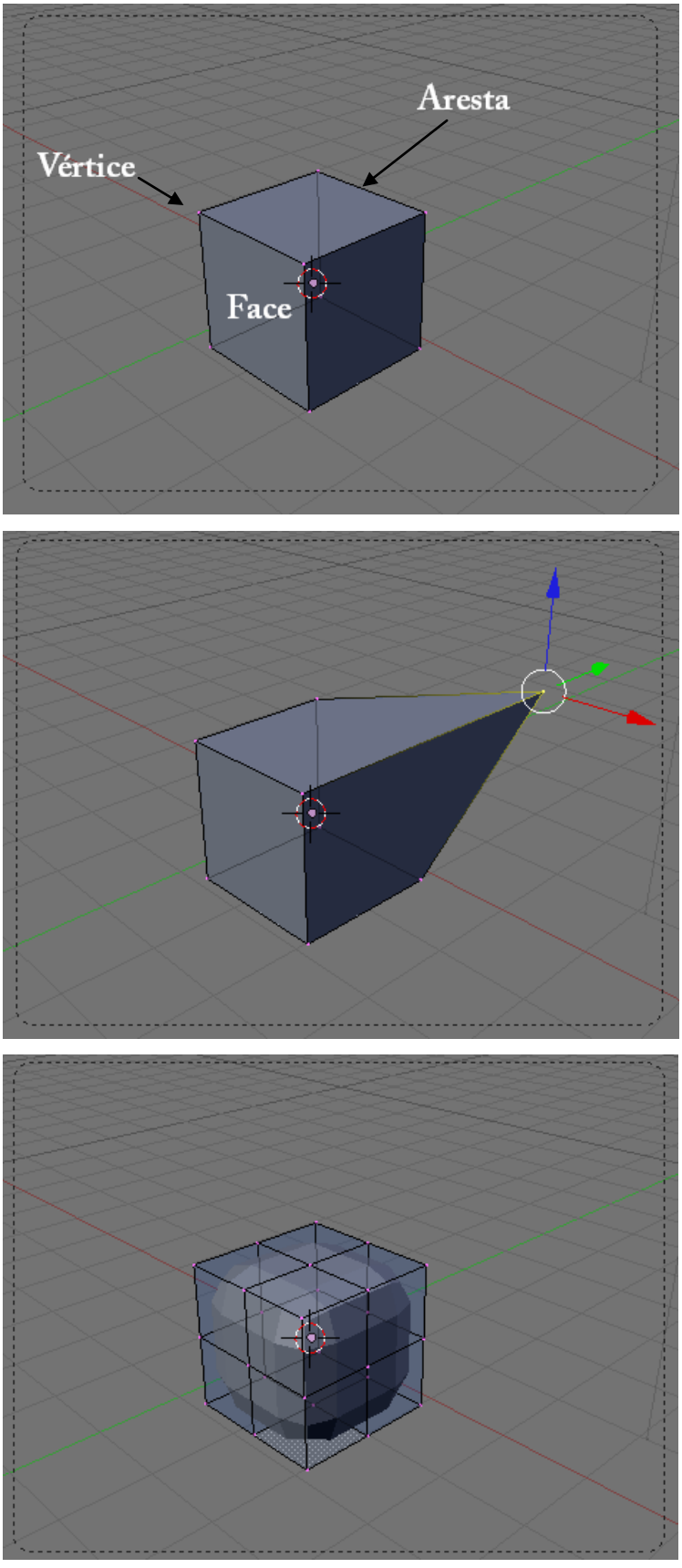


Figura 24 - Passos para modelar um objeto

- Desenvolvendo o jogador inimigo

A imagem de um ser humano de frente e perfil no *Edit Mode*, Figura 25, serviu como modelo para a criação do jogador inimigo a partir de um cubo. Como Cube é o nome padrão do objeto ele será alterado para Corpo.

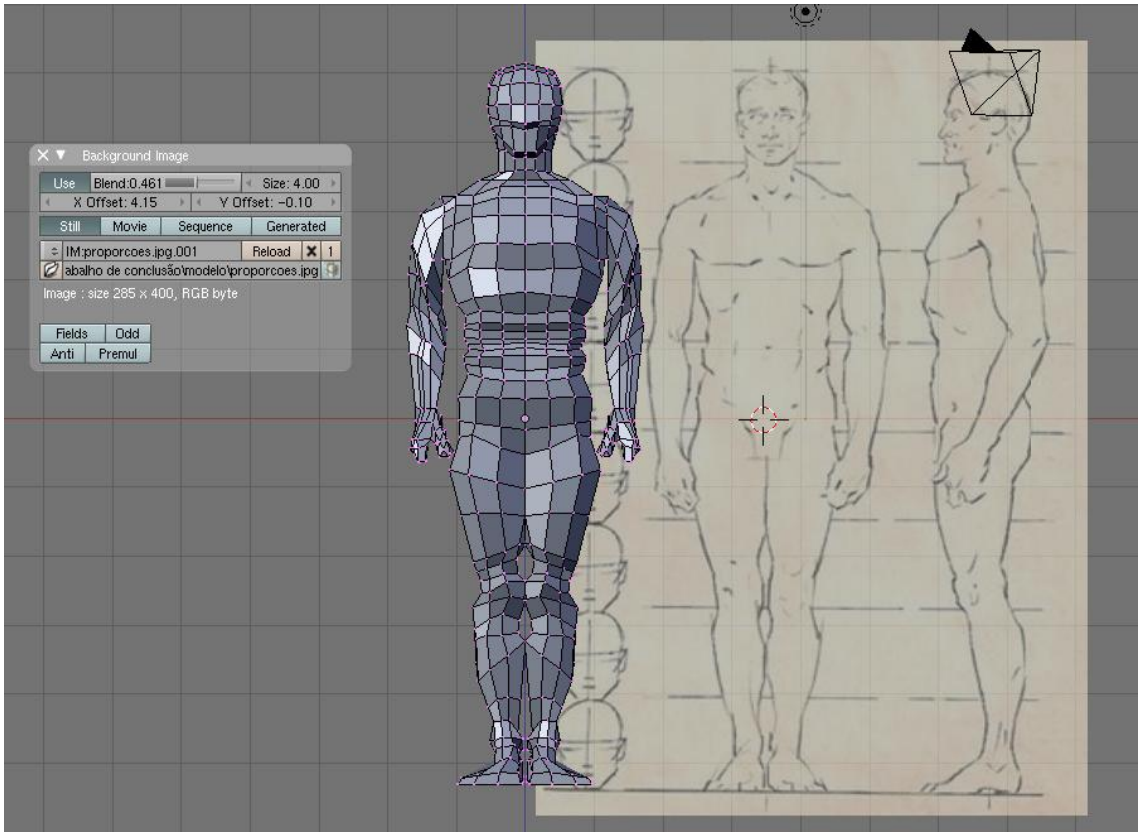


Figura 25 - Modelando inimigo, objeto Corpo.

- Desenvolvimento do jogador

A estrutura do jogador é apenas um cubo com uma câmera. Seu principal atributo é ser invisível.

- Desenvolvimento da arma

Para modelar a arma foi utilizado o mesmo procedimento, uma foto de modelo para a modelagem do objeto arma.

- Desenvolvimento do cenário

No cenário não foi utilizado foto. Ele foi modelado na forma de um cubo grande para colocar todos os outros objetos dentro como árvore, jogador, pedras e inimigos.

- Desenvolvimento das árvores

O aplicativo para desenvolver árvores foi o Arbaro. Com ele foi possível criar uma árvore a partir de um modelo em duas dimensões, como um desenho em uma folha de papel. O aplicativo pede o número de folhas, o formato da copa da árvore, o tipo de tronco, etc, Figura 26.

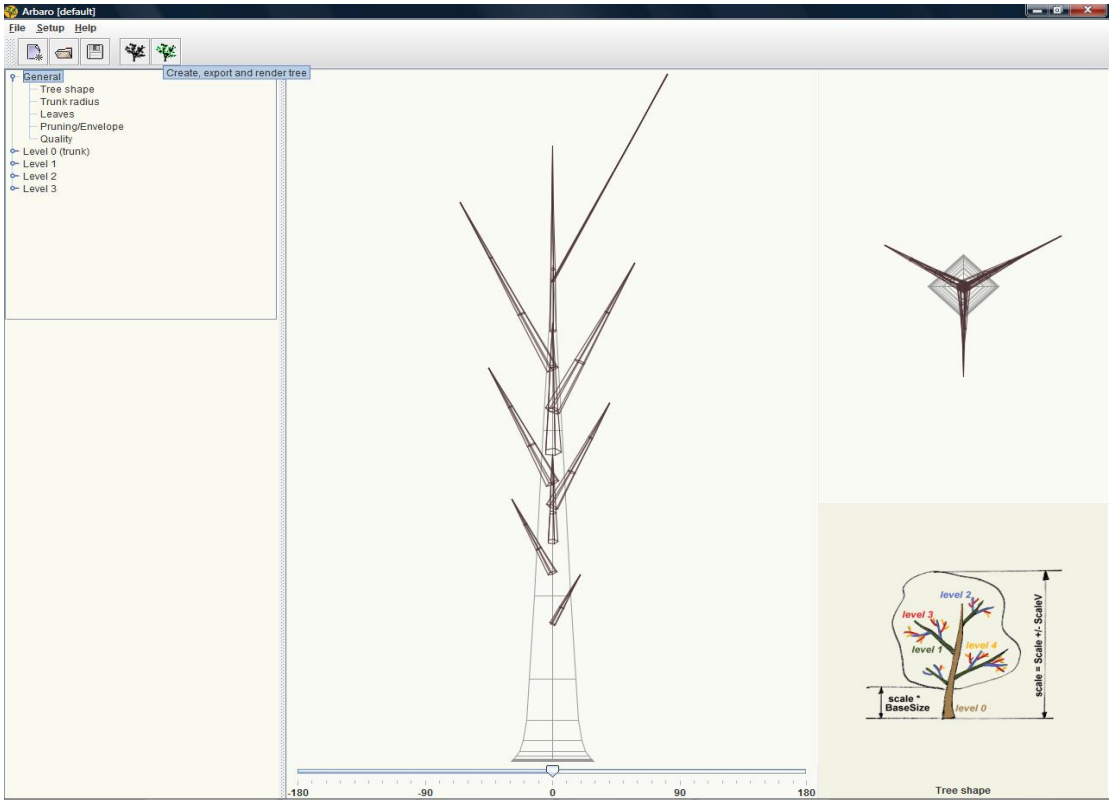


Figura 26 - Aplicativo Arbaro, configurando uma árvore para executar no Blender.

No Blender, importa-se o projeto criado no Arbore com a forma de objeto *Wavefront Object* e, então, a árvore é criada em três dimensões, Figura 27.

Figura 27 - Árvore em três dimensões importada do aplicativo Arbaro.

b) Pintura dos personagens e cenários:

Os objetos foram coloridos de duas formas. A primeira utilizando *Shading* para aplicar cores. Alguns objetos como ícones de vida e munição foram feitos nesta forma selecionando o objeto e, em *Sheding*, criando um novo material ligado de forma automática ao objeto selecionado, no ícone munição ou coração, manipulando a sua cor e sua textura, Figura 28.

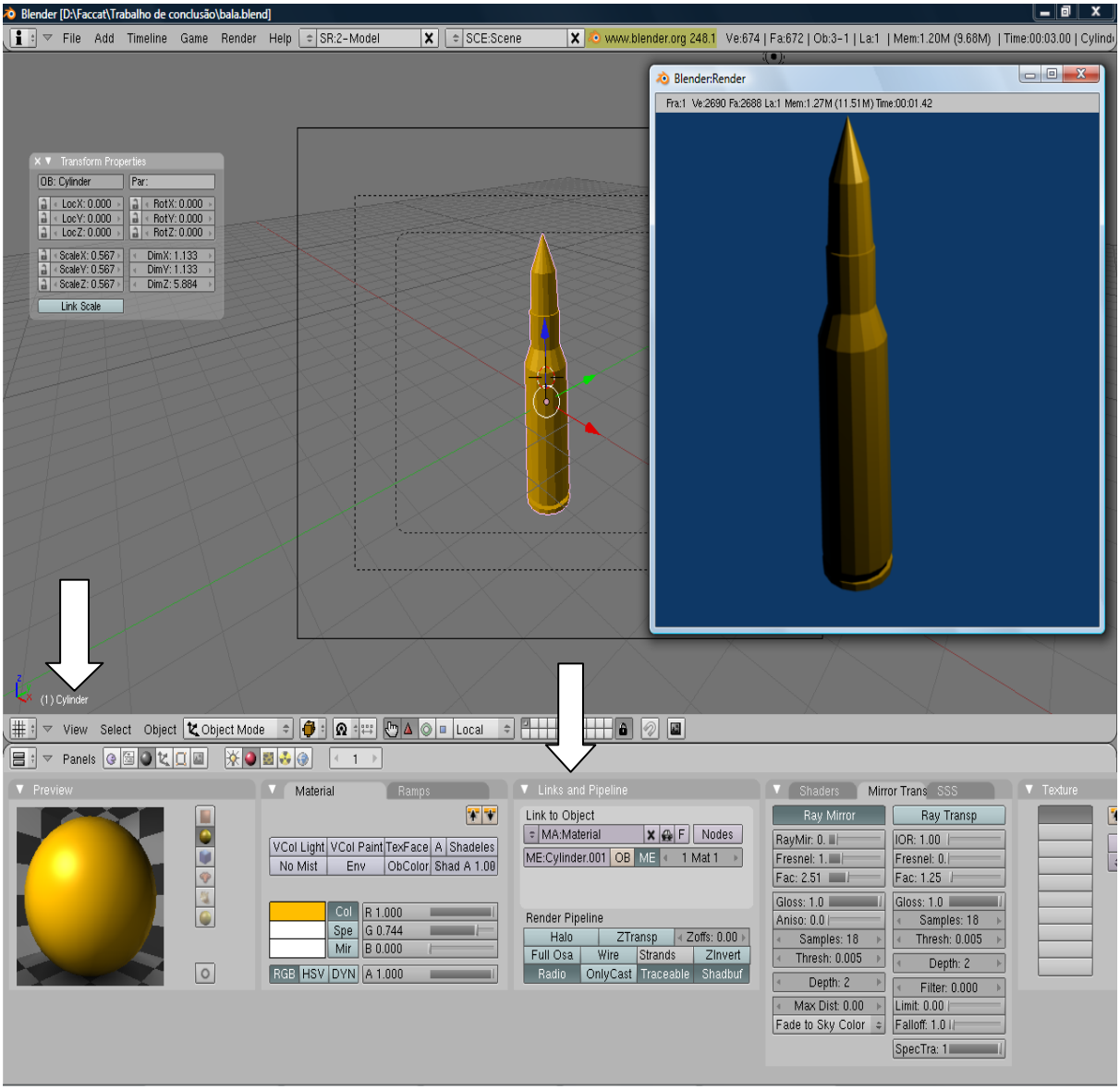


Figura 28 - Textura de um objeto Sheding. Objeto Cylinder ligado ao material chamado Material.

A segunda forma está relacionada a carregar arquivos em formatos de imagens com extensão Jpeg ou Gif. A roupa do inimigo, o chão do cenário e as pedras foram feitas com esta técnica no modo de edição seleciona-se o objeto, ou face do objeto, e aciona-se a UV *Calculation*, para calcular como a imagem se enquadra na superfície do objeto, Figura 29.

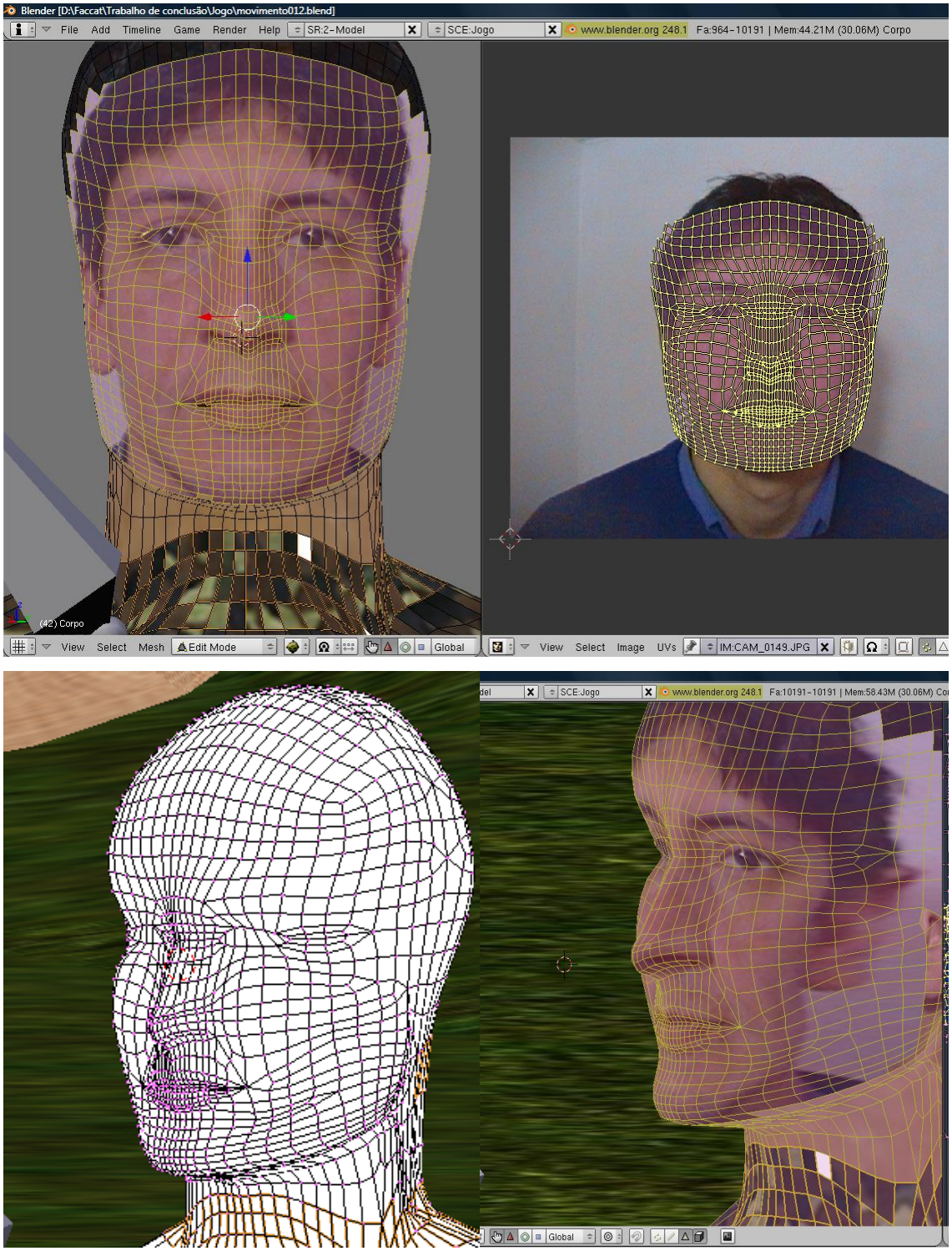


Figura 29 - Mapeamento de uma foto sobre a superfície de um objeto utilizando *UV Calculation*.

c) Animação do personagem inimigo:

O andar do jogador inimigo é desenvolvido na função de animação do Blender, *Timeine*. O objeto responsável pela ação é a *armature* e ela representa o esqueleto do jogador inimigo, sua construção ocorre com a adição do *Bones*, ossos, e um osso está sempre ligado

ao outro, sua hierarquia é o primeiro osso criado é pai do segundo, assim, sucessivamente. Significa dizer se "movimentar o osso fêmur os ossos restantes da perna também serão movidos". Com os ossos da perna direita e esquerda foram aplicadas as técnicas de cinemática inversa. Segundo BRITO (2008), a cinemática serve para mover um conjunto de ossos pré-definido conforme o movimento que apenas um osso deste conjunto executar. Se for movido o pé para cima, os ossos da articulação tornozelo e joelho são influenciados conforme o movimento. Para criar uma animação esse recurso torna-se prático, pois não há a necessidade de controlar os ossos individualmente durante o movimento de andar.

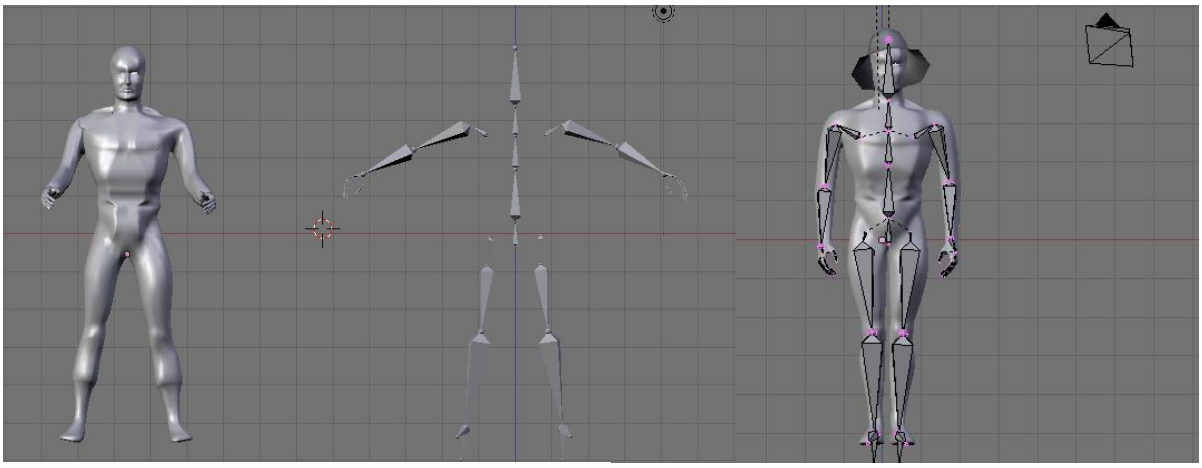


Figura 30 - Objeto Corpo e Armature (esqueleto) separados. Depois alinhados.

Alinhando o objeto *Armature* com o objeto Corpo, Figura 30, pode-se ligar estes dois objetos. O movimento que o esqueleto, *Armature*, executar o corpo também acompanhará. Para este efeito, Figura 31, é selecionado o osso desejado e o conjunto de vértices do objeto Corpo que será deformado em seguida ativando a função "*Assign*". Todos os ossos devem passar por este procedimento. Quando o osso Abdômen for movido o corpo também será.

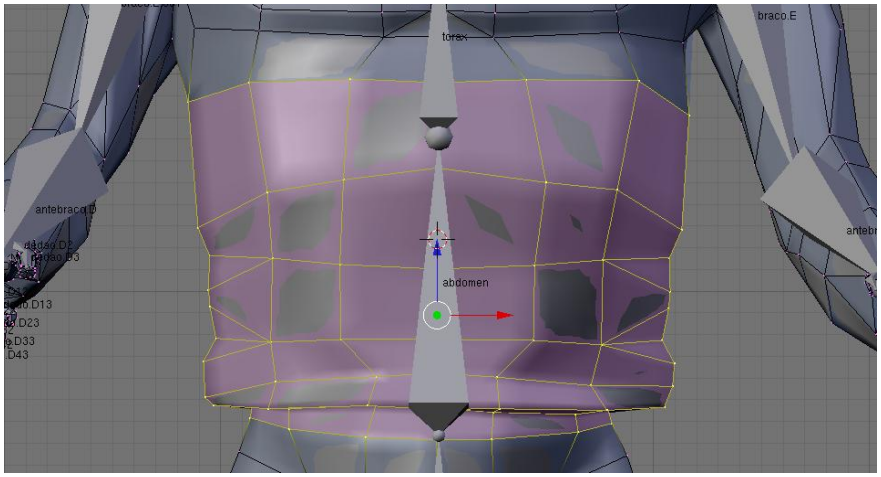


Figura 31 - Seleção dos vértices abdominais (em rosa) do objeto Corpo para ser vinculado ao osso (Bone) chamado "abdômen".

O *Timeline* do Blender produz a animação do jogador inimigo. Iniciando no quadro 1, "*Start:1*", e termina no quadro 20, "*End:20*", sua duração é de 0.8 segundos. Na figura 32, o quadro 1 representa o inicio da animação, então o inimigo é colocado em uma pose de inicialização e o quadro 1 é eleito como quadro-chave para capturar a pose do inimigo. No quadro 5, Figura 32, a pose do inimigo foi modificada para dar seqüência na dinâmica da caminhada. Com esta nova pose é acionado outro quadro-chave no quadro 5. Na figura 33 ocorre a seqüência da caminhada.

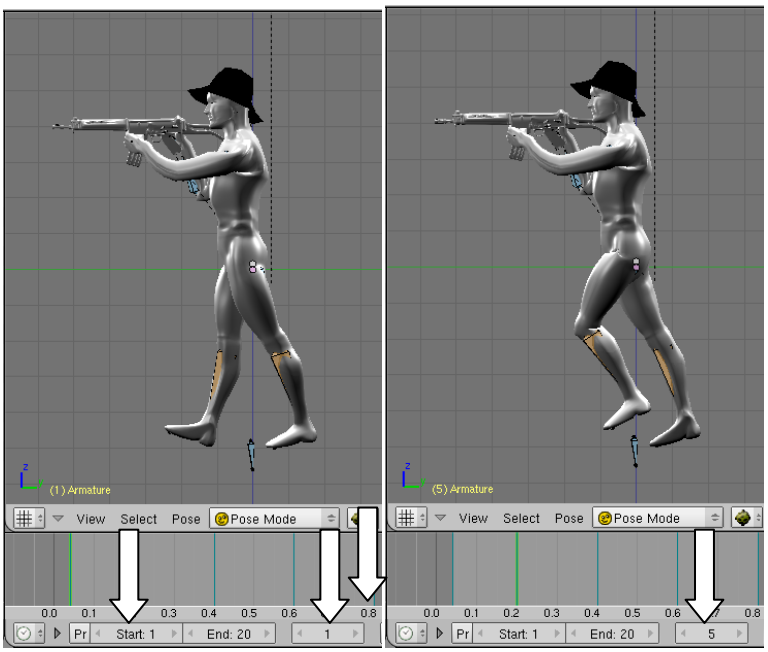


Figura 32 - Quadros-chaves 1 e 5.

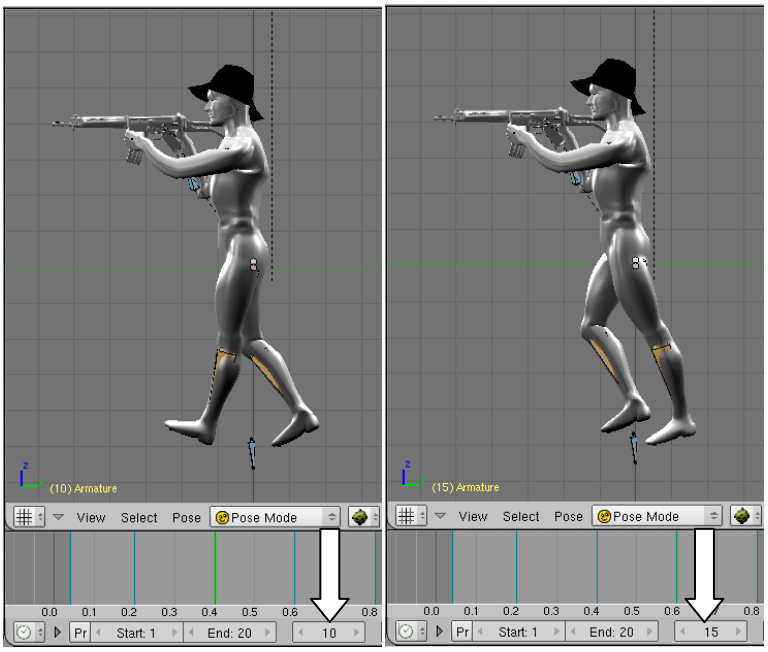


Figura 33 - Criação de quadros-chaves nos intervalos 10 e 15.

No quadro 20, Figura 34, termina a animação e o ciclo da ação de caminhar está completo, pois, o inimigo voltou a pose inicial. No *Timeline*, a régua de tempo 0.0 a 0.8 é ativada para animar o inimigo, as linhas azuis verticais representam os quadros-chaves criados no intervalo. O quadro 1 foi criado em 0.0 segundos, quadro 5 em 0.2 segundos, quadro 10 em 0.4 segundos, quadro 15 em 0.6 segundos e quadro 20 em 0.8 segundos. A linha vertical verde representa a posição da animação em relação ao tempo e o quadro correspondente. Na Figura 34, a linha verde está posicionada na posição quadrado 20 tempo 0.8 segundos.

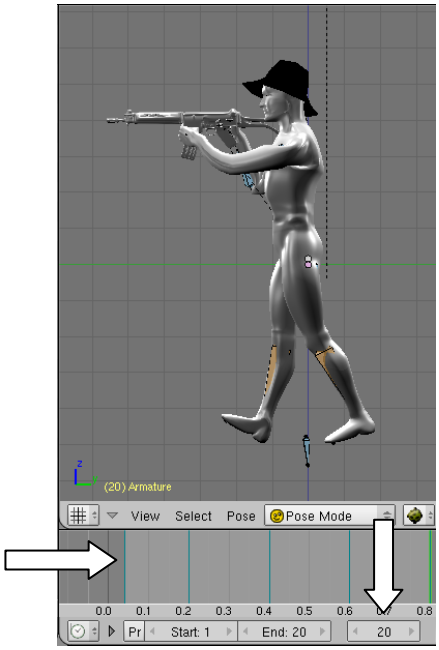


Figura 34 - Volta ao estado inicial.

Ao ativar a animação os quadros restantes se completam. Por exemplo: do quadro-chave 1 até o quadro-chave 5 os quadros que estão no intervalo são completados automaticamente. O deslocamento das pernas do jogador passam do quadro 1 para o quadro 2, 3 e 4 até 5 de uma forma suave e contínua. Na seqüência o deslocamento das pernas estão direcionadas do quadro 5 para o quadro-chave 10, ocorrendo o deslocamento das pernas nos quadros deste intervalo. Esta dinâmica ocorre até o quadro 20 e termina a ação e cada perna executa um passo completo. Acionando a animação de forma continua cria-se a sensação da ação de caminhar. É importante notar que apenas está correndo a animação de caminhar, ele não possui a ação de deslocamento pelo cenário, o deslocamento será criado pelo código *Python*.

d) Implantação do código-fonte:

O código Andar.py desenvolvido em Python serve para controlar inimigo dentro do cenário. O algoritmo A* (A estrela) foi remodelado devido a não eficiência em algumas situações conforme está relatado na seção de testes. O conceito do algoritmo continua A*.

A Figura 35, mostra o código Andar.py importando as bibliotecas para execução de alguns comandos, capturando informações do sensor e atuador através da hierarquia *GameLogic*. Em seguida, está se desmembrando as posições de deslocamento e rotação do

jogador, inimigo e vazio. Em “if clic.isPositiove( ):” está se definindo que, caso o mouse seja acionado, o objeto vazio recebo os dados da posição do jogador, valores em A e valores em B. Os nomes “sensor”, *Empty*, *Cube* e *Armature* em vermelho são nomes dos objetos criados através dos Blocos lógico da Figura 36.

```
import Blender
import GameLogic
import math
import time
import random
print ''
print "INICIO DO CODIGO INICIO DO CODIGO"
print ''
# Captura de dados no Blender
cont = GameLogic.getCurrentController()
own  = cont.getOwner()
clic = cont.getSensor('sensor')
acao = cont.getActuator("act")
#tiro = cont.getSensor('mira')
PegaValor=own.getPosition()
Orientacao=own.getOrientation()

# Inimigo
Inimigo = Blender.Object.Get('Armature')
X=Inimigo.LocX
Y=Inimigo.LocY
ZX=Inimigo.RotX
ZY=Inimigo.RotY
ZZ=Inimigo.RotZ
print "Posicao inimigo =",X,Y

# Jogador
Homem = Blender.Object.Get('Cube')
A=Homem.LocX
B=Homem.LocY
print "Posicao do jogador =",A,B

# Vazio
Vazio = Blender.Object.Get('Empty')
C=Vazio.LocX
D=Vazio.LocY
print "Posicao vazio =",C,D
print ''

if clic.isPositive():
  Vazio.LocX=A
  Vazio.LocY=B

```

Figura 35 - Parte do código Andar.py, capturando dados.

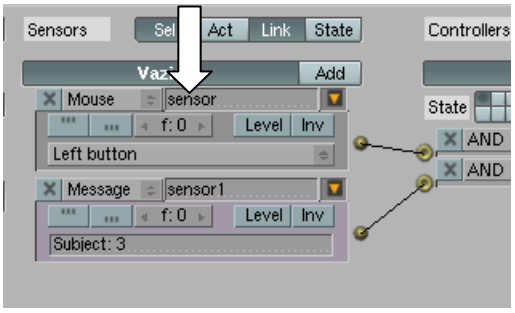


Figura 36 - Sensor: nome do Sensors Mouse ligado ao objeto Vazio

O algoritmo A* (A estrela) consiste em capturar a posição e orientação do jogador, inimigo, objeto vazio e verificar o estado do mouse. Foi organizada uma seqüência de números que representam a posição de um objeto no jogo. Por exemplo: pedra=[20,05], na posição eixo X=20 e eixo Y=05 representa o obstáculo pedra e quando a posição do inimigo, também através de orientação X e Y, for igual a da pedra significa posição proibida para inimigo, ele deve desviar esta posição. Outro controle será aquele caso o inimigo esteja na mesma posição do último disparo do jogador representado pelo objeto vazio. Quando o inimigo chegar na posição do objeto vazio, ele deve andar em direção a outra posição escolhida de forma aleatória entre quatro posições, onde cada uma estará localizada em um canto do cenário. A construção do código neste formato serve para jogador não ficar com a sensação de estar sempre sendo perseguido pelo inimigo. O jogador inimigo "ouvindo" o disparo do tiro, caminha na direção do som até encontrar o jogador ou até o local onde ocorreu o disparo e, chegando lá, se o jogador não está nesta posição ele começa a "procurar o jogador" representado pelo deslocamento para uma das quatro posições pré-definidas pelo código.

Em seguida, o algoritmo verifica as posições que estão em volta do inimigo (frente, atrás e dos lados) Figura 37, e atribui um peso. Os valores do eixo C e D do objeto vazio representam a posição do último disparo do jogador dentro do jogo e elas são comparadas com os eixos X e Y do inimigo. Dependendo da combinação, por exemplo: eixo X do inimigo igual ao C do vazio e eixo Y do inimigo maior do que eixo D do vazio, então se aplica o valor definidos no algoritmo calculando a distância entre inimigo e vazio (PeI) e aplicando o valor 2 a distância para representar o maior peso: PI00=PeI +2. Na seqüência é organizada uma lista que representa as posições em volta do inimigo com peso, posição X e posição Y. As variáveis PI00, PI01, PI02, etc, representam as posições frente, atrás e lados do jogador inimigo.

```
if X==C and Y>D:
  print "X igual C, Y maior D"
  PI00=PeI+2
  PI01=PeI+1
  PI02=PeI+2
  PI10=PeI+1
  PI11=PeI+0
  PI12=PeI+1
  PI20=PeI+0
  PI21=PeI-1
  PI22=PeI+0

if X==C and Y<D:
  print "X igual C, Y menor D"
  PI00=PeI+0
  PI01=PeI-1
  PI02=PeI+0
  PI10=PeI+1
  PI11=PeI+0
  PI12=PeI+1
  PI20=PeI+2
  PI21=PeI+1
  PI22=PeI+2
```

Figura 37 - Atribuição de pesos para as posições em volta do inimigo.

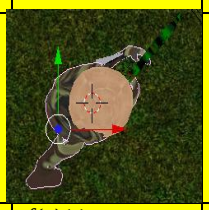
Para contornar objetos é verificado as posições em volta do inimigo (frente, atrás e dos lados), se existir uma posição pedra (vista anteriormente) com os mesmos valor de uma posição em volta do inimigo é acrescentado um peso de 40 a distância desta posição. Agora a lista de posição em volta do inimigo está atualizada com o peso da distância ou peso do obstáculo, e a posição. Por exemplo: PosiçãoFrenteEsquerda[peso,X,Y], PosiçãoFrente[peso,X,Y],PosiçãoFrenteDireita[peso,X,Y] ou [PI00,X00,Y00], etc. PoI é o conjunto das posições do inimigo.

```
# PosicaoInimigo = PoI
PoI=[ [PI00,X00,Y00],[PI01,X01,Y01],[PI02,X02,Y02],
      [PI10,X10,Y10],[PI11,X11,Y11],[PI12,X12,Y12],
      [PI20,X20,Y20],[PI21,X21,Y21],[PI22,X22,Y22] ]
```

Figura 38 - Algoritmo com a posição do jogador inimigo.

Os valores de X e Y do vazio são comparados com o X e o Y do inimigo novamente. Em seguida, é verificado qual o menor peso entre as três posições mais baixa. Por exemplo: se a PosiçãoFrenteEsquerda tiver o peso menor do que a PosiçãoFrente, então, a orientação é acionada para que o inimigo fique voltado para a menor posição, depois é acionado o deslocamento do inimigo. A frente do inimigo está voltada para o eixo Y. Figura 39. A seta verde representa o eixo Y do inimigo apontando para a menor posição que o inimigo pode ir.

Se ela estive-se voltada para baixo sua rotação seria de zero grau, como está voltada para cima sua rotação é de 180 graus. Em comparação com a última posição do inimigo, no caso 135 graus, é acionada a rotação para alinhar o inimigo com sua nova direção, depois se aciona o deslocamento no eixo Y para o inimigo simular uma caminhada para frente.

f(n)10 *PI00*	*f(n)09* *PI002*	*f(n)10* *PI03*
f(n)11 *PI04*		*f(n)11* *PI06*
f(n)12 *PI07*	*f(n)11* *PI08*	*f(n)12* *PI09*

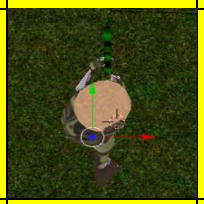
f(n)10 *PI00*	*f(n)09* *PI01*	*f(n)10* *PI02*
f(n)11 *PI03*		*f(n)11* *PI06*
f(n)12 *PI07*	*f(n)11* *PI08*	*f(n)12* *PI09*

Figura 39 - Vista superior código Python verificando caminho. Menor distância.

Os demais códigos ApareceMouse.py e SaiMouse.py e Mouse.py, servem apenas para deixar o mouse visível nos menus, invisível durante o jogo e atualizar a posição câmera do jogador com as teclas de direção.

e) Descrição do componente Lógica do aplicativo Blender:

A lógica dos blocos lógicos do Blender é descritas conforme os objetos:

- Lógica do personagem jogador

O objeto que representa o jogador é o objeto Jogador, ele é feito a partir de um cubo.

Para manipular a câmera como se fosse a visão do jogador o cubo deve andar pelo cenário e através dos sensores, controladores e atuadores ler as entradas do teclado para orientar a direção. Os sensores *Keyboard* chamados de sobe, desce, esq, dir estão todos ligados a um controlador Python e, ao mesmo tempo, cada sensor está ligado a um controlador AND e a um atuador do tipo *Sound* que executa um arquivo de som chamado Caminha.wav. Ele toca um som de passos que é executado em *Loop Stop*, ou seja, executa de forma repetida do inicio ao fim em quanto a tecla correspondente estiver pressionada.

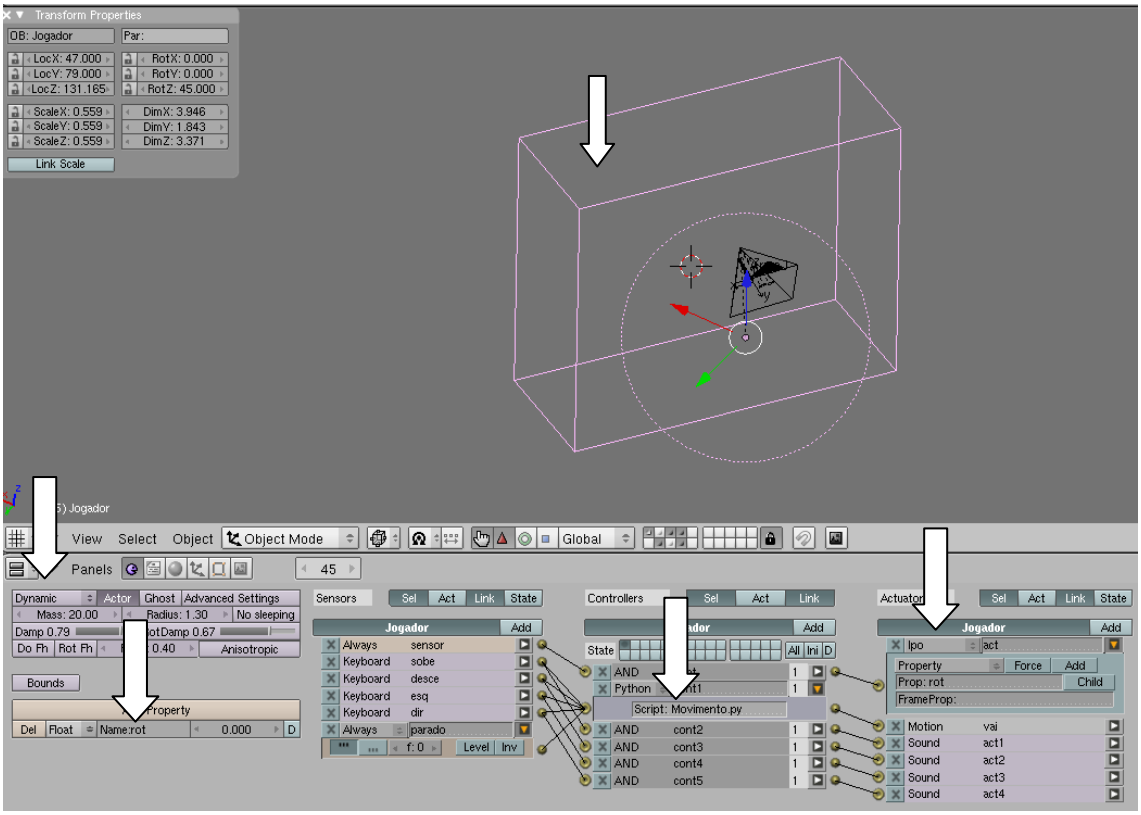


Figura 40 - Funções do objeto Jogador (cor rosa).

O controlador *Python*, Figura 40, está associado a um atuador *Motion*, chamado de "vai", para capturar as variáveis de posição e rotação do objeto. A manipulação dos dados é

feita exclusivamente pelo código *Python* chamado de Movimento.py, ele executa movimento de deslocamento e rotação do Jogador. Na Figura 41, o código importa as bibliotecas necessárias e captura o objeto Jogador em uma variável chamada de Valor. A variável "cont" captura as informações dos sensores do objeto, por exemplo: a variável "so" captura as informações do sensor chamado "sobe". Logo a baixo, o selecionador "*if*" (em vermelho) está associada a uma função "so.isPositive()" que verifica se o mouse foi acionado com a tecla sobe, se confirmado então ocorre o deslocamento no eixo Y com a intensidade 0.1. Se outra tecla for pressionada o código executa o selecionador apropriado e, em seguida, as informações são atualizadas na variável Valor, para ocorrer o movimento de deslocamento desejado pelo jogador.

```
import GameLogic
import Blender
#Para obter um controlador
Valor=Blender.Object.Get('Jogador')
#A=Cilindro.LocX
#B=Cilindro.LocY
cont =GameLogic.getCurrentController()
own =cont.getOwner()
so =cont.getSensor('sobe')
de =cont.getSensor('desce')
es =cont.getSensor('esq')
di =cont.getSensor('dir')
pa =cont.getSensor('parado')
act =cont.getActuator("vai")
if so.isPositive():
  act.setDLoc(0,0.1,0,1)
  GameLogic.addActiveActuator(act,1)
if de.isPositive():
  act.setDLoc(0,-0.1,0,1)
  GameLogic.addActiveActuator(act,1)
if es.isPositive():
  act.setDLoc(-0.1,0,0,1)
  GameLogic.addActiveActuator(act,1)
if di.isPositive():
  act.setDLoc(0.1,0,0,1)
  GameLogic.addActiveActuator(act,1)
if pa.isPositive():
  GameLogic.addActiveActuator(act,0)
PegaJogador=own.getPosition()
Valor.LocX=int(PegaJogador[0])
Valor.LocY=int(PegaJogador[1])
```

Figura 41 - Código Movimento.py

Outra função do objeto Jogador é a capacidade de rotacionar sobre seu eixo Z como se o jogador estivesse virando o corpo em direção ao seu alvo. Figura 42.

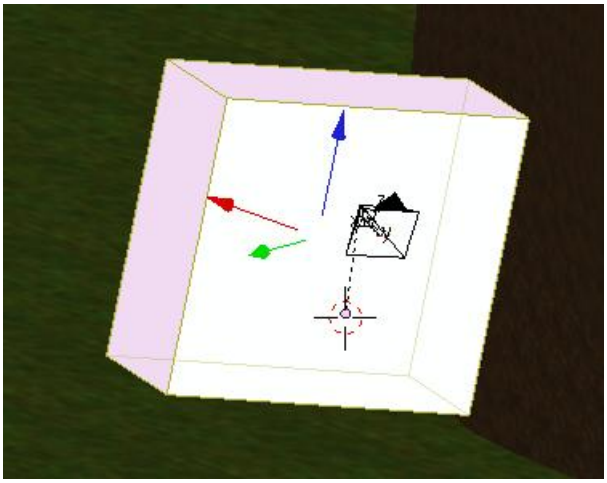


Figura 42 - Seta azul indica eixo Z, seta vermelha eixo X e verde eixo Y.

Vinculado ao objeto Jogador foi criado um *Add Property* do tipo *Float* com nome de rot, ver figura 43, para acumular valores. Um sensor *Always* "sempre" verifica o atuador *Ipo* do tipo *Property* atuando sobre a propriedade rot. Valores do movimento de rotação serão acumulados nesta propriedade.

O movimento de rotação do objeto Jogador é feito selecionando o eixo RotZ no display *Ipo Curve Editor*. Os valores mínimos e máximos de rotação dos eixos X e Y são respectivamente -180 a 180, -18 a 18. A faixa de rotação permite o objeto girar de um lado para o outro apenas entre estes valores e seu valor atual será passado para o rot.

A física do objeto Jogador é dinâmica. Seu peso é de 20 e *Radius* 1.30, significando a distância de colisão para qualquer outro objeto. Ela é representada pela circunferência dentro do objeto jogador. Figura 43. A gravidade é igual a do planeta terra 9.80.

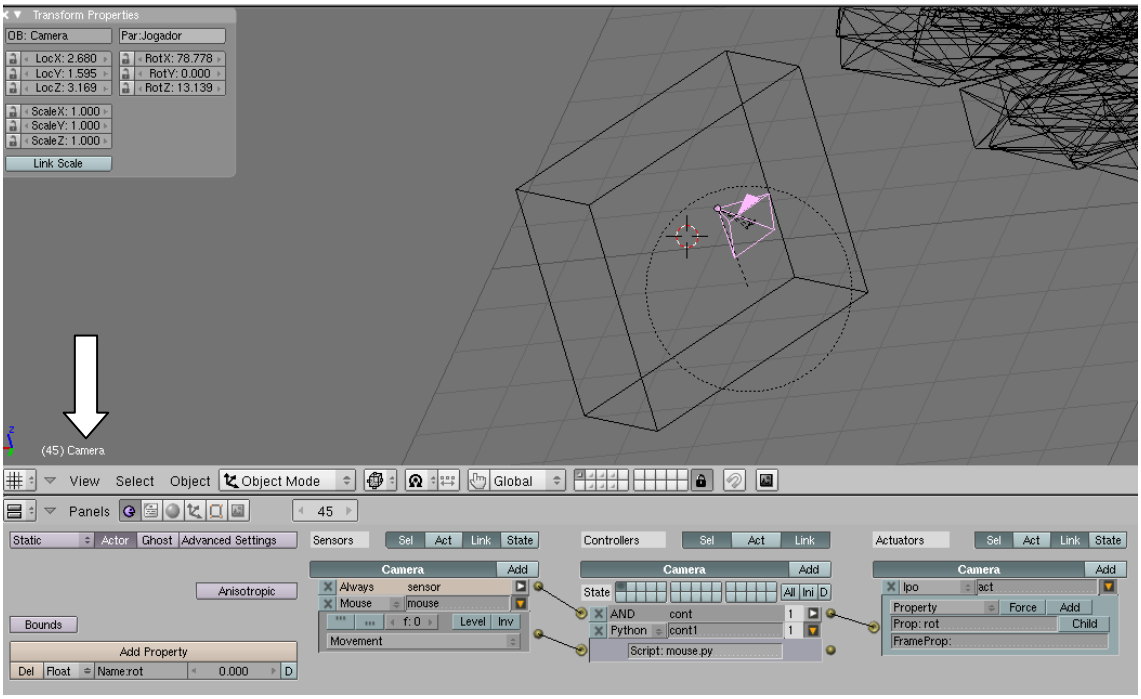


Figura 43 - Jogador (Cube)

O objeto câmera chamada de Câmera, Figura 43, está ligado diretamente ao objeto Jogador, qualquer movimento efetuado por ele é acompanhado pela câmera. O parentesco entre eles é o pai objeto Jogador e o filho objeto Câmera. O sensor criado e vinculado a câmera é um sensor *Always* e está sempre verificando o atuador Ipo com propriedade chamada de rot. Esta propriedade é criada em *Add Prorerty*, ela é do tipo *Float* com nome rot e acumula os valores do deslocamento da câmera, pois está vinculado a ela. Seu deslocamento será de cima para baixo e para gerar este movimento foi utilizado o *Ipo Curve Editor*. Os valores de faixa para o movimento ficaram ente X mínimo -60 e X máximo 89, Y mínimo -2 e Y Maximo 12.

A câmera também possui um atuador Mouse que captura o movimento do mouse e joga em um controlador do tipo *Python*. O código deste controlador, Figura 44, se chama mouse.py e serve para passar a sensação inversa ao mouse deslocando-se pela tela do computador, o mouse fica fixo no centro da tela e a imagem se desloca para enquadrar o mouse no centro da tela. Realmente o mouse está se movendo, porém com a tela fixa nele.

```
import GameLogic as GL
import Rasterizer as R

cont = GL.getCurrentController()
camera = cont.getOwner()
Cube = camera.parent
mouse = cont.getSensor('mouse')

midX = R.getWindowWidth()/2
midY = R.getWindowHeight()/2

sensibilidade = 0.1

if not hasattr(GL, 'init'):
  GL.init = True
else:
  Cube.rot -= (mouse.getXPosition() - midX)*sensibilidade
  if Cube.rot > 180:
    Cube.rot -= 360
  if Cube.rot < -180:
    Cube.rot += 360
  camera.rot -= (mouse.getYPosition() - midY)*sensibilidade
  if camera.rot > 75:
    camera.rot = 75
  if camera.rot < -75:
    camera.rot = -75
R.setMousePosition(midX, midY)

```

Figura 44 - Código mouse.py
Fonte:

A arma do jogador não possui nenhuma função lógica, porém, ela é filha da câmera e qualquer movimento da câmera a arma também acompanha.

Existe um objeto vazio filho, Figura 45, ligada a arma e posicionado em sua frente, ele possui um sensor do tipo mouse que captura um acionamento da tecla esquerda do mouse e passando por um controlador END aciona um atuador Editor de Objetos, *Edit Object*, do tipo Adiciona Objeto, *Add Object*, fazendo o objeto aparecer na tela principal no lugar do vazio. No caso, o objeto que aparece será o objeto Bala. Ao mesmo tempo envia uma mensagem, atuador *Message*, para outro sensor de outro objeto. Esta mensagem se chama “1” e significa a tecla esquerda do mouse foi pressionada.

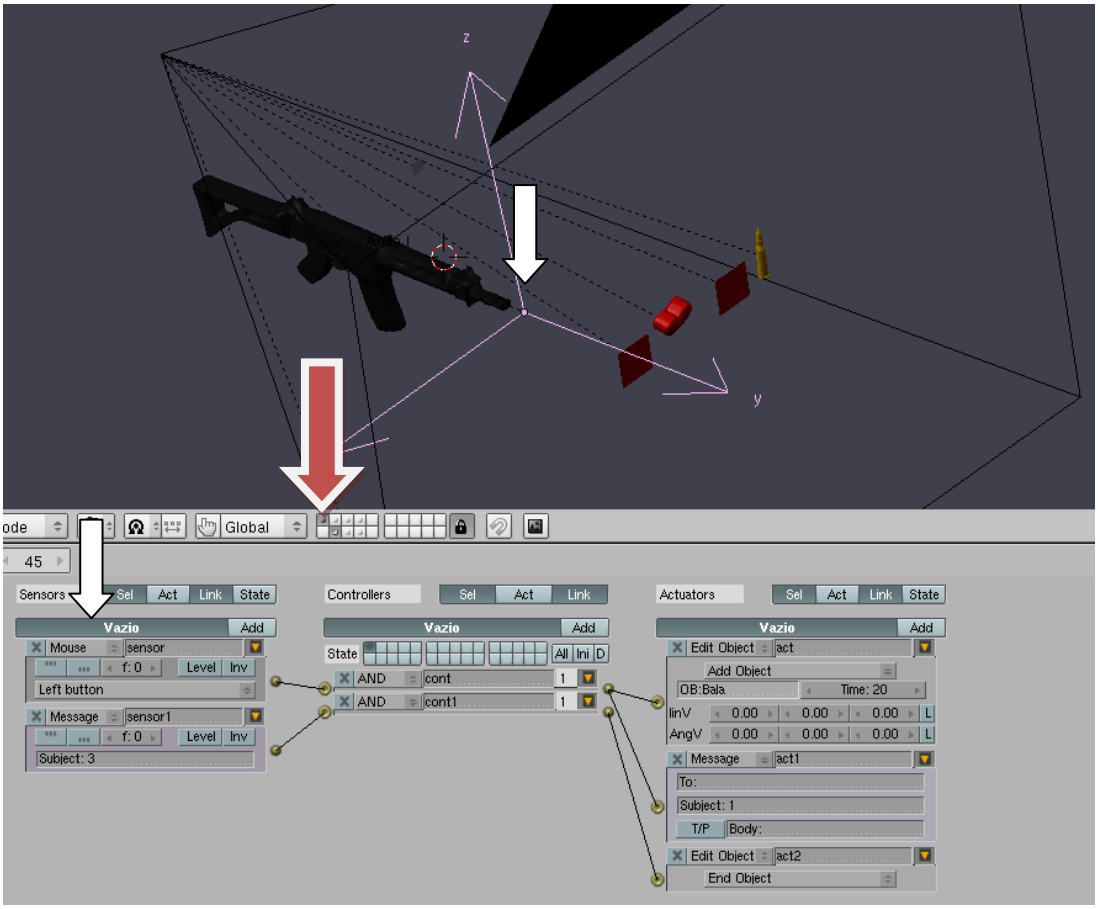


Figura 45 - Objeto vazio ligado a arma. A seta vermelha indica a cena ativa ou tela principal do jogo.

O objeto que aparece na tela principal é chama Bala e está em um Layer invisível, uma tela na mesma sena que não está ativa, os objetos deste layer não estão aparecendo no jogo. Quando a tecla esquerda do mouse é pressionada a Bala aparece na tela principal. Esta bala sempre "*Always*" executará um movimento de deslocamento no eixo Y com velocidade 1.00, Figura 46.

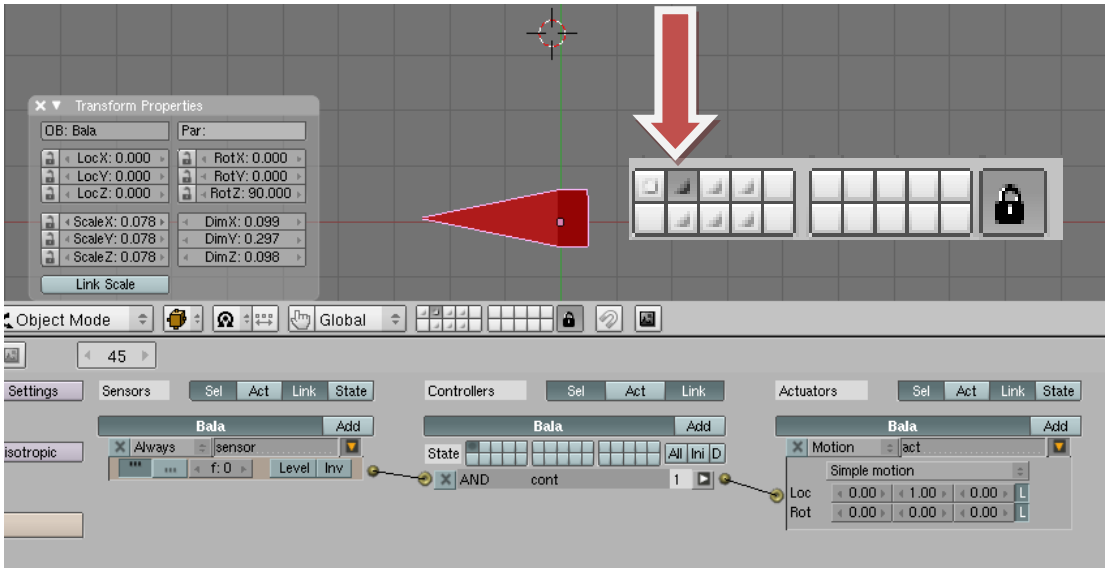


Figura 46 - Layer onde o objeto Bala (em vermelho com pontinho rosa) está quando não é chamado.

- Lógica do personagem inimigo

Todos os sensores do jogador inimigo estão ligados a armadura dele, *Armature*, ela é a estrutura pai de todos os outros objetos ligado ao inimigo como objeto chapéu, objeto pele, etc.

O objeto *Armature* é a estrutura do jogador inimigo, Figura 47. Nele está ligado um sensor Mouse que verifica se o botão esquerdo foi pressionado. Se foi pressionado a informação é enviado para o código *Python* chamado Andar.py que calcula as posições do jogador e do inimigo e efetua a ação de caminhar.

Outro sensor ligado a *armature* é o de aproximação *Near*, quando o objeto Jogador se aproximar a uma distância de 30 metros do jogador, no caso cada unidade representa 1 metro, então o sensor é acionado e através de um controlador AND aciona um atuador *Edit Object*, do tipo *Track To*, que faz o *Armature* sempre aponta seu eixo Y para o jogador dentro desta distância. O arquivo Andar.py apenas aciona o deslocamento do inimigo e não avalia o caminho até o jogador, ele deixa isto para o sensor *Near*.

Há um objeto vazio, chamado de Empty.001, ligado ao esqueleto e seu sensor é um Ray que emite um raio com distancia de 30 metros.O controlador será acionado somente se o raio encontrar um objeto especifico com a propriedade chamada de rot que pertence ao objeto Jogador. Encontrando o jogador o atuador executa um *Edit Object*, que adiciona um objeto, *Add Object*, chamado *Bal* que está invisível em outra tela. Esta estrutura serve para disparar a bala do jogador inimigo. Ao mesmo tempo, um atuador *Message* envia uma mensagem para

outro objeto, ela é chamada de 11. Com a aproximação do jogador o sensor *Near* deixa alinhado o inimigo com o jogador. O objeto Empty.001, Figura 48, ligado ao inimigo também está alinhado nesta posição, logo, quando o sensor *Ray* bate no jogador ele disparado um objeto Bal. Bal é transportada para a tela principal e executa um deslocamento com velocidade 1.00. O tiro do inimigo.

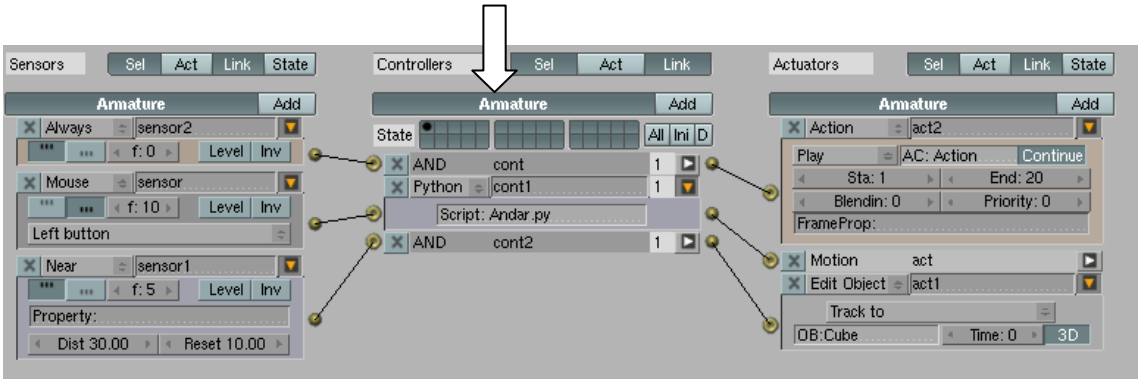


Figura 47 - Conjunto de sensores e atuadores do objeto Armature.

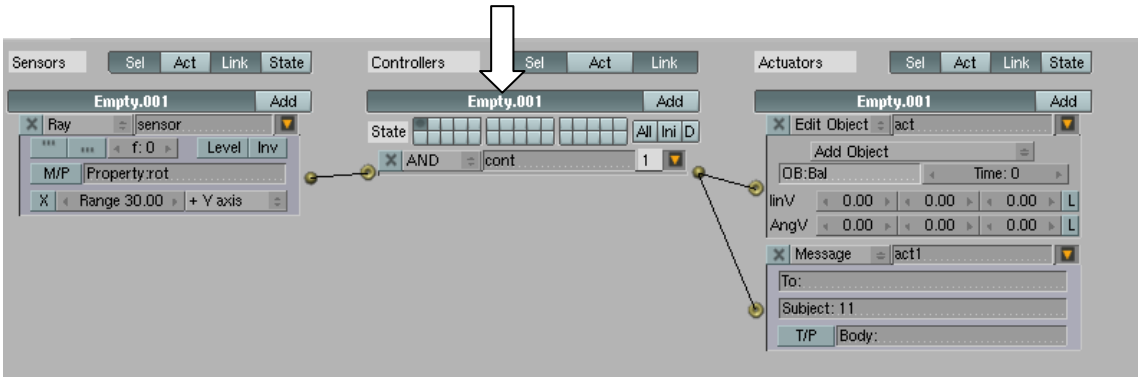


Figura 48 - Sensores e atuadores ligados aos objetos Empty.001.

Figura 49 - Objeto Empty.001(Vazio) filho ligado ao esqueleto (Armatrure) do jogador inimigo, embaixo da textura de camuflagem.

- Lógica dos objetos vazio

Há três objetos vazio no jogo que servem como referência para os blocos lógicos e o código *Python*. Há dois vinculados um jogador e outro ao inimigo, eles servem como referência para quando ocorrer o disparo da arma do jogador e do inimigo. Quando ocorrer os disparos os projéteis, Bala e Bal, são projetados na tela principal e na posição dos respectivos vazios.

O terceiro objeto vazio é reservado para referenciar o caminho analisado pelo inimigo. Quando o jogador dispara um tiro clicando com o botão esquerdo do mouse o código captura a posição do jogador e atribui o valor para o objeto vazio, então, ele assume a posição do jogador. O inimigo analisa o melhor caminho e parte na direção do vazio, independente do jogador estar lá ou não. Se o jogador dispara outro tiro o processo é repetido. O objetivo deste código é simular o jogador andar na direção do som do disparo, propiciando ao jogador poder observar o inimigo andar no cenário procurando seu alvo.

- Contador de munição e vida

O objeto Vida e o objeto Munição, os números que contam a quantia de munição e a quantia de vida do jogador, são feitos pelo *UV Calculation*, com uma imagem de uma letra em um fundo preto, geralmente @ (arroba), colada em um objeto plano. No *Edit Mode*, com o plano selecionado, no menu *Editing* e em *Texture Face*, as opções *Alpha*, para deixar o objeto

transparente onde a cor for preto e *Text*, para editar textos durante o jogo são selecionados e depois adicionada uma propriedade, *Add Property* do tipo inteiro, Int, com o nome de *Text* que será utilizada para acrescentar ou diminuir valores ligado ao objeto Vida.

Na Figura 50, a propriedade em "*Add Property*" chamada *Text* está com a capacidade de vida 5. Nesta figura, O sensor *Message* chamado "sensor" recebe uma mensagem de outro objeto, esta mensagem se chama 2, "*Subject:2*". Este outro objeto é acionado quando ocorrer a colisão do projétil inimigo com o corpo do jogador, ele emitirá uma mensagem chamada 2, ao mesmo tempo, o objeto Vida receberá esta mensagem através do sensor *Message* e executa um atuador *Property* chamado "act". Este atuador tem a característica *Add*, sempre efetua uma adição na propriedade chamada *Text* e o valor é sempre -1(menos um), ou seja, uma contagem regressiva. O contador chegando a zero é acionando outro sensor chamado de sensor 1 que é um *Property*, ele está relacionado com a propriedade *Text* e quando *Text* estiver em zero o sensor "sensor 1"aciona um controlador do tipo AND ligado a um atuador, que encerra o jogo, Figura 50.

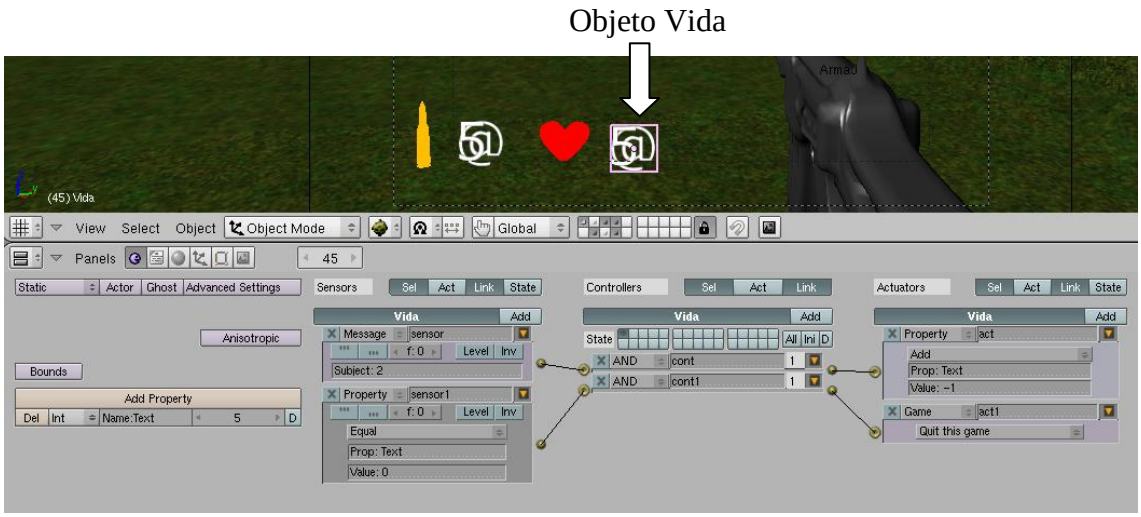


Figura 50 - Contador decrescente da vida.

Para contar a munição do jogador é criado em "*Add Property*" uma propriedade do tipo Int, inteiro, com nome *Text* ligado ao objeto Munição e com capacidade para 5 tiros, conforme a Figura 51. O primeiro sensor do objeto Munição é um *Message* e se chama 1 "*Subject:1*", o objeto que envia esta mensagem para ele é o vazio responsável pelo disparo do tiro, quando este ocorrer. Ao receber a mensagem do disparo o "*Subject :1*" executa um

atuador *Property* com característica *Add*, de somatório, e vinculado a propriedade *Text* com valor de -1 (menos um). Executando uma contagem regressiva na propriedade *Text* da munição. Ao mesmo tempo executa o som do disparo.

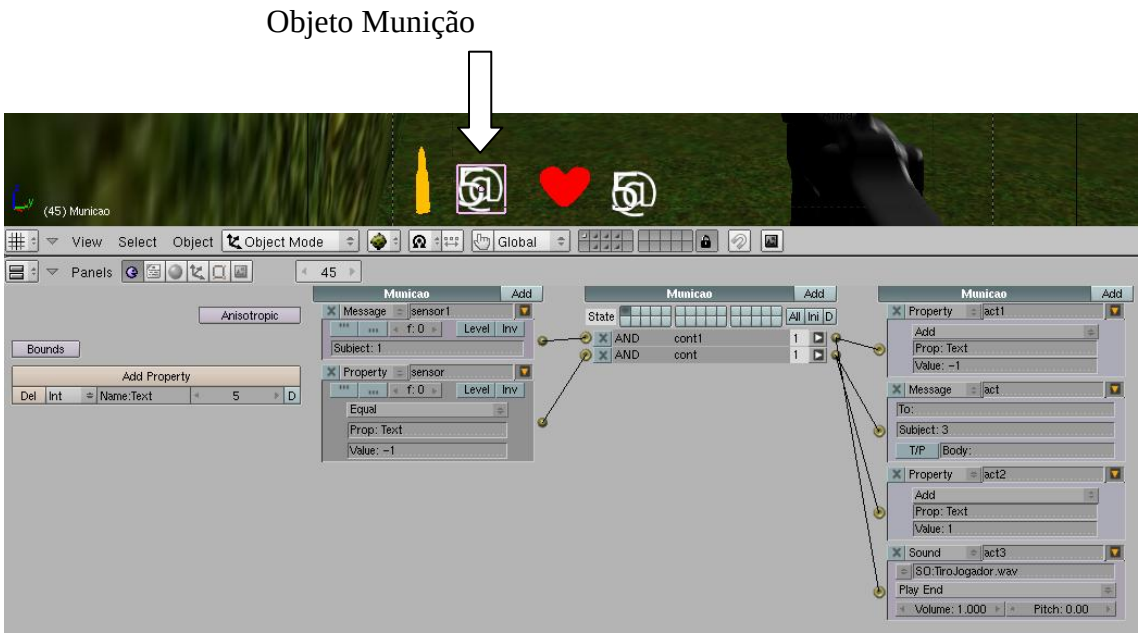


Figura 51 - Contador de munição

- Lógica dos menus

Os menus foram desenvolvidos a partir de objetos planos. O mapeamento *UV Calculation* foi aplicado em sua face e a visão do menu é de cima para baixo. Cada objeto possui um tipo de sensor, no caso do objeto Plane.001, com a imagem escrita "*New Game*", os sensores vinculados ao plano foram um Mouse para executar uma animação e outro mouse do tipo *Left Button* para acionar outra cena quando o botão esquerdo do mouse for acionado. O primeiro sensor Mouse é do tipo *Mouse over* para executar a animação apenas quando o mouse estiver sobre o objeto Plane.001. Como está na Figura 52, a ligação do sensor chamado "sensor" convergem para a animação e para o controlador do sensor "sensor1". Significa que para trocar de cena o mouse deve estar sobre o objeto e ser clicado com o botão esquerdo ao mesmo tempo. Após acionar o menu "New Game" a cena que aparece para o jogador é "Carregando", e esta aciona o jogo. O título não tem função lógica e os demais objetos do menu direcionam o jogador para cenas específicas.

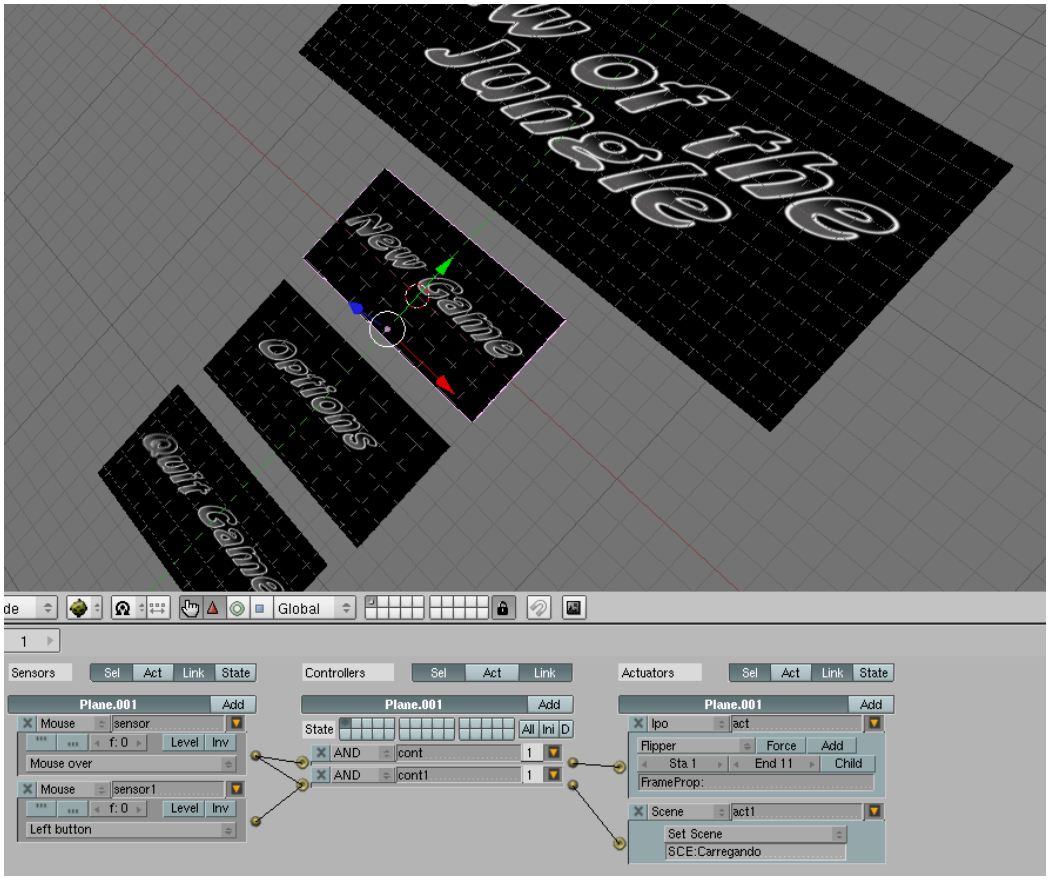


Figura 52 - Menu.

As cenas no Blender, Figura 53, servem para organizar as seções do projeto. No jogo a cena Menu Principal é onde se encontra o menu e onde se encontra o jogador antes de iniciar a partida. Escolhendo iniciar a partida ele é direcionado para a cena Carregando. Esta serve apenas como transição entre MenuPrincipal e o Jogo, Na verdade, a cena Carregando possui uma função estética, a imagem "Laoding" fica travada enquanto o software carrega outra cena.

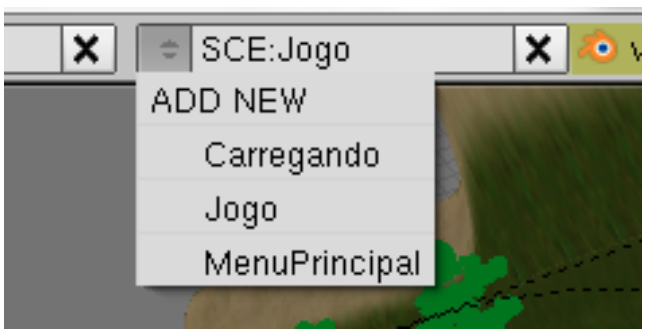


Figura 53 - Cenas

6.3.2 Disciplina testes

Testes efetuados na fase de construção, com código-fonte.

6.3.2.1 Desenvolvimento de caso de teste

Caso de teste: CT2
Cenário/Condição: Distancia em linha reta entre inimigo e jogador sem obstáculo.
Entradas: • Jogador próximo ao inimigo. • Jogador a uma distância média em relação ao inimigo. • Jogador longe do inimigo.
Resultado: • Em todos estes casos o inimigo encontra o jogador e anda em sua direção. • O algoritmo se mostra muito deficiente quando o jogador está a uma distância média ou longa do inimigo. Visualmente a ordenação da rota mais curta deixa o jogo travado, como se estivesse em câmera lenta. A ordenação utiliza muitos comandos "for". Este comando no jogo foi utilizado para procurar dados específicos em uma lista percorrendo ela ou então, ordenar números desordenados em outra lista, por exemplo: uma lista que recebe números aleatórios deve ser ordenada de forma crescente, ou então, deve ser encontrado um número em específico nesta lista. No algoritmo A* são criadas várias listas: uma para guardar valores temporários, outra para guardar valores permanentes, outras listas são criadas como pivô para serem utilizados na lista temporária e outras são criadas para recolher dados atualizados. O algoritmo construído utiliza muitos "for" percorrendo listas dentro de listas, com isto, o processador é ocupado exclusivamente com elas deixando as funções do jogo em um segundo plano. O jogo parece estar em câmera lenta porque a consulta das listas ocorre de forma contínua prejudicando a dinâmica do jogo. • Em distâncias curtas ocorriam o mesmo processo, porém, são poucos números dentro das listas envolvidos no "for". O tamanho das listas diminuiu em função de ser uma distância curta. Visualmente para o jogador o efeito de câmera lenta não era notado.

Tabela 6 - Caso de teste na fase de construção.

6.4 Fase distribuição

Na fase de distribuição, o software deverá se tornar operacional. Conforme (MARTINS, 2004), o software concebido para o mercado em sua versão de teste, beta, deve ser distribuído entre alguns clientes selecionados que irão avaliar o sistema. Se o software for criado para uma empresa, em específico, o sistema será instalado para alguns usuários. O objetivo da versão beta é verificar se as necessidades do usuário foram atendidas, verificar riscos não previstos, problemas não resolvidos e identificar dificuldades ao utilizar o sistema e a necessidade de treinamento. O foco é a transferência do produto para o usuário final, seu objetivo é: Avaliar a verão de teste, implantação do sistema com migrações de dados, treinamento do usuário, prover auto-suficiência e obter satisfação do cliente. Transferência do produto para o usuário e avaliar o projeto concluído.

6.4.1 Implantação

A implantação consiste em apenas desenvolver um arquivo executável para ser utilizado em qualquer computador com sistema operacional Windows. Ele é prático para o usuário porque sua instalação é fácil, basta ter os arquivos em seu computador e clicar no arquivo com extensão EXE e o jogo será inicializado.

6.4.1.1 Conclusão do sistema

Para construir este arquivo foi utilizada a função “*Save Runtime*” do Blender e de forma automática o arquivo executável é gerado. Para o jogo ser executado em qualquer computador com sistema operacional Windows foi criado uma pasta com o nome do jogo e dentro dela colocado o arquivo executável com os arquivos “pthreadVC2.dll”, “python26.dll” e o arquivo compactado zip python25, todos extraídos do Blender. Esta pasta pode ser instalada em qualquer computador e executado o jogo independente do Blender ou Python estar instalados na máquina.

7 CONCLUSÃO

Este trabalho demonstrou a construção de um jogo para computador utilizando tecnologia gratuita com a filosofia de software livre.

A inteligência artificial do jogo foi destacada através do algoritmo A*, responsável pelo inimigo procurar o jogador durante o jogo. Foi demonstrado o conceito, a descrição do algoritmo e a implantação no software.

O estudo sobre o que é atrativo em um jogo para o usuário também foi levado em conta. Como se trata de um protótipo, não foram desenvolvidas fases no jogo e nem desafios complexos. O que foi levado em consideração sobre a “jogabilidade” foi o estudo da interface, projetada para ser intuitiva e prática para o usuário. Os comandos foram projetados de forma simples e direta, baseado nos jogos já disponíveis no mercado.

A estrutura do projeto utilizando RUP foi adequada, pois as fases de desenvolvimento proporcionaram uma visão clara e objetiva do software. Em cada fase ficou evidente a evolução do projeto. Na fase inicial a coleta dos dados, fornecendo os objetos de como seria o jogo. Na fase de construção representando como passar estas idéias para o papel, de forma clara e em uma linguagem acessível para todos. Na fase de construção o desenvolvimento do software em si e na fase de distribuição a entrega do produto.

8 REFERÊNCIA

LARMAN, Craig. **Utilizando UML e Padrões**. Porto Alegre: Bookman, 2007.

RUSSELL, Stuart; PETER, Norvig**. Inteligência Artificial**. Rio de Janeiro: Elsevier, 2004.

LUTZ, Mark; ASCHER, David. **Aprendendo Python**. Porto Alegre: Bookman, 2007.

BRITO, Alan. **Blender 3D Guia do Usuário**. São Paulo: Novatec Editora, 2008.

SCHUYTEMA, Paul. **Design de Games um Abordagem Prática**. São Paulo: Cengage Learning, 2008.

MARTINS, José Carlos Cordeiro. **Gerenciando projetos de desenvolvimento de software com PMI, RUP e UML**. Rio de Janeiro: Brasport, 2004.

DEITEL, H. M. **Java como programar**. São Paulo: Pearson Prentice Hall, 2005.

GLOBO, Rede. Sony quer brasileiros na produção de jogos para o PlayStation. **G1**, out. 2009. Disponível em : *<http://g1.globo.com/Noticias/Games/0,,MUL1344366-9666,00-SONY+QUER+BRASILEIROS+NA+PRODUCAO+DE+JOGOS+PARA+O+PLAYSTATION.html.>* Acesso em: 26 set 2010.

CHANNEL, Discovery. **A Era do Videogame**, mai. 2007. Disponível em: <*http://www.discoverybrasil.com/videogame/*>. Acesso em: 29 set. 2010.

PIMENTA, Denize Terra. Tutorial JUDE. **Jude Comunity 5.1**, out. 2007. Disponível em*:<http:// www.assembla.com/spaces/senac-pos/documents/.../TutorialJUDE51.pdf*>. Acesso em: 29 set. 2010.

MILANI, André. Primeiro livro brasileiro sobre o GIMP ganha destaque. **Sisnema Informátia**, dez. 2005. Disponível em: <*http://sisnema.com.br/Materias/idmat016199.htm*>. Acesso em: 29 set. 2010.

CLUA, Esteban Walter Gonzalez BITTENCOURT João Ricardo. Desenvolvimento de jogo 3D: Concepção, design e programação. **Universidade do Vale do Rio dos Sinos,** jul. 2005. Disponível em:<*http://www.ic.uff.br/~esteban/files/Desenvolvimento%20de%20jogos%203D .pdf*>. Acesso em: 22 out. 2010.

PROCEDURALBASE. Hierarquia no Game Engine. **Blender Game Engine,** mai de 2007. Disponível em: <*http://game.proceduralbase.org/category/python-script/*>. Acesso em: 25 set. 2010.

WTHREEX. Rational Unified Process: Fases. **Rational Software Corporation,** versão 2002.05.00. Disponível em: <*http://www.wthreex.com/rup/portugues/index.htm*>. Acesso em: 25 set. 2010.

LESTER, Patrick. A* Pathfinding para Iniciantes. **Policyalmanac**, jun de 2005. Disponível em: <*http://www.policyalmanac.org/games/aStarTutorial_port.htm*>. Acesso em: 25 set. 2010.

DSENHEPINTE. Figura Humana – Proporções. **Aprenda Pintura e Desenho**, set de 2008. Disponível em: <*http://desenhepinte.blogspot.com/2008/09/figura-humana-propores.html*>. Acesso em: 02 out. 2010.

MOREIRA, Altamir. Proporção do rosto humano. **Arteamil**, jan de 2008. Disponível em: <*http://sites.google.com/site/artea1000/home22233*>. Acesso em: 02 out. 2010.

SOUSMAN. Ciclo de caminhada. **Sousman**, mar de 2008. Disponível em: <*http://sousman.blogspot.com/2008/03/ciclo-de-caminhada.html*>. Acesso em: 02 out. 2010.

TERRESTRES, Forças. Os principais fuzis em uso no mundo: Leia, comente e vote na enquete. **Forças Terrestres**, abr de 2009. <Disponível em: <*http://www.forte.jor.br/2009/04/21/principais-fuzis-em-uso-no-mundo/*>. Acesso em 02 out. 2010.