ALEXANDRE NASCIMENTO TOMOYOSE
LEANDRO GASPARI LENZ
MAURO KESSELMAN

DESENVOLVIMENTO DE UMA ENGINE 3D PARA JOGOS DO TIPO RPG

Projeto de Formatura apresentado à disciplina PCS 2502 – Laboratório de Projeto de Formatura II, da Escola Politécnica da Universidade de São Paulo.

São Paulo
2004

ALEXANDRE NASCIMENTO TOMOYOSE
LEANDRO GASPARI LENZ
MAURO KESSELMAN

DESENVOLVIMENTO DE UMA ENGINE 3D PARA JOGOS DO TIPO RPG

Monografia do Projeto de Formatura apresentada à Escola Politécnica da Universidade de São Paulo para conclusão do curso de graduação

Área de Concentração:
Engenharia da Computação

Orientador:
Prof. Doutor Romero Tori

Co-Orientador:
João Luiz Bernardes Jr.

São Paulo
2004

RESUMO

O projeto consiste do desenvolvimento de uma "engine" capaz de fornecer um conjunto de funcionalidades para a criação de um jogo de RPG (Role-Playing Game ou jogo de interpretação). A título de demonstração, a mesma será utilizada para a construção de um jogo. Para tanto, serão aplicados os conceitos de orientação a objetos, técnicas de modelagem de software, além de tecnologias de renderização em três dimensões. As ferramentas escolhidas para este desenvolvimento são a linguagem de programação Python e a "engine" gráfica PANDA 3D. A escolha foi feita baseada numa série de vantagens que esta dupla de ferramentas pode oferecer. Para a modelagem do software, foram utilizadas as ferramentas de UML (como diagramas de classe e estado, por exemplo). A descrição do jogo propriamente dito, teve como base um documento de Game Design, amplamente utilizado neste estilo de jogo (RPG).

ABSTRACT

The project consists of the development of an engine capable to supply a set of functionalities to help the creation of a RPG game (Role-Playing Game). For demonstration purposes, this engine will be used for the construction of a game. To achieve this, it will be applied concepts of object-oriented, techniques of software modeling and technologies of three dimension rendering. The tools chosen for this development were the programming language Python and the graphical engine PANDA 3D. For the software modeling, we´ve used the UML tools (Class diagrams and State diagrams, for example). For the description of the game itself, we´ve used a Game Design document, widely applied in this style of game (RPG).

SUMÁRIO

1. INTRODUÇÃO

1.1 Objetivo

Desenvolvimento de uma "engine" 3D especializada em jogos do tipo RPG, utilizando ferramentas "open source", e de um software de entretenimento 3D para avaliação técnica e funcional da "engine" criada.

1.2 Motivação

A indústria de jogos é uma das áreas da informática que mais têm crescido e para acompanhar este movimento mundial é necessário promover o desenvolvimento deste tipo de aplicação nacionalmente.

Com este projeto, buscamos desenvolver uma referência no processo de criação de "engines", somado o desenvolvimento de jogos apoiados nesses recursos, integrando técnicas de Engenharia de Software e de design de jogos, gerando-se assim novos conhecimentos na área de projeto de software para entretenimento 3D.

1.3 Organização

O capítulo 1 mostra uma introdução ao trabalho, indicando seu propósito e o motivo do mesmo.

No capítulo 2, apresentamos alguns conceitos sobre RPG´s e sua evolução do papel ao mundo eletrônico.

No capítulo 3, introduzimos informações de caráter técnico envolvendo ferramentas de apoio ao desenvolvimento de jogos como as "engines".

No capítulo 4 encontra-se a descrição do projeto em duas etapas: a "engine" e o jogo. Além disso, encontram-se também os recursos necessários para o desenvolvimento do projeto.

O capítulo 5 aborda os métodos utilizados no desenvolvimento do projeto, incluindo a pesquisa dos mesmos.

No capítulo 6 encontra-se a descrição das etapas de desenvolvimento do

projeto, ou seja, sua implementação, testes e avaliação.

O capítulo 7 apresenta os comentários finais do grupo a respeito do trabalho como um todo, os processos, as ferramentas, o aprendizado, as dificuldades, o que deu certo, o que saiu errado e o porquê.

Como Anexos, estão localizados os documentos de apoio ao entendimento deste trabalho, como por exemplo, o documento de Game Design.

Na última seção está a lista de referências utilizadas durante o trabalho.

2. ROLE PLAYING GAME

RPG (Role Playing Game) é um tipo de jogo onde o jogador assume o papel de um ou mais personagens, geralmente algum tipo de herói. Durante o decorrer do jogo, o personagem deverá vencer desafios como, na maior parte do tempo, derrotar inimigos. [1]

Uma das principais características desse tipo de jogo é o roteiro elaborado. Todos os RPG's seguem o fluxo de alguma história e acompanhar o desenrolar da história é justamente uma das principais motivações dos jogadores. Outra característica importante é a evolução do personagem, que normalmente desenvolve atributos físicos, mentais e habilidades a partir da experiência ganha durante o jogo.

As raízes do RPG datam do final da década de 60, início de 70. Baseado no livro de J.R.R. Tolkien, "The Lord of the Rings", foi criado um novo modelo de jogo de guerra. Neste novo jogo, os participantes vivenciariam o mundo criado por Tolkien, onde humanos, elfos e anões travavam batalhas monumentais contra orcs, mercenários e demônios. [2]

O primeiro produto comercial que utilizou esta nova idéia para jogos foi o "Dungeon & Dragons". O objetivo neste jogo era explorar cavernas e labirintos com seu personagem, superando os obstáculos, especialmente inimigos, que surgissem em seu caminho. O jogo estipulava regras para a criação dos personagens e para definir o resultado de um combate.

A partir do final da década de 70, houve uma grande popularização deste modelo de jogo e muitos outros produtos similares ao "Dungeons & Dragons" passaram a ser comercializados. Com o passar dos anos, o RPG deixou de estar atrelado à ambientação criada por Tolkien, ficando apenas a essência do jogo, que é o jogador controlar um personagem no seu dia-a-dia em um mundo imaginário.

Isto marcou a aparição de jogos de RPG baseados nas mais criativas ambientações, de vampiros e lobisomens a naves espaciais e conquista de novos planetas. Entretanto, continuavam usando dados e folhas de papel com fichas técnicas dos personagens, para acompanhar o desenrolar da história. Alguns até continham ainda objetos como tabuleiros e miniaturas, o que deixava o jogo mais realista.

A partir da década de 80, a combinação de computadores e RPG foi inaugurada. Embora as limitações tecnológicas da época impedissem algo mais elaborado, alguns dos clássicos nasceram nesse período, um deles é a série "Ultima". Iniciada com o "Ultima I" de 1980, esta série possui além de oito continuações diretas, variações da história principal como "Ultima Underworld" e também uma versão online através da internet, o "Ultima Online".

Na sua encarnação eletrônica os RPG´s são jogos que não exigem rapidez com um joystick, mas sim paciência e raciocínio para resolver problemas lógicos. Tradicionalmente, a visão do seu personagem é através de uma câmera aérea e as escolhas são realizadas através da seleção de itens em menus.

Variações deste estilo consagrado são os RPG´s com visão em primeira pessoa ou então com combates em tempo real, onde além de raciocínio é exigido do jogador habilidade em manipular um joystick.

3. TECNOLOGIA

Na época em que o DOS era o sistema operacional mais utilizado nos microcomputadores, os desenvolvedores de jogos tinham acesso direto aos recursos de "hardware" da máquina. Assim era possível ordenar as diferentes partes do "hardware", como placas de som, de vídeo e dispositivos de entrada, a fazerem exatamente o que era necessário. [3]

Após alguns anos, essa aparente vantagem de controlar diretamente o "hardware" se tornou um pesadelo. O aumento exponencial das configurações de microcomputadores, com peças dos mais variados fabricantes, tornou o tempo necessário para programar o suporte de toda esta diversidade de "hardware", maior que o tempo trabalhando no desenvolvimento do jogo em si.

O surgimento dos sistemas operacionais de mais alto nível, que administravam os recursos da máquina automaticamente, motivou a criação de uma camada que deixasse o acesso aos dispositivos transparentes ao programador. Estes componentes que abstraíam o acesso à máquina foram chamados de "engines".

Portanto, as "engines" são módulos prontos que automatizam determinadas funcionalidades que são freqüentemente usadas em jogos, reduzindo o tempo necessário para a produção de um novo jogo.

Com o passar do tempo, se fez necessário subdividir ainda mais esse módulo, marcando a divisão de "engines" de baixo e alto nível. As de alto nível são as mais próximas do programador e disponibilizam para este os métodos de acesso de que precisam para desenvolver o jogo, enquanto que as de baixo nível, que serão referenciadas como "graphics engines" no decorrer deste documento, consistem dos módulos que formam uma camada mais próxima da máquina, comunicando com os seus recursos de hardware.

As "graphics engines" formam uma interface entre a máquina e as "engines" de alto nível, facilitando o desenvolvimento das últimas. As "graphics engines" fazem uso de uma camada ainda mais próxima da máquina, bibliotecas que fazem as requisições de "hardware" diretamente ao sistema operacional, como o DirectX [4] e o OpenGL [5]. Ambos contam com várias funções de acesso, como métodos para renderizar modelos 3D ou alterar a iluminação do ambiente.

No caso do DirectX, as funções gráficas são apenas parte de suas funcionalidades, fora estas ele oferece meios para utilização de outros recursos de hardware como controle de som e até de periféricos como joystick e mouse. A tecnologia DirectX teve sua primeira versão em 1995 e possui como desvantagem suportar apenas as plataformas com sistema operacional Microsoft.

O OpenGL é um padrão criado em 1992 e consiste de um ambiente de desenvolvimento para aplicativos 2D e 3D. Ele possui diversas funções para renderizar modelos e utilizar efeitos especiais, mas seu ponto forte é a presença nas mais variadas plataformas.

Além deste aspecto das "engines", outra tecnologia que estamos usando em nosso projeto é o XML [6] (*Extensible Markup Language*), que se trata de uma espécie de "meta-linguagem" para descrever de forma estruturada e padronizada qualquer tipo de dado. Sua primeira versão data de 1998.

Uma característica importante deste padrão é a sua independência de plataforma. Após ser definida como será a estrutura de armazenamento em XML de uma determinada informação, não é necessário realizar nenhuma modificação para exportá-la a outra plataforma. A principal razão para esta grande compatibilidade é a descrição da informação em XML consistir simplesmente de um arquivo texto, escrito no padrão especificado pela tecnologia.

O XML é uma derivação do SGML, que é um formato padronizado pela ISO em 1986. O SGML é uma linguagem descritiva que teve como motivador para sua criação, a necessidade de um padrão de armazenamento de informação que não fosse interpretável apenas para as máquinas, mas também para as pessoas. [7]

A segunda característica importante do XML é esta facilidade de ser compreendida pelo ser humano. Assim, mesmo sem preparar uma interface mais amigável, o usuário consegue ler e modificar as informações diretamente no arquivo XML.

4. ESPECIFICAÇÃO DO PROJETO

O projeto possui duas partes: a primeira é a construção da "engine" e, a segunda, a construção do jogo para avaliar as funcionalidades da "engine".

Portanto, iremos definir ambos, separadamente.

4.1 Especificação da "Engine"

A "engine" do projeto vem a ser um pacote com módulos funcionais, sendo que estas funções são aquelas que fornecerão todo o tipo de controle que um jogo de RPG tradicional (comandos através de menus) necessita.

A seguir, podemos citar algumas funções genéricas que são necessárias: inserção e exclusão de objetos 3D na tela, inserção e exclusão de textos na tela, movimentação de objetos na tela, menus de comando, controle de som, controle de câmera, cálculos de combate e evolução de personagem, controle de animações, etc.

Como o nosso objetivo é permitir que outras pessoas possam utilizar nossa "engine" com relativa facilidade, algumas dessas funções possuem parâmetros que podem ser facilmente editáveis, pois foram construídos seguindo um padrão determinado. No nosso caso, utilizamos arquivos no formato XML para fornecer estes parâmetros.

Portanto, para inserir novas características ao jogo, basta editar o arquivo XML onde estão contidas as informações da característica que desejamos modificar do jogo.

4.2 Especificação do Jogo

Trata-se de um jogo de RPG (Role-Playing Game), no qual o usuário tem uma visão de terceira pessoa, ou seja, ele observa o cenário e os personagens de uma posição superior, como uma câmera aérea. O jogo foi dimensionado para apenas um jogador.

Os gráficos utilizados possuem, em sua maioria, três dimensões. A interface do usuário com o jogo será feita inicialmente através do teclado, podendo evoluir

para outras formas de interface homem-máquina em futuras versões. Serão atribuídas funções específicas para cada tecla, como movimentação (para cima, para baixo, para direita, para esquerda), acionamento de menus, confirmação e cancelamento de ações. Esta configuração será muito apropriada no caso da inserção de um joystick, tornando o jogo facilmente portável para console. No caso da presença de mais de um personagem controlado pelo jogador, os outros seguirão aquele que estiver sendo diretamente movimentado pelo jogador.

O combate, realizado através de menus, acontecerá com inimigos colocados em lugares pré-estabelecidos. Os inimigos atacarão segundo um padrão especificado na Inteligência Artificial do jogo.

Para maiores detalhes do jogo (história, personagens, evolução, monstros, itens, equipamentos, músicas, cenários, etc...), consulte o documento de "Game Design" [8], no Anexo A.

4.3 Recursos e Infra-Estrutura

Para o desenvolvimento deste projeto foram utilizados os seguintes recursos: microcomputador com processador Pentium 3 de 500Mhz, 256 MB de RAM, 32 MB de Vídeo e uma placa aceleradora 3D compatível com OpenGL ou DirectX. O Sistema Operacional utilizado foi o Windows XP. Para o desenvolvimento da "engine" foi utilizada uma "graphics engine" aberta de nível inferior, dedicada a modelagem 3D denominada Panda3D [9], que suporta tanto o padrão DirectX como o OpenGL.

5. METODOLOGIA

5.1 Pesquisa

Na primeira etapa de pesquisa, nosso objetivo foi buscar a alternativa mais viável para realizar este projeto, no período de um ano. Buscamos, então, linguagens interpretadas, que nos forneceriam resultados mais rápidos do que linguagens compiladas. As linguagens interpretadas são normalmente escolhidas para a criação de protótipos, categoria esta em que se enquadra nosso projeto.

O passo seguinte foi pesquisar "engines" 3D de fácil manipulação e que pudessem ser utilizadas a partir de uma das linguagens selecionadas anteriormente.

Para finalizar a nossa pesquisa, buscamos métodos já existentes de desenvolvimento de jogos, tanto do ponto de vista de software quanto do ponto de vista de jogo propriamente dito, também conhecido como "Game Design".

5.2 Game Design (jogo)

O "Game Design", documento empregado em jogos de RPG, consiste na descrição das funções do jogo, cálculos utilizados, comportamento do usuário, interface, cenários, personagens, história, sons, "engines" utilizadas pelo jogo, etc. Estas informações são agrupadas em categorias específicas como Arte, Som e História, por exemplo, de tal forma que os responsáveis por estas áreas saibam exatamente onde procurar as definições que eles deverão seguir. Este documento vai sendo preenchido à medida que o projeto avança, possuindo no fim uma especificação completa. Este documento encontra-se no Anexo A.

5.3 Engenharia de Software ("engine")

Durante o nosso curso na faculdade, tivemos uma experiência significativa na parte de projeto de software. Podemos dizer que, neste trabalho, aplicamos os conhecimentos obtidos no curso. A metodologia empregada não possui uma nomenclatura própria, porém pode ser descrita através de seus passos.

Como primeira etapa, mapeamos o sistema indagando quais são as tarefas que ele deverá cumprir, sem maiores detalhes, visando suas funções principais e suas interfaces, ou seja, os outros sistemas com os quais ele se comunicará. A seguir, são listadas e detalhadas cada uma das funções do sistema, indicando os requisitos necessários para ser iniciada, quem a executa e qual seu resultado sobre o sistema. A partir dessas descrições, pode-se identificar as entidades fundamentais ao sistema, responsáveis por iniciar funções, sofrer a ação delas ou apenas fornecer ferramentas para que outros possam utilizá-las. Essas entidades são as Classes do sistema. Uma vez de posse das classes, faz-se necessário mapear os relacionamentos entre as mesmas, suas características e operações.

Para efetuarmos os testes, utilizamos um programa básico, um tutorial, que acompanha a "engine" gráfica. Isso facilitava nosso trabalho pois não havia a necessidade de criarmos um programa específico para determinado teste. O tutorial era amplo, permitindo explorar muitas das características da "engine" desenvolvida sem grandes ajustes.

Como auxílio a este desenvolvimento, foram utilizadas as ferramentas UML [10], como diagramas (fluxo de dados, classes, estado) e lista de casos de uso.

O Diagrama de Classes encontra-se no Anexo B.

6. PROJETO

Seguindo a metodologia descrita anteriormente, demos início ao projeto da "engine". De posse das ações que o sistema iria executar (casos de uso), concluímos quais deveriam ser as classes a serem implementadas. Dentre os tipos, podemos citar as classes de comunicação direta com o Panda3D, classes de negócios, responsáveis pelas funcionalidades do jogo, e algumas classes abstratas, que serviriam apenas como "caixa de ferramentas", possuindo alguns métodos de uso interno. As ferramentas UML foram de grande auxilio nesta etapa. Para maiores detalhes, ver Anexo B.

Após esta definição de classes, partimos para a etapa de implementação.

6.1 Implementação

Para iniciarmos a implementação, foi decidido que a "engine" 3D a ser utilizada no projeto seria o Panda3D, devido a sua fácil utilização e o seu código ser aberto, o que possibilita que outras pessoas possam contribuir para o seu desenvolvimento. O Panda3D disponibiliza todos os recursos que julgávamos necessários, como a renderização e manipulação de modelos na tela, suporte a música ambiente e efeitos sonoros, exibição de imagens 2D, entre outros. Outra vantagem é sua versatilidade, já que o Panda3D trabalha tanto com o Python [11], que é uma linguagem interpretada, própria para prototipação, quanto com o C++, que é compilada. Para este projeto, o C++, por se tratar de uma linguagem mais complexa e com muitos recursos que seriam sub-utilizados, não seria a escolha mais adequada.

Portanto, utilizamos o Python, que é uma linguagem orientada a objetos, interpretada e muito semelhante a um "script", o que torna fácil seu manuseio e rápido os testes e a depuração.

Iniciamos o nosso trabalho pelas classes básicas, ou seja, aquelas que seriam responsáveis pelo manuseio direto do Panda3D. São elas: SoundManager (controle do som), TextManager (controle de textos na tela), 3Dmanager (controle de modelos 3D na tela) e Câmera (posicionamento e direção da camera). O Diagrama de Classes

encontra-se no Anexo B.

Terminada a implementação das classes básicas, partimos para a implementação de algumas classes de negócios primárias, como ActiveCharacter e sua classe pai SuperModel, Menu, Movement, Enviroment e Battle, sendo que a classe Stage foi simulada dentro da classe Main. Em seguida, foram as classes de apoio como EffectManager, IA, InteractionHandler, etc. Para finalizar, foram criadas as classes abstratas SuperMain e SuperStage, cuja função é apenas fornecer métodos para suas classes filhas.

A "engine" deveria ser personalizável, ou seja, estar preparada para receber informações variadas, e não só aquelas com respeito ao nosso jogo. A intenção é que outras pessoas pudessem usar esta "engine" para criar seus próprios jogos de RPG. Para tanto, definimos como formato padrão da "engine" o XML (um arquivo texto contendo dados estruturados). Sendo assim, basta que novos arquivos XML sejam criados, contendo descrição de personagens, itens, modelos 3D, etc, para que seja possível criar um novo jogo. A especificação desse arquivo encontra-se no Anexo D.

6.2 Testes e Avaliação

O teste padrão era efetuado num programa chamado "tut", derivado do tutorial inicial do Panda3D. Para cada nova funcionalidade a ser testada, associávamos a ela uma tecla e disparávamos esta tecla durante a execução do programa.

Por se tratar de uma "engine" para jogos, o software possui uma infinidade de testes possíveis. Para exemplificar, citamos a seguir alguns dos testes realizados para verificar o funcionamento da "engine". Outros testes que efetuamos estão relacionados com: inserção dinâmica de modelos, movimentação dos personagens, posicionamento da câmera, reprodução de músicas e efeitos sonoros, utilização dos menus, visualização de textos, inteligência artificial, etc.

Colisões: um dos aspectos mais importantes nas colisões são os sólidos colocados ao redor dos modelos 3D. O Panda3D trabalha com dois tipos de sólidos: esferas e tubos (parecidos com comprimidos) O primcipal teste feito era o de colocar

sólidos ao redor dos personagens e movimentá-los um contra o outro para se chocarem e verificar o efeito. Por diversas vezes os personagens se atravessavam pois os sólidos de colisão não estavam configurados corretamente. Para isso era necessário definir uma máscara. A máscara funciona como um código, especificando para um determinado sólido com quais outros sólidos ele pode se chocar.

Teste de performance: para dimensionarmos o impacto dos modelos 3D renderizados na tela, desenvolvemos um procedimento da seguinte maneira: um algoritmo obtia o valor do frame rate do programa em execução, e outro inseria um modelo na tela ao pressionarmos uma determinada tecla. Observando estas variações e o efeito final na tela, concluímos que o valor máximo aceitável de modelos simultâneos na tela ficava em torno de 15. Este número desconsidera outras características que poderiam estar em uso ao mesmo tempo, como o menu. Neste caso, o número caía pela metade.

Teste de entrada em batalha: no início, forçávamos a entrada na batalha através de uma tecla. Depois, alteramos a engine para que a batalha fosse iniciada por uma área determinada no mapa. Assim, quando o personagem se movimentasse e atingisse esta área, a batalha era iniciada.

Teste de batalha: a batalha consiste na repetição de um ciclo que envolve: escolha da ação, execução da ação, resultados da ação. Este ciclo se repete para cada personagem que participa da batalha. Para que a batalha termine, é necessário que todos os personagens de um lado (aliados ou inimigos) estejam sem energia (mortos). Um exemplo de ação: um ataque simples de um personagem que causa a alteração do valor de energia de outro personagem. Esta variação era baseada em valores específicos do personagem que ataca e do personagem que sofre o ataque. Todos estes valores estavam previamente definidos no arquivo XML correspondente. Terminado o ataque, passava-se a vez para o próximo personagem fazer sua ação e assim sucessivamente até o fim da batalha.

Teste da funcionalidade dos Itens: os itens possuem dois atributos principais,

o efeito que ele causa e o alvo deste efeito. Ambos são definidos no arquivo XML relativo aos itens. Uma vez no modo de batalha, ativávamos o item através do menu e selecionávamos o personagem alvo. Através das informações do jogo veiculadas na tela (personagens vivos, energia, etc.), verificávamos se o efeito final correspondia com aquele especificado no arquivo XML.

7. CONSIDERAÇÕES FINAIS

Neste projeto, obtivemos um aprendizado considerável. Além do aprendizado de uma nova linguagem e uma "engine" gráfica, existem outros aspectos que merecem ser considerados.

Constatamos como o planejamento de um projeto é essencial para o mesmo. Sempre trabalhamos com prazos e tarefas reais, ou seja, que podiam ser concretizados. Uma prova disto é que nosso cronograma foi criado em Fevereiro de 2004 e sofreu apenas pequenas alterações nos meses de Outubro e Novembro.

Outro fator importante foi o Diagrama de Classes. Apesar de ensinado pelos professores do curso, foi no projeto que pudemos observar sua real importância. Nos preocupamos em construí-lo de forma a mantê-lo o mais desconectado possível, facilitando o processo de depuração.

Existe uma peculiaridade neste projeto. O software desenvolvido ("engine") não é um software final, mas sim uma plataforma de desenvolvimento, a qual será utilizada na criação de outro software. Por isso, foi importante que tornássemos seu manuseio o mais transparente possível. O criador de jogos, possível usuário da nossa "engine", não deve se preocupar com aspectos da implementação interna da "engine", apenas com a forma de usá-la.

Uma das dificuldades encontradas foi o fato da "engine" gráfica ser relativamente nova, ou seja, sua documentação era precária e os meios de comunicação com os responsáveis eram ineficientes.

Nossa maior dificuldade foi trabalhar com a equipe de designers da Faculdade de Arquitetura e Urbanismo (FAU). Apesar da grande empolgação demonstrada por eles no início do projeto, seu compromisso com o mesmo foi mínimo, senão nulo. De uma equipe de mais de dez pessoas, apenas um deles realmente cumpriu o acordo e nos ajudou até os últimos dias de trabalho. Para suprir esta falta, nos vimos obrigados a buscar auxílio em outros lugares, como a Internet. Felizmente, conseguimos a ajuda de outras pessoas, algumas até de outros países. Porém, esta ajuda não foi capaz de suprir nossas necessidades e, portanto, fomos obrigados a reduzir em cerca de 90% o jogo que seria utilizado como teste da "engine".

Uma melhoria do nosso projeto seria a concretização do sistema de técnicas e

mágicas dos personagens, que foi iniciada mas, devido à nossa falta de recursos artísticos, não pode ser colocada em prática.

Um fator que não foi explorado no nosso projeto e que poderia ser implementado é a movimentação no eixo "Z", ou seja, relevos com subidas e descidas.

Avaliando os resultados obtidos nos testes como um todo, as funcionalidades que foram definidas no início do projeto estão implementadas e apresentaram os resultados esperados.

ANEXO A

Game Design

Livro Mestre

Introdução

Este documento tem por objetivo descrever as características do software, sob o ponto de vista de um jogo de RPG.

Ele está dividido em seis seções, ou livros, cada um responsável por um aspecto específico do jogo. Estes livros vão apresentar algumas definições sobre a história, os gráficos, as músicas, o hardware, ou seja, todos os aspectos necessários para definir um jogo e que serão úteis no processo de desenvolvimento do mesmo.

Idéia

O objetivo deste jogo é demonstrar algumas das capacidades do "engine" desenvolvido como trabalho de formatura, servindo de exemplo para o mesmo, assim como proporcionar uma aventura em três dimensões com criaturas, itens mágicos, habilidades especiais e batalhas. Através da evolução do personagem, o jogador poderá experimentar novas técnicas, novos lugares e inimigos mais fortes no caminho para atingir seu objetivo. Contará também com a ajuda de personagens comuns, moradores das cidades e vilas do mundo, que fornecerão dicas valiosas para o prosseguimento do jogo.

Resumo da História

A história se inicia com a morte trágica de um garoto, aparentemente sem explicação, e que no desenrolar dos acontecimentos, descobre seu destino: salvar o mundo dos monstros que ameaçam invadi-lo. Para isto, deverá aprender a controlar seu poder, ainda desconhecido, com a ajuda de alguns amigos deste e do outro mundo.

Introdução aos Personagens

- "Alan Greiner" : Herói da história, descendente de uma linhagem de salvadores do mundo.

- "Guedhaia" : Guerreira mística, vive em um lugar afastado no mundo dos mortos. Tem grandes conhecimentos que podem ajudar o herói na sua busca.

- "Velha" : Aquela que guiará o herói da historia em sua jornada.

Descrição do Jogo

O jogo fornece uma interface 3D para o usuário, incluindo personagens, inimigos, cenário, mapa, etc... . O controle da movimentação dos personagens é feito através do teclado. As diferentes funções dos personagens são acionadas através de dois menus, um genérico e outro específico para batalhas. Neste menu genérico, o jogador tem acesso às informações de seus personagens (status, equipamentos, técnicas, entre outros), aos itens possuídos, etc. No menu de batalha, as opções são voltadas para a mesma: ataque, técnica, item, fuga. Dependendo do personagem, existirão outras opções neste menu. Os monstros encontram-se espalhados por todo o mundo, podendo ser encontrados a qualquer momento.

Elementos do Jogo

Controle:

Personagem 1: teclado;
Demais Personagens: seguem automaticamente o personagem 1, em fila;
Menus: teclado;

Inimigos:

Mapa: aleatório;
Labirintos: pré-estabelecidos;

IA:

Inimigos comuns: aleatório
Inimigos principais ("chefes"): segundo padrão ainda não estabelecido (por exemplo, atacar aquele de menor nível, ou de menor energia, etc.)

Menu:

Genérico: status, item, equipamento, técnica, salvar
Batalha: ataque, técnica, item, especial, fuga

Personagens:

Alan Greiner, Ghedaia, Velha

Especificações de Hardware

Mínimo:
- P3 500 MHz, 256 MB de RAM, placa aceleradora 3D com 32 MB.

Recomendado:
- P4 1.6 GHz, 512 MB de RAM, placa aceleradora 3D com 64 MB.

Cronograma

Modelagem inicial pronta até 05/05.
Engine do jogo pronta até final de agosto

Conteúdo acrescentado até final de outubro.
Pronto e sem erros até final novembro.

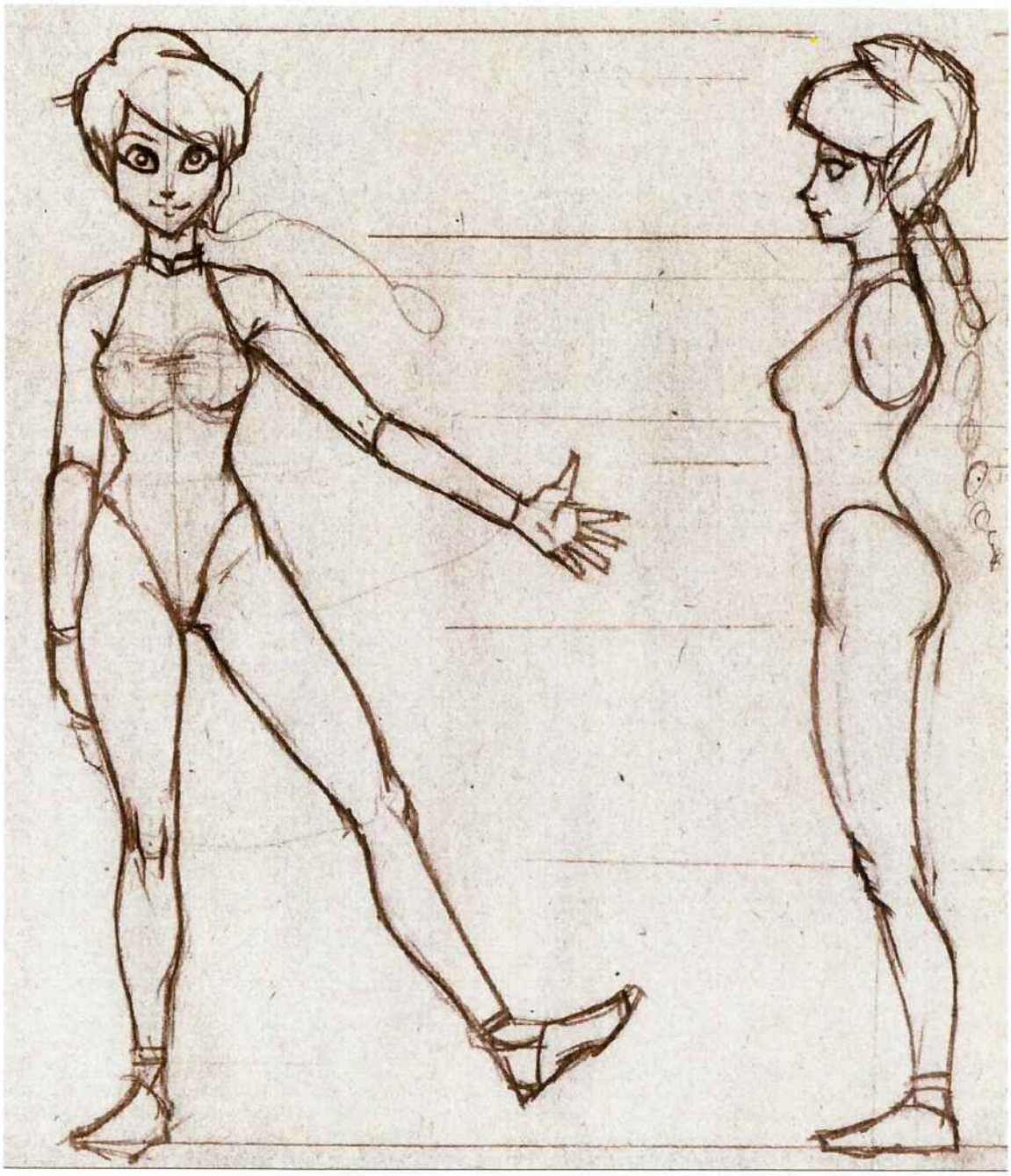
Membros da Equipe

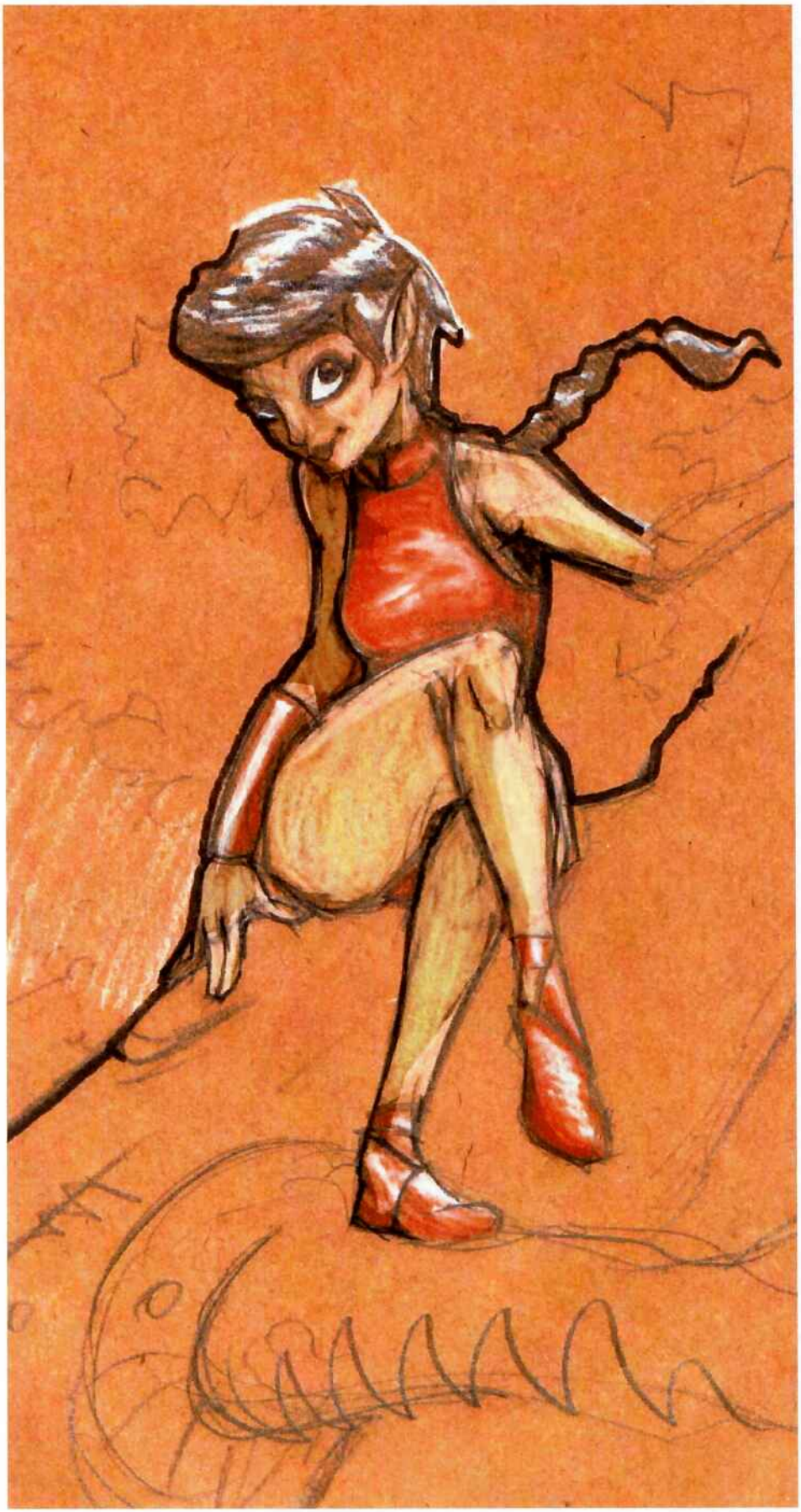
Projetistas e Programadores
- Alexandre Nascimento Tomoyose
- Leandro Gaspari Lenz
- Mauro Kesselman

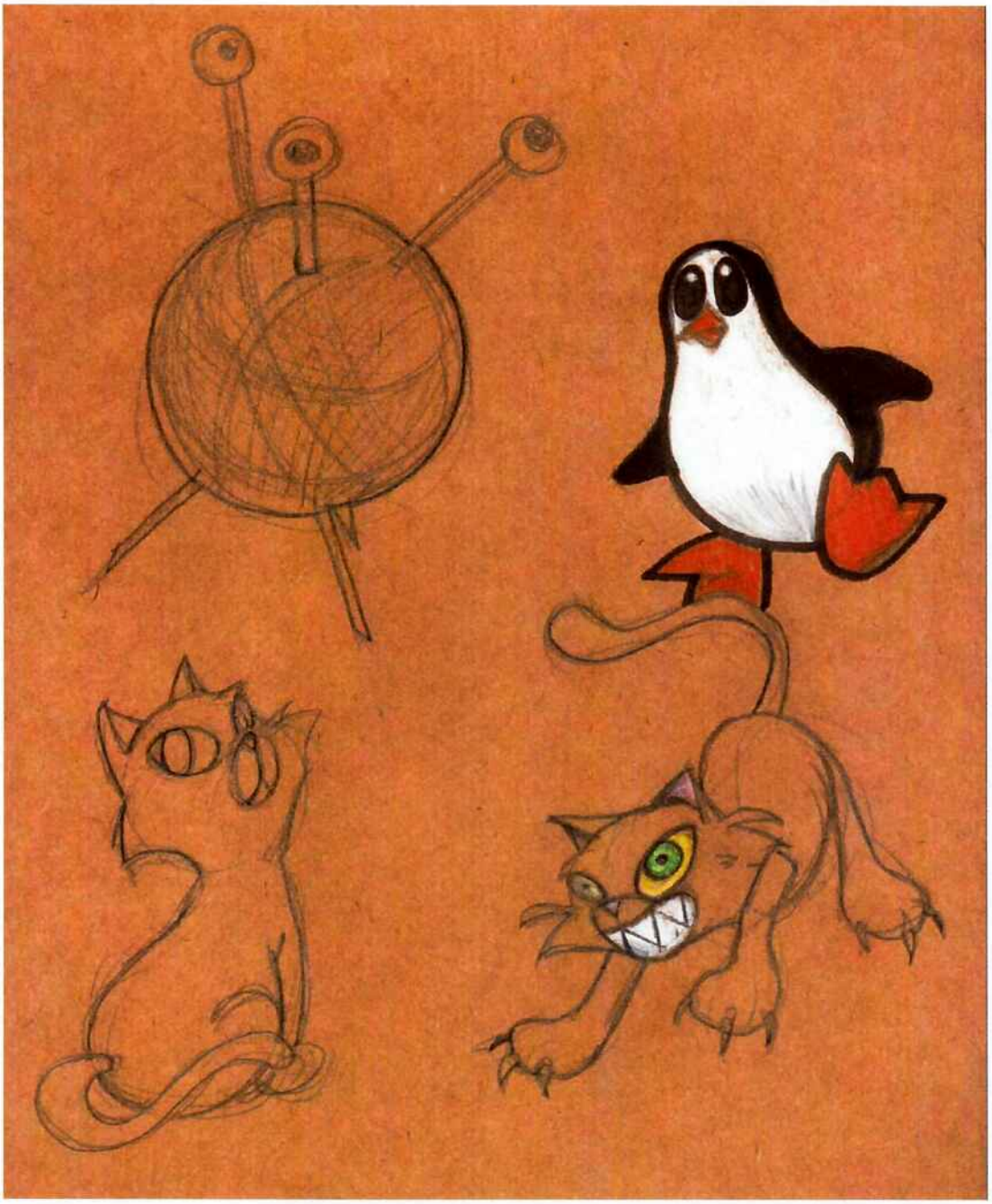
Designers Gráficos
- Rodrigo Degani e sua equipe

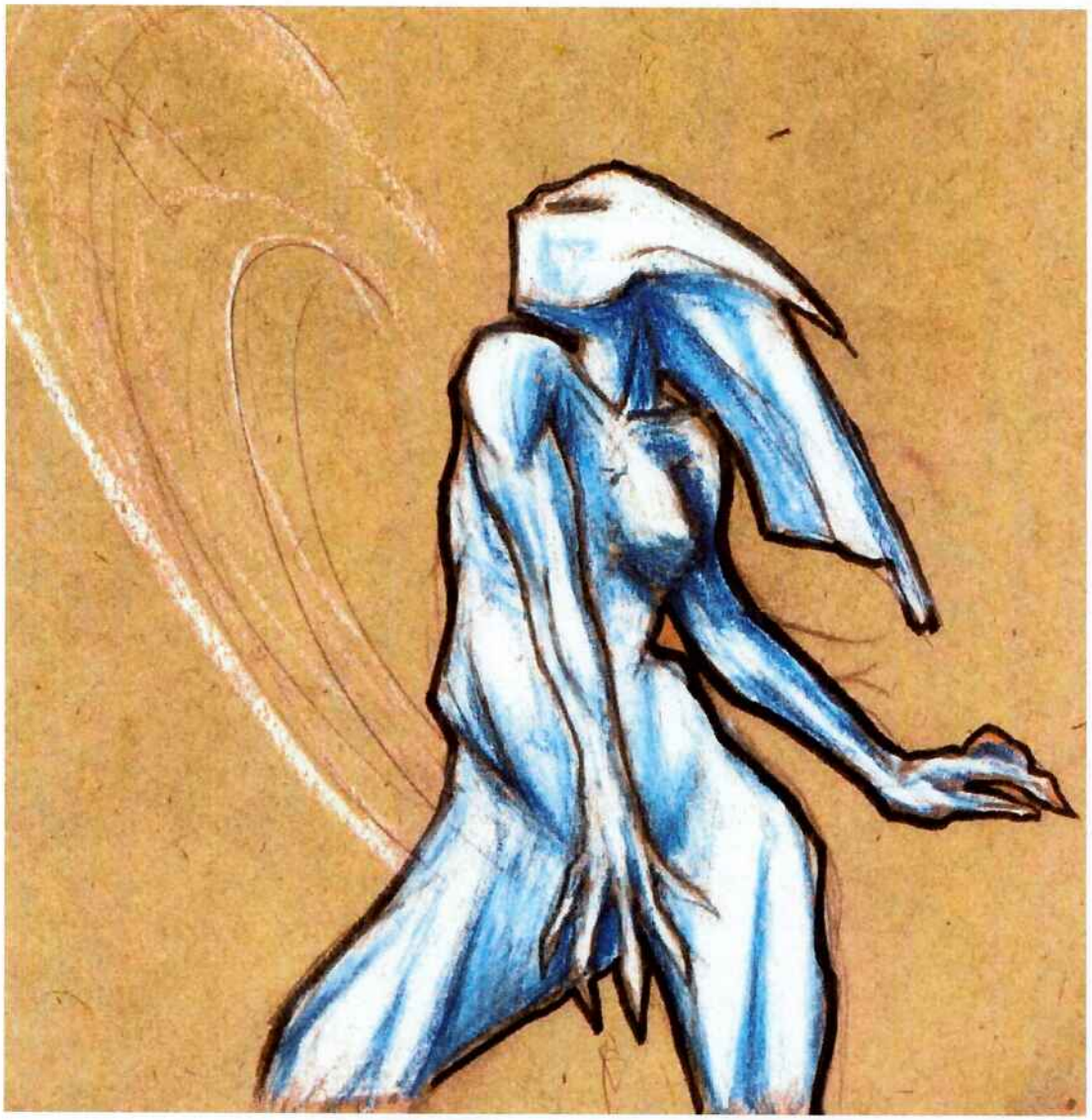
Revisores
- Equipe do INTERLAB

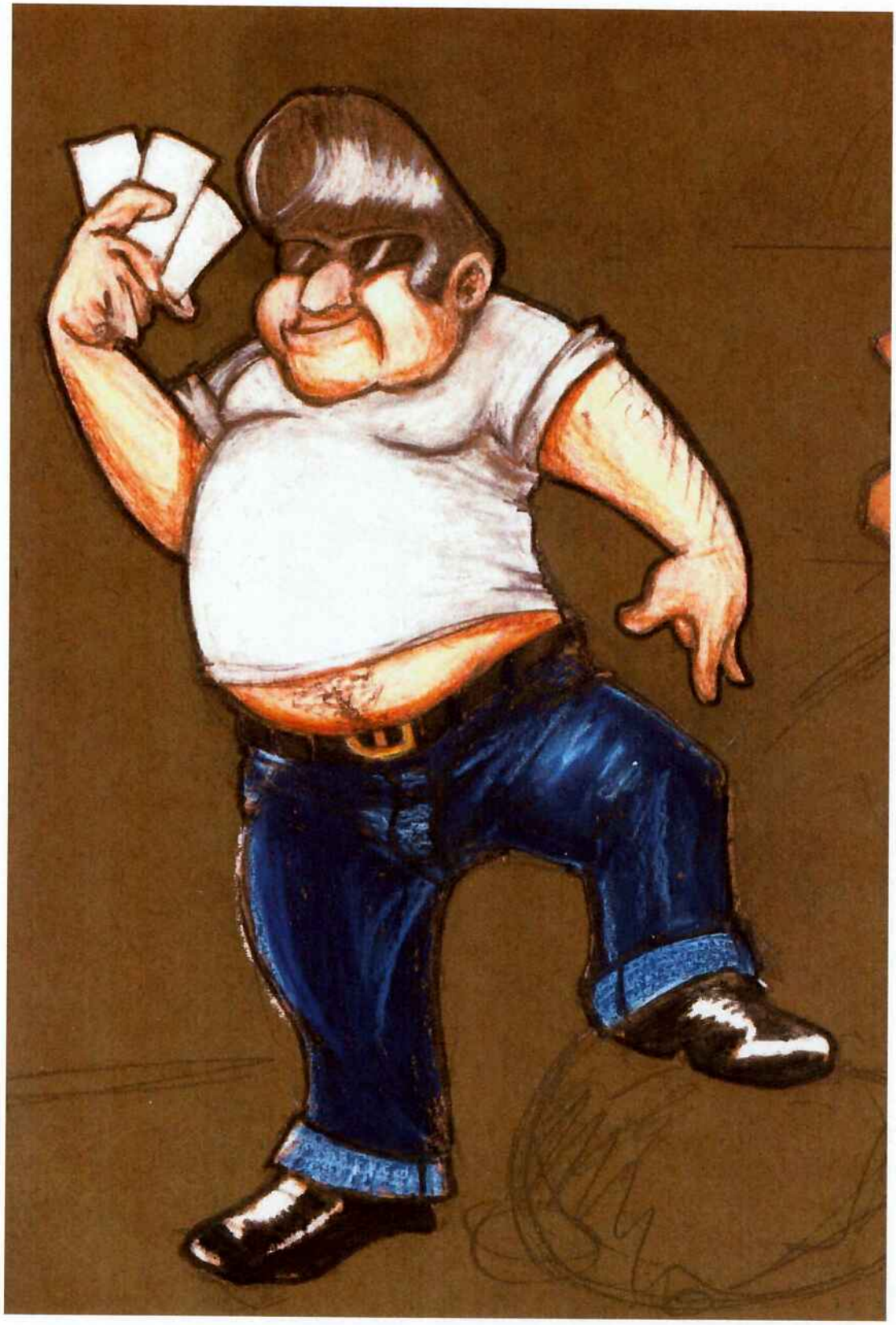
Livro de Arte

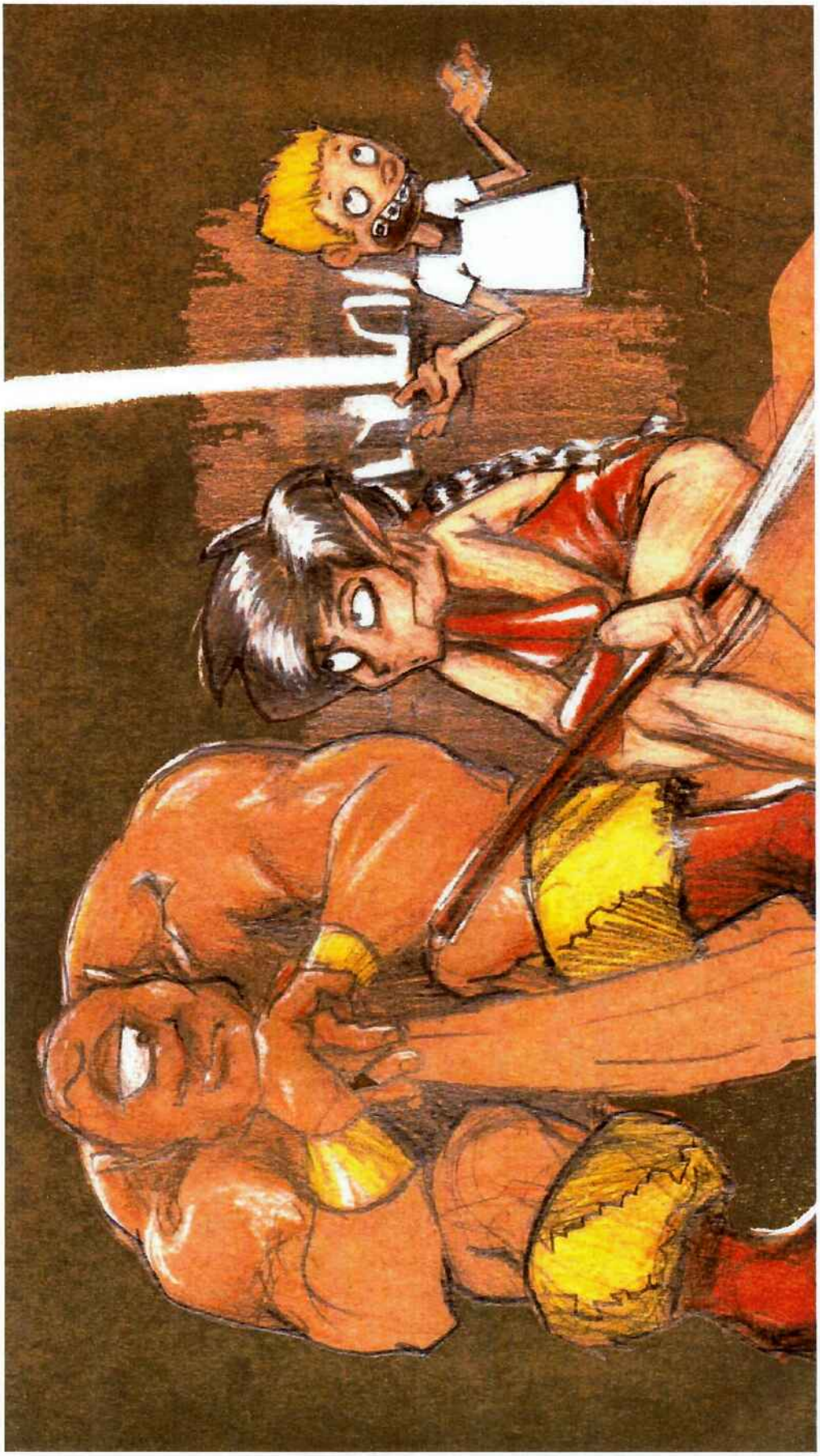


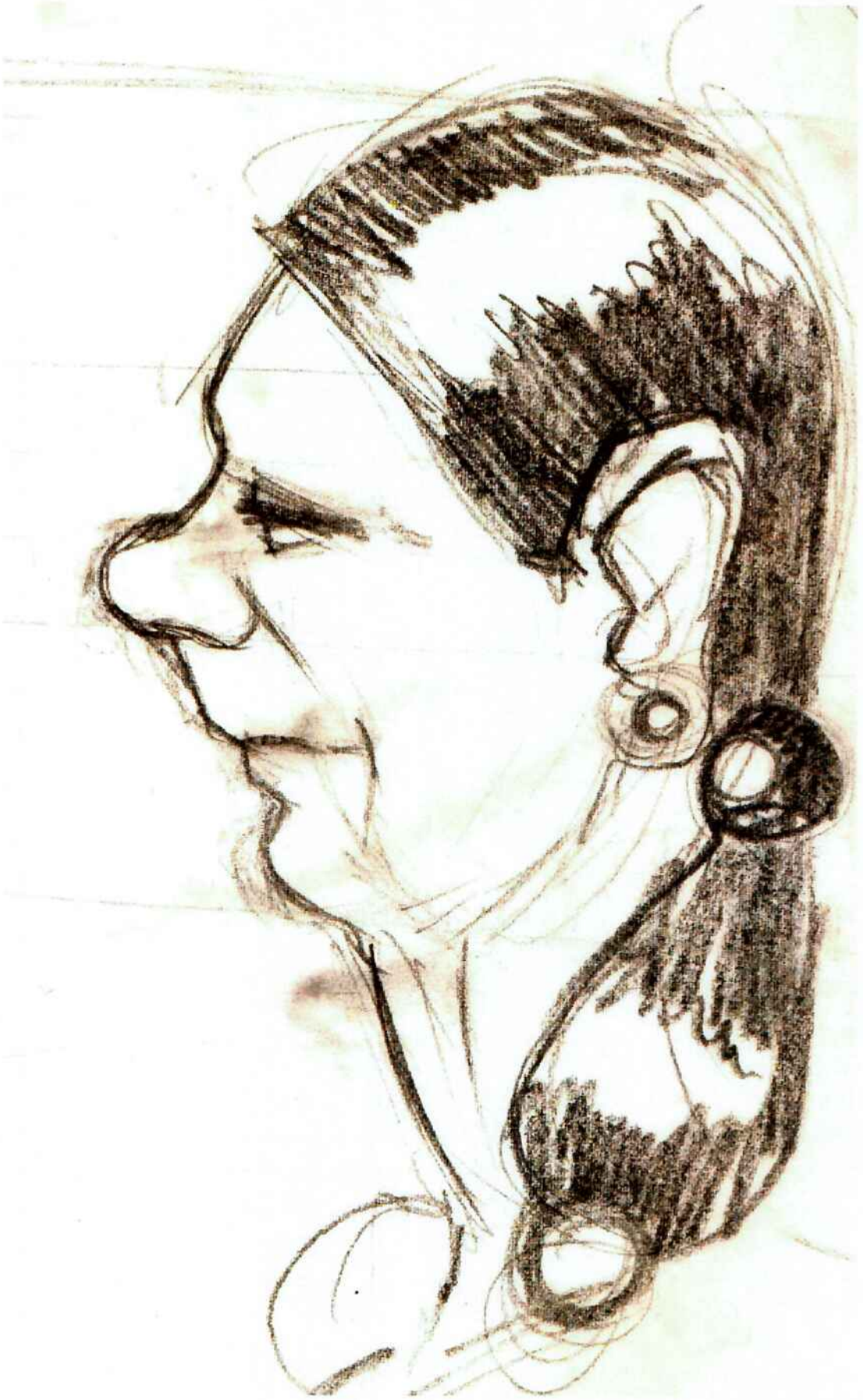












Livro de História

Roteiro

- texto: 7 dias atrás.....
  - Imagem de fundo: jornal cujo texto esta ilegível, apenas a data e nome do mesmo esta em evidência. O título é: Tablóide do Povo
  - Texto deverá aparecer scrolling up na tela, devidamente espaçado:

Durante as semifinais do torneio escolar de KartenSpiel, promovido pelo Diretório Academico do Colégio DeutschSchule, o jogador Alan Greiner, 15 anos morreu sem causa aparente. Uma investigação policial se instaurou, pois não foi encontrada a causa da morte.

- Texto: 3 Dias atrás.....
  - Imagem de fundo: cenário escuro e tétrico, com vultos e sombras, sem coloração, um garoto deitado e uma velha senhora sentada a seu lado
  - Texto deverá aparecer scrolling up na tela, devidamente espaçado:

Velha: Descanse mais um pouco, garoto. Não é todo dia que almas chegam aqui dessa maneira.

Alan: Como assim descanse??? Não posso, tenho o torneio para vencer. Meus pais me matam se não voltar logo pra casa. Preciso ir...

Alan: Perai! Como assim "Almas"???? Onde estou? Quem é você?

Velha: Voce esta morto. Sua alma deixou seu corpo devido a sua herança. Você

Alan: NÃO QUERO SABER DESSA PO@@@!!!! VOU VOLTAR PRA CASA, PRECISO VOLTAR, DEVO VOLTAR, NÃO MORRINÃOMORRINÃO MORRI NÃÃÃÃOOO!

Velha: Pois bem rapazinho, há uma maneira de VOCE voltar... Não será fácil, mas é possível

Alan(pensando): Isso tem que ser um pesadelo, ou a pior pegadinha que já fizeram...

Alan: Fala logo!!!!! Deixa de suspense!!!!

Velha: No local que não é, no final de todas as coisas, existe uma confluência onde as realidades se chocam. Lá, e apenas lá, aqueles que não mais são podem ser.

Alan: Quer fazer sentido, sua velha maluca??? Fala logo, ca@@@@@!!! Não saquei nada....

Velha: Siga aquela trilha sombria, e se voce realmente desejar voltar, talvez o local que não é se revele a você

(Imagem de fundo transiciona, garoto "flutuando" em cenário acinzentado com ponto luminoso branco em cima a direita)

Alan: Que diabos é essa @@@%$&#! ?? Bem, já que é a única coisa diferente aqui, vou pra lá....

Alan: Ai... doi tudo... difícil respirar... onde estou?? Parece a floresta que fica perto da escola...

- Texto: 1 hora atrás.....

- Imagem de fundo: Um pátio de escola, algumas crianças correndo, outras jogando card-game (KartenSpiel) em um canto. Garoto da cena anterior de pé no meio da cena, com um pouco de destaque pela coloração
- Texto deverá aparecer scrolling up na tela, devidamente espaçado:

Alan: Que bom, era um pesadelo!!! Consegui chegar na escola de novo!!!

Luis: Alan!!!! o que você esta fazendo aqui??? Seu funeral foi semana passada, a missa de sétimo dia acabou de ocorrer??? Devo estar sonhando

Pedro: Olha, é o Alan!!! Se tudo aquilo foi uma piada, voce vai apanhar muito!!!!

Carlos: Dane-se que foi uma piada, pelo menos ele esta vivo e bem. Vem jogar conosco, Alan!!!

Alan (perplexo, pensando): Não estou sacando nada, mas vou jogar!!!

(Imagem de fundo transiciona, garoto jogando card game com os outros, um bicho se materializando a partir da carta que está na mão do Alan. O restante do cenário de fundo permanece igual. Maior foco nas crianças jogando)

Luis: Ahhhhh!!!!!!

Pedro: SOCORROOOO!!!

Carlos: O QUE É ISSO?? AHHHHH!!!!!

Alan: MEUDEUSMEUDEUSMEUDEUS!!! SOME DAQUI!!!!! MANHEEEE!!!!

Bicho: Sumirei, mestre.

(Imagem de fundo transiciona, garoto de pé sozinho no pátio, outras crianças fugindo, bicho de pé, +-90% transparente. Velha do cenário anterior atrás dele)

Velha: Muito bem, voce conseguiu usar o poder....

Alan: Caraca!!! não me assusta desse jeito!!! O que foi aquilo??

Velha: A passagem permaneceu aberta, e os youkais estão fluíndo para este plano. Voce tem a obrigação de fechá-lo, uma vez que sua passagem o abriu.

Alan: ??? Continua sem fazer sentido, velha? O que voce quer dizer com isso???

Velha: voce verá..... Outro irá lhe ajudar em sua jornada.... Voce saberá onde ele está e o que fazer....

(Imagem de fundo transiciona, garoto de pé sozinho no pátio)

Alan: Acho que fiquei louco.... hummm.... vou para a floresta próxima a escola, lá é perto de onde me lembro ter acordado.....

1a Fase

Cenário: O jogo se inicia numa caverna escuro, relativamente pequena, com alguns seres estranhos se movendo. No fundo da caverna tem um ser estático envolto num manto negro, e em dois pontos quaisquer estão um espelho e uma carta de KartenSpiel, carta esta talhada em pedra.

Eventos da Fase:

- Chegar perto dos itens (Espelho das Almas e Carta de pedra)
  - Ao chegar perto dos itens, existem dois eventos possíveis:
    - Se já conversou com o estranho, pega o item e mostra o texto:
      - "Opa, isso parece algo legal"

    - Se ainda não conversou com o estranho, apenas mostra o texto:
        - "Há um objeto estranho no chão, mas voce não quer saber disso agora"

- Conversar com o estranho
    - Ao conversar com o estranho a primeira vez, é liberada a captura dos itens
        - Dialogo:

          Estranho: Se deseja minha ajuda, voce deverá redescobrir o que perdeu.
          Alan: Afffffff.... Tem alguém que não fale por enigmas nesse jogo??
          Estranho: busque a essência de seu ser, que está perdida
          Alan: ???
          Estranho: busque a essência de seu poder, que está presa fora de ti.
          Alan: Voce poderia fazer sentido, pelo menos??? Ou tá difícil???
          Estranho: Este local está selado, e seus temores materializados, juntamente com o que perdeu. Derrote seus temores e me traga a prova que está inteiro novamente.
    - Ao conversar com o estranho a segunda vez
        - Se os itens já foram obtidos, inicia um combate com o estranho (Boss Battle)
        - Se os itens não foram obtidos, o estranho fala:
            - "Prove que é digno de minha ajuda. Busque sua alma e seu poder, materializados nesta caverna"
    - Ao perder o combate, ocorre o diálogo e tela de game over
        - "Fraco demais.... Não há esperança para voce."
    - Ao vencer o combate, entra um evento automático e a transição para a próxima fase
        - O evento automático é o player efetuar o summon do bicho que apareceu na abertura, e o bicho golpear o estranho, derrubando ele.

- Combate com os bichos, ganhando XP e níveis

Transição de Fase

- Texto: Finalmente o escolhido chegou
    - Imagem de fundo: cenário da caverna, com apenas o estranho de manto e Alan de pé, se encarando. Explicações básicas do porque as coisas estão acontecendo. Estranho revela seu nome, e que é mulher. O diálogo ainda precisa ser escrito.
    - Texto deverá aparecer scrolling up na tela, devidamente espaçado:

      Estranho:
      Alan:
      Estranho:
      Alan:
      Estranho:
      Alan:
      Estranho:
      Alan:
      Estranho:
      Alan:

Estranho:
Alan:
Estranho:
Alan:
Guedaia:
Alan:
Guedaia:
Alan:

2a Fase

Cenário: Floresta próxima a escola. Cenário grande que parece um labirinto, com monstros espalhados. No final do labirinto está a velha do primeiro cenário, que na verdade é o culpado pelo infortúnio do garoto.

Eventos da Fase:

- Conversar com a velha
  - Ao conversar com a Velha, ocorreu um diálogo e ela se transforma em um demônio, Ahnatnom. Após o diálogo se inicia o combate final.
    - Dialogo:

Guedaia: Finalmente retornou a este plano, Ahnatnom?

Alan: Que nome mais bizarro...

Velha: Ola Guedaia. Voce AINDA não morreu? Achei que tinha cuidado disso na última vez....

Alan: ???

(modelo 3D da Velha muda para o de Ahnatnom)

Alan: !!! CARACA !!!

Ahnatnom: Nada pessoal, Alan, mas preciso de sua alma e seu poder, heheheh.

Guedaia: O poder dele despertou totalmente, ele não será um brinquedo seu!!!

Ahnatnom: Hehehehe.. Eu também estou mais forte. Irei matar e aprisionar a alma do escolhido, e com seu poder romper a barreira que me impede de dominar este mundo!

Alan: ... Haja clichê..... suponho que agora é quando nós chutamos seu traseiro, né?

Guedaia (sorrindo): muito bem criança, vamos selar Ahnatnom novamente... Voce está mais forte do que ele imagina, mas ainda não tem poder para destruí-lo... Não nessa encarnação.

Ahnatnom: Bah! Desta vez conseguirei te destruir, velha maldita!!!

Guedaia: Tente!!!

- Combate com os bichos, ganhando XP e níveis

Final

Cenário: Final da Floresta próxima a escola, no ponto do combate final. Diversas arvores caídas ou queimadas, o corpo do Bosso no chão, Guedaia e Alan de pé.

Animação com a carta de pedra virando a carta suprema, e Alan selando o portal com seu poder. Gueadaia vai ficando transparente até sumir, sempre com um sorriso no rosto. Música feliz de fundo, sem falas.

Ocorre a transição para o cenário da escola novamente, com Alan jogando KartenSpiel com seus amigos, e o cenario vai fazendo fade out to black enquanto rolam os créditos.

Livro de Projeto

Idéias para o Jogo

Diálogos inteligentes e espirituosos, inimigos medonhos com dificuldade progressiva, heróis com personalidades fortes e distintas, batalhas que exijam muita estratégia, enredo misterioso, variedade de técnicas e magias com força crescente.

Processo de Jogo

O processo pode ser entendido pelo seguinte ciclo:

1- o jogador se familiariza com o local onde está;
2- através dos diálogos, obtém as informações dos personagens locais referente a um determinado objetivo (enredo principal ou sub-enredo);
3- adquire os itens e equipamentos que julgar necessário para a tarefa (pode ser necessário adquirir mais dinheiro através de batalhas);
4- inicia a jornada até o local da tarefa;
5- se for outra cidade, retorna ao passo 1;
6- executa tarefa;
7- retorna a última cidade;
8- reinicia o ciclo a partir do passo 2.

Controle Geral

Controle via teclado.
Geral: Movimentação relativa ao personagem 1. Os demais personagens o seguirão.
Batalha: controle dos menus.

Personagens

Atributos: Health Power (HP – pontos de vida), Magic Power (MP- pontos de magia), Experience (XP- experiência), Força, Sabedoria, Constituição, Nível, Ataque, Ataque Mágico, Defesa, Defesa Mágica.

Armas: Alan Greiner – cartas, iô-iô
Guedhaia – espada (sabre de luz)

Itens

potion, eter, antidote, fenix down, elixir, entre outros
items equipaveis : amuletos e anéis mágicos

Fases

As cavernas - Guedhaia
Labirinto Final – Boss

Sistema de Jogo

- **Atributos :**
HP – Pontos de Vida (barra vermelha)
MP – Pontos de Magia (barra azul)
Strengh – Força física
Constitution – Resistencia
Wisdom – Sabedoria
XP – experiência
Nível – Gráu de evolução do personagem
Ataque = 2 * (Str + 0.2 * wisdom + Nível) + Bonus
Ataque Magico = 2 * (Wisdom + Nível) + Bonus
Defesa = Constitution + 0.3 * Str + Bonus
Defesa Mágica = Constitution + 0.3 * Wisdom +Bonus

-Evolução do Personagem
A cada nível que passa, os personagens ganham:
- Atributo : 50% do valor inicial
- HP : 60% do valor inicial + (Nível * Con * 0.2)
- MP: 20% do valor inicial + (Nível * Wis * 0.04)

-Mudança de nível
O personagem subirá para o nível X, quando tiver XP = Σ X*10
Por exemplo:

Nível	XP
0	0
1	10
2	30
3	60
4	100

-Experiência ganha
A experiência depende de dois fatores:

- O nível do inimigo.
- A diferença entre os níveis do personagem e do inimigo.

Segundo a fórmula

$$XP_{ganha} = \left(1 + \frac{NI - NP}{NP}\right) \times NI + 1$$

Onde NI é o nível do inimigo, e o NP é o nível do personagem.

Observe alguns resultados dessa formula:

NP	NI	XP
1	1	2
1	2	5
1	3	10
1	4	17
1	5	26

NP	NI	XP
5	5	6
5	6	8
5	7	10
5	10	21
5	15	46

NP	NI	XP
10	10	11
10	11	13
10	15	23
10	20	41
10	25	63

NP	NI	XP
5	1	1
5	2	1
5	3	2
5	4	4
5	5	6

NP	NI	XP
10	1	1
10	2	1
10	5	3
10	9	9
10	10	11

NP	NI	XP
20	4	1
20	8	4
20	10	6
20	15	12
20	20	21

- Batalha

O dano será calculado da seguinte forma

- Quando o inimigo atacar o personagem, o dano será :

$(ATK_{inim} - DEF_{pers})$*random(0.7~1.2)

- Quando o personagem atacar o inimigo, o dano será :

3 * $(ATK_{pers} - DEF_{inim})$*random(0.7~1.2)

Livro de Som

Músicas

Caverna
Batalhas comuns
Batalha com Boss
Labirinto final

Livro de Especificações Técnicas

Engine do Sistema

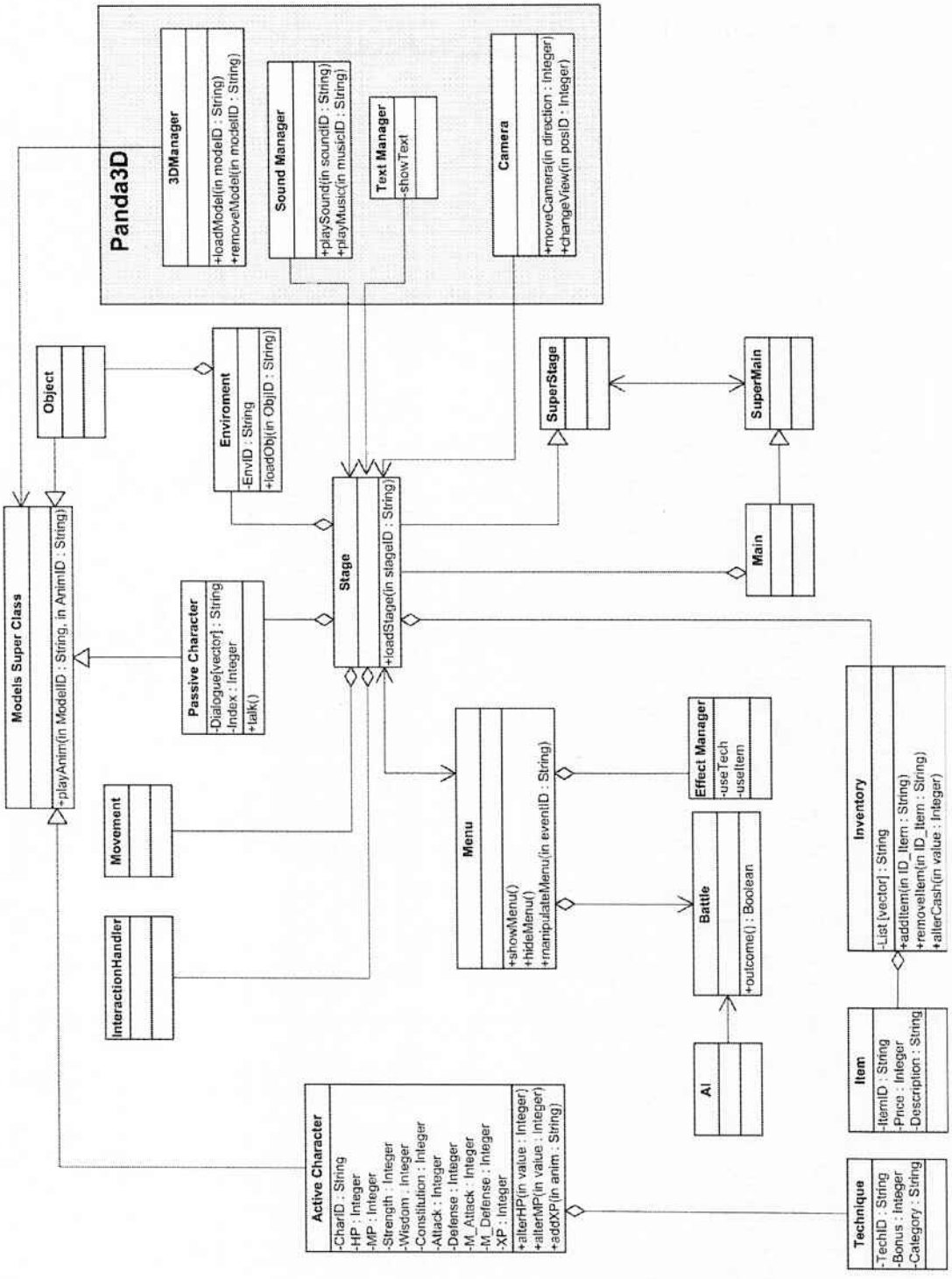
Ver Especificação Técnica da Engine

Engine Gráfica

A engine gráfica utilizada foi o Panda3D, devido a sua fácil utilização e o seu código ser aberto, o que possibilita que outras pessoas possam contribuir para o seu desenvolvimento. O Panda3D disponibiliza todos os recursos que julgávamos necessários como a renderização e a manipulação de modelos na tela, suporte a música ambiente e efeitos sonoros, exibição de imagens 2D, entre outros. Outra vantagem é sua versatilidade, já que o Panda3D trabalha tanto com o Python, que é uma linguagem interpretada, própria para prototipação, quanto com o C++, que é compilada.

ANEXO B

DIAGRAMA DE CLASSES



ANEXO C

CRONOGRAMA 2004

JANEIRO	Definição do projeto
FEVEREIRO	Pesquisa de arquitetura e roteiro do jogo
MARÇO	Definições das arquiteturas
ABRIL	Estudo das ferramentas e Game Design
MAIO	Modelagem inicial do sistema
JUNHO	Desenvolvimento da Engine
JULHO	Desenvolvimento da Engine
AGOSTO	Desenvolvimento da Engine
SETEMBRO	Desenvolvimento do jogo utilizando a engine
OUTUBRO	Desenvolvimento do jogo utilizando a engine
NOVEMBRO	Testes, depuração, Documentação parcial
DEZEMBRO	Documentação final

ANEXO D

Especificação Técnica da Engine

Capítulo I – Introdução

Esse Documento tem o objetivo de orientar o usuário da Engine FF Maker Plus 2005 (FMP), indicando quais passos devem ser seguidos para aproveitar toda potencialidade dessa poderosa ferramenta de desenvolvimento.

Algumas definições:
Engine FMP – conjunto de bibliotecas que fornece quase que infinitas possibilidades
Usuário – Pessoa que está utilizando a Engine para desenvolver um jogo
Jogador – Pessoa que utilizará o software final

Capítulo II – Estrutura de Arquivos

O diretório principal deverá conter os arquivos essenciais, como o código do jogo, além disso, ele deverá conter também os seguinte subdiretórios:

Models – contem os arquivos com os modelos 3D e respectivas animações, bem como os arquivos com as texturas dos modelos.

Music - contem os arquivos com as músicas utilizadas no jogo.

Sounds - contem os arquivos com os efeitos sonoros utilizados no jogo.

Sprite - contem os arquivos com as imagens 2D utilizadas no jogo.

Capítulo III - Arquivos XML

Para a utilização da Engine FMP o usuário deverá configurar uma série de arquivos XML, onde serão armazenadas informações relevantes ao jogo.

Seguem as descrições detalhadas de como configurar cada base em XML:

a) Models.xml

Esse arquivo contem as informações dos modelos 3D e também das suas animações, relacionando com um nome para a sua identificação no jogo.

Existe um Tag Pai chamado <Models>, cada modelo utilizado no jogo será determinado por uma tag <Char> com um atributo name, que designa a sua identificação.

Todo modelo deverá ter obrigatoriamente uma tag <Model>, que terá o nome do arquivo, sem sua respectiva extensão, do modelo 3D.

Além disso ele poderá ter tags que designam os arquivos com as animações. Poderão ser adicionadas quantas animações forem necessárias. O próprio nome da tag identifica a animação.

Exemplo :

```
<Models>
    <Char name = "Tritao">
        <Model>tritao_model</Model>
        <Walk>tritao_normalAndando</Walk>
        <Battle>tritao_ginga</Battle>
        <Punch>tritao_fightPunch</Punch>
        <Tail>tritao_fightTail</Tail>
    </Char>


    <Char name = "Panda">
        <Model>panda-model</Model>
        <Walk>panda-walk4</Walk>
        <Eat>panda-eat</Eat>
        <Turn>panda-turn</Turn>
        <Dance>panda-dance</Dance>
    </Char>

    <Char name = "Cave">
        <Model>cen_mortos</Model>
    </Char>
</Models>
```

b) ActiveChar.xml

Esse arquivo contem as informações dos personagens que lutarão no jogo e configura os seus atributos iniciais:

Existe um Tag Pai chamado <ActiveCharacters>. Cada personagem utilizado no jogo será determinado por uma tag <Char> com os atributos *name*, *model,* e *behavior*. Os dois primeiros são obrigatórios e designam, respectivamente, a identificação do personagem e qual é o modelo 3D relacionado a ele (por exemplo *Tritao* ou *Panda* do arquivo **Models.xml)**. O behavior é o tipo de AI do personagem, porem não é uma tag obrigatória uma vez que personagens controlados pelo jogador não precisam desse tipo de informação. Inimigos que não tiverem esse atributo receberão um padrão *Defaut* de comportamento.

Além disso cada tag Char deverá ter um série de tags com as infomações dos atributos do personagem

Exemplo :

```
<ActiveCharacters>
    <Char name = "Muleke" model = "Tritao" behavior =
"coward">
        <HP>260</HP>
        <MP>300</MP>
        <STR>12</STR>
        <CON>12</CON>
        <WIS>12</WIS>
        <XP>3000</XP>
    </Char>

    <Char name = "Bear" model = "Panda" behavior =
"magicbuster">
        <HP>660</HP>
        <MP>100</MP>
        <STR>8</STR>
        <CON>32</CON>
        <WIS>2</WIS>
        <XP>6000</XP>
    </Char>
</ActiveCharacters>
```

c) Camera.xml

Esse arquivo contem as informações dos diversos posicionamentos possíveis da câmera.

Existe um Tag Pai chamado <Camera>. Cada posicionamento de câmera utilizado no jogo será determinado por uma tag <View> com um atributo *name*, que designa a sua identificação.

Além disso, cada tag View deverá ter uma série de tags com as informações de posicionamento e ângulo da câmera. As tags <X>, <Y>, <Z> são as informações de posição e <H>, <P>, <R> são as informações de ângulo.

Exemplo :

```
<Camera>
      <View name = "Default">
         <X>-70</X>
         <Y>75</Y>
         <Z>50</Z>
         <H>220</H>
         <P>-20</P>
         <R>0</R>
      </View>

      <View name = "Battle">
         <X>-60</X>
         <Y>-60</Y>
         <Z>50</Z>
         <H>330</H>
         <P>-20</P>
         <R>0</R>
      </View>

      <View name = "WorldMap">
         <X>-160</X>
         <Y>160</Y>
         <Z>160</Z>
         <H>225</H>
         <P>-30</P>
         <R>0</R>
     </View>
</Camera>
```

d) **Music.xml**

Esse arquivo contem as informações das músicas utilizadas.

Existe um Tag Pai chamado <Musics>, Cada musica terá uma tag com o arquivo (que pode ser MP3, WAV ou MID). O próprio nome da tag identifica a música.

Exemplo :

```
<Musics>
    <Forest>Under A Violet Moon.mp3</Forest>
    <Battle>Battle Theme.mid</Battle>
    <BossBattle>Lavos Battle.wav</BossBattle>
</Musics>
```

e) **SFX.xml**

Esse arquivo contem as informações dos efeitos sonoros utilizados.

Existe um Tag Pai chamado <SFX>, Cada efeitos terá uma tag com o arquivo (que pode ser MP3, WAV ou MID). O próprio nome da tag identifica o efeito.

Exemplo :

```
<SFX>
    <Attack>hit.wav</Attack>
    <Swoosh>Swoosh.mid</Swoosh>
    <Menu>select.wav</Menu>
</SFX>
```

f) **MenuWorld.xml**

Esse arquivo contem as informações das opções de world map.

Existe um Tag Pai chamado <MenuWorld>. Cada elemento <Menu> indica um item deste nível do menu e o atributo action define qual a ação deste item.

As ações implementadas são: showStats, showAlliesTechs, showWeapons, showArmours, showAmulets, showRings, showItems.

Exemplo :

```
<MenuWorld>
    <Menu name = "Status" action = "showStats">
    </Menu>

    <Menu name = "Inventorio">
        <Menu name = "Equipamentos">
            <Menu name = "Armas" action = "showWeapons">
            </Menu>
        </Menu>
        <Menu name = "Items" action = "showItems">
```

```
        </Menu>
    </Menu>
</MenuWorld>
```

g) **EquipList.xml**

Esse arquivo contem as informações dos equipamentos existentes no jogo.

Existe um Tag Pai chamado <EquipList>, dentro desta temos uma árvore definida pelos tags <Type> e <Subtype>. Dentro de cada <Subtype> colocamos quantos equipamentos desejarmos, especificados pela tag <Equip>. Cada equipamento tem uma descrição, um preço e uma lista de efeitos que causa ao personagem que equipá-lo.

Exemplo :

```
<EquipList>
    <Type name = "weapon">

        <Subtype name = "yoyo">
            <Equip name = "Yo-yo">
                <desc>Arma mortal !</desc>
                <effect stat = "ATK" value =
"100"></effect>
                <price>1000</price>
            </Equip>
        </Subtype>

        <Subtype name = "lightsaber">
            <Equip name = "Ghedai Phaser">
                <desc>Lendaria arma ghedai</desc>
                <effect stat = "STR" value =
"10"></effect>
                <effect stat = "ATK" value =
"100"></effect>
                <effect stat = "mgATK" value =
"200"></effect>
                <price>5000</price>
            </Equip>
        </Subtype>

    </Type>
<EquipList>
```

h) **ItemList.xml**

Esse arquivo contem as informações dos itens existentes no jogo.

Existe um Tag Pai chamado <ItemList>. Cada item é definido pela tag <Item>. Os itens possuem descrição, preço, lista de efeitos e filtro de possíveis alvos.

Exemplo :

```
<ItemList>
    <Item name = "Potion">
        <desc>Recupera 100 de HP</desc>
        <effect action = "alterAttrib" stat = "HP" damage
= "-100"></effect>
        <target char = "any">
          <filter>alive</filter>
        </target>
        <price>100</price>
    </Item>
</ItemList>
```

i) **TechList.xml**

Esse arquivo contem as informações das técnicas existentes no jogo.

Existe um Tag Pai chamado <TechList>. Cada técnica é definida pela tag <Tech>. As técnicas possuem descrição, lista de efeitos, lista de custo, filtro de possíveis alvos.

Exemplo :

```
<TechList>
    <Tech name = "Heavy Punch">
        <desc>Quebra a cara de alguem</desc>
        <effect action = "physicalAttack" stat = "HP"
damage = "100"></effect>
        <cost stat = "HP" damage = "10"></cost>
        <cost stat = "MP" damage = "1"></cost>
        <target char = "any">
          <filter>alive</filter>
        </target>
    </Tech>
</TechList>
```

j) **CharEquips.xml**

Esse arquivo contem as informações dos equipamentos que um determinado ActiveCharacter.

Existe um Tag Pai chamado <CharSlots>. Definimos qual personagem esta lista se refere com a tag <Char>. Por fim, determinamos o <Type> e <Subtype> dos equipamentos que este personagem pode utilizar.

Exemplo :

```
<CharSlots>
    <Char name = "Muleke">
```

```
        <type name = "weapon">
            <subtype>yoyo</subtype>
        </type>
        <type name = "armour">
            <subtype>light</subtype>
        </type>
        <type name = "amulet">
            <subtype>simple</subtype>
            <subtype>magical</subtype>
            <subtype>overpower</subtype>
        </type>
    </Char>
</CharSlots>
```

l) **CharMenus.xml**

Esse arquivo contem as informações dos menus de batalha de um determinado ActiveCharacter.

Existe um Tag Pai chamado <CharMenus>. Definimos qual personagem esta lista se refere com a tag <Char>. Por fim, determinamos as opções de menu que este personagem terá em batalha com as ações associadas.

Exemplo :

```
<CharMenus>
    <Char name = "Muleke">
        <Menu name = "Joga" action = "attack"></Menu>
        <Menu name = "Tecnica" action =
"showTechs"></Menu>
        <Menu name = "Item" action = "showItems"></Menu>
        <Menu name = "Esconde" action = "wait"></Menu>
    </Char>
</CharMenus>
```

m) **CharTechs.xml**

Esse arquivo contem as informações das técnicas de um determinado ActiveCharacter.

Existe um Tag Pai chamado <CharMenus>. Definimos qual personagem esta lista se refere com a tag <Char>. Por fim, determinamos o uso de um item, a técnica referente ao ataque normal e as técnicas referentes aos ataques especiais.

Exemplo :

```
<CharTechniques>
    <Char name = "Muleke">
        <attack name = "Physical Attack" type = "melee"
anim = "Punch"></attack>
```

```
        <item name = "Item" type = "ranged" anim =
"Punch"></item>
        <tech name = "Haduoken" type = "ranged" anim =
"Tail"></tech>
    </Char>
</CharTechniques>
```

Capítulo IV – Criando a sua fase

Quando o usuário for criar uma fase utilizando a Engine FMP ele deverá seguir alguns passos, obrigatoriamente.

Em primeiro lugar devemos entender cada fase como uma classe distinta, e nela serão criados os métodos que manipulam e instanciam os objetos, criando uma fase interativa.

Sabendo disso, o usuário deverá importar o arquivo SuperStage.py e a classe da fase deverá herdar a classe SuperStage, para que o usuário possa usufruir de toda a vasta gama de possibilidades que a Engine FMP oferece.

O começo do arquivo da fase deverá obrigatoriamente conter:

```
from SuperStage import * #Importação do arquivo

class Stage01(SuperStage):#instância da classe da
fase
                          #herdando o SuperStage

        def __init__(self,main): #construtor,
importante
                                 #o parâmetro main

            self.initStage(main)#Inicialização do
        superStage
```

Capítulo V – Funcionalidades do SuperStage

Agora aprenderemos como utilizar todos os recursos que só o SuperStage trás para você. Lembrando que as funções devem ser chamadas utilizando um *self* antes, pois como o SuperStage é herdado, as funcões funcionam como métodos internos da classe.

Obs : usaremos os arquivos XML como exemplos fonte para os exemplos dos métodos.

As funções serão apresentadas da seguinte forma:
NomeDaFunção (parametro1, parametro2, etc...)
Descrição da Função
Descrição de cada parâmetro
Exemplo (se necessário)

a) Metodos de Acesso ao ambiente

São funções que manipulam o cenário

- loadEnvironment(Env):
  Carrega um cenário previamente definido no arquivo Environment.py (descrito melhor no capítulo de cenário)
  Env – identificação do cenário

- setDirectionalColor(DC):
  Define uma iluminação no ambiente
  DC – definição da cor, é do tipo Vec4 (R, G, B. A)
  Ex:

```
self.setDirectionalColor(Vec4(0,0,1,1))#iluminação azul
```

b) Metodos de acesso ao som

- playMusic (music)
  Toca uma música previamente definida no Music.xml. Caso outra música esteja tocando, a anterior é parada para o início da nova.
  music – identificação da música
  Ex:

```
self.playMusic("Forest")
```

- playSound (sound)
  Toca um efeito sonoro previamente definido no SFX.xml.
  sound – identificação do efeito sonoro
  Ex:

```
self. playSound ("Swoosh")
```

- stopMusic ()
  Para de tocar a música

- resumeMusic ()
  Volta a tocar uma música previamente parada

c) **Métodos de acesso à Câmera**

- initCam()
  Inicialização da câmera. Necessário para utilizar as outras funções

- setNearCam(value)
  Delimita um plano a partir do qual serão renderizados os modelos
  value – distância da câmera, a partir de onde haverá renderização

- centerCam()
  Centraliza a câmera no objeto de referência

- disableCam ()
  Desabilita a movimentação. A câmera deixa de seguir o objeto de referência

- enableCam ()
  Rehabilita a movimentação da câmera

- setCamReference (value)
  Define ou muda a referencia da câmera
  value – objeto do tipo ModelSuperClass que será referência da câmera

- setCamView (view):
  Ajusta o angulo de visão da câmera, previamente definido no Camera.xml
  view – identificação do posicionamento da câmera
  Ex:

```
self.setCamView ("WorldMap")
```

- getCamView ()
  Retorna a identificação da posição atual da câmera

- addView (view,orient)
  Define mais um posicionamento de câmera para escolha do usuário
  view – identificação do posicionamento da câmera
  orient – valor inteiro entre 1 e 4, determina a orientação do movimento do personagem associada ao posicionamento da câmera.
  Ex:

```
self.addView("Default",1)
self.addView("Battle",3)
self.addView("WorldMap",1)
```

Esse código faz com que cada vez que o usuário aperte a tecla "C" o sistema mude para outro posicionamento da câmera. Não funciona se o disableCam() estiver acionado.

d) Metodos de acesso à Movimentação

- initMovement(reference = 'default'):
  Inicia o movimento acionado por teclas. O personagem movimentado será o primeiro da lista (lista de personagens serão explicadas mais adiante) se não for definido nenhum valor ao *reference*.
  reference - objeto do tipo ModelSuperClass que será movimentado

- addFollower(follower)
  Adiciona objetos que seguem a referência quando se movimenta
  follower - objeto do tipo ModelSuperClass que seguira o movimento

- disableMovement()
  Desabilita a movimentação via teclado

- enableMovement()
  Reabilita a movimentação via teclado

- changeOrientation(value):
  Muda a orintação do movimento, útil em trocas de posicionamento da câmera.
  value – valor inteiro entre 1 e 4

e) Metodos de instância de ActiveCharacters e PassiveCharacters

Esses são dois objetos que herdam o ModelSuperClass e servem como parâmetro de algumas das funções citadas acima.

Os métodos a seguir simplificam a instanciação desses objetos, mas nada lhe impede de utilizá-los diretamente, como será explicado mais adiante.

- addAlly(x, y, z, name)
  Instancia ActiveCharacter amigo, adicionando-o a um array chamado ally, que poderá ser utilizado para manipulação pelo usuário durante a fase. É passado de uma fase para a outra diretamente, mantendo todas as informações relevantes.
  A identificação vem do arquivo ActiveCharacters.xml
  Essa função retorna o objeto instanciado.
  x, y, z – Coordenada com a posição inicial do objeto.
  name – identificaçãodo personagem
  Ex:

```
self.addAlly(10, -5, 0, "Muleke")
```

- getAlly()
  Retorna o array ally

- addEnemy(x, y, z, name):

Como o addAlly(), só que instancia um ActiveCharacter inimigo, e adiciona-o num array chamado enemy.

A identificação vem do arquivo ActiveCharacters.xml
x, y, z – Coordenada com a posição inicial do objeto.
name – identificaçãodo personagem
Ex:

```
self.addEnemy(10, -5, 0, "Bear")
```

- getEnemy()
  Retorna o array enemy

- def addPC(x, y, z, model, text, type):
  Instancia um objeto PassiveCharacter e adiciona-o a um array chamado PC
  A identificação vem do arquivo Models.xml
  x, y, z – Coordenada com a posição inicial do objeto.
  model – identificaçãodo personagem
  text – é um array de strings, contendo várias partes de uma conversa
  type – modo de exibição dos textos
  Ex:

```
self.addPC(10, -5, 0, "Bear", ["olá", "até mais"], "last")
```

Mais será dito sobre esses objetos em outro capítulo.

Nos próximos dois capítulos discutiremos alguns pontos que exigem um pouco mais de atenção, apesar de ainda serem funcionalidades do SuperStage.

Capítulo VI – Tratamento de Colisões

Para lidar com as colisões utilizaremos a técnica de bounding box, ou seja, definiremos sólidos de colisão que irão se chocar. Para tanto, existirão conjuntos de sólidos, sendo que eles apenas colidirão com sólidos de conjuntos diferentes.

Para simplificar a vida do usuário definimos alguns métodos.

Os métodos apresentados a seguir são, ainda, métodos do SuperStage.

- startCollidePoligon(name)
  Inicia geração de um grupo de sólidos de colisão
  name – identificação do grupo

- addTube(x1,y1,z1, x2,y2,z2, radius):
  Adiciona um tubo ao grupo de colisão
  x1, y1, z1 – ponto de início do tubo
  x2, y2, z2 – ponto final do tubo
  radius – raio do tubo

- addSphere(x1,y1,z1, radius):
  Adiciona uma esfera ao grupo de colisão
  x1, y1, z1 – centro da esfera
  radius – raio da esfera

- endCollidePoligon(reference)
  Finaliza o grupo de colisão
  reference – todo grupo deve ser relacionado a algum ModelSuperClass

Como podemos observar, os sólidos possíveis são esferas e tubos, para acrescentar ainda mais a esse fabuloso aparato de desenvolvimento, criamos uma função que simula um arco utilizando tubos:

- addArch(x1,y1, x2,y2, angle, n, name, reference, radius, z)

Descreve um arco entre (x1, y1) até (x2,y2) de <angle> graus
x1,y1 – ponto inicial do arco
x2,y2 – ponto final do arco
angle – angulo descrito pelo arco
n – número de tubos utilizado
name – identificação do grupo
reference – ModelSuperClass de referência
radius – raio dos tubos utilizados
z – altura z do grupo

Os métodos apresentados até agora são do conjunto que consideramos como cenário.

Existe outro método que será utilizado para criar sólidos para o conjunto que consideramos como dos personagens:

- setCollideSphere (reference, radius)
  Cria uma esfera em torno da referência
  reference - ModelSuperClass de referência
  radius - raio da esfera

Exemplo:

```
#cria uma esfera no primeiro personagem inserido na
fase
self.setCollideSphere(self.ally[0], 5)

#cria uma sólidos em volta de uma lago
self.startCollidePoligon('Lago')
self.addTube(57, -13, 0, 35 , -33, 0, 8)
self.addTube(35, -35, 0, 27 , -68, 0, 5)
self.addTube(35, -75, 0, 89 , -39, 0, 10)
self.addTube(109,-16, 0, 64 , -17, 0, 7)
self.endCollidePoligon(self.cenario)
```

Capítulo VII – ModelSuperClass

Vários dos objetos que comentamos até agora são do tipo ModelSuperClass, que é na verdade uma classe herdada por três outras, com uma característica em comum: todas são classes que representam algo visível em 3D na tela.

Já mostramos alguns métodos que instanciam estas classes para nós. Entretanto você pode chamar a classe diretamente. Como fazer isso e algumas das funções especificas de cada classe serão detalhadas a seguir. Lembrando que os métodos de objetos deverão ser chamados utilizando-se *Objeto.nomeDoMetodo()*.

As três Classes são:

a) **ActiveCharacter** – objetos com participação em batalhas, tem atributos de personagem. Ex: Heróis e monstros
Exemplo de como instanciar:

```
self.meuPersonagem = ActiveCharacter(10,-20 , 0 ,
"Muleke")
```

O Active Character tem alguns métodos próprios:

- alterHP(amount)

Modifica os pontos de vida do personagem
Amount – quantidade modificada (pode ser negativa)

- alterMP(amount)

Modifica os pontos de energia mística do personagem
Amount – quantidade modificada (pode ser negativa)

- addXP(amount)

Aumenta os pontos de experiência do personagem, caso ele mude de nível, recalcula os atributos dele.
Amount – quantidade modificada

Ex.

```
self.meuPersonagem.alterHP(-100) #dá 100 de dano
```

b) **PassiveCharacters** – não participam de batalhas, mas interagem, normalmente com alguma fala ou texto com o jogador. Ex. Comerciante, Placa.

Exemplo de como instanciar:

```
self.meuPersonagem = PassiveCharacter(10, -5, 0,
"Bear", ["olá", "até mais"], "last")
```

O último parâmetro (type) pode ter os seguintes valores:
"last" – após o fim do diálogo, se for solicitado, ele repetirá a última fala
"repeat" – o PC repete o diálogo todo indefinidamente
"stop" – o PC não fala mais após a última fala

O Passive Character, apesar de não lutar, tem vários recursos de interação com o jogador. O usuário pode definir três métodos que serão chamados em momentos distintos. Para tanto ele usará o seguinte método:

- setFunctions(firstFunction, middleFunction, lastFunction)

firstFunction – é definido aqui o método que será chamado na primeira interação, antes mesmo de dizer a primeira fala.

middleFunction - é definido aqui o método que será chamado junto com a primeira fala.

lastFunction - é definido aqui o método que será chamado junto após a última fala.

Cada método será chamado apenas uma vez, se o usuário quiser que seja utilizado mais vezes, poderá utilizar o método:

- reactivateFunctions()

c) **Objects** – Objetos estáticos. Ex. Árvore, Cenário.

Existem uma série de métodos que são do ModelSuperClass e podem ser chamados por objetos de qualquer uma das três classes, são eles:

- setAsReference()
  Todas as fases deverão ter algum ModelSuperClass que definido como referencia para uma série de eventos (normalmente o mesmo da câmera e do movimento).

- setScale(value)
  Define a escala do objeto
  value – proporção da escala (1 = 100%)

- setH (value)
  Muda a direção do modelo.
  value – valor em graus da direção

- setPos (x, y, z)
  Define a posição do objeto
  x, y, z – Ponto que o objeto será posicionado.

- getPos ()
  Retorna as posições do objeto em um dictionary
  Exemplo de como mostrar na tela a posição X de um objeto:

```
posicao = self.meuPersonagem.getPos
X = posicao["X"]
```

```
Print X
```

- playAnim(anim, mode, rnd):
  Roda uma animação das previamente definidas no Models.xml
  anim = identificação da animação
  mode = escolha entre rodar uma vez, rodar em loop a animação ou então rodar uma vez e depois voltar para animação anterior.
Valores:
  "nomal" - uma vez
  "loop" – infinitas vezes
  "volta" – roda uma e depois volta

- playBattleAnim(anim, otherModel, nome, duration, dist)
  Animação complexa, idealizada para ser utilizada na batalha. O objeto irá na direção de um segundo objeto, irá parar um pouco antes dele, rodará a animação, voltará para posição inicial e para a animação anterior
  Anim - identificação da animação
  otherModel – Segundo objeto
  nome –
  duration - distância entre ele e o otherModel (default = 10)
  dist - duração de cada um dos deslocamentos

- stopAnim()
  Para a animação em andamento

- resumeAnim()
  Volta a rodar a animação parada

- keepWalking(self,x1,y1,x2,y2,duration)
  Faz o objeto ir de um lado a outro indefinidamente
  x1,y1 - posição inicial
  x2,y2 - posição final
  duration - duração de um ciclo completo em segundos (default = 20)

Capítulo VIII – Cenários

Para construir os cenários é necessário modificar o arquivo Environment. Neste arquivo temos definidos os cenários e também todos os modelos que precisam ser carregados junto com ele.

Estudando um código exemplo:

```
if type == 'Forest':
   self.obj.append(Object(0,77,0,'For1'))
   self.obj[0].setScale(0.5)

   self.obj.append(Object(0,77,0,'For2'))
   self.obj[1].setScale(0.5)

elif type == 'Toon' :
   self.obj.append(Object(50,50,0,'Toon'))
   self.obj[0].setScale(1)
   self.obj[0].setH(270)

elif type == 'Cave' :
   self.obj.append(Object(0,0,0,'Cave'))
   self.obj[0].setScale(1)
   self.obj[0].setH(0)
   self.lights(directionalColor = Vec4( 1, 0.8, 0.2, 0.1) , ambientColor =
Vec4( 0.5, 0.5, 0.5, 1))
```

Neste caso temos três cenários distintos: Forest, Toon e Cave.

Podemos observar que o cenário Forest consiste de dois modelos: For1 e For2, ambos previamente especificados no Models.xml. O cenário Toon é mais simples, baseado em apenas um modelo 3D. O cenário Cave também é baseado em apenas um modelo, entretanto, para deixá-lo mais elaborado, utilizamos um fantástico recurso da classe Environment: o método lights.

Com esta funcionalidade, podemos modificar a luminosidade ambiente do cenário com o parâmetro ambientColor e a fonte de iluminação com o parâmetro directionalColor. Os argumentos definem a cor da luz correspondente, seguindo o padrão:Vec4(R, G, B, A), sendo Red, Green, Blue e Alpha (transparência), respectivamente.

Capítulo IX – Main Menu

Para personalizar o main menu, precisaremos modificar o arquivo do super-herói sem fronteiras, o SuperMain. Dentro do SuperMain, além de definirmos a aparência do main menu, com as opções de seleção desejadas, também escolhemos qual será a primeira fase do jogo, através da variável stage.

```
def pleaseKillMe(self,gamePoint,ally,oldStage,newStage,inv):
  oldStage.destructor()
  if newStage == 1:
      self.stage = Fase01(self,gamePoint, ally,inv)
  elif newStage == 2:
      self.stage = Fase02(self,gamePoint, ally,inv)
  elif newStage == 3:
      self.stage = Fase03(self,gamePoint, ally,inv)
```

8. LISTA DE REFERÊNCIAS

[1] SPELLBRASIL ROLEPLAYING GAMES. Fevereiro de 1998. Site sobre RPG e informações em geral. Disponível em <http://www.rpg.com.br>. Data de acesso: 18 de novembro de 2004.

[2] PLACES TO GO, PEOPLE TO BE. 1998. Site com artigos sobre RPG. Disponível em <http://ptgptb.org/0001/history1.html>. Data de acesso: 3 de dezembro de 2004.

[3] DEVELOPER.COM. Site sobre desenvolvimento de software. Disponível em <http://www.developer.com/open/article.php/968461>. Data de acesso: 3 de dezembro de 2004.

[4] MICROSOFT CORPORATION. 2004. Site sobre DirectX. Disponível em: <http://www.microsoft.com/directx>. Data de acesso: 18 de novembro de 2004.

[5] ARTLAB. 2004. Site sobre OpenGL. Disponível em: <http://www.opengl.org>. Data de acesso: 18 de novembro de 2004.

[6] OASIS OPEN. 2004. Site sobre o formato XML e suas aplicações. Disponível em: <http://www.xml.org>. Data de acesso: 18 de novembro de 2004.

[7] HTML UNLEASHED. Junho de 1997. Site sobre HTML. Disponível em: <http://www.webreference.com/dlab/books/html/3-2.html>. Data de acesso: 3 de dezembro de 2004.

[8] ADAMS, JIM. Programming Role Playing Games With DirectX 8.0. 1ª Edição. EUA: Premier Press, 2002.

[9] PRATT, JASON, PANDA MANUAL. 15 de maio de 2004. Manual da engine. Disponível em <http://www.etc.cmu.edu/panda3d/manual/>. Data de acesso: 16 de junho de 2004.

[10] OBJECT MANAGEMENT GROUP. 1997. Especificações das ferramentas UML. Disponível em: <http://www.uml.org>. Data de acesso: 18 de novembro de 2004.

[11] VAN ROSSUM, GUIDO, PYTHON TUTORIAL. 20 de maio de 2004. Tutorial da linguagem. Disponível em: < http://docs.python.org/tut/tut.html >. Data de acesso: 16 de junho de 2004.