UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MARCO DÜREN CENTENO

Gerador Genérico de Jogos de RPG

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Raul Fernando Weber

Porto Alegre
2014

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Graduação: Prof. Sérgio Roberto Kieling
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do Curso de Ciência da Computação: Prof. Raul Fernando Weber
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

RESUMO

O RPG é gênero de jogos cuja importância e representatividade é muito grande. O jogo consiste em uma aventura em que os jogadores interpretam papéis de heróis em suas jornadas e suas regras são bastante computacionais. Há uma longa tradição de jogos deste tipo nas mais diversas plataformas, desde tabuleiros até ambientes computacionais. O presente trabalho tem por objetivo desenvolver um núcleo para facilitar a criação deste tipo de jogos, permitindo que apenas através de arquivos externos se possa dar uma forma e uma história, ou seja, permitir que com uma rápida edição de conteúdo do jogo seja possível criar jogos com bastante facilidade. No presente trabalho, além do resultado concreto da implementação do trabalho, são também descritos em duas partes o projeto e a sua implementação. Na primeira são discutidas as suas decisões e objetivos, e na segunda, analisados os caminhos seguidos de acordo com as técnicas de construção de programas e de engenharia de software.

Palavras-chave: Jogos eletrônicos, RPG

Generic RPG Games Generator

ABSTRACT

The RPG genre of games is a very important one. The game consists of an adventure in which players interpret roles of heroes on their journeys and its set of rules are quite computable. There is a long tradition of such games in various platforms, from board games to computer systems. This study is aimed to develop a core to facilitate the creation of such games, allowing through only external files to be given a shape and a story, ie, to allow quick editing of game content making possible to create games easily. In this paper, besides the concrete result of its implementation, are also described in two parts the design and the implementation. The first discusses the decisions and goals, and second, analyzing the paths followed in accordance with the program construction techniques and software engineering.

Keywords: Electronic games, RPG

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RPG	Jogo de interpretação de papel (Role Playing Game)
NPC	Personagem Não Jogador (Non Player Character)

Sumário

1 INTRODUÇÃO E MOTIVAÇÃO

O presente trabalho tem por objetivo implementar uma plataforma de jogos do tipo RPG.[1]

O RPG é um jogo em que o jogador interpreta um personagem de ficção, seguindo o enredo definido em um roteiro.

Como em qualquer jogo, há regras que o definem, porém o aspecto mais importante é a existência de uma história, em que o jogo se desenvolve no seu desenrolar. No RPG, existem dois tipos básicos de jogadores muito bem definidos: O primeiro tipo é o jogador Personagem e o segundo é o Mestre do Jogo.

O jogo consiste em uma aventura em que os jogadores interpretam papéis de heróis em suas jornadas, geralmente ambientadas em cenários que remetem a fantasias medievais, com castelos, fortalezas, magos, princesas, monstros, etc. Ele é fortemente baseado na fantasia dos jogadores e o papel do mestre do jogo, em inglês “Game Master”, é fundamental, pois ele é quem define a história e controla o fluxo do jogo de acordo com os eventos acontecidos, o que pode exigir muito de sua criatividade, pois embora a história possa ser predefinida, os detalhes determinados pelas escolhas dos jogadores podem resultar em situações imprevisíveis.

Exceto por esta flexibilidade no que diz respeito ao enredo, o jogo tem uma mecânica bastante rígida e precisa, consolidada em GURPS,[2] um acrônimo de “Generic and Universal Role Playing System”, que se fundamenta em linhas gerais no resultado de uma função entre dois elementos: as características dos personagens reduzidas a atributos numéricos, e a sorte, jogada nos dados. O presente trabalho não segue estritamente essas regras, porque elas envolvem aspectos que não são explorados em um RPG para computador, mas apenas um modelo simplificado criado com base nelas.

Os atributos numéricos dos personagens geralmente são medidos através de pontuações sobre características como força, inteligência, vitalidade, poder de ataque, defesa, velocidade, etc.

O elemento de sorte é obtido pelos lances de dados, que nos sistemas tradicionais de

---

1 ROLE-PLAYING GAME. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2014. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Role-playing_game&oldid=39202841>. Acesso em: 23 jun. 2014.

2 GURPS. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2014. Disponível em: <http://pt.wikipedia.org/w/index.php?title=GURPS&oldid=38152668>. Acesso em: 23 jun. 2014.

RPG variam de acordo com a quantidade de lados: quatro, seis, oito, dez, doze, vinte.

Existe grande variedade de tipos de sistemas de RPG e cada um define a sua forma de tratar com os eventos, mas em regra são dadas soluções que envolvem um sorteio cujo resultado é informado por um ou mais lances de dados de acordo com as possibilidades dadas pelos atributos do personagem. Para dar um exemplo, um combate em que, de acordo com o nível dos envolvidos, é possível que para um personagem mais forte seja exigido menos sorte do que para um personagem mais fraco. No caso poderíamos ter uma função de acordo com os pontos de ataque.

Supondo então, para exemplificar, uma simples fórmula para ilustrar uma circunstância normal do jogo, como a decisão sobre uma tentativa de atacar o inimigo ser ou não bem-sucedida. Por exemplo, façamos com que seja necessário o jogador atingir pelo menos vinte e cinco pontos, valor que arbitramos para o atributo pontos de defesa (PD) do inimigo. Estes pontos são informados por uma soma dos pontos de ataque (PA) do jogador, somados a um lance de um dado de vinte lados (D20).

A gama de possibilidades portanto ficaria definida de forma que, quanto mais pontos de ataque o jogador tenha, maior a quantidade de resultados possíveis para ser bem-sucedido, e portanto menos sorte seja necessária.

Vejamos algumas possibilidades de acordo com os Pontos de Ataque do Jogador. Para o ataque ser bem-sucedido é necessário satisfazer a seguinte fórmula:

PA + D20 >= PD

Sendo PD=25, as possibilidades são:

PA >= 24: Nem é necessário o lance de dados, pois o menor resultado possível é 1.

PA = 23: Qualquer resultado maior que 1 é suficiente.

PA = 22: Qualquer resultado maior que 2 é suficiente.

E assim por diante até:

PA = 5: Apenas com 20 é bem-sucedido.

PA < 5: Impossível.

O demonstrado pelo exemplo é a regra simplificada de tratamento de eventos em jogos de RPG cuja constante é fazer da soma da força do personagem mais sorte o evento resultante.

Como se percebe, esta mecânica é bastante computacional, tornando o gênero muito propício a criação de soluções bastante criativas.

No que diz respeito ao fluxo da história e suas improvisações, comuns na forma original do jogo, não é o tema do presente trabalho analisá-las, pois consistem em um

elemento demasiadamente criativo e humano, que, embora se possa pensar em alguma gramática que concatene aleatoriamente alguns clichês do tipo, matar dragão, encontrar tesouro, salvar princesa, etc., não é o escopo deste trabalho. No que diz respeito a isso, os RPGs para serem jogados em computadores via de regra contém a história com todas as suas possibilidades já previamente definidas no programa, e, no que diz respeito a fantasia da fala e do texto, os elementos de computação gráfica e interface também assumem o papel.

A escolha do tema foi em razão de enxergar no gênero a imensa gama de aspectos comuns a todos os tipos de jogos. Em um RPG encontramos um núcleo de jogo com alta complexidade, roteiro, elementos de interface, animação de figuras e labirintos, o que viabiliza a construção de um framework para criação de jogos desde os mais simples aos mais complexos.

Outro fator de motivação se deve ao presente momento em que a tecnologia está se direcionando para os dispositivos móveis, como celulares e tablets, aparelhos de menor poder computacional e recursos de interação alguns mais simples e outros mais inovadores, que estão reorientando o mercado de jogos a uma espécie de volta ao passado, especialmente dos jogos 2D.

Se observarmos os jogos para computadores de mesa e videogames atuais, podemos constatar em suas listas de pessoas envolvidas contingentes de pessoal enormes, muitas vezes já superando múltiplas vezes os de produções cinematográficas. Isso na prática significa que estes espaços estão ocupados pelas grandes empresas já enraizadas no mercado. Tal momento de rebaixamento de requisitos e exigências causarão ainda uma grande reviravolta no mercado, e nisto pode haver grandes oportunidades para que empresas independentes de pequeno porte possam triunfar[3].

Além da questão do poder computacional, há também as novidades em relação ao público bastante distinto do que se chama “hardcore gamer”, que seria a forma de jogar com um nível de dedicação absoluta, como a condição de estar em frente a um computador de mesa ou um videogame na TV da sala, por exemplo, estabelece.

No caso dos dispositivos portáteis, como celular e tablet, o perfil do jogador pode mudar, sendo mais ou menos uma forma de ocupar um tempo livre não intencional e de circunstâncias restritas, como, por exemplo, em que o interesse de jogar não é primário, mas basicamente um interesse residual sobre como preencher o tempo em uma situação em que não teria nada para fazer, como por exemplo, esperando a comida ficar pronta num restaurante, o trajeto de um ônibus, uma sala de espera de um consultório médico. O contrário

3 ZECHNER, M.; GREEN R. Beginning Android 4 Games Development. Apress, 2011, p. 54

de como quando alguém em sua casa decide ligar seu videogame e não fazer nenhuma outra coisa.

Estes perfis de jogadores são multifacetados, pois muitas vezes podem conviver vários na mesma pessoa. Assim, um RPG, que geralmente é um gênero muito absorvente e que carrega consigo as vezes até a estigmatização pela quantidade de tempo que consome da vida da pessoa, pode limpar-se dessa carga se jogado na forma casual, em que o progresso incremental tem relação de proporção direta com o tempo inútil da pessoa e não de proporção inversa com as coisas que deixou de fazer.

Além disso, teria outras vantagens que jogos casuais mais simples não proporcionam, mas pelo contrário, pois as vezes criam frustrações baseadas na perpétua repetição do mesmo desafio apenas com dificuldade incrementada, ou na sensação de tempo perdido em toda a vez em que se joga uma partida de um jogo, que é essencialmente sempre igual, e não se quebra o próprio recorde.

Assim, um RPG casual, jogado em um tempo em que, pela circunstância, não teria uma aplicação melhor, pode ser um gênero bastante promissor e popular.

1.1 Organização do Trabalho

O trabalho é organizado de forma a expor, após introdução e um breve relato sobre o Estado da Arte, o projeto e a implementação.

A parte referente ao projeto é focada nos componentes e nas relações entre eles. Em primeiro lugar são descritos os elementos da lógica do núcleo do jogo, na segunda parte, a interface gráfica, e na terceira, a integração entre os componentes.

A parte da implementação se divide em elementos do núcleo e interface, e seus respectivos ciclos de desenvolvimento de acordo com a ordem cronológica em que foram trabalhados.

2 ESTADO DA ARTE

Jogos de RPG tem sido implementados desde muito tempo, sendo o primeiro datado de 1975 em mainframes PDP-10 e Unix.[4] O trabalho porém traça uma linha que pinça alguns em uma linha de tempo, que não tem a preocupação de ser uma rigorosa linha de evolução dos jogos do gênero, mas apenas assinalar alguns critérios de incrementos de aspectos de jogabilidade e aproximação com o objetivo do projeto.

A linha inicia em 1986 com uma implementação para o padrão MSX. O nome do Jogo é Dungeon Master.[5] É um jogo com os elementos básicos da mecânica e sem nenhuma história, exceto o objetivo de progredir na exploração de labirintos, buscando encontrar chaves que permitem entrar em níveis mais profundos, até encontrar um anel branco, o que faz com que o jogador ganhe o título de "Dungeon Master". Após isso, nada muda e ele continuar jogando normalmente como se nada tivesse acontecido, aumentando a sua experiência, matando inimigos e enriquecendo com os itens valiosos e ouro que siga encontrando pelos labirintos.

O jogo é uma das primeiras implementações de RPG, e de certa forma antecipa o RPG de ação, pois os combates acontecem quando o personagem do jogador e os inimigos tentam ocupar a mesma posição. O jogador utiliza as teclas direcionais para fazer o personagem se locomover pelo cenário e a barra de espaços para abrir os menus para visualização de informações sobre o personagem, como experiência, quantidade de ouro, pontos de ataque, defesa e itens do seu inventário.

Embora não tenha o que se possa chamar de uma história, possui muitos elementos de RPG. O jogo começa em uma cidade onde existe uma estalagem, onde o personagem pode ver suas informações, recarregar seus pontos de vida e gravar seu progresso e uma loja em que podem ser comprados alguns itens com o ouro adquirido.

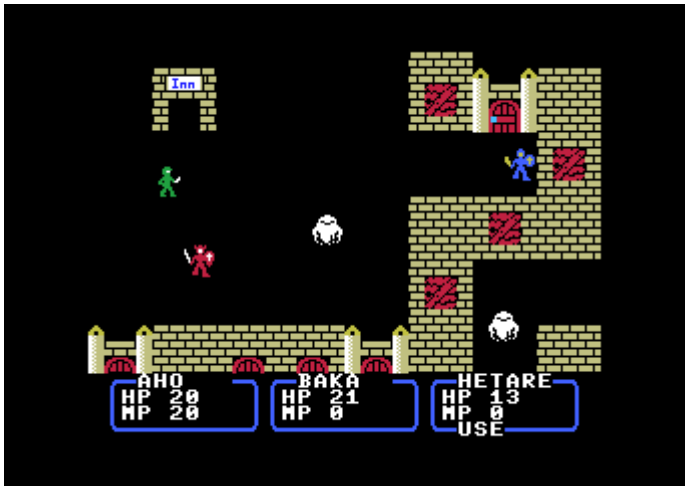
Na cidade só existem fantasmas e morcegos, os inimigos mais fracos, tanto pelas ruas quanto pelas salas, e a medida que progride para lugares mais profundos encontra inimigos mais desafiadores, até dragões.

---

4 RPG ELETRÔNICO. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2013. Disponível em: <http://pt.wikipedia.org/w/index.php?title=RPG_eletr%C3%B4nico&oldid=34912200>. Acesso em: 22 jun. 2014.

5 Dungeon Master (1986, MSX, ASCII) | Generation MSX. Disponível em: <http://www.generation-msx.nl/software/ascii/dungeon-master/787/>. Acesso em: 23 jun. 2014

Figura 2.1 – Tela do Jogo Dungeon Master da ação na cidade



Fonte: Dungeon Master (1986, MSX, ASCII) | Generation MSX. (Disponível em: <http://www.generation-msx.nl/software/ascii/dungeon-master/787/>. Acesso em: 23 jun. 2014.)

Nas salas, além dos inimigos, existem os baús que podem conter ouro ou itens e basicamente o jogo se resume a isso: matar inimigos e coletar o conteúdo das arcas.

Figura 2.2 – Tela do jogo Dungeon Master em uma sala

Fonte: Dungeon Master User Screenshot #3 for MSX - GameFAQs (Disponível em: < http://www.generation-msx.nl/software/ascii/dungeon-master/787/ >. Acesso em: 23 jun. 2014.)

O segundo jogo é o Final Fantasy IV,[6] lançado em 1992. Ao contrário do primeiro, o principal traço deste jogo é a importância da história e interface que interrompe a ação do jogo. É um jogo que começa com uma longa introdução utilizando os próprios componentes de interface do jogo, em vez de desvio para vídeos ou imagens, e após isto entrega o controle

6 FINAL FANTASY IV. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2014. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Final_Fantasy_IV&oldid=39175216>. Acesso em: 22 jun. 2014.

ao jogador para que ele explore os lugares, converse com os personagens não jogadores e trave os combates, sendo as vezes tirado do controle durante eventos importantes da história.

Este jogo oferece uma interface rica e com muitos menus para permitir ao jogador um bom controle do jogo sem exigir habilidade, pois um dos traços essenciais do RPG, ao qual este jogo é muito fiel, é ser um jogo mental e não de ação.

A interface alterna entre modos de navegação pelo mapa, diálogos e combate.

Na navegação, o personagem utiliza as setas direcionais e circula pelos mapas indo atrás de cumprir os objetivos. O mecanismo do jogo faz com que a cada passo haja um sorteio que defina se há ou não inimigos a serem combatidos nesta posição. Nos diálogos, pode-se acelerar a exibição do texto e os menus são simples seleções e confirmações das opções tomadas.

Figura 2.3 – Tela do Jogo Final Fantasy IV na ação no mapa do mundo

Fonte: Final Fantasy IV Part #2 - Cecil Slays a Dragon and Rescues a Damsel in Distress (Disponível em: < http://lparchive.org/Final-Fantasy-IV/Update%2002/ >. Acesso em: 23 jun. 2014.)

Na tela de combate, as batalhas são processadas. Nela, aparecem o jogador e seus companheiros e os inimigos, que são visualizados em lados opostos e em formas gráficas mais elaboradas e detalhadas, como se fossem vistos de perto, e existem os menus e painéis onde escolhas e informações são exibidas.

Figura 2.4 – Tela do Jogo Final Fantasy IV em modo combate

Fonte: The Greatest Games of All Time: Final Fantasy II - GameSpot
(Disponível em: <http://www.gamespot.com/articles/the-greatest-games-of-all-time-final-fantasy-ii/1100-6132899/ >. Acesso em: 23 jun. 2014.)

O terceiro jogo da lista é Diablo.[7] Lançado em 1996, trata-se de um jogo em que toda sua ação ocorre na mesma interface. A parte gráfica é muito bem trabalhada, em perspectiva isométrica, mas a novidade apresentada é a consolidação do RPG de ação, onde não existe o cadenciamento da ação através de escolhas em menus. O personagem do jogador apenas aponta para o inimigo e o ataca, e este por sua vez também o ataca como que em tempo real, sem nenhuma mediação e nenhuma escolha do jogador que faça o jogo esperar. Embora a história seja um pouco pobre, a mecânica de jogo foi muito bem definida.

Figura 2.4 – Tela do Jogo Diablo I

Fonte: PC Games of the 90's (102 pics) ~ Crack Two
(Disponível em: < http://www.cracktwo.com/2011/02/pc-games-of-90s-102-pics.html >. Acesso em: 23 jun. 2014.)

7 Diablo - GameSpot. Disponível em: <http://www.gamespot.com/diablo/>. Acesso em: 23 jun. 2014

Após isso, quarto e último da lista é World of Worcraft.[8] Este jogo é muito bem elaborado na parte gráfica e bem complexo no que diz respeito ao mecanismo do jogo. A grande novidade que este jogo traz porém é ser do gênero MMO, do inglês “massive multiplayer online”, onde uma multidão de jogadores compartilha um universo comum.

Figura 2.5 – Tela do jogo World of Warcraft

Fonte: Duncan Jones to direct 'World of Warcraft' movie. (Disponível em: <http://social.entertainment.msn.com/movies/blogs/the-hitlist-blogpost.aspx?post=996dda3c-23ba-4a17-bff6-689e596291ac>. Acesso em: 23 jun. 2014)

8 World of Warcraft. Disponível em: <http://us.battle.net/wow/pt/>. Acesso em: 23 jun. 2014.

3 O PROJETO

O presente trabalho foi feito visando contemplar as características de uma das modalidades vistas no capítulo anterior com a melhor forma possível de ser ter um RPG leve para poder ser executado em dispositivos móveis e jogado de forma casual.

A decisão foi criar um jogo o mais semelhante possível com o Final Fantasy IV, pois o objetivo principal é criar um núcleo bem definido quanto a mecânica do jogo e a história. Sendo assim, a escolha recaiu sobre aquele mais fiel ao gênero.

O projeto só modificou um aspecto em relação ao jogo escolhido como paradigma, que são os combates aleatórios, pois embora isso seja extremamente fiel as regras clássicas de RPG, na prática, torna o jogo desagradável, com o jogador ao andar pelo mapa ser surpreendido com a abertura de um combate sem que nada indicasse que isso estivesse por ocorrer, portanto, o jogo desenvolvido, assim como os outros três elencados, tem uma espécie de contrato com o jogador, que é: Você não entra em combate com nenhum inimigo que já não esteja vendo de antemão.

Definido o paradigma, um objetivo estabelecido, tendo em vista as indefinições inerentes ao mercado de tecnologia, foi possibilitar a migração entre as mais diversas plataformas existentes ou possíveis de existir, e a melhor forma de se atingir isto é evitando o acoplamento dos componentes de software. Desta forma, visou-se fazer um programa com um núcleo do jogo o mais desacoplado possível dos componentes de interface e estes, por sua vez, o mais desacoplados possível da tecnologia e do framework escolhidos para implementação. Para atingir este fim, o projeto foi feito com base em especificações e modelos teóricos. Todo componente teve sua concepção de forma isolada de forma a prever possíveis conexões com os demais.

Esta decisão teve por finalidade atingir o objetivo de obter a maior modularidade possível, pois o isolamento dos componentes possibilita uma maior manutenibilidade, ajudando a identificação de erros e a impedir suas propagações.

Estabelecidas as decisões de projeto, da perspectiva da definição do jogo em si, de forma bem simplificada, ele pode ser descrito da seguinte forma: Existe o personagem do jogador. Ele possui seus atributos, que são, pontos de vida, de magia, de defesa, de ataque, etc. Também possui itens que são encontrados ou comprados ao longo do jogo e que melhoram os atributos básicos ou tem suas próprias utilidades, e ele circula por um conjunto de lugares, dialoga com outros personagens, combate inimigos e cumpre determinados objetivos.

Basicamente o núcleo do jogo é possibilitar ao jogador andar com seu personagem por um cenário e desta forma desencadear os eventos possíveis estabelecidos no jogo e dirigidos pelo roteiro, como circular pelos mapas, entrar em combate com os inimigos, ler as narrações da história estabelecidas no roteiro, conversar com personagens não jogadores e resolver as missões (“quests”) que lhe são designadas.

Da perspectiva da interface, isso acontece com o jogador controlando o personagem através das setas direcionais, lendo as informações nos painéis de leitura e fazendo as escolhas definidas nos menus através do mouse.

A partir destas definições, o projeto divide os componentes em dois núcleos que podem ser assim analisados e nomeados: Núcleo do Jogo, e Núcleo da Interface.

3.1 O Núcleo do Jogo

Integram o núcleo do jogo o Sistema de Personagens, o Sistema de Batalhas, o Sistema de Mapas e o Sistema de Roteiro que podem ser assim brevemente descritos:

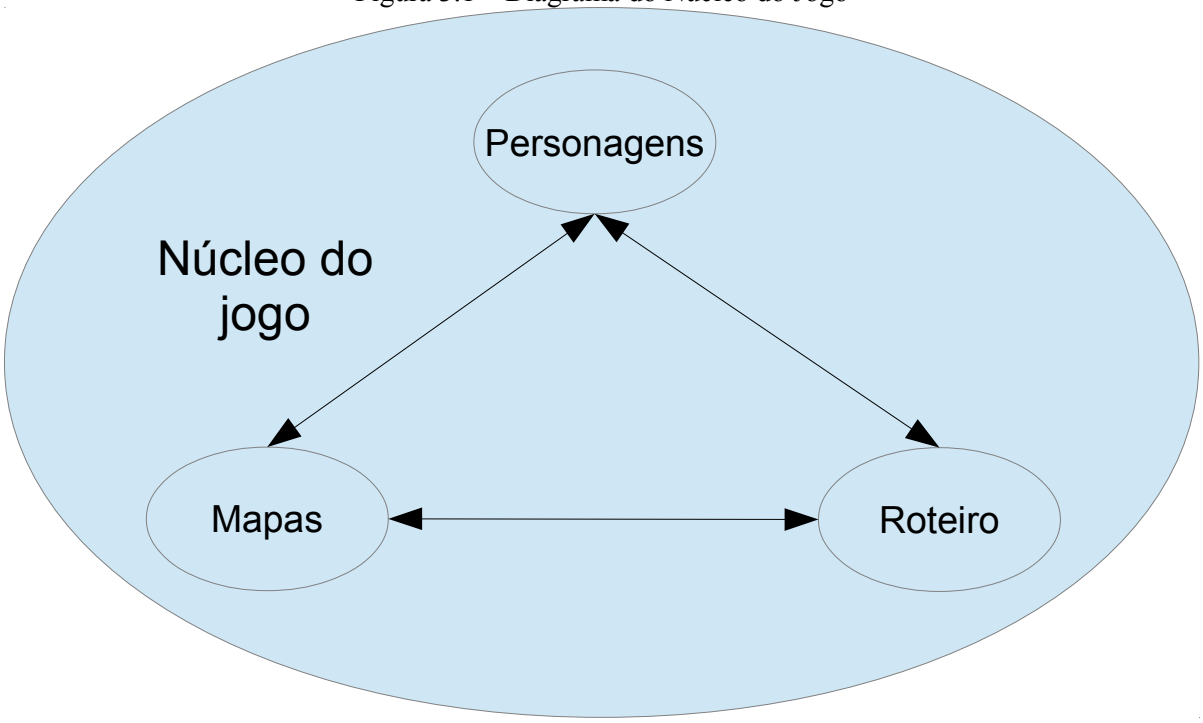
O sistema de personagens define as características dos personagens, armazena seus atributos e controla suas rotinas de ação.

O sistema de batalhas gerencia as situações de confronto entre personagens, controlando o sucesso dos ataques, o infligimento de danos, e itens deixados ao final do combate.

O sistema de mapas define a lógica por trás do cenário, definindo os tipos de terrenos por onde o personagem pode ou não pode passar, os pontos de transição entre os diversos mapas existentes no universo do jogo.

O sistema de roteiro trata do curso da história, o controle do seu desenrolar, das missões e das falas dos personagens.

Figura 3.1 – Diagrama do Núcleo do Jogo



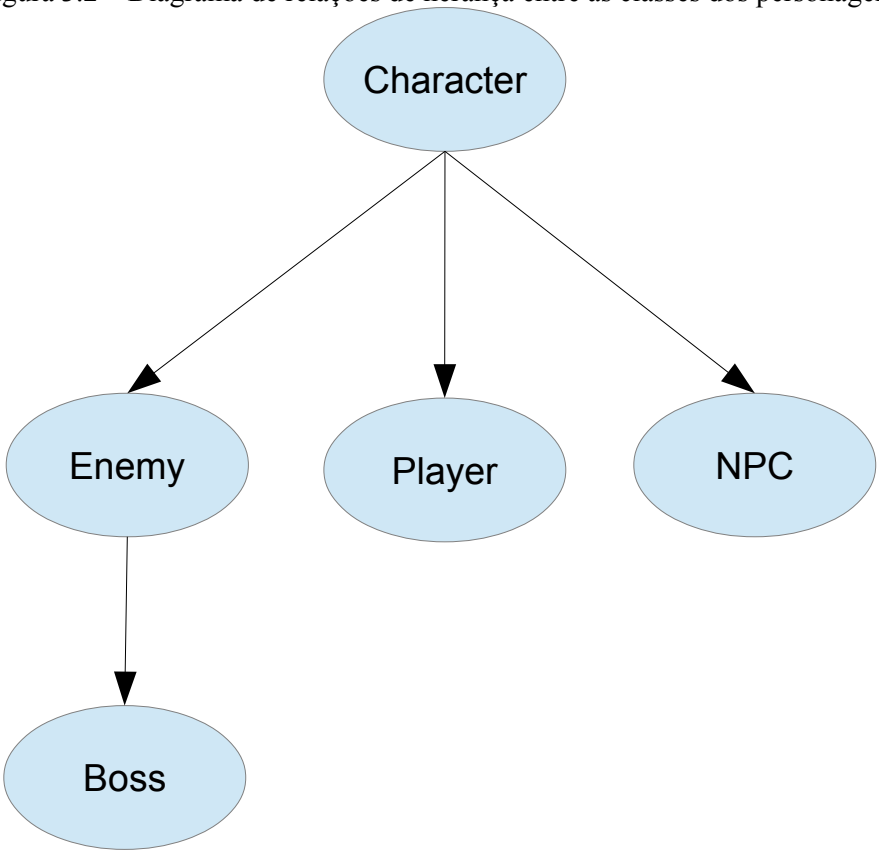
Fonte: Próprio Autor

3.1.1 Sistema de Personagens

O sistema de personagens controla as informações e transformações de dados relativos aos personagens. Contém informações sobre os personagens e funções que resolvem as questões de inventário, de evolução de seus atributos e de combate.

O que temos em primeiro lugar é uma classe básica dos personagens, que é herdada pelo personagem do jogador, pelos inimigos, pelos personagens não jogadores (non-player characters (NPC)) e pelos inimigos chefes (bosses), que por sua vez é herdada da dos inimigos.

Figura 3.2 – Diagrama de relações de herança entre as classes dos personagens



Fonte: Próprio Autor

A classe básica de todos os personagens (“character”) contém as rotinas e os atributos básicos de todo personagem, como a indicação de que o personagem está vivo, seus pontos de vida e de mágica, seus respectivos máximos, defesa e ataque, além da posição no mapa.

As funções básicas relacionadas aos combates são as de causar e de receber dano e funções relacionadas aos mapas, como atualização da posição, possibilidade de movimento e verificação de proximidade entre personagens para produção das consequências, como possibilitar os diálogos com personagens não jogadores e combates com os inimigos.

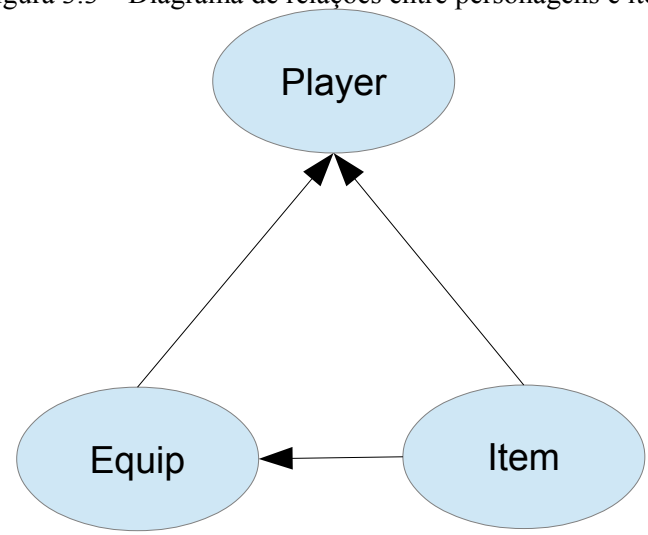
3.1.1.1 Itens e Equipamentos

Para ao personagem do jogador existe um sistema de itens e equipamentos. Os itens têm como origens as lojas e os resultados dos combates. Um item está associado a uma lista de itens e esta por sua vez associada a um jogador.

Embora seja possível implementar para outros personagens também utilizar itens, haja vista que a finalidade é modificar atributos é mais econômico fazê-lo diretamente, pois computadores não se divertem jogando. Entretanto, na perspectiva do jogador este é um

elemento muito importante. No caso, a classe do jogador tem como atributo uma lista de itens e uma lista especial categorizada de acordo com a funcionalidade do item que afeta os atributos do personagem chamada de equipamentos.

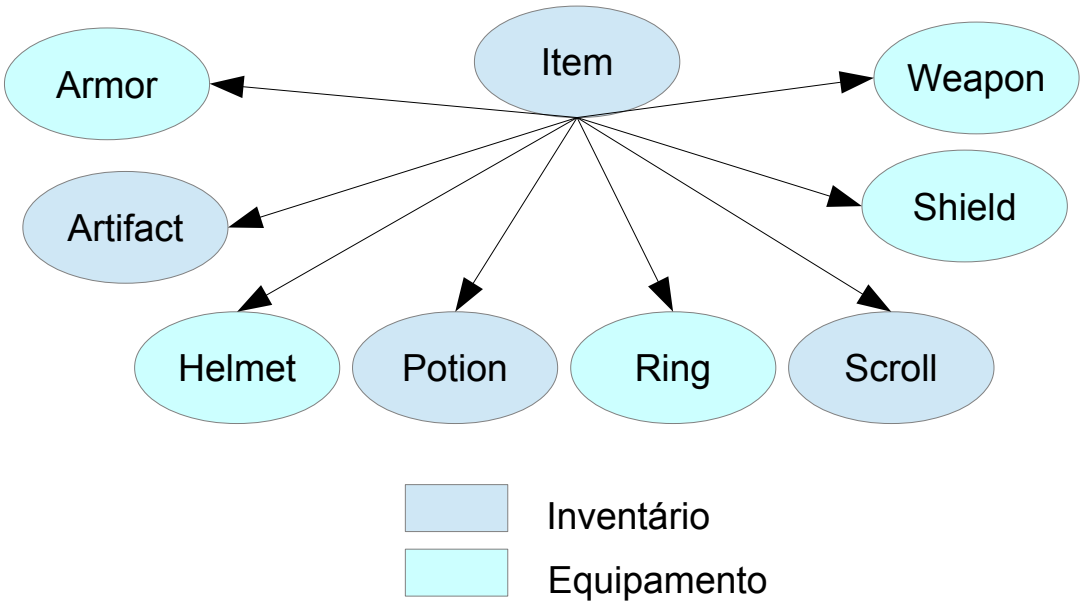
Figura 3.3 – Diagrama de relações entre personagens e itens



Fonte: Próprio Autor

Os itens podem ser de várias espécies, sendo alguns deles equipamentos.

Figura 3.4 – Diagrama de relações de herança entre itens



Fonte: Próprio Autor

As espécies de item foram definidas arbitrariamente por serem as mais comuns em jogos de RPG, e são, Armadura, Artefato, Elmo, Poção, Anel, Papiro, Escudo, Arma, cada uma delas respectivamente são descritas desta forma:

A Armadura serve de proteção para o personagem. O Artefato tem algum poder

mágico, podendo ser usado em batalha ou não. O Elmo também é uma proteção. A Poção consome-se com o uso e serve para recarregar pontos de vida ou de magia, ou incrementar algum atributo temporariamente. O Anel incrementa algum atributo, O Papiro consome-se com o uso e fornece algum poder mágico, Escudo é mais um equipamento que aumenta o atributo de defesa. A Arma aumenta o atributo de ataque e define a forma do dano.

3.1.2 O Sistema de Batalhas

O sistema de batalhas é gerenciado a partir do núcleo do jogo, ele tem como pré-condições o personagem do jogador e um inimigo estarem em posições que resultem verdadeiro o teste de proximidade. Satisfeita essa condição, ocorre uma sequência de comandos em que o jogador ataca, e caso o inimigo esteja vivo, ele ataca o jogador. A pós-condição para o fim do combate é pelo menos um dos envolvidos nele estar morto.

A rotina segue este pseudocódigo:

```
Combate() := {
	enquanto(Inimigo.Vivo && Jogador.Vivo) {
		AtaqueDoJogador(Inimigo);
		AtaqueDoInimigo(Inimigo);
		VerificaçãoDeFimDeCombate();
	}
}

AtaqueDoJogador(Inimigo) := {
	Se (LanceDeDado()+Jogador.Ataque>Inimigo.Defesa) {
		Inimigo.DanoRecebido(Jogador.DanoCausado());
		Se (Inimigo.Vivo=Falso) {
			SorteioDeItem();
			Jogador.XP+=Inimigo.Nivel;
			Se (Jogador.XP>=Jogador.PróximoNível
				Jogador.SobeDeNível();
		}
	}
}

AtaqueDoInimigo(Inimigo) := {
	Se (LanceDeDado()+Inimigo.Ataque>Jogador.Defesa)
		Jogador.DanoRecebido(Inimigo.DanoCausado());
}

VerificaçãoDeFimDeCombate() := {
	Se (Jogador .Vivo=Falso)
		FimDeJogo();
	Se (Inimigo.Vivo=Falso)
		FimDoCombate();
}
```

3.1.3 O Sistema de Mapas

O sistema de mapas trata dos cenários do jogo. Nele existem estruturas de dados que contêm as informações dos tipos de cenário que serão exibidos na tela e os pontos de transição entre mapas. A principal é um vetor bidimensional onde ficam armazenados números correspondentes aos desenhos em uma paleta, que formam o desenho do mapa.

Existem outras estruturas que armazenam as transições entre mapas, a primeira armazena os destinos e a segunda as origens. Na primeira os componentes são as coordenadas cartesianas do ponto onde, caso o personagem assuma essa posição, ele deixa o mapa corrente, e o número do mapa para onde ele vai.

A estrutura de origens, por sua vez é isomórfica a dos destinos em termos de tipo de informação, porém, nesta, a função das coordenadas armazenadas é para inicializar o personagem no mapa recém-chegado de acordo com o mapa de onde ele vem.

Além disso, ele contém informações básicas, como nome do mapa e quantidade de inimigos posicionados nele.

Do ponto de vista das funções, há as que indicam os pontos de transição e que indicam locais onde não é possível o personagem atravessar.

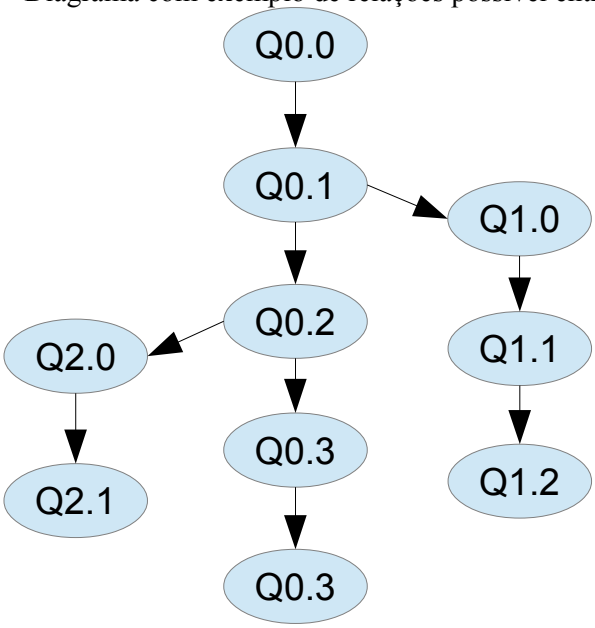
3.1.4 O Sistema de Roteiro

O roteiro contém as informações que dizem respeito a história do jogo. Todas as falas e estados das jornadas ficam armazenadas neste componente.

O sistema tem como centro uma classe que armazena jornadas, chamadas de “quests”, que controla um conjunto de informações e um conjunto de ações encadeadas que o personagem pratica e que levam a progressão do estado da história.

É possível haver muitas jornadas, e elas estarem organizadas em forma de árvore, o que permite alguma alinearidade no desenrolar da história, como ilustra o diagrama abaixo:

Figura 3.5 – Diagrama com exemplo de relações possível entre “quests”



Fonte: Próprio Autor

No caso, existiria uma “quest” principal “Q0” e duas “side-quests” “Q1” e “Q2” com seus respectivos estados. Para cada estado existem informações distintas, que são falas de personagens, narrações de história e posições de personagens.

As mudanças de estado são definidas sobre determinados eventos que alteram o estado da história.

Os eventos podem ser de diversas espécies como conversar com um NPC, matar um Boss, atingir determinado nível, ou entrar em um mapa.

Então, a classe “quest” armazena os seguintes dados em quatro listas:

– Transições de estados;

– Falas e narrações;

– Posições de NPC;

– Posições de Boss.

Para essas quatro listas também foram definidas as classes que armazenam essas informações.

As transições armazenam o status da “quest” onde ela se aplica, o evento e a identidade do ente que dispara a transição, no caso em que se aplique, por exemplo, de ser uma transição disparada por uma conversa com um NPC ou eliminação de um Boss.

As falas e narrações contêm o texto que será exibido ao jogador de acordo com o status da “quest”, e contem a identidade do NPC, o status da “quest” e o conteúdo da fala.

As posições de NPC e Boss são isomórficas, e contém a identidade do personagem o

status da “quest” e a indicação do mapa e a posição.

A classe “quest”, tem apenas uma função, que é a que verifica se ela chegou ao fim.

Para controlar a história, existe ainda uma classe que controla o roteiro (screenplay), ela é a responsável para, com base nas informações contidas em uma lista de “quests”, operar as mudanças de estados.

3.2 Interface Gráfica

A interface gráfica é a parte do programa responsável por fazer com que os resultados do processamento do jogo sejam postos na tela. Tendo em vista o jogo ter muitos componentes, há diversos elementos que precisam existir para oferecer uma boa qualidade de experiência, dos elementos de interação, à arte do jogo.

3.2.1 Elementos de Interação

Os elementos de interação são aqueles que fazem a parte mais semelhante ao ambiente de um sistema operacional. Embora sempre pensemos mais na parte artística e de desenhos, os elementos básicos são indispensáveis para que tenhamos uma experiência interativa rica, pois do contrário não seria um jogo, mas uma animação apenas.

3.2.1.1 Seta

Haja vista o jogo ter elementos que precisam ser apontados e selecionados. Foi necessário criar este elemento trivial de interfaces gráficas que basicamente desenha uma seta na tela conforme a posição do mouse.

3.2.1.2 Painéis

Os painéis servem para colocar texto ou figuras na tela. Todos derivam de uma classe “Panel” que contem os elementos básicos de dimensões, fontes dos caracteres e bordas.

3.2.1.2.1 Painel de Leitura

O Painel de Leitura serve para exibir as narrações do jogo e as falas dos personagens, funciona armazenando o texto em páginas e parágrafos. Estes últimos são entidades definidas para dar a cadência da leitura do texto. A página exige que sempre o texto comece a ser exibido com o painel em branco e o parágrafo apenas em uma nova linha.

O componente imprime texto em uma região delimitada da tela e faz o efeito de exibição progressiva do conteúdo com um retângulo da mesma cor do fundo que cobre o texto e é progressivamente diminuído, linha após linha. O controle ainda permite a leitura rápida clicando com o mouse sobre ele, o que causa a remoção dos retângulos que cobrem o texto.

3.2.1.2.2 Menu

O Menu é um painel em que exibe texto em linhas correspondentes às opções definidas pelo jogo e conforme a linha em que o mouse está sobre, modifica sua cor diferenciando-a das demais, e quando é clicado desaparece e assume um valor numérico correspondente à escolha que pode ser lido pelo componente do núcleo do jogo do qual dependa essa escolha.

3.2.2 Personagem

O primeiro dos elementos de animação, o personagem é desenhado na tela de acordo com uma função que combina a posição dada pelo núcleo do jogo, a janela do “tile engine” e imprime o personagem na tela.

A posição do personagem é dada pelo núcleo do jogo em um par de números inteiros. A posição na tela é dada pela proporção resultante da multiplicação pela granularidade (quantidade mínima de pixeis) das figuras desenhadas na tela.

A movimentação do personagem na tela, embora siga o incremento na tela para suavizar a exibição, é restringida pela granularidade do bloco das figuras do núcleo do jogo. Assim, um movimento é discreto, nunca sendo interrompido de forma a ficar o personagem em uma posição fracionária em relação ao tamanho do bloco (TamanhoBloco) das figuras, algo semelhante a um tabuleiro de xadrez, onde uma peça nunca fica em um posição entre duas casas.

Para isso foi criado um sistema duplo de posicionamento, um do núcleo do jogo, que

segue os dados fornecidos pelo sistema de mapas (PosiçãoMapa) e outro dado pelo tamanho real no mundo (PosiçãoMundo) conforme o tamanho dado pelo bloco, ambos em uma relação de proporção direta:

PosiçãoMundo = PosiçãoMapa * TamanhoBloco

Para a ocorrência de uma movimentação, portanto, a pré-condição, o personagem estar em uma *PosiçãoMundo* cujo resto da divisão inteira pelo *TamanhoBloco* seja zero.

A animação é processada com o incremento da *PosiçãoMundo* de acordo com a velocidade do personagem e a pós-condição é igual a pré-condição, o personagem estar em uma *PosiçãoMundo* cujo resto da divisão inteira pelo *TamanhoBloco* seja zero.

3.2.3 "Tile Engine"

O "tile engine" é o elemento que imprime o cenário na tela. Diretamente ligado ao sistema de mapas, seus principais componentes são: uma imagem que é a fonte do conjunto de blocos a serem imprimidos na tela, uma espécie de paleta; um vetor bi-dimenional que contém as informações sobre a forma do mapa. O "tile engine" faz um produto entre os dois componentes, onde cada para elemento da matriz é desenhado na tela seu correspondente na imagem fonte do conjunto de blocos.

O sistema básico de funcionamento se dá através de um laço duplo que desenha na tela os blocos conforme o mapa.

```
DesenhaCena() := {
	for(i = 0; i < Mapa.Largura; i++) {
		for(j = 0; j < Mapa.Altura; j++) {
			x = i * TamanhoBloco
			y = j * TamanhoBloco
			Desenha((x,y),Bloco[Mapa[i][j]]
		}
	}
}
```

O "tile engine" também acrescenta ao duplo sistema de posicionamento dos personagens mais uma posição, a posição na tela (PosiçãoTela).

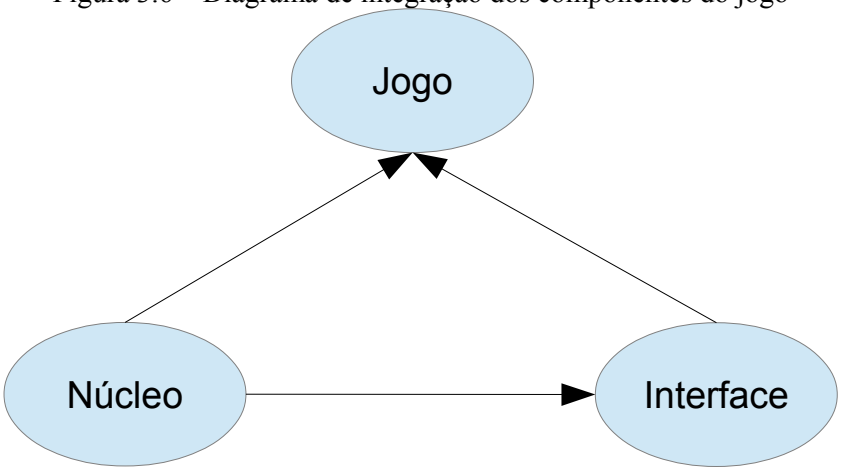
Quando o tamanho do mapa é menor do que o da tela, a *PosiçãoMundo* é igual a *PosiçãoTela*, porém, caso contrário, é necessário acrescentar-se um pivô, geralmente a posição do personagem do jogador, e uma constante que indica o centro a partir de onde a janela de exibição se desloca, o que modifica a função original para:

```
DesenhaCena() := {
	se (Mapa.MaiorQueTela) {
		PosX,Y = PivotX,Y – CentroX,Y * TamanhoBloco
		se (PosX,Y < 0)
			PosX,Y = 0;
		se (PosX,Y > Mapa.Largura * TamanhoBloco – LarguraTela)
			PosX,Y = PivotX,Y – CentroX,Y * TamanhoBloco
	}
	MapaX,Y = PosX,Y / TamanhoBloco;
	para (i = MapaX; i < LarguraTela + MapaX; i++) {
		para (j = MapaY; j < AlturaTela + MapaY; j++) {
			x = i * TamanhoBloco
			y = j * TamanhoBloco
			Desenha((x,y)Bloco[Mapa[i][j]]
		}
	}
}
```

3.3 Integração dos Componentes

Os componentes do jogo são integrados entre os elementos básicos e os de interface em um terceiro elemento, que é o próprio jogo.

Figura 3.6 – Diagrama de integração dos componentes do jogo



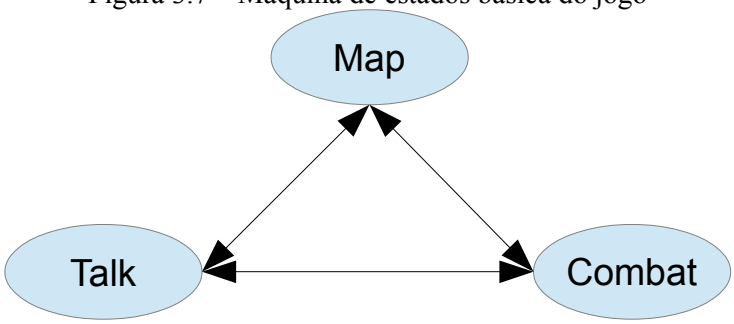
Fonte: Próprio Autor

3.3.1 Entre Elementos Básicos

A operação do núcleo do jogo utiliza uma máquina de estados que faz com que o programa alterne entre as rotinas. Os principais estados são a navegação do personagem pelos cenários utilizando as teclas de direção, a leitura de painéis, e os menus de combate. Ela também avalia todas as condições de alternância entre elementos de interface, executa as rotinas pertinentes aos estados, e as verificações necessárias para efetuar as transições entre os estados.

A máquina de estados essencial do jogo foi então concebida desta forma, um estado em que o personagem transita pelo mapa, outro de leitura dos painéis e outro de combate:

Figura 3.7 – Máquina de estados básica do jogo



Fonte: Próprio Autor

Na prática, a partir deste modelo, algumas modificações de detalhamento foram adicionadas, mas essencialmente o jogo é o que existe acima.

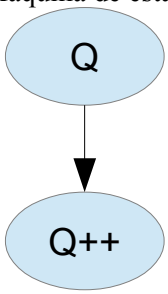
3.3.2 Com o Roteiro

Na parte do roteiro, as missões são armazenadas em arquivos que contem as informações que resultam no andamento do jogo. Também foi concebida uma máquina de estados para lidar com esta parte do jogo. A base da máquina de estados do roteiro é a mais genérica possível e a mais simples possível: de um estado chegamos em outro estado e a ele não retornamos. A mudança é incremental, por mais que na prática possamos ter um roteiro bastante complexo, como uma história com uma "quest" principal bastante extensa e com muitas "side-quests", como por exemplo o da Figura 3.5, onde o primeiro número representa a identidade da "quest" e o segundo, o estado da quest. No caso existe uma "quest" principal, Q0.x e "side-quests" Q1.x e Q2.x que são habilitadas em nos estados da "quest" principal Q0.1 e Q0.2 respectivamente.

No caso, trata-se de uma árvore, porém o traço essencial é a ordenação estrita, pois seja qual for a combinação de evolução na "quests", a evolução sempre é incremental. O estado sempre avança, mesmo que seja uma evolução em quests paralelas, na soma existe uma evolução progressiva global incremental.

Em síntese, colapsando o progresso de todas as quests como o progresso geral da história, a história sempre avança, a máquina de estados básica que trata do roteiro é essa:

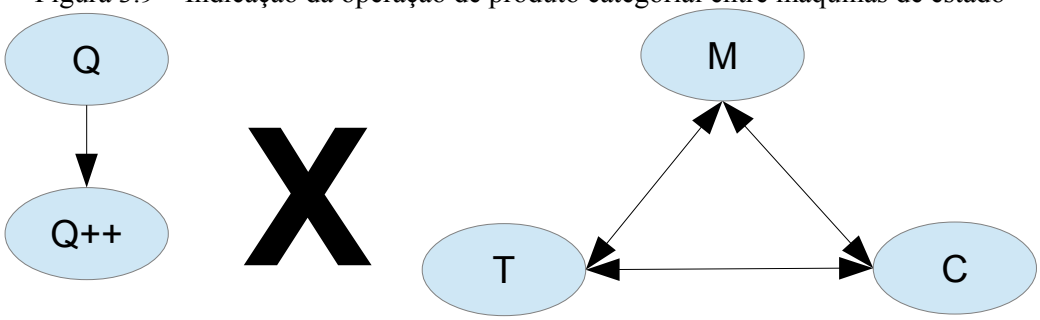
Figura 3.8 – Máquina de estados do roteiro



Fonte: Próprio Autor

Assim, a implementação do jogo baseou-se no produto categorial[9] dessas duas máquinas de estados:

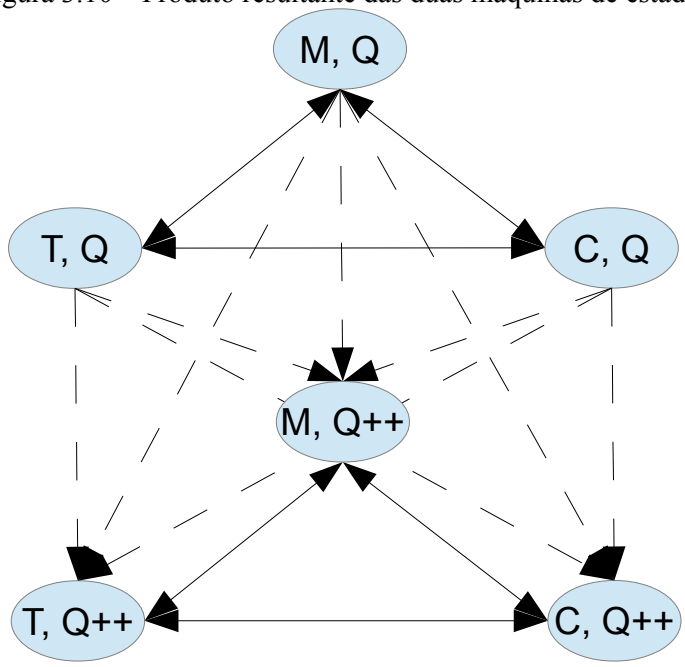
Figura 3.9 – Indicação da operação de produto categorial entre máquinas de estado



Fonte: Próprio Autor

Que resultou em:

Figura 3.10 – Produto resultante das duas máquinas de estados



Fonte: Próprio Autor

9 MENEZES, P. B.; Haeusler E. H. Teoria das Categorias para Ciência da Computação. 2. ed. Porto Alegre: Bookman, 2008. p. 162.

Como se pode observar, o produto categorial resultou na combinação das possibilidades de eventos da máquina de estados básica do jogo com as da do roteiro. O que intuitivamente é correto, pois temos possibilidades de evolução da história, mudança de estado do roteiro, pelas ocorrências de falas, pelas andanças pelo mapa e pelos combates. O produto mantém as relações intrínsecas da base do programa e do roteiro.

4 IMPLEMENTAÇÃO

Na etapa de implementação foi utilizado o paradigma de orientação a objetos, com algumas técnicas e exploração de novos recursos de programação funcional. Foram utilizadas a linguagem C# e os frameworks .NET 4 e XNA 4.0 da Microsoft e ambiente de desenvolvimento o Microsoft Visual C# 2010 Express Edition em um sistema operacional Windows Vista.

Este capítulo segue a ordem cronológica de implementação e integração dos componentes.

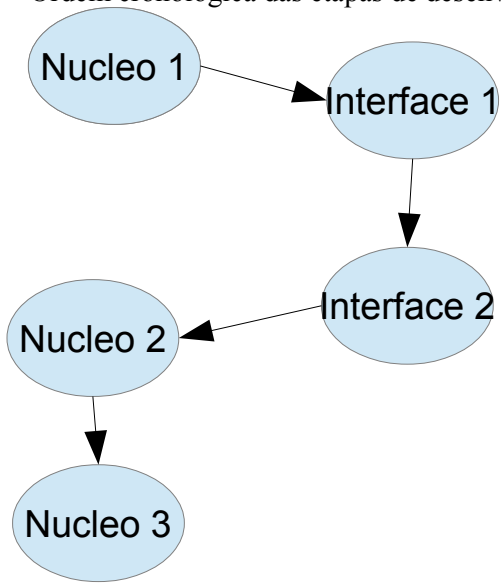
Embora feita de forma parcialmente sobreposta ao projeto, de forma semelhante aos ciclos de desenvolvimento dos métodos ágeis, teve como principais desafios atingir um objetivo estabelecido no projeto que foi a criação de componentes da forma menos acoplada possível para a finalidade de possibilitar futuramente portar para múltiplas diferentes plataformas.

Em especial, o objetivo principal é de futuramente migrar para plataformas móveis, então, além de reduzir o acoplamento ao máximo possível, o componente central do projeto foi o núcleo com a lógica do jogo.

Esta decisão, embora tenha tornado um pouco mais difícil a tarefa, revelou-se vantajosa no fim, pois, embora a dificuldade inicial de se testar a execução do mecanismo do jogo sem uma saída gráfica, e a dificuldade do trabalho extra nas conexões entre os componentes, o código ficou mais organizado e melhor compartimentado.

A implementação, conforme o projeto, se divide em basicamente duas partes, Núcleo do Jogo e Interface Gráfica, mas foi fragmentada em ciclos. A ordem cronológica na prática do processo de desenvolvimento do código foi a seguinte:

Figura 4.1 – Ordem cronológica das etapas de desenvolvimento



Fonte: Próprio Autor

4.1 Núcleo do Jogo

O Núcleo do jogo foi a primeira parte implementada, que contêm a base, que são o personagem jogador e os inimigos e seus combates. Esta primeira fase foi testada em modo texto diretamente e depois em formulários.

4.1.1 Primeiro Ciclo

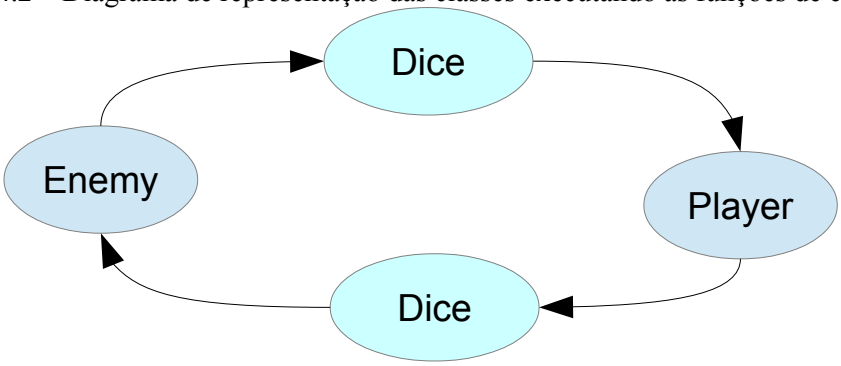
O núcleo começou esta fase utilizando apenas a instanciação de dois objetos e executava as rotinas de combate, abstraindo qualquer informação sobre posição ou relação de qualquer espécie com alguma história.

Foi criada uma classe estática que controla uma simulação de sorteio de dados (CygnusRPGLibrary/Game/Dice.cs).

Os combates invocam esta classe com suas funções que simulam os dados clássicos dos jogos de RPG, de 20, 12, 10, 8, 6 e 4 lados.

Assim, a base do jogo é basicamente um laço que se encerra com a eliminação de ao menos um dos personagens.

Figura 4.2 – Diagrama de representação das classes executando as funções de combate



Fonte: Próprio Autor

Para a criação dos personagens foi criada a classe "Character" (CygnusRPGLibrary/Chars/Character.cs) com os seguintes atributos e funções básicas utilizadas pelas classes "Enemy" e "Player" (CygnusRPGLibrary/Chars/Enemy.cs e CygnusRPGLibrary/Chars/Player.cs):

bool Alive, indica se o personagem está vivo ou não.
int MaxHP , número máximo de pontos de vida.
int HP, número de pontos de vida.
int MaxMP, número máximo de pontos de mágica.
int MP, número de pontos de mágica.
int Defense, nível de defesa do personagem.
int Attack, nível de ataque do personagem.
int Level nível do personagem.

Os combates também têm influência dos itens, criados na classe "Item" (CygnusRPGLibrary/Items/Item.cs) que estão equipados pelo jogador, controlados pela através "Equip" (CygnusRPGLibrary/Chars/Equip.cs). As principais funções são:

A função que mede o dano infligido é bastante simples. Ela recebe um valor de entrada, o valor do dano, subtrai dos pontos de vida do personagem e verifica se ele está vivo.

```
public void InflictedDamage(int damage)
{
      this.HP -= damage;
      if (this.HP <= 0)
      this.Alive = false;
}
```

A outra função, que lida com o ataque em si, envolveu a criação de uma expressão

regular para lidar com os danos que os personagens podem causar em combate.

O dano é informado por duas parcelas, uma pelos dados e outra, por um incremento, como por exemplo “1d6+2” que, no caso, é resultado do sorteio de um dado de seis lados mais dois. Há ainda a possibilidade de o personagem causar mais de um dano, como por exemplo, uma arma que além de seu dano normal, causa ainda um dano mágico. A expressão tem essa forma:

*[#]d#[+#](;[#]d#[+#])**

O formalismo utilizado acima assemelha-se a um de expressões regulares[10] e segue as seguintes regras abaixo:

“#”, número;

“[ ]”, elemento opcional;

“( )*”, elemento iterativo que não precisa ocorrer;

“( )+”, elemento iterativo que precisa ocorrer pelo menos uma vez;

“(,)”, opção entre dois valores.

A função faz um parser de uma string, pois o projeto teve como objetivo ter seu conteúdo fornecido por arquivos externos conforme o formalismo simples acima definido e após roda os sorteios:

A chamada de função recebe como parâmetro expressão e antes do laço principal declara a variável “value” com o valor de retorno em 0.

```
public int DealDamage(string damage)
{
    int value = 0;
```

O laço principal divide a expressão e unidades atômicas “[#]d#[+#]” separadas por “;”:

```
    foreach (var damUnit in damage.Split(';'))
    {
        int plusPos = damUnit.IndexOf('+');
        string dicePart = damUnit;
        if (plusPos!=-1)
        {
            value += Convert.ToInt32(damUnit.Substring(plusPos + 1));
            dicePart = damUnit.Substring(0, damUnit.IndexOf('+'));
        }
        int dicePos = dicePart.IndexOf('d');
        int numberOfDices = 1;
        if (dicePos > 0)
```

10 MENEZES, P. B.; Haeusler E. H. Teoria das Categorias para Ciência da Computação. 2. ed. Porto Alegre: Bookman, 2008. p. 51.

```
            numberOfDices = Convert.ToInt32(dicePart.Substring(0, dicePos));
        int diceSides = Convert.ToInt32(dicePart.Substring(dicePos + 1));
```

A segunda parte do laço principal executa os lances de dados segundo o padrão.

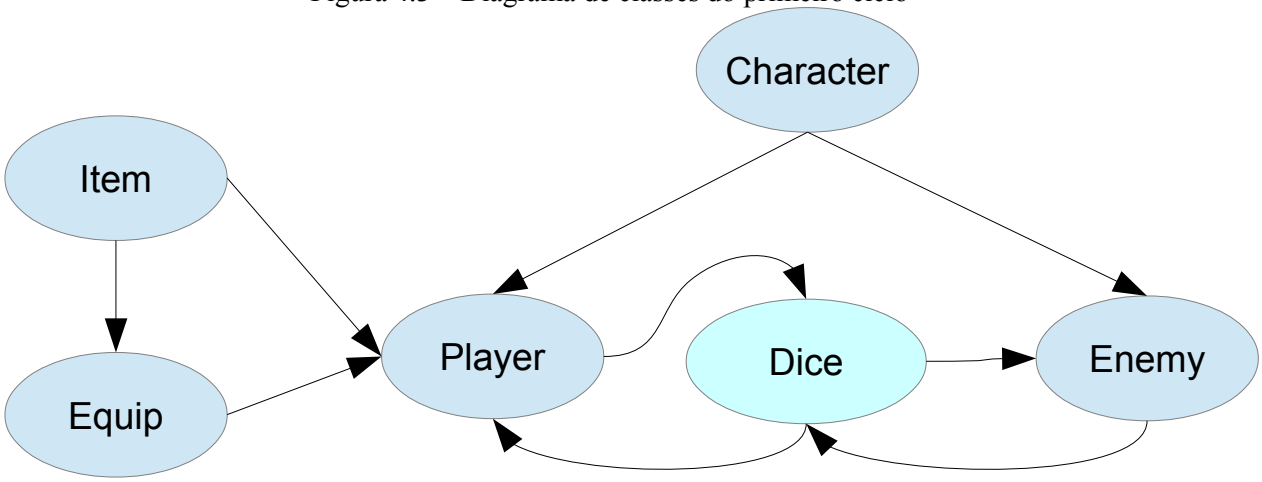
```
        for (int i = 0; i < numberOfDices; i++)
            switch (diceSides)
            {
                case 4:
                    value += Dice.d4();
                    break;
                ...
            }
    }
```

Por fim, a função retorna o valor do dano.

```
    return value;
}
```

Assim, ao fim da implementação do primeiro ciclo o programa assumiu a seguinte forma:

Figura 4.3 – Diagrama de classes do primeiro ciclo



Fonte: Próprio Autor

4.1.2 Segundo Ciclo

O segundo ciclo foi cronologicamente precedido dos primeiros dois ciclos do desenvolvimento da interface gráfica, analisados no próximo capítulo (elementos básicos e "tile engine"), e tiveram por objetivo a implementação dos elementos de posição e movimentação dos personagens na tela. O sistema de mapas também teve o início de seu desenvolvimento nesta etapa.

O sistema de animação envolve controlar a posição dos personagens e está fortemente ligado ao elemento de interface que exibe os personagens e o cenário na tela. Baseado na classe básica dos personagens, “Character”, e na classe que controla as informações dos mapas, “MapInfo”, o sistema de animação controla as possibilidades de movimento.

A classe “MapInfo” (CygnusRPGLibrary/MapSystem/MapInfo.cs) foi a primeira a utilizar dados fornecidos por arquivos externos. No caso, foi criada uma gramática para lidar com a aquisição de dados do arquivo.

O formato do arquivo é bastante simples, apenas uma lista de atributos e uma matriz, como o exemplo abaixo:

```
NAME
string
ORIGINS
(# # #)+
[TRANSITIONS
(# # #)+]
[LEVEL
#]
[ENEMIES
#]
[RENEWABLE
(0,1)+]
MAP
(#)+ i x j
```

Os nomes são bastante autoexplicativos, o nome, origem (local do ponto de inicialização do personagem no mapa de acordo com o mapa de origem, transição (o ponto onde o personagem sai do mapa e para qual mapa vai), o nível do mapa, a quantidade de inimigos que moram ali, renovável, sobre possibilidade de no mapa sempre haver inimigos por mais que você mate todos e o mapa propriamente dito, uma matriz bidimensional.

Abaixo, um exemplo de conteúdo do arquivo de texto do mapa:

```
NAME
Cave of Lisgow
ORIGIN
2 3 0
TRANSITION
2 2 0
LEVEL
1
ENEMIES
2
RENEWABLE
0
MAP
```

```
8 8 8 8 8 8 8 8
8 4 4 4 4 4 4 8
8 4 6 4 4 4 4 8
8 4 4 4 4 4 4 8
8 8 4 4 4 4 4 8
8 8 4 4 4 4 8 8
8 8 4 4 9 9 9 8
8 8 4 4 9 8 8 8
8 8 8 8 8 8 8 8
```

O código do parser é bastante simples e explora a alternância entre linhas que indicam nomes dos campos e dados.

```
private enum FieldName { NONE, NAME, ORIGIN, TRANSITION, LEVEL, ENEMIES, RENEWABLE,
MAP }
    while (!sr.EndOfStream)
    {
        string s = sr.ReadLine().Trim();
        FieldName auxFieldName = GetFieldName(s);
        if (auxFieldName!=FieldName.NONE)
            currentFieldName = auxFieldName;
        else
            DealWithFields(currentFieldName, s);
    }
```

O comando de aquisição da matriz bidimensional, que contém expressões lambda e composições que seguem o estilo de programação funcional:

```
tiles.Add(new List<string>(s.Split(' ')).ConvertAll(new Converter<string, int>(x =>
Convert.ToInt32(x))));
```

A classe ainda contempla funções para gerenciar a transição entre mapas e posições por onde o personagem não pode andar. No caso, por simplicidade não foram implementadas maiores abstrações do que os números pré-definidos do tileset:

Figura 4.4 – Paleta de blocos de cenário

Fonte: Próprio Autor

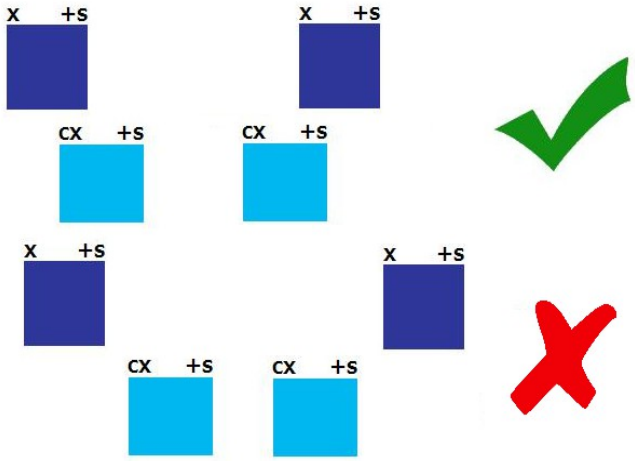
```
public Boolean Blocked(int i, int j)
{
    if (tiles[j][i] >= 8)
        return true;
    else
        return false;
}
public Boolean Transition(int i, int j)
{
```

```
    if (tiles[j][i] == 6)
        return true;
    else
        return false;
}
```

Na classe “Character” foi criada função para verificar a situação de proximidade entre personagens, que são gerenciadas pelo núcleo do jogo, como iniciar conversas com NPCs e combates com inimigos. A função verifica a posição de dois objetos “Character” no mesmo perímetro.

```
public bool Proximity(Character c)
{
    return worldMapPosition.X + size >= c.worldMapPosition.X &&
    worldMapPosition.X <= c.worldMapPosition.X + c.size &&
    worldMapPosition.Y + size >= c.worldMapPosition.Y &&
    worldMapPosition.Y <= c.worldMapPosition.Y + c.size;
}
```

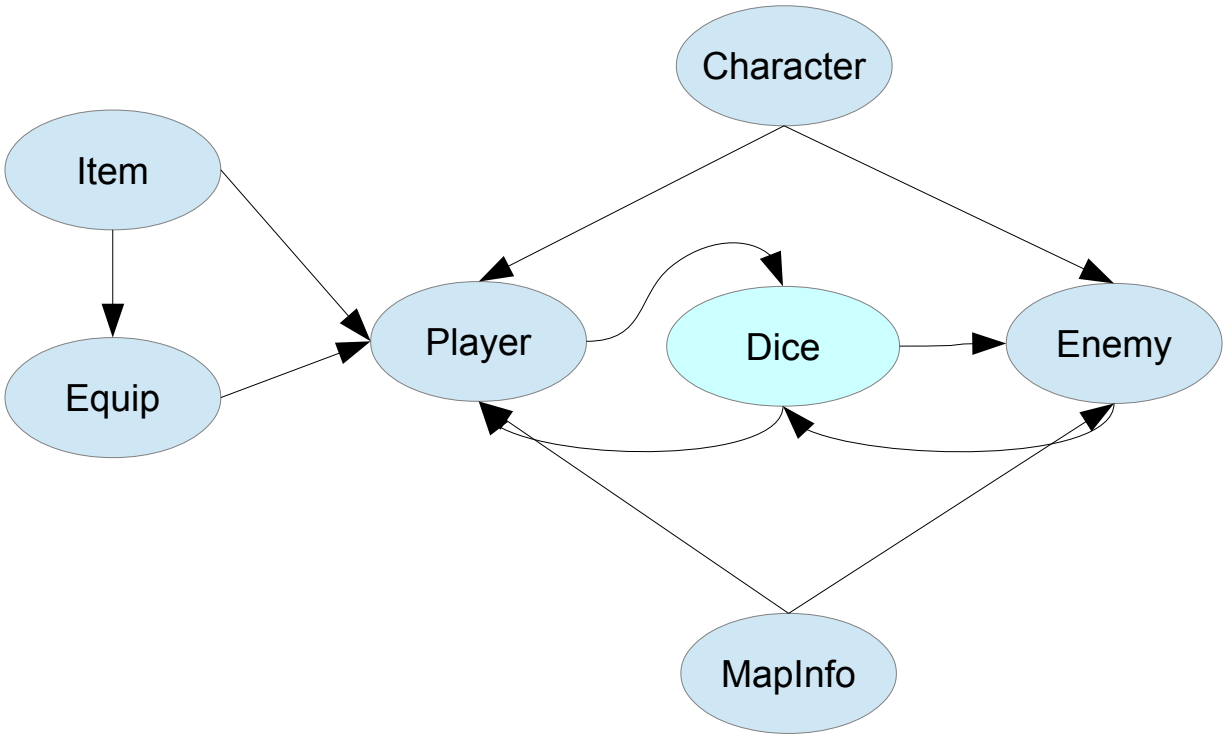
Figura 4.5 – Ilustração dos casos da função proximidade em X



Fonte: Próprio Autor

Assim, o diagrama ao fim da implementação do segundo ciclo ficou assim:

Figura 4.6 – Diagrama de classes do primeiro ciclo



Fonte: Próprio Autor

4.1.3 O Terceiro Ciclo

No terceiro e último ciclo da implementação do núcleo, houve a implementação do roteiro e adaptação da máquina de estados que controla o jogo. Além disso, também foi criada a classe “Boss” (CygnusRPGLibrary/Chars/Boss.cs), que herda da “Enemy”.

Esta última etapa foi a mais complexa, pois enquanto as duas primeiras apenas o nível de conexão entre as duas era relativamente baixo, para a implementação do roteiro, o nível de conexão é quase total, uma vez que o roteiro envolve todos os aspectos do jogo.

A primeira vista, parecia que esta abordagem “bottom-up” adotada até então seria bem-sucedida para a criação de um jogo com o nível de complexidade de um RPG, porém, ela se esgotou ao se tentar implementar o roteiro, quando surgiram problemas na possibilidade de se combinar a lógica do ciclo básico do jogo com a lógica da progressão de estados da história. A grande dificuldade passou a ser transicionar a evolução dos estados da história dados pelo roteiro sem que fosse necessário reescrever trechos de código semelhantes para cada transição da máquina de estados básica, o que acarretaria uma explosão de estados.

Este problema pode ser descrito assim: no estado inicial da história, você tem uma rotina que controla a existência e posição dos inimigos no mapa do jogo. Se formos pensar em

um jogo como Pac-Man, os mesmos quatro fantasmas são exibidos na tela em qualquer circunstância. Entretanto, um inimigo que existe apenas em um dado momento da história e depois é eliminado, causaria a ocorrência de trechos de códigos especiais para lidar com a esta situação, trazendo diversos desvios condicionais no código, o que inviabilizaria a possibilidade de se utilizar conteúdo externo, pois de acordo com o conteúdo, deveria existir sempre sua contraparte diretamente codificada, além de um código muito confuso.

A partir deste ponto, foi tomada uma decisão de projeto de evitar esta situação. Para isso, foi adotada uma abordagem "top-down", tendo como ponto de partida os arquivos de conteúdo.

Abstraindo tudo que já estava pronto, a solução foi especificar o programa em torno da seguinte questão:

Como serão os arquivos de conteúdo?

Obviamente, a parte gráfica ficou inalterada, os mapas apenas receberam um cabeçalho indicando as transições, embora a multiplicidade de mapas não fosse algo que essencialmente envolvesse o roteiro. A questão maior foi como criar o roteiro, o andamento da história evitando uma explosão de estados.

A máquina de estados original era a descrita na Figura 3.7:

1 – Andar pelo mapa;

2 – Exibir texto;

3 – Combate.

O roteiro porém necessitava ser pelo menos um script linear, como um livro em que você começa a ler na primeira página e termina na última, porém mais do que isso, um bom RPG divide a história em "quests" que combinam sequencialidade e paralelismo. Ou seja, seria minimamente uma lista, se o jogo seguisse uma história totalmente linear, ou uma árvore, com a história sendo mais complexa.

Na solução encontrada, o jogo precisava ser um produto de duas máquinas de estado: A do núcleo do jogo e a do roteiro. Do ponto de vista da especificação, um produto categorial, o que resultou na repetição da máquina básica em cada nodo do roteiro, como descrito na figura 3.10.

Foi adicionada a classe "Quest" (CygnusRPGLibrary/Game/Quest.cs) para lidar com as "quests". A "quest" é um elemento central do roteiro pois são nelas que se baseiam a história. De acordo com o projeto, foi implementada a "quest" tendo como elemento principal o seu estado, dado por um inteiro, que indica sua progressão e um booliano que indica se está completa ou não, além de listas para posicionamentos de NPCs, Bosses, falas, e o principal,

eventos de transição de estados.

A classe “Quest” assim ficou:

```
public int Id;
public string Name;
public int Status;
public bool Complete;
public List<QuestTransition> questTrasitions;
public List<QuestLore> questLores;
public List<QuestNPCPosition> questNPCPositions;
public List<QuestBossPosition> questBossPositions;
```

Também foram criadas classes auxiliares QuestTransition, QuestLore, QuestNPCPosition, QuestBossPosition que ficaram assim:

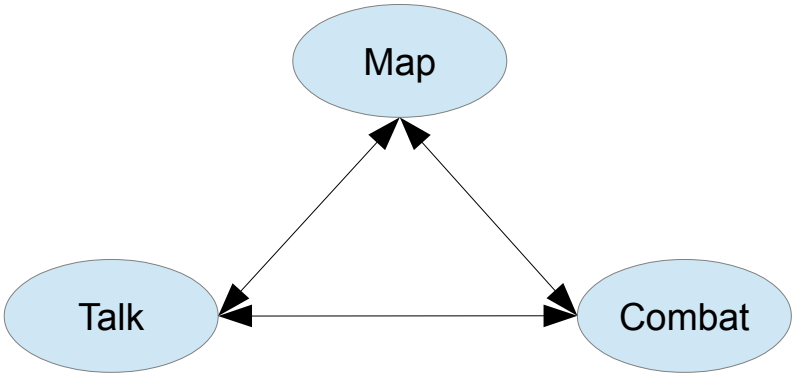
```
public class QuestBossPosition
{
    public int Id;
    public int Status;
    public int Map;
    public Pair<int, int> MapPosition;
}
public class QuestNPCPosition
{
    public int Id;
    public int Status;
    public int Map;
    public Pair<int, int> MapPosition;
}
public class QuestTransition
{
    public int Status;
    public string Event;
    public int Id;
}
public class QuestLore
{
    public int nPCID;
    public int Status;
    public bool Viewed;
    public LoreContent loreContent;
}
```

Todas basicamente armazenam a identidade do elemento e o status a partir de quando podem ser disparadas.

Além da classe “Quest”, foi implementada o principal elemento do roteiro, a máquina de estados que controla todo o jogo, que foi implementada na classe “Screenplay” (CygnusRPGLibrary/Game/Screenplay.cs).

A máquina de estados básica do jogo segundo o projeto era:

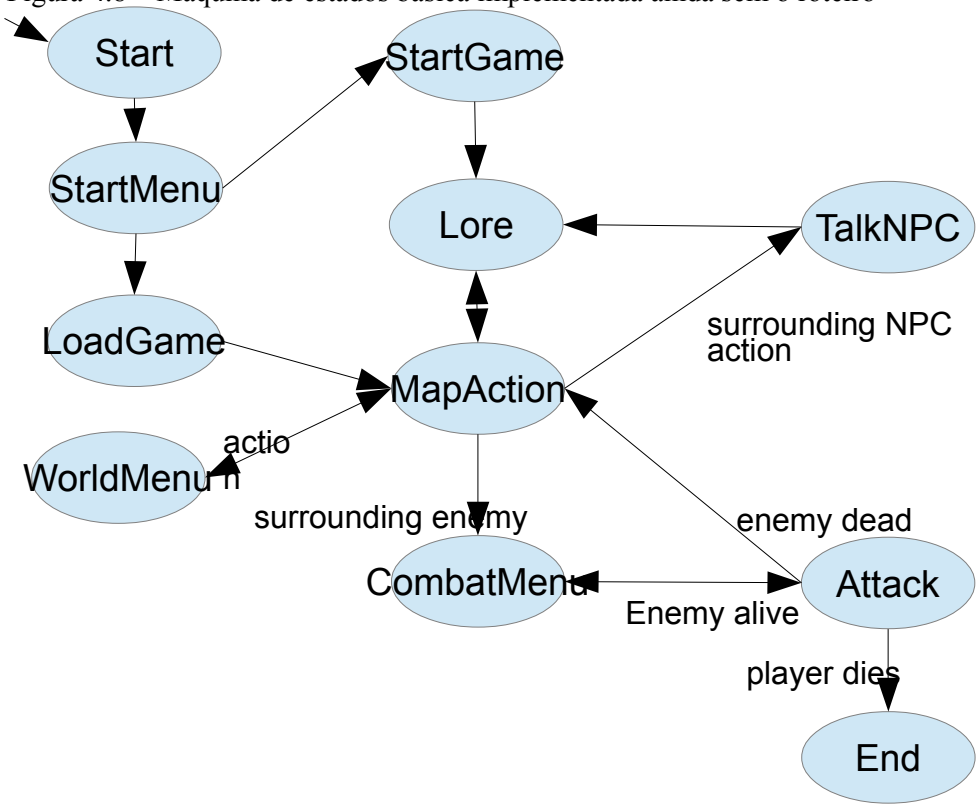
Figura 4.7 – Máquina de estados básica do projeto



Fonte: Próprio Autor

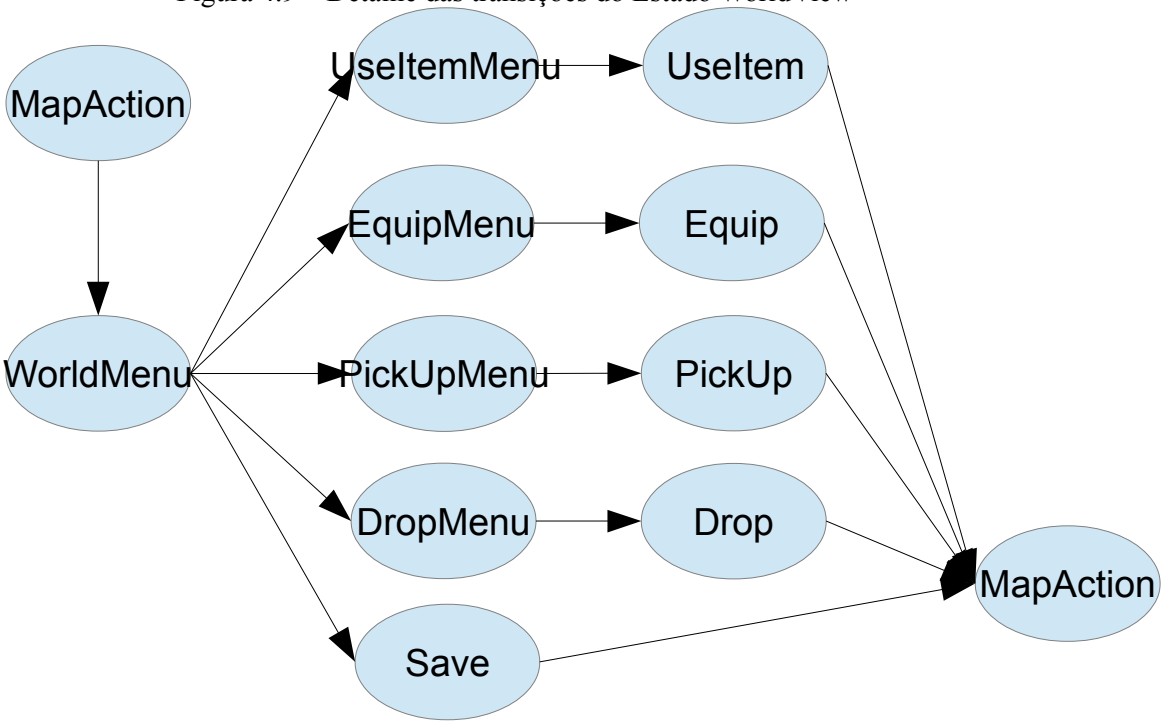
Na etapa de implementação ela assumiu a seguinte forma:

Figura 4.8 – Máquina de estados básica implementada ainda sem o roteiro



Fonte: Próprio Autor

Figura 4.9 – Detalhe das transições do Estado WorldView



Fonte: Próprio Autor

A presença do roteiro, embora não tenha alterado essencialmente a máquina básica do jogo, acrescentou verificações de transição de estados em certos eventos, como transições entre mapas, conversas com NPCs e eliminação de bosses.

Os estados da máquina foram declarados através da seguinte enumeração:

```
public enum FSMStatus { Start, StartMenu, StartGame, LoadGame, InitGame, TalkNarrator,
CheckQuestMap, CheckQuestCombat, Lore, TalkNPC, MapAction, MapTransition, CombatMenu,
Attack, Magic, Shop, WorldMenu, EquipMenu, Equip, DropItemMenu, DropItem, UseItemMenu,
UseItem, UseItemCombat, PickUpMenu, PickUp, Save }
```

As duas principais funções que gerenciam a máquina de estados ficaram bastante extensas e complexas, embora bastante autoexplicativas.

A primeira é a que verifica e resolve as transições segundo os seguintes mapeamentos:

Tabela 4.1 – Mapeamento parcial das transições de estado

Condição	Origem	Destino
	Start	StartMenu
Seleção de menu	StartMenu	StartGame
Seleção de menu	StartMenu	LoadGame
	StartGame	InitGame
	LoadGame	InitGame
	InitGame	TalkNarrator
Existe fala do narrador	TalkNarrator	Lore
	TalkNarrator	MapAction
Encerrada exibição de texto	Lore	MapAction
Inimigo próximo	MapAction	CombatMenu
NPC próximo e ação do jogador	MapAction	TalkNPC
Jogador em posição onde é transição	MapAction	MapTransition
Ativar Botão de Ação 2	MapAction	WorldMenu
Seleção de menu	CombatMenu	Attack
Seleção de menu	CombatMenu	Magic
	MapTransition	CheckQuestMap
	TalkNPC	Lore
	Attack	MapAction
Inimigo próximo	Attack	CombatMenu
	Magic	MapAction
Inimigo próximo	Magic	CombatMenu
	CheckQuestMap	MapAction

Fonte: Próprio Autor

Tabela 4.2 – Mapeamento das transições do estado WorldMenu

Condição	Origem	Destino
Seleção de menu	WorldMenu	UseItemMenu
Seleção de menu	WorldMenu	EquipMenu
Seleção de menu	WorldMenu	PickUpMenu
Seleção de menu	WorldMenu	DropItemMenu
Seleção de menu	WorldMenu	Save
Seleção de menu	UseItemMenu	UseItem
Seleção de menu	EquipMenu	Equip
Seleção de menu	PickUpMenu	PickUp
Seleção de menu	DropItemMenu	DropItem
Seleção de menu	Save	MapAction
	UseItem	MapAction
	Equip	MapAction
	PickUp	MapAction
	DropItem	MapAction

Fonte: Próprio Autor

A segunda, é a que opera as ações relativas a cada estado, são bastante específicas e estão descritas na tabela abaixo:

Tabela 4.3 – Mapeamento das ações executadas em cada estado

Estado	Ação
Start	
StartMenu	
StartGame	Inicializa jogador e primeiro mapa do jogo.
LoadGame	Carrega atributos do jogador, mapa e estado das quests.
InitGame	
TalkNarrator	Verifica atualização estado de quest por fala do narrador e carrega o texto no painel de leitura.
CheckQuestMap	Verifica atualização estado de quest por entrada em mapa.
CheckQuestCombat	Verifica atualização estado de quest por combate.
Lore	
TalkNPC	Verifica atualização estado de quest por fala de NPC e carrega o texto no painel de leitura.
MapAction	
MapTransition	Atualiza Tile Engine.
CombatMenu	
Attack	Ataca inimigo
Magic	Lança magia no inimigo
Shop	
WorldMenu	
EquipMenu	
Equip,	Equipa o jogador com o item selecionado no menu.
DropItemMenu	
DropItem	Remove o item selecionado do menu do inventário do jogador.
UseItemMenu	
UseItem	Utiliza o item selecionado do menu.
PickUpMenu	
PickUp	Adiciona o item selecionado no menu ao inventário do jogador.
Save	Salva estado no jogo.

Fonte: Próprio Autor

Por fim, o último aspecto afetado pela máquina de estados é o comportamento da interface gráfica, de acordo a seguinte tabela:

Tabela 4.4 – Mapeamento do comportamento da interface gráfica em cada estado

Estado	Seta	Menu Iniciar	Menu Combate	Menu Mundo	Menu Item	Tile Engine	Painel de Leitura	Jogador, Inimigos, NPCs, Bosses
Start	X							
StartMenu	X	X						
StartGame	X							
LoadGame	X							
InitGame	X							
CheckQuestLore	X							
Lore	X						X	
MapAction						X		X
CombatMenu	X		X					
MapTransition	X							
TalkNPC	X							
Attack	X							
Magic	X							
CheckQuestMap	X							
WorldMenu	X			X				
EquipMenu	X				X			
Equip,	X							
DropItemMenu	X				X			
DropItem	X							
UseItemMenu	X				X			
UseItem	X							
PickUpMenu	X				X			
PickUp	X							
Save	X							

Fonte: Próprio Autor

Dois aspectos, merecem atenção especial e explicação da solução encontrada. O primeiro deles é sobre a seleção das falas:

A função que seleciona as falas, tendo como parâmetro de entrada o número de identidade do NPC, faz uma varredura na lista de “quests”, restrita às não completas, e na lista de falas de cada, restrita às do NPC, aceitando qualquer fala cujo status da “quest” seja menor ou igual estado atual da “quest”, pressupondo a ordenação por status das falas. Desta forma garante-se que a fala selecionada seja sempre a mais condizente com o progresso do jogo.

A regra utiliza o campo “Viewed” da classe “QuestLore” para interromper a busca selecionando uma fala que não tenha sido ainda dita seja. Após, é feita uma verificação se a fala selecionada produz alguma modificação de estado de alguma “quest”, que é atualizada

neste caso. Caso não haja uma fala ainda não exibida, fica selecionada a fala repetida mais avançada.

O segundo é a descrição do parser para aquisição de dados externos. Do ponto de vista operacional do programa, o núcleo carrega dados do jogo através de arquivos externos. São estes arquivos os “bitmaps” onde estão armazenados os “sprites”, dos personagens e dos cenários; arquivos de texto descrição de mapas; arquivos de texto com narrações do roteiro da história e falas dos personagens, além de arquivos de salvamento que armazenam o estado do jogo.

O arquivo de texto que armazena a “quest” contém as informações organizadas no formato conforme uma gramática mais elaborada que a do arquivo de mapas. Em regra, ela utiliza comentários de linha, ou seja, a regra de desconsiderar o resto da linha após o caractere “#” e utiliza o caractere “:” para estabelecer o tipo de informação e espaços em branco para separar os parâmetros e aspas para unidades de texto.

A especificação, no caso, é a seguinte.

O primeiro tipo de informação é dado pelo tipo “ID”, que é o número de identidade da quest.

A segunda é “NAME”, uma string com o nome da quest.

A terceira é “LORE”, que contém as falas, e os parâmetros são, identidade do NPC, Status da “quest” quando ela é dita e o conteúdo das falas, este seguindo uma regra especial que separa as falas de acordo com o formato de paginação do elemento de interface painel de leitura, que divide o texto em parágrafos e páginas respectivamente com os caracteres especiais “@” e “$”.

As transições (“TRANSITION”) operam as mudanças de estados nas quests, e têm como parâmetros os o estado em que a “quest” transiciona, o evento, de uma lista (“NPC”, “MAP”, “BOSS”, “END”), que correspondem a conversar com um NPC, entrar em um mapa, eliminar um “boss” e o que indica o fim da “quest” e, por fim, a identidade do NPC, “boss” ou mapa.

E por último “NPCPOSITION”, “BOSSPOSITION” são bastante semelhantes, e têm como parâmetros a identidade, o status da “quest” em que ele ocupa naquele estado da “quest”, o mapa e as coordenadas cartesianas da posição.

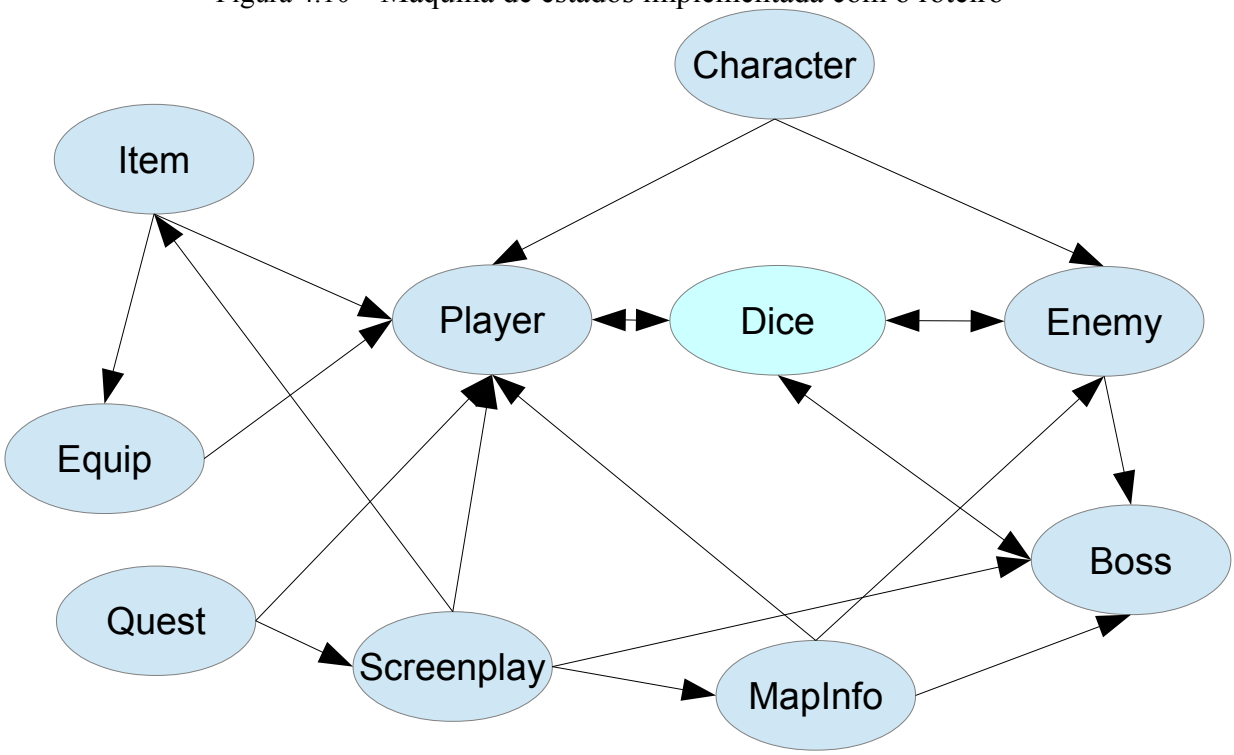
Um exemplo de conteúdo do arquivo, pode ser:

ID: 1
NAME: "Slay Bad Weasel"
LORE: 0 0 "Hi hero! Bad Weasel is making everyone's life miserable, please slay him!"

LORE: 0 2 "Thank you Bad Weasel is dead. Now we are happy."
 # Transitions
TRANSITION: 0 NPC 0
TRANSITION: 1 BOSS 2
TRANSITION: 2 NPC 2
TRANSITION: 3 END -1
 # Bosses Id, Status, Map, Position X, Position Y
BOSSPOSITION: 1 1 1 7 4

A máquina de estados final com a implementação do roteiro então ficou assim:

Figura 4.10 – Máquina de estados implementada com o roteiro



Fonte: Próprio Autor

Podemos exemplificar tomando o conteúdo de um arquivo de conteúdo de uma "quest"

No caso teríamos primeitamente os campos com a ID e o nome da "quest":

ID: 0 # (main quest)
NAME: "Pacify Lester's heart"

Após isso, temos as falas que serão exibidas pelo narrador e pelos personágens em função da "quest":

 # LORE: NPC Id, Status, Text "paragraph1" @" paragraph2" @ "paragraph3" $ "paragraph4" @ "paragraph5" # @ paragraph separator, $ page separator
LORE: -1 0 "Fala -1A"

LORE: 0 1 "Fala 0A"
LORE: 0 2 "Fala 0B"
LORE: 0 7 "Fala 0C"
LORE: 1 1 "Fala 1A"
LORE: 2 1 "Fala 2A"
LORE: 1 2 "Fala 1B"
LORE: 1 4 "Fala 1C"
LORE: 1 6 "Fala 1D"
LORE: 3 1 "Fala 3A"

Conforme temos estabelecido na gramática, primeiramente temos a ID do personagem, depois o status da "quest" em que a fala é dita e por último o conteúdo da fala. Utilzamos a ID -1 para a fala do narrador. Podemos assim perceber que o personagem de ID 1 tem quatro falas diferentes, para os estados 1, 2, 4, 6, respectivamente.

O ponto mais importante do conteúdo são as transições, as quais conforme a gramática indicam o estado em que a transição ocorre, o ente que desencadeia a transição e sua identidade:

 # Transitions: current status, thing, thing Id # thing: NPC, BOSS, MAP, ITEM
TRANSITION: 0 NPC -1
TRANSITION: 1 NPC 2
TRANSITION: 2 NPC 1
TRANSITION: 3 BOSS 1
TRANSITION: 4 NPC 1
TRANSITION: 5 BOSS 4
TRANSITION: 6 NPC 1
TRANSITION: 7 END -1

Temos assim, que a história desta "quest" seguirá os seguintes passos: Primeiramente o narrador dirá sua fala, após, para haver o progresso, será necessário conversar com o NPC 2, depois com o NPC 1, matar o BOSS 1, falar de novo com o NPC 1, Matar o BOSS 4, Falar com o NPC 1 e por fim, após isso, a "quest" estará completa.

Por fim, os valores de posições de NPCs e Bosses são bastante evidentes, indicando a Identidade, o estado da "quest" e a localização, dada pelo número do mapa e as coordenadas de posição.

 # NPC positions: ID, Status, Map, X, Y
NPCPOSITION: 0 0 0 17 11
NPCPOSITION: 1 0 0 10 5
NPCPOSITION: 1 6 1 8 4
NPCPOSITION: 2 0 2 2 5
NPCPOSITION: 3 0 0 5 10
NPCPOSITION: 3 2 1 3 4

NPCPOSITION: 3 4 2 4 1
NPCPOSITION: 3 6 3 4 1
 # Bosses Id, Status, Map, Position X, Position Y
BOSSPOSITION: 1 3 3 6 5
BOSSPOSITION: 4 5 2 4 2

4.2 Interface Gráfica

A etapa de desenvolvimento da interface gráfica se deu em dois ciclos. O primeiro envolvendo os elementos básicos de interface e o segundo, dos elementos do jogo. Foi utilizada para o desenvolvimento da interface gráfica o Framework Microsoft XNA 4.0.

O trabalho não tem por escopo servir de referência do framework XNA, porém uma breve e necessária explicação sobre como ele funciona pode se resumir a isso:

Além das inicializações, que carregam conteúdo e ajustam valores iniciais de variáveis, há duas funções principais que controlam o jogo cujos nomes são autoexplicativos e que são rodadas ciclicamente durante a execução: Update e Draw. São métodos sobrescritos da classe Game e têm as seguintes funções:

O primeiro, Update, é responsável pela atualização. Toda a lógica do jogo é resolvida por ele, cada incremento na posição de um personagem que está se movendo, cada teste de colisão, etc. é processado aqui.

A segunda, Draw, seleciona o que será exibido na tela, desde texto até as figuras, que pedaço da figura fonte será exibido em que posição destino na tela.

O framework dispõe ainda de uma classe chamada DrawableGameComponent que, por assim dizer, clona a clase jogo para desenho de figuras, oferecendo os métodos Update e Draw e assim possibilita desacoplar cada componente a ser desenhado na tela, tornando o código mais compartimentado, claro e organizado.

4.2.1 Primeiro Ciclo

No primeiro foram criados os elementos básicos de interação com o usuário: Seta e Painéis.

O primeiro componente é a seta (CygnusGameLibraryXNA/Interface/Arrow.cs), é trivial, apenas desenha a seta na tela de acordo com a posição do mouse. Tudo que ele faz é na função Update receber as coordenadas do Mouse e na função Draw desenhar a seta na tela.

O painel tem um componente básico e dois derivados, o Painel de Leitura e o Menu.

A base do painel (CygnusGameLibraryXNA/Interface/Panel.cs) tem como parâmetros

as dimensões e utiliza 8 figuras, os quatro cantos e as quatro barras: Horizontal, superior e inferior; vertical, idem. Há duas funções que são chamadas pelas classes derivadas, a que coloca o retângulo de fundo de acordo com a posição e as dimensões dadas, e a que coloca as figuras dos cantos e as barras laterais de acordo com as dimensões dadas:

```
public void DrawBackGround()
public void DrawBroders()
```

A classe do painel não lida com nenhuma lógica na função Update.

O Painel de Leitura (CygnusGameLibraryXNA/Interface/Lore.cs) implementa a lógica de exibir o texto gradativamente de forma animada. O componente tem dois estados, o primeiro, de exibição progressiva do texto, e o segundo, aguardando o usuário passar para a próxima página.

No primeiro estado foi utilizado um retângulo que cobre o texto e é diminuído progressivamente ao longo do tempo, linha por linha, na função Update.

O comportamento deste componente é afetado pelo uso do mouse, que, caso haja um clique sobre a área dele a cobertura do texto é removida e o componente passa a aguardar um novo clique para exibir a próxima página, estado em que fica quando a remoção da cobertura termina.

No segundo estado, para indicar que a exibição da página está encerrada um elemento fica piscando aguardado o clique do mouse para passar para a próxima página ou encerrar a exibição do texto, quando o componente desaparece.

As funções utilizadas foram:

```
private void UpdateMouseState()
private void UpdateBlink()
private void DrawText()
private void DrawCover()
private void DrawReadyMark()
```

O Menu (CygnusGameLibraryXNA/Interface/Menu.cs) exibe as opções em textos separados por linha, o comportamento indica a linha sobre a qual o mouse está sobre exibindo a cor de fundo com cores diferente. Quando uma opção é clicada o componente desaparece. Há uma propriedade que indica a escolha feita. Enquanto nenhuma é escolhida, o valor é menos um, passando para zero, um, dois, etc. De acordo com a posição escolhida.

As funções utilizadas foram:

```
private void UpdateMouseState()
private void DrawContent()
```

4.2.2 Segundo Ciclo

O segundo ciclo incluiu os componentes de animação associados ao sistema de posicionamento e o sistema de mapas. A animação dos personagens e o "Tile Engine" (CygnusGameLibraryXNA/World/TileEngine.cs) foram os componentes incluídos.

A implementação foi bastante fiel ao projeto, e envolveu a criação das seguintes classes:

– CygnusGameLibraryXNA/Character/PlayerXNA.cs

– CygnusGameLibraryXNA/Character/HumanXNA.cs

– CygnusGameLibraryXNA/Character/BossesXNA.cs

– CygnusGameLibraryXNA/Character/EnemiesXNA.cs

– CygnusGameLibraryXNA/Character/NPCsXNA.cs

Foram criadas duas variáveis que informam a posição do "Tile Engine" e a posição do personagem na tela. Utilizando como base as variáveis do núcleo do jogo que posicionam os personagens, uma de acordo com o sistema de mapas, e outra de acordo com a dimensão em pixels para definir a posição na tela dada pela fórmula assim expressa em código na regra do caso genérico aplicado aos personagens que não são o do jogador:

```
screenPosition.X = character.worldPosition.X - tileEnginePosition.X;
screenPosition.Y = character.worldPosition.Y - tileEnginePosition.Y;
```

No caso do jogador o código ficou assim:

```
screenPosition = SetScreenPosition();
if (tileEngine.BigMapWidth)
    IfOutOfCenterX();
else
    screenPosition.X = character.worldPosition.X;
if (tileEngine.BigMapHeigth)
    IfOutOfCenterY();
else
    screenPosition.Y = character.worldPosition.Y;
tileEngine.pivot = new Vector2(character.worldPosition.X,character.worldPosition.Y);
```

Esta função leva em conta a condição de o personagem servir de pivô para o tile engine no caso em que o mapa desenhado ocupa um espaço maior do que a tela, tratado com as funções IfOutOfCenter():

```
private void IfOutOfCenterX()
{
    if (character.worldPosition.X < TileEngine.CENTER_X * GameCore.TILESIZE)
        screenPosition.X = character.worldPosition.X;
    if (character.worldPosition.X > tileEngine.mapInfo.mapWidth * GameCore.TILESIZE +
        TileEngine.CENTER_X * GameCore.TILESIZE – TileEngine.SCREENWIDTH)
        screenPosition.X = character.worldPosition.X - (tileEngine.mapInfo.mapWidth *
        GameCore.TILESIZE - TileEngine.SCREENWIDTH);
}
```

Por último, a implementação do “Tile Engine” também foi bastante fiel ao projeto, a função de exibição do mapa na tela, com o duplo laço que lê as informações do mapa e projeta na tela a correspondente imagem da paleta, ficou assim:

```
int rb = RigthBound();
int bb = BottomBound();
for (int j = this.mapPosition.Y; j < bb; j++)
    for (int i = this.mapPosition.X; i < rb; i++)
    {
        Vector2 blockPosition = new Vector2(i * GameCore.TILESIZE – this.position.X,
            j * GameCore.TILESIZE - this.position.Y);
        Rectangle source = new Rectangle(Column(mapInfo.tiles[j][i]),
            Line(mapInfo.tiles[j][i]), GameCore.TILESIZE, GameCore.TILESIZE);
        spriteBatch.Draw(tileSet, blockPosition, source, Color.White);
    }
```

5 CONCLUSÃO

O processo de desenvolvimento, do projeto à implementação, foi interessante porque começou partindo de pontos isolados paralelos. Primeiro criando-se os elementos de interface e do núcleo com a lógica do jogo. O projeto foi importante para demonstrar a possibilidade de combinar níveis crescentes de complexidade utilizando uma extensa gama de aspectos práticos da Ciência da Computação, como estrutura de dados, algoritmos, e aspectos teóricos, como Teoria das Categorias e Linguagens Formais.

Feita esta primeira parte, a ligação dos elementos com o componente de labirinto até o limite de as colisões de “sprites” acionarem as rotinas de combate compôs um núcleo básico de jogo que por si mesmo já possibilita a criação de diversos tipos de jogos simples.

Do ponto de vista da implementação, o grande desafio foi implementar essas pontes de comunicação entre a máquina de estados básica do jogo e a do roteiro. Mas para isso, primeiro passo foi especificar os formatos dos arquivos de conteúdo relacionados ao roteiro do jogo. O que causou a necessidade de mais um ciclo de desenvolvimento, pois o código no ponto em que a máquina de estados básica do núcleo do jogo já estava pronto.

O grande desafio foi modificar a máquina de estados para armazenar as variações de conteúdo apresentado pela máquina de estados básica do jogo. A classe “Quest” armazena o conteúdo e a informação do estado em que os elementos são exibidos. À máquina de estados foram acrescentados novos estados relativos aos eventos em que é possível ocorrer a transição de estado. São eles, o combate, a conversação com um personagem não-jogador e a transição entre mapas.

Embora alguns detalhes, como o comércio e uso de alguns itens não tenham sido realizados pois tornariam o trabalho excessivamente extenso, a implementação ficou correta, inclusive viabilizando o salvamento do estado, o que foi até um objetivo que não havia sido estabelecido como necessário, mas que só foi possível fazer com a correta implementação da máquina de estados que integra o roteiro.

O que foi implementado até aqui pode evoluir no sentido de servir de base para uma versão mais avançada, pelo aprimoramento em termos de qualidade de apresentação de gráficos e de interface, pela criação de ferramentas de autoria de conteúdo, e por último, por suporte para múltiplos jogadores, o que implicaria na distribuição de tarefas para serem rodadas em conjunto e coordenação com outras aplicações de servidor e trazendo alguns aspectos a serem considerados como o uso de ferramentas de bancos de dados para armazenamento de jogadores.

6 BIBLIOGRAFIA

ROLE-PLAYING GAME. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2014. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Role-playing_game&oldid=39202841>. Acesso em: 23 jun. 2014.

GURPS. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2014. Disponível em: <http://pt.wikipedia.org/w/index.php?title=GURPS&oldid=38152668>. Acesso em: 23 jun. 2014.

ZECHNER, M.; GREEN R. Beginning Android 4 Games Development. Apress, 2011. 685 p.

RPG ELETRÔNICO. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2013. Disponível em: <http://pt.wikipedia.org/w/index.php?title=RPG_eletr%C3%B4nico&oldid=34912200>. Acesso em: 22 jun. 2014.

Dungeon Master (1986, MSX, ASCII) | Generation MSX. Disponível em: <http://www.generation-msx.nl/software/ascii/dungeon-master/787/>. Acesso em: 23 jun. 2014.

FINAL FANTASY IV. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2014. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Final_Fantasy_IV&oldid=39175216>. Acesso em: 22 jun. 2014.

Diablo - GameSpot. Disponível em: < http://www.gamespot.com/diablo/ >. Acesso em: 23 jun. 2014.

World of Warcraft. Disponível em: <http://us.battle.net/wow/pt/>. Acesso em: 23 jun. 2014.

MENEZES, Paulo Blauth. Linguagens Formais e Autômatos. 3 ed. Porto Alegre: Sagra Luzzatto, 2000. 165 p. (Livros Didáticos; 3).

MENEZES, P. B.; Haeusler E. H. Teoria das Categorias para Ciência da Computação. 2. ed. Porto Alegre: Bookman, 2008. 330 p. (Série Livros Didáticos, 12).