UNIVERSIDADE FEDERAL DE PERNAMBUCO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

"GEF – Game Editor Framework
Um framework para Editores de Níveis
para Jogos Móveis"

Por

WABBER MIRANDA DE ARRUDA FILHO

Janeiro de 2005

Arruda Filho, Wabber Miranda de
GEF-Game Editor Framework : um framework para editores de níveis para jogos móveis / Wabber Miranda de Arruda Filho. – Recife : O Autor, 2005.
123 folhas : il., fig., tab.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2005.

Inclui bibliografia.

1. Engenharia de software – Jogos. 2. Jogos móveis – Editores de níveis – Framework – Processo d criação. I. Título.

004.42	CDU (2.ed.)	UFPE
005.5	CDD (22.ed.)	BC2005-547

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

WABBER MIRANDA DE ARRUDA FILHO

"GEF – Game Editor Framework
Um framework para Editores de Níveis para Jogos Móveis"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR GEBER LISBOA RAMALHO

RECIFE, JANEIRO/2005

"Man is a tool-using animal... Without tools he is nothing, with tools he is all." - Thomas Carlyle

Resumo

Editores de Níveis para jogos eletrônicos são ferramentas importantes para desenvolvedores e usuários. O trabalho aqui exposto descreve um esforço original no intuito de construir um *framework* para Editores de Níveis, chamado de GEF (*Game Editor Framework*). Esta abordagem pode conciliar a necessidade de reaproveitamento de código, com a necessidade de se desenvolver um editor específico para cada jogo.

Abstract

Level Editors for games are important artifacts both for developers and users. However, they are tightly coupled with their specific games. This work describes an original effort to build a level editor framework, named GEF (Game Editor Framework). This approach can conciliate the need for code reuse, provided by the framework classes, with the need to implement specific editors for each game.

"*É melhor tentar e falhar, Que preocupar-se a ver a vida passar. É melhor tentar, ainda que em vão, Que sentar-se fazendo nada até o final.*" - *Martin Luther King*

Agradecimentos

Agradeço ao amigo Geber Ramalho, que além de tornar este trabalho possível, tem servido como um bom exemplo de profissional e de pessoa a ser seguido.

À minha noiva, Adriana Barbosa da Silva, que além de estar comigo em todos os momentos difíceis ocorridos no último ano, foi a maior revisora do texto aqui apresentado.

Ao estudante do Centro de Informática Alberto César França que implementou o editor para o jogo Pacman.

Aos desenvolvedores do CESAR, Alexandre Damasceno, Borje Karlsson, Igor Azevedo e Pedro Henrique, que implementaram os jogos Pacman e Pod Race e ao designer Diego Credidio, responsável pela interface gráfica do jogo Space Runner.

Ao site google, o maior guru do mundo. Sempre que uma dúvida surgia, bastava perguntar ao google, que o mesmo indicava quem sabia a resposta.

Obrigado pessoal!!!

Sumário

Índice de figuras

Índice de tabelas

1 Introdução

A indústria dos jogos eletrônicos consolidou-se como um forte ramo da economia mundial, chegando a ter receitas maiores que a indústria cinematográfica de Hollywood, por ano [36]. O avanço dos dispositivos móveis, como celulares, *palm tops*, PDAs (*Personal Digital Assistent*), *pagers* e outros, está permitindo que esta indústria faça uso destes a fim de ganhar ainda mais mercado e conquistar mais simpatizantes e adeptos aos seus produtos [16].

Neste contexto, um grande avanço foi a entrada no mercado da tecnologia Java 2 Micro Edition (J2ME[13]) que permite aos desenvolvedores implementarem jogos de forma mais independente da plataforma em que irão ser executados. Assim, pode-se utilizar o mesmo código para celulares de diferentes fabricantes ou até mesmo para diferentes dispositivos, como *palms*, PDAs, *pagers* e outros, bastando que estes dispositivos possuam uma Máquina Virtual Java[1] (JVM) implementada.

O Centro de Informática (CIn) [30] da Universidade Federal de Pernambuco (UFPE) junto com o Centro de Estudos e Sistemas Avançados do Recife (CESAR) [23] enxergaram no desenvolvimento de jogos um novo nicho de estudo e de mercado. A partir daí uma série de projetos foram desenvolvidos tanto para telefones móveis como para computadores pessoais objetivando minimizar o trabalho de desenvolvimento e aumentar a qualidade nos jogos construídos por alunos e empresas. Neste sentido foram desenvolvidos alguns *frameworks* [27][28][16] para a construção de jogos, também conhecidos como motores.

Uma ferramenta útil no desenvolvimento de jogos que ainda não tinha sido considerada é o Editor de Níveis. Esta é uma ferramenta desenvolvida para facilitar a construção e melhorar a qualidade dos níveis de um jogo. Um nível é o mesmo que uma missão, um estágio ou um mapa. Um exemplo de onde um editor como este poderia ser utilizado é no jogo BreakOut, em que o editor poderia criar facilmente um conjunto de estágios diferentes para o jogo modificando a disposição e quantidade de "tijolos"(Figura 1).

[1] A Máquina Virtual Java dos dispositivos móveis também são conhecidas como KVM (Kilobyte Virtual Machine).

Figura 1 - Níveis do jogo BreakOut

Uma vantagem em se utilizar Editores de Níveis para os jogos é a facilidade que os mesmos oferecem aos desenvolvedores no momento da criação e ajuste dos níveis que irão compor o jogo. Níveis bem criados e balanceados podem deixar o jogo mais interessante para o usuário final.

Um Editor de Níveis pode também ser usado diretamente pelo usuário final. Um fato observado é que por maior que seja o esforço em construir jogos divertidos e de boa qualidade, devido à repetição existente dos movimentos que o usuário realiza e dos cenários que são visualizados, com o tempo o jogador acaba cansando-se de jogá-lo [1]. Ao lançar mão de um Editor de Níveis pode-se também evitar que este tempo seja muito curto, pois a introdução do mesmo fornece ao jogador (usuário final) a possibilidade de personalizar o ambiente e a dificuldade do jogo, proporcionando assim maior divertimento. De fato, com um Editor de Níveis o usuário poderia jogar em um cenário de sua própria autoria, podendo assim, aumentar o chamado "*replay value*" de um jogo, ou seja, o interesse deste em jogar o mesmo jogo diversas vezes.

Enfim, o uso de Editores de Níveis pode ser também útil para a criação de comunidades de jogadores. De fato, um jogador pode desafiar colegas a superar os níveis criados por ele. A interação entre pessoas ao redor de novos níveis do jogo ajuda a criar comunidades de jogadores, algo que pode ser extremamente interessante e lucrativo para a indústria de jogos.

A criação de um Editor de Níveis genérico, que contenha todos os requisitos necessários, é uma tarefa extremamente complicada, devido ao forte acoplamento existente entre um jogo e seu editor, afinal cada jogo eletrônico possui suas próprias

características, funcionalidades, regras, peculiaridades, etc. Isto pode ser melhor visualizado quando se tenta imaginar o mesmo Editor de Níveis sendo utilizado para jogos de estilos diferentes como o de estratégia, Age of Empires[5], o de tiro, Counter Strike[41], ou mesmo o de corrida, GP3[3]. Por esta razão, cada jogo normalmente possui o seu próprio Editor de Níveis.

Infelizmente Editores de Níveis necessitam de um tempo de desenvolvimento razoável e este tempo se repete a cada jogo produzido, gerando um alto custo para a indústria. Pensando nisto, este trabalho propõe uma solução, até onde se sabe original, no intuito de diminuir este custo que é a elaboração de um *framework* para desenvolvimento de Editores de Níveis voltados para alguns estilos de jogos. Este *framework* realizaria tarefas fundamentais de Editores de Níveis e seria re-utilizado para a construção de editores específicos, usando uma arquitetura bem definida, organizada e facilitando o trabalho de desenvolvimento. Com o uso deste *framework* desenvolvedores podem aumentar sua produtividade, diminuindo o tempo de construção de um novo editor para cada jogo e também a qualidade dos editores desenvolvidos.

O *framework* proposto deve ter duas características essências para ajudar os desenvolvedores. A primeira é cobrir uma quantidade relevante de requisitos essências a um editor e a segunda é diminuir de forma considerável a quantidade de código que o desenvolvedor precisará escrever, depurar e testar.

Como o CIn e o CESAR vêm se destacando na produção de jogos para dispositivos móveis, o *framework* foi elaborado para este tipo de jogo. Para realizar este trabalho, foram seguidas várias etapas. Primeiramente foi realizado um estudo na literatura especializada, com o intuito de descobrir quais são os requisitos essenciais de um editor. Como pouco material sobre isto foi encontrado, foi realizado um extenso estudo de caso envolvendo editores de jogos específicos, disponíveis no mercado a fim de fazer um levantamento das principais características e requisitos implementados pelos mesmos. Este levantamento de requisitos acabou atribuindo mais originalidade a este trabalho.

O passo seguinte foi estudar o que são *frameworks*, como concebê-los, quais os benefícios que eles poderiam trazer e quais as dificuldades em usar os mesmos. Seguindo uma metodologia recomendada na literatura sobre *frameworks* [24], foram

desenvolvidos três editores de cenários para jogos específicos. Em seguida, foi possível abstrair o *framework* denominado de GEF-*Game Editor Framework*. Com o GEF elaborado, dois dos editores foram re-construídos com o intuito de validar a execução correta dos seus métodos e os critérios de sucesso propostos anteriormente.

No próximo capítulo, será abordado o editor como ferramenta auxiliar ao desenvolvimento de jogos. Em seguida, será feita uma contextualização sobre o que são *frameworks* e as vantagens e desvantagens da sua utilização. No capítulo 4 será exibido o estudo realizado em alguns editores disponíveis no mercado e uma lista de requisitos extraída do mesmo. O capítulo 5 trás o estudo de casos feito para a elaboração deste trabalho. No capítulo 6 será apresentado o *Game Editor Framework*, obtido do estudo casos realizado, seus pacotes e estruturação. O capítulo 7 trás as conclusões do trabalho e sugestões para trabalhos futuros.

2 Editor de Níveis

Este capítulo descreve na primeira seção, o que é o balanceamento de um jogo. A seção 2.2 comenta o papel de um Editor de Níveis, a seção 2.3 fala um pouco sobre um Editor de Mapa disponível no mercado e a última seção mostra por que editor e jogo são *softwares* fortemente acoplados.

2.1 Balanceamento de um Jogo

Uma das tarefas fundamentais dos profissionais especializados em criar níveis de jogos eletrônicos, os chamados *level designers*, é ajustar ou balancear os níveis que compõem um jogo.

A construção de um bom Editor de Níveis permite ao *level designer* ter tempo suficiente para investir no refinamento dos níveis de um jogo. Grande parte do sucesso de um jogo depende do quanto este foi testado e balanceado para se atingir uma boa jogabilidade [35]. Balancear um jogo é um processo bastante repetitivo que inclui testá-lo e modificá-lo inúmeras vezes com o intuito de melhorar seu funcionamento. Quanto mais fácil for modificar e testar cada nível, mais vezes o *designer* o fará, deixando o jogo cada vez melhor. A figura abaixo ilustra este processo (Figura 2).

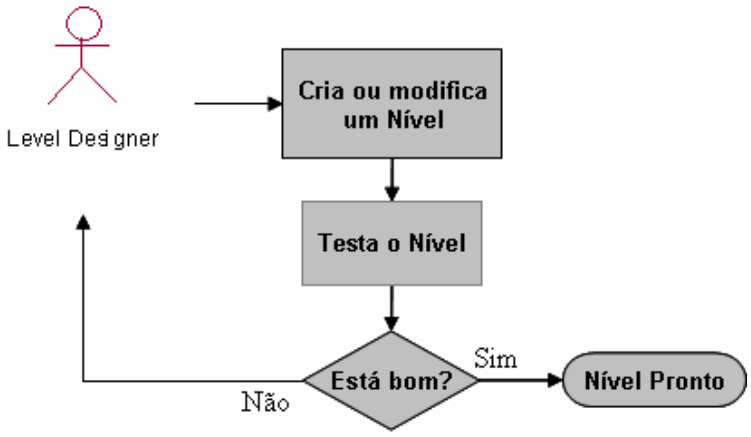


Figura 2 - Processo de balanceamento de um nível

Um fator fundamental em jogos é manter o jogador concentrado. Um alto nível de concentração ocorre quando a mente é submetida ao seu limite em um esforço voluntário para acompanhar algo complexo e interessante [43].

Para que um jogo obtenha sucesso é necessário fazer um esforço no intuito de evitar que o jogador se desinteresse deste Existem dois conceitos fundamentais que são utilizados para evitar tal desinteresse [42]. O primeiro é não construir níveis difíceis demais, ou seja, que não possam ser superados, pois isso certamente frustraria o usuário. O segundo conceito é não fazer níveis fáceis demais, deixando assim, o jogador entediado.

Com isso, inicialmente poder-se-ia sugerir que a dificuldade do jogo deva ser incrementada de forma linear como visto na linha pontilhada do gráfico abaixo (Figura 3). Esta linha representa o que é conhecido como "melhor lógico" [42], pois a proporção dificuldade versus habilidade adquirida pelo jogador, seria, em teoria, ideal durante todo o jogo.

Contudo, quando se trata de jogos eletrônicos existe ainda uma terceira possibilidade de entediar o jogador, que é a previsibilidade causada pela a falta de variância [42]. Com isso, a linha pontilhada deixa de ser a solução ideal, o fato é que as pessoas preferem que a dificuldade do jogo seja incrementada de forma não previsível. Isto faz com que a curva indicada no gráfico como "melhor real" seja ideal, quando se fala de incremento de dificuldade em jogos.

O que ocorre na verdade é que, a taxa de dificuldade deve ser incrementada alternando períodos de tensão, que exijam uma maior concentração do usuário, como pode ser observado em "A" no gráfico, seguidos de momentos onde o jogador possa relaxar um pouco, como em "B" no mesmo gráfico.

Além de se preocupar com a dificuldade de superar cada nível é importante também prover ao jogador uma boa quantidade de variantes do jogo, exigindo do mesmo criatividade para lidar com as novas situações deste e ao mesmo tempo deixando-o interessado em saber o que pode acontecer nos próximos estágios. Para cumprir esta tarefa, boas estratégias são: manter uma boa estória e fazer com que novos elementos do jogo surjam, de acordo com a evolução do usuário no mesmo.

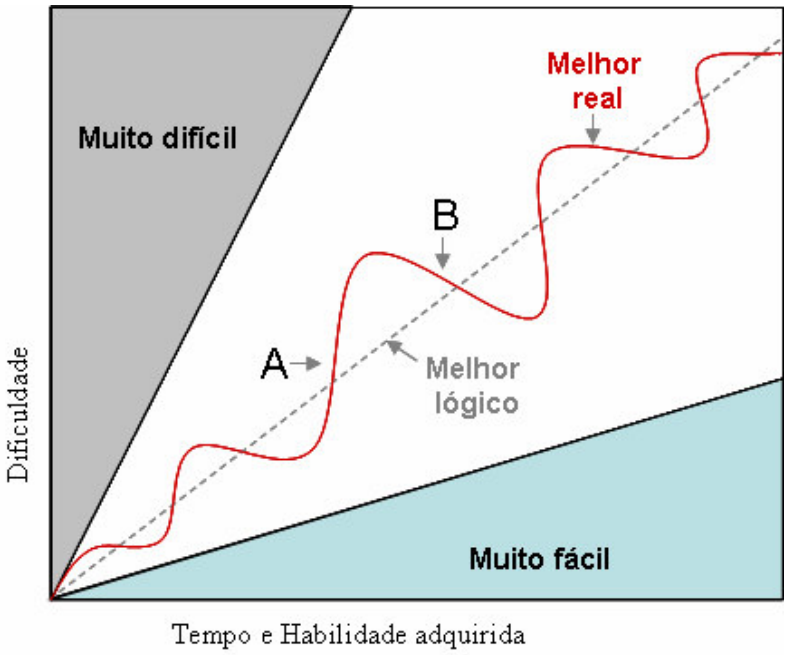


Figura 3 - Comparação entre o melhor lógico e real

2.2 Papel do Editor de Níveis

O papel fundamental do Editor de Níveis é ser uma ferramenta que ajude a construir e melhorar a qualidade dos níveis de um jogo. O editor pode ser uma ferramenta acoplada aos jogos, ou uma aplicação à parte.

Uma característica importante de um Editor de Níveis é permitir que o usuário possa ver o mundo que está sendo desenvolvido, enquanto o cria ou o modifica. Isto é freqüentemente chamado de "*What You See is What You Get"* (WYSIWYG)[1] – o que você vê é o que você consegue – pode ser chamado, também, de visão do jogador, pois é o que o mesmo verá quando estiver jogando.

Para o jogador, o papel do Editor de Níveis é permitir que ele crie cenários alternativos para o seu jogo, aumentando assim seu interesse pelo mesmo. Existem casos de jogos para computadores pessoais que, mesmo depois de vários anos no mercado, ainda despertam o interesse de milhares de jogadores por todo o mundo. Este sucesso deve-se,

em parte, à presença de um editor poderoso, que permite que os usuários criem novas situações de jogo. Entre estes jogos, pode-se citar *Age of Empires* [5] e *Starcraft* [7].

Um exemplo da dificuldade gerada pela falta de um editor pode ser visto no documento de *Postmortem*[2] do jogo *Tropico* [29]. Seus criadores assumem o erro de não ter dedicado mais tempo no desenvolvimento do seu Editor de Níveis durante a etapa de desenvolvimento, fato este que implicou no uso de mapas aleatórios para criar o desenho original do jogo. O editor só foi concluído depois do lançamento do jogo e isto acabou decepcionando os fãs do *Tropico* que gostariam de criar seus próprios cenários [29].

2.3 Mappy Editor

Uma outra ferramenta que pode auxiliar desenvolvedores de jogos é o Editor de Mapas. Sua funcionalidade é baseada no desenho de mapas de ladrilhos, conhecidos e como *tiles*[3]. Um exemplo é o Mappy Editor [40]. Este tipo de editor cobre parte das funcionalidades de um Editor de Níveis e são utilizados por desenvolvedores de jogos que não possuem um editor próprio para o seu jogo.

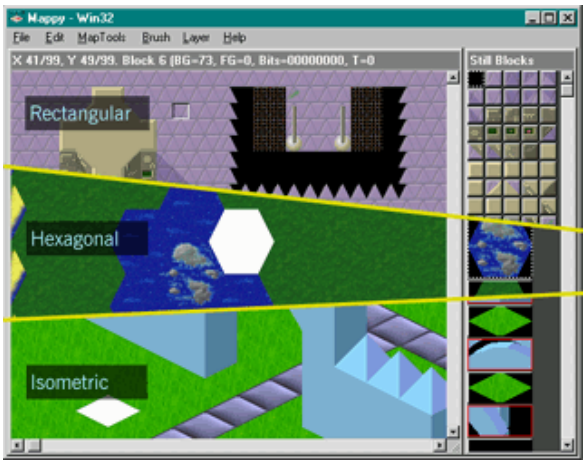


Figura 4 - Mappy

[2] Postmortem - documento que reflete o pensamento da equipe após o fim de um projeto

[3] Tiles - pequenas "partes" que compõe um cenário, normalmente quadrados ou triângulos

O Mappy Editor tem a função de ser um utilitário para criar mapas 2D e 3D para jogos baseados em ladrilho. Ao utilizá-lo, a primeira coisa a fazer é definir o tamanho de cada ladrilho e quantos ladrilhos serão aceitos nas posições horizontal e vertical. O próximo passo é importar uma imagem que contenha todos os componentes do jogo e então criar o mapa desejado (Figura 5).

Uma das primeiras coisas a se observar é que o Editor de Mapas não permite que uma imagem de fundo seja adicionada ao cenário, podendo assim a imagem exibida no editor ser diferente da que será visualizada no jogo. O editor também não permite que um conjunto de mapas seja salvo, forçando que cada mapa tenha que ser salvo, em um arquivo diferente.

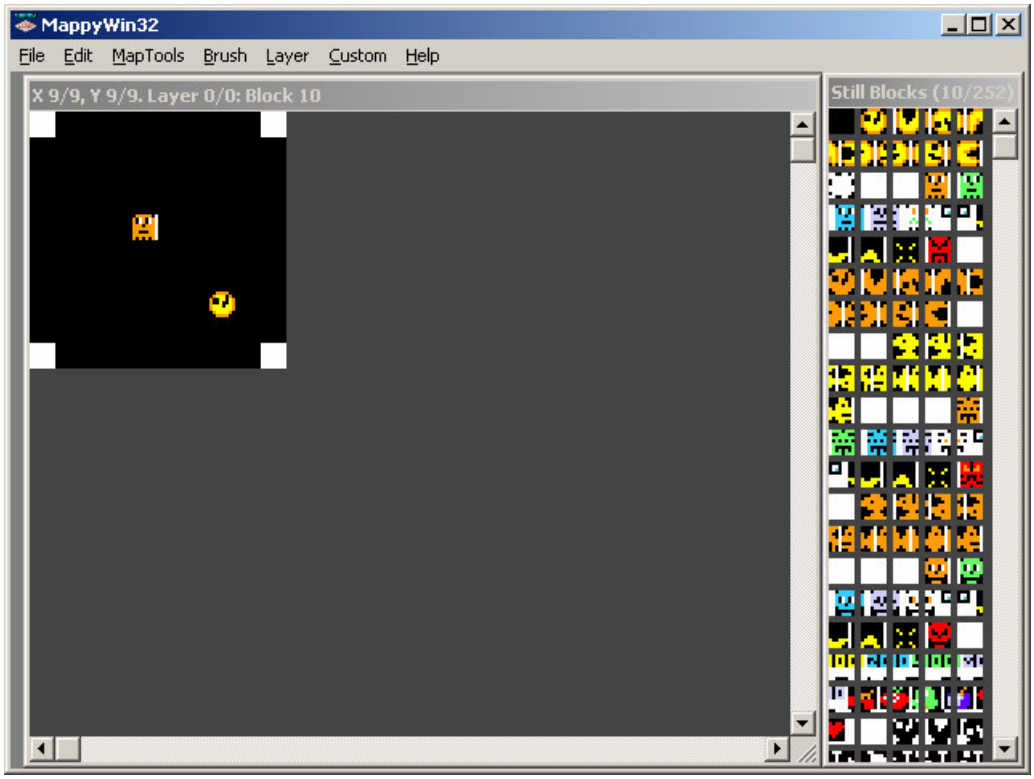


Figura 5 - Criando o mapa

Apesar do formato em que o mapa é salvo ser específico para o Mappy, o editor permite que os mapas sejam exportados para diferentes tipos de formato: MAR, TXT, CSV, BMP, CMA, ABD e GFX. Abaixo segue um exemplo de exportação em arquivo texto (Tabela 1).

```
const short mapa02_map1[10][10] = {
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 10 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 10, 10, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 13, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 10, 0, 0, 0, 0, 0, 0, 0, 0, 10 }
};
```

Tabela 1 - Arquivo texto exportado

O trabalho deste tipo de ferramenta se encerra na exportação do mapa para algum tipo de arquivo, depois disto todo o trabalho de programação, customização e adaptação ao jogo deve ser feito pelo desenvolvedor. Fato este que inviabiliza o uso deste tipo de ferramenta pelo usuário final.

Em síntese, este tipo de ferramenta é útil na elaboração de mapas para jogos em que as equipes não estão dispostas a investir em um Editor de Níveis. Ao contrário dos Editores de Níveis os Editores de Mapas perdem sua utilidade uma vez que o jogo está pronto e seus mapas concluídos.

2.4 Acoplamento Jogo - Editor

Em geral os Editores de Níveis nascem atrelados aos jogos como uma ferramenta secundária, fato este que gera um forte acoplamento entre esta ferramenta e o jogo para o qual ela foi produzida.

Durante o início deste trabalho foi pesquisado se existia no mercado um Editor de Níveis que pudesse ser utilizado para qualquer jogo. No entanto, nenhuma ferramenta com estas características foi encontrada. Isto pode ser justificado pela falta de uma

documentação na literatura sobre o problema do acoplamento e pelas dificuldades encontradas em se desenvolver este tipo de ferramenta. Os itens a seguir descrevem algumas destas dificuldades:

- Construir uma semântica, representação lógica, para representar níveis para qualquer tipo de jogo é uma tarefa complicada. Equipes de desenvolvimento diferentes podem ter visões diferentes de como representar os níveis. Ocorrendo também o fato de uma mesma equipe utilizar semânticas distintas para diferentes estilos de jogos;

- Construir algoritmos genéricos para validar os níveis criados também é uma tarefa difícil, pois enquanto jogos como *BreakOut* tem regras de validação extremamente simples, como a de possuir ao menos um tijolo inserido em cada nível, jogos mais rebuscados podem exigir regras de maior complexidade, como saber se um nível pode ser vencido pelo usuário;

- Alguns jogos possuem ainda os chamados NPCs (*Non-Player Characters*), ou seja, personagens do jogo que são controlados pelo computador. Um jogo pode conter vários tipos de NPCs com comportamentos diferentes. Construir uma ferramenta onde se possa modelar o comportamento de cada NPC pode ser uma tarefa não trivial e estritamente interligada ao jogo;

- Uma outra dificuldade é construir uma interface gráfica adaptável para representar os diferentes tipos de objetos gráficos como: imagens, formas geométricas, imagens animadas, etc, que sirva para diferentes estilos de jogos.

Além destas dificuldades, foi observado que algumas mudanças nos jogos requisitavam mudanças no editor e algumas mudanças no mesmo acarretavam modificações nos jogos. Um simples exemplo disto seria a modificação do tamanho dos ladrilhos utilizados ou mudanças nas regras de validação de um nível.

O próximo capítulo apresenta a solução proposta para resolver o problema de aperfeiçoar a construção de Editores de Níveis para jogos, uma contextualização do que

são *frameworks* e como os mesmos devem ser concebidos, e também a metodologia de trabalho utilizada.

3 Abordagem Proposta

O Desenvolvimento de software tem sido largamente motivado pela questão de ajudar desenvolvedores a produzi-lo de forma mais rápida e com maior qualidade para o usuário final. Com este objetivo, pode-se lançar mão de um *framework*, pois ele reúne o conhecimento de programação necessário para resolver problemas de um domínio particular [26].

A próxima seção comenta a idéia de utilizar *frameworks* para Editores de Níveis, a seção 3.2 explica o que são *frameworks* e como concebê-los, por fim a última seção descreve a metodologia utilizada para a elaboração deste trabalho.

3.1 Um Framework para Editores de Níveis

Devido ao forte acoplamento entre jogo e editor, decidiu-se encontrar uma alternativa para facilitar a construção de editores de jogos. A alternativa encontrada foi a elaboração de um *framework* para desenvolvimento de Editores de Níveis. A idéia é: ao invés de possuir um editor genérico que sirva para todos os jogos, possuir um *framework* que ajude a construir um editor para cada jogo (Figura 6).

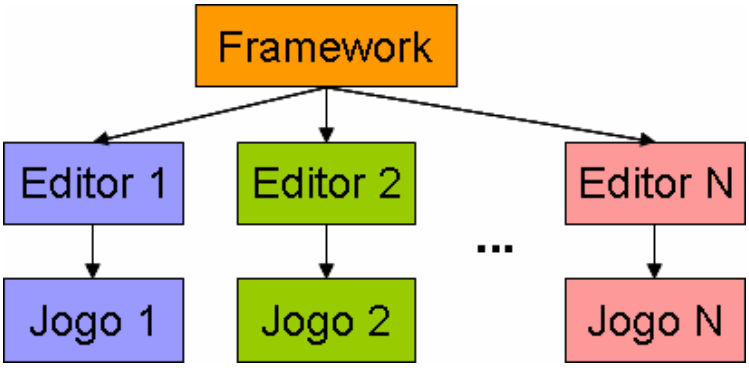


Figura 6 - Utilizar um framework para contruir editores

Para elaborar este *framework* foi necessário fazer um estudo sobre o que são *frameworks*, como concebê-los, quais os benefícios que os mesmo poderiam trazer e quais as dificuldades em utilizá-los.

3.2 Frameworks

Robert e Johnson [24] definem *frameworks* como um projeto re-usável por completo ou de uma parte do *software,* descrito por um conjunto de classes abstratas e uma forma de instanciar classes que colaboram para o desenvolvimento do sistema. Um *framework* orientado a objetos é um tipo de biblioteca de classes, mas ao contrário de uma biblioteca, o fluxo de controle é bi-direcional, ou seja, o código escrito pelo usuário pode chamar métodos do *framework* e o *framework* pode chamar o código escrito pelo usuário.

Uma boa definição também é dizer que um *framework* típico é um esqueleto extensível que provê funcionalidades básicas em algum domínio específico de problema. Os desenvolvedores de aplicação preenchem a necessidade de funcionalidade extra, estendendo as funcionalidades do *framework*. Isto é feito derivando novas classes das já existentes e sobre-escrevendo métodos, e/ou introduzindo combinações diferentes de objetos [26]. Uma forma gráfica de visualizar isto pode ser visto abaixo (Figura 7).

Johnson chega a dizer que o reuso de larga escala de bibliotecas orientada a objetos requer um *framework,* pois este provê um contexto para componentes de uma biblioteca para serem reusados. Apesar disto, pode-se notar que algumas bibliotecas comportam-se como *frameworks* e alguns deles podem ser usados como bibliotecas. Isto pode ser visto como uma continuação, com classes de bibliotecas tradicionais de um lado e sofisticados *frameworks* do outro [26].

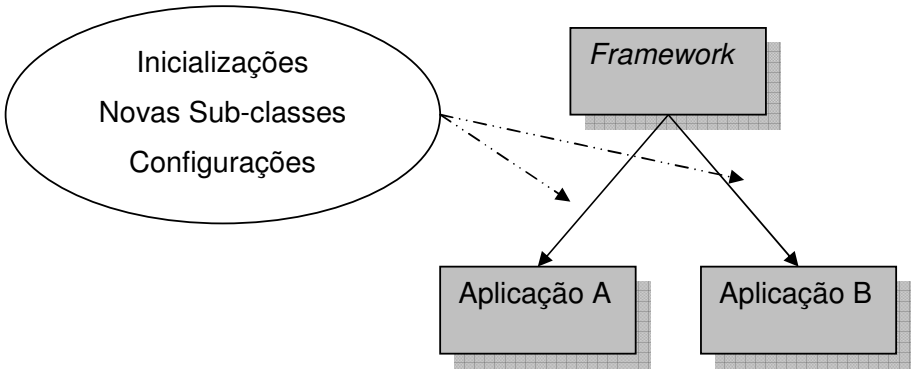


Figura 7 - Um framework e duas aplicações instanciadas

Podem-se dividir os *Frameworks* em caixa-branca e caixa-preta. Os *frameworks* caixa-branca são baseados em herança e podem ser difíceis de utilizar, pois em geral requerem a escrita de uma quantidade substancial de código para produzir o comportamento desejado. Os *frameworks* caixa-preta são baseados em composição de objetos e por isto, podem ser mais limitados. Um *framework* tem mais valor quando ele pré-implementa a parte mais complexa do domínio do problema.

Pela definição dada, é possível concluir que com o uso de *frameworks,* os desenvolvedores podem focar suas atenções em problemas particulares do seu domínio, ao invés de se preocuparem com todos os detalhes do programa. Através do uso de *frameworks* o tamanho do código escrito, testado e depurado pelo desenvolvedor pode diminuir significativamente.

O fato é que *Frameworks* armazenam experiência, pois os problemas são resolvidos uma vez e as regras de negócio são usadas de forma mais consistente. Isto permite que o trabalho de desenvolvimento comece a partir de uma base mais sólida.

Uma outra vantagem é que parte da análise, incluindo os requisitos, os algoritmos e as estruturas de dados utilizadas são reutilizados a cada aplicação construída. Isto facilita o processo de desenvolvimento e melhora a qualidade do software, pois muito do trabalho já foi pensado, realizado e testado.

Um outro benefício dos *frameworks* é que eles ajudam na coordenação de pessoas que trabalhem no mesmo projeto, pois através do uso de componentes é fornecida uma boa maneira dividir o trabalho que precisa ser feito. Com o uso de um *framework* também é possível desenvolver de forma mais rápida um protótipo do produto desejado, pois parte do sistema já está desenvolvido.

Em síntese alguns dos benefícios em utilizar *frameworks* são:

- Desenvolvedores podem focar suas atenções em problemas particulares do seu domínio.
- *Frameworks* diminuem o tamanho do código que o desenvolvedor precisa: codificar, testar e depurar.
- *Frameworks* armazenam experiência.
- Aplicações criadas com o mesmo *framework* possuem uma manutensão mais fácil.
- *Frameworks* ajudam a coordenar pessoas trabalhando no mesmo projeto.
- *Frameworks* permitem rápida prototipagem.
- Parte da análise de projeto é reutilizada.
- Algoritmos, estrutura de dados, e o projeto de *frameworks* são todos reusáveis.

Para se criar um *framework* Roberts e Johnson [24][51] aconselham que primeiramente se construam três aplicações, onde se acredite que o uso do *framework* ajudaria na elaboração delas. A idéia é baseada no fato que *frameworks* são abstrações, e é mais fácil abstrair algo de exemplos concretos do que tentar começar a partir de algo já abstrato. Além disto, a chance de sucesso aumentará, pois o desenvolvedor já passou pela experiência de construir aplicações reais. De fato, espera-se que, ao construir as três aplicações pertencentes ao mesmo domínio de problemas, as abstrações comuns se tornem aparente e então possam ser retiradas as regras que serão necessárias para a criação do *framework.*

Roberts e Johnson apresentam oito passos para a elaboração de *framework*, mas afirmam que não é necessário seguir todos eles para se ter um *framework* [51]. Na maioria dos casos seguir apenas as primeiras etapas já é suficiente. Os passos descritos por Roberts e Johnson são:

- Implementação de três exemplos;
- Elaboração do *framework* caixa-branca;
- Elaboração de uma biblioteca de componentes;
- Identificações de *Hot-Spots*;
- Adição de objetos conectáveis;
- *Refactoring* da biblioteca de componentes;
- Elaboração de um *framework* caixa-preta;
- Construção de uma ferramenta visual para trabalhar com o *framework* criado;
- Criação de linguagem para trabalhar com o *framework.*

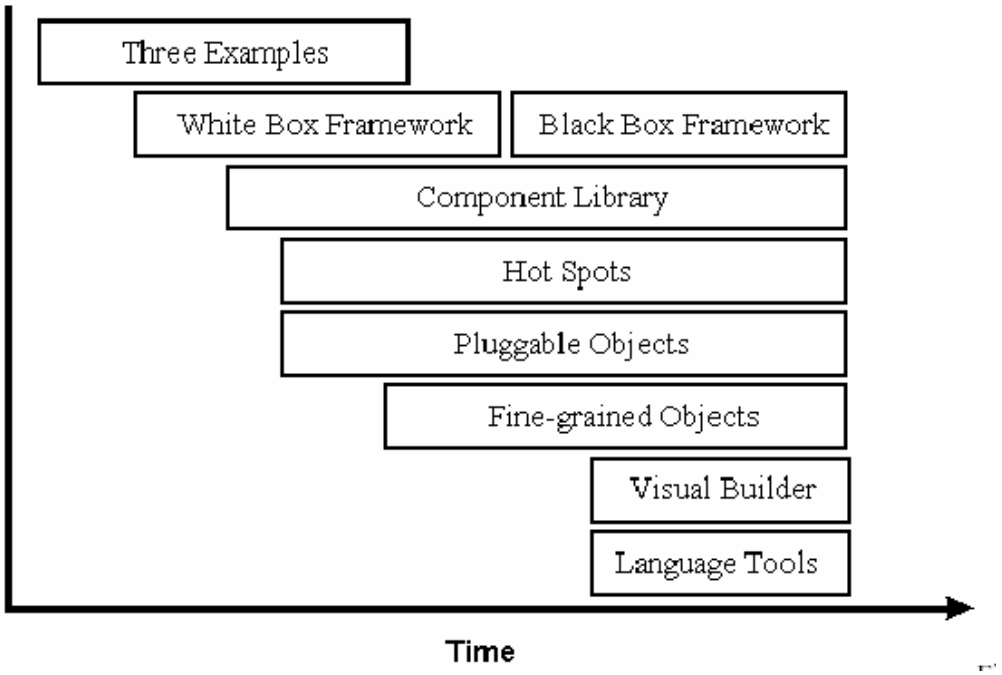


Figura 8 – Relação entre os passos para se contruir um framework x tempo

Infelizmente, desenvolver elementos genéricos no mundo da computação é um verdadeiro desafio, diante da enorme gama de peculiaridades e exceções existentes, tanto no mundo real quanto no mundo virtual. Para elaborar um *framework* é necessário experiência e conhecimento em desenvolvimento de aplicações de um domínio específico. Obter esta experiência e conhecimento é um processo demorado, complexo e geralmente caro. Uma recomendação feita no intuito de diminuir o custo de produção de

framework é aproveitar aplicações reais, que já seriam desenvolvidas, para conceber os mesmos [51].

Uma outra dificuldade é encontrar o chamados "*hot spots*" [26], pontos de entradas específicos do domínio. Estes são lugares onde o *framework* será estendido pelos desenvolvedores da aplicação. Identificar os "*hot spots"* é uma tarefa não trivial, pois exige cuidadosa observação, parametrização e documentação por parte dos desenvolvedores.

Mesmo com uma boa metodologia para a construção de um *framework,* o uso do mesmo pelos desenvolvedores ainda exigirá uma curva de aprendizado, pois aqueles que estiverem dispostos a fazer uso deste precisarão saber qual sua utilidade, como estendê-lo e como utilizá-lo. Devido a isso, muitas vezes é mais simples desenvolver um *software* sem auxílio de um *framework*, pois o tempo utilizado no aprendizado pode ser bastante significativo em relação ao tempo total utilizado no desenvolvimento do sistema.

Outro fator negativo em relação ao uso de *frameworks* é que toda ferramenta desta espécie necessita de atualizações constantes em função de sua natureza. Para isso, é necessário que a equipe responsável pelo desenvolvimento deste, trabalhe sempre de forma constante e continua, para que as novas versões do *framework* não contenham erros, e não sejam incompatíveis com os softwares já construídos.

3.3 Metodologia

O estudo sobre *frameworks* foi fundamental para se chegar à conclusão que seria necessário construir ao menos três casos de estudo no intuito de conceber o produto desejado. O estudo também deu uma idéia real que, mesmo após a elaboração do *framework*, não se deve esperar que todos os problemas estejam resolvidos. Faz-se necessário ainda, que o mesmo seja utilizado pelos seus usuários, ou seja, os desenvolvedores, para que ele sofra as devidas críticas, sugestões e alterações.

Como a literatura especializada é muita restrita em relação aos Editores de Níveis e não define um conjunto de requisitos ou características básicas para se construir um editor, foi realizado um extenso estudo envolvendo editores de alguns jogos disponíveis no mercado a fim de faze um levantamento das principais características e requisitos providos pelos mesmos. Um fruto deste trabalho foi o artigo "Requisitos Funcionais para Editores de Cenário 2D e 2½D" [32].

Depois de realizado este estudo foi decidido que seria construído um jogo em J2ME com um Editor de Níveis acoplado ao mesmo, com o intuito conhecer mais sobre o desenvolvimento de jogos e como seria a experiência de possuir um editor acoplado a um jogo feito para dispositivos móveis. O desenvolvimento deste jogo ajudou também na elaboração do artigo "Implementando Jogos J2ME: Problemas & Soluções" [48].

Terminado o jogo, foi seguido o que é recomendado pela literatura de *frameworks* [24]. Três Editores de Níveis foram desenvolvidos e testados antes de se abstrair o *framework* desejado. O mesmo recebeu o nome de GEF - *Game Editor Framework* e como forma de validação dois dos editores foram reescritos, no intuito de corrigir falhas em qualquer método provido pelo *framework*.

O próximo capítulo visa mostrar um pouco do estudo feito em alguns editores disponíveis no mercado e a lista de requisitos extraída deste estudo.

4 Levantamento de Requisitos

Este capítulo visa mostrar um pouco do estudo feito em alguns editores disponíveis no mercado e a lista de requisitos extraída deste estudo. O estudo começou a ser feito com editores mais simples e de preferência 2D, depois ele evolui para alguns jogos isométricos e 3D para se ter uma idéia do que poderia ser feito em um editor.

As seções 4.1 a 4.6 comentam sobre editores específicos existentes no mercado e a seção 4.7 apresenta as características e requisitos obtidos através deste estudo.

4.1 Come–Come II do *Odyssey*

O Come-Come II do *Odyssey*[12] não chega a ter um Editor de Níveis, todavia ele apresenta uma característica interessante para sua época. Ele já se preocupava em permitir que o usuário pudesse selecionar o labirinto em que desejasse jogar.

Ao entrar no jogo, o usuário pode pressionar a tecla "P" e se dirigir a um ambiente (Figura 9) que lhe apresenta uma espécie de matriz de opções de labirintos, como o visto abaixo:

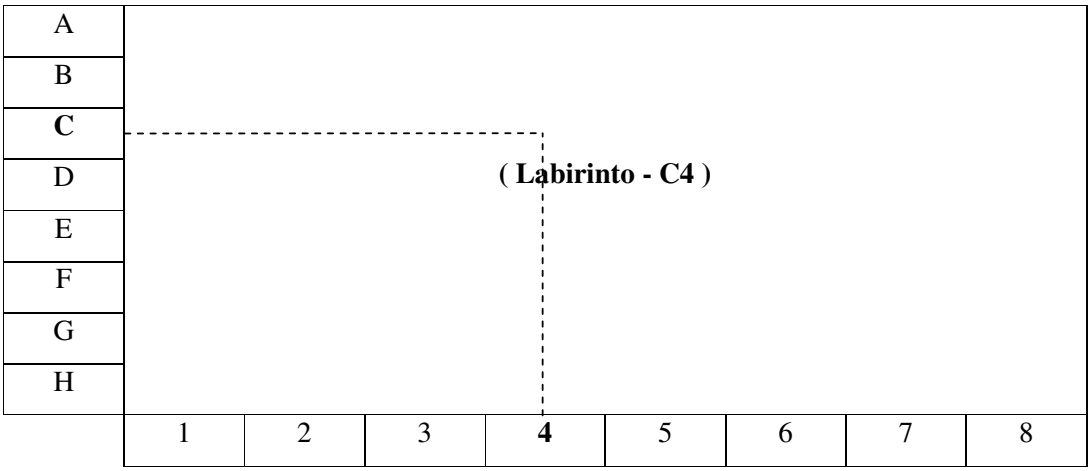


Figura 9 - Tela de Configurações do Come-Come II

Fazendo uma combinação de letras e números o labirinto exibido na tela é modificado. Este tipo de opção dada ao usuário é o primeiro passo para a criação de editores de cenário.

4.2 *Laser Tank*

Laser Tank[2] é um jogo *puzzle*, espécie de quebra-cabeça, com o objetivo de ser um *HelpWare*[4], implementado pela JEK Software, para que usuários possam criar seus próprios cenários e posteriormente enviar para o autor, Jim Kindley, afim que o mesmo possa incrementar o jogo com bons estágios. O jogo é constituído de um tanque capaz de atirar lasers e tem como objetivo, alcançar uma bandeira em cada estágio (Figura 10).

Figura 10 - Laser Tank 4.0

Como o próprio autor do jogo sugere que os usuários construam seus estágios e os enviem para a JEK Software, o jogo vem acompanhado de um Editor de Níveis bastante simples e de fácil utilização.

O jogo tem a característica peculiar de permitir que o usuário entre no editor e altere o estágio em que está jogando a qualquer momento, o que deixa o usuário bastante livre para balancear a dificuldade do estágio corrente.

O editor do *Laser Tank* (Figura 11) possui uma paleta de componentes, alguns campos para informações adicionais e um menu que permite operações básicas como salvar, carregar, salvar como, etc.

A paleta de componentes possui os ladrilhos do jogo incluindo o tanque, que o usuário controla, a bandeira que ele deve alcançar e um ladrilho idêntico ao *background* do jogo que serve para apagar um ladrilho indesejado.

[4] Helpware significa que se pode usar o que for aprendido; no caso do jogo o autor pede para que os níveis criados com o editor sejam enviados para o mesmo.

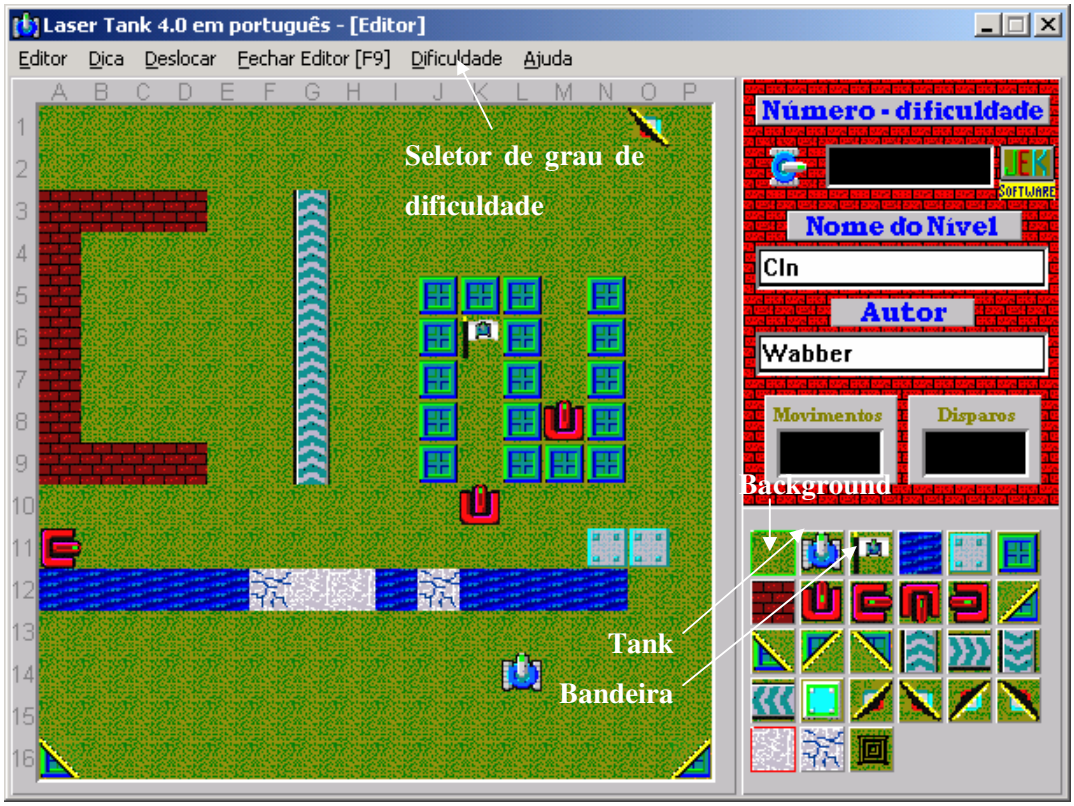


Figura 11 - Laser Tank 4.0 [Editor]

O editor também permite que o usuário associe alguns atributos ao nível que está sendo criado; como nome do autor, nome do nível criado e grau de dificuldade associada ao nível. Além disto, existe um menu que provê as seguintes operações:

- **Apagar** – Apaga todo o cenário construído.
- **Carregar Nível** – Carrega no editor um nível salvo anteriormente.
- **Salvar** – Salva o nível que está sendo editado em arquivo.
- **Salvar Como** – Salva o nível com outro nome.
- **Deslocar (direita, esquerda, acima, abaixo) –** Desloca todos os ladrilhos na direção indicada.
- **Dica –** Permite ao criador do cenário armazenar uma dica de como vencer o cenário, que poderá ser vista pelos jogadores daquele cenário.

O editor não faz nenhuma avaliação se o nível criado pode ser completado, ou seja, ele não faz a verificação se o tanque foi inserido no cenário, se existe ao menos uma bandeira no mesmo ou se é possível para o tanque alcançar a bandeira. Porém quando o

usuário pede para jogar em um nível criado sem o tanque, o jogo se encarrega de inseri-lo em uma posição inicial.

4.3 Lode Runner

O jogo *Lode Runner*[4] "*The Legend Return*" da Presage Software Development Company (Sierra) é um jogo de ação/*puzzle* onde o objetivo é resgatar todas as pedras de ouro de cada estágio e evitar os monges malucos. Os monges têm o poder de comer o herói do jogo, Jake Peril ou Wes Reckless, e se isso acontecer, uma vida do jogador será perdida.

Este jogo possui um ótimo Editor de Níveis (Figura 12) para ser avaliado neste documento, pois ele consegue reunir várias características básicas de editores de cenário como persistência, criação de mapas, barra de componentes, etc.

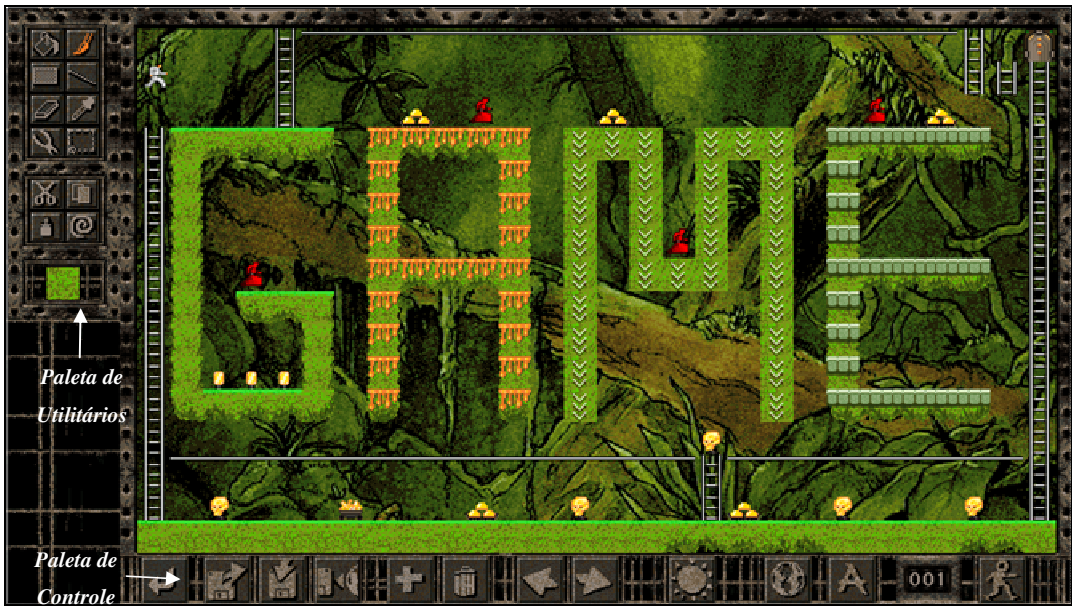


Figura 12 - Lode Runner Editor

O Editor de Níveis possui três paletas essenciais: Paleta de Componentes, Paleta de Utilitários e Paleta de Controle que serão mais bem detalhadas a seguir.

4.3.1 Paleta de Componentes

O editor apresenta uma paleta de componentes (Figura 13) com todos os ladrilhos possíveis de serem inseridos no jogo. Entre os ladrilhos possíveis de o usuário inserir no seu cenário estão os heróis, monges e portões que levam ao próximo estágio. Para salvar ou jogar o estágio criado o editor faz um pós-processamento que obriga ao usuário ter pelo menos um dos dois heróis inseridos no cenário. Contudo, este pós-processamento não se preocupa se o usuário inseriu um portão de saída, nem tão pouco se caso inserido, será possível alcançá-lo.

Figura 13 - Paleta de Componentes

4.3.2 Paleta de Utilitários

O editor possui também uma paleta (Figura 14) com algumas ferramentas especiais que podem ajudar o usuário a editar o seu cenário. Ela mostrou não ser essencial para a existência do editor, mas pode ser de grande valia para usuários mais avançados ou até mesmo ao *level designer* do projeto.

	1. ***Fill*** - Permite o preenchimento de uma área fechada com o ladrilho

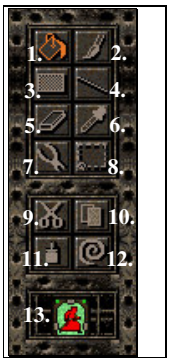


Figura 14 - Paleta de utilitários

4.3.3 Paleta de Controle

A paleta de controle (Figura 15) é de fundamental importância ao editor, pois é ela quem permite que o usuário carregue seus níveis para continuar editando, salve-os para poder jogá-los, navegue em seus níveis salvos no mesmo grupo de níveis e teste o cenário que está sendo construído.

Figura 15 - Paleta de Controle

1. ***Return to Main Menu*** - Retorna para a tela principal do jogo.
2. ***Load Level Group*** - Carrega um grupo de níveis (o jogo) salvo anteriormente.
3. ***Save Level Group*** - Salva o grupo de níveis corrente.
4. ***New Level Group*** - Inicia um novo grupo de níveis.
5. ***Add Level*** - Adiciona um nível ao grupo de níveis corrente.
6. ***Delete Level*** - Apaga o nível atual do grupo de níveis corrente.
7. ***Previous Level*** - Navega para o nível anterior.
8. ***Next Level*** - Navega para o próximo nível.
9. ***Darkness Model*** - Transforma o nível nos modos dia ou noite.
10. ***Background Select*** - Permite que o usuário selecione um *background* pré-definido para seu nível.

11. ***Enter Title*** - Permite ao usuário re-nomear o nível corrente.
12. ***Current Level Number*** - Mostra em que nível o usuário está editando.
13. ***Test Level*** - Executa o nível que estar sendo editado.

4.3.4 Observações Importantes

O editor do *Lode Runner* possui algumas peculiaridades que merecem ser expostas neste documento:

- **Troca automática do layout dos ladrilhos:** Como o editor permite a troca do *background* dos níveis que estão sendo gerados ele também faz a troca automática do *layout* de cada ladrilho para que os mesmo combinem com o *background* escolhido para o cenário. Essa troca de texturas (*skins*) pode ser facilmente observada nas figuras (Figura 16) e (Figura 17).

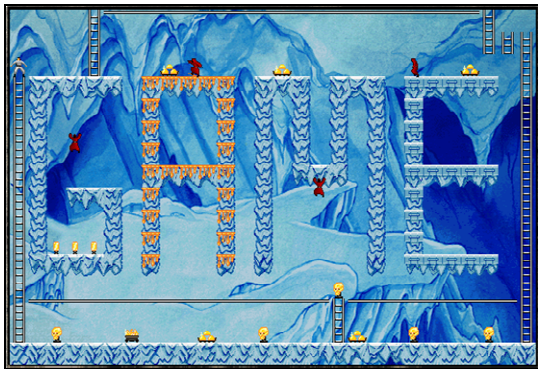


Figura 16 - Layout de uma Geleira

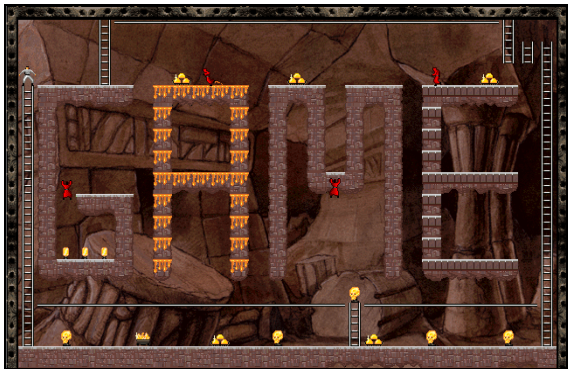
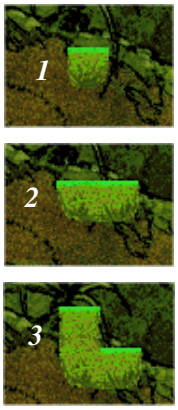


Figura 17 - Layout de cavernas rochosas

- **Adaptação automática das formas de cada ladrilho utilizada:** Para cada ladrilho inserido, o editor faz um pré-processamento da posição dos outros ladrilhos do mesmo tipo já inseridos e com isto, pode-se fazer pequenas alterações de ajustes na forma de cada ladrilho recém inserido ou em algum(s) inserido anteriormente para que o *design* do jogo fique mais agradável (Figura 18).

- **O mesmo tipo de ladrilho pode assumir três formas diferentes:**



1. Ladrilho em seu formato normal
2. Ladrilho com a ponta inferior esquerda arredondada devido ao seu ladrilho vizinho.
3. Ladrilho com a ponta inferior esquerda arredondada e sem um verde mais escuro na parte superior dele.

Figura 18 - Adaptação dos ladrilhos

- **O comportamento dos objetos do jogo não pode ser editado**: Os inimigos do herói no jogo, os monges, possuem um comportamento único que não pode ser editado pelo jogador.

Estas observações reforçam o fato que exite um forte acoplamento entre o jogo e o editor, algumas das funcionalidades implementadas pelo editor são específicas para o jogo em questão.

4.4 Grand Prix 3 (GP3)

Grand Prinx3[3] (Figura 19) consiste em um excelente simulador de fórmula 1, implementado pela empresa Hasbro Sports. Sua realidade física e condições aleatórias do tempo são tão boas que GP3 chega a ser considerado por alguns críticos como o melhor simulador de F1 do mundo [37]. A maior crítica que a Hasbro recebe pelos jogadores do *Grand Prix* é falta de uma atualização anual, assim como acontece com os jogos da EA Sports.

Figura 19 - Grand Prix 3

O jogo vem acompanhado de um editor de ótima qualidade, porém o mesmo não chega a ser um "editor do cenário" realmente, pois o mesmo não permite que alguns objetos essenciais ao jogo como, curvas, retas, elevações, zebras, brita, grama ou detalhes do cenário como túneis, lagos, edifícios, árvores, etc. possam ser editados.

Todavia, seu editor é bem poderoso ao permitir que vários atributos do jogo sejam alterados com facilidade. Alguns destes atributos podem modificar bastante o próprio comportamento do jogo.

O editor de GP3 possui 13 telas de atributos configuráveis. Sua enorme gama de funcionalidades pode atrapalhar usuários iniciantes, no entanto pode fazer usuários avançados se tornarem ainda mais atraídos pelo jogo.

A seguir, será apresentada uma breve explicação sobre algumas das 13 telas que o usuário pode usufruir para personalizar o GP3:

Editor de Times - Permite modificar o nome da equipe e do motor utilizado. Também permite alterar características dos carros e/ou pilotos. Para os carros de cada equipe o usuário pode mudar a força do motor em treino ou em corrida e a probabilidade de falha do motor. Para os pilotos podem ser alteradas sua performance e habilidade em treinos ou corridas (Figura 20).

Editor de Performance – Permite que a qualidade de guiar o carro de cada piloto seja editada e comparada com a dos outros pilotos. Também permite modificar a qualidade de performance do motor do carro e estabilidade do mesmo sejam alteradas (Figura 21).

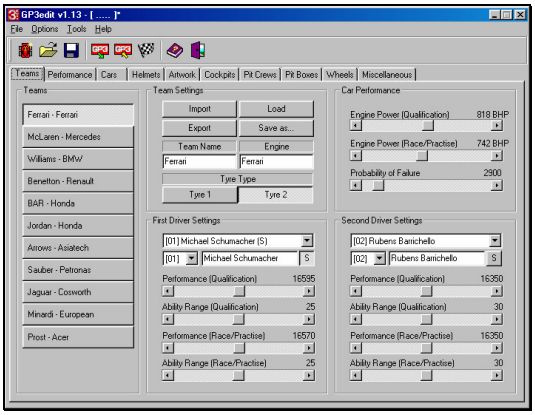


Figura 20 - Editor de Times

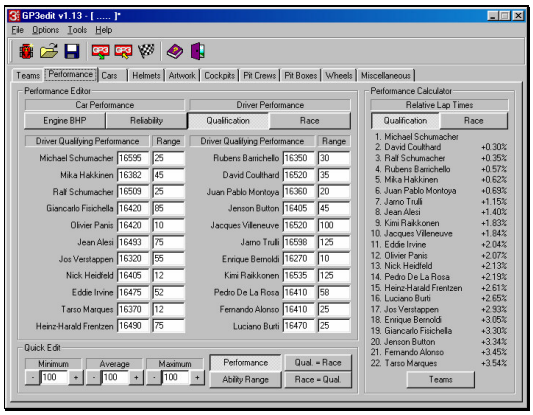


Figura 21 - Editor de Performance

Editor de Carros – Permite ao usuário modificar a aparência dos carros carregando um novo *Bitmap* (Figura 22).

Editor de Capacetes – Permite ao usuário modificar a aparência dos capacetes dos pilotos carregando um novo *Bitmap* (Figura 23).

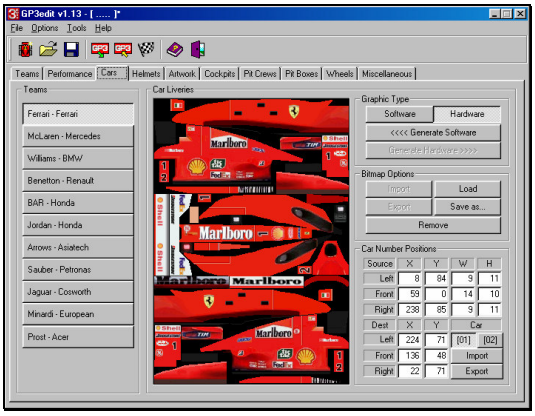


Figura 22 - Editor de Carros

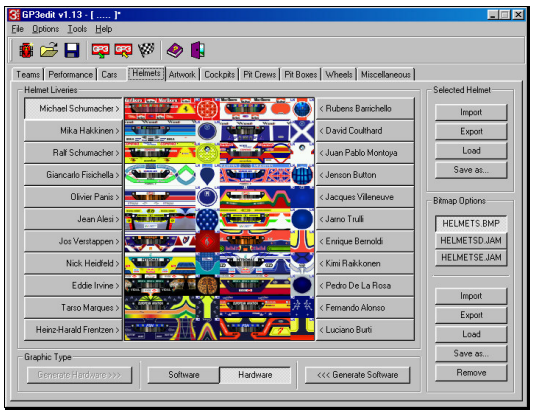


Figura 23 - Editor de Capacetes

Editor de arte – permite modificar algumas imagens que aparecem nos menus do jogo. Como fotos dos pilotos, logo marcas, miniaturas do carro, etc (Figura 24).

Editor de *Cockpits* – Permite ao usuário modifica a aparência dos *cockpits* de cada carro carregando um novo *Bitmap* (Figura 25).

Figura 24 – Editor de arte

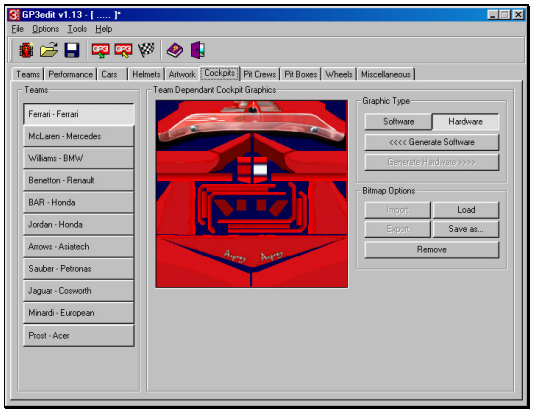


Figura 25 – Editor de Cockpits

4.5 Fifa 2002

Fifa 2002[6] é um excelente simulador de futebol da EA Sports com atualizações anuais e versões especiais em anos de copa do mundo. Lançado em 1994 Fifa já possui mais de 15 versões diferentes e é construído para várias plataformas, sendo que atualmente as que mais se destacam são o PC e o Play Station II, mas Fifa fez muito sucesso em vídeo games mais antigos como o Mega Drive da SEGA [31].

Fifa permite que seus jogadores joguem da forma que lhe é mais atraente, para isto ele dá a estes algumas possibilidades de posicionamento (4-3-3, 4-4-2, 3-5-2, 3-4-3, etc.), estilo de jogo (contra-ataque, posse de bola), postura ofensiva (variando em 5 possibilidades) e escalação do seu time. Essas opções forçam o usuário a possuir o mínimo de estratégia antes de colocar o seu time em campo.

Para dar mais realidade ao jogo os “jogadores” (personagens do jogo) possuem características físicas e atributos de habilidade baseados nos jogadores reais de clubes e seleções.

Como seria extremamente difícil manter uma base de times e jogadores tão grande como a do Fifa sempre atualizada, o jogo permite que seleções, times, copas, torneios e jogadores sejam facilmente criados ou editados.

A seguir haverá uma breve descrição do que o Fifa permite editar:

4.5.1 Copa / Torneio

Fifa permite ao usuário criar ou editar seus próprios campeonatos. Isto dá uma noção ao jogador, que ele pode criar as "regras" do jogo.
Para obedecer bem os critérios de campeonatos reais o usuário pode escolher:

- **Tipo de torneio:** Pontos corridos, Eliminatórias ou Misto.
- **Nome e data de início do campeonato:** O usuário pode dar nome ao seu campeonato e escolher o mês que ele se inicia.
- **Número de equipes do campeonato:** Quantos clubes ou seleções jogarão o campeonato, posteriormente o usuário selecionará os clubes ou seleções que desejar.
- **Número de partidas vs. cada time:** Quantas partidas os times devem jogar entre si na fase de pontos corridos.
- **Número de grupos:** Em quantos grupos os times devem se dividir.
- **Times que avançam para as eliminatórias:** Quantos times passam em cada grupo.
- **Número de partidas vs. cada time nas eliminatórias:** Quantas partidas os times devem jogar entre si nas fases eliminatórias.
- **Número de partidas vs. cada na final:** Quantas partidas serão disputadas na final.

4.5.2 Times

Times também podem ser criados ou editados (Figura 26). Nesta opção o usuário pode modificar o nome do time, o estilo e as cores dos uniformes.

Figura 26 - Editor de uniforme

4.5.3 Jogador

A opção do Fifa que realmente altera o comportamento do jogo é a edição de jogadores, pois além de atributos visuais como cabelo, olhos, rostos, cor da pele e outros (Figura 27 e Figura 28) cada jogador também possui atributos que fazem a diferença no seu rendimento em campo (Figura 29).

A alteração das propriedades dos jogadores faz uma grande diferença na partida, pois o algoritmo do jogo modela bem o comportamento de cada jogador baseado em seus atributos. Um time com bons jogadores, dificilmente poderá ser derrotado por um time de jogadores bem inferiores.

Figura 27 - Editor de aparência

Figura 28 - Editor de informações

O Usuário pode alterar alguns atributos de comportamento:

- Preparo Físico
- Velocidade
- Chute
- Passe
- Força
- Habilidade
- Posicionamento
- Agressividade

Figura 29 - Editor de atributos

4.5.4 EA Graphics Editor

Para uma maior liberdade de edição da parte gráfica, a EA Sports disponibiliza na *Web* um outro editor chamado de *EA Graphics Editor* (Figura 30). Com ele os usuários dos jogos da EA Sports (Fifa, NBA, NHL e outros) podem editar em mais detalhes a parte visual do jogo. Como exemplo, pode-se editar o uniforme completo de um time, com o escudo e marcas de patrocinadores.

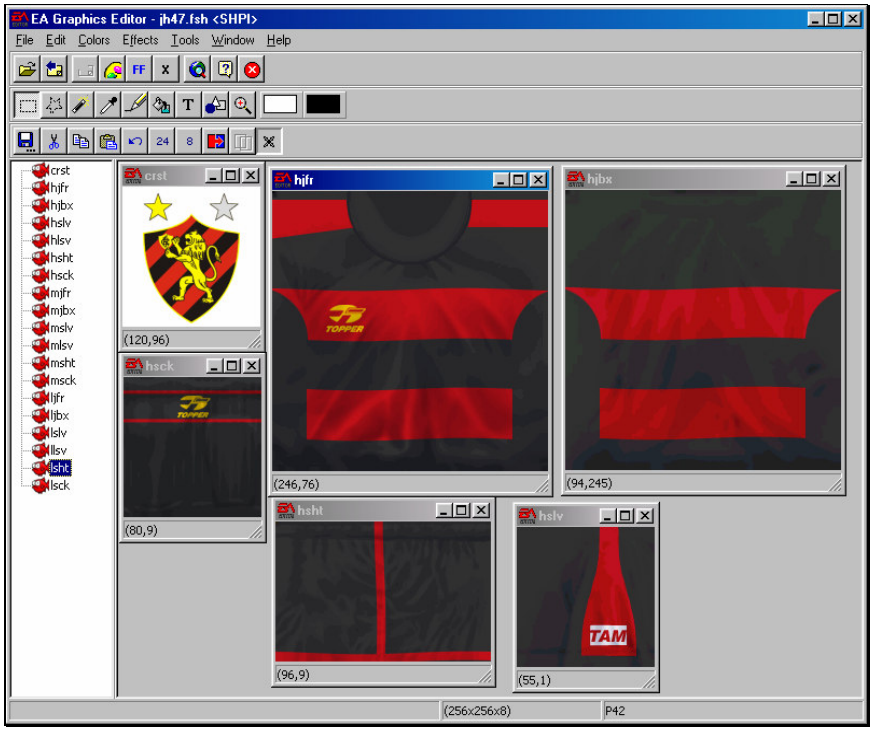


Figura 30 - EA Graphics Editor

O *Graphics Editor* também permite que telas de menus ou fotos de jogadores, no caso da NBA, sejam alteradas ou editadas, afinal, este editor foi feito para trabalhar com todos os formatos gráficos utilizados em jogos da EA Sports.

4.6 Age Of Empires

Age of Empires[5] é um jogo de estratégia em tempo real, onde os jogadores são responsáveis pela evolução de pequenas tribos para grandes civilizações da história mundial. O jogador pode participar da construção de impérios como o do Egito, Babilônia, Grécia, etc.

Além de ajudar na evolução de uma civilização, *Age of Empires* permite que o usuário jogue estágios apenas para competir com seu(s) oponente(s), seja ele o computador ou outro(s) jogadore(s). Para obter ainda mais sucesso, o jogo da Microsoft® traz um excelente Editor de Níveis (Figura 31), que permite um alto grau de personalização do jogo.

Figura 31 - Editor de Age of Empires

O editor de *Age of Empires* traz várias funcionalidades divididas em menus distintos.

A seguir temos uma breve descrição dos principais menus e suas funcionalidades.

4.6.1 O Mapa

O usuário pode partir de um mapa em branco ou pedir ao editor que crie um mapa aleatório para começar a criar o seu nível. Algumas propriedades básicas do mapa também podem ser editadas:

- **Tamanho:** Minúsculo, Pequeno, Médio, Grande ou Enorme.
- **Terreno padrão:** Ilhas Pequenas, Ilhas Grandes, Litoral, Interior ou Planalto para mapas aleatórios e Grama, Deserto, Floresta, Água, Palmeiral, Selva, Banco de areia, Floresta de Pinheiros ou Águas Profundas para mapas começados do nada.

4.6.2 Terreno

No menu "Terreno", pode-se inserir ladrilhos ligados ao terreno (grama, deserto, floresta, água, palmeiral, selva, banco de areia, floresta de pinheiros ou águas profundas) ou inserir elevações pré-definidas. Pode-se também, escolher o tamanho da área que o novo ladrilho inserido vai ocupar (Minúsculo, Pequeno, Médio, Grande ou Enorme).

4.6.3 Jogadores

Este é um dos menus mais interessantes do editor. Ele permite ao usuário informar quantos jogadores poderão jogar aquele cenário, variando sempre entre 1 e 8 jogadores que podem ser humanos ou não. Depois de escolhido o número de jogadores pode-se configurar os atributos inicias de cada jogador, e dentre eles pode-se citar:

- **Período em que o jogador começa**: Idade da Pedra, Idade da Ferramenta, Idade do Bronze, Idade do Ferro ou Pós-Idade do Ferro.
- **Riquezas:** Quantidade inicial de Alimento, Pedra, Ouro e Madeira.
- **Tipo de Jogador:** Humano ou Máquina
- **Civilização:** Egípcia, Grega, Babilônica, Assíria, Minoana, Persa, Suméria, etc.
- **Personalidade:** Agressiva, Padrão, Defensiva, Passiva, etc.
- **Nome da tribo**

4.6.4 Unidades

Este menu é responsável pela inserção dos objetos do jogo. Nestes objetos estão contidos tantos os personagens do jogo (aldeões, guerreiros, heróis, sacerdotes) como também artefatos, construções e monumentos. Vale ressaltar que o comportamento destes personagens é pré-definido e não pode ser editado pelo usuário.

4.6.5 Diplomacia

É outro menu interessante, a partir dele, o usuário pode definir através de uma matriz qual jogador é inimigo ou aliado de quem, estabelecendo assim a relação existente entre os jogadores, e se a vitória deve ser individual ou em conjunto.

4.6.6 Vitória Individual

Nele o usuário define as condições de vitória e regras para ganhar o jogo, para cada jogador. Cada jogador pode ter 12 condições de vitória. Existem 18 possibilidades de condições de vitória e algumas delas são: trazer objeto até uma área específica, criar um certo número de objetos, destruir objetos específicos, destruir jogador, estoque de ouro, evoluir até uma certa idade, etc, sendo cada condição de vitória desta editada individualmente (Figura 32).

Figura 32 - Condições de vitória

4.6.7 Mensagens

Permite ao usuário registrar mensagens que servirão como instruções para o cenário, como dicas, texto de vitória ou derrota e fatos históricos do seu cenário. Este menu permite passar de forma escrita a idéia do usuário ao criar um determinado nível.

4.6.8 Observações

O ponto forte de *Age of Empires* não está no seu módulo gráfico ou na variação do *gameplay,* como nos jogos de ação, está sim na grande possibilidade de decisões que o jogador pode tomar, no enorme número de objetivos diferentes que devem ser alcançados nos níveis e na diversidade de estágios bem planejados e com boas histórias, que fazem com que os jogadores queiram conhecer cada nível e vencê-los com entusiasmo.

O Editor de Níveis merece um destaque especial no jogo. Sua enorme gama de funcionalidades e sua flexibilidade de criar cenários completamente diferentes fazem deste editor uma das referências mais ricas deste documento.

Alguns dos requisitos do editor que merecem destaque são:

- **Tamanho do mapa variável:** Isto permite que usuários mais avançados possam criar cenários maiores e mais ricos, para aumentar o grau de dificuldade dos jogos e seus desafios, e que usuários iniciais criem cenários menores e mais simples apenas pela diversão de criar e jogar o estágio criado por ele mesmo.

- **Tamanho de ladrilhos variável:** O fato de o editor permitir que o jogador insira ladrilhos de tamanhos diferentes, faz com que a edição do cenário não necessite de muito tempo para ser consolidada, pois ele pode ficar sempre inserindo ladrilhos grandes e ficar arrastando o mouse sobre o mapa para que o ladrilho seja repetido. Ao mesmo tempo é possível melhorar os detalhes de um cenário ao se utilizar ladrilhos de tamanho pequeno.
- **Ladrilhos com animação:** Além de fazer a adaptação dos ladrilhos conforme os que o cercam, para que o cenário fique com uma riqueza maior de detalhes. O ladrilho da "água", em *Age of Empires,* fica mudando de imagem dando a impressão de movimento da água.
- **Definir o número de jogadores:** O editor é capaz de criar uma comunidade de jogadores, pois os usuários podem desafiar outros jogadores a vencê-los no seu estágio.
- **Definir diplomacia entre os jogadores:** Permitir ao criador do cenário definir o relacionamento entre os jogadores e aumentar a possibilidade da criação de uma comunidade de jogadores. Com isso, os usuários podem se dividir em grupos e jogar contra outros.
- **Definir objetivo de cada jogador:** Embora seja uma particularidade do jogo a existência de mais de uma condição de vitória, o editor mostra boa maturidade ao permitir que cada jogador do cenário possa ter um objetivo completamente diferente dos outros jogadores.

4.7 Requisitos e Características Gerais

Após os estudos sobre editores apresentados nas seções anteriores, foi concluído que um editor pode ter três funcionalidades distintas, mas nada impede que elas coexistam em um único editor: Gráfica, Comportamento e Cenário.

Os editores podem ser classificados de acordo com suas funcionalidades em:

- Editor Gráfico - Permite alterações no visual do jogo e/ou dos personagens deste, tais como alterações de *background* ou de *layout* dos objetos do jogo. Bons exemplos deste tipo de editor são os do Fifa 2002 e GP3. O editor de Fifa permite modificação do rosto, cor de pele e cabelo dos jogadores e a criação de novos times

com novos uniformes e escudos como já citado anteriormente. Já o Editor de GP3 faz algo semelhante com suas escuderias, carros, mecânicos, etc.

- Editor de Comportamento - Atributos que refletem no comportamento do jogo em si podem ser modificados. Assim usuários podem alterar diretamente a dificuldade do jogo. Os editores de Fifa e GP3 também implementam isto muito bem. Fifa permite alterar atributos dos jogadores como precisão de chute, qualidade no passe, habilidade, força, dividida, velocidade, etc. Já o GP3 possibilita alterações tanto nos atributos dos pilotos como performance e habilidade em corrida e em treino. Permite também modificações nos atributos dos carros como potência do motor, probabilidade de falhas, etc, como citado anteriormente. Infelizmente nenhum dos editores estudados mostrou ser um profundo modelador de comportamento para os NPCs, em geral existem comportamentos pre-modelados e é dada a opção ao usuário de escolher qual comportamento ele deseja.
- Editor de Níveis – São os editores que realmente possibilitam ao usuário criar seus novos cenários ou níveis. Alguns exemplos são: *Laser Tank*, *Lode Runner*, *Age of Empires*, *Star Hero*, *Starcraft*.

Apesar de nenhum dos editores estudados ter sido feito para um jogo de dispositivo móvel, o estudo dos editores focados na criação de cenários foi essencial para se saber quais os requisitos básicos de que um Editor de Níveis necessita.

Os requisitos encontrados em editores com objetivo gráfico ou de comportamento são muito dependentes do jogo. Como o objetivo deste estudo é focar em requisitos que possam ser utilizados em editores de jogos diversos, os aqui mencionados, serão em sua maioria, para editores que tenham como objetivo construir novos níveis ou cenários.

[RF001] O sistema deve prover uma paleta de componentes: A paleta de componentes deve possuir todos os objetos e/ou ladrilhos possíveis de serem inseridos no cenário ou nível. O usuário deve ser capaz de selecionar estes objetos ou ladrilhos e inseri-los no cenário que está criando.

[RF002] O sistema deve criar campanhas: O Editor de Níveis deve permitir que campanhas (conjuntos de níveis) sejam criadas.

[RF003] **O sistema deve salvar campanhas:** O Editor deve permitir que as campanhas criadas possam ser salvas em disco.

[RF004] **O sistema deve editar campanhas:** O Editor deve permitir que campanhas salvas possam ser editadas.

[RF005] **O sistema deve criar níveis:** O Editor deve permitir que níveis sejam criados dentro de uma campanha.

[RF006] **O sistema deve editar níveis:** O Editor deve permitir que níveis criados possam ser modificados.

[RF007] **O sistema deve remover níveis:** O Editor deve permitir que níveis sejam removidos de uma campanha.

[RF008] **O sistema deve criar um cenário inicial:** O Editor deve permitir que o cenário inicial de um nível seja criado pelo usuário.

[RF009] **O sistema deve prover um modo de testes:** Os jogos com editores devem permitir ao usuário jogar os níveis criados por ele. Quanto mais fácil for para o usuário jogar no cenário que ele criou, mais fácil será fazer testes e balancear os níveis criados.

[RF010] **O sistema deve armazenar mensagens:** Dependendo do jogo, pode ser interessante ao usuário armazenar textos, sons ou vídeos com a história do nível, dicas para o jogador vencer aquele nível, ou até mesmo textos que sirvam de provocação aos outros jogadores.

[RF011] **O sistema deve prover troca Automática do *layout* dos ladrilhos:** Caso o editor permita a troca do *background* de cada nível, ele pode fazer a troca automática do *layout* ("*skin*") de cada ladrilho para que os mesmos combinem com o *background* escolhido para o cenário.

[RF012] O sistema deve gerar automaticamente mapas aleatórios: Para que o usuário não precise criar um cenário totalmente do início. O editor pode gerar automaticamente mapas aleatórios para o usuário e o mesmo apenas alterar o que desejar. Contudo, deve-se ter em mente que o mapa criado precisa obedecer às regras lógicas do jogo, ou seja, deve haver uma forma possível do nível ser vencido.

[RF013] O sistema deve prover uma paleta de utilitários: O editor pode fornecer uma paleta com alguns utilitários para o usuário. Esta paleta pode ter funcionalidades como: copiar/recortar e colar uma área específica do mapa, pode permitir que um mapa seja preenchido com um ladrilho base ou permitir que o usuário modifique o tamanho do ladrilho para que a edição do cenário não seja tão cansativa, etc.

[RF014] O sistema deve permitir tamanho variável do terreno: Mesmo para jogos menos rebuscados o tamanho do cenário pode influenciar bastante no *gameplay*. Sendo assim, o editor poderia permitir aos usuários configurar o tamanho do cenário que queiram criar.

[RF015] O sistema deve prover ladrilhos com animação: Para dar um efeito de movimentação ao cenário, alguns ladrilhos podem ficar mudando de imagem, para dar a impressão que o cenário se movimenta. Dependendo da animação que se queira passar, faz-se necessária uma sincronização das trocas de imagens.

[RF016] O sistema deve permitir modificações nas regras do jogo: Algumas regras do jogo podem ser editadas pelo usuário, tais como: regras do campeonato, modificação do objetivo que deve ser alcançado para vencer o estágio, número de jogadores que podem jogar no cenário criado, etc.

[RF017] O sistema deve prover Pré e Pós-tratamento de ladrilhos: Objetivando uma maior riqueza na parte gráfica, os ladrilhos podem ter suas extremidades um pouco alteradas. Para cada ladrilho inserido, o editor pode fazer um processamento da posição dos outros ladrilhos do mesmo tipo inseridos anteriormente e com isto, pode-se fazer pequenas alterações de ajustes na forma de cada ladrilho, no intuito de deixar que o *design* do jogo fique mais contínuo.

[RF018] O sistema deve realizar checagem de validação do estágio: Quando o usuário for salvar o nível que ele criou, o editor deve verificar se os componentes mínimos necessários ao *gameplay* do estágio foram adicionados. O editor também pode fazer uma verificação da existência de alguma solução para que exista possibilidade de vitória no estágio.

[RF019] O sistema deve criar um grafo de níveis: Dependendo do jogo, um estágio pode levar a mais de um estágio diferente, dependendo de que ações o usuário execute enquanto está jogando, funcionando assim, como um grafo de níveis. Caso o jogo possua este requisito, o editor deve de alguma forma permitir que o usuário controle que ação levará a que nível.

[RF020] O sistema deve denominar jogos e níveis: O editor pode permitir aos usuários dar nomes tantos aos jogos (conjunto de níveis) que ele cria, quanto para os níveis de cada jogo.

[RF021] O sistema deve permitir modificação do comportamento dos objetos: Caso os objetos do jogo possuam atributos como velocidade, força, resistência, tamanho, etc, pode ser interessantes ao editor possibilitar a alteração dos valores destes atributos, modificando assim o comportamento do jogo.

[RF022] O sistema deve permitir modificação da aparência dos objetos: O editor pode permitir que a aparência de alguns ou todos os objetos do jogo sejam modificados pelo usuário.

A Tabela 2 mostra um mapeamento entre os principais requisitos levantados e em quais editores estudados eles foram implementados. Isto permite uma melhor visualização dos requisitos principais de um editor.

	Laser Tank	Lode Runner	GP3	Fifa 2002	Age of Empires
[RF001]	X	X			X
[RF002]	X	X		X	X
[RF003]	X	X			X
[RF004]	X	X			X
[RF005]	X	X			X
[RF006]	X	X			X
[RF007]	X	X			X
[RF008]	X	X			X
[RF009]	X	X			
[RF010]	X				X
[RF011]		X			
[RF012]					X
[RF013]		X			
[RF014]					X
[RF015]					X
[RF016]				X	X
[RF017]		X			X
[RF018]		X			X
[RF019]					
[RF020]	X			X	X
[RF021]			X	X	
[RF022]			X	X	

Tabela 2 - Editores x Requisitos

5 Estudo dos Casos Implementados

Após todos os estudos feitos, concluiu-se que era preciso conhecer melhor o procedimento de construção tanto de jogos para J2ME, como também, de editores para estes jogos. Com isso, foi decidido que seria desenvolvido um jogo de teste, um editor para este jogo e mais dois editores para dois jogos desenvolvidos pela equipe do CESAR .

Durante o desenvolvimento do jogo em J2ME, teve-se a idéia de construir mais um editor que seria acoplado ao jogo em desenvolvimento. Isto seria útil para saber como seria desenvolver um editor que para um dispositivo móvel com limitações de espaço, memória e processamento, etc.

Depois que os três editores para os três diferentes jogos foram desenvolvidos, foi feito um estudo para se obter partes do código que eram comuns aos editores de jogos, abstraindo-se assim, o *framework* desejado.

A primeira seção deste capítulo fala sobre o jogo Space Runner e algumas peculiaridades existentes em aplicações Java para dispositivos móveis, a seção seguinte apresenta o editor acoplado ao jogo Space Runner. A seção 5.3 descreve o primeiro editor para computador pessoal utilizado como caso de estudo, a seção seguinte descreve o jogo Pacman e seu editor, por fim a seção 5.5 descreve o último caso de estudo com o jogo Pod Race e seu editor.

5.1 O Jogo Space Runner

O jogo Space Runner, desenvolvido durante este trabalho, faz parte do primeiro caso de estudos aqui apresentado.

5.1.1 O jogo

Para observar, na prática, como seria a construção de um jogo com o Editor de Níveis na plataforma J2ME, foi implementado um jogo baseado em clássico do vídeo game chamado *Lode Runner* [4]. Este jogo foi denominado, a princípio, de *Gold Hunter* (Figura 33). Além de ser implementado com o objetivo de ajudar na elaboração deste trabalho, o *Gold Hunter* foi submetido ao concurso mundial, *Asia Java Mobile Challenge* [11], ocorrido em 2002-2003, voltado a aplicações para celular desenvolvidas em J2ME. O jogo foi classificado entre as 20 melhores aplicações submetidas.

Em março de 2003 o jogo recebeu uma nova versão para concorrer no concurso Mobilidade, promovido pela Nokia [21] e o Sou Java [22] e teve por conseqüência, seu nome modificado para *Space Runner* (Figura 34). Desta vez, o jogo conquistou a primeira colocação, na categoria de jogos para celulares da série 60 da Nokia. O jogo *Gold Hunter* sofreu novas melhorias por parte da equipe do CESAR, e hoje é comercializado por algumas operadoras de telefonia. Neste documento, a referência ao jogo será feita pelo seu nome mais atual: *Space Runner*.

[W1] Comentário: EXPLICAR

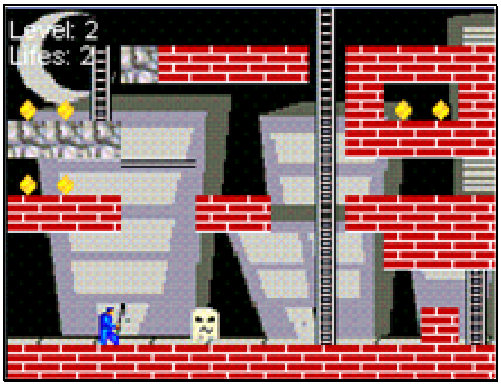


Figura 33 - Gold Hunter

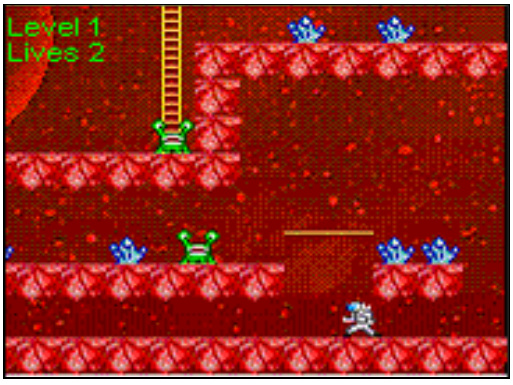


Figura 34 - Space Runner

No *Space Runner,* a missão do jogador é viajar através do espaço coletando uma fortuna em diamantes. Contudo, estes diamantes são guardados por alienígenas perigosos, e para conseguir completar a sua missão, o jogador precisará ser ágil para poder fugir do perigo, resolver pequenos quebra-cabeças e não ser pego pelos extraterrestres. A única arma que o jogador possui, é o poder de destruir um tipo específico de piso, com sua

pistola a laser. Depois de coletar todas as pedras de diamante de um estágio, um portal será exibido em algum lugar do cenário e o herói precisará então, atravessar o portal para ser conduzido ao próximo estágio.

Este jogo possui um Editor de Níveis acoplado (Figura 35), permitindo ao jogador criar e jogar seus próprios níveis, diretamente no dispositivo móvel onde o jogo estiver instalado. Este fato faz com que o jogo seja constituído de uma campanha padrão e de campanhas criadas pelo usuário final.

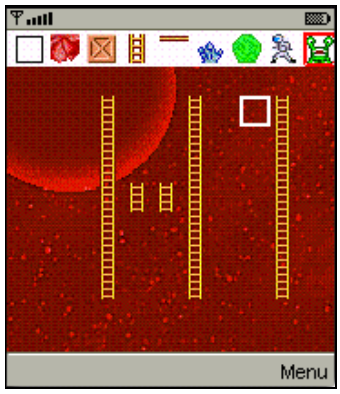


Figura 35 - Editor acoplado ao jogo

5.1.2 Arquivos que Compõem o Space Runner

Apesar do foco deste trabalho não ser o jogo em si, faz-se necessário entender os arquivos que compõem uma aplicação Java para dispositivos móveis, como também saber o nome do arquivo onde os níveis são armazenados dentro do jogo Space Runner. Alguns requisitos dos editores desenvolvidos no caso de estudo, são baseados nestes conceitos.

Toda aplicação J2ME é composta por dois arquivos, um descritor de aplicação e um arquivo compactado. O descritor de aplicação, conhecido como “jad”, contém algumas propriedades da aplicação, como por exemplo: nome da aplicação, nome do vendedor, tamanho do arquivo compactado, etc. O arquivo compactado, conhecido como “jar”, contém todos os arquivos que realmente compõe a aplicação, ou seja, as classes, imagens e outros arquivos de recursos necessários para a execução da mesma.

No caso do jogo Space Runner o arquivo que contém os níveis originais do jogo, é o arquivo “map.bin”, encontrado dentro do arquivo “jar”. O mesmo é armazenado de forma binária para que seu tamanho total seja reduzido e o espaço de memória no dispositivo móvel economizado.

5.2 Editor Móvel do Space Runner

Para construir o editor no celular, juntamente com o jogo, foi necessário elaborar um modelo navegacional intuitivo, que dispensasse a necessidade de um manual, embora um manual em inglês do jogo [19] tenha sido elaborado. Infelizmente, a navegação do editor pode ser considerada grande (Figura 36), devido ao mecanismo de persistência provido pelo mesmo.

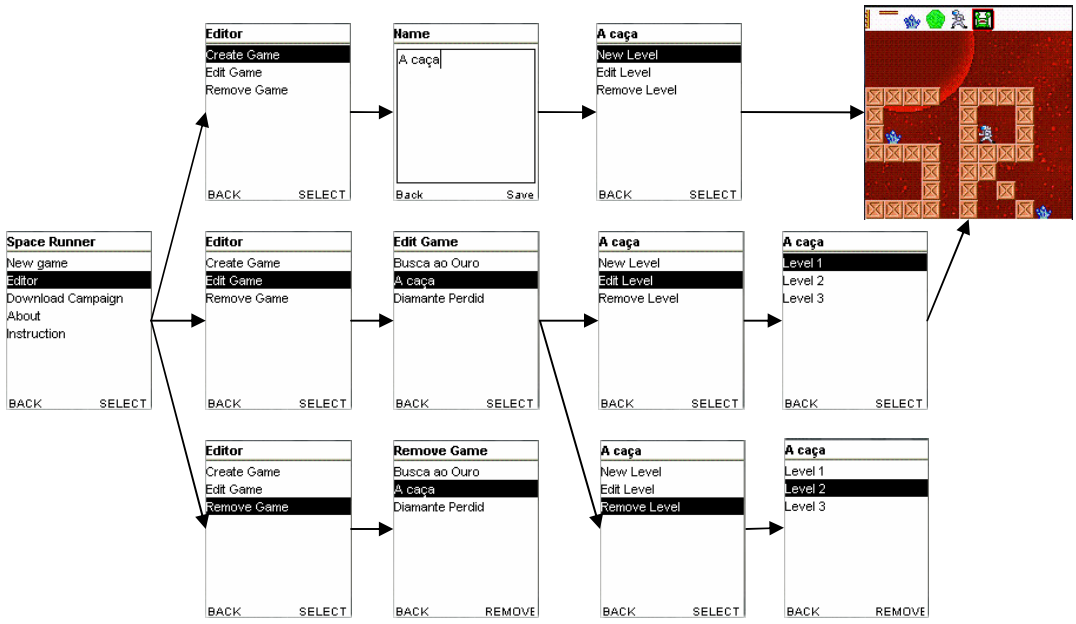


Figura 36 - Navegação do editor

5.2.1 Requisitos Funcionais

Este editor é constituído por uma paleta de componentes e por uma área onde o cenário, imagem estática de um nível, é desenhado (Figura 37). Devido ao fato da tela do celular poder ser menor que o tamanho necessário para ambos os componentes, os mesmos possuem a capacidade de movimentação na tela de acordo com a posição de seu cursor, isto é conhecido como "*scroll*".

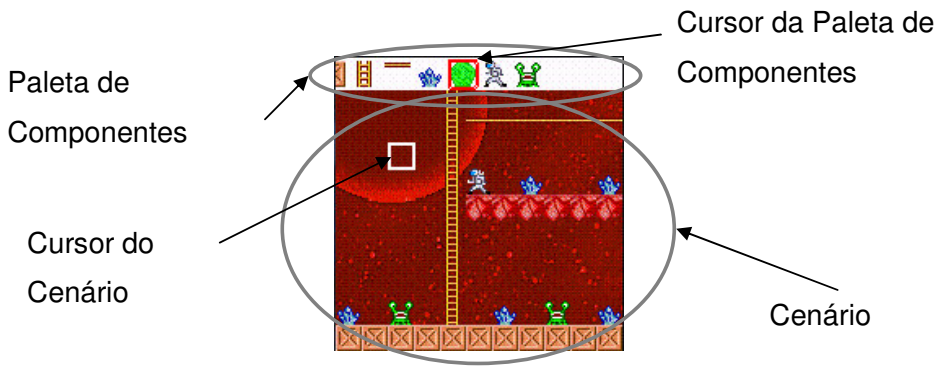


Figura 37 - Editor do Space Runner no celular

Para construir de fato o nível, é necessário desenhar o cenário do mesmo. Neste editor, isto pode ser feito primeiramente selecionando um item da paleta de componentes. Isto é feito utilizando teclas navegacionais do dispositivo móvel e o botão *fire*[5]. Em seguida é escolhido o local que este item deve ser inserido no cenário. Caso deseje apagar um objeto inserido, o usuário deverá realizar o mesmo procedimento de inserção de um item, com a diferença que o objeto a ser selecionado é representado por um quadrado em branco.

Em síntese, cabe ao usuário escolher em que posições itens do jogo como: escada, corda, piso, diamantes, inimigos, herói e portal deverão estar posicionados no começo do nível. Este posicionamento inicial pode ser de fundamental importância para a dificuldade e jogabilidade do nível. Este editor implementa alguns dos requisitos, comentados na seção 4.7. São eles:

- **[RF001] - O sistema deve prover um paleta de componentes ;**
- **[RF002] - O sistema deve criar campanhas;**
- **[RF003] - O sistema deve salvar campanhas;**

[5] Botão de fire é o botão do celular utilizado para realizar chamadas.

- **[RF004] - O sistema deve editar campanhas;**
- **[RF005] - O sistema deve criar níveis;**
- **[RF006] - O sistema deve editar níveis;**
- **[RF007] - O sistema deve remover níveis;**
- **[RF008] - O sistema deve criar um cenário inicial;**
- **[RF009] - O sistema deve prover um modo de testes;**
- **[RF018] - O sistema deve realizar checagem de validação do estágio.**

Além dos requisitos listados acima foi necessário definir outros requisitos específicos para este editor. São eles:

[EM_RF001] – O sistema deve nomear jogos e níveis: O editor permite ao usuário nomear os seus jogos, contudo, o nome dos níveis é gerado pelo editor, para economizar espaço de armazenamento.

[EM_RF002] – O sistema deve verificar se o herói foi inserido: Para suportar o requisito [RF018] uma das responsabilidades do editor é verificar se o herói foi inserido em cada nível.

[EM_RF003] – O sistema deve verificar se o portal foi inserido: Também com o intuito de suportar o requisito [RF018], outra responsabilidade do editor é verificar se o portal foi inserido em cada nível.

[EM_RF004] – O sistema deve verificar se existe ao menos um diamante: O último requisito para suportar o requisito [RF018] é verificar se ao menos um diamante foi inserido em cada nível.

5.2.2 Arquitetura

Devido ao fato do editor encontrar-se em um ambiente com capacidade de processamento e memória extremamente limitados, sua arquitetura foi projetada para ser o mais simples e enxuta possível. Ela é constituída por duas classes: `EditorCanvas` (Figura 38) e `Persitence` (Figura 39). Além destas classes, outras classes pertencentes ao jogo sofreram modificações principalmente em função do modelo navegacional, já ilustrado anteriormente.

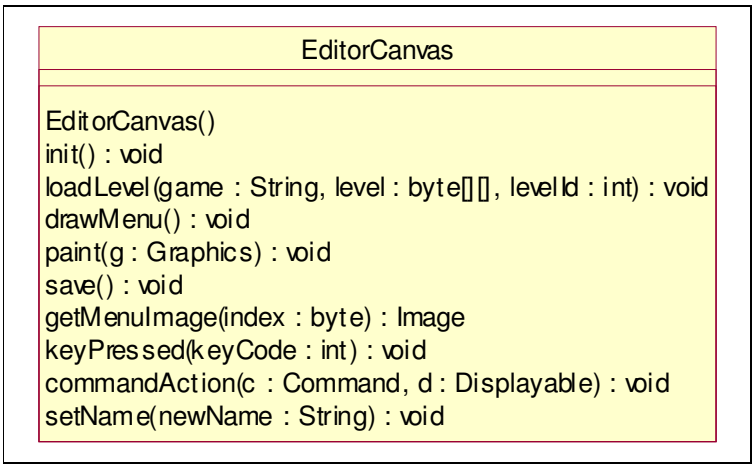


Figura 38 - Classe EditorCanvas

A classe `EditorCanvas` é responsável por implementar a parte lógica e gráfica do Editor de Níveis. Na parte gráfica, ela é quem desenha tudo que o usuário visualiza na tela, possibilita a navegação do cursor, faz *scroll* quando necessário e mantém a visualização do cenário consistente com o mapa de ladrilhos. É de sua atribuição também, fazer alguns controles como: a verificação da validade de cada nível e a chamada à classe `Persistence`, para que a mesma garanta a integridade dos níveis criados.

A classe `Persistence` é quem controla todo o acesso aos registros físicos do dispositivo móvel, os chamados RMS (*Record Memory Store*). Como o usuário pode salvar alguns grupos de níveis (campanhas), a persistência é feita com a existência de um RMS para cada campanha criada e um *RMS Master* que contém o nome dos RMSs utilizados. Isso se faz necessário, pois um RMS armazena apenas uma seqüência de bytes e não, estruturas bem definida.

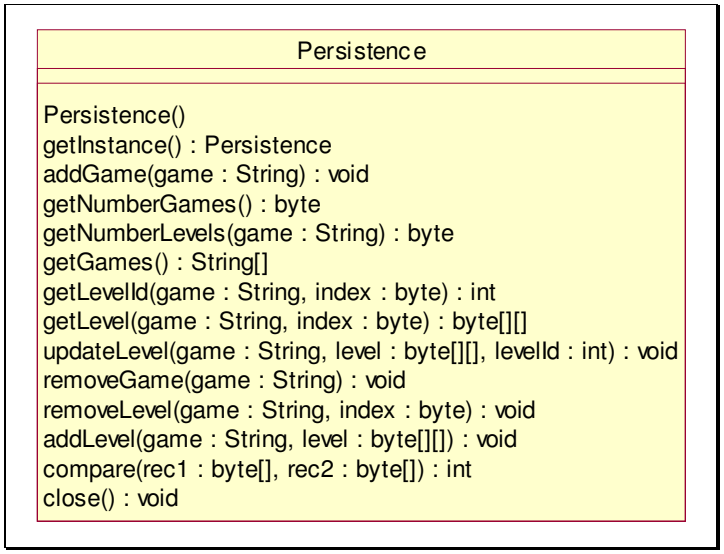


Figura 39 – Classe Persistence

5.2.3 Avaliação Crítica

Infelizmente, utilizar este editor não é uma tarefa agradável, pois o mesmo é executado em um dispositivo com poucos recursos. Em um celular, por exemplo, não é possível visualizar todo o cenário de uma única vez e não se pode lançar mão do uso de um dispositivo como o *mouse*. Embora, o editor tente superar isto com a rolagem da tela e uso das teclas direcionais, construir um bom nível diretamente no celular é uma tarefa desafiadora.

Partiu-se então para o desenvolvimento de um Editor de Níveis para o jogo *Space Runner,* que pudesse ser executado em computador pessoal, pois, este seria sem dúvida, um ambiente de produção contendo uma quantidade maior de recursos e um ambiente mais agradável de se trabalhar.

5.3 Space Runner Editor para Computador Pessoal

O editor do *Space Runner* para computador pessoal (Figura 40) foi o primeiro editor construído cujos seus requisitos contribuíram para a elaboração do *framework* desejado. Antes da construção deste editor dois problemas fundamentais foram analisados: O

primeiro seria sobre qual formato semântico e de dados, as campanhas seriam salvas e o segundo, foi como permitir que os diferentes tipos de usuários, desenvolvedor e jogador, pudessem testar cada nível ou campanha construída.

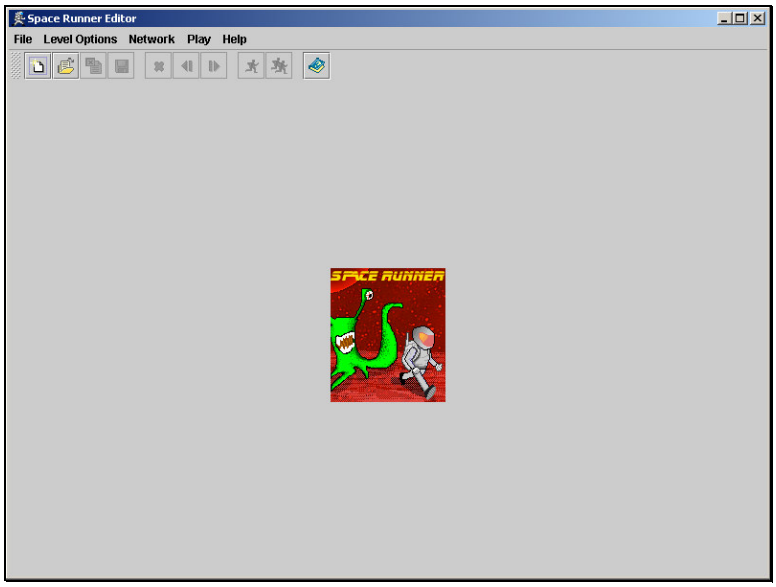


Figura 40 - Space Runner Editor

5.3.1 Utilizando XML para Salvar Campanhas

Para resolver o problema de formato de dados, em que uma campanha deve ser salva, optou-se pelo formato XML [44]. XML pode ser visto como um conjunto de regras, convenções ou diretrizes para projetar formatos de texto que permitam estruturar dados [45].

Dentre os motivos para a escolha do formato XML, pode-se citar a facilidade de ler e interpretar este tipo de arquivo e a facilidade na coleta de informações, caso seja necessário. Além disto, XML é extensível, independente de plataforma e de fácil manutenção.

No caso do editor para o jogo Space Runner, foi decidido que seriam armazenados, no arquivo XML, informações sobre a campanha. Como, por exemplo, nome da mesma, nome de quem a criou, altura e largura dos cenários e o número de níveis que a mesma

continha; e informações sobre cada nível, como por exemplo: nome, dica para vencer o nível e o mapa inicial do cenário. Com isto, chegou-se então, ao formato de arquivo XML exibido na tabela abaixo (Tabela 3):

```
<?xml version="1.0" encoding="UTF-8" ?>
<Campaign>

  <Name>Nome da Campanha</Name>
  <Owner>Nome do usuário que criou a campanha</Owner>
  <Width>Altura do cenário</Width>
  <Height>Largura do cenário</Height>
  <NumberOfLevels>Número de níveis dentro da campanha</NumberOfLevels>

  <AllLevels>
    <Level>
      <Name>Nome do nível</Name>
      <Tip>Dica para o nível</Tip>
      <Map>Mapa incial do nível</Map>
    </Level>

     ...

    <Level>
      <Name></Name>
      <Tip> </Tip>
      <Map> </Map>
    </Level>

  </AllLevels>
</Campaign>
```

Tabela 3 -Exemplo de arquivo XML

5.3.2 Possibilitando os Testes

O problema mais difícil, quando se pensou em construir um Editor de Níveis executado em computador pessoal para jogos móveis foi conceber como os níveis criados seriam testados por desenvolvedores e usuários. Algumas soluções, como a utilização de

infravermelho ou *Bluetooth*, formas de comunicação entre dispositivos, foram descartadas logo de início, devido ao fato dessas tecnologias não serem obrigatórias na especificação de MIDP 2.0 [46], camada da linguagem J2ME, da Sun Microsystem.

Para resolver o problema de como testar os níveis criados, três soluções foram estudadas e postas em prática. A solução mais simples, é completamente voltada para os desenvolvedores, a mesma consiste em substituir o arquivo que armazena os níveis, "map.bin", ainda não compactado e inserido no arquivo "jar". O desenvolvedor pode então, empacotar o jogo novamente e testar o nível, ou níveis criados.

A segunda solução é um pouco mais rebuscada. Esta assume que o jogo encontra-se terminado e o desenvolvedor ou usuário possui apenas os arquivos "jad" e "jar" que compõem uma aplicação Java para dispositivo móvel. A solução consiste em o editor abrir o arquivo compactado, "jar", substituir o arquivo de níveis, "map.bin", compactar novamente o arquivo, alterar a propriedade que indica o tamanho do arquivo "jar" no descritor de aplicação, "jad", e finalmente chamar o simulador de aplicações J2ME provido pela Sun Microsystem.

A última solução foi pensada para o jogador, que possui o jogo apenas no seu dispositivo móvel. A mesma depende de um servidor WEB, utilizado para armazenar níveis, que seja capaz de se comunicar com o computador pessoal do usuário e com o aparelho móvel do mesmo, utilizando o protocolo HTTP [47].

A solução consiste em, após o usuário criar a campanha desejada em seu computador pessoal, o mesmo envia-la para o servidor web utilizando HTTP. O servidor armazena as informações do usuário e a campanha enviada no formato XML. Depois, o mesmo faz uma leitura no arquivo XML e gera um arquivo correspondente em formato binário. O usuário deverá, então, utilizar o dispositivo móvel para solicitar a campanha desejada ao servidor; em seguida, o servidor enviará o arquivo binário que contém a campanha, no intuito de utilizar poucos recursos da rede e consequentemente ter custo mais barato para o usuário final. A figura abaixo ilustra este processo (Figura 41):

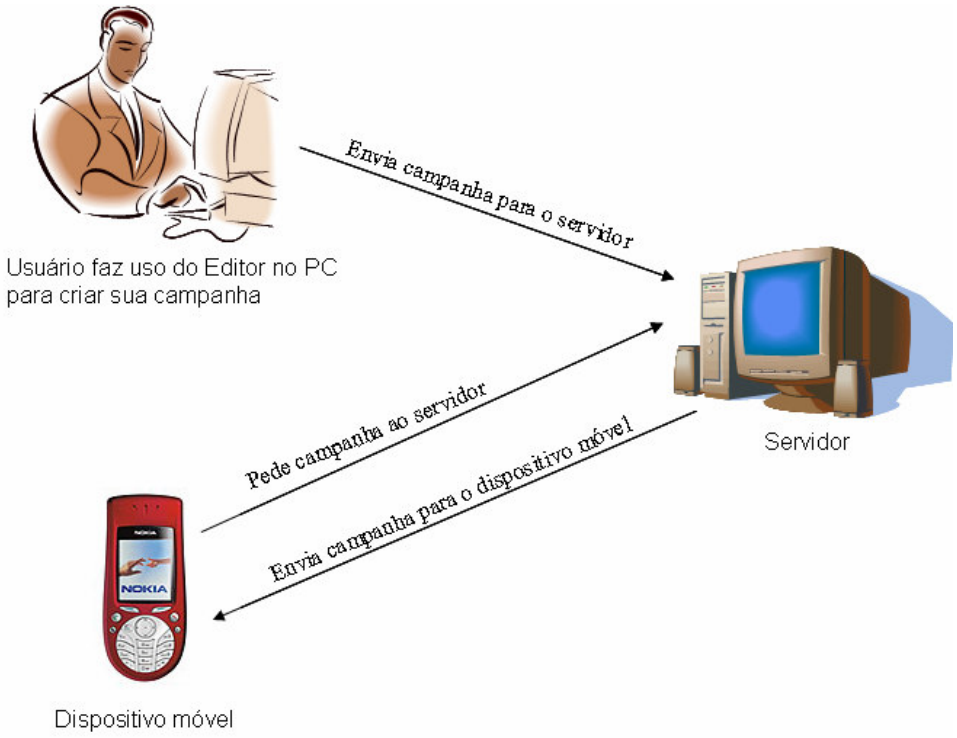


Figura 41 - Solução de teste para o jogador

Um servidor WEB simples, também foi construído durante a elaboração deste trabalho. Suas responsabilidades são: receber dados para criar um novo usuário no sistema, criar o novo usuário, receber uma campanha, armazena - lá, transformar esta campanha em um formato binário, para ser enviada para o dispositivo móvel, receber requisição do aparelho móvel e enviar campanha para o dispositivo. Este documento não exibirá mais detalhes do servidor WEB, pois os detalhes da sua implementação não são relevântes para o trabalho aqui apresentado.

5.3.3 Requisitos Funcionais

O editor do Space Runner para computador pessoal, provê alguns dos requisitos listados nas seções 4.7 e 5.2.1 e alguns requisitos específicos do jogo em questão. Abaixo segue uma lista destes requisitos:

- **[RF001] - O sistema deve prover um paleta de componentes ;**
- **[RF002] - O sistema deve criar campanhas;**
- **[RF003] - O sistema deve salvar campanhas;**

- **[RF004] - O sistema deve editar campanhas;**
- **[RF005] - O sistema deve criar níveis;**
- **[RF006] - O sistema deve editar níveis;**
- **[RF007] - O sistema deve remover níveis;**
- **[RF008] - O sistema deve criar um cenário inicial;**
- **[RF009] - O sistema deve prover um modo de testes;**
- **[RF018] - O sistema deve realizar checagem de validação do estágio;**
- **[RF020] - O sistema deve denominar jogos e níveis;**
- **[EM_RF002] – O sistema deve verificar se o herói foi inserido;**
- **[EM_RF003] – O sistema deve verificar se o portal foi inserido;**
- **[EM_RF004] – O sistema de verificar se existe ao menos um diamante.**

Os requisitos de criação de uma campanha e de criação do cenário inicial, providos pelo editor, podem ser melhor visualizados nas figuras abaixo (Figura 42 e Figura 43):

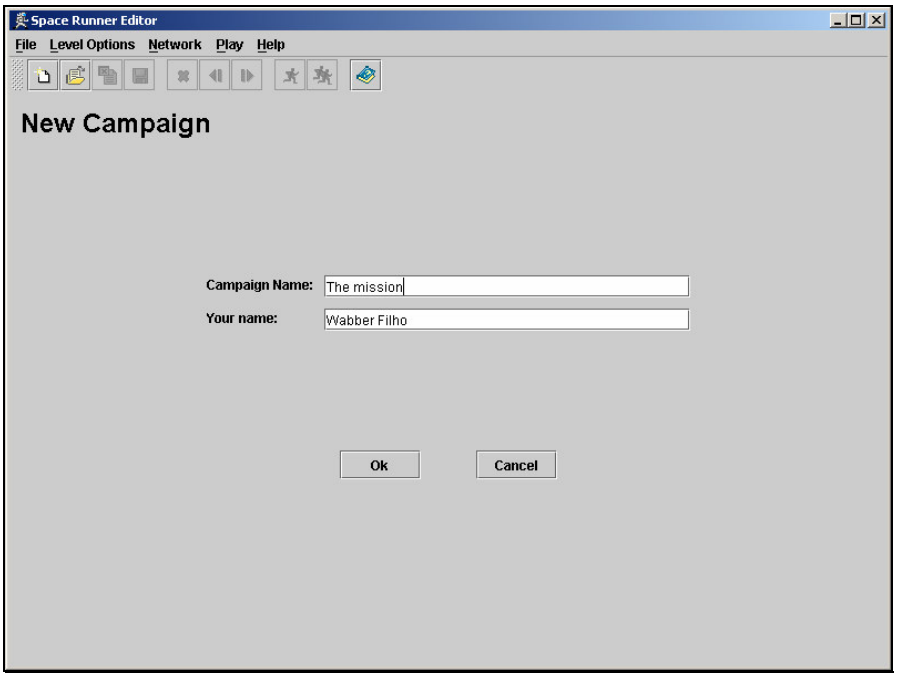


Figura 42 – Criando uma campanha

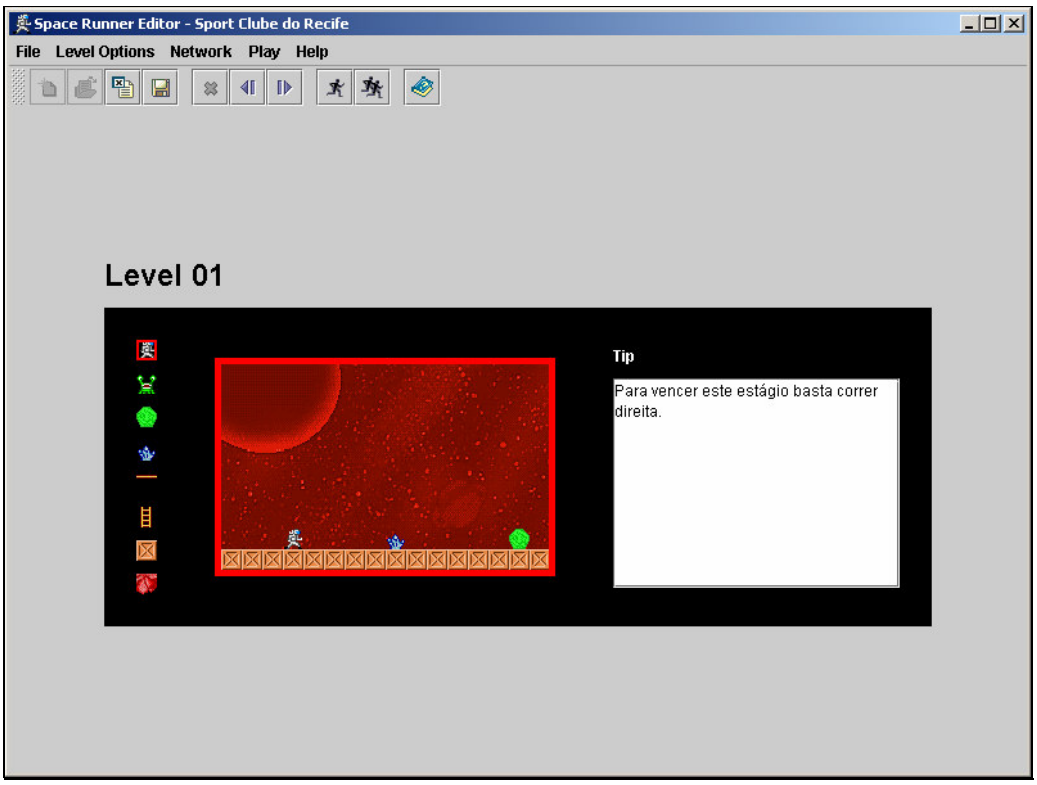


Figura 43 - Desenhando um cenário

Como mencionado anteriormente, foi necessário implementar alguns requisitos específicos para o editor em questão e alguns deles dão suporte aos requisitos acima listados. Os requisitos específicos estão listados abaixo:

[SR_RF001] – O sistema deve permitir a existência de atributos para uma campanha: Uma campanha pode conter o seu nome e do seu criador

[SR_RF002] – O sistema deve impor limite de níveis: Uma campanha pode ser constituída de 1 até 10 níveis.

[SR_RF003] – O sistema deve salvar campanhas em formato XML**:** Uma campanha é salva em formato XML.

[SR_RF004] – O sistema deve navegar entre os níveis: O editor permite a navegação entre os níveis.

[SR_RF005] – O sistema deve prover modo para testes de uma campanha para desenvolvedores: O editor permite modificar o arquivo com os níveis padrões do jogo com os níveis de uma campanha criada no editor.

[SR_RF006] - O sistema deve prover modo para testes de um nível para desenvolvedores: Provê a mesma funcionalidade do requisito anterior, entretanto apenas o nível que está sendo exibido é inserido no novo arquivo.

[SR_RF007] - O sistema deve prover modo de testes rápido de uma campanha: O editor permite que o usuário possa jogar a campanha editada. Isto é feito alterando os arquivos "jar" e "jad" do jogo, e chamando o simulador de J2ME.

[SR_RF008] - O sistema deve prover modo de testes rápido de um único nível: Provê à mesma funcionalidade do requisito anterior, entretanto apenas o nível que está sendo exibido é inserido no novo arquivo.

[SR_RF009] – O sistema deve criar usuários no servidor: O editor permite que o usuário possa se cadastrar no servidor (Figura 44).

[SR_RF010] – O sistema deve enviar campanhas para o servidor: O editor permite ao usuário enviar uma campanha salva para o servidor. Esta funcionalidade serve para que o usuário possa fazer "*download"* da campanha enviada, para o seu celular.

[SR_RF011] – O sistema deve armazenar Logs: O editor faz uso de dois arquivos de log.

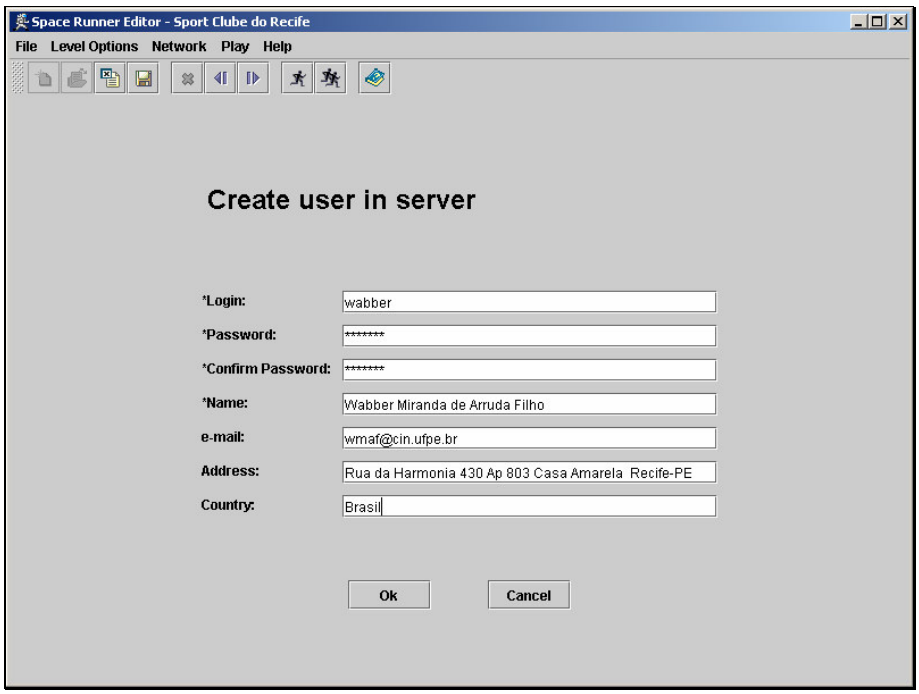


Figura 44 - Criando um usuário no servidor

5.3.4 Arquitetura

Esta subseção apresenta de forma superficial, a arquitetura utilizada no editor do Space Runner, para computador pessoal. O mesmo foi construído em Java (J2SE), fazendo uso dos conceitos de Orientação a Objetos. O editor é constituído por cinco pacotes: *Util, Network, Logic, Facade* e *GUI*.

O pacote *Util* é responsável por conter as classes que são utilizadas pelos outros componentes do editor, com a finalidade de prover funções gerais aos mesmos. Suas classes são:

- `Constants`: Armazena todas as constantes do sistema.
- `SpaceDynamicProperties`: Representa propriedades do sistema que podem ser iniciadas ou modificadas, enquanto o sistema está sendo utilizado. Um exemplo é armazenar o *login* do usuário.
- `EmulatorNotFoundException`: Exceção que é levantada, caso o simulador J2ME, não seja encontrado.
- `SpaceException`: Exceção geral do sistema.

O componente *Network* provê uma forma de comunicação HTTP com o servidor WEB. Ele é constituído pelas classes:

- `DocumentHTML`: Responsável por realizar conexões HTTP.
- `Answer`: Representa a resposta vinda do servidor WEB.

O pacote *Logic* contém classes que representam estruturas do sistema, como campanha e nível, como também a persistência do mesmo. Suas classes são:

- `PositionMap`: Representa uma posição de um componente no mapa.
- `Level`: Representa um nível do jogo.
- `Campaign`: Representa uma campanha do jogo.
- `Persistence`: É responsável pela persistência do sistema.

O pacote *Facade* é constituído por uma única classe, chamada de `Facade`. Este pacote é responsável pelos métodos de negócios do sistema e pela comunicação do sistema com o pacote de interface gráfica, GUI.

O pacote *GUI (Graphic User Interface)* é responsável pela interface gráfica do sistema. Suas classes são:

- `MainScreen`: Possui a função de gerenciar as telas do sistema.
- `DefaultScreen`: Trata-se de uma super classe que deve ser estendida pelas classes que representam telas do sistema.
- `PresentationScreen`: Representa a tela de apresentação do sistema.
- `NewCampaignScreen`: Representa a tela que permite ao usuário criar uma nova campanha.
- `NewLevelScreen`: Representa a tela que permite ao usuário criar uma novo nível.
- `CreateSceneryScreen`: Representa a tela que permite ao usuário montar o cenário inicial do nível. A mesma faz uso das classes `Background` e `Item` para isto.
- `Item`: Representa os itens do jogo e suas imagens.
- `Background`: Representa o plano de fundo do cenário e o local onde os itens do jogo devem ser inseridos.

- `ConfirmationScreen`: Representa a tela que exibe um pergunta de confirmação do tipo "sim ou não" para o usuário.
- `CreateUserServerScreen`: Representa a tela que permite ao usuário criar um usuário no servidor WEB.
- `SendCampaignScreen`: Representa a tela que permite ao usuário enviar uma campanha para o servido WEB.
- `BrowserScreen`: Esta classe é capaz de exibir arquivos HTML como *browser*.
- `Run`: Classe responsável por iniciar a aplicação.

5.4 Pacman

O segundo editor, desenvolvido para computador pessoal, foi para o jogo Pacman. O jogo é um clássico do vídeo game que foi adaptado para dispositivos móveis pela equipe do CESAR (Figura 45). Esta seção fala um pouco sobre o jogo e sobre o editor construído para o mesmo.

5.4.1 O Jogo

O jogo ocorre em um labirinto bi-dimensional, onde se encontra o personagem principal conhecido como *pacman*. O jogador pode mover o *pacman* em qualquer sentido do labirinto desde que não esbarre em nenhuma parede. Seu objetivo é coletar todas as cápsulas do nível, sem ser pego pelos fantasmas que assombram o labirinto.

Ao capturar uma cápsula especial, conhecida como cápsula de força, o *pacman* passa a ter o poder de machucar seus inimigos por um determinado espaço de tempo. Isto faz com que, os fantasmas tentem fugir do personagem principal. Após capturar todas as cápsulas de um nível, o jogador será levado ao próximo estágio.

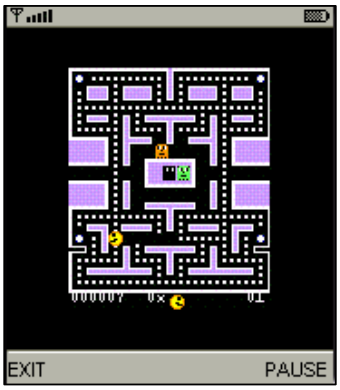


Figura 45 - Jogo PacMan para celular

5.4.2 O Editor e seus Requisitos Funcionais

O Editor construído, além de possuir a maioria das funcionalidades contempladas pelo editor do Space Runner, buscou alcançar diferentes requisitos de interface. Por exemplo, no Pacman Editor (Figura 46) é possível visualizar mais de um nível ao mesmo tempo, sendo permitida ainda, uma fácil navegação entre eles, sem necessariamente passar os níveis na ordem em que eles foram desenvolvidos.

O editor do Pacman apresenta outras diferenças em relação ao primeiro editor implementado, principalmente, no que diz respeito às regras de validação do cenário. Por exemplo, neste jogo existe o elemento de bônus, que não necessariamente deve ser colocado no cenário, mas quando colocado, não pode ser repetido.

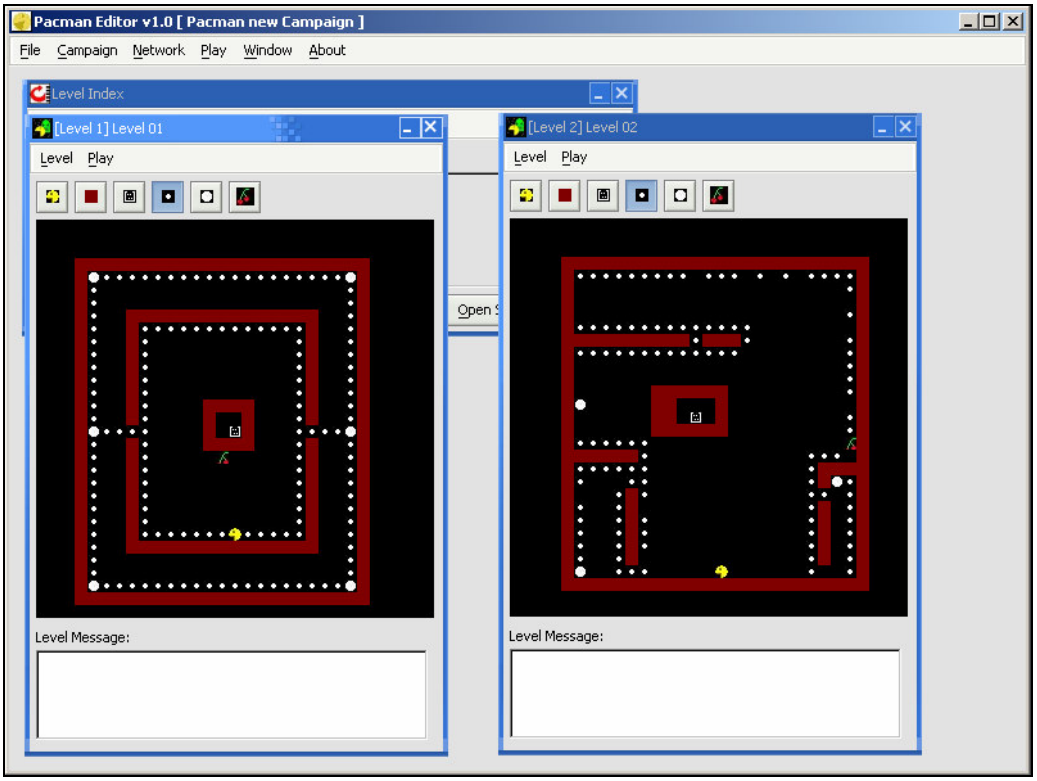


Figura 46 - Pacman Editor

O editor permite ainda, que várias informações sejam editadas por nível, como por exemplo, informações relativas à pontuação dos objetos do cenário (pastilhas, cápsulas do poder, bônus e fantasmas comidos). Abaixo segue uma lista dos requisitos específicos para este editor.

[PM_RF001] – O sistema deve exibir mais de um nível: O editor permite que mais de um nível seja visualizado ao mesmo tempo

[PM_RF002] – O sistema deve navegar para qualquer nível: O editor permite ao usuário ir de um nível para qualquer outro dentro de uma campanha.

[PM_RF003] – O sistema deve editar vários atributos de um nível: O editor permite ao usuário alterar vários atributos como: quantidade de fantasmas inteligentes, quantidade de bônus que deve ser dado e valor dos pontos ganhos para as várias tarefas do jogo (Figura 47).

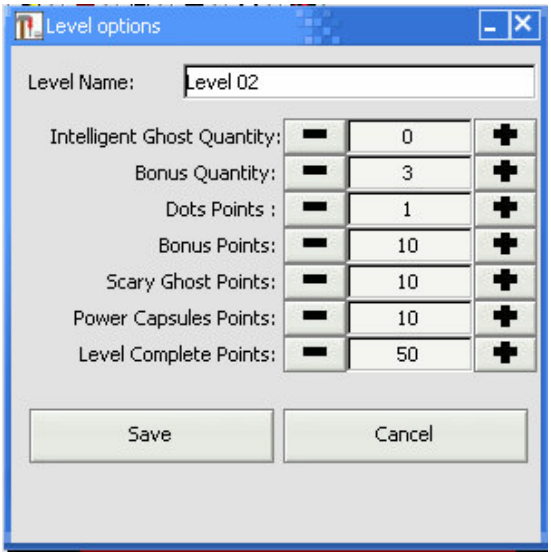


Figura 47 - Edição de vários atributos de um nível

5.4.3 Arquitetura

O Pacman Editor também é constituído por cinco pacotes: *Util, Network, Logic, Facade* e *GUI*. Alguns destes pacotes, no entanto, sofreram pequenas alterações em relação ao editor do Space Runner. A seguir será apresentada uma breve descrição destas modificações.

O pacote *Util* traz como novidade as classes: `PacXMLConstants` e `Images`. A classe `PacXMLConstants` tem a função de separar as constantes utilizadas no arquivo XML. A classe `Images` contém todas as imagens utilizadas pelo sistema.

No pacote *GUI* as principais diferenças estão nas classes: `LevelMenuBar` e `LevelToolBar`. A classe `LevelMenuBar` representa o menu utilizado no sistema, sua existência permite uma maior organização a nível de código. A classe `LevelToolBar`, segue a mesma filosofia do `LevelMenuBar`, representando a barra ferramentas do editor.

5.5 Pod Race Editor

O terceiro editor construído foi feito para o jogo Pod Race (Figura 48), um jogo de Rally espacial, construído por Alexandre Luiz G. Damasceno, Igor Sampaio e Borje Felipe Fernandes Karlsson para participar do concurso Mobilidade, mencionado anteriormente. O jogo obteve o segundo prêmio na categoria de jogos para celulares da série 60 da Nokia, ficando atrás apenas do jogo Space Runner.

Figura 48 - Jogo Pod Race

5.5.1 O Jogo

No jogo Pod Race, o jogador pode participar de corridas em Plutão, Andrômeda, Orion e Lua terrestre. Além de lutar por posições na pista, o jogador deve evitar colisões com outros objetos, pois além de retardar a nave pilotada, acarreta em perda de energia pela mesma. Caso a nave perca muita energia o jogador será instantaneamente desclassificado.

O jogo também possui um modo de tomada de tempo, no intuito de permitir ao jogador adquirir mais prática no jogo. Por fim, o jogo possui uma área para armazenar os recordes obtidos pelo jogador.

5.5.2 O Editor e seus Requisitos Funcionais

A interface gráfica do Pod Race Editor (Figura 49) foi completamente baseada na interface do editor do *Space Runner*. Isto foi feito, por se acreditar que esta era uma interface mais fácil de ser utilizada pelo usuário final.

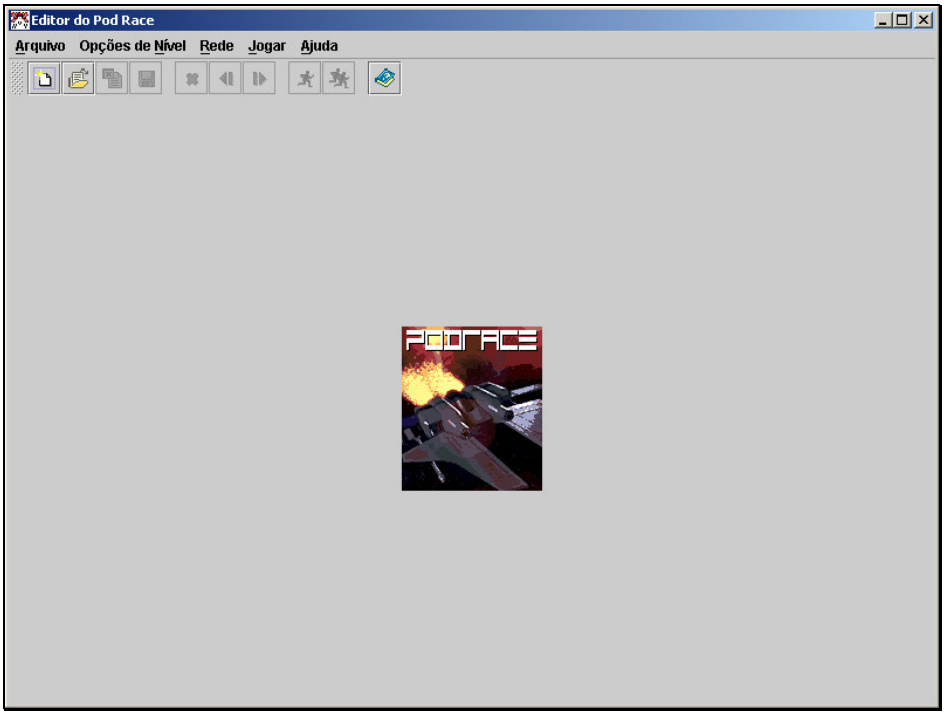


Figura 49 – Editor do Pod Race

A principal peculiaridade deste editor, é que por se tratar de um editor para construir um circuito, os "pedaços" de pista precisam estar interligados, ou seja, o usuário não pode colocar os ladrilhos em qualquer ponto do cenário. Para resolver este problema, sempre que o usuário clicar em um "pedaço" de pista, o editor já o concatena ao último inserido (Figura 50). Assim sendo, o editor precisa conhecer a posição que o último ladrilho foi inserido, o seu tipo e a direção (esquerda-direta; cima-baixo) em que a pista está progredindo.

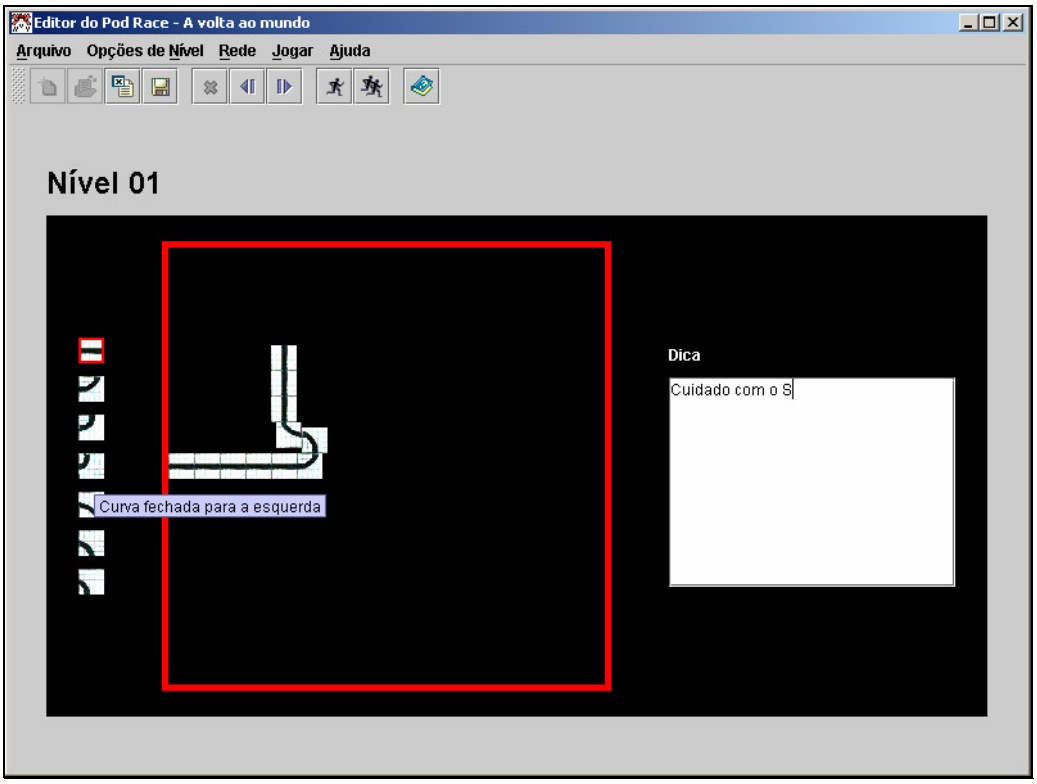


Figura 50 – Construindo um circuito

A seguir será apresentada a lista de requisitos que foram específicos para este editor:

[PR_RF001] – O sistema deve concatenar pedaços pista: Sempre que um "pedaço" de pista é clicado na paleta de componentes, o editor concatena o mesmo ao circuito existente no cenário.

[PR_RNF001] – O sistema de prover internacionalização: O editor permite que todas as mensagens e palavras exibidas na tela sejam internacionalizadas, ou seja, podem estar na língua utilizada pelo computador do usuário.

5.5.3 Arquitetura

Além dos cinco pacotes utilizados nos outros editores (*Util, Network, Logic, Facade* e *GUI*), o Pod Race Editor ganhou mais dois pacotes: *Persistence* e *Languages*. A

existência do pacote *Persistence* gerou apenas uma mudança lógica a nível arquitetural, mas não causou nenhum impacto no nível de implementação.

O editor ganhou também, três arquivos de propriedades. O primeiro serve para representar propriedades do sistema, como número máximo de níveis, tamanho da tela, tamanho do ladrilho, etc. Os outros dois arquivos armazenam as mensagens que devem ser exibidas na tela, nas línguas inglesa e portuguesa.

O pacote *Languages* é responsável pela internacionalização do sistema. O mesmo é provido apenas pela classe `Messages`, que por sua vez, tem a função de retornar as mensagens que serão exibidas na tela. Para receber uma determinada mensagem, o desenvolvedor deve passar a chave, palavra que identifica uma propriedade, correta para mesma.

O pacote *Util* traz de novidade as classes `Point` e `EditorProperties`. A classe `Point` representa um ponto no mapa e a classe `EditorProperties` é responsável por ler os arquivos de propriedades do editor e retorna-las quando as chaves da mesmas são recebidas.

6 GEF – Game Editor Framework

Depois de construídos os três editores, foi feita uma abstração do que era comum a eles, para se construir o *framework* chamado GEF – Game Editor Framework. O GEF foi construído como um *framework* caixa-branca, devido ao fato de seu desenvolvimento ser feito de forma mais simples e seu entendimento pelos usuários ser supostamente mais fácil. Isso ocorre principalmente, porque um *framework* de caixa-branca é baseado em herança, ou seja, permite que o desenvolvedor reutilize os métodos já existentes, apenas adaptando-os as suas necessidades, e com isso tornando o *framework* bastante flexível.

Este capítulo apresentará o GEF e como utilizá-lo A primeira seção apresentará uma visão geral da arquitetura utilizada, em seguida será comentado quais são os arquivos de recursos utilizados pelo mesmo e onde podem ser encontrados; a terceira seção abordará os arquivos de propriedades que são utilizados e, por último, serão abordados em maiores detalhes todos os pacotes de implementação deste *framework*.

6.1 Visão da Arquitetura

O *framework* aqui apresentado foi projetado para exigir a menor quantidade possível de linhas de código a serem escritas por parte do desenvolvedor. No intuito atingir este objetivo, além de classes o GEF faz uso de outros tipos arquivos que armazenam informações diversas sobre o comportamento do sistema e informações que devem ser exibidas para o usuário final de acordo com a aplicação que esteja sendo construída. Uma visão de alto nível sobre a arquitetura é ilustrada na figura abaixo (Figura 51):

Figura 51- Arquitetura em uma visão de alto nível

Os arquivos de recursos representam imagens e arquivos HTML que podem ser utilizados pelo sistema. Os arquivos de propriedades representam variáveis, constantes e mensagens que podem ser exibidas pelo sistema. A camada nomeada como "Campanhas Salvas" representa os conjuntos de níveis que devem ser salvos e recuperados pelo editor.

A camada nomeada de "Aplicação" representa o código feito pelo desenvolvedor para construir um editor específico para o seu jogo. Para construir o editor a aplicação deve fazer uso dos arquivos de recursos e arquivos de propriedades fornecidos pelo *framework,* além de se comunicar com as classes providas pelo GEF através de heranças e composições.

As classes do GEF foram divididas em camadas com o intuito de facilitar o entendimento e manutenção do *framework.* Suas camadas são (Figura 52):

- *GUI*: responsável pela interface gráfica do sistema;
- *Logic*: responsável pelas regras de negócios;
- *Persistence*: responsável pela persistência do sistema.
- *Network*: responsável pela comunicação com outros sistemas através da rede.

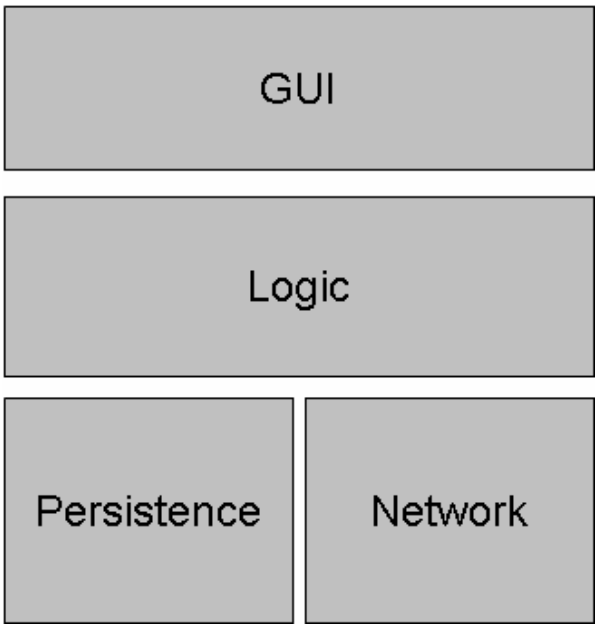


Figura 52 - Arquitetura das classes do GEF

O usuário final deve utilizar a *GUI* para se interagir com o sistema, a *GUI* repassa a requisição para a camada *Logic* que deve responder a requisição podendo fazer uso das camadas de *Persistence* ou *Network* caso necessário.

As próximas seções apresentarão em maiores detalhes o que são e a finalidade dos arquivos de recursos e de propriedades. Em seguida, será exibida em detalhes a organização das classes do GEF.

6.2 Arquivos de Recursos

Com o objeto de ter um *framework* mais flexível e com a menor necessidade de codificação possível por parte do usuário. O GEF contém não apenas código, mas também recursos, como imagens e arquivos de extensão html, que podem ser utilizados na construção de um novo editor. Os arquivos de imagens e html podem ser encontrados no diretório “res”, do GEF. Estes contêm basicamente imagens para a barra de ferramentas e arquivos html preparados para que o desenvolvedor construa o “sobre”, contendo informações sobre o sistema, e a “ajuda” do mesmo.

Com a existência destes arquivos o desenvolvedor, não precisará alterar o código para trocar as imagens dos botões da barra de menu, nem para construir o “sobre” e a “ajuda” do sistema. Pois os últimos podem ser feitos em HTML e o GEF se encarrega de exibi-los.

6.3 Arquivos de Propriedades

O GEF foi construído para se tornar um editor flexível e facilmente parametrizável. Pensando nisto, o desenvolvedor deve fazer uso de dois ou mais arquivos textos, que servirão para configuração e internacionalização do mesmo. O GEF também faz uso de um outro *framework* para gerenciamento de Log, chamado Log4J[34] da Apache [50].

Os arquivos de propriedade deverão ter seu caminho indicado na variável de ambiente chamada "*classpath*". O "*classpath"* é uma das variáveis de ambiente mais importantes do Java. Essa variável é importante tanto para compilação, quanto para execução de um programa Java, pois especifica onde estão armazenados os arquivos e bibliotecas necessários, tanto para a compilação, quanto para a execução desse programa.

O primeiro arquivo a ser descrito é o "editor.properties". Este é responsável pelas configurações básicas do sistema e as configurações utilizadas pelo Log4J. A seguir, será exibido e explicado o significado das propriedades:

```
########## Main Properties ######################
GAME=Put here the game name
MAXIMUM_NUMBER_LEVEL=10
SCREEN_WIDTH=800
SCREEN_HEIGHT=600
TILE_WIDTH=16
TILE_HEIGTH=16
NUMBER_TILES_X=16
NUMBER_TILES_Y=10
```

Tabela 4 – Arquivo de propriedades

- GAME: Propriedade onde deverá ser colocado o nome do jogo para o qual o editor foi desenvolvido.
- MAXIMUM_NUMBER_LEVEL: Indica o número máximo de níveis que podem existir em uma campanha.
- SCREEN_WIDTH: Representa a largura do editor em pixels.
- SCREEN_HEIGTH: Representa a altura do editor em pixels.
- TILE_WIDTH: Indica a largura utilizada por um ladrilho.

- TILE_HEIGTH: Indica a altura utilizada por um ladrilho.
- NUMBER_TILES_X: Representa o número de ladrilhos que existem ou podem existir na horizontal.
- NUMBER_TILES_Y: Representa o número de ladrilhos que existem ou podem existir na vertical.

```
############### EMULATOR ##########################
MIDLET_JAR_SIZE=MIDlet-Jar-Size:
EMULATOR_PARAMETER=-Xdescriptor
PATH_EMULATOR=C:/WTK20/bin/emulator.exe
############### PATH ##########################
LEVEL_FILE=map.bin
PATH_LEVEL_FILE_IN_JAR=space/res/
SERVER_URL=http://localhost:8080/ServletEditor
```

Tabela 5 - Arquivo de propriedades

- MIDLET_JAR_SIZE: Indica apenas a propriedade em um arquivo "jad", na qual o GEF deverá procurar para saber o tamanho do arquivo "jar". Esta propriedade será importante na funcionalidade em que o editor recria os arquivos "jar" e "jad" do jogo, e chama o simulador de aplicações J2ME para testar os níveis criados no editor. Esta propriedade não deverá ser modificada, a não ser que algo seja modificado na especificação J2ME.
- EMULATOR_PARAMETER: Esta propriedade indica quais parâmetros que devem ser passados para o simulador J2ME.
- PATH_EMULATOR: Indica o caminho onde o simulador J2ME pode ser encontrado.
- LEVEL_FILE: Representa o nome do arquivo, geralmente binário, que armazena os níveis dentro do arquivo "jar" de um jogo J2ME.
- PATH_LEVEL_FILE_IN_JAR: Representa o caminho dentro do arquivo "jar", onde se encontra o arquivo que armazena os níveis do jogo.
- SERVER_URL: Indica o caminho onde se encontra o servidor de níveis na WEB, para que o GEF possa enviar as campanhas construídas e que futuramente serão baixadas para o celular.

```
########## Log Configuration ###############
log4j.rootLogger=DEBUG, RF, A1, FILE_ERROR
#log4j.rootLogger=ERROR, RF, A1, FILE_ERROR

######### text file ###########
log4j.appender.RF=org.apache.log4j.FileAppender
log4j.appender.RF.file=editorLog.log
log4j.appender.RF.layout=org.apache.log4j.PatternLayout
log4j.appender.RF.layout.ConversionPattern=%d %p [%t - %c] => %m%n

######### console ###########
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d %p [%t - %c] => %m%n

######### error file ###########
log4j.appender.FILE_ERROR=org.apache.log4j.FileAppender
log4j.appender.FILE_ERROR.file=editorLogError.log
log4j.appender.FILE_ERROR.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE_ERROR.layout.ConversionPattern=%d %p [%t - %c] =>
%m%n

log4j.appender.FILE_ERROR.Threshold=WARN
```

Tabela 6 - Arquivo de propriedades

As outras propriedades dizem respeito ao Log4J, e não serão detalhadas neste documento. O desenvolvedor precisa saber apenas que o GEF vem configurado para utilizar três logs: Um que será exibido na tela com nível de detalhe de *debug*, para facilitar a vida do desenvolvedor em caso de erro, outro que será impresso no arquivo editorLog.log, também com o nível de detalhe de *debug,* também com o intuito de ajudar o desenvolvedor, e por último o no arquivo editorLogError.log, com nível de detalhe de *error,* que pode ser enviado a equipe de desenvolvimento pelo usuário final no intuito de melhorar a aplicação em questão.

Os outros arquivos de propriedades são responsáveis pela internacionalização do sistema. O GEF vem com dois arquivos de internacionalização:

- editormessages_pt.properties: Possui as mensagens do sistema na língua portuguesa.

- editormessages.properties: Contém as mensagens do sistema em inglês, ele é utilizado quando um computador está configurado para um idioma diferente de português.

Caso o desenvolvedor queira utilizar outra linguagem no sistema, basta que ele crie outro arquivo de propriedade, utilizando a linguagem desejada e seguindo o padrão editormessages_xx.properties. Não será necessária nenhuma modificação no código fonte do sistema.

Abaixo, segue uma parte do arquivo editormessages_pt.properties para ilustração:

```
#### Menus #####
FILE=Arquivo
NEW_CAMPAIGN=Nova Campanha
OPEN_CAMPAIGN=Abrir Campanha
####  ####

#### Mnemonic Menus ####
MNM_FILE=A
MNM_NEW_CAMPAIGN=N
MNM_OPEN_CAMPAIGN=A
####  ####

#### Mensagens Diversas ####
ADDRESS=Endereço
CAMPAIGN_NAME=Nome da campanha
CANCEL=Cancela
CONFIRM_PASSWORD=Confirmação da senha
####  ####

#### Mensagens de Erro ####
CAN_NOT_OPEN_THE_SCREEN_ERROR=Não é possível abrir esta tela.
CAN_NOT_INSERT_MORE_ITENS_IN_LEVEL_ERROR=Não é possível inserir mais
itens neste nível.
CAN_NOT_ESTABLISH_CONNECTION_ERROR=O editor não pode estabelecer
conexão com o servidor.
####  ####

#### Caminho para arquivos em português ####
```

```
ABOUT_PATH=file:res/about/index.htm
HELP_PATH=file:res/help/index.htm
####  ####
```

Tabela 7 - Arquivo de mensagens

6.4 Pacotes do GEF

Pode-se dividir o GEF em seis em pacotes: *GUI, Network, Logic, Languages, Persistence* e *Util* (Figura 53).

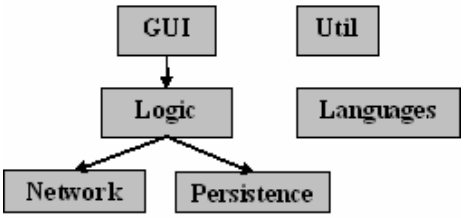


Figura 53 - Pacotes do GEF

A seguir, será apresentada uma visão superficial, de cada pacote e como eles devem ser utilizados pelo desenvolvedor.

6.4.1 Util

O pacote *Util* provê classes que são utilizadas por outros pacotes do GEF e que poderão ser utilizadas pelo código do novo editor, que está sendo desenvolvido. O pacote é composto por nove classes, que poderão ser vistas abaixo:

- `EditorProperties`: Classe responsável por ler o arquivo de propriedades e retornar o valor de uma propriedade específica como, inteiro ou String.
- `EditorDynamicProperties`: O GEF permite que o desenvolvedor crie propriedades dinâmicas, que terão seus valores definidos pelo usuário final. Como exemplo, o editor poderia guardar o nome e outras informações do usuário. Para isto, o desenvolvedor pode utilizar métodos desta classe para criar e recuperar novas propriedades em tempo de execução.

- `EditorException`: Esta classe encapsula a maioria das exceções produzidas pelo GEF. Uma peculiaridade é que um de seus construtores aceita uma mensagem a ser passada para o desenvolvedor e esta mensagem será escrita no log do sistema.
- `EmulatorNotFoundException`: Esta classes encapsula uma exceção que é lançada quando o emulador J2ME não é encontrado.
- `Point`: Representa um ponto no espaço, sendo útil para a parte lógica do sistema.
- `EditorImages`: Esta classe tenta centralizar todas as imagens utilizadas pela a aplicação. Ela possui algumas imagens *default* do *framework,* porém pode conter outras imagens adicionadas em tempo de execução, ou pode também ser estendida e ter o método abstrato "`public` **`abstract`** `void changeImages()`" re-implementado, modificando as imagens utlizadas pelo sistema.
- `EditorScreen`: Com a mesma filosofia da classe `EditorImages`, esta classe centraliza as telas utilizadas pelo sistema, facilitando o trabalho do gerenciador de telas usado na interface gráfica do sistema. As telas são adicionadas a esta classe, de forma dinâmica em tempo de execução.
- `EditorConstants`: Esta classe representa as constantes utilizadas pelo sistema e serve apenas para organização do código do *framework*, o desenvolvedor não precisa conhece-la.
- `XMLConstants`: Esta classe representa as constantes referentes aos arquivos XML utilizadas pelo sistema e também serve apenas para organização do código do *framework*, o desenvolvedor não precisa conhece-la.

A figura abaixo (Figura 54) representa as classes do pacote *Util* e mostra alguns de seus métodos e atributos.

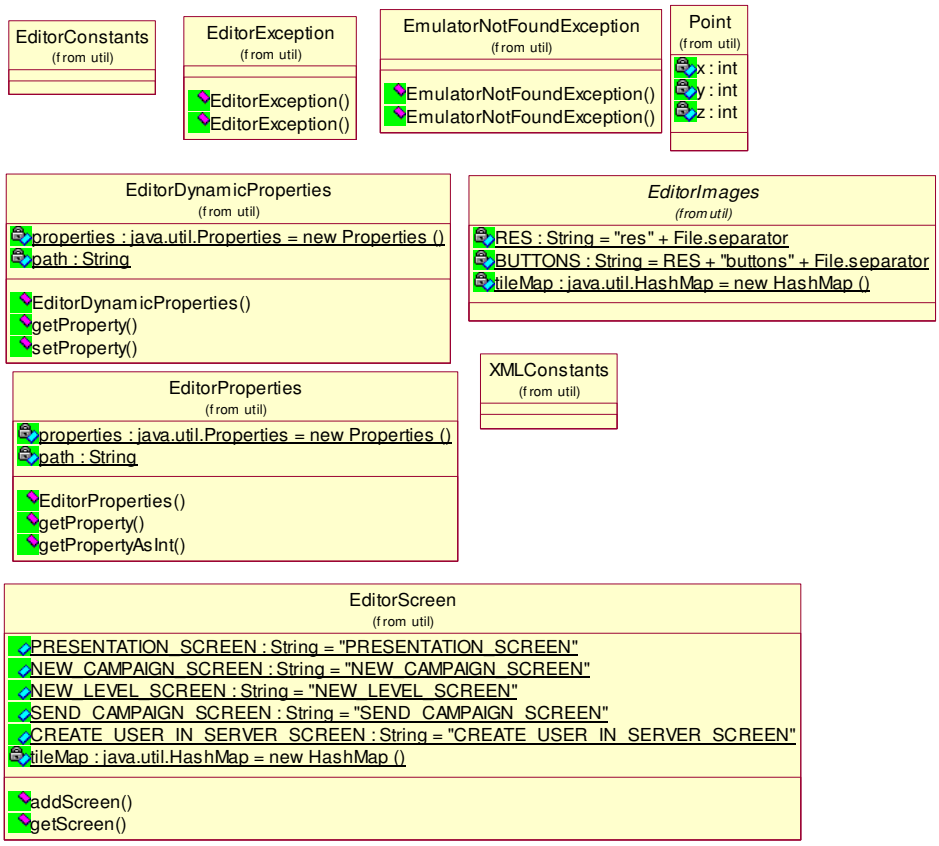


Figura 54 -Classes do pacote Util

6.4.2 Languages

O pacote *Languages* é responsável pela internacionalização do sistema. Ele possui apenas uma classe chamada `Messages`. Esta classe possui várias constantes, que servem como chaves para as mensagens que serão exibidas na tela. Além disto, ela possui quatro métodos estáticos que valem ser ressaltados:

- `getLocale` e `setLocale`: Que servem, respectivamente, para recuperar e modificar o objeto *Locale* (Objeto que representa o local e linguagem do computador) utilizado pelo sistema operacional.

- `getMessage(String key)`: Busca no arquivo de propriedades a mensagem a ser exibida na tela, de acordo com chave passada. Em geral, o desenvolvedor deverá chamar este método para exibir mensagens. Ex:
  `String msg = Messages.getMessage("key");`
- `getMessageChar(String key)`: Tem a mesma função do método anterior, mas retorna apenas um caractere.

6.4.3 Network

O pacote *Network* provê uma forma bastante simples para enviar e receber informações sobre o protocolo HTTP. Ele é constituído por quatro classes: `ConnectionHTTP, EditorContentHandler, Answer e EditorContentHandlerFactory.`

A classe `ConnectionHTTP` é a classe que realiza a conexão HTTP. Para utiliza - lá o desenvolvedor precisa fornecer algumas informações: A primeira informação necessária é o endereço que se deseja acessar. Isto pode ser feito através de um dos construtores de classe "`ConnectionHTTP(String url)`" ou, através do método público "`setUrlString(String url)`". O próximo passo é informar qual o método HTTP se deseja utilizar, GET ou POST, o desenvolvedor pode fazer isto, chamando o método "`setMethodRequest(String method)`" e passando os valores "GET" ou "POST". Caso, opte-se por usar o método POST é necessário preencher os campos e valores do suposto formulário *Web*, isto pode ser feito através do método "`addField(String key, String value)`" que adiciona um campo e seu valor ao formulário HTML. Pode-se também passar todos os campos e valores de uma única vez, através do método "`setFields(Hashtable newfields)`". Passadas estas informações, para fazer a requisição o desenvolvedor deve chamar o método "`getContent()`" e receber a String retornada. Ex:

```
ConectionHTTP doc = new ConectionHTTP("http://www.cin.ufpe.br");
doc.setMethodRequest("GET");
String content = doc.getContent();
```

A classe `EditorContentHandler` representa o conteúdo retornado da requisição antes de ser passado para o formato String. O desenvolvedor não precisa obter conhecimento sobre esta classe.

A classe `EditorContentHandlerFactory` serve como uma fábrica de `EditorContentHandler`. O desenvolvedor também não precisa obter conhecimento sobre esta classe.

A classe `Answer` representa a resposta do servidor. Para saber se esta foi sucesso ou falha, o desenvolvedor só precisa acessar o método `getStatus` provido pela mesma.

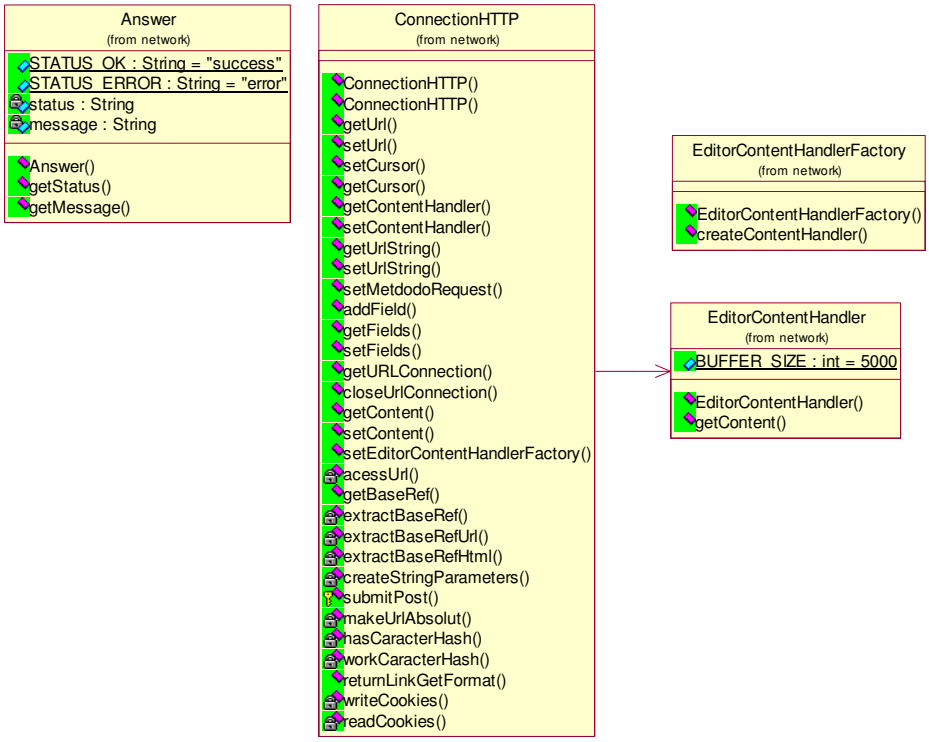


Figura 55 - Pacote Network

6.4.4 Logic

O pacote *Logic* contém classes que representam as estruturas dos sistemas, como uma campanha, um nível e um usuário, e também a Fachada do sistema, que é a classe responsável pelos métodos de negócio do mesmo. As quatro classes deste pacote são: `Level, Campaign, User` e `Facade`.

A classe `Level` representa os níveis de um jogo. O GEF já traz quatro atributos para esta classe, um identificador do nível, um nome para o mesmo, uma String onde o usuário possa armazenar uma mensagem ou uma dica, para o nível em questão e uma matriz bi-dimensional de inteiros, que pode servir para armazenar o nível propriamente dito. As dimensões desta matriz são definidas pelas propriedades NUMBER_TILES_X e NUMBER_TILES_Y do arquivo "editor.properties" comentado anteriormente. Seria um erro esperar que todos os jogos possam utilizar esta estrutura para representar seus níveis, no entanto, o GEF tenta apenas prover uma estrutura que pode ser comum a uma grande quantidade de jogos. Caso o desenvolvedor não queira utilizar os atributos providos pelo *framework* para representar seus níveis, só é estender esta classe, sobre-escrever os métodos que julgar necessário, e escrever outros atributos e métodos que o mesmo julgue necessário.

A classe `Campaign` serve basicamente para representar um conjunto de níveis. Seus atributos são, um nome para a campanha, o nome do criador da campanha, ou seja, o usuário e uma lista de níveis. Para facilitar o acesso aos níveis, a classe provê métodos para recuperar o total de níveis inserido, recuperar um nível, recuperar todos os níveis, modificar todos os níveis, adicionar e remover um nível.

A classe `User` representa uma estrutura para armazenar informações sobre o usuário final como login, senha, nome, endereço, e-mail, país, etc.

A classe `Facade é` provavelmente a classe mais importante do pacote *Logic*. Trata-se de uma classe abstrata que deve ser estendida pelo usuário e que provê todos os métodos de negócio do GEF. Toda requisição feita pela interface gráfica do sistema pede a fachada que a resolva, ou repasse esta requisição para o pacote responsável.

Como a `Facade` é uma classe que merece mais atenção, alguns de seus métodos públicos estão descritos abaixos:

- `getUserLogin()`: Retorna o Login do usuário vindo do arquivo de propriedades dinâmicas, propriedades criadas em tempo de execução, caso ele exista.
- `getCurrentLevelNumber()`: Retorna o número do nível que está sendo visualizado pelo usuário.
- `getCurrentLevel()`: Retorna o nível que está sendo visualizado pelo usuário.
- `setCurrentLevelNumber(int currentLevelNumber)`: Modifica o nível corrente.
- `goNextLevel()`: Navega para o próximo nível se possível.
- `goLevelBefore()`: Navega para o nível anterior se possível.
- `setCampaign(Campaign campaign)`: Altera a campanha em que está se trabalhando.
- `addLevel(Level newLevel)`: Adiciona um nível a campanha corrente
- `removeLevel()`: Remove um nível da campanha corrente.
- **`abstract`** `validateCampaign()`: É um método abstrato, que deve ser re-implementado pelo desenvolvedor. Sua função é informar se a campanha atual é válida ou não. Para validar uma campanha é necessário aplicar regras específicas do jogo para o qual o editor está sendo desenvolvido.
- `openCampaign(File file)`: Chama o pacote responsável pela persistência, para abrir uma campanha salva anteriormente.
- `saveCampaign(File file)`: Salva a campanha que está sendo criada ou editada pelo usuário. Para uma campanha ser salva, ela precisa passar pelo método de validação comentada anteriormente.
- `testCampaingWithSourceCode(File file)`: Chama o método do pacote de persistência, responsável por fazer as alterações no arquivo que armazena os níveis do jogo em questão.
- `testLevelWithSourceCode(File file)`: Possui a mesma funcionalidade do método anterior, contudo escreve apenas o nível corrente no arquivo de níveis do jogo.

- `testCampaignWithJarAndJad(File file)`: Chama o método do pacote de persistencia responsável por alterar os arquivos "jad" e "jar" do jogo J2ME em questão para testa-lo no simulador de aplicativos J2ME.
- `testLevelWithJarAndJad(File file)`: Possui a mesma funcionalidade do método anterior, contudo altera os arquivos "jad" e "jar" do jogo J2ME para conter apenas o nível que está sendo visualizado pelo usuário.
- `createUserInServer(User user)`: Cria um usuário no servidor.
- `sendCampaignServer(File campaignFile, String login)`: Envia uma campanha para o servidor.
- **`abstract`** `getNewInstanceLevel(String levelName, int id)`: Retorna uma instância de um novo nível. Este método é abstrato para que o desenvolvedor possa herdar da classe Level sem precisar reescrever outros métodos do GEF.

Para uma melhor visualização das classes do pacote *Logic*, e da interação da classe `Facade` com o pacote de Persistence, é exibida a figura abaixo (Figura 56):

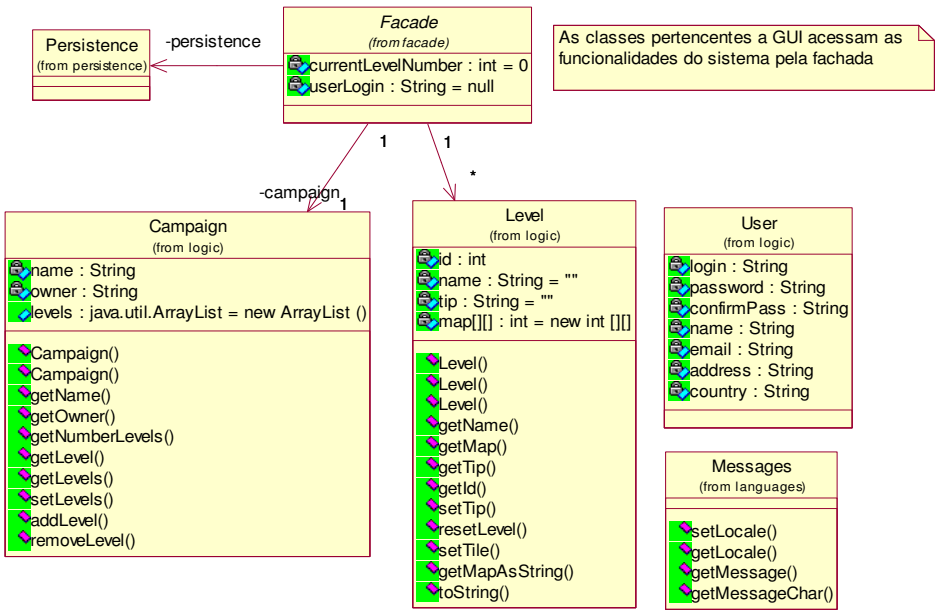


Figura 56 - Pacote Logic

6.4.5 Persistence

O pacote *Persistence* é de extrema importância, pois além de ser responsável por salvar, abrir e interpretar os arquivos XML, onde uma campanha é salva, ele também é responsável por modificar os arquivos "jad" e "jar" de um jogo desenvolvido em J2ME e também o arquivo que armazena os níveis do mesmo. O pacote é constituído apenas pela classe abstrata `Persistence` que deve ser extendida pelo desenvolvedor. A seguir, será apresentado o comportamento de cada método público provido por esta classe:

- `saveCampaign(Campaign campaing, File file)`: Salva uma campanha com seus atributos e níveis no formato XML, caso o desenvolvedor sobre-escreva as classes Campaign e Level, este método deve ser sobre-escrito também.
- `openCampaign(File file)`: Lê e interpreta um arquivo XML salvo anteriomente, seu objetivo é retornar uma campanha com seus atributos e níveis preenchidos. Este método também deverá ser sobre-escrito caso as classes Campaign e Level tenha sido também.
- `testWithSourceCode(ArrayList levels, File file)`: este método abre o arquivo do jogo que contém os níveis e chama o método abstrato fillNewLevels para que o mesmo preencha o arquivo de níveis com o formato exigido pelo jogo.
- `testWithJarAndJad(ArrayList levels, File jarFile)`: este método é responsável por alterar os arquivos "jad" e "jar" do jogo J2ME e chamar o simulador de aplicativos J2ME passando o jogo a ser testado. O método encontra os arquivos "jad" e "jar" do jogo e cria um arquivo temporário, em seguida, começa a ler o arquivo compactado "jar" como se fosse um arquivo zip e vai copiando-o para o arquivo temporário, até encontrar dentro do mesmo, o arquivo que realmente armazena os níveis do jogo. Depois, re-escreve este arquivo com os níveis criados no editor utilizando o método abstrado `fillNewLevels`. Em seguida, termina de copiar o arquivo "jar" no arquivo temporário, remove o arquivo "jar" e renomeia o arquivo temporário, altera o arquivo "jar" informando o novo tamanho do arquivo "jar" e finalmente chama o simulador de aplicativos J2ME passando o jogo que se deseja testar.

- **abstract** `fillNewLevels(OutputStream out, ArrayList levels)`: é um método abstrato que é responsável por preencher o arquivo de níveis do jogo com o formato exigido pelo mesmo. Devendo, portanto, ser sobre-escrito pelo desenvolvedor.
- **abstract** `getNewInstanceLevel (String levelName,String tip, int map[][], int id)`: Retorna uma instância de um novo nível. Este método é abstrato para que o desenvolvedor possa herdar da classe `Level` sem precisar re-escrever outros métodos do GEF.

6.4.6 GUI

O pacote GUI contém as classes responsáveis pela interface gráfica do sistema. Apesar do uso deste pacote ser opcional, recomenda-se que o desenvolvedor tente reaproveitar o máximo de código possível, no intuito de economizar tempo de projeto. O mesmo ainda pode optar pelo recurso de estender algumas classes, para tornar o seu editor personalizado.

Antes de utilizar este pacote é necessário entender que a maioria das classes representa uma tela do sistema, e para controlá-las, existe um gerenciador que decide quando cada tela deverá ser exibida. Além disto, existem também classes responsáveis pelo tratamento da barra de menu e da barra de ferramenta, que podem existir no editor.

A classe responsável pelo gerenciamento das telas é a `ManagerScreen`. Trata-se de uma classe que estende a classe `JFrame` do Java e possui como atributo, todas as classes que representam uma tela, além de métodos para gerenciá-las. Os principais métodos públicos desta classe são:

- `enableItens(int state)`: Habilita ou desabilita alguns itens do menu e da barra de ferramenta de acordo com o estado passado. Os dois estados providos pelo framework são: o CAMPAIGN_OPENED_STATE e o CAMPAIGN_CLOSED_ STATE, que representam à existência de uma campanha aberta ou fechada respectivamente.
- `actionPerformed(ActionEvent ev)`: É o método responsável por receber os eventos vindos da barra de menu e da barra de ferramenta, de acordo

com o menu ou botão ativado. Este método chamará o método responsável por responder ao evento passado. O método chamado, geralmente terá o seu nome iniciado com a palavra "*answer*". Isto foi feito para facilitar a compreensão e possível reimplementação de algum método pelo desenvolvedor.

- `changeScreen(String screenName)`: Modifica a tela que está sendo exibida pela nova tela, que teve seu o nome passado.
- `showErrorScreen(Exception e)`: Exibe uma janela com uma mensagem de erro na tela.
- `showErrorScreen(String message)`: Tem a mesma funcionalidade do método anterior, porém este método permite passagem de uma mensagem em formato de String.
- **`abstract`** `CreateSceneryScreen getNewInstanceCreateSceneryScreen()`: É um método abstrato que deverá ser reimplementado pelo desenvolvedor. Ele deverá retornar uma nova instância da classe `CreateSceneryScreen`, e esta reapresentará a tela onde o usuário cria um novo cenário para o nível.
- `answerMenuNewCampaign()`: Chama a tela onde uma nova campanha pode ser criada.
- `answerMenuOpenCampaing()`: Abre uma janela onde o usuário poderá escolher que campanha ele deseja abrir, e em seguida, o método chama a funcionalidade da Fachada responsável por abrir a campanha e finalmente cria uma instância da classe.
- `answerMenuCloseCampaign()`: Fecha a campanha que está sendo editada pelo usuário.
- `answerMenuSaveCampaing()`: Abre uma janela onde o usuário poderá escolher o caminho e nome do arquivo que irá conter a campanha e chama a funcionalidade da Fachada responsável por salvar esta campanha.
- `answerMenuSaveCampaingAs()`: Possibilita que a campanha que está sendo trabalhada seja salva com outro nome.
- `answerMenuNewLevel()`: Chama a tela que permite que um novo nível seja adicionado a campanha.
- `answerRemoveLevel()`: Pergunta se o usuário tem certeza que deseja remover o nível, em caso de resposta afirmativa, chama a funcionalidade

responsável pela remoção do nível, volta a exibir o nível anterior ou algum outro existente.

- `answerNextLevel()`: Avança para o próximo nível.
- `answerLevelBefore()`: Retrocede para o nível anterior.
- `answerMenuSendCampaign()`: Chama a funcionalidade de enviar a campanha aberta para o servidor. Caso seja a primeira vez que esta funcionalidade está sendo ativada, o GEF pede que o usuário preencha um formulário com suas informações pessoais.
- `answerMenuTestCampaingWithSourceCode()`: Chama a funcionalidade que substitui o arquivo de níveis do jogo, por um novo arquivo com os níveis criados no editor.
- `answerMenuTestLevelWithSourceCode()`: Possui a mesma funcionalidade do método anterior, com a diferença que apenas o nível que está sendo visualizado é escrito no arquivo.
- `answerMenuTestCampaignWithJarAndJad()`: Chama a funcionalidade que substitui o arquivo "jar" do jogo como o novos níveis, modifica o arquivo "jad" para ser compatível com o novo "jar" e chama o simulador de aplicativos J2ME.
- `answerMenuTestLevelWithJarAndJad()`: Possui a mesma funcionalidade do método anterior, com a diferença que apenas o nível que está sendo visualizado é escrito no arquivo "jar".
- `answerMenuHelp()`: Exibe a tela com a ajuda do sistema.
- `answerMenuAbout()`: Exibe a tela com as informações sobre o sistema.

Para utilizar esta classe o desenvolvedor precisa apenas estendê-la e implementar o seu método abstrato, contudo é importante saber da existência dos outros métodos, e quais são suas responsabilidades, para que o mesmo possa alterar e/ou adicionar qualquer comportamento desejado ao gerenciador de telas.

Como mencionado anteriormente, a maioria das classes do pacote GUI representam telas que podem ser oferecidas pelo editor, então para aproveitar melhor os conceitos da programação orientada a objetos, cada classe desta, deve estender uma super classe

chamada `DefaultScreen`. Esta é uma classe abstrata que estende uma outra classe do Java, denominada de `JPanel`. Seus principais métodos são:

- **`abstract`** `actionPerformed(ActionEvent e)`: Este método é abstrato para que todas as classes que representem uma tela do sistema sejam responsáveis por responder os eventos de componentes ocorrido em seus domínios, ou seja, cada tela é responsável pelas ações e funcionalidades que a mesma pode prover.
- **`abstract`** `void clean()`: Este método, também abstrato, deve prover a funcionalidade de limpar todos os componentes pertencentes a uma tela.
- `AddComponentInGridBagLayout(...,...)`: Existem quatro variações deste método providas nesta classe, com grande variação de parâmetros. Sua funcionalidade é adicionar um componente a um *container* que use *Grid Bag Layout*, ou seja, é uma forma de organizar os componentes de uma tela de forma um pouco mais amigável e diminuindo a quantidade de código escrito em relação a provida pelo Java[6]. Este documento não se aterá ao detalhe de explicar o que é um *Grid Bag Layout* ou qualquer outro tipo de *Layout* provido pelo Java. Maiores informações podem ser encontradas no site da Sun Microsystem [38].

O pacote GUI possui sete classes que estendem a classe `DefaultScreen`, ou seja, sete classes que representam telas do sistema. Para construir um nova classe que represente uma nova tela é aconselhável ao desenvolvedor que estenda da classe `DefaultScreen`. A seguir será comentado um pouco sobre as classes deste pacote.

A classe `PresentationScreen` corresponde a primeira tela do sistema, ela possui apenas uma figura que representar o jogo.

A classe `NewCampaignScreen` é a tela responsável por criar uma nova campanha, ela possui dois campos para que o usuário entre com o nome da campanha e o seu próprio nome. Esta tela possui também botões de “Ok” e “Cancel” (ou em uma língua

[6] Grid Bag Layout é o Layout mais poderoso do componente de desenho de telas do Java, no entanto o mesmo exigi uma quantidade de código muito grande para ser ajustado.

diferente dependo dos arquivos de propriedades e o local onde o sistema está sendo executado) para confirmar ou cancelar a operação respectivamente.

A classe `NewLevelScreen é a` tela que permite que um novo nível seja criado na campanha, o único campo provido pelo *framework* serve para armazenar o nome do nível. O GEF sugere que o nome do nível seja "Level x", onde x é o número do nível em questão.

A classe `CreateUserServerScreen` apresenta um formulário solicitando algumas informações pessoais do usuário, para a criação de uma conta no servidor. As informações requisitadas são: login, senha, nome, e-mail, endereço e país. Após o usuário preencher as informações e clicar no botão "Ok", esta classe chamará a funcionalidade do sistema responsável por enviar estes dados para o servidor.

A classe `SendCampaignScreen` representa uma tela que será exibida, uma vez que o usuário foi criado no servidor. Ela apresenta uma pergunta, questionando se o usuário deseja enviar a campanha para o servidor, e o mesmo então, deverá clicar no botão "Sim" ou "Não" de acordo com sua vontade.

A classe `BrowserScreen` é capaz de exibir arquivos HTML como um *browser*, e ela é utilizada para exibir a "ajuda" e as informações do "sobre" do sistema. Esta classe faz uso de uma classe interna chamada `HelpLinkListener`, que possibilita a navegação por *links* na página HTML.

Para uma melhor visualização destas classes, é apresentada a figura abaixo (Figura 57):

Figura 57 - GUI 1

A última classe que representa uma tela é a classe abstrata `CreateSceneryScreen`, e esta será comentada em mais detalhes devido a sua importância. Sua responsabilidade é permitir que o usuário crie os seus cenários. Para entender como esta classe funciona é fundamental entender a funcionalidade das classes Background e Item (Figura 58).

A classe Item representa um item do jogo (um ladrilho), no qual o usuário pode clicar e adiciona-lo ao cenário que está construindo. Esta classe possui três atributos importantes, um identificador do item, um identificador do ladrilho para o jogo e a imagem do item. Ela ainda implementa a interface `MouseListener`, com o intuito de responder a eventos provenientes do *mouse*.

A classe abstrata Background representa o plano de fundo do cenário, onde o usuário colocará os itens que deseja, ou seja, é nela que o usuário visualizará o nível que está sendo construído. Ela também implementa a interface `MouseListener` para responder a eventos do *mouse*.

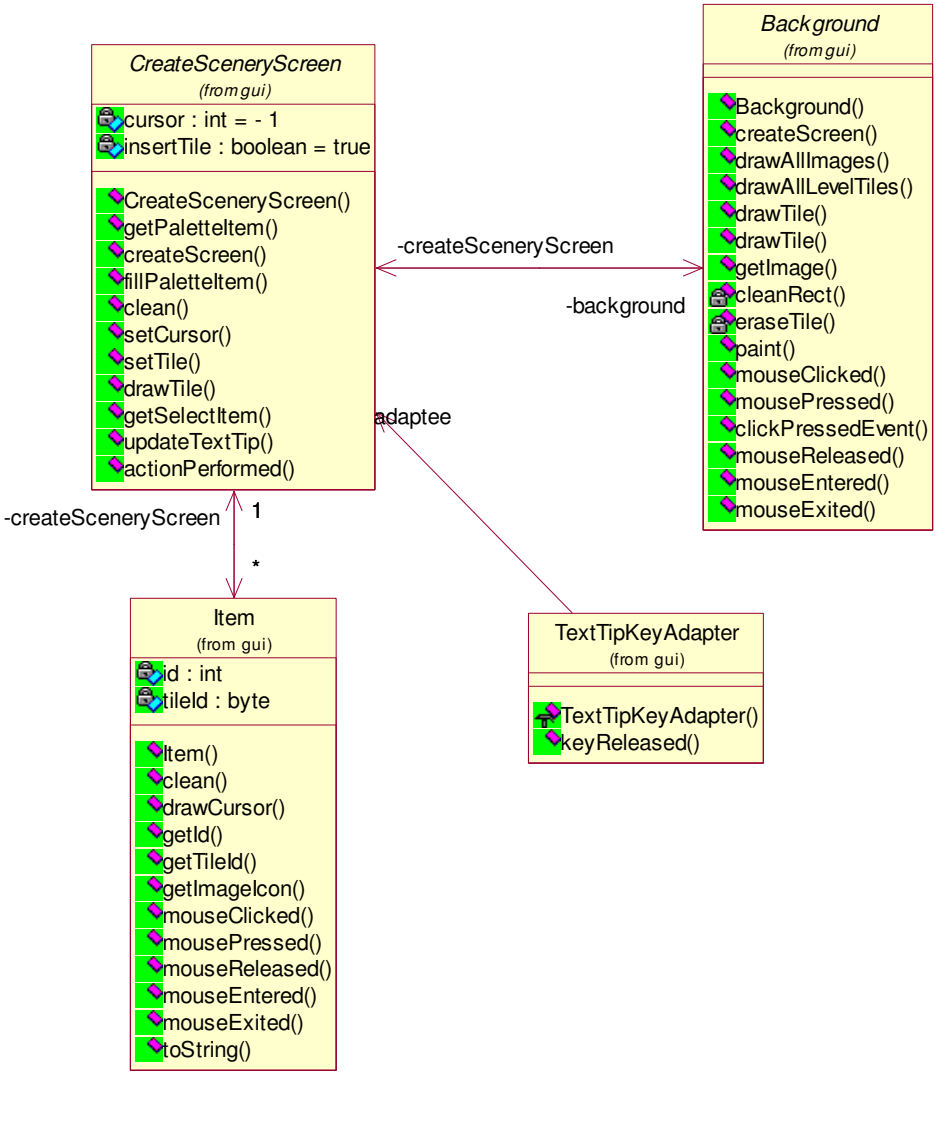


Figura 58 - GUI 2

Tendo-se um conhecimento mais profundo sobre estas classes, pode-se entender melhor a classe `CreateSceneryScreen`. Esta possui uma paleta de itens, um plano de

fundo e uma área de texto, onde o usuário poderá armazenar qualquer dica ou mensagem para o nível construído. Para "ouvir" os eventos provenientes da área de texto citada, existe uma classe interna chamada de `TextTipKeyAdapter`, cuja sua funcionalidade é a de sempre que uma tecla for digitada nesta área, informar a classe pai para atualizar a mensagem em memória. Os principais métodos públicos da classe `CreateSceneryScreen` são:

- `createScreen()`: Cria a aparência da tela com os seus componentes.
- **`abstract`** `fillPaletteItem()`: Método abstrato que deverá ser re-escrito pelo desenvolvedor, cuja sua funcionalidade é preencher a paleta de itens com os ladrilhos possíveis de serem inseridos no jogo. A paleta de itens é representada pelo atributo `paletteItem` do tipo `List`.
- `clean()`: Limpa os componentes desta tela.
- `setCursor(Item item)`: Para indicar ao usuário qual item está ativo, é utilizado um cursor, este método apaga o cursor da posição antiga e o redesenha no novo item escolhido.
- **`abstract`** `setTile(int id)`: Método abstrato que também deverá ser re-implementado pelo desenvolvedor, sua funcionalidade deve ser de inserir o novo ladrilho na parte lógica do sistema, ou seja na estrutura que realmente representará o nível.
- `drawTile()`: Apenas chama o método `drawTile ()` da classe `Background`, que por sua vez desenhará um item na posição indicada.
- `getSelectItem()`: Retorna o item selecionado da paleta de itens.

Os principais métodos públicos da classe Item são:

- `clean()`: Apaga o cursor caso o mesmo esteja no objeto correspondente.
- `drawCursor()`: Desenha o cursor no objeto corresponde.
- `mouseClicked(MouseEvent e)`: Quando o mouse é clicado sobre o objeto correspondente, o mesmo chama o método `setCursor(Item item)` passando-se como parâmetro.

Os principais métodos da classe Background são:

- `createScreen()`: Desenha o fundo de tela no componente e caso trate-se de um nível que já possua um cenário feito, chamará os métodos responsáveis por desenhar cada ladrilho em sua posição.
- **`abstract`** `void drawAllLevelTiles()`: Método abstrato que deve desenhar todos os ladrilhos em um nível.
- `drawTile()`: Recupera o identificador do ladrilho do item selecionado e o repassa para o método abaixo.
- `drawTile(byte tile)`: Desenha o item na posição desejada.
- `abstract Image getImage(byte tile)`: Método abstrato que deve retornar a imagem do ladrilho a ser inserida no cenário.
- `cleanRect(Graphics g, int realX, int realY)`: Este método recorta uma parte do cenário e redesenha o plano fundo que existia no determinado local. Esta funcionalidade serve para apagar um item ou conjunto deles.
- `eraseTile()`: Apaga um item do cenário.
- `paint(Graphics g)`: Pinta todo o cenário.
- **`abstract`** `mouseClicked(MouseEvent e)`: Método abstrato que deve responder ao evento de clique no cenário.

Outra classe provida pelo pacote GUI é `EditorMenuBar`, que representa a barra de menu do sistema. O GEF trabalha com dois estados para a barra de menu, no primeiro estado a campanha está aberta e sendo editada pelo o usuário e no segundo estado a mesma encontra-se fechada. De acordo com estes estados, alguns menus e sub-menus serão habilitados ou não. Para facilitar o seu o uso, a classe possui um `HashMap` para cada um dos estados, ou seja, cada `HashMap` é preenchido com o nome do menu e um outro `HashMap` é preenchido com o nome do sub-menu e um booleano indicando se o sub-menu deverá estar habilitado ou não, de acordo com o estado que o `Hash` representa.

Alguns dos métodos públicos desta classe estão descritos abaixo:

- `createMenus()`: Cria todos o menus e sub-menus providos pelo GEF.

- `addMenu(String label, char mnemonic)`: Adiciona um menu a barra de menus.
- `addMenuItem(JMenu jMenu, String label, int mnemonic, boolean showMenuCampaignClosed, boolean showMenuCampaignOpened)`: Adiciona um sub-menu a um menu adicionado anteriormente.
- `enableMenu(int state)`: Habilita um dos estados da barra de menu provido pelo *framework*.
- `enableMenu(HashMap table)`: Caso o desenvolvedor prefira preencher seu próprio `Hash` com os menus que devem ser habilitados ou não, o GEF provê este método para varrer os menus e sub-menus fazendo as devidas modificações. Segue um exemplo de como este `Hash` deve ser preenchido:

Ex:

```
["File",["New Campaign",new Boolean(true)]]
["File",["Close Campaign",new Boolean(false)]]
......
["Help",["Help",new Boolean(true)]]
["Help",["About",new Boolean(about)]]
```

A última classe provida pelo pacote é a classe `EditorToolBar` que representa a barra de ferramentas do sistema e possui um comportamento bem parecido com a da classe `EditorMenuBar`. A classe também possui dois `HashMap` representando os estados de campanha aberta ou fechada, contudo os mesmos são mais simples pela não existência de sub-menus.

Alguns dos métodos desta classe são:

- `createButtons()`: Cria todos os botões providos pelo GEF.
- `addButton(Image image, String toolTipText, boolean showButtonCampaignClosed, boolean showButtonCampaignOpened)`: Adiciona um botão a barra de ferramenta.
- `enableButton(int state)`: Habilita ou desabilita os botões de acordo com o estado passado.

- `enableButton(HashMap table)`: Tal como a classe `EditorMenuBar`, esta classe também possui um método que permite ao desenvolvedor criar seu próprio Hash informando o que deverá ser habilitado ou não. A figura abaixo (Figura 59) representa as últimas classes do pacote:

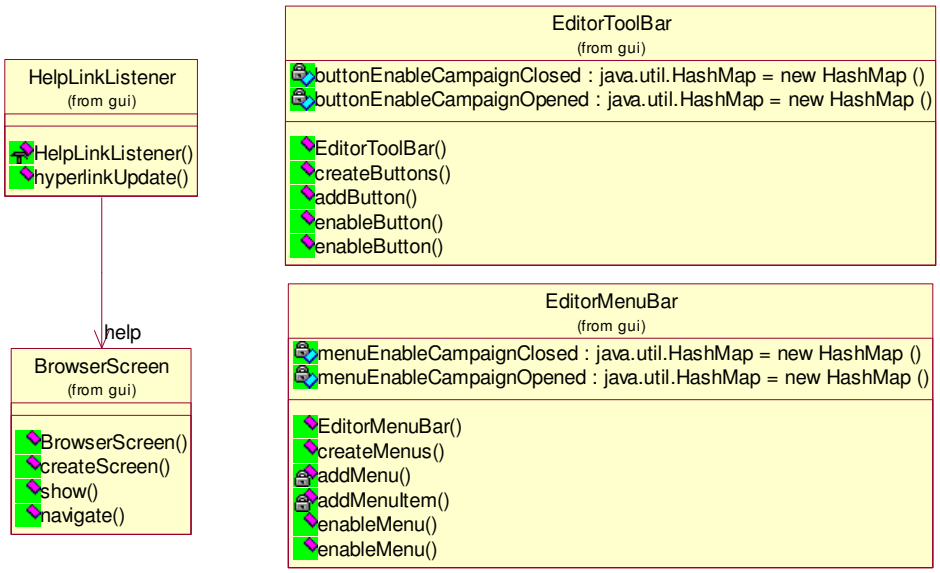


Figura 59 - GUI 3

6.5 O Processo de Abstração do GEF

O processo de construir três casos de uso, para só então abstrair o GEF, mostrou-se bastante sensato e útil. No entanto, foi necessário fazer uso do bom senso para abstrair o GEF da melhor forma possível. A tabela abaixo (Tabela 8) tenta esclarecer em mais detalhes o que ocorreu com as classes originais do primeiro caso de estudo, até chegar ao *framework* desejado.

Compo-nentes	SpaceRunnerEditor	PacmanEditor	PodRaceEditor	GEF
util	EmulatorNotFoundException	(+) Images, XMLConstants	(+) EditorProperties, Point	(#) Alguns nomes de classes modificados
	SpaceException			
	Constants			
	SpaceDynamicProperties			
languages	-	-	(+) Messages	(#) Mantido
network	DocumentHTML	(+) PacNetworkConstants, NetworkException (-) Test	(-) PacNetworkConstants, NetworkException	(#) Alguns nomes de classes modificados (+) EditorContentHandler, EditorContentHandlerFactory
	Answer			
	Test			
logic	Campaign	(+) LevelEditionException, PersistenceException (-) PositionMap	(-) LevelEditionException, PersistenceException	(#) Mantido com as classes Level e Facade passando a serem abstratas
	Level			
	PositionMap			
	Facade			
Presitence	Persistence	-	-	(#) Mantido com a classe passando a ser abstrata
gui	Background	(+) LevelMenuBar, LevelToolBar, AboutSoftwareScreen, DefaultInternalDialog, DefaultInternalFrameListener (-) ExampleFileFilter	(-) AboutSoftwareScreen, DefaultInternalDialog, DefaultInternalFrameListener (+) ExampleFileFilter	(#) Alguns nomes de classes modificados (-) ExampleFileFilter, Run, ConfirmationScreen
	ConfirmationScreen			
	Item			
	CreateSceneryScreen			
	TextTipKeyAdapter			
	NewLevelScreen			
	PresentationScreen			
	NewCampaignScreen			
	BrowserScreen			
	HelpLinkListener			
	DefaultScreen			
	SendCampaignScreen			
	CreateUserServerScreen			
	Run			
	MainScreen			
	ExampleFileFilter			

(+) - Nova(s) classe(s)

(-) - Classe(s) removida(s)

(#) – Comentário

Tabela 8 - Evolução das classes até o GEF

Basicamente o que foi feito após a construção do primeiro editor, foi "copiá-lo" e adaptá-lo para o segundo. Fato que se repetiu para o terceiro editor. O GEF, então, nasceu de uma "cópia" adaptada do terceiro editor, removendo as partes que eram específicas ao mesmo e modificando alguns métodos para serem utilizados como *framework*. Com isto fez-se ver a necessidade de validar o *framework,* reescrevendo dois dos editores construído.

6.6 Validação do GEF

No intuito de validar o GEF e remover erros existentes, o primeiro e o terceiro editor do estudo de casos foram re-escritos utilizando o *framework* elaborado. Para reescrever o editor do jogo Space Runner foi necessário escrever 9 classes sendo 7 delas classes filhas de outras que pertencem ao GEF. Essas são:

- SpaceImages: herda da classe EditorImages e serve para armazenar as imagens que são específicas do jogo.
- SpacePersistence: herda da classe Persistence e implementa a funcionalidade de ler parte do arquivo onde o nível é representado e preencher objetos do tipo Level.
- SpaceLevel: herda da classe Level. Apesar da classe Level não ser abstrata e não necessitar de reimplementação, uma classe filha foi construída para ajudar o editor a garantir que todo nível criado possua os elementos básicos para o funcionamento do jogo em questão, como um portal e um herói.
- SpaceFacade: herda da classe Facade e sua principal função é validar um nível criado.
- SpaceBackground: herda da classe Background e é responsável por funções essências do editor, como por exemplo, desenhar e apagar ladrilhos em um nível.
- SpaceCreateSceneryScreen: herda da classe CreateSceneryScreen e tem a função de preencher a paleta de componentes de acordo com o jogo.
- SpaceManagerScreen: herda da classe ManagerScreen e tem com função implementar um método que instancie uma classe CreateSceneryScreen, no caso uma SpaceCreateSceneryScreen.

As outras duas classes do editor são para organização de constantes e o armazenamento do método principal do sistema, ou seja, o responsável pelo início da execução do mesmo. A figura abaixo ilustra melhor as heranças que foram realizadas (Figura 60).

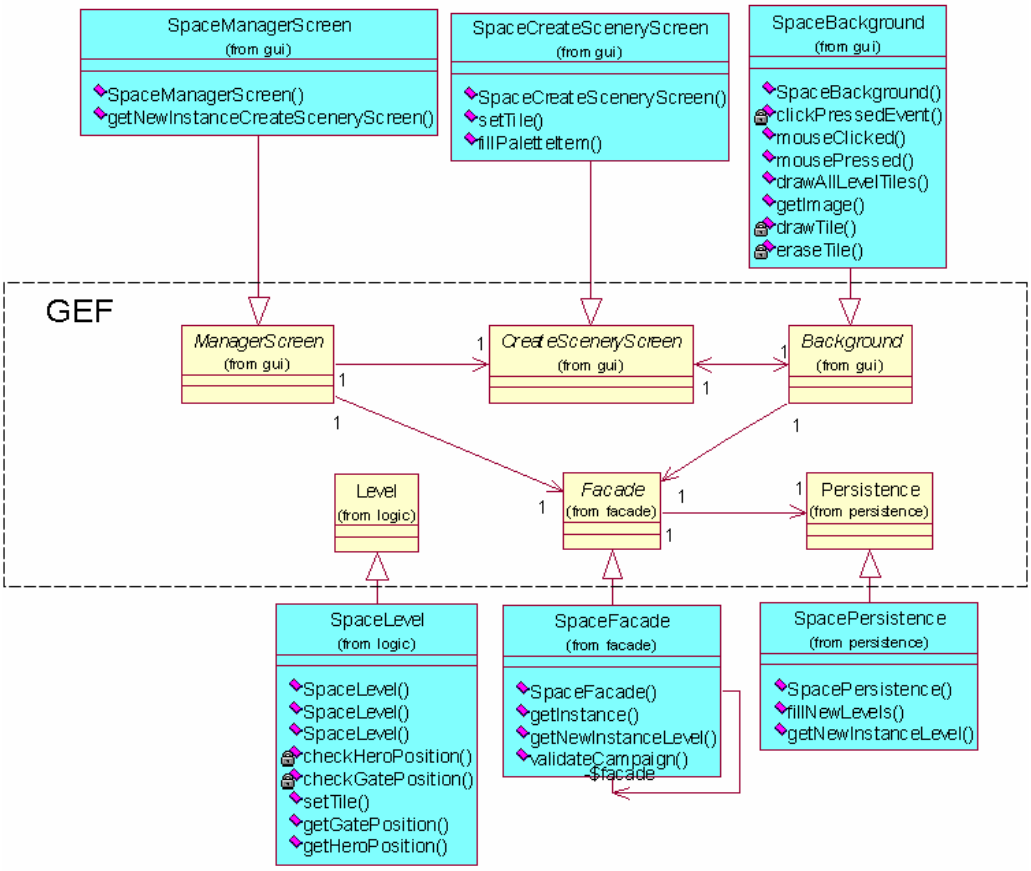


Figura 60 - Diagrama das principais classes Space Runner Editor com GEF

O segundo editor reconstruído utilizando o GEF foi o PodRaceEditor. Para sua reconstrução também foram necessárias 9 classes com nomes e funcionalidades semelhantes ao do SpaceRunnerEditor.

Com o intuito de possuir alguma estatística, sobre quão útil pode ter sido utilizar o *framework*, foi medido o número de pacotes, classes, métodos e linhas de códigos que constituem cada caso de estudos e o GEF. As estatísticas encontradas podem ser visualizadas na tabela abaixo (Tabela 9).

	Pacotes	Classes	Métodos	Linhas de Código (sem considerar comentário)
SpaceRunnerEditor	6	32	266	2594
PacmanEditor	8	39	312	3420
PodRaceEditor	7	29	187	2139
GEF	7	34	281	2444
SpaceRunnerEditor fazendo uso do GEF	6	9	33	320
PodRaceEditor fazendo uso do GEF	6	9	40	597

Tabela 9 - Estatística dos Editores

Pode-se observar que ao utilizar o *framework* para construir o editor, o esforço de implementação é reduzido de forma significativa, a redução em número de linhas de código para o SpaceRunnerEditor foi de 87,66% e para o PodRaceEditor de 72,09%. A diferença entre a redução no número de linhas de código entre os dois editores é justificada pelo fato do PodRaceEditor possuir mais regras específicas. Ainda assim, os números mostram que ambos os editores reutilizam boa parte do código, cabendo ao desenvolvedor escrever apenas o que é específico para o editor.

6.7 Análise Crítica

Após o desenvolvimento dos três editores principais dos casos de estudo, ficou claro que a maior parte do código era reaproveitada do editor anterior e modificado em locais pontuais. Além disto, outras classes eram adicionadas ou re-organizadas no intuito de melhorar a qualidade do código fonte.

A experiência também gerou uma percepção real do que é desenvolver este tipo de aplicação. Vários dos requisitos listados na seção 4.7 e outros requisitos específicos para cada editor foram implementados. Abaixo segue uma tabela para facilitar a visualização dos requisitos desenvolvidos (Tabela 10).

	Editor no dispositivo móvel	SpaceRunner Editor	Pacman Editor	PodRace Editor	GEF	SpaceRunner Editor (fazendo uso do GEF)	PodRace Editor (fazendo uso do GEF)
[RF001]		X	X	X	X	X	X
[RF002]		X	X	X	X	X	X
[RF003]		X	X	X	X	X	X
[RF004]		X	X	X	X	X	X
[RF005]		X	X	X	X	X	X
[RF006]		X	X	X	X	X	X
[RF007]		X	X	X	X	X	X
[RF008]		X	X	X	X	X	X
[RF009]		X			X	X	
[RF010]		X	X	X	X	X	X
[RF011]							
[RF012]							
[RF013]							
[RF014]							
[RF015]							
[RF016]							
[RF017]							
[RF018]		X	X		X	X	
[RF019]							
[RF020]		X	X	X	X	X	X
[RF021]							
[RF022]							
[EM_RF001]	X	X	X	X	X	X	X
[EM_RF002]	X	X				X	
[EM_RF003]	X	X				X	
[EM_RF004]	X	X				X	
[SR_RF001]		X	X	X	X	X	X
[SR_RF002]	X	X	X	X	X	X	X
[SR_RF003]		X	X	X	X	X	X
[SR_RF004]		X	X	X	X	X	X
[SR_RF005]		X			X	X	
[SR_RF006]		X			X	X	
[SR_RF007]		X			X	X	
[SR_RF008]		X			X	X	
[SR_RF009]		X	X	X	X	X	X
[SR_RF010]		X	X	X	X	X	X
[SR_RF011]		X		X	X	X	X
[PM_RF001]			X				
[PM_RF002]			X				
[PM_RF003]			X				
[PR_RF001]				X			
[PR_RNF001]				X	X	X	X

Tabela 10 – Casos de Estudo x Requisitos

O uso do GEF na reconstrução de dois dos editores do caso de estudo, serviu para ter-se uma validação, ao menos em parte, de alguns critérios que podem ser utilizados para indicar o sucesso ou insucesso de um *framework.* Os critérios são: implementação de requisitos, corretude, redução do tempo de desenvolvimento e facilidade de uso.

O GEF implementa ou auxilia na implementação da maioria dos requisitos utilizados nos editores dos casos de estudo, demonstrando assim uma vasta cobertura dos requisitos. Os editores reconstruídos utilizam todos os métodos providos pelo *framework*, garantindo assim que estes métodos não possuem erros aparentes.

No entanto, o GEF não se encontra em um estado de maturidade suficiente para que se possa afirmar que o mesmo é fácil de ser entendido ou utilizado por outros desenvolvedores. Para chegar a este nível de maturidade, o GEF precisa ainda ser mais utilizado, criticado e alterado por seus usuários.

7 Conclusão e Trabalhos Futuros

O trabalho aqui exposto fez um esforço na tentativa de elaborar uma ferramenta que ajude na elaboração de Editores de Níveis, visando diminuir o tempo necessário de construção dos mesmos e consequentemente os seus custos de produção. A idéia de construir um *framework* para isto é original, até onde sabemos, e parece ser uma abordagem promissora para conciliar a necessidade de reuso de código, com o alto grau de acoplamento entre o editor e o jogo.

O processo utilizado na elaboração deste trabalho, pode não ter seguido um caminho curto e de rápidos resultados; no entanto, cada etapa realizada foi extremamente útil no processo de aprendizagem e amadurecimento exigido para a obtenção de um *framework* útil.

A etapa do levantamento de requisitos a partir de editores existentes no mercado foi, além de original, de grande valia para formalizarmos uma idéia do tipo que ferramenta que queríamos ajudar a construir. Obviamente, alguns requisitos só foram visualizados no momento em que os editores estavam sendo construídos. Isto se deve ao fato de que editores, são realmente muito acoplados aos jogos e ao fato dos jogos escolhidos terem sido desenvolvidos para dispositivos móveis.

A etapa envolvendo o estudo realizado sobre *frameworks*, ajudou a definir o resto do processo a ser seguido. O fato de construir pelo menos três casos de estudo, para só depois abstrair o que era comum aos casos desenvolvidos, mostrou-se uma idéia bastante coerente.

O desenvolvimento do jogo Space Runner, apesar de parecer um pouco fora do foco deste trabalho, foi bastante útil no intuito de conhecer melhor o que existe por trás do desenvolvimento de jogos eletrônicos e também nos deu uma boa idéia de como um editor é realmente interligado ao seu jogo.

Construir um Editor de Níveis dentro do Space Runner em um dispositivo móvel, teve alguns aprendizados. O primeiro foi a experiência na construção do primeiro editor, o segundo foi a constatação de como é mais confortável para o usuário final, não ter que

buscar este tipo de ferramenta em outro lugar, basta acessar uma opção que já vem acoplada ao seu jogo. Infelizmente, a utilização do editor em um dispositivo móvel, mostrou ser trabalhosa e não confortável para usuários e desenvolvedores.

O primeiro editor construído para computador pessoal, foi o que levou maior tempo na parte de projeto. Apesar do fato de que já tínhamos estudado alguns editores disponíveis no mercado, quando se pensa em construir uma ferramenta que seria utilizada por desenvolvedores e por usuários comuns, um grande desafio é tentar fazer que sua interface seja simples e intuitiva.

Um outro desafio, foi pensar em como permitir que um nível criado, em um editor que é executado em um computador pessoal, fosse "facilmente" testado no jogo em questão. Talvez nenhuma das três soluções abordadas seja tão boa quanto ter o editor embutido no próprio jogo, mas acreditamos que as soluções propostas, ao menos para os desenvolvedores, são altamente viáveis. Infelizmente, a solução encontrada para o usuário final, de enviar a campanha criada para um servidor e depois baixá-la, para o dispositivo móvel, não é tão eficiente e gera um custo de comunicação com a rede.

Os editores providos para os jogos Pacman e Pod Race contribuíram para o amadurecimento no desenvolvimento de Editores de Níveis e para uma melhor organização do código escrito. Além disto, foi possível observar os pontos em que se fazia realmente necessário inserir código específico para o editor em questão.

A partir do momento que os três casos de estudos tinham sido desenvolvidos, o próximo passo foi a elaboração do produto desejado. Apesar de ser aparentemente um passo simples foi necessário utilizarmos o senso crítico do que poderia realmente ser útil ao desenvolvedor ou não.

Por fim, a reimplementação de dois dos editores após a elaboração do *framework*, GEF, foi fundamental para remover erros de execução em alguns métodos e para mostrar quais instanciações de classes deveriam ser feitas pelos desenvolvedores, e não dentro do próprio *framework*. Um bom exemplo é imaginar que se o pacote *GUI* do GEF criasse uma instância da classe `Level` e o desenvolvedor precisasse criar uma classe

filha da mesma classe `Level`, o usuário teria que re-escrever toda a funcionalidade do pacote GUI que utilizasse a mesma.

Neste trabalho, almejamos dois objetivos importantes. O primeiro era o de abordar um assunto tão pouco comentado na literatura, que é o desenvolvimento de Editores de Níveis. O segundo era de prover uma ferramenta que ajudasse os desenvolvedores a construir seus editores. Provavelmente muito sobre Editores de Níveis ainda precisa ser estudado e escrito, mas acreditamos que ao menos, parte do primeiro objetivo foi alcançado, sendo a maior contribuição nesta área, a lista de requisitos levantada durante o trabalho. Quanto ao segundo objetivo, talvez mais audacioso, acreditamos que o GEF é um primeiro passo em direção ao mesmo. O trabalho também reforça a validade do uso de *frameworks* para resolver o problema de reutilização e valida parte da metodologia proposta por Roberts e Johnson para a construção de *frameworks*. Além disto o trabalho também já rendeu seus frutos em termos de publicações [32], [48] e [49].

Sabemos que o GEF não se encontra em um estado completo, ou seja, ainda possui melhorias a serem feitas e requisitos a serem adicionados. Sabemos também que, este ainda não é uma solução definitiva para o problema de reusabilidade de código para Editores de Níveis. Para esta solução ser aprimorada, é preciso que o mesmo seja utilizado em um ambiente real de desenvolvimento, para que seja mais criticado, modificado, amadurecido e evoluído por seus desenvolvedores. No entanto, acreditamos que a utilização do GEF pode acelerar o processo de desenvolvimento de Editores de Níveis.

Um trabalho futuro como complemento do que foi feito, é utilizar o GEF para construir editores para jogos que sejam executados em computador pessoal. Provavelmente a adaptação do *framework* para jogos 2D em qualquer plataforma, não deve ser complicada, pois todos os casos de estudo implementados foram baseados em mundos bi-dimensional. No entanto, modificar o GEF para ser útil na construção de editores $2^{1/2}$D e 3D é algo que certamente precisará de um estudo mais profundo.

8 Referências

[1] Rouse III, Richard. Designing Design Tools. Gamasutra. Disponível em <http://www.gamasutra.com/features/20000323/rouse_01.htm>. Acesso em 14 agosto de 2002.

[2] JEK Software.Laser Tank. Disponível em <http://lasertank.br-homepage.com>. Acesso em 14 de agosto de 2002.

[3] Microprose. Grand Prix 3. Disponível em <http://gp3.alphaf1.com/>. Acesso em 14 de agosto de 2002.

[4] Sierra. Lode Runner. Disponível em <http://www.angelfire.com/games2/loderunner/_html/_general/main.htm>. Acesso em 14 de agosto de 2002.

[5] Microsoft. Age of Empires. Disponível em <http://www.microsoft.com/games/empires/>. Acesso em 14 de agosto 08 de 2002.

[6] Electronic Arts. Fifa. Disponível em <http://www.fifa2002.com> e <http://www.ea.com/easports/platforms/games/fifa2002/home.jsp>. Acesso em 14 de agosto 2002.

[7] Blizzad. Starcraft. Disponível em <http://www.blizzard.com/worlds-starcraft.shtml>. Acesso em 14 de agosto de 2002.

[8] CESAR. StarHero: Jogo desenvolvido em J2ME. Recife, 2001.

[9] Mount, P. Gameplay: The Elements of Interaction, Gamasutra. Disponível em <http://www.gamasutra.com/education/theses/20020403/mount_01.htm>. Acesso em 14 de agosto de 2002.

[10] Ryan. T. Beginning Level Design Part 2: Rules to Design By and Parting Advice. Gamasutra. Disponível em

<http://www.gamasutra.com/features/19990423/level_design_01.htm>. Acesso em 14 de agosto de 2002.

[11] Asia Java Mobile Challenge. Disponível em <http://javachallenge.singtel.com/>. Acesso em 14 de agosto de 2002.

[12] Odyssey. Come-Come II. Disponível em <http://outerspace.terra.com.br/retrospace/materias/comecome/comecome.htm>. Acesso em 14 de agosto de 2002.

[13] Sun. J2ME. Disponível em <http://java.sun.com/j2me/>. Acesso em 10 de agosto de 2002.

[14] Sun. MIDP. Disponível em <http://java.sun.com/products/midp/>. Acesso em 14 de agosto de 2002.

[15] Sun. CLDC. Disponível em <http://java.sun.com/products/cldc/>. Acesso em 05 de setembro de 2002.

[16] Pessoa, Carlos; Ramalho, Geber; Battaiola, André. wGEM: um *Framework* de Desenvolvimento de Jogos para Dispositivos Móveis. Anais do SEMISH 2002.

[17] Tim Ryan.Beginning Level Design, Part 1: Level Design Theory. Disponível em <http://www.gamasutra.com/features/19990416/level_design_01.htm>. Acesso 19 de agosto de 2004.

[18] Tim Ryan. Beginning Level Design, Part 2: Rules to Design By and Parting Advice. Disponível em <http://www.gamasutra.com/features/19990423/level_design_01.htm>. Acesso em 19 de agosto de 2004.

[19] Wabber Filho. Manual do Gold Hunter. Disponível em <http://www.cin.ufpe.br/~wmaf/gef/SPACE_RUNNER_MANUAL.doc>. Acesso em 19 de agosto de 2004.

[20] WJogos 2002. Workshop Brasileiro de Jogos Brasileiro de Jogos e Entretenimento Digital. Disponível em <http://www.icad.puc-rio.br/wjogos/princp.html>. Acesso em 12 de dezembro de 2004.

[21] Nokia. Fabricante de Aparelhos de Celular. Disponível em <http://www.nokia.com>. Acesso em 19 de agosto de 2004.

[22] Sou Java. Sou de Usuários Java. Disponível em <http://www.soujava.com.br>. Acesso em 19 de agosto de 2004.

[23] CESAR. Centro de Estudos e Sistemas Avançados do Recife. Disponível em <http://www.cesar.org.br>. Acesso em 19 de agosto de 2004.

[24] Roberts, Don; Johnson, Ralph. Evolve Frameworks into Domain-Specific Languages. University of Illinois.

[25] Butler, Greg; Object-Oriented Application Frameworks. Concordia University. Montreal.

[26] Hautamäki,.Juha. A Survey of Frameworks. University of Tampere.

[27] Madeira, C.A.G.. Forge V8: Um Framework para desenvolvimento de jogos de computador e aplicações multimídia. 2001. Dissertação de Mestrado. Universidade Federal de Pernambuco. Recife.

[28] Torres, Eduardo. Forge 16V: Um Framework para Desenvolvimento de Jogos Isométricos. 2003. Dissertação de Mestrado. Universidade Federal de Pernambuco. Recife.

[29] Poptop Software's. Tropico. Disponível em <http://www.gamasutra.com/features/20011010/smith_03.htm> e <http://www.poptop.com/Tropico.htm>. Acesso em 19 de agosto de 2004.

[30] CIn-UFPE. Centro de Informática da Universidade Federal de Pernambuco. Disponível em <http://www.cin.ufpe.br>. Acesso em 19 de agosto de 2004.

[31] Sega. Sega Corporation. Disponível em <http://www.sega.com/>. Acesso em 19 de agosto de 2004.

[32] Arruda Filho, Wabber; Rodrigues, Talita; Florêncio Mauro; Ramalho, Geber. Requisitos Funcionais para Editores de Cenário 2D e 2½D. Wjogos 2002.

[33] Apache Software Foundation. Community of Open-Source Software Projects. Disponível em <http://logging.apache.org>. Acesso em 19 de agosto de 2004.

[34] Apache. Log4j. Disponível em <http://logging.apache.org/log4j/docs/>. Acesso em 19 de agosto 2004.

[35] Ryan, Tim. *What Good Level Design Mean for Players*. Gamasutra 1999. Disponível em <http://www.gamasutra.com/features/19990416/level_design_02.htm>. Acesso em 19 de agosto de 2004.

[36] Jornal da Globo. Criação digital. Disponível em <http://jg.globo.com/JGlobo/0,19125,VTJ0-2742-20040913-61397,00.html>. Acesso em 13 de setembro de 2005.

[37] F1GP. Formula 1 Grand Prix. Disponível em <http://www.f1gp.com.br/artman/publish/article_51.shtml>. Acesso em 09 de novembro de 2004.

[38] Sun Microsystem. Java. Disponível em <http://java.sun.com/>. Acesso em 10 de agosto de 2004.

[39] Perazzo, Eric; Ramalho, Geber; Santos, André. Avaliação Experimental de Detecção de Colisão para Jogos J2ME. SBGames 2004

[40] Mappy Editor. Editor de Mapas. Disponível em <http://www.tilemap.co.uk/mappy.php>. Acesso em 09 de novembro de 2004.

[41] Counter Strike. Disponível em <http://www.counter-strike.net/>. Acesso em 09 de novembro de 2004.

[42] Falstein, Noah. The Flow Channel. Revista Game Developer exemplar de maio 2004.

[43] Csikszentmihalyi Mihaly. The Psychology of Optimal Experience.

[44] XML ORG. XML. Extensible Markup Language. Disponível em <http://www.xml.org/>. Acesso em 14 de novembro de 2004.

[45] W3C. XML em 10 pontos. Disponível em <http://paginas.terra.com.br/informatica/mja/W3C/XML-in-10-points.pt-BR.html>. Acesso em 14 de novembro de 2004.

[46] Especificação MIDP 2.0. Disponível em <http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html> Acesso em 14 de novembro de 2004.

[47] HTTP - Hypertext Transfer Protocol. Disponível em <http://www.w3.org/Protocols/>. Acesso em 14 de novembro de 2004.

[48] Arruda Filho, Wabber; Henrique Pedro; Rousy Danielle e Damasceno Alexandre. Implementando Jogos J2ME: Problemas & Soluções. Wjogos 2002.

[49] Arruda Filho, Wabber; César, Alberto. Ramalho, Geber. GEF: Um framework para Editores de Níveis. SB game 2004.

[50] Apache. Disponível em <http://www.apache.org/>. Acesso em 18 de novembro de 2004.

[51] Roberts, Don; Johnson, Ralph. Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks. University of Illinois.

Dissertação de Mestrado apresentada por **Wabber Miranda de Arruda Filho** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"GEF – Game Editor Framework. Um *Framework* para Editores de Níveis para Jogos Móveis"**, orientada pelo **Prof. Geber Lisboa Ramalho** e aprovada pela Banca Examinadora formada pelos professores:

Prof. André Luis de Medeiros Santos
Centro de Informática / UFPE

Prof. Jacques Philippe Sauvé
Departamento de Sistemas e Computação / UFCG

Prof. Geber Lisboa Ramalho
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 1 de março de 2005.

Prof. JAELSON FREIRE BRELAZ DE CASTRO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.