UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA (CIn)

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ELEGY: APLICANDO O PROCESSO DE FÁBRICA DE JOGOS AO DOMÍNIO DE ROLE-PLAYING GAMES (RPGs).

TRABALHO DE GRADUAÇÃO

Aluna: Leila Soriano de Souza Tenório de Azevedo (lssta at cin.ufpe.br)

Orientador: André Luís de Medeiros Santos (alms at cin.ufpe.br)

Co-orientador: André Wilson Brotto Furtado (awbf at cin.ufpe.br)

Recife, Pernambuco, Brasil

22 de Junho de 2009

UNIVERSIDADE FEDERAL DE PERNAMBUCO (UFPE)

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CENTRO DE INFORMÁTICA (CIn)

Leila Soriano de Souza Tenório de Azevedo

ELEGY: Aplicando o processo de Fábrica de Jogos ao domínio de Role-Playing Games (RPGs).

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: André Luís de Medeiros Santos

Co-orientador: André Wilson Brotto Furtado

Recife, 22 de Junho de 2009.

"As dificuldades são como as montanhas. Elas só se aplainam quando avançamos sobre elas."

Provérbio Japonês

"Jamais se desespere em meio às sombrias aflições de sua vida, pois das nuvens mais negras cai água límpida e fecunda."

Provérbio Chinês

Agradecimentos

Aos meus pais, pela vida e por prezarem sempre pela minha educação.

À minha irmã, com quem pude aprender a importância de saber dividir e compartilhar.

Aos amigos que fiz no Centro de Informática, por todos os momentos dentro e fora da graduação, em especial meus queridos Peter, Mephisto, Chimpa, Doido, Preto, Marlus, Padovan e Klaus (sem seu DVD do VS eu estaria perdida!). Sem vocês essa caminhada não teria sido tão recompensadora. Vocês estarão para sempre em minha memória e em meu coração!

Ao meu orientador André Santos sensei, pela paciência e disposição em esclarecer minhas dúvidas e contribuir para o bom andamento deste trabalho, e por ser um exemplo de profissional, um verdadeiro mestre.

A André Furtado, por toda a ajuda e paciência. Sem seu auxílio não sei o que seria de mim no desenvolvimento deste projeto! Muito obrigada e desculpe por te perturbar todos os dias com milhões de e-mails, que você pacientemente respondeu!

A André Marques por me ensinar a apreciar a vida, dentre tantas outras coisas incríveis.

Aos que eu não citei diretamente, mas contribuíram para eu ser o que sou hoje.

A Deus por ter permitido encontrar tantas pessoas incríveis, com as quais eu pude aprender o valor da amizade e do amor. Pela força para continuar sempre. Por ter passado por bons e maus momentos, e ter aprendido tantas coisas maravilhosas. Nada como um dia após o outro...

Arigatoo gozaimasu!

Leila-san 桜.

“Ku wa raku no tane.”

“Seishin itoo nani gotoka narazan.”

Assinaturas

Este Trabalho de Graduação é resultado dos esforços da aluna Leila Soriano de Souza T. de Azevedo, orientada pelo professor André Luís de Medeiros Santos, com o título "*ELEGY: Aplicando o processo de Fábrica de Jogos ao domínio de Role-Playing Games (RPGs)*". Todos abaixo estão de acordo com o conteúdo deste documento e os resultados deste Trabalho de Graduação

Orientador

_____

Professor André Luís de Medeiros Santos.

Avaliador

_____

Professor Géber Lisboa Ramalho.

Aluna

_____

Leila Soriano de Souza Tenório de Azevedo.

Resumo

A indústria de jogos eletrônicos tornou-se uma das mais expressivas no ramo do entretenimento. Porém a produção de *softwares* em geral tem enfrentado o desafio de atender a alta demanda do mercado. Para isso, uma das soluções é transformar a produção quase artesanal dos jogos numa espécie de manufatura, onde componentes são reusados, aumentado a produtividade.

Este trabalho apresenta como solução uma fábrica de jogos, com domínio alvo nos jogos *Role-Playing Games*. O processo utilizado para fomentação deste projeto é a combinação da análise de domínio e desenvolvimento de domínio específico, que introduz o conceito de *Domain-Specific Languages*.

Durante a análise de domínio, será utilizada a metodologia SharpLudus, onde serão mostrados conceitos como análise de funcionalidades. A fábrica será composta por uma linguagem visual, que irá gerar código para ser consumido por uma *engine.*

A fábrica, denominada *Elegy*, tem como objetivo tornar a criação de jogos mais prática, segura, diminuindo o tempo de desenvolvimento e assim aumentando o custo/benefício.

Palavras-chave: Jogos, RPG, Análise de Domínio, *Feature Model*, Fábricas de Jogos, DSL.

Abstract

The industry of electronic games became one of the most expressive in the entertainment's branch. However the yield of software in general faces a challenge to meet the market high demands. An option to solve it is to turn the almost craftsmanship production in a kind of manufacture, where components are reused, improving the productivity.

This work shows as solution a game factory, focused in the Role-Playing Games domain. The process used to promote this project is the combination of domain analysis and domain specific development, which introduces the Domain-Specific Languages concept.

During the domain analysis, will be used a methodology called SharpLudus, where concepts like feature analysis will be shown. The factory will be composed by a visual language, which will generate code to be consumed by an engine.

The game factory, called Elegy, has the objective to make the game production more practice, secure, decreasing the development time thus improving the cost/benefit.

Keywords: Games, RPG, Domain Analysis, Feature Model, Games Factories, DSL.

Sumário

Índice de Figuras

Índice de Tabelas

1. Introdução

Jogar é uma atividade tão antiga quanto à própria cultura. A origem exata é desconhecida, mas é estimado que existia em culturas como a Suméria (5000 A.C.). O ato de jogar implica essencialmente em atividades intelectuais, ou uma mistura entre atividades intelectuais e físicas, e interação entre pessoas e/ou diferentes tipos de objetos, como cartas e bolas (Furtado, 2006).

Os primeiros jogos eletrônicos surgiram com o advento da computação, e datam da década de 50 do século XX. Um importante marco é o Pong, criado em 1951 pelo engenheiro Ralph Baer, considerado por muitos o criador do conceito videogame (Pong Story, 2008). O jogo Pong atravessou gerações, com inúmeras versões lançadas.

Com o passar dos anos, os jogos digitais tornaram-se cada vez mais complexos e interdisciplinares, trazendo conceitos de diferentes áreas da Computação, como Computação Gráfica, Inteligência Artificial, Computação Musical, entre outras. Domínios não ligados a Computação também estão presentes, como Psicologia, Artes, Educação, Estratégia (Furtado, 2006).

A indústria de jogos eletrônicos tornou-se uma das mais expressivas no ramo do entretenimento, alcançando cerca de 7.4 bilhões de dólares em vendas no ano de 2006 (ESA, 2007). Essa gigante da economia mundial equipara-se a indústria cinematográfica, e, de acordo com estudos recentes, se gasta mais em jogos eletrônicos do que em produtos de entretenimento musical (Furtado, 2006).

Analisando-se os números da indústria de jogos digitais, um estilo que possui uma das vendas mais expressiva é o *Role- Playing Game*, conhecido por sua sigla RPG. Em 2006, focando-se em jogos para computador, o RPG galgou o 2º lugar entre os mais vendidos por unidade, com 13.9%, perdendo para jogos de estratégia, com 35.4%. Já em jogos para console, encontra-se em 5º lugar, com 9.3% (ESA, 2007).

Os *Role-Playing Games* digitais originaram-se de jogos de mesmo nome, onde os jogadores interpretam seus papéis e utilizam fichas de papel com todos os atributos de seu personagem. Um dos mais famosos RPGs de mesa é o *Dungeons and Dragons*, mais conhecido como *D&D*, que recentemente lançou a sua 4ª edição pela editora *Wizards* (Wizards, 2009).

Nesta versão, chamada de RPG de “mesa”, existe um jogador mestre que narra as aventuras e dita as regras do jogo. Os demais jogadores escolhem personagens e interpretam diversas situações, utilizando a sorte nos dados para decisões em batalhas, uso de habilidades como percepção, diplomacia, blefe, entre outras no decorrer da campanha. Esse tipo de jogo traz uma grande complexidade no que se trata da narrativa da história, que pode tomar diversos caminhos a depender das ações de cada jogador.

O RPG eletrônico herdou, dentre muitas outras características, a complexidade em relação à história do jogo. Isto impacta no que cada diferente jogador irá viver dentro do mesmo jogo, dependendo do caminho escolhido. Alguns jogos possuem finais diferenciados de acordo com as ações tomadas no desenrolar da história, o que torna extremamente interessante a experiência de jogar assim como de criar um RPG. Jogos como *Final Fantasy*, *Chrono Trigger, Secret of Mana,* dentre outros, foram imortalizados dentre os RPGs eletrônicos.

A indústria de jogos digitais, assim como a de softwares em geral, tem sofrido constantes mudanças para atender a alta demanda, adotando métodos que aumentem eficientemente a produtividade (Greenfield & Short, 2004). Este projeto propõe o uso do paradigma de Fábrica de Jogos, que será denominada Elegy, como alternativa para criação de jogos RPG baseada no reuso para aumento de produtividade.

1.1. Motivação

A indústria de software, ainda hoje, funciona de forma artesanal, dependente das habilidades individuais para realização intensiva de trabalho manual. Assim como ocorreu com outras indústrias, a de software necessita de métodos que automatizem a produção, de forma a reduzir custos e tempo, aumentando a produtividade dos desenvolvedores (Greenfield & Short, 2003).

Na fase em que se encontra atualmente, a produção de software não tem atendido de forma eficiente à demanda do mercado. Por exemplo, os Estados Unidos gastam anualmente mais de 250 bilhões de dólares no desenvolvimento de software, com custo por projeto entre 430 e 2.300 dólares (Greenfield & Short, 2003). Desses projetos 16% conseguem ser finalizados dentro do orçamento e tempo estimados, enquanto 31% são cancelados,

criando perdas de 81 bilhões de dólares anuais. Outros 53% excedem orçamento, criando perdas de aproximadamente 59 bilhões de dólares.

Neste cenário, surgem opções para automatizar a produção de software. Uma abordagem interessante é a implementação de Fábricas de Software, que se baseia na produção de softwares de uma mesma família, através do uso de linguagens, frameworks e ferramentas (Greenfield & Short, 2003). Neste paradigma são montadas verdadeiras linhas de produção de software, focadas em soluções para problemas de uma mesma espécie.

No âmbito da produção de jogos muito já foi feito no que concerne à automatização. O uso de *game engines* facilitou bastante o desenvolvimento de jogos, trazendo benefícios da Engenharia de Software e orientação a objetos (Furtado & Santos). Porém ainda é necessária uma maior abstração para os desenvolvedores, diminuindo a complexidade, e desta forma aumentando a produtividade.

Desta forma a produção de Fábricas de Jogos, lançado-se mão de ferramentas visuais, é uma alternativa eminente para a automatização do desenvolvimento de jogos.

1.2. Objetivos

O objetivo principal deste projeto é a implementação da *Elegy*, uma fábrica de jogos RPG, que virá a integrar o projeto já existente SharpLudus (Furtado, SharpLudus).

A contribuição primária deste projeto é lançar uma alternativa para o aumento da produtividade no desenvolvimento de jogos RPG. Para isso será utilizada a modelagem de domínio específico (DSM), para a descrição da especificação da fábrica, e o desenvolvimento de uma linguagem de domínio especifico (DSL).

Será reusada uma *engine* específica para jogos RPG, o *Role-Playing Game Starter Kit* (Role-Playing Game Starter Kit), para criação de jogos no XNA Game Studio. Portanto os jogos produzidos pela *Elegy* serão portáveis para o *console* XBox.

1.3. Desafios

O maior desafio deste projeto encontra-se na complexidade do estilo RPG. A diversidade desses jogos é tanta que, por mais completo que um projeto desse tipo possa parecer, sempre existirão características que foram deixadas de lado. Mesmo assim, não deixa de ser uma experiência enriquecedora e com potencial para ser utilizada após sua conclusão.

Outro desafio é o reuso do Role-Playing Game Starter Kit, que exigirá um amplo estudo do que já foi implementado e possíveis adaptações ao projeto atual. Todo esse processo de reuso, assim como dificuldades encontradas e até mesmo inviabilidades serão apresentadas neste documento.

1.4. Organização do Documento

Este documento está organizado da seguinte forma: o capítulo 2 aborda conceitos sobre Fábricas de Software e desenvolvimento de domínio específico, apresentando a definição de *Domain-Specific Languages.* O capítulo 3 define o processo de Análise de Domínio, e traz o detalhamento da metodologia SharpLudus (Furtado, 2006) , adotada para este projeto.

O capítulo 4 mostra a aplicação da metodologia para análise do domínio escolhido, a dos *Role-Playing Games*. Em seguida, o capítulo 5 traz o detalhamento da modelagem e implementação da ferramenta e o capítulo 6 mostra um estudo de caso para validação do projeto.

Por fim, o capítulo 7 traz as considerações finais em relação aos resultados obtidos e projetos futuros.

2. Fábrica de Software e Desenvolvimento de Domínio Específico

A necessidade de redução de custos e tempo diante da crescente demanda por sistemas de *software* impulsionou a criação de métodos automatizados de modo a tornar os desenvolvedores mais produtivos. O conceito de Fábricas de *Software* promete maiores ganhos no desenvolvimento de sistemas na medida em que propõe reuso a fim de obter automatização. Desta forma o desenvolvimento de *software* passa a ser baseado numa espécie de manufatura, diferente do sistema quase artesanal que ainda é praticado.

Muitos sugerem que o desenvolvimento de software não pode ser industrializado devido ao seu caráter criativo (Greenfield & Short, 2003). Porém um progresso significante pode ser feito no fundamento da orientação a objetos, principalmente à luz do crescimento de metodologias ágeis de desenvolvimento. Considerando que o desenvolvimento de *software* é uma disciplina inerentemente orientada a pessoas e não pode ser reduzida puramente a processos mecânicos e determinísticos, uma abordagem para industrialização com uso de vocabulário próximo ao domínio do problema, e que delega a maior parte dos aspectos mecânicos e determinísticos a ferramentas de desenvolvimento é mais adequada.

Ao longo dos últimos anos, a indústria de *software* tem encontrado demandas de uma sociedade cada vez mais automatizada, baseada no aperfeiçoamento das habilidades individuais dos desenvolvedores, como os artesãos encontraram a demanda de uma sociedade cada vez mais industrializada nos primeiros estágios da revolução industrial, baseada no aperfeiçoamento das habilidades individuais dos artesãos. No entanto, os meios de produção estão sendo rapidamente sobrecarregados, uma vez que a capacidade de uma indústria artesanal é limitada em métodos e ferramentas, além da quantidade de trabalhadores qualificados.

Perante desafios similares, outras indústrias evoluíram do artesanato para a industrialização através da customização e montagem de componentes padronizados voltados para produção de bens similares ou distintos. Essa evolução foi possível através da padronização, integração e automatização dos

processos de produção, do desenvolvimento de ferramentas extensíveis que podem ser configuradas para automatizar tarefas repetitivas, do desenvolvimento de linhas de produção para redução de custos e tempo e uma cadeia de fornecedores altamente especializados e independentes para distribuir custos e riscos. Essas mudanças possibilitaram uma produção mais eficiente e uma ampla variedade de produtos satisfazendo a crescente demanda da clientela.

2.1. Fábricas de Software

Uma família de produtos provê um contexto no qual muitos problemas comuns aos membros da família podem ser resolvidos coletivamente. Construída no conceito de linhas de produção de *software,* fábricas de *software* exploram este contexto para prover uma ampla família de soluções, enquanto gerenciam variações entre membros da família (Greenfield & Short, 2003).

Uma fábrica de *software* sistematicamente captura conhecimentos de como produzir os membros de uma família específica de produtos, fazendo isto disponível na forma de *assets* (bens), como padrões, frameworks, modelos e ferramentas, e então sistematicamente aplica estes *assets* para automatizar o desenvolvimento de membros da família, reduzindo custos e tempo para o mercado, e aumentando a qualidade dos produtos desenvolvidos.

Observando como a automação é feita, podemos encontrar um padrão recorrente (Greenfield & Short, 2004):

- Após desenvolver um número de sistemas num dado domínio de problemas, podemos identificar um conjunto de abstrações reusáveis para este domínio, e então documentar um conjunto de padrões para usar estas abstrações.
- Ao desenvolver um framework para codificar abstrações e padrões. Isto permite construir sistemas no domínio instanciando, adaptando, configurando e montando componentes.
- Ao definir uma linguagem para o domínio e construir ferramentas que suportem a linguagem, como editores, compiladores e *debbugers*, para automatizar o processo de montagem. Isto ajuda a responder

rapidamente às mudanças de requisitos, já que parte da implementação é gerada, e pode ser facilmente modificada.

Uma fábrica de *software* é uma linha de produção que configura ferramentas de desenvolvimento extensíveis. Uma fábrica de software contém três idéias chaves: um *schema* da fábrica, um *template* e um ambiente de desenvolvimento. Abaixo uma analogia muito interessante proposta por Greenfield (Greenfield & Short, 2004) é usada para explicar cada uma dessas idéias chave.

- O *schema* da fábrica de software é como uma receita. A lista de ingredientes, como projetos, diretórios de código fonte, arquivos de SQL e arquivos de configuração, e uma explicação de como eles devem ser combinados para criar o produto. Especifica como DSLs devem ser usadas e descreve como modelos baseados nessas DSLs podem ser transformadas em código ou outros artefatos, ou em outros modelos. Descreve a arquitetura de linha de produção, e as relações chaves entre componentes e frameworks que compõem a fábrica.
- O *template* da fábrica de software é como uma bolsa de mercearia contendo os ingredientes listados na receita. Provê padrões, orientação, *templates*, frameworks, amostras, ferramentas customizadas, como um editor visual de DSL, *script*s, XSDs, folhas de estilo, entre outros elementos usados para construir o produto.
- Um ambiente de desenvolvimento extensível como o VSTS é como uma cozinha onde a refeição é preparada. Quando configurado com o *template* da fábrica de software, o VSTS torna-se uma fábrica de software para família de produtos.

Os produtos servidos são como refeições servidas num restaurante. Os *stakeholders* são como clientes que pedem refeições do menu. Uma especificação é como um pedido específico de refeição. Os desenvolvedores do produto são como cozinheiros que preparam as refeições descritas nos pedidos, e quem podem modificar receitas, ou prepararem refeições que não estão no menu. Os desenvolvedores da linha de produção são como *chefs* que decidem o que aparecerá no menu, e quais ingredientes, processos, e equipamentos de cozinha usados para prepará-los.

Um esquemático de uma fábrica de software está apresentado na Figura 1. Construir uma fábrica de *software* é um caso especial de linha de produção de *software* (Furtado, 2006). A criação do *schema* da fábrica é uma especialização de duas atividades de linha de produção: análise de linha de produção, que define quais produtos a fábrica deverá desenvolver, e *design* de linha de produção, que define como a fábrica irá desenvolver produtos neste escopo. A criação de um *template* de fábrica de *software,* em outra mão, é a especialização da atividade de implementação de linha de produção, que supre e empacota *assets* para a fábrica. Coletando *feedback* como membros da família desenvolvidos, a fábrica pode ter sua definição e implementação refinada.

Construir um produto usando fábricas de *software* é um caso especial de construção através de linhas de produção. Isto inclui atividades de desenvolvimento comuns para o cenário onde a fábrica de *software* não é usada, como análise de problemas, especificação de produtos, *design* de produtos, implementação de produtos, teste de produtos e implantação de produto. No entanto, o uso de fábricas de *software* garante que o trabalho é feito de uma forma metódica, sob o controle de um processo de enquadramento, bem como reduz a quantidade de trabalho a ser feito através do reuso de *assets* de produção existentes. Os *assets* de produção (requisitos, modelos, configuração de implantação, etc.) são especificados, reusados, gerenciados, e organizados por múltiplos pontos de vista, como definido pelo *schema* da fábrica. O *schema* e o *template* da fábrica podem ser configurados durante o desenvolvimento do produto. Finalmente, o desenvolvimento do produto é organizado pelo processo de desenvolvimento que foi definido quando a fábrica foi gerada, e então customizado durante a configuração do *schema* da fábrica para o produto sob desenvolvimento.

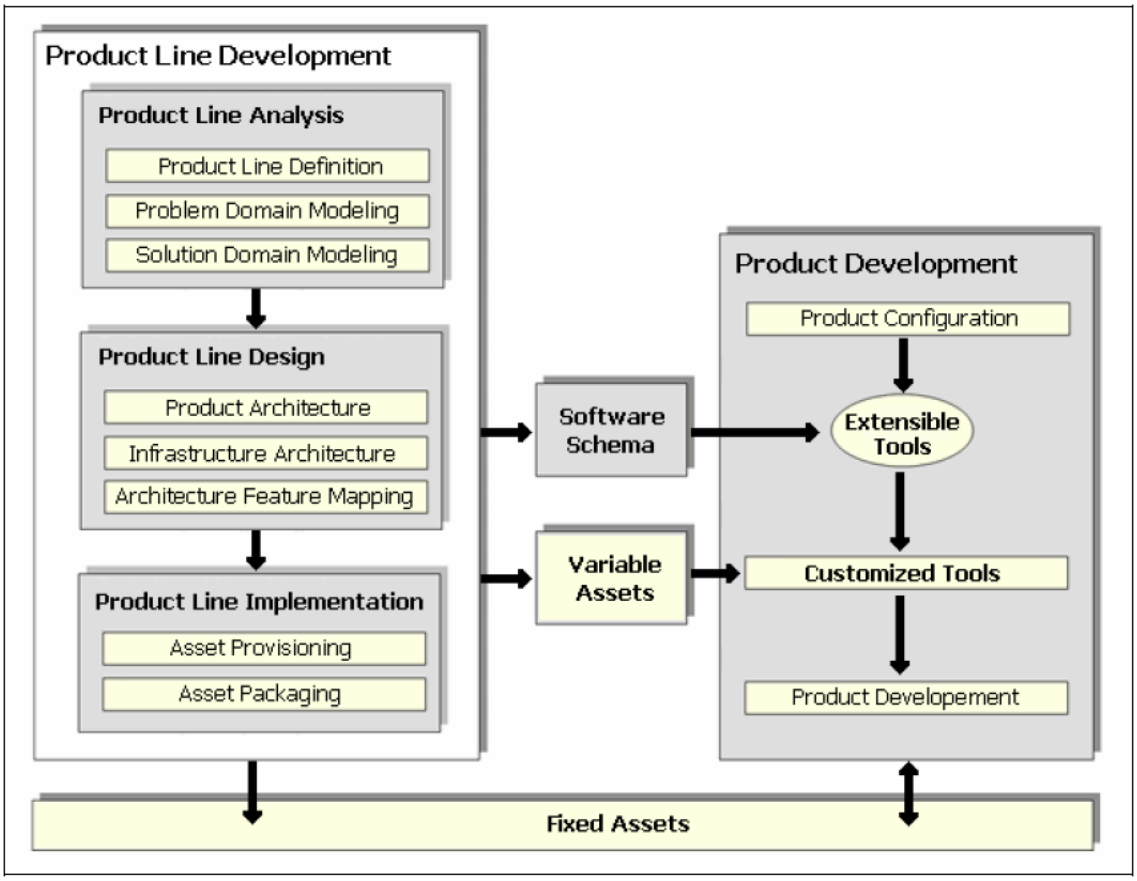


Figura 1 – Uma fábrica de software.

2.2. Desenvolvimento de Domínio Específico

O Desenvolvimento de domínio específico é uma abordagem para resolver problemas que pode ser aplicada quando um problema particular ocorre repetidamente. Baseia-se no conceito de Linguagens de Domínio Específico (*Domain-Specific LanguagesI – DSLs*), forma limitada de linguagens computacionais projetadas para uma classe específica de problemas que oferecem, através de notações e abstrações apropriadas, expressivo poder de foco, e usualmente restrito, a um problema do domínio (Furtado).

Alinhado à iniciativa de fábricas de *software*, o conceito de Modelagem de Domínio Específico (*Domain-Specific Modeling*) se sobressai. Este é construído sobre DSLs gráficas ou visuais e baseia-se em modelos, que podem ser processados por ferramentas, para prover conjuntos de abstrações para satisfazer as necessidades de uma família específica de sistemas. Eleva

especificações do programa para notações visuais de domínio específico compactas, que são fáceis de escrever e manter, aumentando o nível de abstração do desenvolvimento do sistema além da programação para especificação da solução diretamente usando conceitos do domínio.

2.2.1. Domain-Specific Languages

Em todos os ramos da ciência e engenharia existe uma distinção entre abordagens que são *genéricas* e as que são *específicas* (Deursen, Klint, & Visser). Uma abordagem genérica provê uma solução para vários problemas numa certa área, mas essa solução pode ser sub-ótima. Uma abordagem específica provê uma solução muito melhor para um pequeno conjunto de problemas. Uma das encarnações dessa dicotomia na ciência da computação é: linguagens de domínio específico versus linguagens de programação genéricas.

Esse tópico não é novo. Linguagens de programação antigas (Cobol, Fortran, Lisp) surgiram como linguagens dedicadas para solucionar problemas numa certa área (respectivamente processamento de negócios, computação numérica e processamento simbólico). Gradualmente elas evoluíram para linguagens de propósito geral e cada vez mais a necessidade por mais linguagens especializadas para resolver problemas em domínios de aplicações bem definidos volta a surgir. Ao longo do tempo, as seguintes soluções têm sido experimentadas:

- *Bibliotecas de sub-rotinas* contêm sub-rotinas que realizam tarefas relacionadas num domínio bem definido como, por exemplo, equações diferenciais, gráficos, *interfaces* de usuário e banco de dados. A biblioteca de sub-rotinas é o método clássico para empacotar conhecimento de domínio reusavel.
- *Frameworks orientados a objetos e Component Framework* continuam a idéia da biblioteca de sub-rotinas. Bibliotecas clássicas têm uma estrutura plana, e a aplicação invoca a biblioteca. Em *frameworks* orientados a objetos é freqüente o caso em que o *framework* está no controle, e invoca os métodos providos pelo código da aplicação específica.

- *Uma linguagem de domínio específico* (DSL) é uma linguagem pequena, usualmente declarativa, que oferece expressivo poder focado num problema de um domínio particular. Em alguns casos, programas DSL são traduzidos para chamadas para uma biblioteca de sub-rotinas comum, e a DSL pode ser vista como um meio de esconder detalhes da biblioteca.

A característica chave da DSL é seu poder expressivo de foco. DSLs são usualmente pequenas, oferecendo apenas um conjunto restrito de notações e abstrações. Linguagens de domínio específico são geralmente declarativas. Conseqüentemente, elas podem ser vistas como especificações de linguagens, bem como linguagens de programação. Exemplos comuns de DSL são SQL, HTML, TeX e BNF.

Adotar uma abordagem de DSL para engenharia de *software* envolve tanto riscos quanto oportunidades. DSLs bem projetadas trabalham para encontrar um balanço apropriado entre as duas conseqüências. Os benefícios das DSLs incluem:

- DSLs permitem soluções que são expressas no idioma e no nível de abstração do domínio do problema. Conseqüentemente, especialistas no domínio podem entender, validar, modificar, e muitas vezes desenvolver programas DSL.
- Programas DSL são concisos, auto-documentados em grande extensão, e podem ser reusados para diferentes propósitos.
- DSLs aumentam a produtividade, segurança, manutenabilidade e portabilidade.
- DSLs incorporam conhecimentos do domínio, e então permitindo a conservação e reuso deste conhecimento.
- DSLs permitem validação e otimização no nível do domínio.
- DSLs aumentam a testabilidade.

As desvantagens no uso das DSLs são:

- O custo de projeto, implementação e manutenção de DSLs.
- O custo de treinamento para usuários de DSLs.
- A disponibilidade limitada de DSLs.
- A dificuldade de encontrar um escopo apropriado para uma DSLs.

- A dificuldade no balanceamento entre linguagens de domínio específico e linguagens de propósito geral.
- O potencial de eficiência perdido quando comparado com códigos feitos à mão.

No próximo capítulo será introduzida a noção de Análise de Domínio (*Domain Analysis)* e os métodos utilizados para tal.

3. Análise de Domínio: Estado da Arte

Por quase três décadas, a área de Análise de Domínio tem servido outras iniciativas da Engenharia de Software, como Desenvolvimento baseado em Componentes, Linhas de Produção de Software e Reuso de Software em geral (Furtado, 2006). Segundo Prieto-Diaz (Prieto-Diaz, 1990), é definida como um processo pelo qual a informação utilizada no desenvolvimento de sistemas de software é identificada, capturada e organizada com o propósito de fazê-la reusável na criação de novos sistemas.

O termo Análise de Domínio foi inicialmente introduzido por Neighbors (Neighbors, 1980) como “a atividade de identificar os objetos e operações de uma classe de sistemas similares num domínio particular de problemas”. Ainda segundo ele, “a chave para softwares reusáveis é capturada na análise de domínio na medida em que se salienta a reusabilidade da análise e *design*, não código”.

Durante o desenvolvimento de software, diversos tipos de informação são gerados, de análise de requisitos ao código fonte. O código fonte está no último nível de abstração e é considerada a representação mais detalhada do sistema. Informações complementares são também geradas durante o desenvolvimento do software, como documentação de código, histórico de decisões, planos de teste e manual do usuário, que são essenciais para um entendimento total do sistema (Prieto-Diaz, 1990).

Um dos objetivos da análise de domínio é fazer com que toda a informação seja facilmente acessível. Ao se fazer a decisão de reuso o engenheiro de software deve entender o contexto que levou o componente reusável a ser construído do modo que ele é. Com essas informações disponíveis, o reuso é feito de maneira mais efetiva.

Uma drástica melhoria no processo de reuso resulta quando, através de análise de domínio em arquiteturas comuns, obtemos modelos genéricos ou linguagens especializadas que alavancam o processo de desenvolvimento de software numa área específica. Para obter essas arquiteturas e linguagens devem-se identificar características comuns ao domínio das aplicações, selecionando e abstraindo os objetos e operações que as caracterizam e criando procedimentos para automatizar estas operações.

Nem Neighbors nem Prieto-Diaz, dentre outros autores, resolveram o problema de como fazer a análise de domínio. Então é necessário desenvolver abordagens para análise de domínio, a fim de explorar verdadeiramente o reuso em ambientes industriais.

Uma abordagem que pode ser utilizada na criação de fábricas de jogos, foco deste projeto, é a proposta por Furtado, que engloba as atividades de definição de características sob análise, seleção de amostras da linha de produção, execução da análise e, finalmente, a definição das características comuns e variabilidades.

A análise de escopo, características comuns e variabilidades, conhecida como SCV (*Scope, Commonality and Variability*), dá aos engenheiros de software uma maneira sistemática de pensar e identificar a família de produtos que estão criando (Coplien, Hoffman, & Weiss, 1998). Entre outras coisas auxilia os desenvolvedores a criar um projeto que contribua para o reuso e facilidade de mudanças, predizer como um projeto pode falhar ou ser bem-sucedido e identificar oportunidades para automatizar a criação de produtos membros de uma família.

Abordagens como *Family-Oriented Abstraction, Specification and Translation* (FAST) e *Feature-Oriented Domain Analysis* (FODA), utilizam a análise SCV como base. Neste projeto, será utilizada a metodologia desenvolvida por Furtado (Furtado, 2006), citada anteriormente, que faz uso de alguns conceitos do FODA. Nas próximas sessões será dada uma descrição da metodologia proposta por Furtado.

3.1. Processo SharpLudus

A metodologia desenvolvida por Furtado (Furtado), denominada SharpLudus, propõe uma abordagem pragmática de análise de domínio para a criação de fábricas de jogos, composta pelas seguintes atividades: estabelecimento da visão e escopo, definição de funcionalidades sob análise, a seleção de amostras de linha de produção para serem analisadas, execução da análise e, finalmente, a identificação das semelhanças e variabilidades com o auxílio de uma abordagem mais sólida como a *featuring model.* A intenção final é reunir um conjunto de abstrações que compreensivelmente represente jogos pertencentes ao domínio escolhido.

3.1.1. Estabelecimento de visão e escopo

A primeira tarefa para criar fábricas de jogos digitais é chegar a um acordo numa visão de alto nível e entendimento comum do domínio a ser abordado. Uma vez que o domínio pode ser novo ou apenas parcialmente entendido pelos projetistas, isto não é necessário para chegar a uma especificação completa do domínio neste ponto. De fato, este é um processo interativo: como as amostras da linha de produção são estudadas e catalogadas, o domínio é mais bem entendido, o escopo pode ser refinado, e, como conseqüência, novas DSLs podem surgir para capturar semelhanças e variabilidades no domínio.

O processo de definição de escopo em fábricas de jogos é iterativo e incremental, devendo ser baseado nas funcionalidades do domínio. O escopo final da fábrica de jogos é a soma dos escopos de cada funcionalidade juntamente com alguma decisão técnica, como a plataforma alvo.

As DSLs irão surgir de funcionalidades específicas do domínio, não simplesmente do próprio domínio da fábrica. Visto que um dos maiores benefícios de uma DSL é seu foco expressivo, não é uma boa abordagem ter um mapeamento um a um entre o domínio da fábrica e uma única DSL. Domínios de jogos são quebrados em subdomínios, e as funcionalidades variáveis dos produtos pertencentes a esses subdomínios serão *ultimate drivers* para o projeto de DSL.

3.1.2. Funcionalidades do domínio sob Análise

A cada iteração, o grupo de funcionalidades do domínio a serem analisadas deve ser definido, ou refinado. Quando o escopo base é definido, uma série de funcionalidades é identificada e pode ser usada como ponto de partida. No entanto, a natureza iterativa do processo e um conseqüente entendimento profundo do domínio podem adicionar, remover, mesclar, dividir ou modificar o conjunto de funcionalidades.

3.1.3. Seleção de Amostras de Linha de Produção

Esta tarefa trata de selecionar produtos existentes, em desenvolvimento ou ainda não desenvolvidos para serem analisados. Esses produtos

provavelmente não foram criados como parte de uma abordagem de linha de produção. Pelo contrário, eles devem ter sido desenvolvidos através de uma abordagem de desenvolvimento "*one-off*" com pouco ou nenhum reuso.

O número de jogos a serem analisados é definido principalmente pelos recursos da fábrica e cronograma. Desse modo, uma preocupação importante é selecionar aqueles jogos que são mais representativos. Por exemplo, um jogo que foi relançado através de *remakes* recebeu amplo reconhecimento da indústria e da mídia, podendo ser considerado uma amostra representativa.

Às vezes, um jogo pode possuir várias versões de títulos similares. Neste caso, projetistas da fábrica devem optar por analisar todos eles como um único grupo, considerando as funcionalidades como variações e extensões do jogo original. Se essas versões do jogo têm peculiaridades expressivas, pode ser uma indicação que o domínio da fábrica pode ser particionado em subdomínios.

Uma questão importante que surge durante a seleção de amostras é onde descobri-las. A tarefa de elicitação pode ser auxiliada por especialistas do domínio, usuários finais e fontes especializadas de informação. Como uma regra chave, títulos anteriores de sucesso na indústria, que podem ser classificados como pertencentes ao domínio, são uma das muitas fontes seguras.

3.1.4. Execução da Análise de Domínio

Esta tarefa é responsável pela atual execução da análise de domínio, registrando informação sobre cada uma das amostras selecionadas da linha de produção diante do conjunto de funcionalidades especificado. É sugerido que toda informação colhida deve ser agrupada juntamente num catálogo, dessa forma o processo de extração de características comuns e variáveis é feita de maneira simples. Projetistas da fábrica podem optar por usar ferramentas para auxiliar esta tarefa. Uma opção é o uso do *Game Evaluation Records* (GERs) para uma análise informal do domínio.

Um GER apresenta campos correspondendo às funcionalidades do domínio sob análise, onde a informação específica da amostra é documentada. Essas estruturas informais destinam prover uma abordagem uniforme dentre as muitas avaliações das amostras, ainda sendo ágil o suficiente para o processo.

3.1.5. Extração de Semelhanças e Variabilidades

Até o momento, tudo o que os projetistas têm são análises individuais das amostras. O propósito dessa tarefa é estudar as informações obtidas para criar um entendimento do que é comum e do que é variável no domínio da fábrica, assim finalmente atingindo um nível de abstração que compreensivelmente representa todos os jogos pertencentes ao domínio.

Variabilidade é a habilidade de mudar ou customizar um sistema. A fim de expressar semelhanças e variabilidades do domínio, o uso de *feature models* (Kang, Cohen, Hess, Novak, & Peterson, 1990), extraído da metodologia FODA, é sugerido. Um *feature model* representa uma hierarquia de propriedades dos conceitos do domínio. As propriedades são usadas para discriminação entre instâncias, por exemplo, sistemas ou aplicações dentro do domínio. As propriedades são relevantes para os usuários finais. Na raiz da hierarquia existe uma funcionalidade conceitual assim chamada, representando toda uma classe de soluções. Abaixo desta funcionalidade existem sub-funcionalidades hierarquicamente estruturadas mostrando propriedades refinadas. Cada uma dessas funcionalidades é comum a todas as instâncias, exceto as marcadas como opcionais, desse modo não necessariamente fazendo parte de todas as instâncias.

Um *feature model* representa uma visão abstrata para propriedades de todas as instâncias do domínio (Riebisch, 2003). Cada funcionalidade cobre um conjunto de requisitos. Selecionando um conjunto de funcionalidades opcionais uma instância do domínio pode ser definida. Todas as funcionalidades mandatórias são parte da instância por definição. *Feature Models* são usados para desenvolvimento e aplicação de linhas de produção de *software*.

Uma opção de modelagem é a notação de *extended feature model* introduzida por Czarnecki, onde exclusões mútuas (XORs) são distinguidas do caso onde mais do que uma sub-funcionalidade é permitida numa dada funcionalidade (ORs). A multiplicidade introduzida por Riebisch (Riebisch, Böllert, Streitferdt, & Philippow, 2002) pode ser usada para evitar a ambigüidade em alternativas XOR. Finalmente, restrições entre funcionalidades (como *requires* ou *excludes*) podem ser aplicadas. Um sumário destas notações está abaixo na Tabela 1.

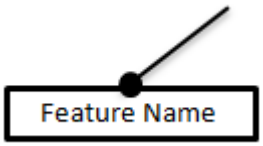
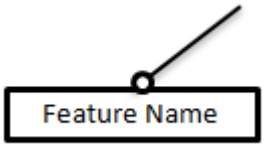
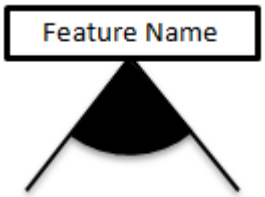
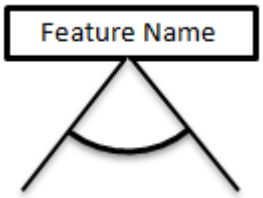
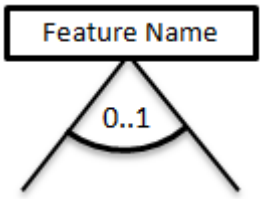
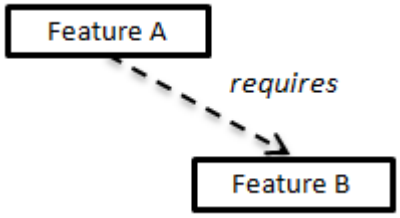
Notation	Description
	Funcionalidade mandatória. Todas as instâncias do domínio devem ter tal funcionalidade.
	Funcionalidade Opcional. Uma instância do domínio pode ter tal funcionalidade.
	Alternativa OR. Uma dada funcionalidade mãe pode permitir mais de uma sub-funcionalidade.
	Alternativa XOR. Sub-funcionalidades de uma funcionalidade mãe podem ser mutuamente exclusivas.
	Alternative XOR com multiplicidade.
	Uma restrição. Nesse caso, a restrição *requires* indica que sempre que uma Funcionalidade A está presente a Funcionalidade B deverá estar presente também. Outras restrições podem ser usadas, como *excludes* e *refinement*.

Tabela 1 – Notação do extended feature model.

Finalmente, relações hierárquicas entre uma funcionalidade e suas sub-funcionalidades controlam a inclusão de funcionalidades para instâncias. Se uma funcionalidade opcional é selecionada para uma instância, então todas as sub-funcionalidades mandatórias devem ser incluídas, e sub-funcionalidades opcionais podem ser incluídas.

Abaixo se pode ver um exemplo de *feature model* para um carro, na Figura 2.

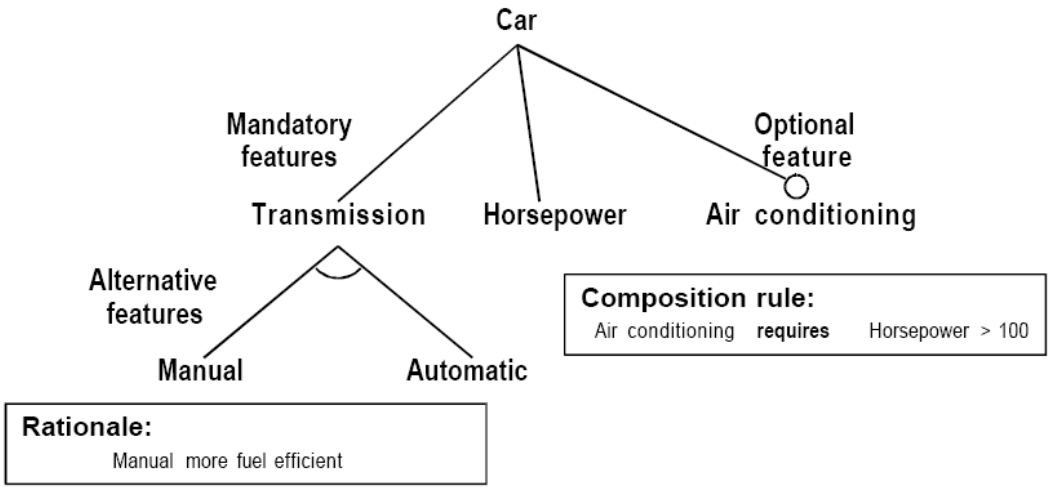


Figura 2 – Exemplo de feature model de um carro.

3.1.6. Validação do Domínio

Nesta etapa, com o *feature model* pronto, uma amostra que não foi incluída na análise será usada para determinar a generalidade do modelo, ou seja, o jogo escolhido deverá ter suas funcionalidades mapeadas no *feature model* projetado.

3.1.7. Análise Incremental e Critério de Parada

Na análise de domínio, experiência e conhecimento são acumulados até atingir-se um limiar. No entanto, entender se o domínio é maduro o suficiente para desenvolver-se *assets* reusáveis (e construir-se uma fábrica de *software*) é um desafio.

Como mencionado anteriormente, o processo de análise de domínio num contexto de fábrica de jogos é incremental. Processos incrementais, no entanto, deverão contar com um critério de parada bem definido, como a conclusão de um artefato ou realização de um marco, desse modo a metodologia poderá sair do laço e ir para a próxima fase.

Critério de parada para execução da análise de domínio em fábricas de jogos pode ser definido antecipadamente como uma conseqüência das restrições de projeto (cronograma e/ou recursos), baseadas em:

- Tempo;
- Orçamento;
- Uma combinação de critérios;

- Ou numa quantidade de amostras pré-definidas, que pode ser o resultado de outra restrição de projeto.

No entanto, independente das restrições do projeto, existe um número ótimo de amostras que maximizam o resultado da análise de domínio: menos amostras que o número ótimo tem um impacto negativo na efetividade da fábrica, mais amostras que o número ótimo implica em perda de recursos. Descobrir o número ótimo de amostras, desse modo, é o maior desafio a ser superado.

Um importante indicador que pode ser usado para dizer empiricamente se a análise pode ser finalizada é o *domain understanding churn*, por exemplo, quanto o entendimento do domínio mudou após cada amostra ser analisada. Se, após a análise das últimas amostras, novas funcionalidades e variações forem identificadas, significa que a compreensão do domínio não é suficiente e a análise deve continuar. Em contrapartida, se não é relevante adições ao entendimento do domínio após a análise das amostras, pode se assumir que as semelhanças e variabilidades mais importantes foram cobertas e o esforço para descobrir novas, se existirem, pode não valer a pena.

3.2. Considerações sobre a Metodologia aplicada

Para a elaboração deste projeto, duas metodologias para análise de domínio foram estudadas, a SharpLudus, proposta por Furtado (Furtado) e a *Feature-Oriented Domain Analysis* (FODA) (Kang, Cohen, Hess, Novak, & Peterson, 1990). A metodologia SharpLudus foi escolhida e será aplicada no próximo capítulo sobre o domínio de jogos RPG.

Focada na automatização no desenvolvimento de jogos, a SharpLudus torna-se mais adequada ao objetivo deste projeto. A praticidade em sua aplicação e os resultados obtidos em cada fase tornam o processo de análise de domínio mais rápido e menos cansativo, em contrapartida ao uso do FODA. Apesar de *feature model* ser derivada do FODA, o uso desta metodologia em sua totalidade se mostrou muito extensa e desgastante, principalmente pelo fato de ser genérica, isto é, usada para análise de qualquer espécie de *software.* Como foi visto anteriormente, um ponto forte em projetos de desenvolvimento de domínio específico é o foco. Desta forma, SharpLudus se

encaixa perfeitamente, com foco em projetos de automatização na produção de jogos.

4. Domínio Alvo: *Role-Playing Games*

Neste capítulo a metodologia SharpLudus será aplicada ao domínio escolhido, o de jogos do estilo *Role-Playing.* Todas as etapas da análise do domínio serão apresentadas nas próximas sessões, após uma breve descrição do domínio alvo.

4.1. Jogos RPGs Eletrônicos

Derivados dos RPGs de “mesa”, os RPGs eletrônicos ainda guardam as principais características das versões originais. Uma história bem estruturada e complexa, a importância da atuação dos jogadores, o caráter aventureiro e estratégico, estão presentes na maior parte dos jogos digitais deste estilo.

É útil salientar que a adaptação da versão de mesa para a eletrônica trouxe alguns desafios e impôs mudanças em algumas características fortes do estilo. O *dungeon master*, ou simplesmente mestre, é uma peça fundamental nos RPGs de mesa. Ele narra a aventura e controla as ações dos personagens, muitas vezes tendo que tomar decisões e mudar o rumo da história de improviso. Na versão digital não existe a noção de mestre, sendo este papel delegado ao programa em si. Nem por isso as ações deixam de sofrer controle ou a história deixa de tomar rumos diferentes a partir destas ações.

Outra questão que trouxe certas desavenças aos adeptos do estilo é a interpretação do papel. Na mesa, o jogador incorpora o personagem, como numa peça teatral. As expressões faciais e corpóreas são cruciais, assim como a criatividade para encarar as diversas situações. Na versão digital o jogador move seu personagem, ou grupo, e atua de forma limitada, muitas vezes em modo *single player*. Dessa forma o caráter teatral desaparece, o que faz como que muitos jogadores ortodoxos de RPG desconsiderem os jogos eletrônicos como pertencentes ao estilo. Alguns jogos permitem o modo *multiplayer*, o que de certa forma aproxima do caráter original. Outros jogos mais atuais *online*, os *Massively Multiplayer Online RPG* (MMORPG), proporcionam um ambiente de imersão onde os jogadores podem conversar, até mesmo com voz, e dessa forma atuar e planejar suas ações abertamente para os jogadores do grupo.

Além de história e interpretação, outras noções são importantes quando se trabalha sobre o domínio RPG. Os personagens possuem atributos, como força, destreza, inteligência, dentre outros, e características que definem sua personalidade. A noção de raça é bastante pertinente, pois dependendo dela alguns atributos sofrem variações, além das características inerentes a cada raça. A classe define a especialização do personagem não só em relação à batalha, mas a conhecimentos gerais, como medicina, história, religião, dentre outros.

O mundo onde ocorrem as aventuras geralmente possui características medievais, com elementos fantásticos e imaginários. Lugares como cidades, florestas, *dungeons* (masmorras), são muito comuns. Os personagens andam em grupos, e dessa forma percorrem o mundo batalhando com inimigos e fazendo *quests*, em busca de recompensas. *Quests* são missões, como matar um determinado inimigo, encontrar um objeto mágico, que são dadas por *Non-Player Characters* (NPCs), personagens controlados pelo mestre ou computador. Missões são importantes não só para ganhar recompensas, como ouro e itens, mas para o desenvolvimento do personagem, que ganha *experience points* (XP), passando de níveis, melhorando suas habilidades de batalha, conhecimentos e até mesmo alterando seu rumo e personalidade.

Uma característica muito forte nesse estilo é a batalha. No momento de batalha, os jogadores decidem quais habilidades do personagem irão usar, além da tática utilizada pelo grupo para vencer mais rapidamente e sem perdas na equipe. Ao final do combate os jogadores podem pilhar os corpos, atrás de objetos de valor, e ganham pontos de experiência Nos RPGs eletrônicos, o subdomínio que foca no combate é denominado RPG Tático. O subdomínio focado nas ações dos personagens é denominado RPG de ação. Neste trabalho esses subdomínios não serão detalhados.

Outro detalhe importante é o sistema comercial. As transações comerciais geralmente são feitas com metais preciosos, mas alguns jogos apresentam uma moeda própria. Os itens, como equipamentos, poções de cura, entre outros, são vendidos geralmente por NPCs, mas podem ser trocados entre personagens.

O estilo RPG é bastante complexo, como pôde ser observado. Isto dificulta bastante mapear todas as características do estilo e abranger todas as

possibilidades do domínio. Desta forma a ênfase deste projeto será em um grupo de características consideradas mais pertinentes, definindo assim o escopo da fábrica de jogos.

4.2. Funcionalidades do domínio sob Análise

Nesta etapa foram definidas funcionalidades e características consideradas importantes no domínio de jogos RPG eletrônicos. A Tabela 2 apresenta as variáveis escolhidas para análise.

Características	Descrição
Descrição	Principais características do jogo em geral.
História	Breve descrição do enredo.
Fluxo da história	Descrição de como se dá o desenrolar da história, linearidade, presença ou não de missões alternativas e impacto das ações dos jogadores na história.
Personagens	Características, atributos e habilidades de personagens jogáveis, presença ou não de personagens não jogáveis, quantidade de personagens que podem ser controlados.
Desenvolvimento dos Personagens	Como os personagens evoluem no decorrer da história.
Cenário	Descrição do mundo e época em que se passa a história.
Sistema de batalha	Descreve como as características inerentes aos personagens podem afetar no resultado das ações de batalha. Lógica e visão estratégica da batalha também são pontos cruciais neste item.

Fluxo em batalha	Ações que podem ser tomadas dentro da batalha e como os personagens podem interagir neste ambiente.
Fluxo fora da batalha	Ações que podem ser tomadas fora da batalha e como os personagens podem interagir.
Itens	Itens e equipamentos que podem ser encontrados durante o jogo.

Tabela 2 – Variáveis utilizadas para análise.

4.3. Seleção de Amostras

Foram escolhidos 5 jogos considerados representativos no domínio:

- Final Fantasy VI
- Chrono Trigger
- Baldur's Gate II
- Seiken Densetsu III
- Neverwinter Nights

A seleção dos jogos foi feita baseada no sucesso de mercado e aceitação por parte dos adeptos ao estilo. Todas as amostras escolhidas foram lançadas por empresas consolidadas e famosas em todo o mundo. *Final Fantasy VI, Baldur's Gate II* e *Seiken Densetsu III* são seqüências de jogos de mesmo nome que também foram grande sucesso. *Chrono Trigger* e *Neverwinter Nights* originaram seqüências com grande potencial de aceitação: *Chrono Cross* e *Neverwinter Nights 2.*

4.4. Execução da Análise de Domínio

Nesta etapa as variáveis escolhidas anteriormente foram analisadas em relação a cada um dos jogos escolhido. As tabelas seguintes mostram o resultado da análise, documentado utilizando GERs.

Chrono Trigger

Descrição: jogo lançado em 11 de Março de 1995, pela SQUARE ENIX, uma produtora japonesa de jogos bastante consolidada no mercado. Desenvolvido inicialmente para a plataforma Super Nintendo, e posteriormente adaptado para Playstation, da Sony, e Nintendo DS. O jogador controla o protagonista e seus companheiros, que compõem o núcleo de personagens jogáveis, num mundo ficcional de duas dimensões.

História: o ano é 1000 D.C. Crono, um jovem habitante de uma vila no Reino de Guardia, acorda numa bela manhã animado para ver a mais nova invenção de sua amiga Lucca na feira *MilleniaII,* a máquina de tele transporte. Seguindo para seu intento, encontra uma moça que se apresenta com o nome de Marle. Os dois seguem juntos para prestigiar a invenção e se tornam voluntários para testar a máquina. Ao entrar em funcionamento a máquina reage com um pendante que Marle carrega, abrindo um portal para outra era. Após Marle desaparecer no portal, Crono e Lucca decidem ir resgatá-la, recriando um portal para o ano de

600 D.C. O que eles não sabem é que a partir daí o objetivo de suas vidas irá mudar drasticamente de apenas salvar a jovem Marle para salvar o mundo de ser destruído por uma criatura gigante chamada Lavos.
Fluxo da História: essa característica é de extrema complexidade neste jogo. Seguindo por diversas eras, 1000 D.C, 600 D.C, 1999 D.C, 2300 D.C, 12.000 A.C e 65.000.000 A.C., novos personagens são encontrados, podendo integrar ou não o grupo principal. Cada um desses personagens possui uma história, com missões que deverão ser cumpridas durante o jogo. Um ponto de extrema complexidade são os inúmeros finais que este jogo possui, dependendo do grupo principal escolhido e suas ações no decorrer do enredo. O jogo permite saltos em eras e possui momentos de *flashback* dos personagens.
Personagens: o grupo principal é integrado por três personagens, às vezes devendo possuir o protagonista. Os atributos dos personagens são: pontos de vida, pontos de magia, experiência, ataque, força, velocidade, evasão, defesa mágica, defesa física, precisão, magia e vigor. Equipamentos utilizados pelos personagens, como armas, elmos, armaduras e acessórios podem trazer bônus para os atributos. As habilidades dos personagens são dependentes do tipo de arma que cada um é especializado, além de magias elementais, sendo que cada personagem possui especialidade em um elemento. As magias podem ser utilizadas dependendo dos pontos de magia do personagem no momento da batalha, que podem ser repostos através de itens, assim como os pontos de vida. As magias podem ser combinadas entre os personagens do grupo, formando técnicas diferentes.
Desenvolvimento dos personagens: os personagens evoluem através do ganho de pontos de experiência em batalhas e missões. Atingindo um limiar, o personagem passa de nível, ocorrendo um incremento em seus atributos. A passagem de níveis não é igual para todos os personagens. Os personagens aprendem novas habilidades com o passar de níveis, e podem utilizar equipamentos mais aprimorados.
Cenário: o mesmo mundo é representado na ótica de cada era em que a história se encontra. A navegação ocorre via um mapa *overworld,* termo usado para designar perspectiva em terceira pessoa do mundo fictício, representando a paisagem a partir de uma visão superior reduzida. O jogador visualiza apenas

uma porção deste mapa, e a medida que se movimenta pode visualizar outras porções que deseja explorar. Um mapa de todo o mundo pode ser acessado, sendo muito útil ao deslocar-se por ele. Áreas florestas, cidades e *dungeons* compõem o mapa. Ao entrar nessas áreas, o jogador passa a ter uma visão superior reduzida apenas do ambiente em que está situado e todos os detalhes que o compõem, podendo interagir com pessoas e objetos. A movimentação pode ser feita em qualquer direção, dentro dos limites do ambiente.
Sistema de batalha: os ataques são dependentes da iniciativa dos personagens, que define quem irá fazer a ação primeiro. A iniciativa muda de acordo com o atributo velocidade. O personagem tem direito de fazer, durante sua ação de batalha, um ataque físico, mágico ou utilizar um item. As batalhas não são aleatórias, os inimigos podem ser visualizados no mapa. Desta forma o jogador escolhe se irá entrar ou não no campo de visão do inimigo. Ao se vencer uma batalha, o grupo recebe pontos de experiência, moedas de ouro e podem receber itens.
Fluxo em batalha: ao entrar no modo de batalha, os personagens não podem ser movimentados. Um menu aparece, com opções e informações de batalha, como os ataques, itens a serem usados ou fuga. A iniciativa de cada personagem é representada por uma barra, que é incrementada até que o ele esteja pronto para fazer uma ação. Como a batalha ocorre em tempo real, personagens que tenham sua barra de iniciativa incrementada mais rapidamente podem agir várias vezes antes dos outros. As ações tomadas se restringem a atacar (fisicamente e magicamente) e utilizar itens.
Fluxo fora da batalha: fora das batalhas os personagens têm livre movimentação, e podem interagir com NPCs, conversando ou negociando itens. Os personagens também podem interagir com o ambiente, abrindo baús, portas, coletando itens, descobrindo segredos, entre outros. Podem também usar itens para cura, entre outros, e em certos locais salvar o jogo. Ataques a NPCs fora do modo de batalha não são permitidos.
Itens: os itens coletados durante o jogo podem ser equipamentos, como armas, armaduras,elmos; itens de cura, recuperação de pontos de magia, pontos para aumentar atributos, e até itens especiais pertencentes a missões.

Tabela 3 – Análise do jogo Chrono Trigger.

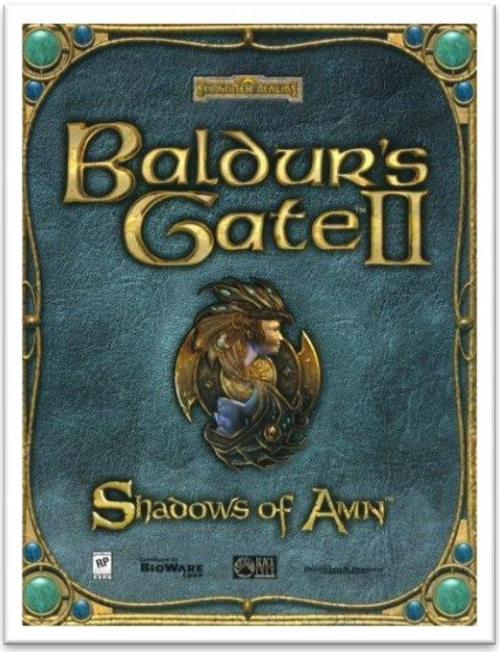
Final Fantasy VI
   
Descrição: jogo lançado em 2 de Abril de 1994, pela Square Enix. Desenvolvido inicialmente para a plataforma Super Nintendo, e posteriormente adaptado para Playstation, da Sony, e Game Boy Advance, da Nintendo. O jogador controla um grupo de personagens jogáveis, num mundo fictício de duas dimensões.
História: Terra, uma meia-humana que perdeu a memória, é controlada pelo império numa missão na cidade de Narshe em busca de um Esper congelado. Espers são seres mágicos, que viviam num mundo paralelo, que foi destruído pelos humanos. Ao encontrar o Esper, uma reação muito forte acontece. Terra é resgatada por opositores do Império, e ao recobrar sua consciência conta que foi explorada e controlada pelo Império. Um ladrão participante do grupo promete protegê-la, e a partir daí começam uma jornada lutando contra o Império e seus objetivos malignos. Durante essa jornada encontram aliados e inimigos, e descobrem os segredos por trás da dominação do Império.
Fluxo da História: a história possui dois momentos, antes e pós destruição do mundo. Personagens são encontrados em diversos momentos, chegando a somar quatorze jogáveis. Os personagens principais possuem assuntos pendentes com relação a suas histórias de vida, fazendo parte das missões a

serem cumpridas no jogo. A história é linear em sua maior parte, com alguns momentos de *flashback* dos personagens.
Personagens: o numero de personagens jogáveis no grupo varia bastante, podendo ser de um até quatro. Em alguns momentos mais de um grupo pode ser controlado. Os atributos dos personagens são: nível, pontos de vida, pontos de magia, experiência, ataque, força, velocidade, evasão, defesa mágica, defesa física, evasão mágica, vida e magia. Os personagens também possuem um *status*, que é alterado caso sejam envenenados, petrificados, cegados, dentre outras maldições. Os equipamentos utilizados, como armas, escudos, armaduras e elmos trazem bônus para os atributos. Cada personagem possui uma especialidade em armas e alguns possuem magia inata. Porém personagens sem magia podem aprendê-la.
Desenvolvimento dos personagens: os personagens evoluem através do ganho de pontos de experiência em batalhas e missões. Atingindo um limiar, o personagem passa de nível, ocorrendo um incremento em seus atributos. A passagem de níveis não é igual para todos os personagens. Os personagens aprendem novas habilidades com o passar de níveis, e podem utilizar equipamentos mais aprimorados. Os personagens podem aprender magias de mais diversos tipos, através do uso de *magicites*, que são Espers em modo dormente. Dessa forma, todos os personagens podem aprender todas as magias, porém alguns recebem bônus dependendo do tipo da magia.
Cenário: o mundo fictício é composto por elementos medievais e futuristas. A navegação também é feita por um mapa *overworld*, com visão limitada à porção em que o jogador se encontra. Um mapa do mundo pode ser acessado. O mapa é composto de cidades, florestas, *dungeons*, e seus interiores são visualizados com mais detalhes a partir do momento em que os personagens entram no ambiente. Os jogadores interagem com objetos do ambiente e NPCs. A movimentação pode ser feita em qualquer direção, nos limites do ambiente.
Sistema de batalha: é também similar ao jogo anterior. O personagem possui uma barra que representa a sua iniciativa, sendo a batalha em tempo real. Uma grande diferença aqui é que as batalhas são randômicas, o jogador não pode evitá-las.

Fluxo em batalha: as ações que podem ser tomadas durante a batalha são limitadas a ataque físico ou mágico. O uso de itens é permitido em vez de um ataque. Ao entrar em modo de batalha, um menu com informações e opções de batalha pode ser visualizado. Ao se vencer uma batalha, o grupo recebe pontos de experiência, moedas de ouro e podem receber itens.
Fluxo fora da batalha: fora das batalhas os personagens têm livre movimentação, e podem interagir com NPCs, conversando ou negociando itens. Os personagens também podem interagir com o ambiente, abrindo baús, portas, coletando itens, descobrindo segredos, entre outros. Podem também usar itens para cura, entre outros, e em certos locais salvar o jogo Ataques a NPCs fora do modo de batalha não são permitidos.
Itens: os itens coletados durante o jogo podem ser equipamentos, itens de cura, recuperação de pontos de magia, alteração de *status*, e até itens especiais pertencentes a missões.

Tabela 4 – Análise do jogo Final Fantasy VI.

Baldur's Gate II: Shadows of Amn™



Descrição: O segundo jogo da linha Baldur's Gate, produzida pela Bioware, continua a história do primeiro jogo, com o mesmo protagonista e trazendo de volta muitos dos personagens do jogo anterior. Utilizando um sistema e um cenário de grande sucesso em RPGs de mesa, respectivamente Advanced Dungeons and Dragons e Forgotten Realms, Baldur's Gate II é um mundo de vastidão e detalhe imenso. O jogo pode ser jogado por um ou vários jogadores, onde eles controlam até seis personagens jogáveis por vez e interagem com uma infinidade de personagens não jogáveis enquanto viajam e se aventuram pelo imenso cenário.

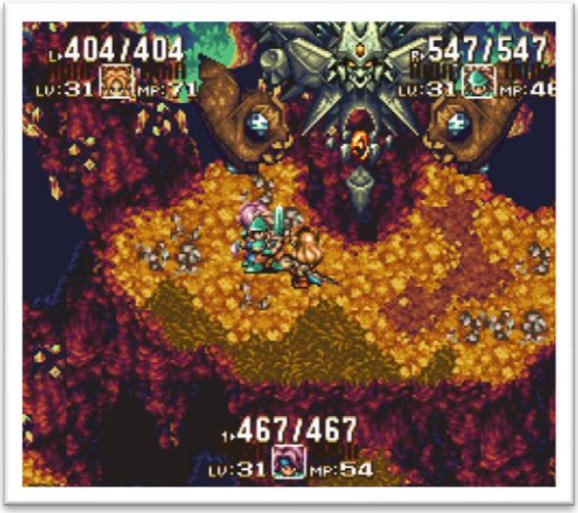
História: O personagem principal, cujo nome é escolhido pelo jogador, é capturado juntamente com seu grupo de amigos para um tipo de masmorra. Lá

eles sofrem todos os tipos de torturas e experimentos na mão de um homem chamado Irenicus. Durante uma invasão que ocorre a esta masmorra, eles conseguem escapar, porém a irmã adotiva do personagem principal, Imoen, é levada em custódia por uma organização chamada Cowled Wizards. A partir daí, o protagonista e seus amigos tentam salvar Imoen, até que a história muda drasticamente quando ele começa a entender melhor sua própria natureza.
Fluxo da História: as ações tomadas por cada personagem não impactam no final da trama, mas sim no seu desenrolar. Existem algumas *side quests*, missões alternativas, que não necessitam ser cumpridas para o término do jogo, mas impactam no desenvolvimento do mesmo.
Personagens: O grupo consiste de 1 a 6 personagens controlados pelo jogador. Cada personagem é definido pelo sistema de regras de Advanced Dungeons and Dragons. Os principais atributos e características dos personagens são: Força, Destreza, Constituição, Inteligência, Sabedoria, Carisma, Pontos de Vida, Classe de Armadura e Tentativa de Acerto em Categoria de Armadura Zero (um atributo que define a precisão dos ataques físicos do personagem). Fora essas características, existem a Raça e Classe de cada personagem, sendo que cada Raça e Classe podem ter características específicas ou proporcionar bônus ou penalidades aos atributos do personagem.
Desenvolvimento dos personagens: Ao vencer batalhas e cumprir missões, os personagens ganham pontos de experiência. Quando completam uma certa quantidade de pontos, que é definida por sua classe atual, o personagem ganha um novo nível. O jogador escolhe em que classe o personagem vai ganhar seu novo nível, e suas habilidades e características serão melhoradas dependendo da classe que teve seu nível aumentado. Personagens humanos podem ter duas classes, enquanto que personagens de outras raças podem ter até 3 classes.
Cenário: O é baseado em Forgotten Realms, um cenário muito famoso em RPGs de mesa. Dessa vez o principal palco do jogo é o reino de Amn. O jogador controla os personagens numa visão isométrica por mapas que representam uma parte específica da cidade de Athkatla, uma *dungeon* ou uma área ao ar livre. Quando os personagens querem viajar entre um mapa e outro, um mapa do reino de Amn é mostrado para que o jogador escolha o destino desejado.

Sistema de batalha: Assim que um dos personagens decide atacar ou quando ele entra no campo de visão de uma criatura hostil, o combate começa. Não existe um modo exclusivo para combate, sendo a mesma visão isométrica de movimentação utilizada. O combate é feito em turnos, cada personagem age somente no seu turno, podendo fazer ataques físicos, utilizar magias ou outras habilidades. Criaturas derrotadas podem ser pilhadas por ouro ou itens. Embora existam combates aleatórios enquanto os personagens se movimentam de um mapa para o próximo, a maioria dos combates não é aleatório e podem ser evitados ou causados por escolhas em diálogos com NPCs.
Fluxo em batalha: Embora o combate seja feito em turnos, isso é transparente para o jogador, que vê o combate em tempo real. A qualquer momento o jogador pode pausar o combate para pensar melhor nas estratégias e escolher as ações que seus personagens tomarão no seu próximo turno. Um sistema de atalhos existe para facilitar o uso das habilidades dos personagens no combate. O jogador pode designar ações específicas para espaços numa barra de atalho que fica na parte inferior da tela. Cada personagem tem uma barra específica que é exibida quando o jogador seleciona o mesmo.
Fluxo fora da batalha: Fora da batalha o jogador pode movimentar os personagens, conversar com NPCs, negociar itens, roubar lojas e pessoas, iniciar uma batalha atacando uma criatura, utilizar magias e uma infinidade de outras ações. Um ponto interessante é a interação entre o protagonista e os outros membros do grupo de aventureiros. As ações dele melhoram ou pioram a sua relação com cada um dos seus amigos. Eles interagem através de conversas iniciadas pelo jogador ou que ocorrem em momentos aleatórios do jogo. Nessas conversas o protagonista aprende mais sobre os seus companheiros aventureiros e da ao jogador uma visão mais humana deles.
Itens: Existem vários tipos de itens: armas, armaduras, poções, pergaminhos mágicos, chaves, varinhas, etc. Cada personagem tem seu inventário particular, onde pode carregar seus itens. Certos itens só podem ser equipados ou utilizados se o personagem cumprir com certos pré-requisitos, que podem variar desde ter um certo valor num atributo até pertencer a uma certa classe.

Tabela 5 – Análise do jogo Baldur's Gate II: Shadows of Amn.

Seiken Densetsu 3



Descrição: jogo lançado em 30 de Setembro de 1995, pela Square Enix, para plataforma Super Nintendo. É parte de uma série bastante famosa de RPGs, a série *Mana,* sendo conhecido não oficialmente como Secret of Mana II. O jogador escolhe três personagens para começar a jogar dentre seis principais, das mais diferentes classes e habilidades.

História: Todos os personagens estão inicialmente a caminho de uma cidade sagrada (Wendel) até que são contactados por uma fada e no desenrolar dos acontecimentos são envolvidos numa trama onde vários vilões que têm por objetivo ganhar poder e estão indiretamente destruindo a árvore de Mana que caso seja destruída despertará os 8 monstros sagrados que destruirão o mundo.

Fluxo da História: O jogo não é linear e depende inclusive da seleção dos personagens. Existem várias missões alternativas, algumas contribuindo para a história principal, outras não. Mas todas as ações tomadas pelos jogadores nas missões relacionadas à história tem impacto no desenrolar do jogo.
Personagens: No início do jogo o jogador deve escolher três personagens dentre seis disponíveis, cada um com uma história própria e com sua parte na história principal e então controlará os três durante o jogo. Os personagens possuem os seguintes atributos: nível, experiência, ataque, defesa, evasão, defesa, mágica, força, agilidade, vitalidade, inteligência, espírito, pontos de vida e pontos de magia. Alguns personagens utilizam magias enquanto outros apenas utilizam de suas armas, mas ambos possuem ataques especiais que só podem ser usados quando estão carregados. O jogo apresenta NPC's e é com eles a maior parte da interação fora de combate do jogador.
Desenvolvimento dos personagens: Possui um sistema característico de jogos eletrônicos do gênero RPG, com uma ficha de personagem, níveis e experiência. O jogo dispõe de uma árvore de evolução de classe para cada personagem, sendo necessário aumentar certos atributos e conseguir certos itens para avançar para a próxima classe. Há também o desenvolvimento da personalidade dos personagens e conquistas pessoais de cada um deles.
Cenário: O cenário é um mundo fantástico, cheio de criaturas místicas, magia e espíritos. A navegação é feita também através de um mapa *overworld*, da mesma forma que os jogos anteriores. A iteração com o ambiente também é similar, os personagens abrem baús, portas, adentram casas, acionam mecanismos.
Sistema de batalha: O sistema de batalha é classificado como Action-RPG, tal como outros jogos que se utilizam desse sistema, o posicionamento dos personagens é importante, assim como analisar as fraquezas dos oponentes e a maneira como atacam, para colocá-los em posição para serem atacados por uma magia ou ataque especial. A batalha ocorre em tempo real.
Fluxo em batalha: A interação na batalha é livre, exceto em combates classificados como chefes. O jogador não é obrigado a lutar, podendo simplesmente fugir dos oponentes e todas as ações que realiza fora de batalha (exceto falar) podem ser realizadas dentro da batalha.
Fluxo fora da batalha: Os personagens interagem com NPC's e objetos, também podendo utilizar algumas formas de locomoção controlada ou não.
Itens: os itens coletados durante o jogo podem ser equipamentos (armas, escudos, armaduras, etc.), itens de cura, recuperação de pontos de magia, itens que aumentam certos atributos ou que permitem um novo tipo de locomoção.

Tabela 6 – Análise do jogo Seiken Densetsu 3.

Neverwinter Nights
  
Descrição: Neverwinter Nights é um RPG épico que usa as regras do Dungeons and Dragons 3ª edição e é ambientado em Forgotten Realms. O jogo tem um modo de um jogador e um modo de vários jogadores que interagem pela internet. Cada jogador controla o personagem criado por ele e pode contratar um mercenário para ajudá-lo, que é controlado pela inteligência artificial do jogo. O jogo também inclui um criador de campanhas, para que os jogadores possam criar suas próprias campanhas para jogar online, onde um jogador assume o

papel de *Dungeon Master* e os outros controlam seus personagens.
História: a antes poderosa cidade de Neverwinter é um antro de pânico e terror. Milhares já morreram de uma misteriosa praga, chamada de Morte Uivante, e outros milhares estão infectados. Com risco de uma epidemia se espalhar por toda Faerûn, os Lords de Neverwinter declaram quarentena e trancam os portões, prendendo doentes e sadios dentro dos muros da cidade. Lady Aribeth de Tylmarande convocou todos os aventureiros dentro da cidade, pedindo a eles para manter a ordem e a ajudá-la a encontrar uma cura. Promessas de honra e riqueza trouxeram muitos para o lado de Aribeth, mas em vão. A praga se espalha a cada dia e arrasa os bairros mais pobres, como um terremoto. Muitos heróis tomabaram e não há cura a vista.
Fluxo da História: assim como Baldur's Gate II, as ações não impactam no final da história, mas sim no seu desenrolar. Missões alternativas são bastante comuns, e influenciam fortemente no desenvolvimento dos personagens.
Personagens: Cada personagem possui uma raça, uma classe, um alinhamento, um conjunto de seis atributos, um conjunto de habilidades e talentos e uma lista de magias. O jogador controla somente o personagem criado por ele, mas pode dar algumas ordens simples ao seu mercenário, como atacar o inimigo mais próximo ou ficar parado. Todas essas características dos personagens foram tiradas do sistema de RPG Dungeons and Dragons 3ª edição.
Desenvolvimento dos personagens: Ao progredir no jogo, o personagem principal acumula pontos de experiência, que ao atingirem um certo patamar fazem com que ele suba de nível. A cada nível, o jogador pode escolher a classe em que o personagem ganhará seu próximo nível, ficando mais forte numa classe que ele já possuía, ou adquirindo uma nova no nível 1. A cada nível o personagem ganha pontos de vida, talentos e atributos, alem das características que são melhoradas dependendo da classe escolhida.
Cenário: O cenário é baseado no mundo Forgotten Realms. O jogador interage com o mundo através de uma visão tridimensional, onde ele pode movimentar a câmera livremente. O mundo é exibido como uma série de mapas tridimensionais. Quando o jogador deseja passar de um mapa para o próximo, esse deve movimentar o personagem até a borda do mapa atual para passar

para o próximo. Cada mapa pode dar saída para vários outros.
Sistema de batalha: o sistema deixa livre ao jogador tornar mais importante a ação ou estratégia na batalha, ou equilibrar os dois. É bastante semelhante a Baldur's Gate II, com batalhas não aleatórias, pois o jogador tem a opção de evitá-las.
Fluxo em batalha: Não existe um modo de jogo específico para batalhas. Assim como em Baldur's Gate II, o combate é travado em turnos, porém isso fica transparente para o usuário, podendo ele pausar o combate a qualquer momento. Neverwinter Nights também possui o mesmo sistema de barra de atalho, que facilita ao jogador acessar as várias ações e habilidades que o seu personagem pode utilizar no combate. O jogador pode movimentar seu personagem pelo mouse ou telo teclado, e usa o mouse para interagir com outros personagens e com o ambiente. As possíveis ações que estão disponíveis para o personagem são dispostas em um menu radial, que é ativado pelo clique direito do mouse.
Fluxo fora da batalha: O jogo se comporta da mesma maneira fora e dentro da batalha, inclusive com os mesmos comandos. A diferença é que fora de combate o protagonista pode interagir conversando com outros personagens, comprando e vendendo itens, etc.
Itens: Existem vários tipos de itens: armas, armaduras, poções, pergaminhos mágicos, chaves, varinhas, etc. Cada personagem tem seu inventário particular, onde pode carregar seus itens. Certos itens só podem ser equipados ou utilizados se o personagem cumprir com certos pré-requisitos, que podem variar desde ter um certo valor num atributo até pertencer a uma certa classe.

Tabela 7 – Análise do jogo Neverwinter Nights.

4.5. Extração de Semelhanças e Variabilidades

Nesta etapa as funcionalidades semelhantes e variáveis serão apresentadas através de *feature models*. Os diagramas abaixo foram desenvolvidos através da ferramenta Feature Model DSL (Furtado, 2008). Alguns modelos ficaram muito extensos e foram divididos. O Apêndice A traz explicações detalhadas sobre cada funcionalidade.

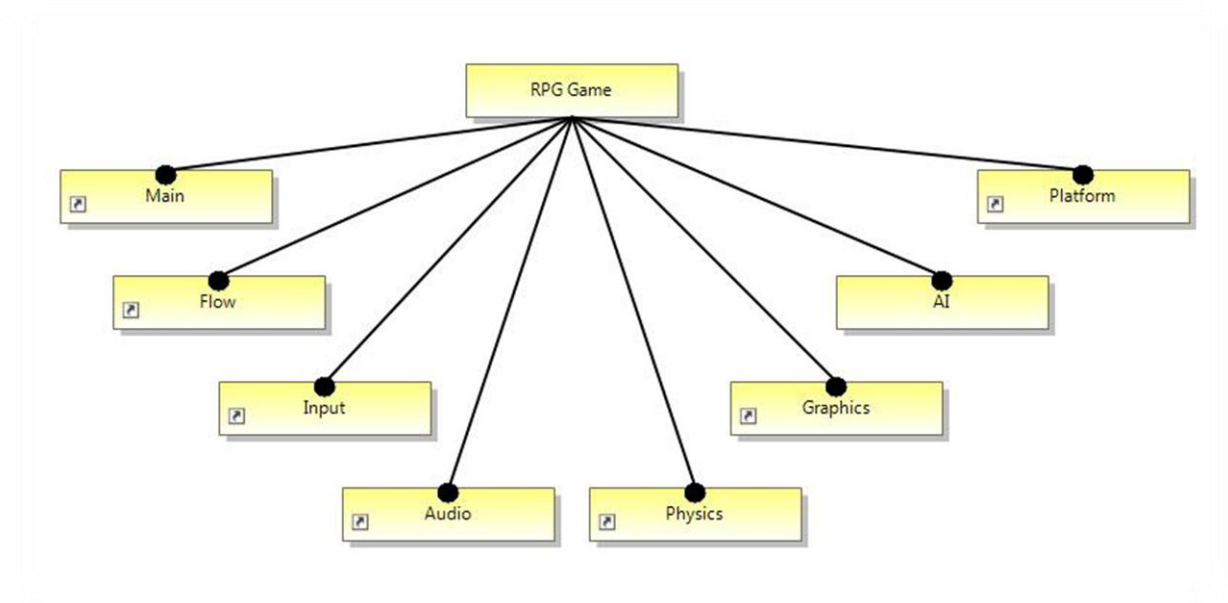


Figura 3 – Feature Model de um RPG.

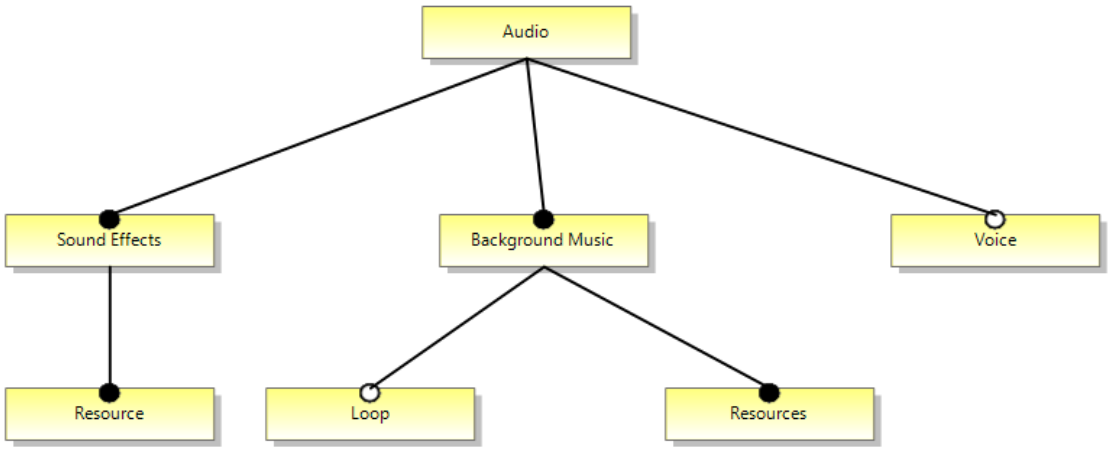


Figura 4 – Feature Model para o sistema de áudio do jogo.

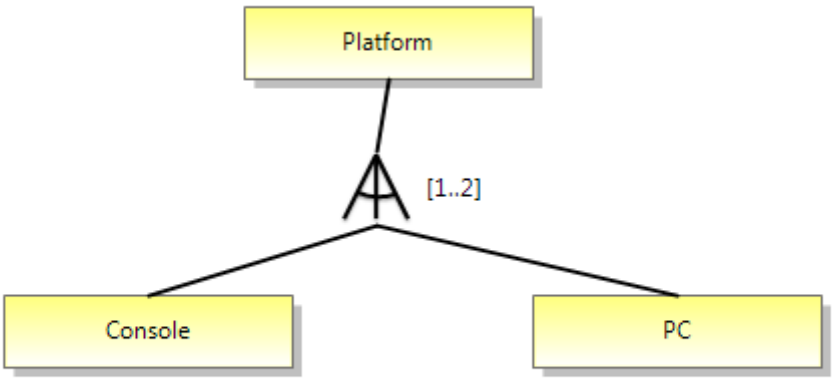


Figura 5 – Feature Model para plataformas de programação.

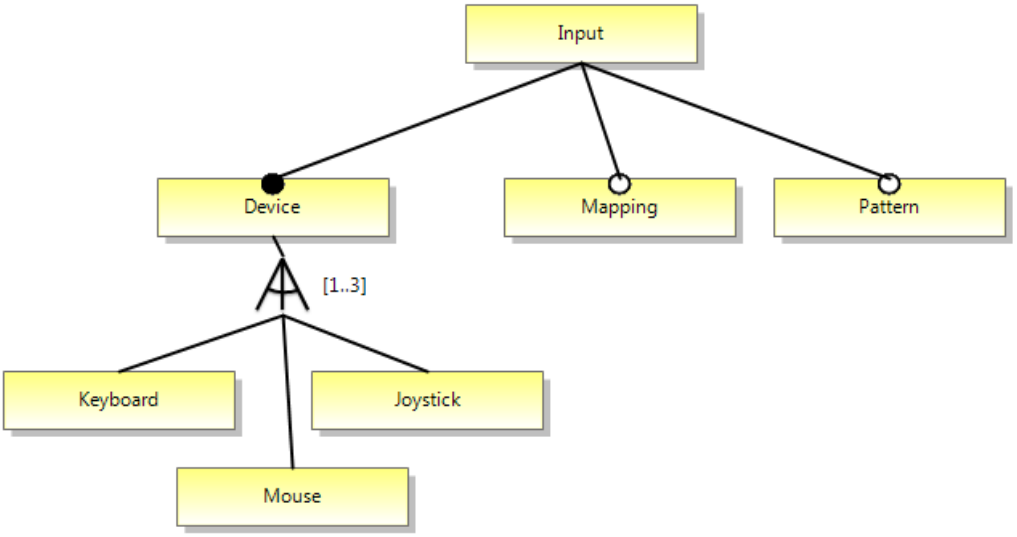


Figura 6 – Feature Model para sistema de entrada.

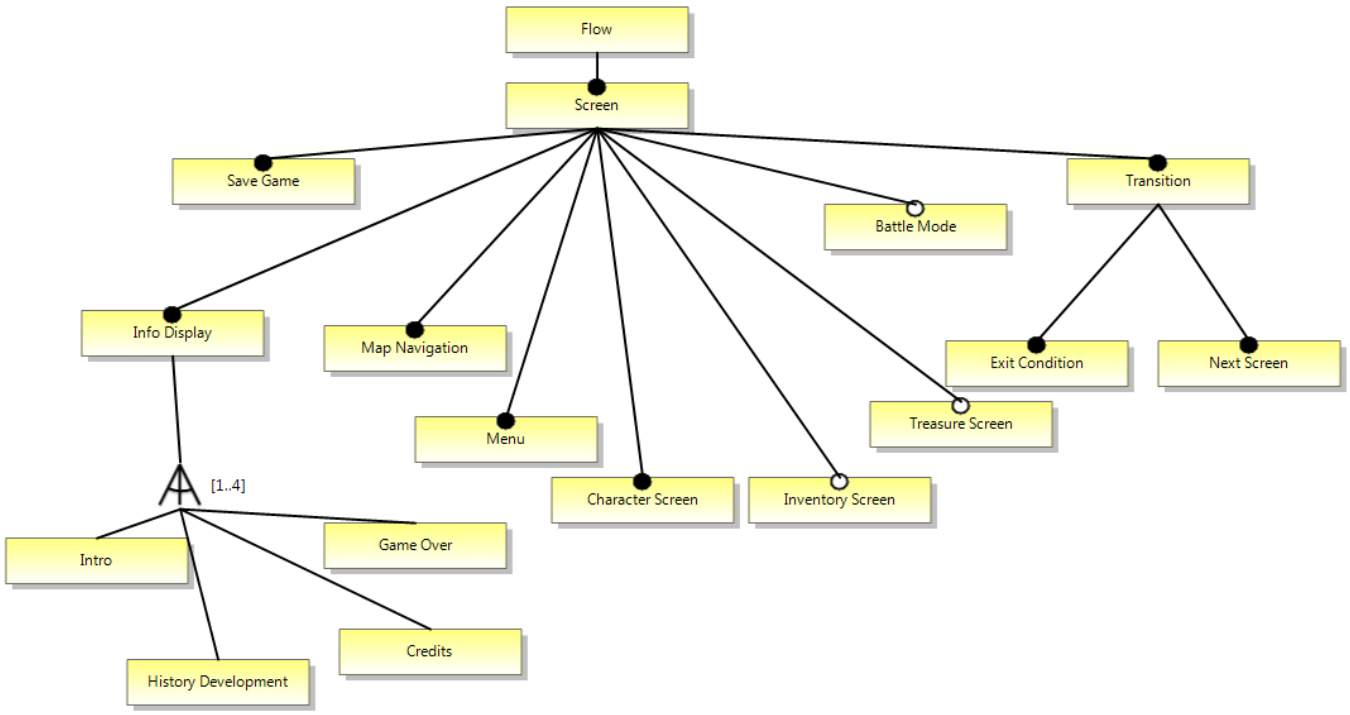


Figura 7 – Feature Model para o sistema de fluxo de telas.

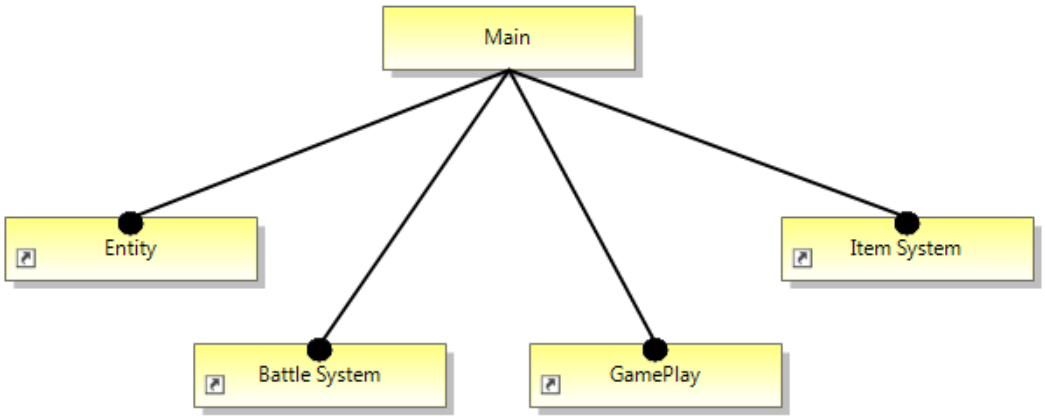


Figura 8 – Feature Model para as características principais do jogo.

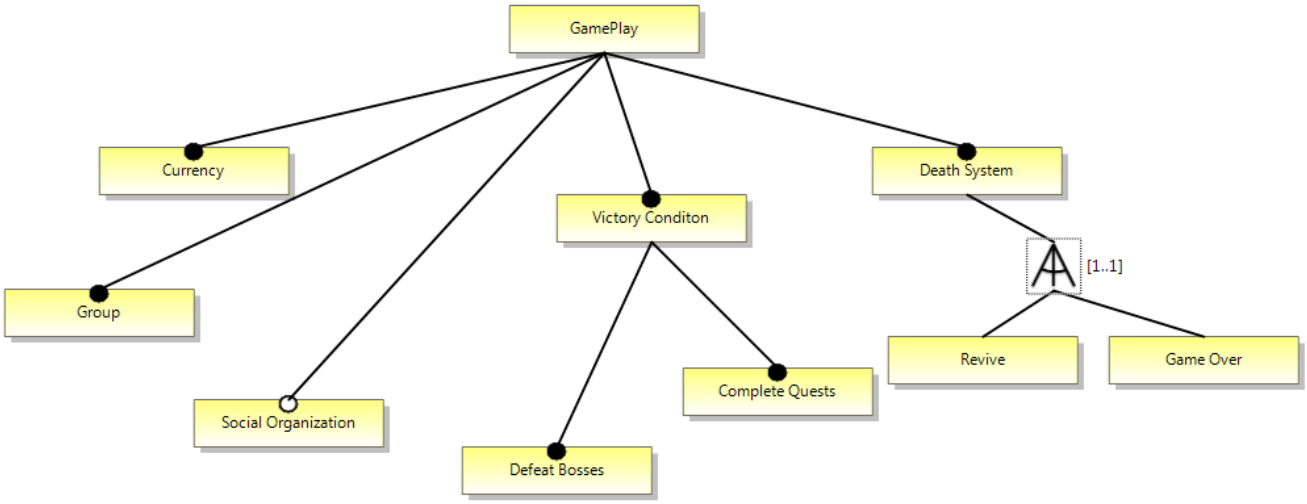


Figura 9 – Feature Model para as características diferenciais de um RPG.

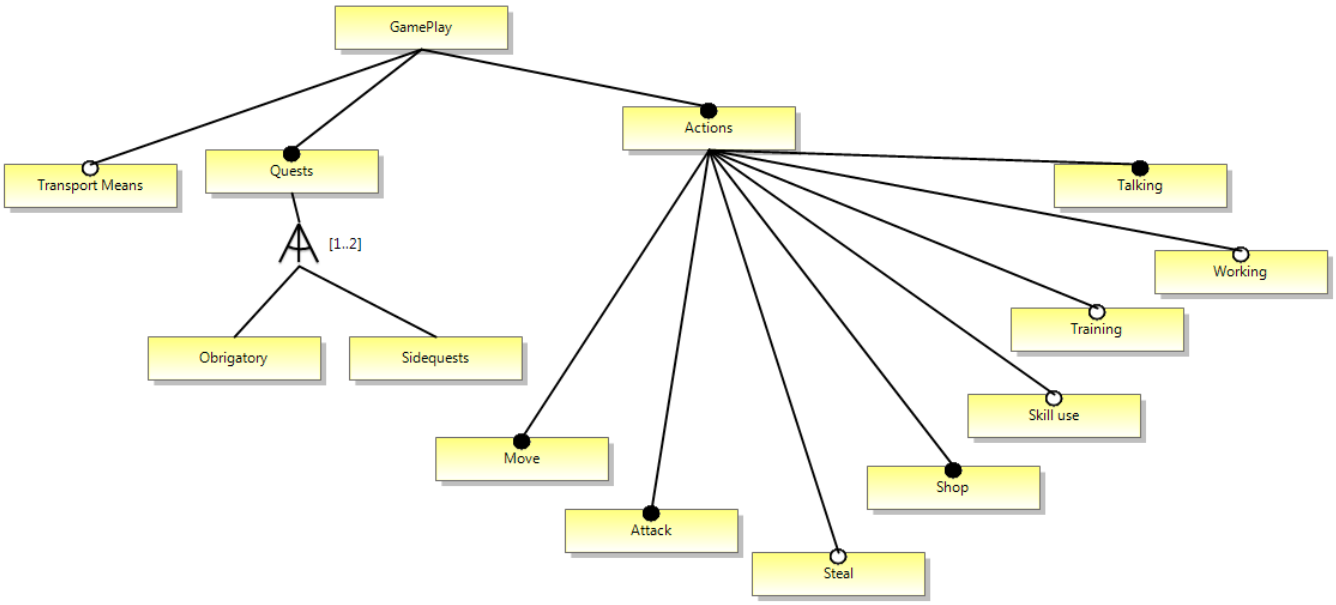


Figura 10 – Feature Model para as características diferenciais de um RPG.

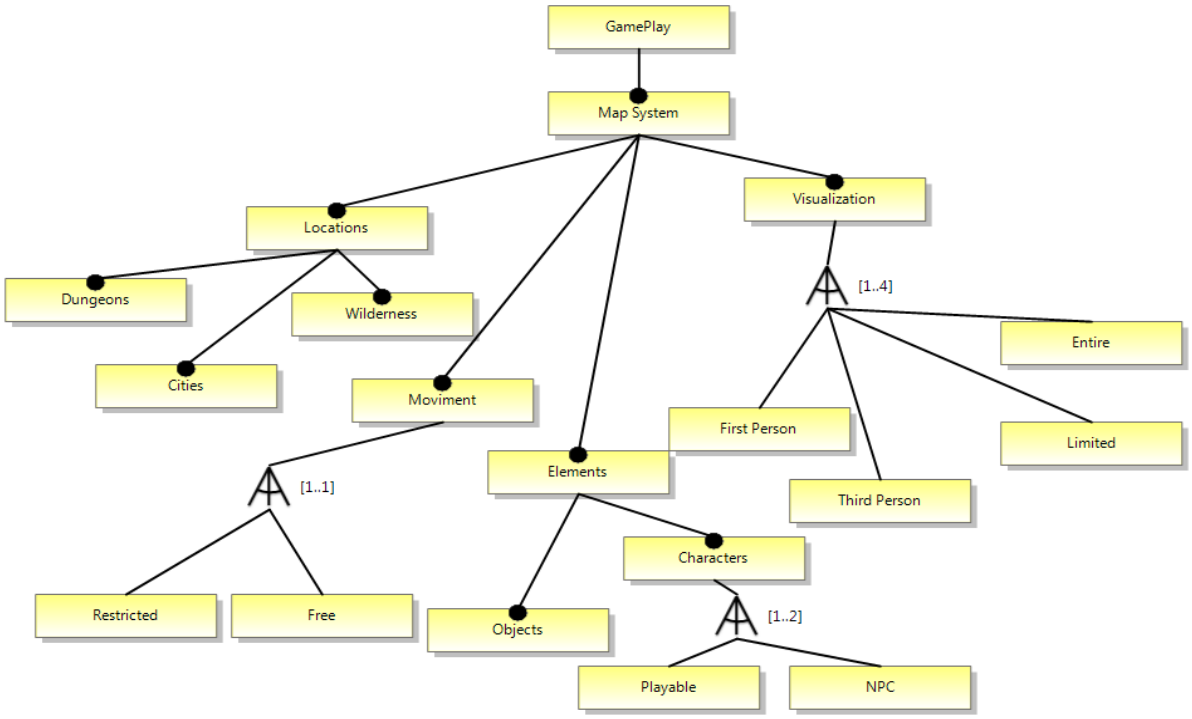


Figura 11 – Feature Model para as características diferenciais de um RPG.

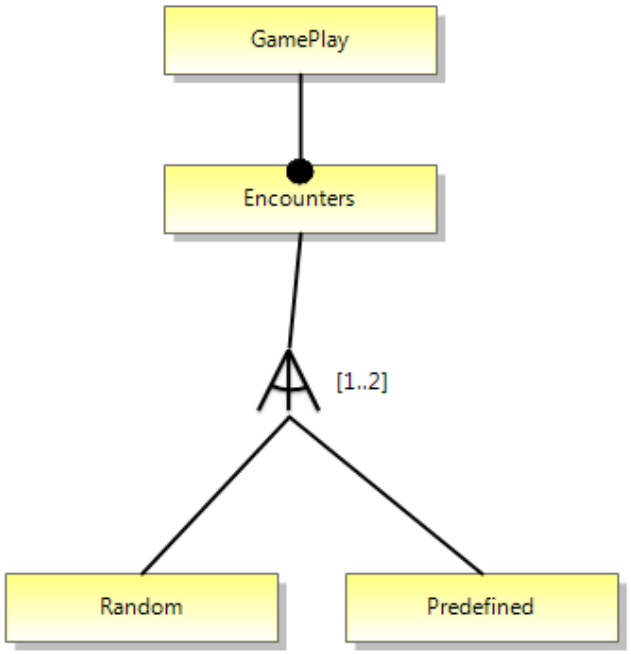


Figura 12 – Feature Model para as características diferenciais de um RPG.

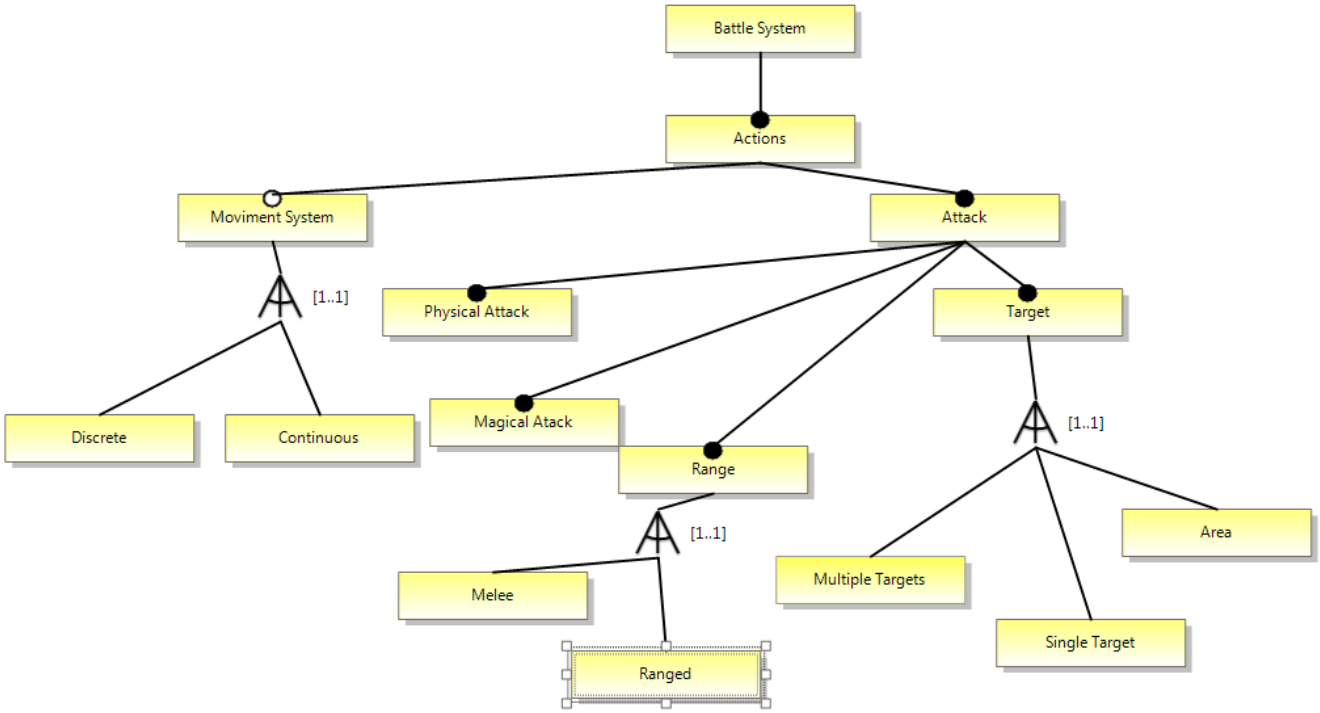


Figura 13 – Feature Model para o sistema de batalha.

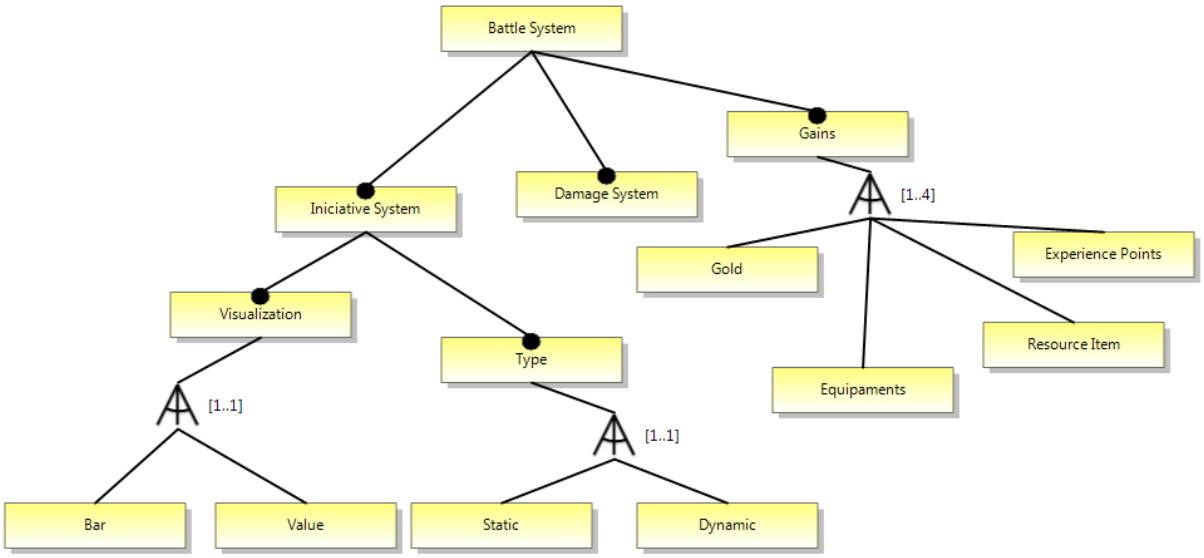


Figura 14 – Feature Model para o sistema de batalha.

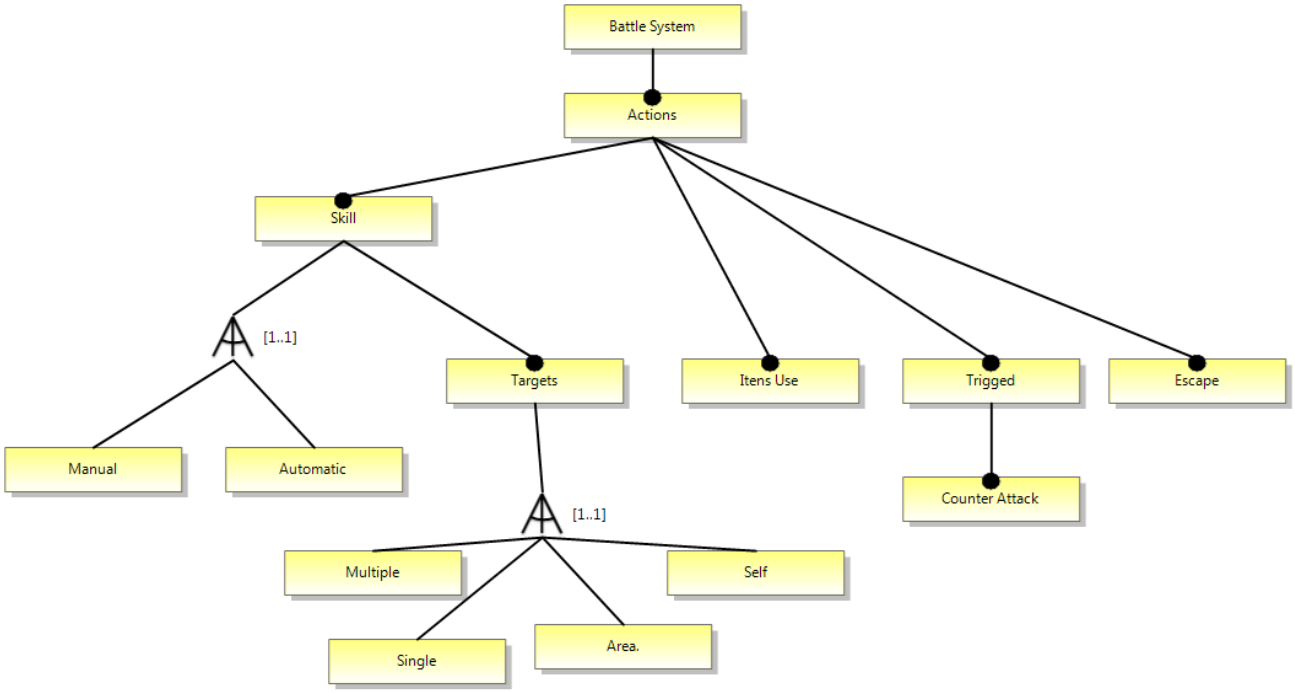


Figura 15 – Feature Model para o sistema de batalha.

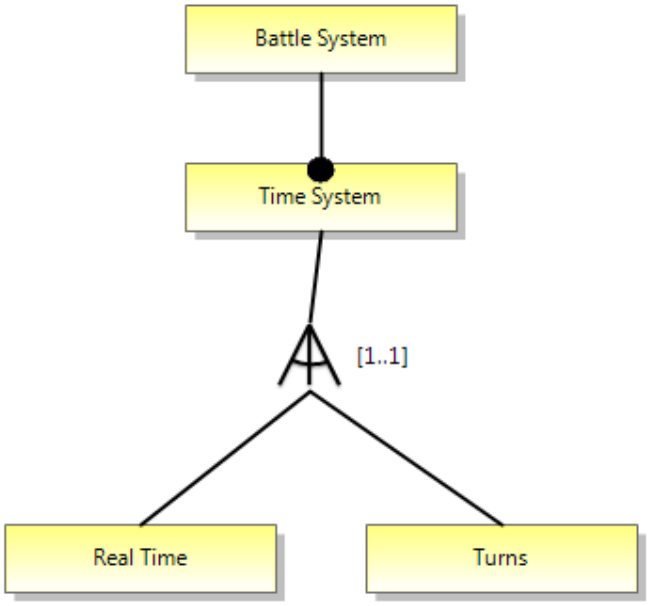


Figura 16 – Feature Model para o sistema de batalha.

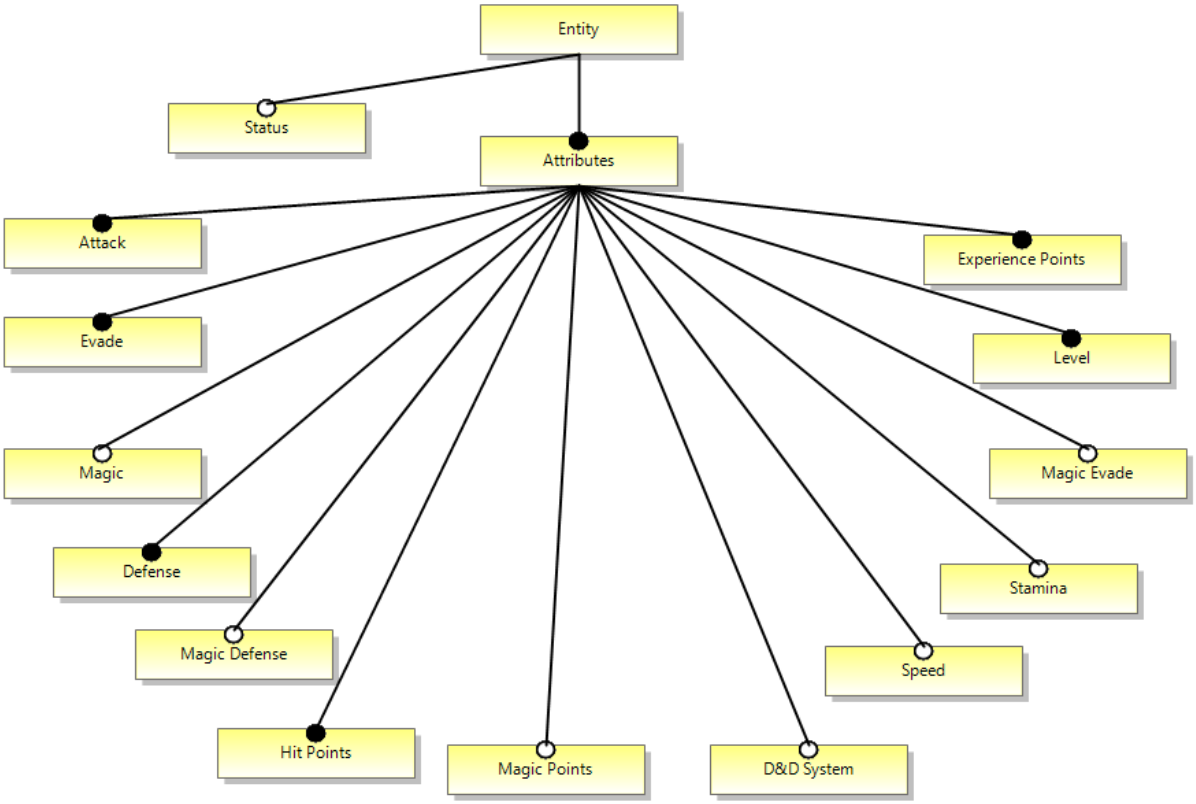


Figura 17 – Feature Model para as entidades do jogo.

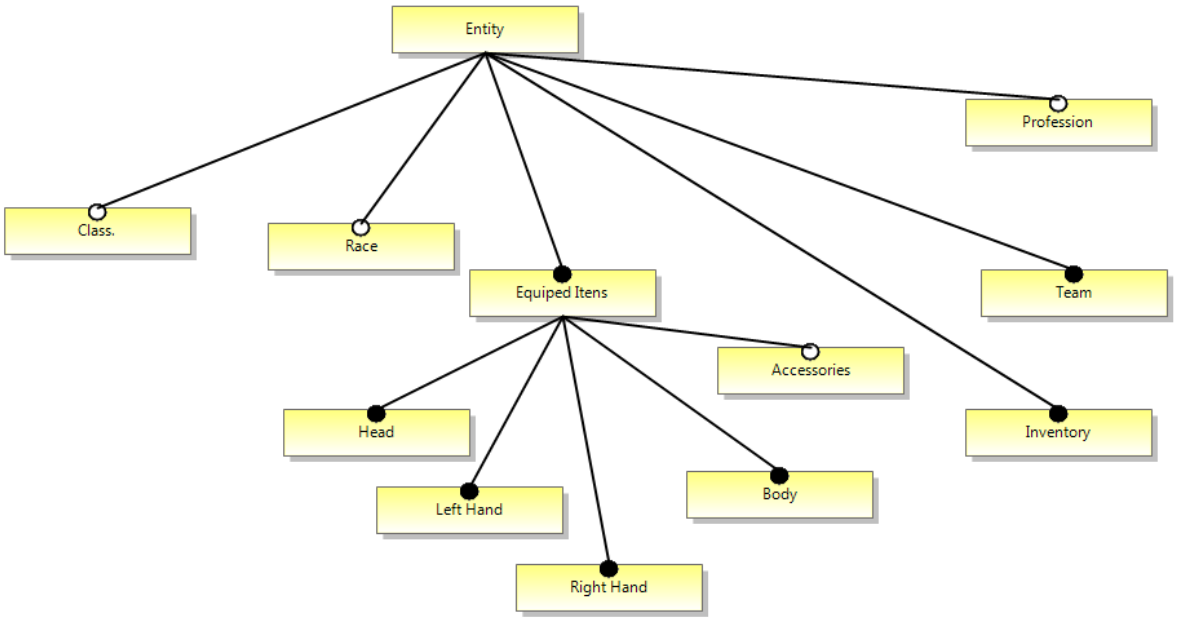


Figura 18 – Feature Model para as entidades do jogo.

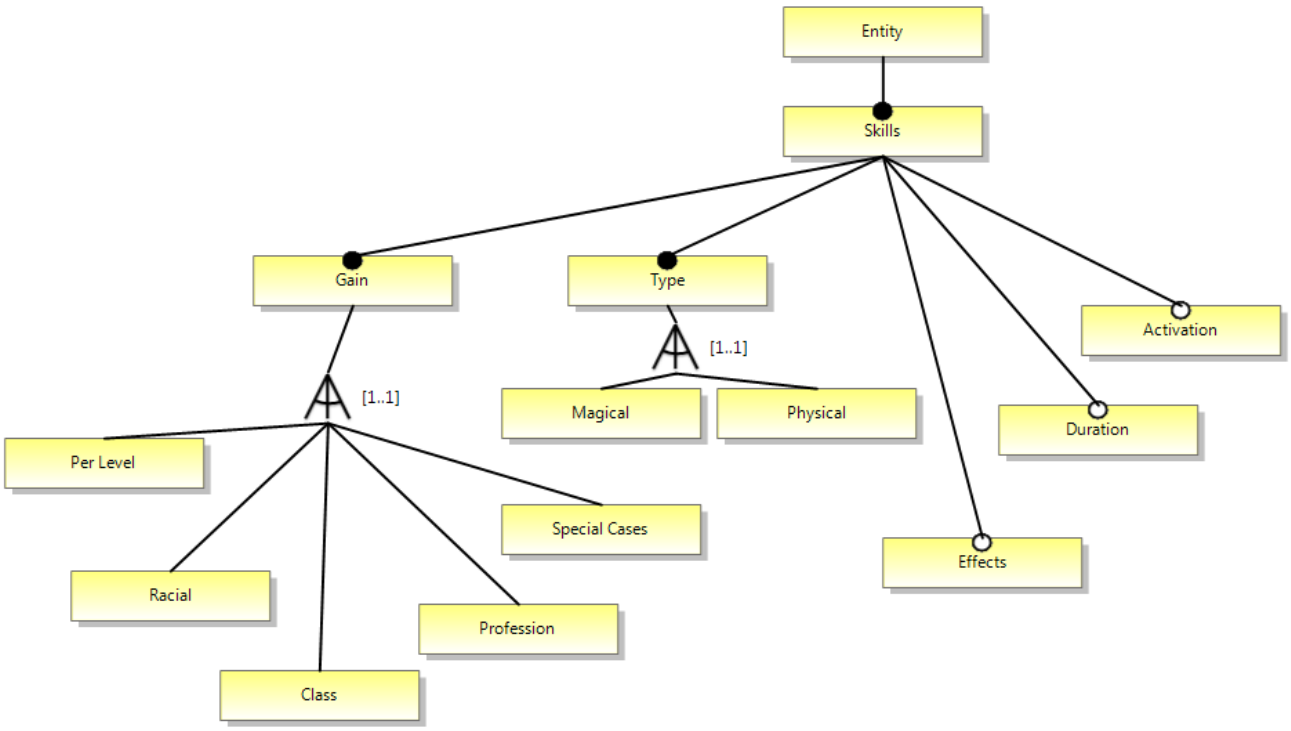


Figura 19 – Feature Model para as entidades do jogo.

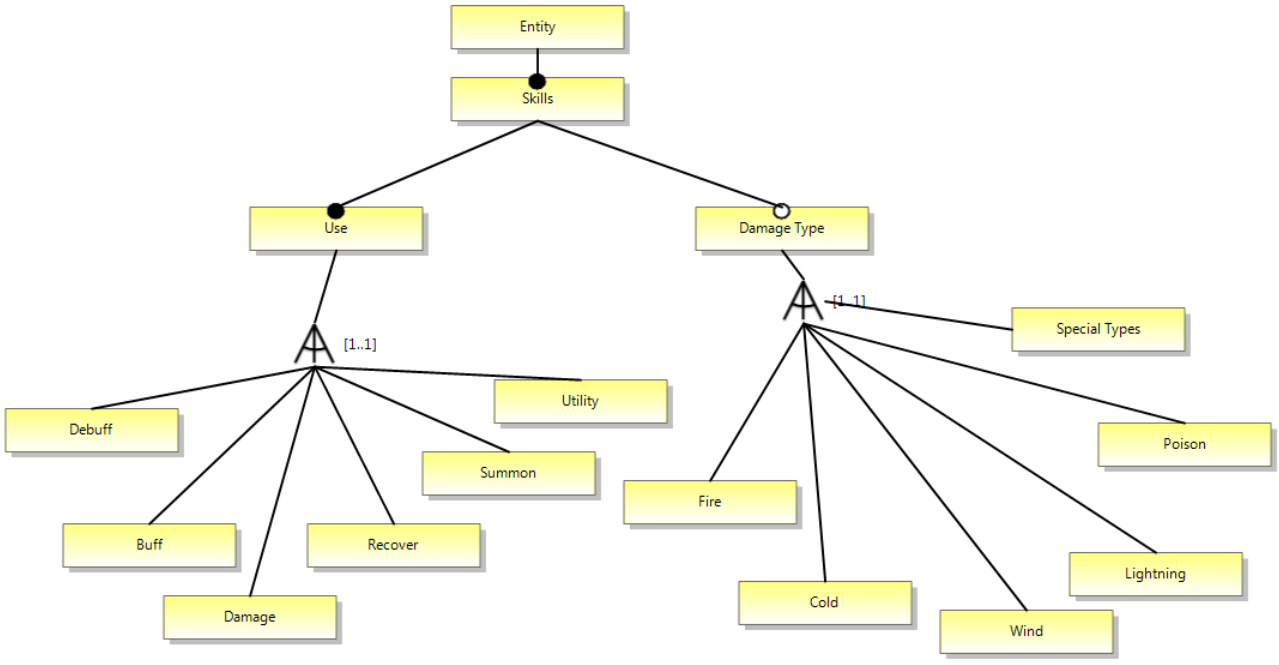


Figura 20 – Feature Model para as entidades do jogo.

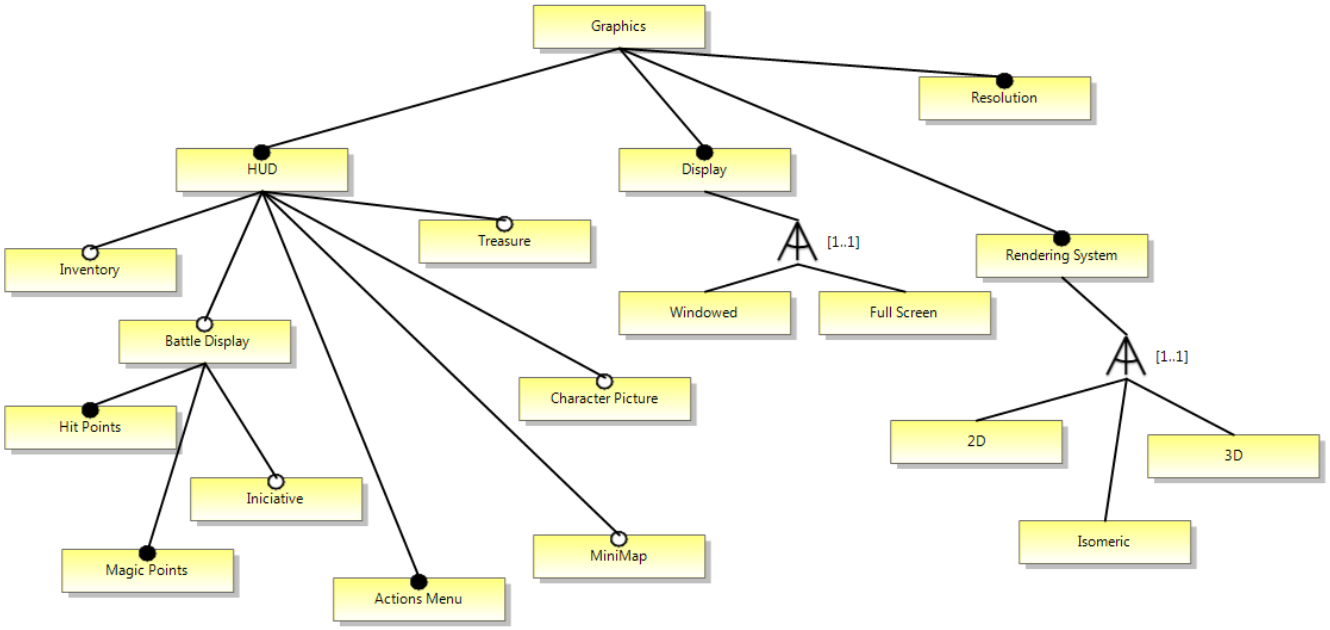


Figura 21 – Feature Model para o sistema gráfico.

4.6. Validação do domínio

O domínio foi validado pelo jogo *Persona 3*, da série *Megami Tensei*, lançado em 2006 pela *Atlus* para a plataforma *Playstation 2*.

4.7. Critério de Parada

O tempo foi o critério levado mais fortemente em conta para a finalização da análise. Como os jogos RPG costumam ser bastante complexos, o número de funcionalidades mapeadas no modelo certamente representa apenas um subconjunto, ainda que bastante representativo e generalizado. Desta forma termina o processo de análise de domínio e funcionalidades, passando agora para o momento de implementação da fábrica.

5. Elegy Game Factory

Finalizada a aplicação da metodologia, chegamos à etapa de modelagem da DSL gráfica. Os *feature models* gerados e a *engine RPG Starter Kit* foram utilizados para guiar a modelagem, através da combinação de abordagens *top-down e bottom-up*.

A abordagem *top-down* abrange toda aplicação da metodologia sobre o domínio, resultando nos *feature models*, ao mesmo tempo em que é feito um estudo em cima da *engine* a ser reusada, no sentido *bottom-up*. O resultado da aplicação destas duas abordagens em paralelo culmina na modelagem da linguagem visual, juntamente com o código a ser gerado. Desta forma uma camada de abstração surge de modo a facilitar a criação de jogos através do uso de elementos do domínio, como personagens, mapas, inimigos, *quests,* entre outros, sem a necessidade de programação em linhas de código.

Neste capítulo será detalhada toda a construção da linguagem visual da fábrica de jogos *Elegy*. O resultado do estudo sobre a *engine* escolhida também será mostrado. Antes será feita uma breve abordagem em relação às DSLs gráficas e as ferramentas utilizadas para modelagem.

5.1. DSLs Gráficas

Uma DSL gráfica possui diversos aspectos importantes que devem ser definidos. Os mais importantes são notação, modelo do domínio, geração, serialização e integração com a ferramenta (Cook, Jones, Stuart, & Wills, 2007). O *framework Microsoft Domain-Specific Language Tools* oferece suporte para todos estes aspectos de maneira prática e intuitiva. Abaixo estão explicações breves de cada aspecto.

Notação: a forma como a linguagem é representada. Utilizam componentes similares a UML. Blocos, formas e conectores são associados numa superfície de desenho bidimensional, de forma a exibir relações entre classes do domínio. As formas e conectores possuem decoradores que mostram informações adicionais como texto e ícones.

Modelo do domínio: é um modelo dos conceitos descritos pela linguagem. O modelo do domínio para uma linguagem gráfica faz um papel

similar na sua definição ao da gramática BNF para linguagens textuais. Mas para linguagens gráficas, o modelo do domínio é usualmente representado graficamente. Os componentes básicos para este modelo são as classes do domínio (*domain classes*) e os relacionamentos do domínio (*domain relationships*). Cada classe representa um conceito do domínio, e cada relacionamento do domínio representa o relacionamento entre os conceitos do domínio. Outro aspecto importante é a definição de restrições, que checam se os diagramas criados usando a linguagem são válidos.

Geração: após criar modelos usando a linguagem, normalmente é desejável gerar alguns artefatos como código, dados, arquivos de configuração, outros diagramas, ou até mesmo a combinação de tudo isto.

Serialização: após criar o modelo é desejável salvá-lo para carregá-lo posteriormente. A informação salva inclui detalhes sobre formas e conectores, onde eles estão posicionados, de quais cores eles são, e também de detalhes dos conceitos representados por estas formas. O *DSL Tools* utiliza o formato XML, o que aumenta a flexibilidade e interoperabilidade entre ferramentas.

Integração com a ferramenta: determina como o projeto de DSL aparece no ambiente do *Visual Studio*. Isto envolve o tipo de extensão associada à linguagem, quais janelas aparecem e qual é o escopo da informação representada quando um arquivo da linguagem é aberto, se a linguagem possui uma árvore estruturada de exploração e como seus nós são representados, quais propriedades dos itens selecionados aparecem no navegador de propriedades, editores customizados da linguagem, entre outras questões.

5.2. Modelagem da DSL gráfica

Como já abordado neste documento, é importante manter o foco expressivo das DSLs, não sendo adequado o mapeamento do domínio em uma única DSL. Desta forma três DSLs foram modeladas separadamente, mas como pontos de integração entre si. O escopo deste projeto não abrange a integração entre as DSLs, devido a seu caráter complexo diante da pouca maturidade em relação a este tipo de projeto e tempo restrito. Portanto os

pontos de integração serão sempre apontados no momento do detalhamento de cada DSL produzida.

As DSLs foram definidas levando em consideração primeiramente os *feauture models* obtidos na análise. Devido ao seu caráter muito amplo, o pouco tempo e até mesmo a necessidade de foco, não foi possível mapear todas as funcionalidades presentes no modelo. Foram escolhidas funcionalidades que são consideradas importantes em jogos do domínio, como definição de mapas, personagens e *quests.*

Após esta primeira modelagem alterações e adaptações foram feitas baseadas na *engine* escolhida, o *RPG Starter Kit*. Esta *engine* gerencia a criação de mapas, personagens, *quests*, entre outros aspectos próprios de jogos RPG através de arquivos XML. Desta forma, estes arquivos foram estudados e alterados de maneira a entender seu funcionamento dentro da *engine*. Assim foi possível definir como as DSLs poderiam contribuir para agilizar o processo de criação dos arquivos XML de maneira segura e intuitiva para o usuário, desta forma impactando no aumento de produtividade na criação de jogos utilizando o *RPG Starter Kit*.

A próxima seção fará uma abordagem sobre como as funcionalidades escolhidas são definidas na *engine.*

5.3. RPG Starter Kit

Como já dito anteriormente, o gerenciamento de mapas, personagens e *quests*, que são as funcionalidades foco deste projeto, é feito através de arquivos XML separados, que devem ser definidos manualmente.

Abaixo está um exemplo de arquivo de definição de mapa.

```
<?xml version="1.0" encoding="utf-8" ?>
<XnaContent>
  <Asset Type="RolePlayingGameData.Map">
      <Name>Map002</Name>
      <MapDimensions>23 25</MapDimensions>
      <TileSize>64 64</TileSize>
      <SpawnMapPosition>11 2</SpawnMapPosition>
      <TextureName>ForestTiles</TextureName>
      <CombatTextureName>CombatBkgdVillage</CombatTextureName>
      <MusicCueName>BeachTheme</MusicCueName>
      <CombatMusicCueName>BattleTheme</CombatMusicCueName>
      <BaseLayer>
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
          7   4   4   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   7   4   4   7
          41  41  41  16  41  41  41  41  41  22  16  16  16  22  41  41  41  41  41  16  41  41  41
          41  41  16  16  16  22  41  41  41  22  16  16  16  22  41  41  41  22  16  16  16  41  41
          41  41  16  16  16  36  22  36  22  36  35  16  35  36  22  36  22  22  16  16  16  41  41
          41  41  41  16  41  44  49  49  49  49  49  49  49  49  49  49  49  44  41  16  41  41  41
          41  41  41  41  41  54  50  50  50  50  50  50  50  50  50  50  50  53  41  41  41  41  41
          41  41  41  41  41  42  75  77  77  77  77  95  77  77  77  77  74  40  41  41  41  41  41
          49  49  49  49  49  51  79  95  95  95  95  95  95  95  95  95  78  47  49  49  49  49  49
          50  50  50  50  50  52  79  95  95  95  95  95  95  95  95  95  78  48  50  50  50  50  50
          77  77  77  77  77  77  99  95  95  95  95  95  95  95  95  95  98  77  77  77  77  77  77
          95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95
          95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95  95
      </BaseLayer>
     -1 149 150  -1  -1 149 150  -1  -1  -1  -1  -1  -1  -1  -1  -1 149 150  -1  -1 149 150  -1
     150  -1  -1 149 150  -1  -1 149 150  -1  -1  -1  -1  -1 149 150  -1  -1 149 150  -1  -1 149
     -1 149 150  -1  -1 149 150  -1  -1  -1  -1  -1  -1  -1  -1  -1 149 150  -1  -1 149 150  -1
     150  -1  -1 149 150  -1  -1 149 150  -1  -1  -1  -1  -1 149 150  -1  -1 149 150  -1  -1 149
     -1 149 150  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1 149 150  -1  -1 149 150  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
  </ObjectLayer>
```

```
<CollisionLayer>
    0   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   0
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    1   1   1   1   1   0   0   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1
    0   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0   1   1   0
    0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
    0   1   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   1   0
    0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
    0   1   0   0   0   1   1   1   1   1   1   0   1   1   1   1   1   1   0   0   0   1   0
    0   1   0   1   0   1   1   1   1   1   1   0   1   1   1   1   1   1   0   1   0   1   0
    0   1   0   0   0   1   0   0   0   0   1   0   1   0   0   0   0   1   0   0   0   1   0
    1   1   1   1   1   1   0   0   0   0   1   0   1   0   0   0   0   1   1   1   1   1   1
    1   1   1   1   1   1   0   0   0   0   1   0   1   0   0   0   0   1   1   1   1   1   1
    0   0   0   0   0   0   0   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0
</CollisionLayer>
<Portals>
    <Item>
        <Name>P1</Name>
        <LandingMapPosition>9 2</LandingMapPosition>
        <DestinationMapContentName>Map001</DestinationMapContentName>
        <DestinationMapPortalName>P1</DestinationMapPortalName>
    </Item>
    <Item>
        <Name>P2</Name>
        <LandingMapPosition>10 2</LandingMapPosition>
        <DestinationMapContentName>Map001</DestinationMapContentName>
        <DestinationMapPortalName>P1</DestinationMapPortalName>
    </Item>
    <Item>
        <Name>P3</Name>
        <LandingMapPosition>11 2</LandingMapPosition>
        <DestinationMapContentName>Map001</DestinationMapContentName>
        <DestinationMapPortalName>P2</DestinationMapPortalName>
    </Item>
    <Item>
        <Name>P4</Name>
        <LandingMapPosition>12 2</LandingMapPosition>
        <DestinationMapContentName>Map001</DestinationMapContentName>
        <DestinationMapPortalName>P3</DestinationMapPortalName>
    </Item>
    <Item>
        <Name>P5</Name>
        <LandingMapPosition>13 2</LandingMapPosition>
        <DestinationMapContentName>Map001</DestinationMapContentName>
        <DestinationMapPortalName>P4</DestinationMapPortalName>
    </Item>
    <Item>
        <Name>P6</Name>
        <LandingMapPosition>11 22</LandingMapPosition>
        <DestinationMapContentName>Map003</DestinationMapContentName>
        <DestinationMapPortalName>P1</DestinationMapPortalName>
    </Item>
</Portals>
```

```
<PortalEntries>
    <Item>
        <ContentName>P1</ContentName>
        <MapPosition>9 0</MapPosition>
    </Item>
    <Item>
        <ContentName>P2</ContentName>
        <MapPosition>10 0</MapPosition>
    </Item>
    <Item>
        <ContentName>P3</ContentName>
        <MapPosition>11 0</MapPosition>
    </Item>
    <Item>
        <ContentName>P4</ContentName>
        <MapPosition>12 0</MapPosition>
    </Item>
    <Item>
        <ContentName>P5</ContentName>
        <MapPosition>13 0</MapPosition>
    </Item>
    <Item>
        <ContentName>P6</ContentName>
        <MapPosition>11 24</MapPosition>
    </Item>
</PortalEntries>
```

```
<ChestEntries />
<FixedCombatEntries />
<RandomCombat>
    <CombatProbability>0</CombatProbability>
    <FleeProbability>50</FleeProbability>
    <MonsterCountRange>
        <Minimum>1</Minimum>
        <Maximum>3</Maximum>
    </MonsterCountRange>
    <Entries>
        <Item>
            <ContentName>GoblinGrunt</ContentName>
            <Count>1</Count>
            <Weight>100</Weight>
        </Item>
    </Entries>
</RandomCombat>
<QuestNpcEntries>
    <Item>
        <ContentName>JangElder</ContentName>
        <MapPosition>2 14</MapPosition>
        <Direction>SouthEast</Direction>
    </Item>
    <Item>
        <ContentName>Ujar</ContentName>
        <MapPosition>20 14</MapPosition>
        <Direction>SouthWest</Direction>
    </Item>
</QuestNpcEntries>
```

```
    <PlayerNpcEntries>
      <Item>
        <ContentName>Nykkas</ContentName>
        <MapPosition>11 16</MapPosition>
        <Direction>North</Direction>
      </Item>
    </PlayerNpcEntries>
    <InnEntries />
    <StoreEntries />
  </Asset>
</XnaContent>
```

Figura 22 – Arquivo XML para criação de um mapa na engine.

A definição do *background* do mapa é feito por várias camadas, que no arquivo são representadas por uma série de números organizados numa matriz que indexam os *tiles*. Uma matriz também define os locais onde deverá haver colisões. Os pontos de transição, definidos como portais, especificam onde será realizado o fluxo de um mapa para outro. Um ponto interessante nestes arquivos de definição de mapas é que eles possuem relações com elementos que deverão ser definidos por outros arquivos XML, como *quests* que ocorrerão no determinado mapa, os NPCs responsáveis por estas *quests*, itens de *quests* que estão localizados no mapa, dentre outros, o que torna bastante confuso visualizar e gerenciar em todo o projeto do jogo estes relacionamentos. Além disso, ter que alterar manualmente cada um desses arquivos é uma tarefa por vezes enfadonha e lenta.

A definição de personagens e *quests* também são feitas através de arquivos XML, ainda que menos complexos que os de mapas, possuem também relação com outros elementos, tornando a criação desses arquivos uma tarefa igualmente cansativa e que não contribui para uma maior produtividade.

Nas próximas seções serão apresentadas as DSLs implementadas, atendo-se aos aspectos fundamentais em DSLs gráficas: modelo, notação, geração e integração. Para todos os casos a serialização é a padrão de *DSL Tools*, definidas por arquivos XML.

5.4. Map Manager DSL

Esta DSL é responsável por gerenciar a criação de mapas e a transição entre eles.

5.4.1. Modelo do Domínio

A idéia principal para a modelagem da *Map Manager DSL* foi baseada na análise de funcionalidades, adaptando-se depois às necessidades da *engine*. Abaixo está o modelo que apresenta as classes do domínio e seus relacionamentos, seguido de um detalhamento destes elementos.

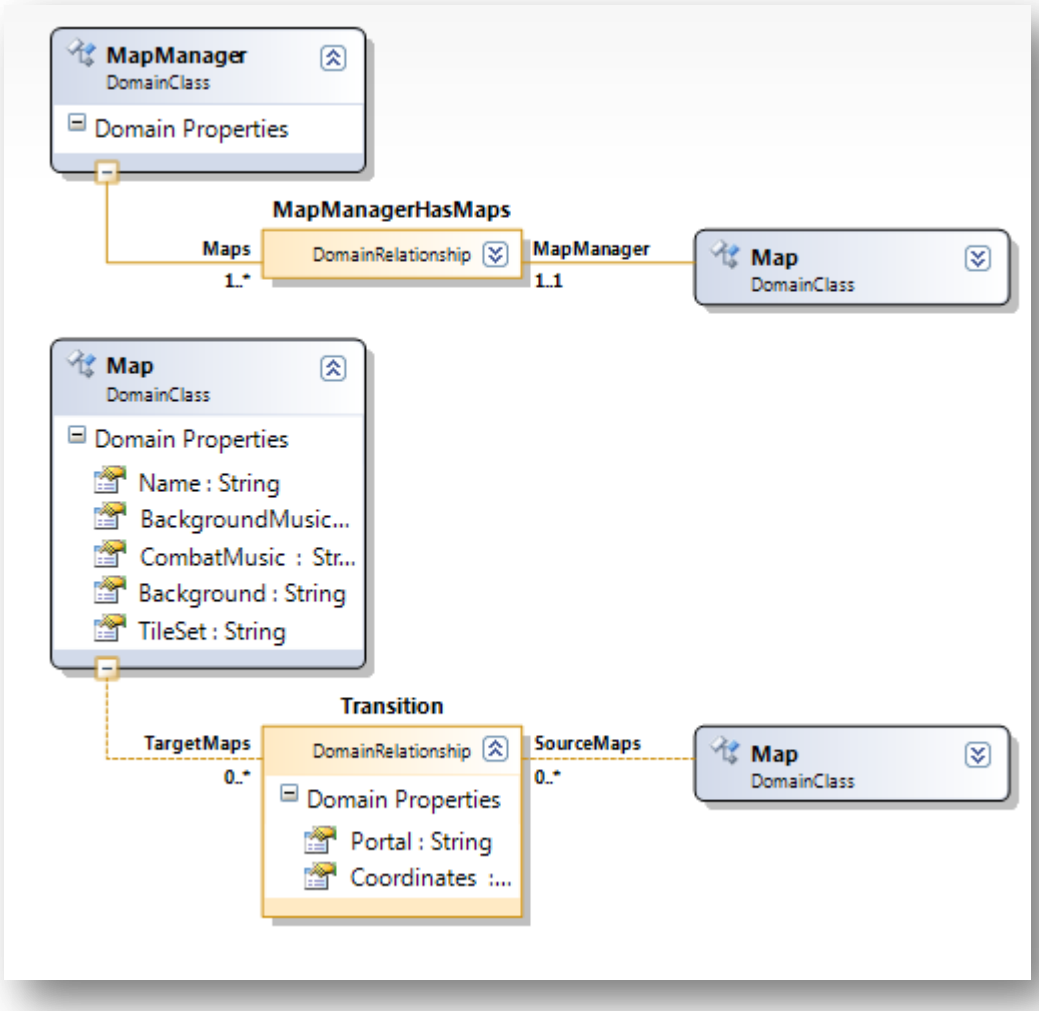


Figura 23 – Classes e relacionamentos do domínio.

MapManager: esta é a classe raiz do modelo, que reúne todos os mapas criados.

Map: a classe que define o mapa criado. Um mapa possui um nome, uma música de fundo, um *background,* um conjunto de tiles que irão formar *background* e uma música de fundo para combates que acontecem neste mapa.

Transition: este relacionamento do domínio define a maneira como é feita a transição de um mapa para o outro pelo personagem principal. Possui como propriedades portal, que indica o ponto de transição, e suas coordenadas no mapa.

O impacto do estudo da *engine* nesta modelagem se deu no modo como as camadas de *tiles* são organizadas e como a transição é realizada. Estes detalhes foram abstraídos ainda mais para o usuário. As relações com elementos como *quests*, combates, baús, entre outros, não foi realizado nesta versão.

5.4.2. Notação

Para representar a linguagem ao usuário é necessária a definição de um diagrama, que irá conter todos os elementos visuais. As classes que necessitam de representação visual serão mapeadas em formas, e os relacionamentos em conectores.

Neste modelo apenas a classe mapa possui necessidade de representação visual, pois o usuário poderá adicionar vários mapas no seu projeto e conectá-los com a representação visual do relacionamento *Transition*.

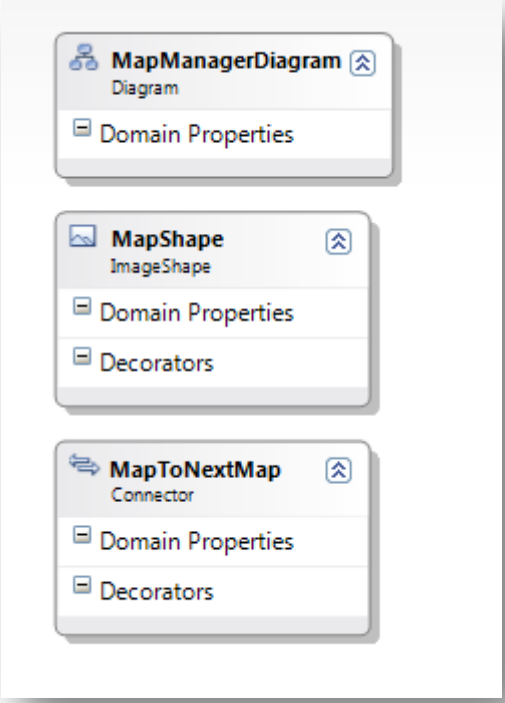


Figura 24 – Definição da notação da MapManagerDSL.

O *MapManagerDiagram* possui como classe raiz *MapManager*, e *a* classe *Map* é mapeada na forma *MapShape*, que será representado apenas por uma imagem. O relacionamento *Transition* é representado pelo conector *MapToNextMap.*

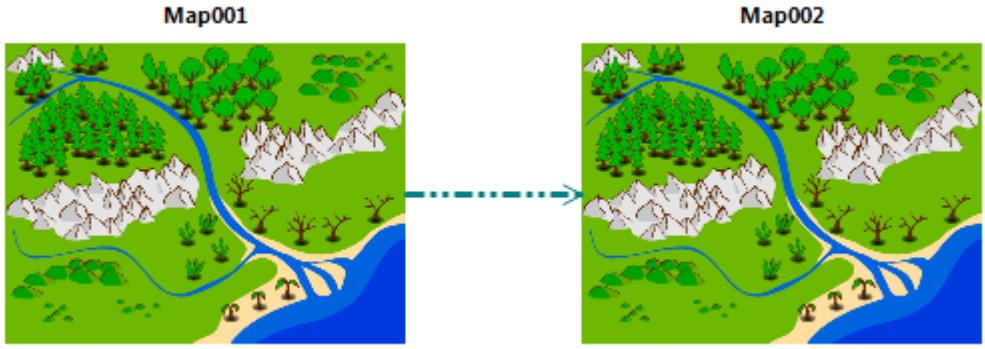


Figura 25 – Representação Visual dos mapas e transições.

O *DSL Tools* também permite regras de validação para o projeto a ser gerado. Neste caso, por exemplo, foi definido que no caso de existirem dois mapas criados é necessária a presença de transições para o modelo ser válido, caso contrário a ferramenta mostrará um erro. É importante definir erros de

validação para que o código gerado esteja em perfeito estado para ser consumido pela *engine*.

5.4.3. Integração

A integração à ferramenta, no caso *Visual Studio 2008,* determina a forma como o ambiente será apresentado para a criação de arquivos do projeto. A extensão para os arquivos de gerenciamento de mapas foi definida como ".map". Também é necessário determinar quais elementos aparecerão na *toolbox*, de que modo será exibida a árvore de visualização e os navegadores de propriedades, e se existirão editores customizados.

Para o *MapManagerDSL* a *toolbox* apresenta a opção de criação de mapas e uso de transições. Esses elementos podem ser escolhidos e arrastados para o diagrama. Para facilitar a alteração das propriedades foi criado um editor *Elegy Map Creator*, onde o usuário monta todo o mapa com os *tiles* desejados e define locais de colisão e transição.

Abaixo está a figura que mostra como ficou a integração com o *VisualStudio*. Na esquerda encontra-se a *toolbox*, no centro o diagrama para criação de mapas e transições entre eles, abaixo a caixa de erros e na direita a janela de propriedades.

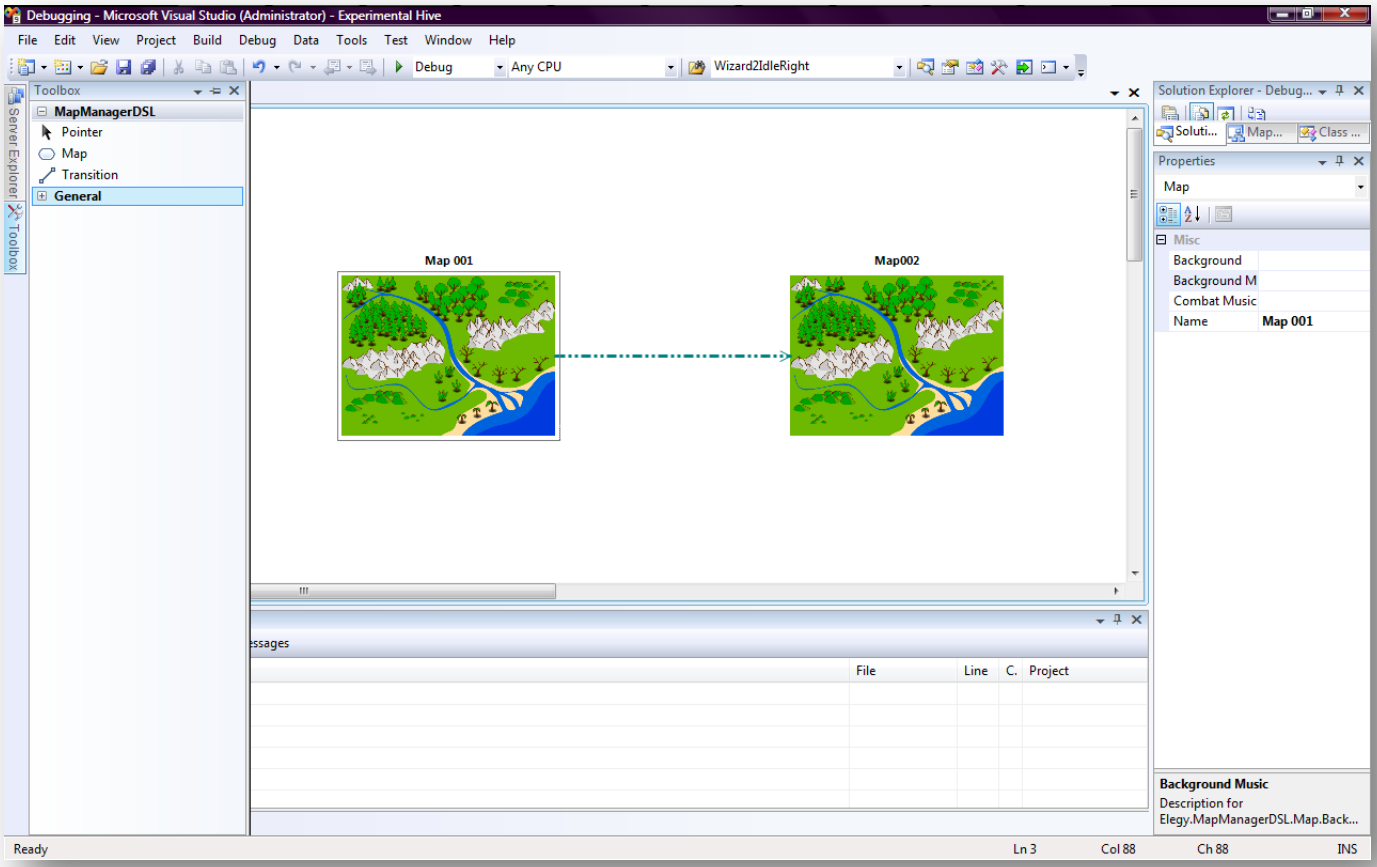


Figura 26 – MapManager integrado ao Visual Studio 2008.

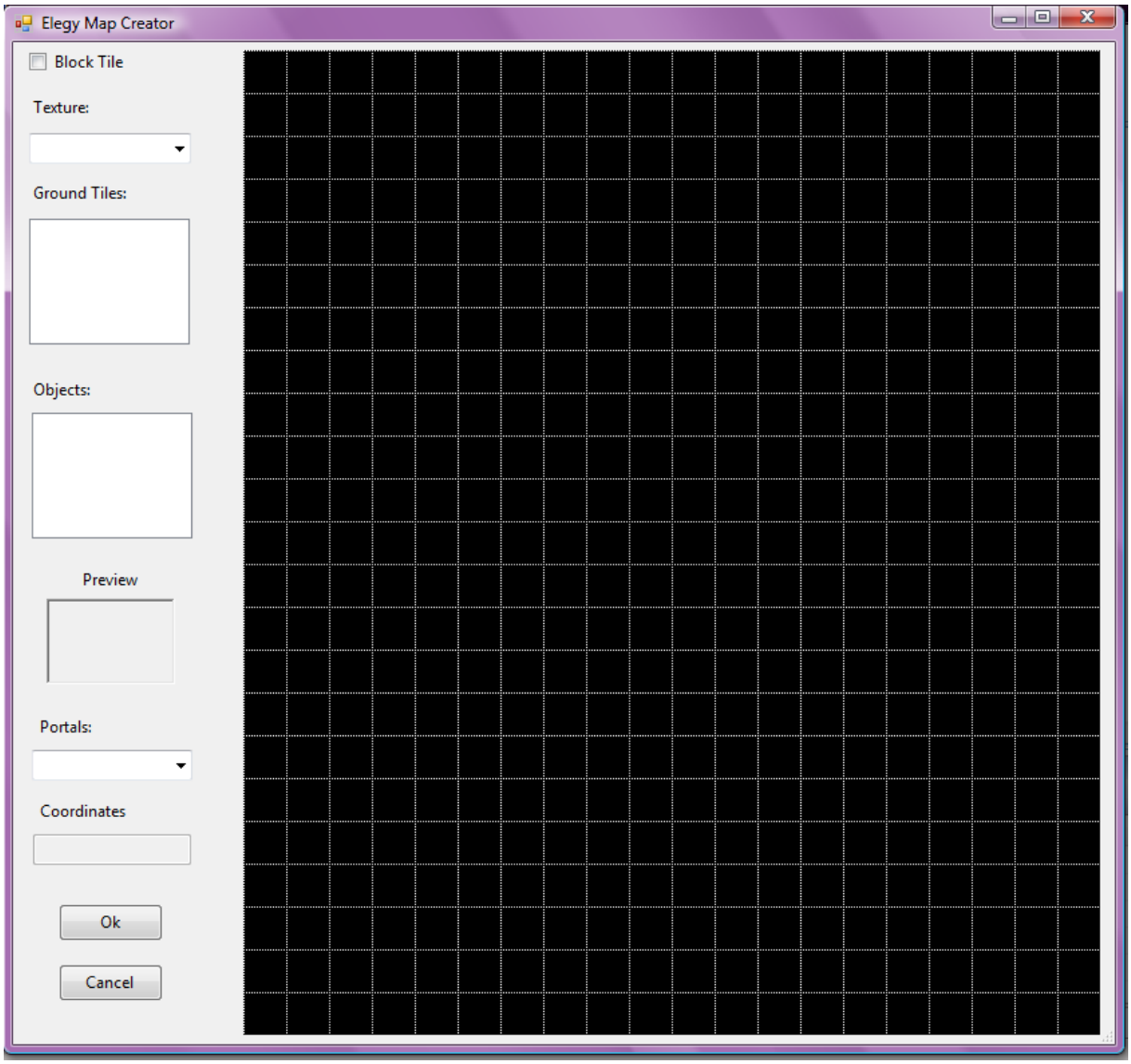


Figura 27 – Editor customizado de propriedades do mapa.

5.4.4. Geração de Código

O código gerado pelo *MapManagerDSL* será consumido pela *engine*, e portanto está no formato XML, igualmente ao apresentado na Figura 21. O código gerado é suportado pelo XNA *Game Studio*. Como é gerado apenas um arquivo para todos os mapas criados e a *engine* aceita arquivos separados, é necessário que este arquivo passe por um pré-processamento que irá separá-lo em arquivos distintos para cada mapa.

5.5. Quest Definiton DSL

Esta DSL é responsável pela criação de *quests.*

5.5.1. Modelo do Domínio

A modelagem desta DSL teve como base a análise de funcionalidades e o código a ser consumido pela *engine*. O modelo, com suas classes e relacionamentos, está representado abaixo.

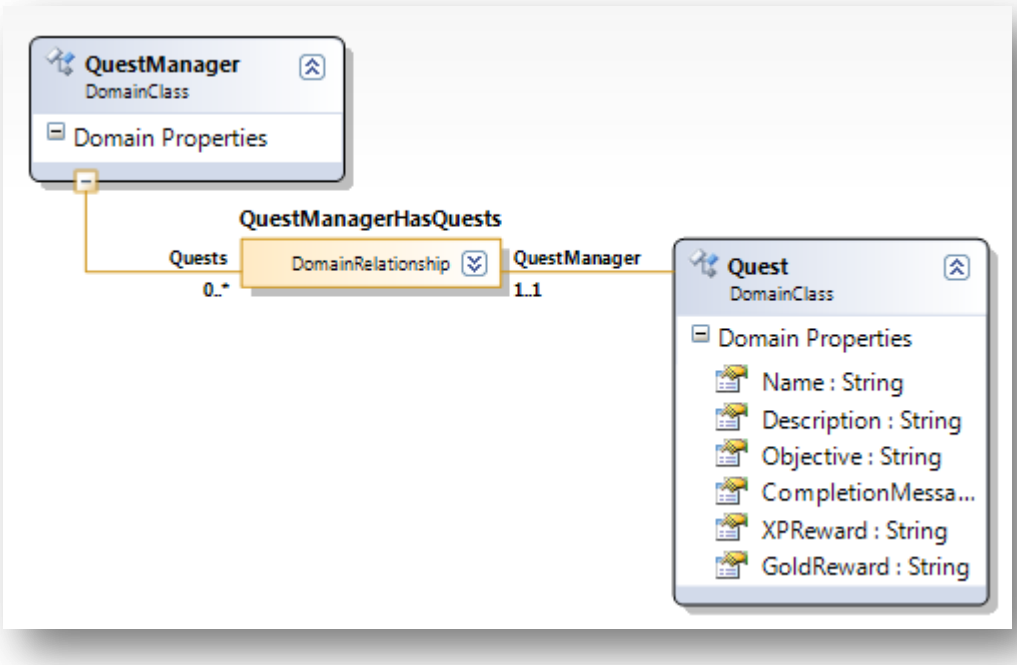


Figura 28 – Classes e relacionamentos do domínio.

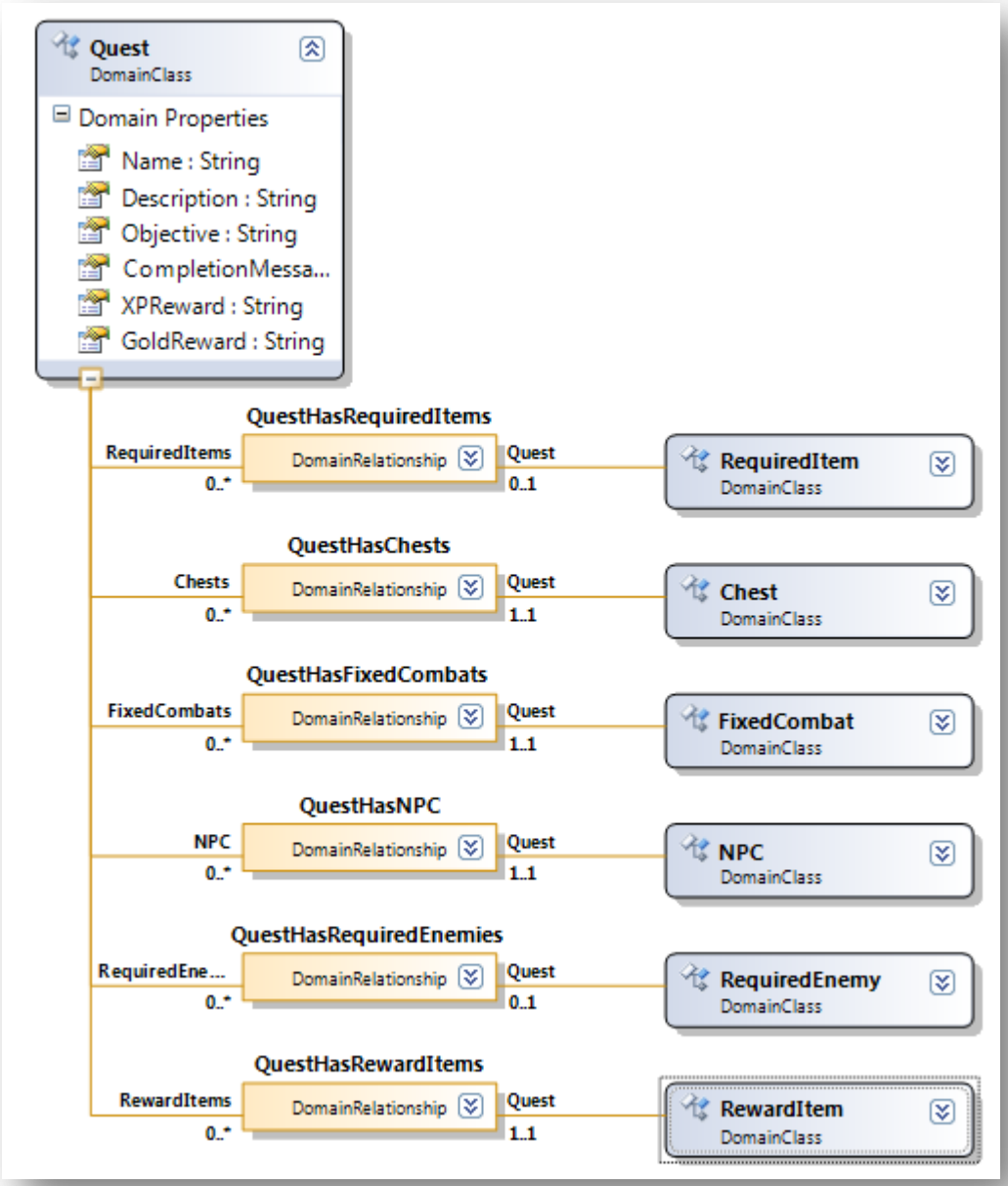


Figura 29 – Classes e relacionamentos do domínio.

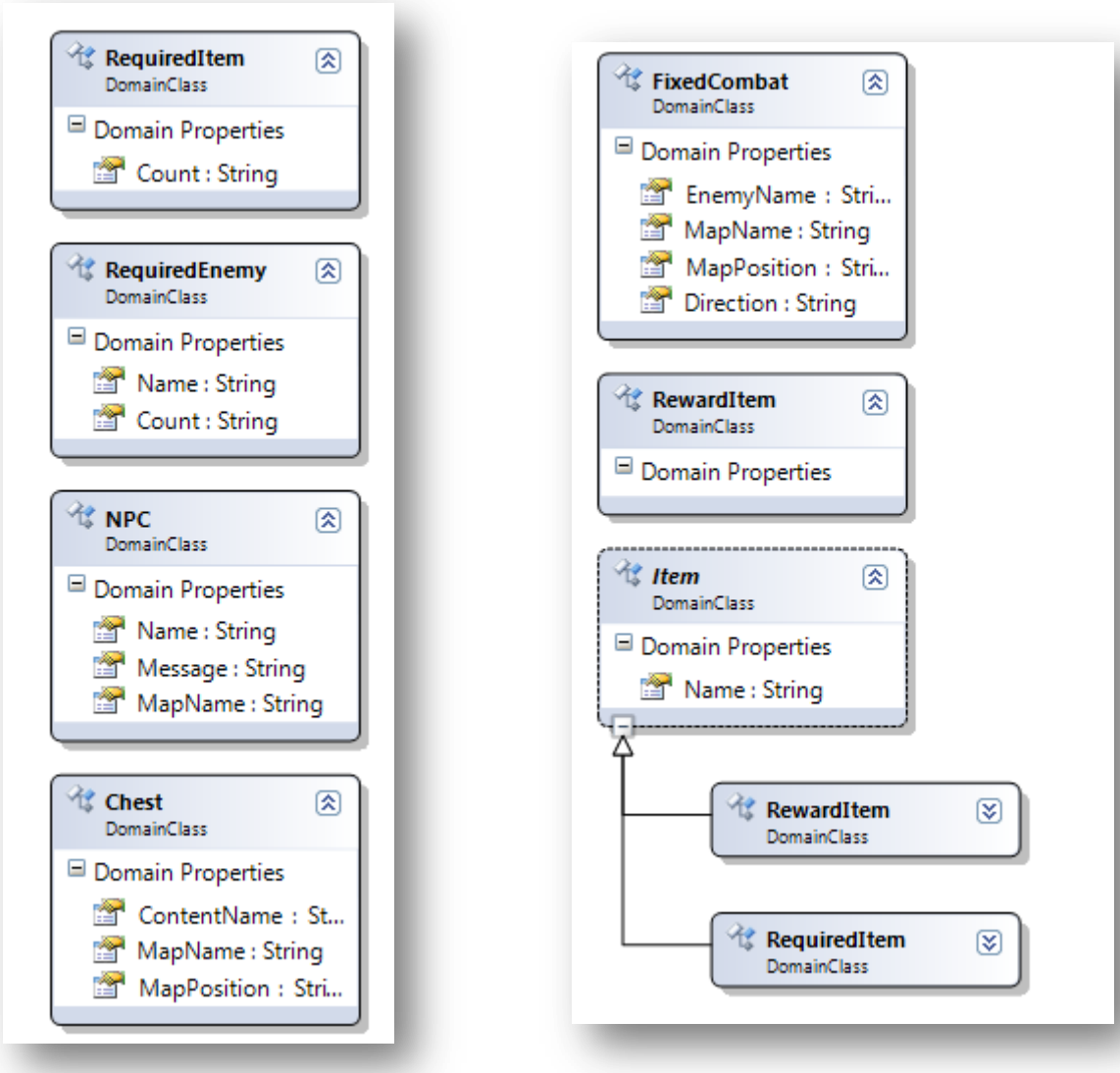


Figura 30 – Classes do domínio.

QuestManager: classe raiz do modelo. Agrupa todas as *quests* criadas.

Quest: classe que define uma *quest*. Possui como propriedades nome, descrição, objetivo, mensagem de finalização da *quest*, pontos de experiência e peças de ouro ganhos ao se completar a *quest.* Possui relacionamentos com as classes NPC, *RequiredItem, RequiredEnemy, RewardItem, Chest* e *FixedCombat.*

NPC: classe que representa o NPC que irá oferecer a *quest* ao personagem principal. Possui um nome, uma mensagem de apresentação e referência ao mapa onde ele se encontra. Esta classe é um ponto de integração entre as DSLs. A DSL de definição de entidades produz o NPC que

será utilizado nesta DSL, e a de gerenciamento de mapas possui a referência para o NPC que está localizado no mapa criado. Como a integração de DSLs não fará parte do escopo deste projeto, alguns artifícios foram utilizados para resolver os relacionamentos entre as DSLs. Esses artifícios serão mostrados adiante.

Item: classe abstrata que define os itens de uma *quest*.

RequiredItem: classe que define o item requerido pelo NPC para se completar a *quest*. Possui também um ponto de integração entre DSLs, já que os itens na *engine* também são criados através de arquivos XML a parte. Nesta versão uma DSL para criação de itens não foi implementada.

RewardItem: classe que define o item recebido ao se completar uma quest.

RequiredEnemy: classe que define o inimigo a ser combatido para se completar a *quest*. Possui como propriedades o nome do inimigo e a quantidade. Como um inimigo é um artefato produzido pela DSL de definição de entidades esta classe também é um ponto de integração entre as DSLs.

FixedCombat: esta classe foi adicionada por necessidade da *engine*. Um *fixed combat* representa um único *sprite* atrelado a um inimigo ou aum grupo de inimigos. Possui também pontos de integração tanto com a DSL de mapas como a de definição de personagens.

Chest: esta classe também foi adicionada devido aos estudos sobre *engine*. O item requerido pode ser o conteúdo de um baú. Possui pontos de integração com a DSL de mapas e itens.

5.5.2. Notação

Neste modelo apenas a classe *quest* possui uma representação visual. Para isto foi utilizada uma forma específica que possui compartimentos que agrupam as classes com as quais *quest* possui relacionamentos.

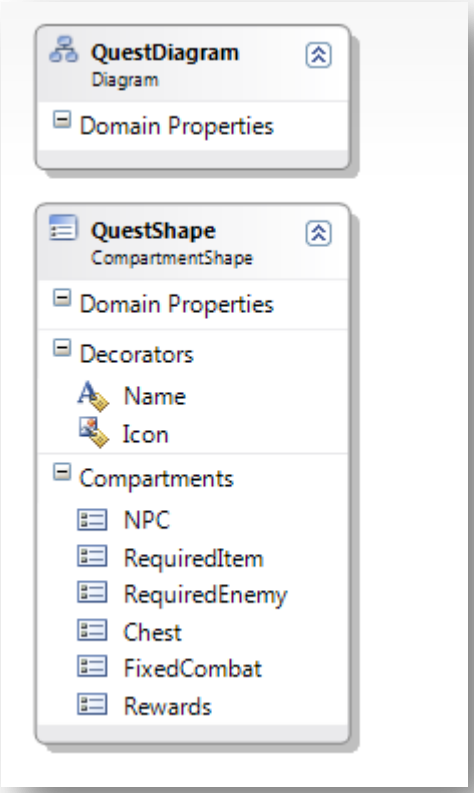


Figura 31 – Definição da notação da QuestDefinitionDSL.

A *QuestShape* é a representação da classe *quest* e possui no campo de compartimentos as demais classes com as quais exerce relacionamentos. Possui decoradores para o nome da *quest* e um ícone. A figura abaixo mostra como uma *quest* é representada visualmente no diagrama.

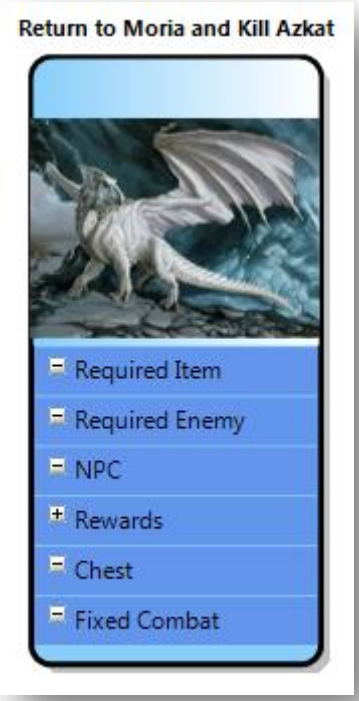


Figura 32 – Representação visual de uma quest.

5.5.3. Integração

O formato para os arquivos criados com a *QuestDefinitionDSL* foi denominado de ".quest". A *toolbox* possui a opção de criação de *quests,* e editores customizados foram criados para facilitar a alteração de propriedades.

Abaixo está a figura que mostra como ficou a integração com o *VisualStudio*. Na esquerda encontra-se a *toolbox*, no centro o diagrama para criação de *quests*, abaixo a caixa de erros e na direita a janela de propriedades.

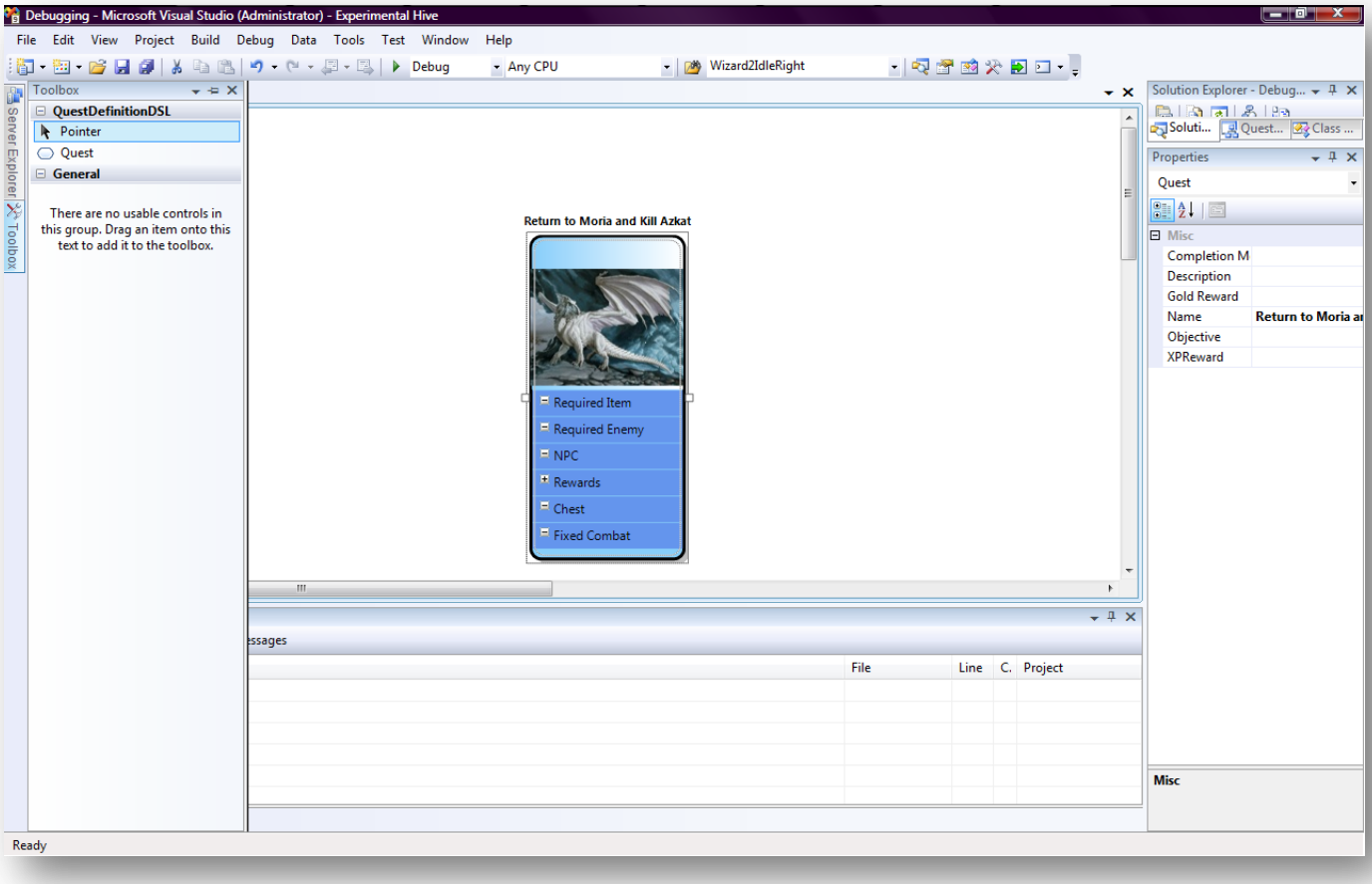


Figura 33 – QuestDefinition integrado ao Visual Studio 2008.

As figuras seguintes mostram os editores customizados criados.

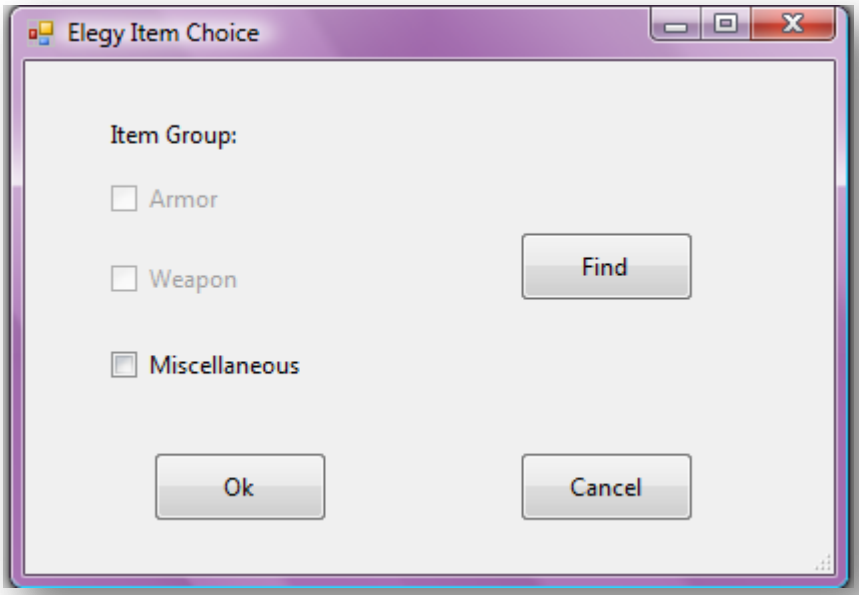


Figura 34 – Editor para a escolha de itens, tanto para RequiredItems quanto para RewardItems.

Este editor é um dos artifícios para resolver o problema da falta de integração entre as DSLs nesta versão. O usuário tem a opção de procurar no diretório que possui todos os arquivos XML de criação de itens o item desejado. Desta forma apenas itens já criados poderão ser adicionados a uma *quest.*

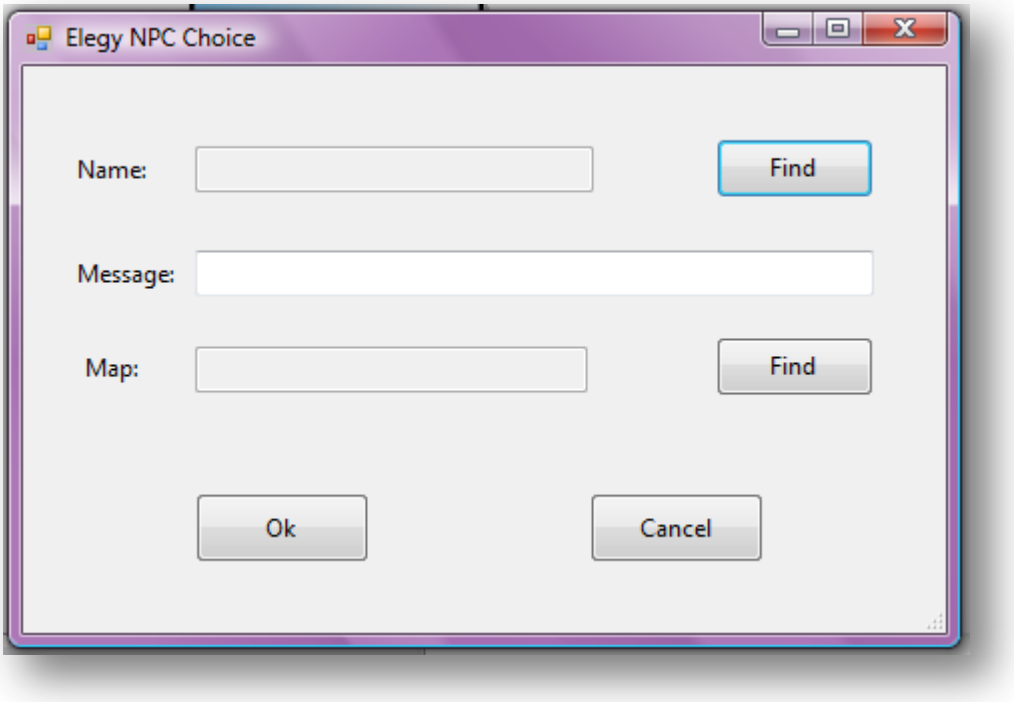


Figura 35 – Editor para a escolha do NPC.

Através deste editor o usuário escolhe um NPC já criado no diretório de personagens NPC. Da mesma forma o mapa onde ele deve estar situado é escolhido no diretório de mapas. Essas duas buscas também são artifícios para a falta de integração de DSLs.

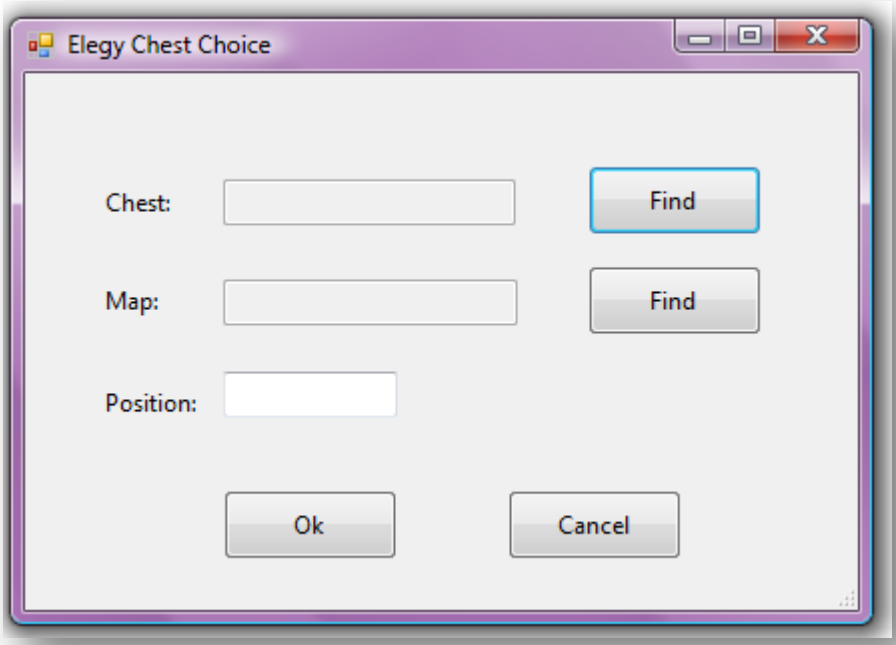


Figura 36 – Editor para adição de chest.

Em mais um ponto de integração, o usuário tem a opção de escolher dentro do diretório de itens qual será o conteúdo do baú. Da mesma forma irá escolher o mapa onde ele será adicionado.

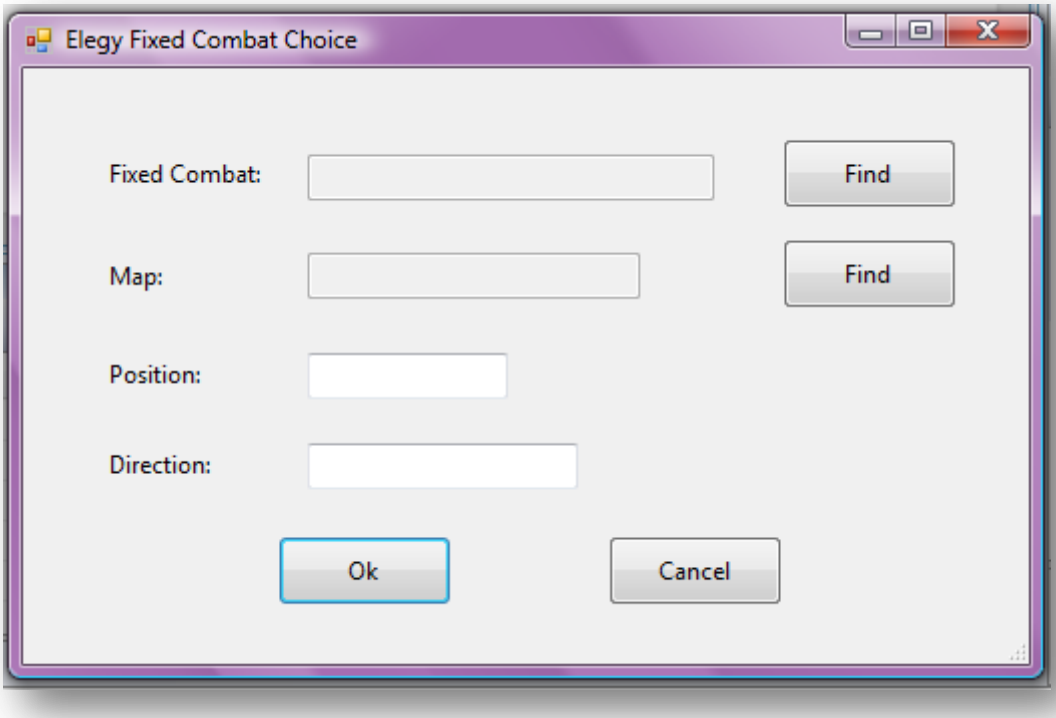


Figura 37 – Editor para a adição de Fixed Combats.

De modo análogo aos demais editores este permite ao usuário escolher o *fixed combat* desejado no diretório de personagens inimigos, juntamente com o mapa onde ele deverá ser adicionado. Abaixo está o editor para a escolha do *RequiredEnemy,* que possui o mesmo funcionamento dos demais, baseado na busca de inimigos já criados no diretório de personagens inimigos.

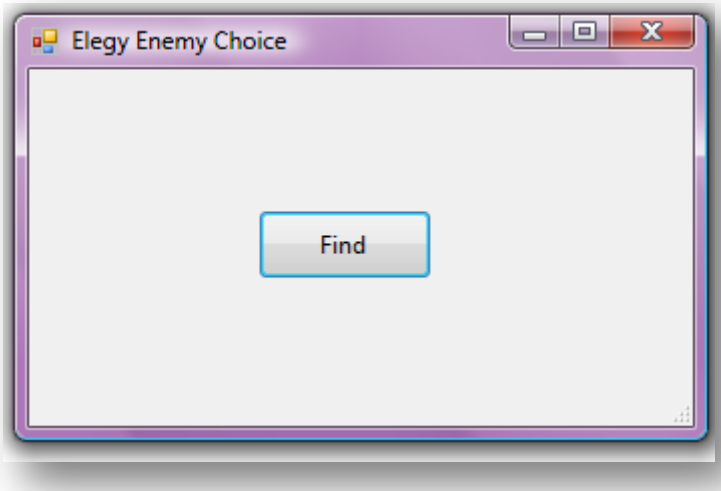


Figura 38 – Editor para a busca de inimigos requeridos a serem adicionados.

5.5.4. Geração de código

O código gerado pelo *QuestDefinitionDSL* está também no formato XML, pronto para ser consumido pela *engine.* Como é gerado apenas um arquivo para todos as *quests* criadas e a *engine* aceita arquivos separados, é necessário que este arquivo passe por um pré-processamento que irá separá-lo em arquivos distintos para cada *quest*.

5.6. EntityDefinitionDSL

Esta DSL é responsável pela criação de personagens, tanto personagem principal, inimigos e NPCs de *quests.*

5.6.1. Modelo do domínio

A modelagem desta DSL, de modo análogo as demais, teve como base a análise de funcionalidades e o código a ser consumido pela *engine*. O modelo, com suas classes e relacionamentos, pode ser visto nas figuras abaixo.

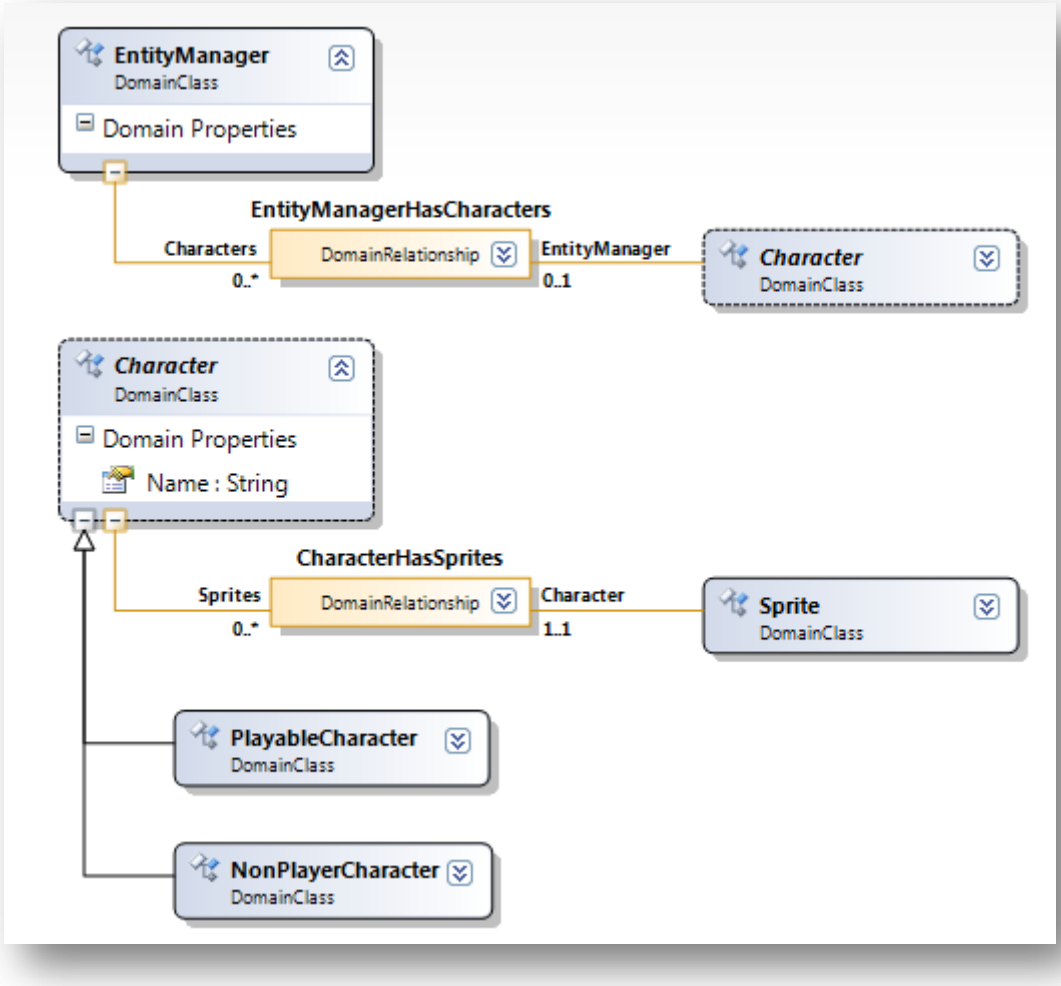


Figura 39 – Classes e relacionamentos do domínio.

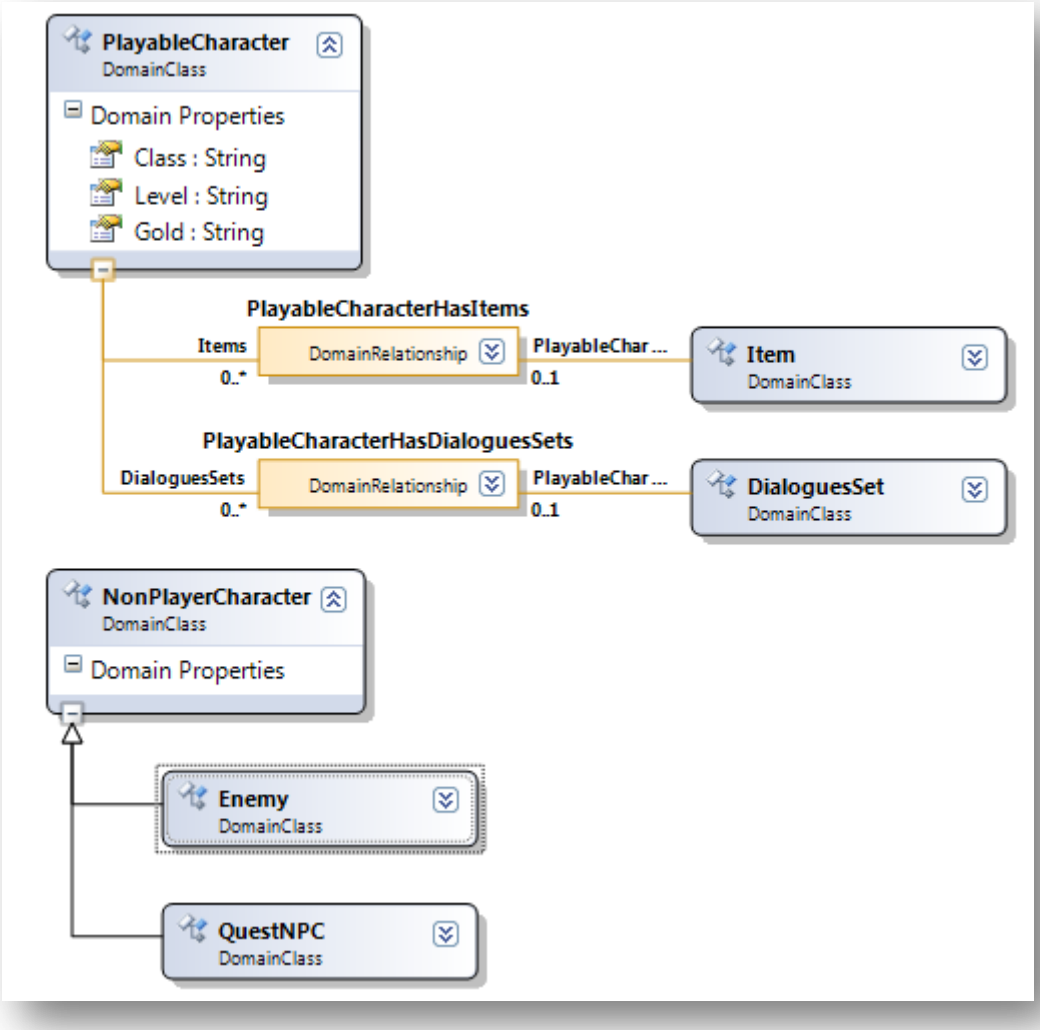


Figura 40 – Classes e relacionamentos do domínio.

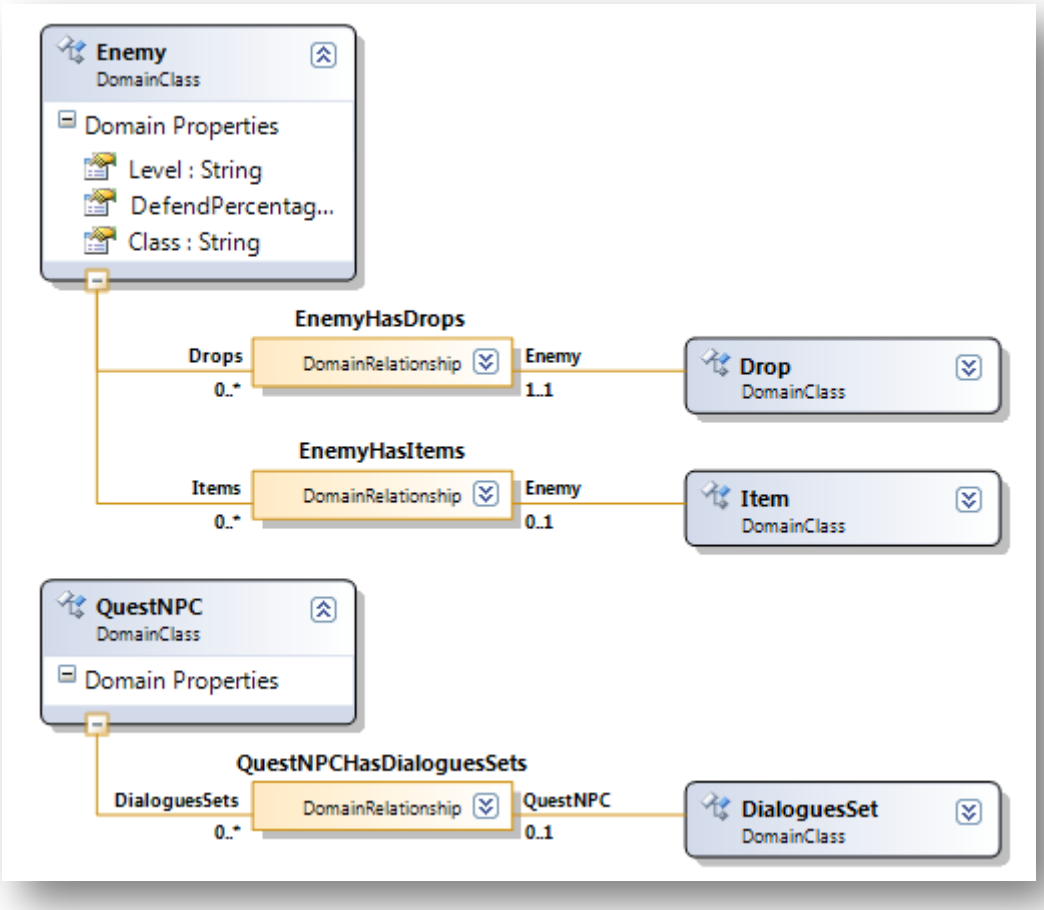


Figura 41 – Classes e relacionamentos do domínio.

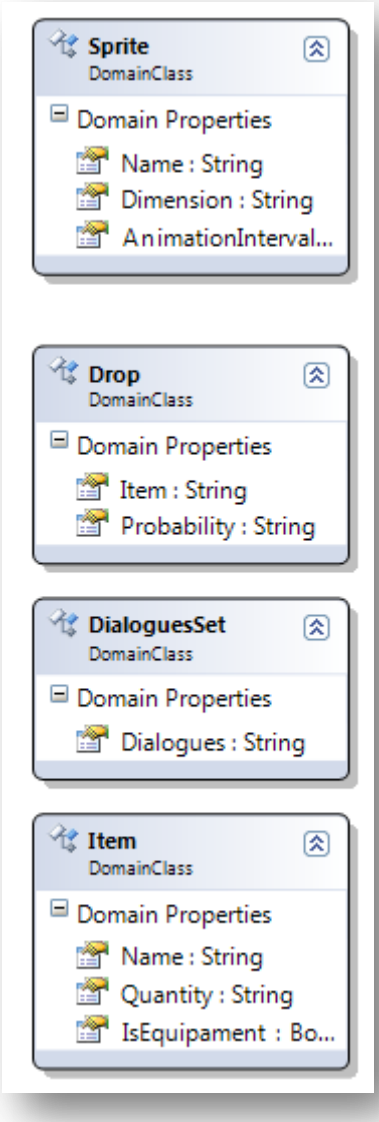


Figura 42 – Classes do domínio.

EntityManager: classe raiz do domínio. Agrupa todos os personagens criados.

Character: classe abstrata que representa um personagem. Possui como propriedade um nome e um relacionamento com a classe *Sprite.*

PlayableCharacter: classe que herda de *Character*. Representa o personagem principal, que o jogador controla. Possui como propriedades uma classe, nível e peças de ouro. Possui relacionamentos com as classes *Item* e *DialogueSet.* A propriedade classe também representa um ponto de integração entre DSLs, pois as classes também são definidas por arquivos XML. Nesta versão não foi implementada a DSL de criação de classes.

NonPlayableCharacter: classe que representa os personagens que não são controlados pelo jogador, como inimigos e NPCs de *quests*.

Enemy: classe que representa os inimigos. Possui como propriedades nível, percentual de defesa e classe, que analogamente a *PlayableCharacter* representa um ponto de integração. Possui relacionamentos com as classes *Drop* e *Item*.

QuestNPC: classe que representa os NPCs que oferecem *quests* ao personagem principal. Possui relacionamento com a classe *DialogueSet*.

Item: classe que representa os itens que podem ser equipados pelos personagens ou constar em seus inventários. Possui como propriedades nome, quantidade e um booleano que define se o item será diretamente equipado no personagem, caso contrário constará no inventário. Esta classe é mais um ponto de integração, pois todos os itens do jogo são definidos por arquivos XML. Como já dito, nesta versão não existe uma DSL de criação de itens.

Drop: classe que representa os itens que são deixados pelos inimigos após serem derrotados. Possui como propriedades o item e a probabilidade de ser deixado pelo inimigo. Também é um ponto de integração, analogamente à classe *Item.*

Sprite: classe que representa os *sprites* de cada personagem. Possui como propriedades nome, dimensão, intervalo de animação.

DialogueSet: classe que representa os diálogos travados durante o jogo entre os personagens. Faz parte do desenvolvimento da história do jogo.

5.6.2. Notação

Para este modelo as classes *PlayableCharacter*, *Enemy e QuestNPC* possuem representação visual através, respectivamente, das formas *PlayCharShape, EnemyShape e QuestNPCShape.*

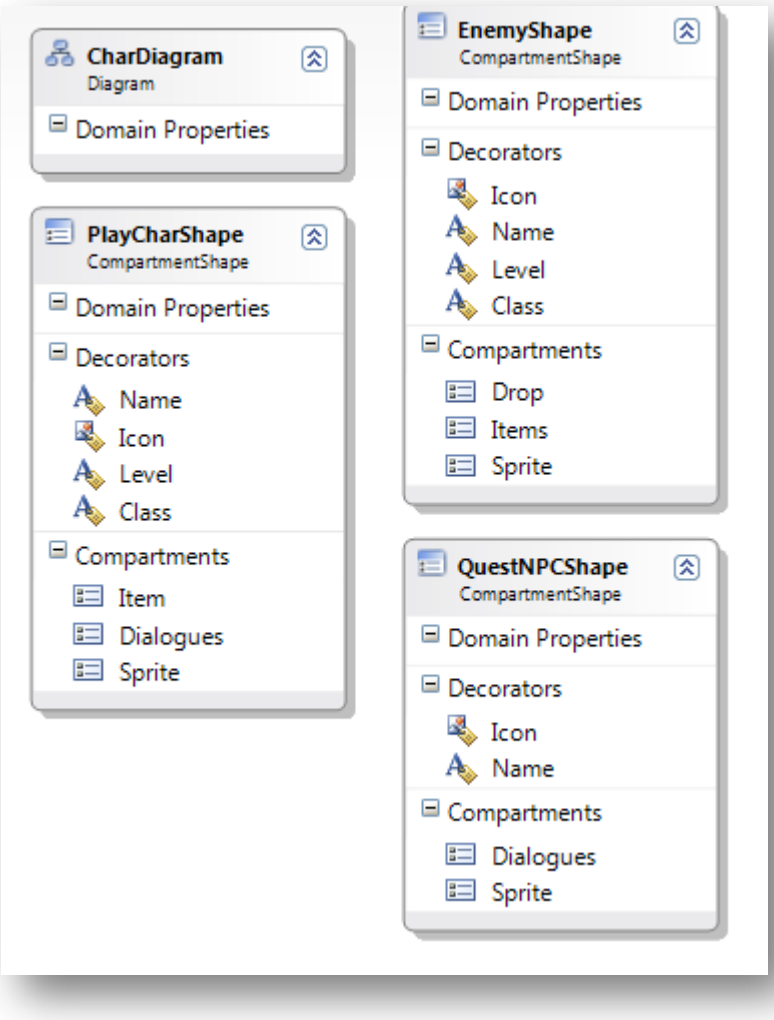


Figura 43 – Definição da notação da EntityDefinitionDSL.

A *PlayCharShape* possui como decoradores o nome do personagem, um ícone representativo, o nível e a classe, que aparecerão em destaque. Possui também compartimentos onde aparecem as classes com as quais *PlayableCharacter* mantém relacionamentos. A *EnemyShape* também possui decoradores de nome, ícone, nível e classe, e compartimentos para as classes com as quais se relaciona. A *QuestNPCShape* possui decoradores de nome e ícone, e compartimentos para a adição de diálogos e *sprite*, que são as classes com as quais *QuestNPC* mantém relacionamento. Abaixo está a representação visual das classes de um personagem principal, um NPC e um inimigo.

Figura 44 – Representação visual das classes.

5.6.3. Integração

O formato dos arquivos criados nesta DSL foi definida como ".char". A *toolbox* traz opções de criação de personagem principal, inimigos e NPCs. Foram criados editores customizados para definição de propriedades.

Abaixo está a imagem que mostra a integração da linguagem ao *Visual Studio*. Pode-se ver a *toolbox* à esquerda com as opções de criação de personagens, que podem ser arrastados para o diagrama ao centro e à direita, acima a árvore de visualização e abaixo a janela de visualização de propriedades.

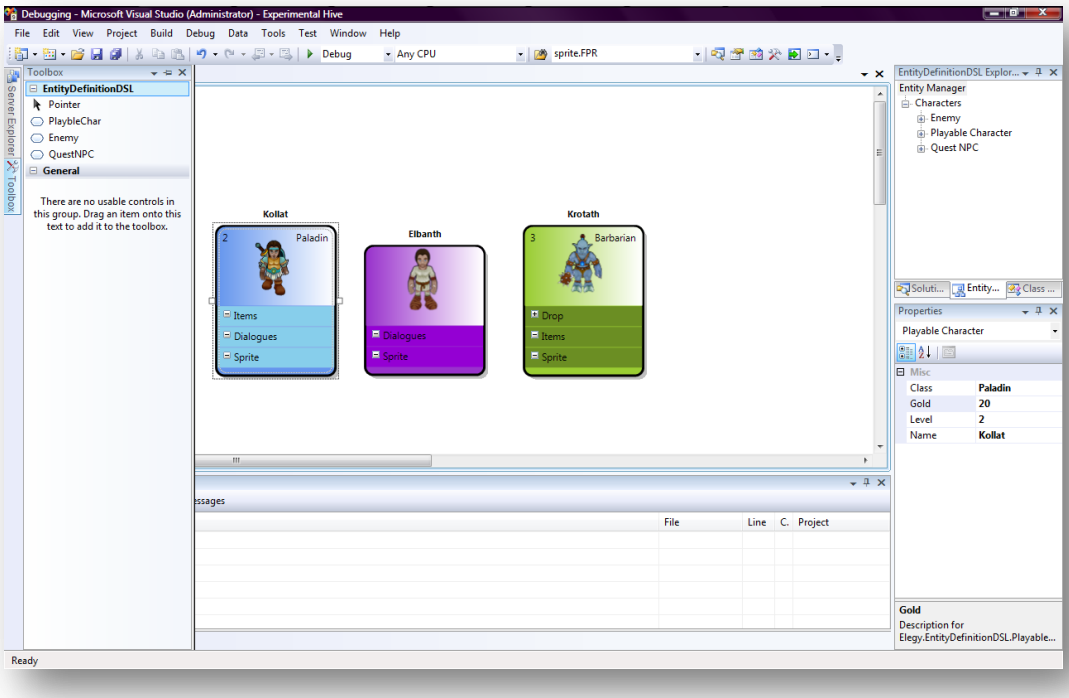


Figura 44 – *EntityDefinition* integrada ao Visual Studio.

Abaixo estão os editores customizados criados para facilitar a edição das propriedades das classes do domínio.

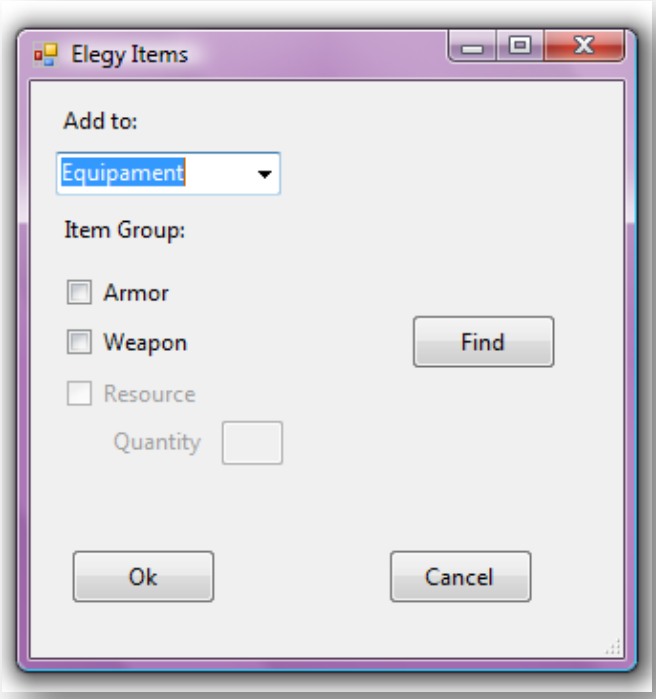


Figura 45 – Editor de dição de itens ao personagem.

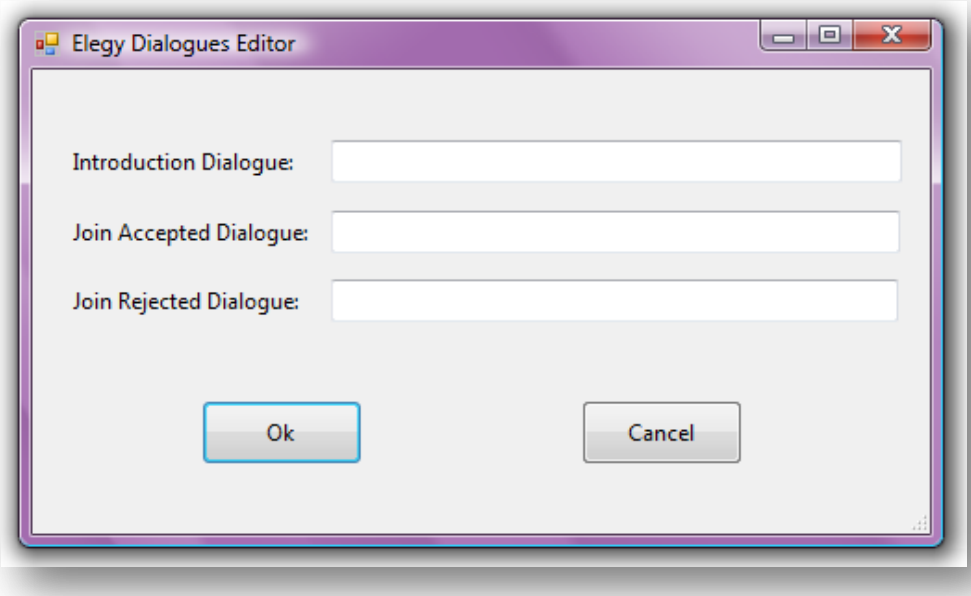


Figura 46 – Editor de adição de diálogos aos personagens.

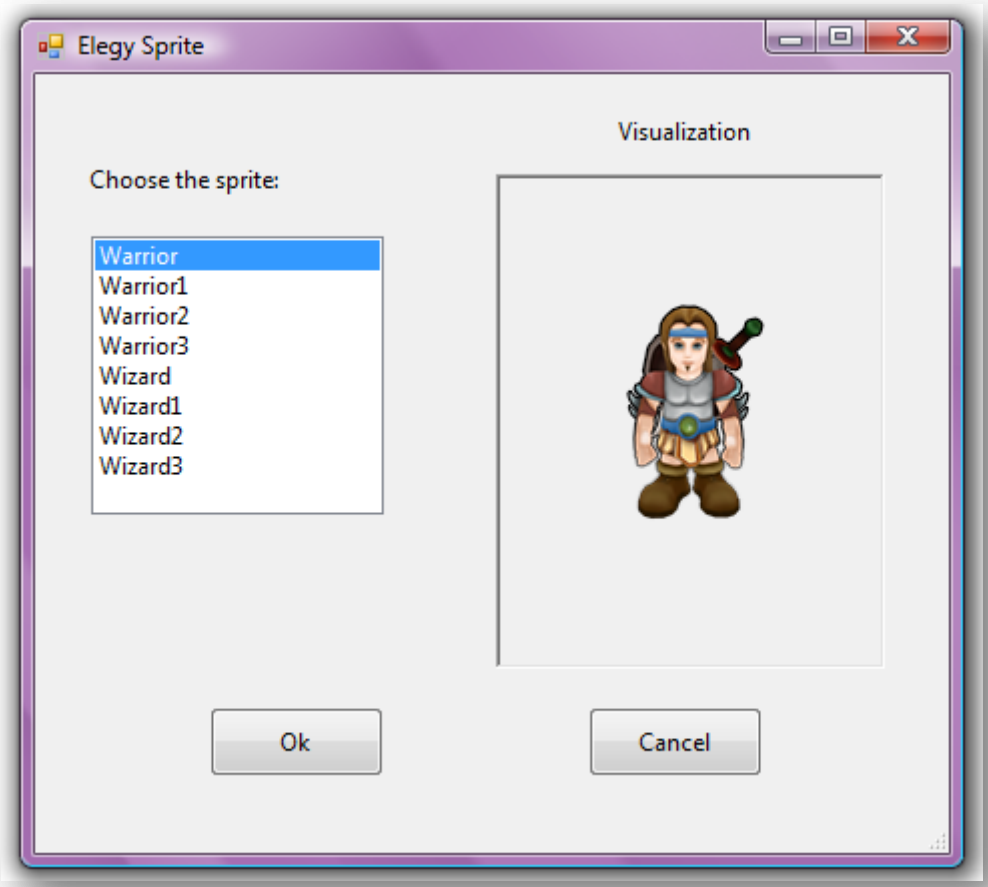


Figura 47 – Editor de escolha dos sprites do personagem.

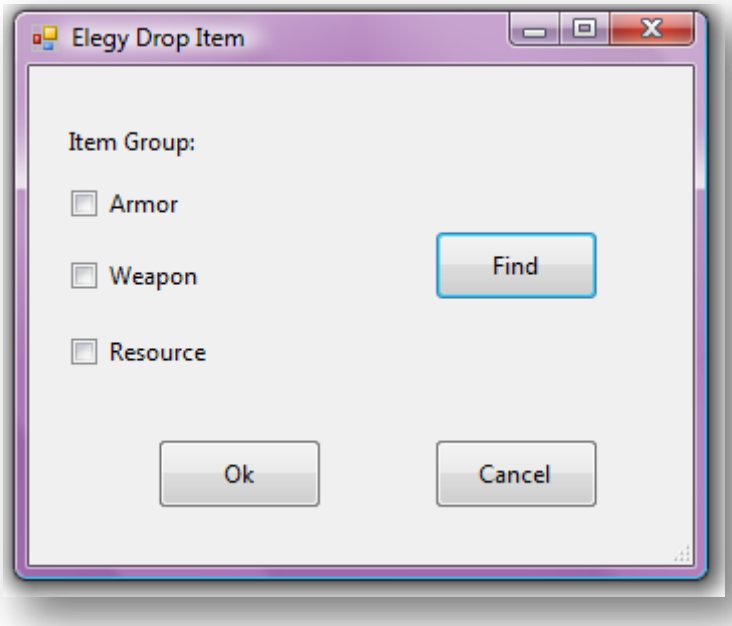


Figura 48 – Editor de escolha de drops dos inimigos.

5.6.4. Geração de código

O código gerado pela *EntityDefinitionDSL* também é no formato XML, pronto para ser consumido pela *engine*. Como é gerado apenas um arquivo para todos os personagens criados e a *engine* aceita arquivos separados, é necessário que este arquivo passe por um pré-processamento que irá separá-lo em arquivos distintos para cada entidade.

O próximo capítulo traz um estudo de caso para validar o funcionamento das três DSLs implementadas junto à *engine RPG Starter Kit.*

6. Estudo de Caso

Neste capítulo será mostrada a construção de um pequeno jogo para validar o funcionamento das DSLs modeladas junto ao *RPG Starter Kit.* O jogo é bastante simples, contendo um personagem principal, um NPC e uma *quest*, apenas para ilustrar as funcionalidades já consolidadas nesta versão do projeto.

Apesar da simplicidade aparente dos jogos que podem ser implementados utilizando a *Elegy*, é possível adicionar uma diversidade de mapas distintos, com transições, de maneira prática, diferente do que a *engine* no modo atual oferece, apenas permitindo alterações manuais de arquivos XML complexos.

6.1. Construção do Jogo

Primeiramente criaremos os mapas. Apenas dois mapas serão criados, com uma transição. Abaixo se pode ver a imagem dos dois mapas criados e da transição do primeiro para o segundo mapa. Alguns erros de validação foram apontados, como a falta de definição da música de fundo.

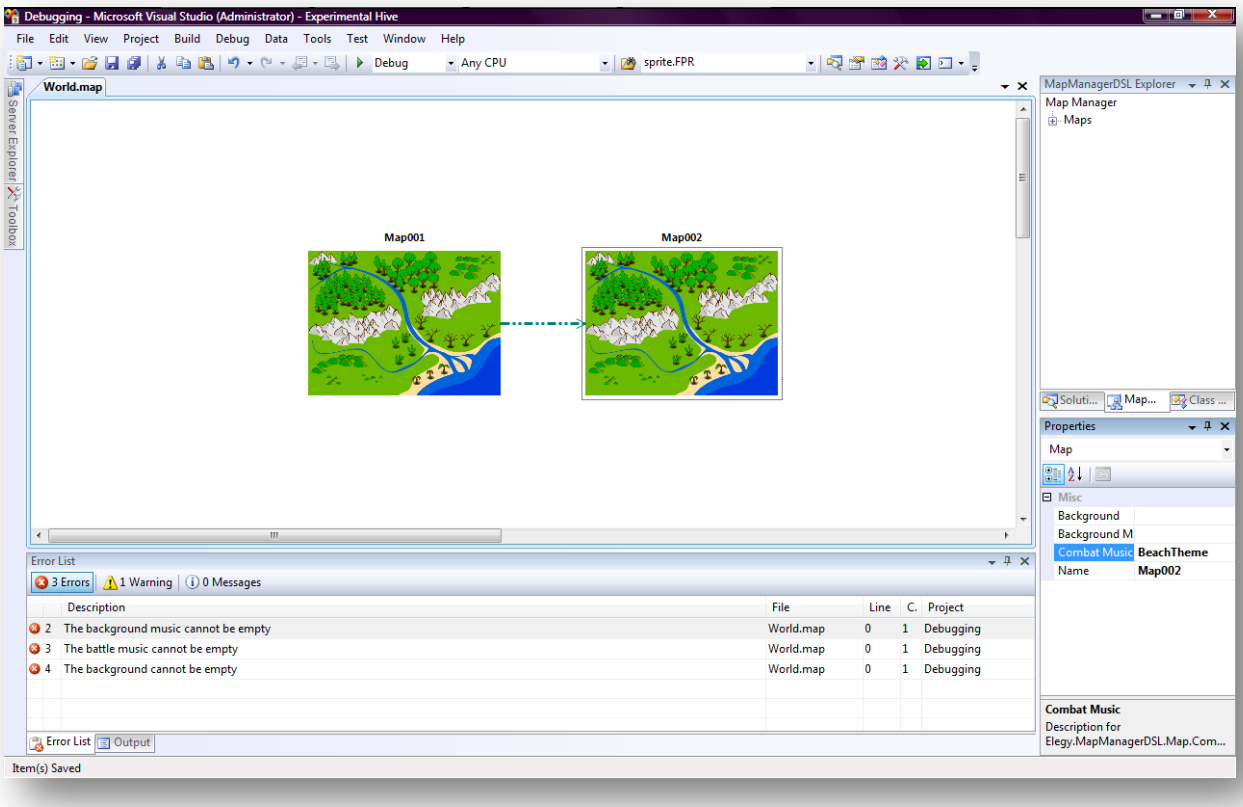


Figura 49 – Criação dos mapas.

Nomeamos o primeiro mapa de *map001* e o segundo de *map002*, criamos uma transição do *map001* para o *map002* e definimos propriedades como música de fundo e música de combate. A próxima propriedade a ser definida é o *background*, através do editor *MapCreator,* como pode ser visto nas figuras abaixo.

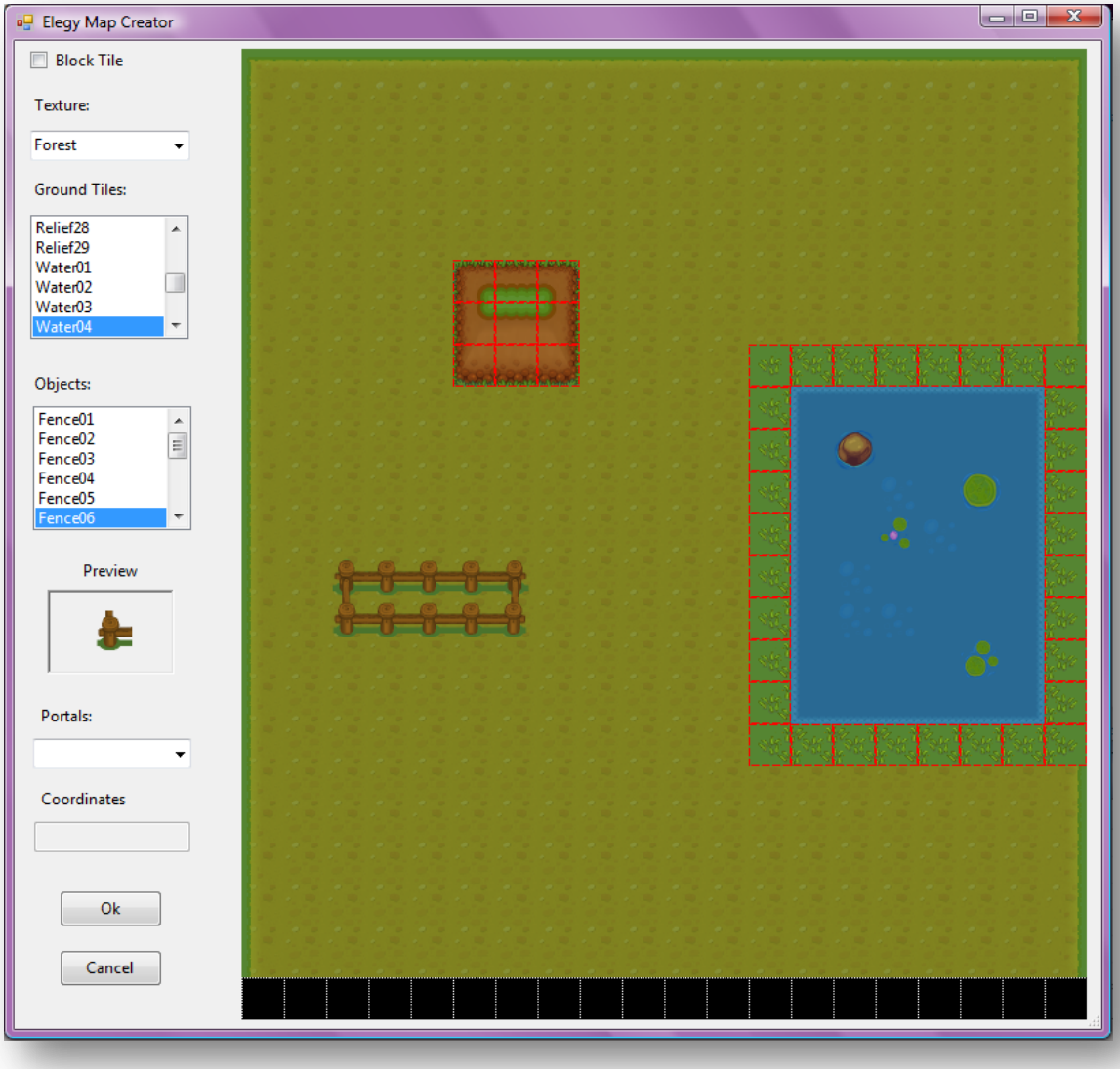


Figura 50 – Editando o background do map001.

Através deste editor podemos escolher os *tiles* que irão formar o mapa. Nesta versão os mapas são limitados a 20 x 23 *tiles*. Os *tiles* contornados em vermelho são onde ocorrem colisões, ou seja, são intransponíveis. *Tiles* como cercas e árvores já são naturalmente intransponíveis. O campo *Portals* é preenchido com todas as transições definidas no modelo. Através dele podemos decidir onde serão as coordenadas de transição.

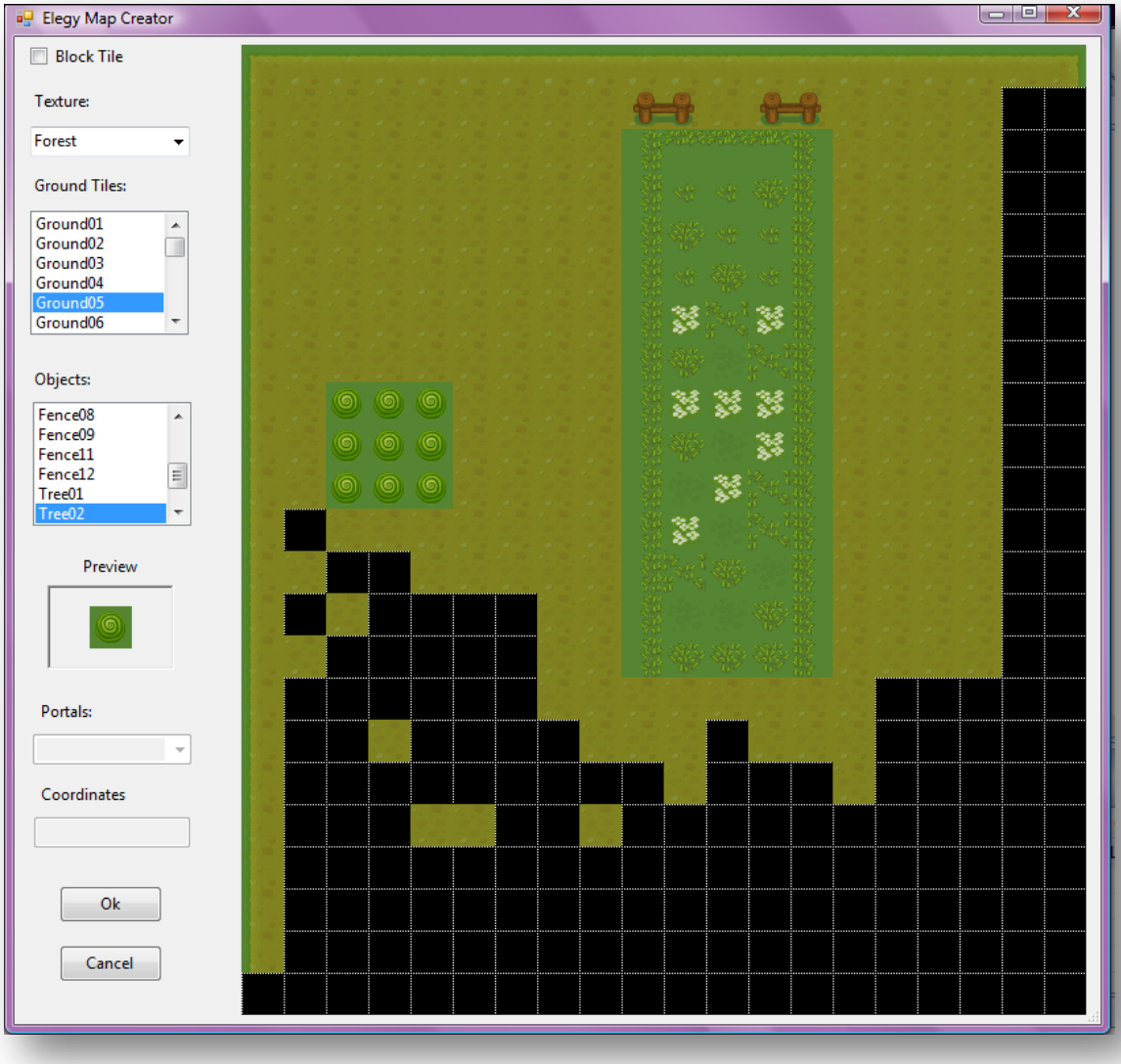


Figura 51 – Editando o background do map002.

Agora definiremos os personagens. Neste jogo haverá o personagem principal, chamado Kollat, um NPC chamado Elbanth e um inimigo, chamado Krotath. A figura seguinte mostra os três personagens criados. Caso alguma propriedade obrigatória, como *sprite,* não seja adicionada, ocorrerá um erro de validação.

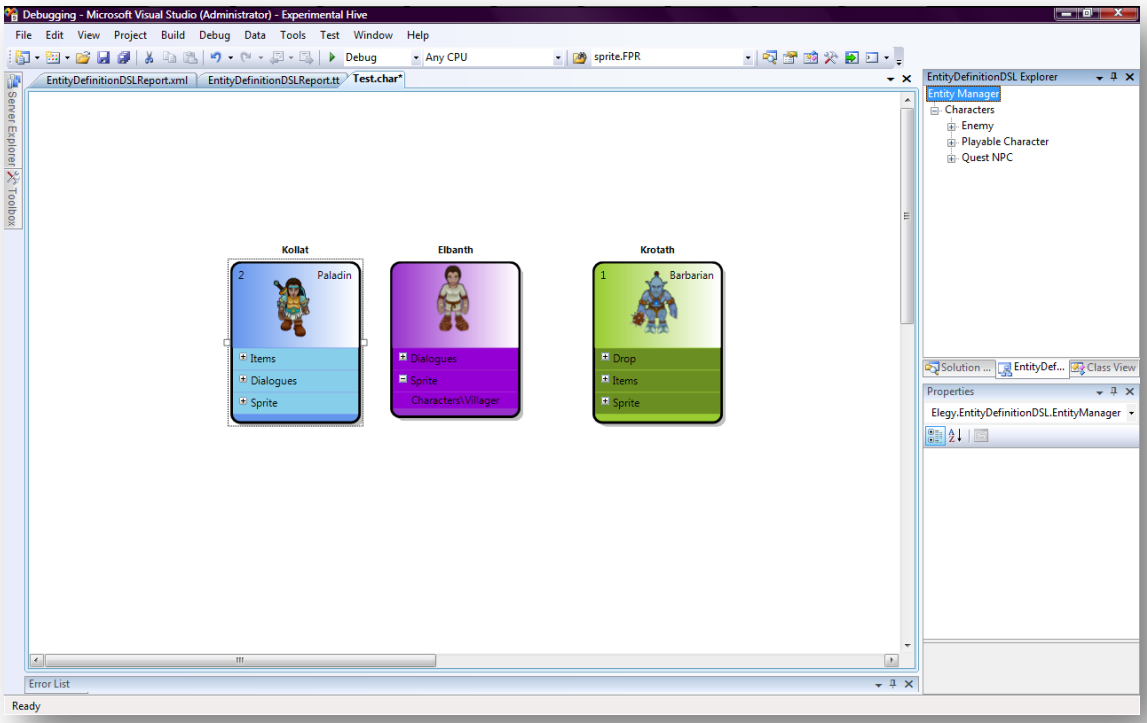


Figura 52 – Criando personagens.

A classe é escolhida através de uma busca no diretório de classes. Os itens para equipamento e inventário são escolhidos da mesma forma, através dos editores já mostrados no capítulo anterior, e no caso do inimigo também é escolhido o item que poderá ser deixado após ser vencido em uma batalha. Os *sprites* podem ser adicionados através do editor, e para cada tipo de personagem existe um grupo distinto disponível. Os diálogos podem ser adicionados através do editor de diálogos. Abaixo estão os editores das propriedades do personagem principal. Para os demais personagens o processo é análogo.

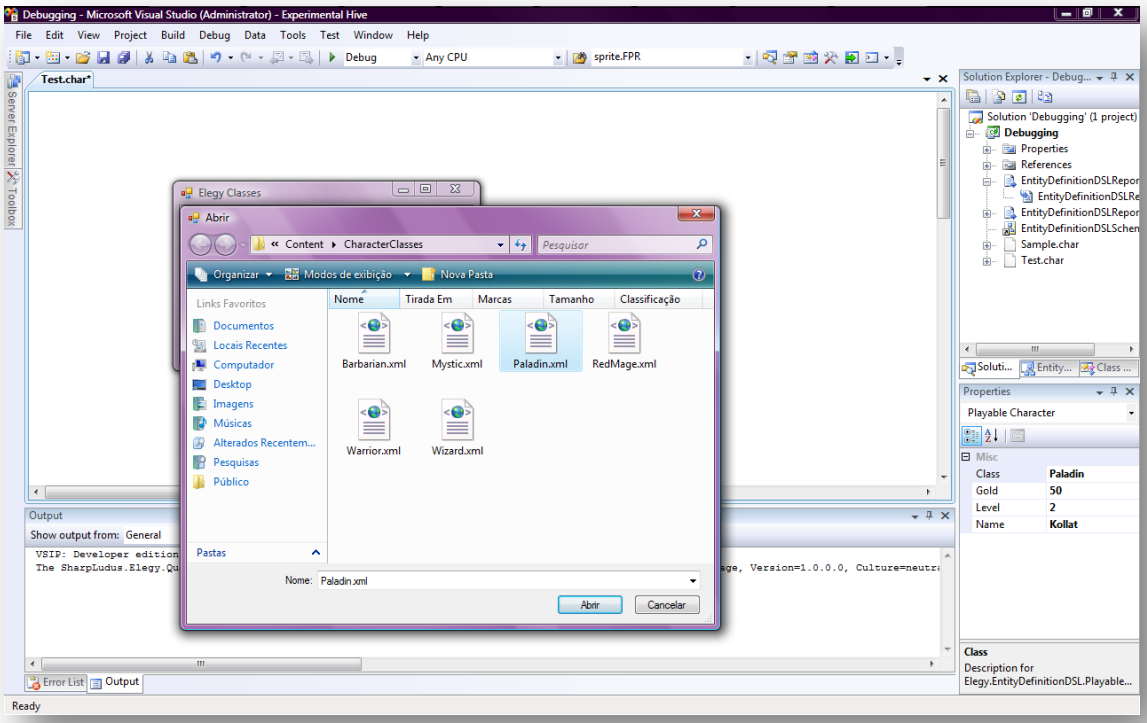


Figura 53 – Editor para escolha de classe.

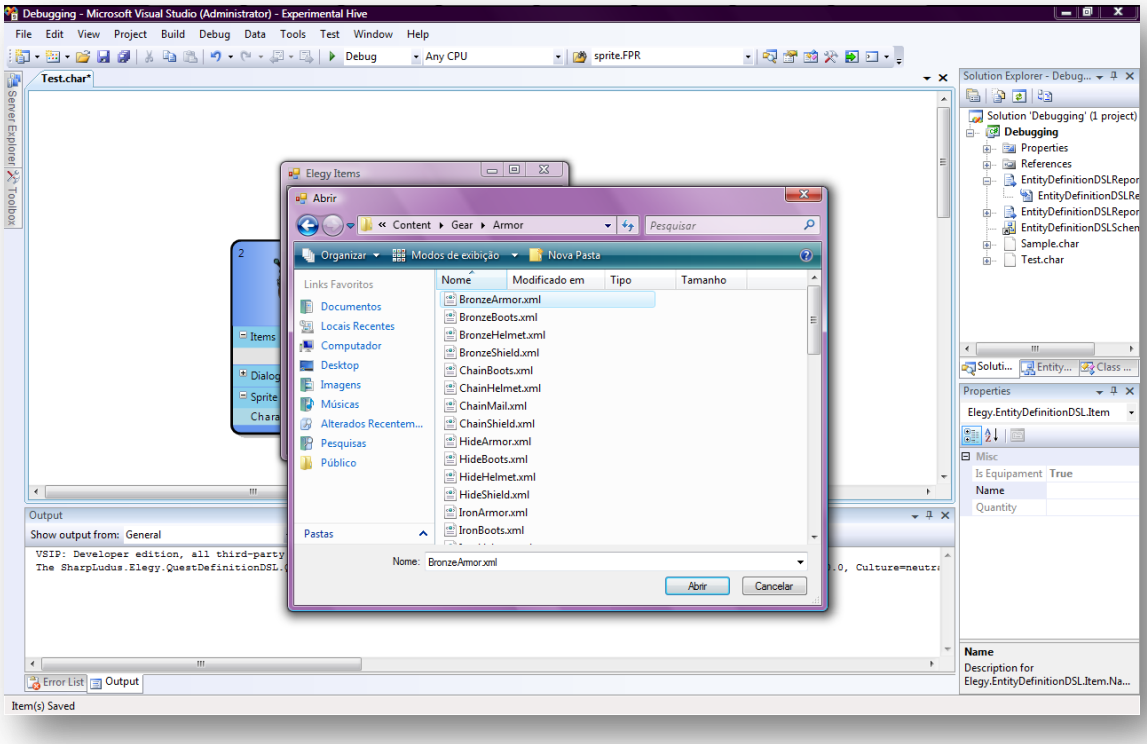


Figura 54 – Editor para escolha de itens.

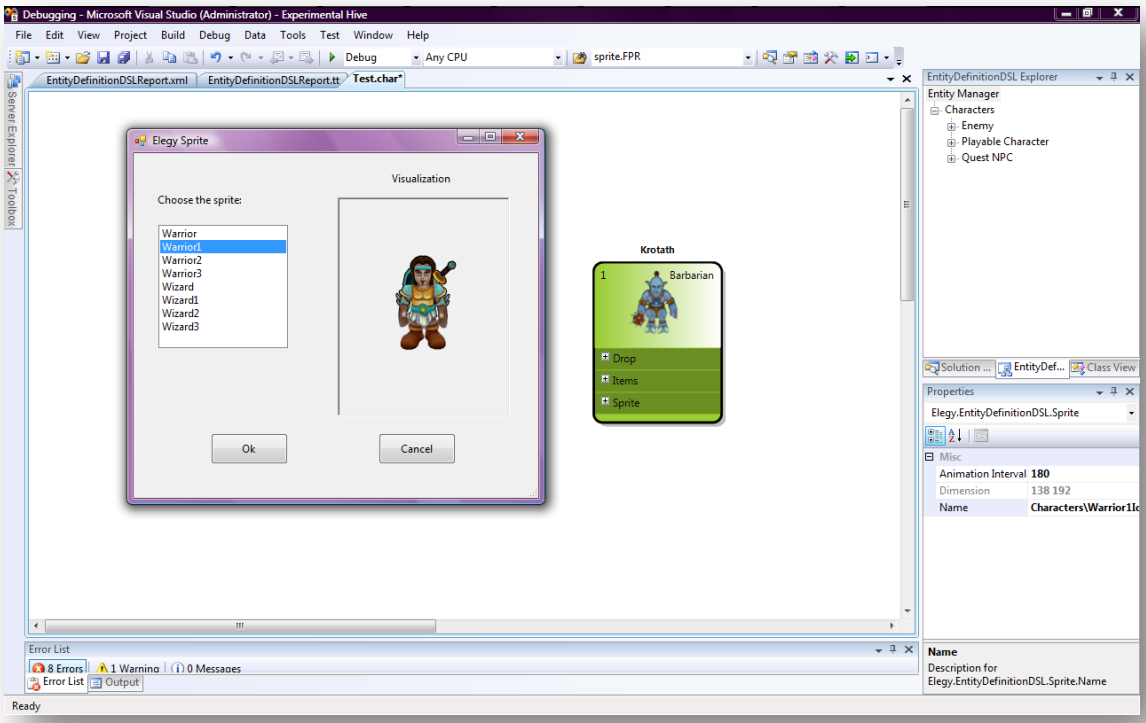


Figura 55 – Editor de sprite.

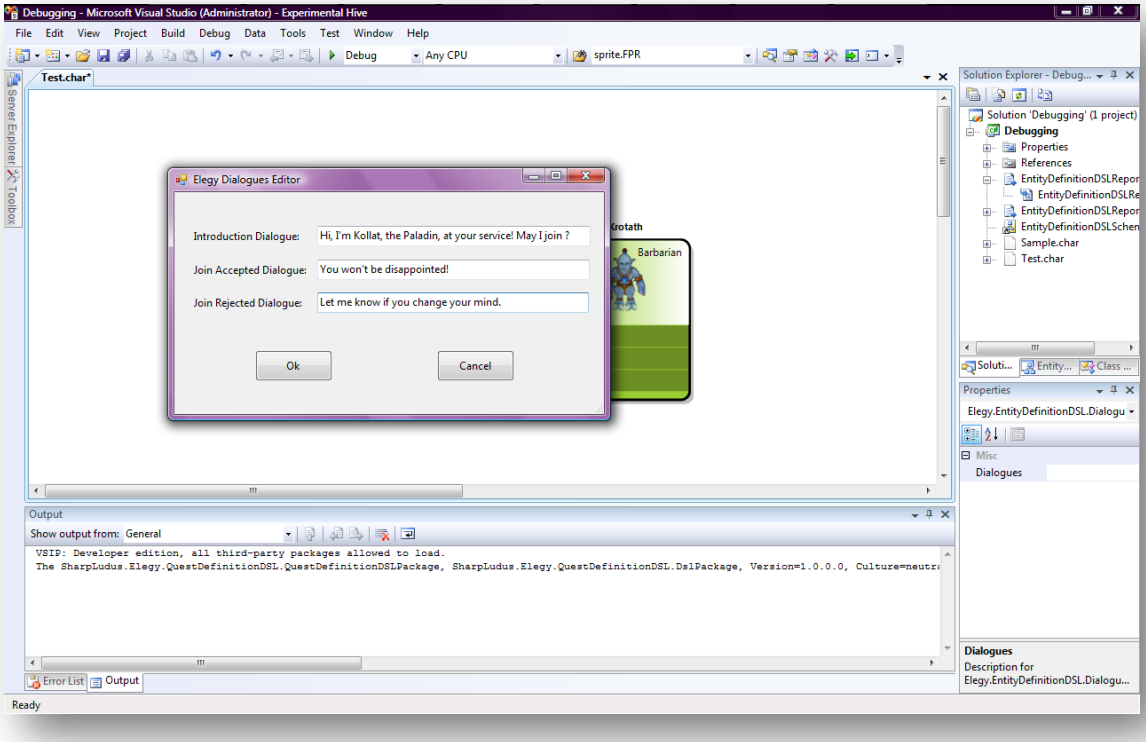


Figura 56 – Editor de diálogos.

Terminada a edição de personagens e a geração dos arquivos XML, podemos criar uma *quest*. Definimos o nome da *quest* como *What is happening in the Village"*. Essa *quest* tem como objetivo exterminar um monstro que está aterrorizando a aldeia, no caso o personagem criado anteriormente, Krotath. O NPC relacionado a esta *quest* será Elbanth. Definimos também os pontos de experiência, itens de recompensa e peças de ouro. Abaixo imagens da criação da *quest.*

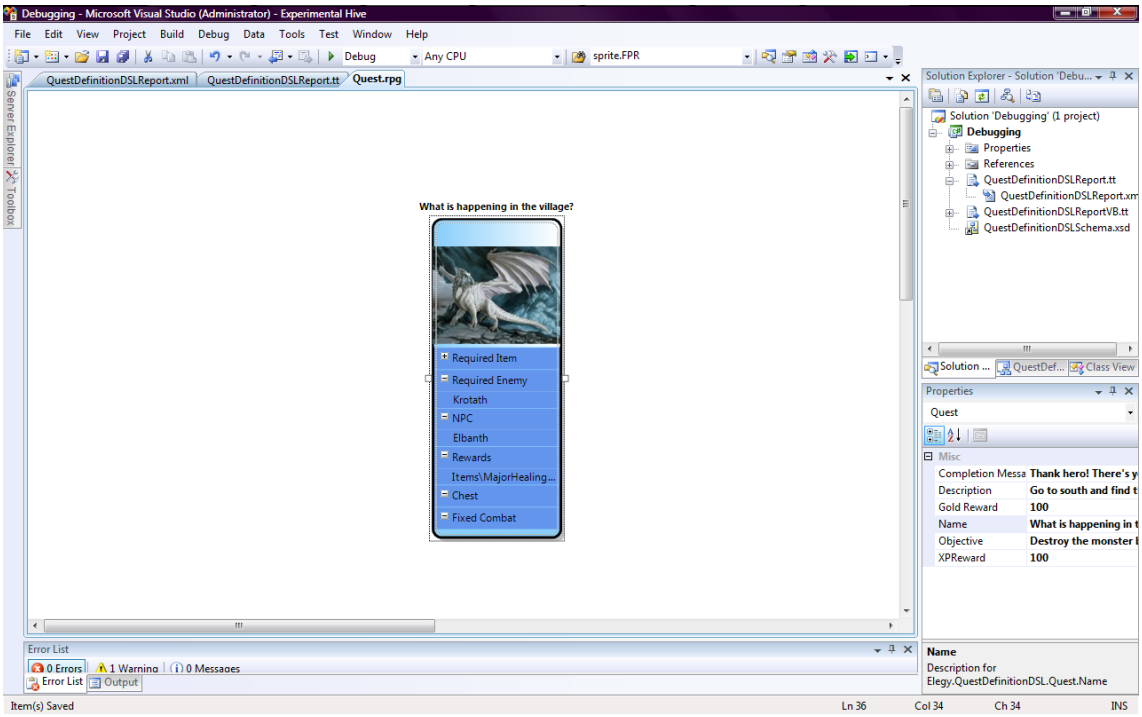


Figura 57 – Criação da quest.

Após definirmos a *quest* foi necessário alterar manualmente algumas *tags* no arquivo XML dos mapas para a introdução do NPC e do inimigo a ser combatido, pois ainda não existe a integração entre as DSLs. Ao se adicionar o NPC e à *quest* o mapa em que ele está localizado deve ser definido (Figura 34), e, portanto não precisa ter seu arquivo gerado alterado.

Abaixo estão as imagens do jogo criado.

Figura 58 – Imagem do personagem principal e o NPC no mapa.

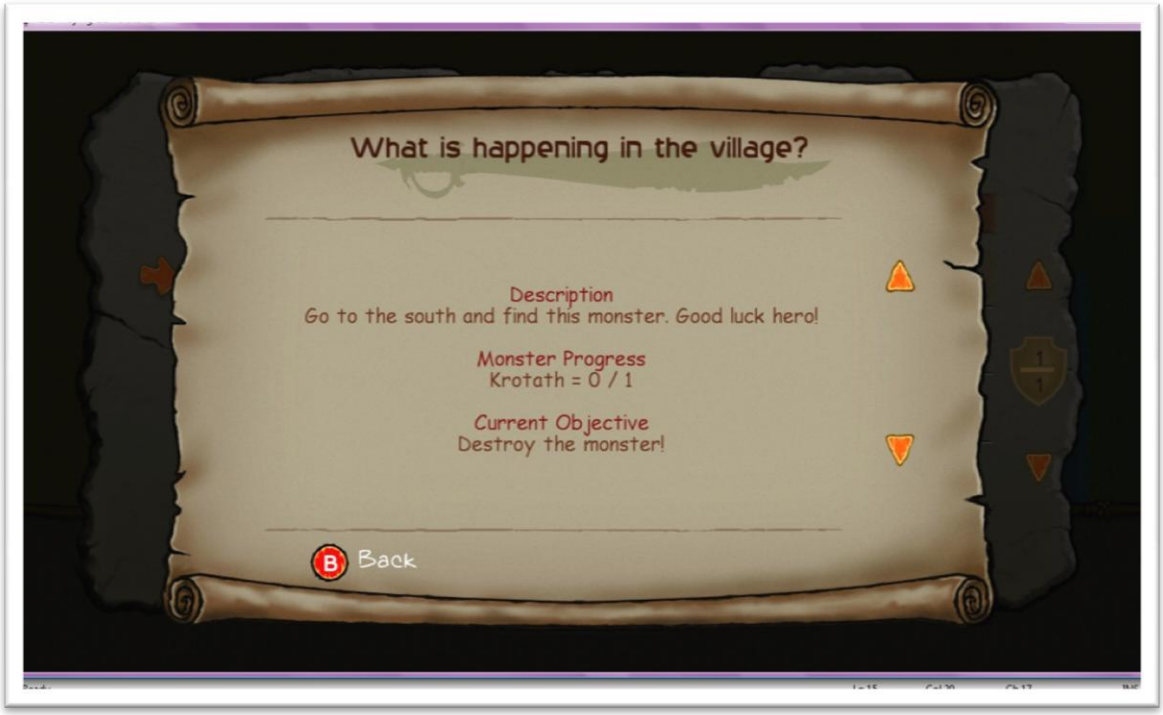


Figura 59 – Descrição da quest.

Figura 60 – Diálogo com o NPC.

Figura 61 – Batalha com Krotath.

Figura 62 – Batalha vencida.

Figura 63 – Quest finalizada.

7. Considerações Finais

A importância deste trabalho não se resume à visualização de uma ferramenta consolidada e com grande potencial. O grande desafio na implementação produtiva de *games* através da abordagem de Fábricas de *Software* está na escolha de um processo adequado, confiável e que traga resultados satisfatórios. Este processo deve abranger todas as etapas de elaboração do projeto, desde sua idealização, *design*, até sua implementação.

O alvo do projeto *Elegy* esteve justamente no estudo e aplicação da metodologia SharpLudus durante a etapa de análise de domínio e funcionalidades, trazendo contribuições para esta metodologia, que ainda está em produção. Nesta fase o domínio alvo, no caso dos jogos RPG, foi estudado exaustivamente, de modo que pudesse ser compreendido em nível de funcionalidades. Desta forma foi possível fazer um diagrama de funcionalidades amplo o suficiente para ser base da implementação do *software*, trazendo contribuições para o projeto *FeatureModelDSL*, que mostrou grande potencial para esta tarefa.

Complementando o uso da metodologia, o desenvolvimento de domínio específico traz como grande vantagem o foco expressivo, através da modelagem de *Domain-Specific Languages*. A ferramenta escolhida para a modelagem de DSLs foi a *DSL Tools*, da *Microsoft*, por sua praticidade na elaboração dos diagramas e o grande interesse na programação com a linguagem C#.

Para validar o projeto de DSL é necessário um *framework*, que consumirá o código gerado. A *engine RPG Starter Kit* foi escolhida para esta finalidade, com a vantagem de trazer portabilidade para a plataforma XNA. Desta forma a criação da *Elegy* não foi feita de forma linear. Enquanto a análise era processada, um estudo sobre a *engine* foi feito e culminaram na produção de três DSLs: *MapManagerDSL*, para criação e gerenciamento de transição de mapas, *EntityDefinitionDSL*, para criação de personagens e *QuestDefinitionDSL*, para a criação de *quests.*

Muitos desafios foram encontrados devido à *engine* não ter sido produzida propriamente para a *Elegy.* Alguns deles puderam ser contornados, fazendo com que a criação de jogos RPG com esse motor de jogos se tornasse

mais produtiva e interessante. Outros, como a necessidade de integração de DSLs, não puderam ser resolvidos nesta versão.

Por final, um estudo de caso foi realizado, mostrando o bom desempenho dos projetos de DSL junto à *engine*. O próximo tópico traz sugestões de trabalhos futuros.

7.1. Trabalhos Futuros

Existem vários pontos em que o projeto pode ser melhorado. O uso da metodologia para cobrir o maior número de funcionalidades do domínio é bastante importante, necessitando então de maiores iterações no processo. Além disso, como foi apontado, a produção de DSLs para criação de itens, classes, *spells,* entre outros, não pode ser realizada nesta versão. No entanto é de grande importância tê-las no projeto, e integrá-las, de forma a se obter jogos complexos e aumentar o potencial da ferramenta, e quem sabe colocá-la em nível de ser comercializada. Para isso também é necessário um estudo sobre o que pode ser alterado na própria *engine*, para que o projeto DLS + motor de jogos funcione naturalmente de modo satisfatório.

8. Bibliografia

Cook, S., Jones, G., Stuart, K., & Wills, A. C. (2007). *Domain-Specific Development with Visual Studio DSL Tools.* Addison-Wesley Professional.

Coplien, J., Hoffman, D., & Weiss, D. (1998). Commonality and Variability in Software Engineering. *IEEE Software* .

Deursen, A. v., Klint, P., & Visser, J. (s.d.). *Domain-Specific Languages: An Annotated Bibliography.* Acesso em 6 de Março de 2009, disponível em http:\\citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.8207

ESA. (2007). *Essential Facts About the Computer Game Industry.* Acesso em 1 de Março de 2009, disponível em www.theesa.com/facts/pdfs/ESA_EF_2008.pdf

Furtado, A. W. (2008). *Feature Model DSL*. Acesso em 22 de Abril de 2009, disponível em http://www.codeplex.com/FeatureModelDSL

Furtado, A. W. (s.d.). *SharpLudus*. Acesso em 1 de Março de 2009, disponível em http://www.codeplex.com/sharpludus

Furtado, A. W. (2006). *SharpLudus: Improving Game Development Experience through Software Factories and Domain-Specific Languages.* Recife: UFPE.

Furtado, A. W. (s.d.). Streamlining Digital Games Development through Software Factories Automation (Título temporário). *Tese de doutorado em andamento.*

Furtado, A. W., & Santos, A. L. (s.d.). *Tutorial: Applying Domain-Specific Modeling to game Development with the Microsoft DSL Tools.* Acesso em 6 de Março de 2009

Greenfield, J., & Short, K. (2004). *Moving to Software Factories.* Acesso em 6 de Março de 2009, disponível em www.softwarefactories.com/ScreenShots/MS-WP-04.pdf

Greenfield, J., & Short, K. (2003). *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools.* Acesso em 6 de Março de 2009, disponível em www.softmetaware.com/oopsla2003/greenfield.pdf

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (Novembro de 1990). *Feature-Oriented Domain Analysis (FODA): Feasibility Study.* Acesso em 22 de Março de 2009, disponível em http://selab.postech.ac.kr/classes/eece700A/materials/papers/2_FODA.pdf

Neighbors, J. M. (1980). Software Construction Using Components Software Construction Using Components.

***Pong Story*.** (2008). Acesso em 23 de Março de 2009, disponível em http://www.pong-story.com/intro.htm

Prieto-Diaz, R. (1990). Domain Analysis: An Introduction. *ACM SIGSOFT* .

Riebisch, M. (2003). *Towards a More Precise Definition of Feature Models.* Acesso em 22 de Abril de 2009, disponível em http://www.theoinf.tu-ilmenau.de/~riebisch/home/publ/06-riebisch.pdf

Riebisch, M., Böllert, K., Streitferdt, D., & Philippow, I. (2002). *Extending Feature Diagrams with UML Multiplicies.* Acesso em 22 de Abril de 2009, disponível em http://www.theoinf.tu-ilmenau.de/~riebisch/home/publ/IDPT2002-paper.pdf

***Role-Playing Game Starter Kit*.** (s.d.). Acesso em 1 de Março de 2009, disponível em http://creators.xna.com/en-US/starterkit/roleplayinggame

Wizards. (2009). Acesso em 1 de Março de 2009, disponível em Dungeons and Dragons: http://www.wizards.com/default.asp?x=dnd/insider

9. Apêndice A

Este apêndice apresenta de maneira detalhada as funcionalidades contidas nos *feature models* do capítulo 4. Este relatório foi gerado pela ferramenta *FeatureModelDSL.*

Root

Feature model raiz para o domínio RPG

Features

(8 mandatory, 0 optional, 0 under alternatives, 7 references, 9 total)

- **RPG Game**
  - **Main**: Funcionalidades próprias ao domínio RPG.
  - **Flow**: Descrição do fluxo das telas.
  - **Input**: Definição dos dispositivos de entrada.
  - **Audio**: Definição do sistema de som.
  - **Physics**: Definição da física.
  - **Graphics**: Definição dos aspectos gráficos.
  - **AI**: Definição da inteligência artificial existente.
  - **Platform**: Plataformas para as quais o jogo pode ser feito.

Main

Feature Model que descreve as características mais peculiares em relação a jogos RPG.

Features

(4 mandatory, 0 optional, 0 under alternatives, 4 references, 5 total)

- **Main**
  - **Entity**: Definição de como são as entidades do jogo: personagens jogáveis ou não.
  - **Battle System**: Definição do sistema de batalha.
  - **Item System**: Define os tipos de itens existentes no jogo.
  - **GamePlay**: Características próprias do Role-Play.

Entity

Feature Model que descreve as entidades do jogo, ou seja, os personagens.

Features

(21 mandatory, 16 optional, 19 under alternatives, 0 references, 57 total)

- **Entity**: Personagens do jogo.
    - **Attributes**: Atributos do personagem.
        - **Attack**: Potência de ataque físico.
        - **Evade**: Evasão em relação a ataques físicos.
        - ***Magic*** (optional): Potência de ataque mágico.
        - **Defense**: Defesa em relação a danos físicos.
        - ***Magic Defense*** (optional): Defesa em relação a ataques mágicos.
        - **Hit Points**: Pontos de vida.
        - ***Magic Points*** (optional): Pontos de Magia.
        - ***Speed*** (optional): Velocidade do personagem.
        - ***Magic Evade*** (optional): Evasão em relação a ataques mágicos.
        - **Experience Points**: Quantidade total de pontos de experiência adquiridos.
        - **Level**: Nível.
        - ***Stamina*** (optional): Vigor físico.
        - ***D&D System*** (optional): Sistema de atributos D&D. Possui como atributos principais força, destreza, inteligência, sabedoria, constituição física e carisma.
    - **Shape**: Forma do personagem.
        - **Image**: Arquivos de imagem do personagem.
            - **Animation**: Animação produziada através dos arquivos de imagens do personagem.
    - **Skills**: Habilidades do personagem.
        - **Gain**: Forma como a habilidade é adquirida.
            - Alternative [1..1]
                - **Per Level**: Ao atingir determinado nível a habilidade pode ser aprendida.

        - **Racial**: Habilidades dependentes da raça.
        - **Class**: Habilidades dependentes de classe.
        - **Profession.**: Habilidades dependentes de profissão.
        - **Special Cases**: Casos especiais, como por exemplo: em determinadas quests pode-se obter uma habilidade, um item especial pode dar habilidade ao personagem que o usa, etc.
- **Type**: Tipos de habilidade.
    - Alternative [1..1]
        - **Magical**: Mágica. Possui as mais variadas finalidades, como ataque, cura, recuperação de status, etc.
        - **Physical**: Física. Geralmente de ataque físico.
- **Use**: Habilidades quanto a finalidade.
    - Alternative [1..1]
        - **Debuff**: Habilidade que prejudica atributos diversos.
        - **Buff**: Habilidade que oferece ganho em atributos diversos.
        - **Damage**: Utilizada para dano.
        - **Recover**: Recuperar atributos e status.
        - **Summon**: Invocar seres.
        - **Utility**
- ***Damage Type*** (optional): Tipos de danos provocados por habilidades com esta finalidade. Geralmente são elementais.
    - Alternative [1..1]
        - **Fire**: Dano de fogo.
        - **Cold**: Dano de gelo/frio.
        - **Wind**: Dano de vento.
        - **Lightning**: Dano de Luz.
        - **Poison**: Dano de veneno.
        - **Special Types**: Tipos que fogem a regra.
- ***Effects*** (optional): Efeitos em relação ao status do personagem.
- ***Duration*** (optional): Duração da habilidade.
- ***Activation*** (optional): Modo de ativação da habilidade.

- **Team**: Grupo ao qual o personagem faz parte.
- ***Status*** (optional): Condições do jogador geralmente em relação a envenenamento, cegueira, petrificação, dentre outros tipos de danos permanentes.
- ***Class.*** (optional): Classe ao qual o personagem faz parte, como paladinos, clérigos, arqueiros, magos, etc.
- ***Race*** (optional): Raça a qual pertence o personagem. Bastante comum a jogos no sistema D&D ou similares. Alguns exemplos comuns são: elfo, humano, anão, orc, morto-vivo, etc.
- ***Profession*** (optional): Profissão do personagem.
- **Equiped Itens**: Itens que podem ser equipados pelos personagens.
  - **Head**: Equipamentos que podem ser utilizados na cabeça, como elmos, tiaras, chapeus, máscaras, etc.
  - **Left Hand**: Equipamentos que podem ser utilizados na mão esquerda. Geralmente escudos.
  - **Right Hand**: Equipamentos que podem ser utilizados na mão direita. Geralmente armas de ataque, como espadas, martelos, machados, cajados, etc.
  - **Body**: Equipamentos que podem ser utilizados no corpo, como armaduras, capas, calças, etc.
  - ***Acessories*** (optional): Acessórios possuem uma enorme diversidade. Podem ser colares, brincos, anéis, entre outros.
- **Inventory**: Inventário do personagem, ou seja, lugar onde estão especificados os itens .

Battle System

Feature Model que descreve os possíveis sistemas de batalha de um jogo RPG.

Features

(18 mandatory, 1 optional, 23 under alternatives, 0 references, 43 total)

- **Battle System**: Sistema de batalha.
  - **Time System**: Sistema de temporização das batalhas.
    - Alternative [1..1]
      - **Real Time**: Batalhas em tempo real.
      - **Turns**: Batalha em turnos.
  - **Actions**: Ações possíveis no momento da batalha.
    - **Attack**: Ações relacionadas a ataque.

        - **Physical Attack**: Ataque físico, feito geralmente com armas.
        - **Magical Atack**: Ataque mágico.
        - **Range**: Ataques em relação ao alcance.
            - Alternative [1..1]
                - **Melee**: Ataque próximo.
                - **Ranged**: Ataque em distância.
        - **Target**: Quantidade de alvos.
            - Alternative [1..1]
                - **Multiple Targets**: Vários alvos atingidos.
                - **Single Target**: Um único alvo atingido.
                - **Area**: Ataques que atingem em área.
    - **Itens Use**: Utilização de itens, para recuperação de pontos de vida, de magia, entre outros.
    - **Skill**: Utilização de habilidades próprias de cada personagem.
        - **Targets**: Habilidades classificadas em relação a quantidade de alvos.
            - Alternative [1..1]
                - **Multiple**: Vários alvos.
                - **Single**: Apenas um alvo.
                - **Area.**: Ação em área.
                - **Self**: Permite ser utilizada no próprio personagem.
        - Alternative [1..1]
            - **Manual**: As habilidades podem ser controladas de modo manual, ou seja, o jogador escolhe quando quer usar.
            - **Automatic**: A habilidade é controlada automaticamente.
    - **Escape**: Escapar do combate.
    - ***Moviment System*** (optional): Sistema de movimentação das entidades no mapa no momento de batalha.
        - Alternative [1..1]
            - **Discrete**: Movimentação discreta.
            - **Continuous**: Movimentação contínua.
    - **Trigged**: Ações engatilhadas.

            - **Counter Attack**: Contra-ataque.
    - **Damage System**: Sistema que calcula os danos provocados por determinado ataque.
    - **Gains**: Ganhos obtidos na batalha.
        - Alternative [1..4]
            - **Gold**: Dinheiro (depende do sistema monetário vigente no jogo)
            - **Equipaments**: Equipamentos, como armas, armaduras, acessórios.
            - **Resource Item**: Itens de cura, recuperação de atributos, etc.
            - **Experience Points**: Pontos de Experiência.
    - **Iniciative System**: Sistema que define a iniciativa dos personagens.
        - **Visualization**: Modo de visualização da iniciativa.
            - Alternative [1..1]
                - **Bar**: Em barra de iniciativa.
                - **Value**: Valor de Iniciativa.
        - **Type**: Tipo de iniciativa.
            - Alternative [1..1]
                - **Static**: Iniciativa estática (sempre se mantém a mesma)
                - **Dynamic**: Iniciativa dinâmica.

Item System

Feature Model que descreve os itens encontrados no jogo.

Features

(9 mandatory, 7 optional, 13 under alternatives, 0 references, 30 total)

- **Item System**
    - **Equipable**: Itens equipáveis.
        - ***Acessories*** (optional): Acessórios.
        - **Armor**: Armaduras.
        - **Weapon**: Armas.
        - **Bonus**: Bonus oferecidos pelos equipamentos.
            - Alternative [1..1]
                - **Attribute Increase**: Ganho em atributos.

                - **Skill Increase**: Aumento na potência de habilidades.
                - **Skill Gain**: Ganho de habilidades.
        - ***Labor Instrument*** (optional): Instrumento de trabalho.
    - **Quest Itens**: Itens de quests.
    - ***Miscellaneous*** (optional)
        - ***Flavor Itens*** (optional): Itens que não apresentam utilidade específica.
    - **Resources**: Itens dos mais variados tipos, de grande utilidade para os personagens.
        - **Recover**: Itens de recuperação.
        - ***Cure Status*** (optional): Itens para cura de status alterados.
        - ***Revives*** (optional): Itens utilizados para ressucitar personagens.
        - **Buff**: Itens de buff.
            - Alternative [1..1]
                - **Attribute Increase Buff**: Buff para ganho em atributos.
                - **Magical Effect Buff**: Buff com efeito mágico.
                - **Skill Increase Buff**: Buff para ganho em habilidades.
        - ***Miscellaneous Resources*** (optional)
    - **Prerequisites**: Pré-requisitos para uso de determinados itens.
        - Alternative [1..1]
            - **Skill**: Perícia em determinada habilidade.
            - **Class**: Personagem pertencente a determinada classe.
            - **Attribute**: Personagem com valor de atributo igual ou maior ao determinado.
            - **Level**: Personagem com nível igual ou maior ao determinado.
            - **Profession**: Personagem com proficiência em determinada profissão.
            - **Race**: Personagem pertencente a determinada raça.
            - **Character**: Itens que apenas podem ser usados por um personagem específico.

Gameplay

Feature model que descreve as características mais peculiares do estilo RPG.

Features

(23 mandatory, 6 optional, 14 under alternatives, 0 references, 44 total)

- **GamePlay**
  - **Encounters**: Modo como ocorrem os encontros para eventuais combates.
    - Alternative [1..2]
      - **Random**: Encontros aleatórios. O jogador não visualiza os inimigos para evitá-los.
      - **Predefined**: Neste modo, o jogador pode ver os inimigos e optar por entrar em seu campo de visão.
  - **Map System**: Sistema que define mapas, visualização e movimentação.
    - **Locations**: Localidades.
      - **Dungeons**: Masmorras. Geralmente são locais onde existem um grupo estruturado de inimigos.
      - **Cities**: Cidades, vilas, casas, estabelecimentos, etc.
      - **Wilderness**: Desertos, florestas, campos, etc.
    - **Moviment**: Modo de movimentação.
      - Alternative [1..1]
        - **Restricted**: Restrita a lugares definidos.
        - **Free**: Livre, qualquer lugar do mapa pode ser desbravado.
    - **Visualization**: Modo de visualização.
      - Alternative [1..4]
        - **First Person**: Primeira pessoa.
        - **Third Person**: Terceira pessoa.
        - **Limited**: Visão limitada, com foco no local em que o jogador se encontra e áreas circunvizinhas.
        - **Entire**: O mapa é visto inteiramente.
    - **Elements**: Elementos presentes nos ambientes.
      - **Characters**: Personagens
        - Alternative [1..2]
          - **Playable**: Jogáveis.

        - **NPC**: Non-Player Characters. Apesar de serem geralmente não jogáveis, alguns jogos trazem NPCs que podem ser guiados pelos jogadores.
    - **Objects**: Objetos mais variados, como baús, mecanismos, mesas, etc.
- **Quests**: Tipos de quests.
    - Alternative [1..2]
        - **Obrigatory**: Quests que são obrigatórias para o sucesso no jogo.
        - **Sidequests**: Quests alternativas.
- **Group**: Grupo de personagens.
- **Currency**: Sistema monetário vigente.
- ***Social Organization*** (optional): Organização social. Geralmente ocorre em jogos mais complexos. Um exemplo muito comum são guildas.
- ***Transport Means*** (optional): Meios de transporte, como naves, navios, cavalos, etc.
- **Actions**: Ações permitidas ao jogador durante o jogo.
    - ***Skill use*** (optional): Usar habilidades.
    - ***Steal*** (optional): Roubar estabelecimentos.
    - **Attack**: Atacar inimigos.
    - **Move**: Movimentação.
    - **Shop**: Comprar itens.
    - ***Training*** (optional): Treinar habilidades ou profissão.
    - **Talking**: Conversar com outros personagens.
    - ***Working*** (optional): Trabalhar.
- **Victory Conditon**: Condição de vitória no jogo.
    - **Defeat Bosses**: Derrotar chefes.
    - **Complete Quests**: Completar todas as quests obrigatórias.
- **Death System**: Sistema que define o que poderá ocorrer dada a morte de um personagem.
    - Alternative [1..1]
        - **Revive**: Opção de reviver, através de itens, magias ou habilidades.
        - **Game Over**: Derrota.

Flow

Feature model que descreve o sistema de fluxo de telas.

Features

(9 mandatory, 3 optional, 4 under alternatives, 0 references, 17 total)

- **Flow**: Fluxo de telas.
    - **Screen**: Telas.
        - **Transition**: Modo de transição das telas.
            - **Exit Condition**: Condição para sair de determinada tela.
            - **Next Screen**: Passagem para a próxima tela.
        - ***Battle Mode*** (optional): Tela que indica a entrada em modo de batalha.
        - **Menu**: Telas de Menu.
        - **Map Navigation**: Telas de navegação no mapa do jogo.
        - **Info Display**: Telas de informação.
            - Alternative [1..4]
                - **Game Over**: Informação de derrota.
                - **Intro**: Introdução do jogo.
                - **History Development**: Desenvolvimento da história, com falas de personagens.
                - **Credits**: Créditos.
        - **Character Screen**: Tela do personagem. Apresenta geralmente informação de atributos, equipamentos, itens, etc.
        - ***Inventory Screen*** (optional): Tela do inventário do personagem.
        - ***Treasure Screen*** (optional): Tela que aparece durante a pilhagem de tesouros/itens.
        - **Save Game**: Tela para salvar o jogo.

Audio

Feature model que descreve o sistema de som do jogo.

Features

(4 mandatory, 2 optional, 0 under alternatives, 0 references, 7 total)

- **Audio**: Definição do sistema de áudio.
    - **Sound Effects**: Efeitos sonoros, como som de baús abrindo, passos, magias, etc.
        - **Resource**: Arquivo fonte do efeito.
    - **Background Music**: Música de fundo.
        - ***Loop*** (optional): A música de fundo pode se manter em um laço.
        - **Resources**: Arquivos fonte das músicas de fundo.
    - ***Voice*** (optional): Voz dos personagens, gravadas em arquivos fonte.

Physics

Feature Model que descreve a física presente em jogos RPG.

Features

(4 mandatory, 1 optional, 8 under alternatives, 0 references, 14 total)

- **Physics**: Definição da física.
    - **Collision Detection System**: Sistema de colisão entre os elementos do jogo, como entidades e cenário.
        - Alternative [1..1]
            - **2D**: Sistema de colisão de jogos em duas dimensões.
                - Alternative [1..1]
                    - **Tile Based**: Colisão baseada em tiles.
                    - **Pixel Based**: Colisão baseada em pixels.
            - **Isomeric**: Sistema de colisão em jogos isoméricos.
            - **3D**: Sistema de Colisão em jogos em três dimensões.
    - ***Lighting*** (optional): Iluminação do ambiente.
    - **Tile System**: Definição de como são representados os tiles, tanto quanto ao tipo quanto ao formato geométrico.
        - **Format**: Definição da geometria dos tiles.

        - **Type**: Tipos de tiles
            - Alternative [1..3]
                - **Transposable**: Tiles que podem ser transponíveis.
                - **Obstacle**: Obstáculos.
                - **Destroyable**: Tiles que podem ser destruídos.

Graphics

Feature model que descreve os gráficos do jogo.

Features

(7 mandatory, 6 optional, 5 under alternatives, 0 references, 19 total)

- **Graphics**: Gráficos do jogo.
    - **Display**: Forma como o display do jogo é mostrado.
        - Alternative [1..1]
            - **Windowed**
            - **Full Screen**
    - **Resolution**: Resolução em que o jogo será apresentado.
    - **HUD**: Heads-Up Displays que podem estar presentes no jogo.
        - ***Battle Display*** (optional): HUD para informações de batalha.
            - **Hit Points**: Pontos de vida dos personagens.
            - **Magic Points**: Pontos de Magia dos personagens.
            - ***Iniciative*** (optional): Iniciativa dos personagens.
        - ***Character Picture*** (optional): Foto do personagem.
        - ***Inventory*** (optional): HUD do inventário, mostrando os itens contidos.
        - ***Treasure*** (optional): HUD mostrando os tesouros pilhados.
        - **Actions Menu**: Menu de ações do personagem.
        - ***MiniMap*** (optional): Mapa em miniatura.
    - **Rendering System**: Sistema de renderização do jogo.
        - Alternative [1..1]
            - **2D**
            - **Isomeric**
            - **3D**

Platform

Feature model que mostra plataformas de programação para as quais os jogos são produzidos.

Features

(0 mandatory, 0 optional, 2 under alternatives, 0 references, 3 total)

- **Platform**: Plataformas em que os jogos podem ser executados.
  - Alternative [1..2]
    - **Console**: Console, conhecido como video-game.
    - **PC**: Computadores pessoais.

"Pouco se aprende com a vitória, mas muito com a derrota."
Provérbio Japonês