UNIVERSIDADE FEDERAL FLUMINENSE

ALEX FERNANDES DA VEIGA MACHADO

Uma Engine em XNA e PROLOG para Apoio ao Ensino de Programação Declarativa

NITERÓI

2009

UNIVERSIDADE FEDERAL FLUMINENSE

ALEX FERNANDES DA VEIGA MACHADO

Uma Engine em XNA e PROLOG para Apoio ao Ensino de Programação Declarativa

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Computação Visual e Interfaces.

Orientador:

Prof. Esteban Walter Gonzalez Clua, D.Sc.

NITERÓI

2009

Uma Engine em XNA e PROLOG para Apoio
ao Ensino de Programação Declarativa

ALEX FERNANDES DA VEIGA MACHADO

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Computação Visual e Interfaces.

Aprovada por:

_____

Prof. D.Sc. Anselmo Antunes Montenegro, IC-UFF

_____

Prof. D.Sc. Esteban Walter Gonzalez Clua, IC-UFF

_____

Prof. D.Sc. Flavio Soares Correa da Silva, IME-USP

_____

Prof. D.Sc. Leonardo Gresta Paulino Murta, IC-UFF

_____

Prof. D.Sc. Marcelo da Silva Corrêa, IM-UFF

Niterói, 19 de Fevereiro de 2009

Agradecimentos

Primeiramente agradeço a Deus que é a luz de nossos caminhos.

Agradeço minha mãe por todo amor, carinho e apoio incondicional.

Resumo

Este trabalho apresenta uma ferramenta para motivar e auxiliar o ensino de uma linguagem de programação declarativa através do desenvolvimento de jogos. Para tanto, utilizamos a linguagem P# como integração entre as linguagens PROLOG e C# da plataforma .Net, possibilitando assim o uso da biblioteca XNA para a criação de jogos como base para o desenvolvimento de aplicações centradas em programas declarativos. A metodologia pedagógica proposta usada para validar este ambiente é baseada na aprendizagem por resolução de problemas explorados (Sequência Fedathi).

Palavras-chave: PROLOG, XNA, educação em ciência da computação, games.

Abstract

This work presents a novel game oriented tool which can motivate the teaching of declarative languages programming. The P# language is used to integrate the PROLOG and the C# languages, allowing the usage of the XNA library for the creation of games as a base for developing applications focused on declarative languages. The pedagogic methodology used for validating this approach is based on the learning by resolution of explored problems (Fedathi Sequence).

Keywords: PROLOG, XNA, education in computer science, games.

Siglas e Abreviações

XNA	:	XNA is Not an Acronym
PNA	:	PNA is Not an Acronym
RAD	:	Rapid Application Development
C#	:	C-Sharp
P#	:	P-Sharp
.NET	:	Dot Net
API	:	Application Programming Interface
IDE	:	Integrated Development Environment

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Conforme apresentado em [SILVA & MELO 2006] [SILVA & MELO 2007], a programação declarativa e a programação imperativa se fundamentam em modelos de computação que, embora matematicamente equivalentes, enfatizam conceitos distintos. A programação imperativa se baseia no conceito de máquinas de estados, e se presta melhor à resolução de problemas que conceitualmente sejam mais bem caracterizados dessa forma, portanto descrevendo ações (comandos) e estados (variáveis) em seu escopo. Muitos problemas difíceis de serem resolvidos imperativamente em linguagens convencionais, tais como VB, C++ e Java podem se tornar simples com o uso do paradigma declarativo. Por exemplo, queremos achar um conjunto de valorações para os termos de uma fórmula lógica de tal maneira que o valor da fórmula seja verdadeiro, esse é um problema particularmente difícil [SILVA & SILVA 2007]. De acordo com [SILVA & SILVA 2007], este tipo de questão pode ser facilmente resolvido através da construção de regras simples em PROLOG. A programação declarativa faz sentido na medida em que traz elementos da lógica matemática para a programação de computadores. Parte do apelo à programação lógica vem do fato de que muitos problemas são expressos de maneira natural como teorias representadas por meio de construções em um sistema lógico. É um estilo mais descritivo de programação, onde o programador precisa saber quais relações existem entre diversas entidades, enquanto na programação imperativa, o

programador precisa dizer exatamente o que o computador precisa fazer, passo a passo [SILVA & SILVA 2007].

Não há muitos esforços registrados na literatura no que se refere a compatibilizar sistemas de programação declarativa com plataformas de desenvolvimento modernas voltadas ao desenvolvimento de jogos, fundamentalmente construídas visando o atendimento de linguagens imperativas. Por exemplo, não existem esforços no sentido de criar uma interface PROLOG para uma ferramenta de desenvolvimento como o XNA, possibilitando a sua integração à plataforma .Net. De fato, a criação de um sistema completo nesta linguagem para a plataforma .Net pode trazer complicações de *design*, pois a programação em lógica não possui uma estrutura para a criação de sistemas complexos de *back-end* compatível com as estruturas existentes e já implementadas no *Visual Studio*. Como conseqüência, o aprendizado de técnicas de programação declarativa pode se tornar mais árido que o de programação imperativa.

Neste trabalho é desenvolvido e documentado um ambiente inédito denominado *PNA Game Engine* (Fig. 1), que utiliza a programação em lógica como entrada para a definição do comportamento inicial de um jogo implementado utilizando a biblioteca XNA. Esta ferramenta tem como finalidade auxiliar o docente no processo de ensino-aprendizagem, motivando o estudo deste paradigma por permitir também a geração de elementos visuais em jogos a partir de programas declarativos simples em PROLOG.

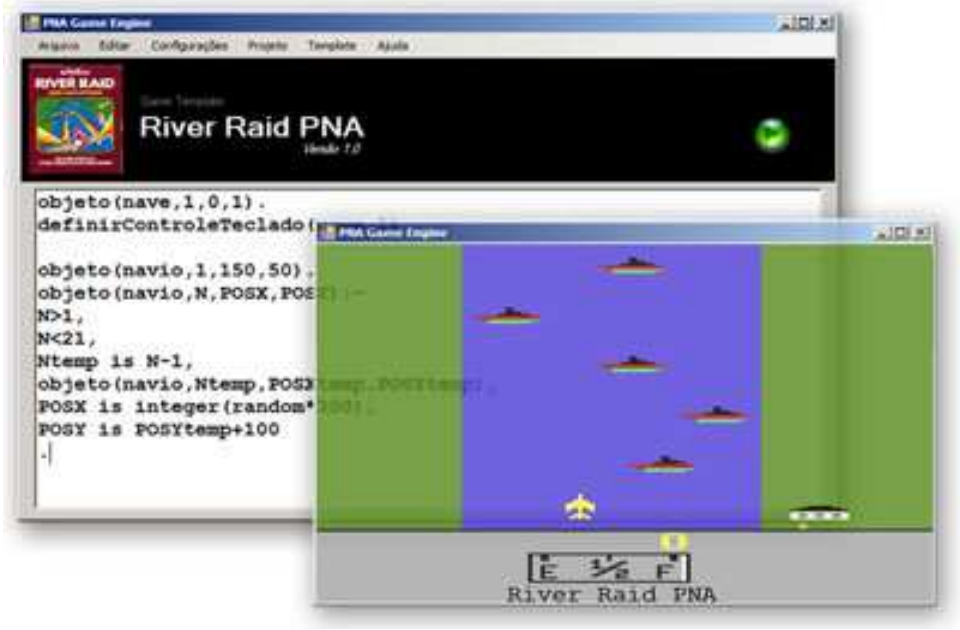

Fig. 1: Interface do PNA Game Engine com a visualização de uma instância do jogo do template River Raid.

Esse sistema também disponibiliza *templates* para jogos (com modelos 2D e 3D prontos, mas com as classes principais sem instanciação, como será explicado no capítulo 4.4) que podem ser acessados através de uma API batizada de PNA (*Pna is Not an Acronym*) pela interface em PROLOG. Portanto, neste contexto um *template* será definido como uma biblioteca de códigos para o controle de um determinado jogo através da API PNA, a ser programada no paradigma declarativo.

Para a adoção desta ferramenta didática em uma aula de Inteligência Artificial é proposta uma adaptação da Sequência Fedathi [NETO & SANTANA 2001] para intermediar no processo ensino/aprendizagem.

O trabalho está organizado da seguinte forma: O Capítulo 2 apresenta uma justificativa para o uso de PROLOG no ensino de computação e as várias maneiras de como esta linguagem pode ser utilizada para o desenvolvimento de jogos. Ainda neste faz-se uma revisão sobre uma ferramenta similar à proposta do presente trabalho. No Capítulo 3 são detalhadas as principais tecnologias presentes no desenvolvimento deste *software*, em especial a plataforma .Net, a integração de XNA com formulários e o compilador P#. No Capítulo 4 demonstra-se a aplicação dessas tecnologias no

processo de desenvolvimento do *PNA Game Engine*, bem como seus principais componentes, funcionalidades e o *template* elaborado. No Capítulo 5 é definido um *workflow* pedagógico para auxiliar o professor de Inteligência Artificial no uso da ferramenta proposta. Nos Capítulos 6 e 7 são apresentadas algumas conclusões e trabalhos futuros.

Capítulo 2

Visão Geral Sobre PROLOG

A linguagem de programação PROLOG possibilita resolver problemas lógicos através da programação em um computador. Ela foi desenvolvida na década de 70, visando especificamente a formalização e resolução de certos problemas de linguística computacional, e seu uso posteriormente se estendeu para outras áreas em que a programação declarativa e a caracterização conceitual de problemas baseada em lógicas formais – clássicas ou não clássicas – se mostrasse conveniente [SILVA & MELO 2006]. Dentre muitos problemas da vida real, que podem ser modelados por meio de linguagens lógicas, destacamos os jogos de computador. As características do PROLOG associadas às características estratégicas de um jogo proporcionam um ambiente adequado para a programação declarativa, especificamente aquela fundamentada em uma modelagem baseada em inferências lógicas.

Os termos "programação em lógica" e "programação Prolog" tendem a ser empregados indistintamente. Deve-se, entretanto, destacar que a linguagem Prolog é apenas uma particular abordagem da programação em lógica. As características mais marcantes dos sistemas de programação em lógica em geral - e da linguagem Prolog em particular - são as seguintes:

- Especificações são Programas: A linguagem de especificação é entendida pela máquina e é, por si só, uma linguagem de programação.
- Capacidade Dedutiva: A execução de um programa é a prova do teorema representado pela consulta formulada, com base nos axiomas representados pelas cláusulas (fatos e regras) do programa.
- Não-determinismo: Os procedimentos podem retornar múltiplas respostas.
- Reversibilidade das Relações: (Ou "computação bidirecional"). Os argumentos de um procedimento podem representar parâmetros de entrada, ou de saída. Os procedimentos podem atender a múltiplos propósitos. A execução pode ocorrer em qualquer sentido, dependendo do contexto.
- Tríplice Interpretação dos Programas em Lógica: Um programa em lógica pode ser semanticamente interpretado de três modos distintos: (1) por meio da semântica declarativa, inerente à lógica, (2) por meio da semântica procedimental, onde as cláusulas dos programas são vistas como entrada para um método de prova e, (3) por meio da semântica operacional, onde as cláusulas são vistas como comandos para um procedimento particular de prova por refutação.
- Recursão: A recursão, em Prolog, é a forma natural de ver e representar dados e programas.

Neste capítulo será justificada a relevância do ensino de PROLOG, assim como sua área de aplicação no que diz respeito ao desenvolvimento de jogos de computador.

2.1 Vantagens do Ensino de PROLOG

Além do fato de a linguagem de programação PROLOG ser adotada por muitos desenvolvedores da área de inteligência artificial, o ensino desta linguagem possui diversas vantagens [PALAZZO 1997], tais como:

- Aprendizado mais fácil e natural em comparação com as linguagens imperativas;
- Permite a implementação de sistemas reflexivos;
- Exercita fundamentos de linguagens de especificação lógica.
- Facilita a implementação de regras de gramáticas livres de contexto.
- Permite o estudo de recursão através de mecanismos simples como *backtracking*, *cut* e *fail*.

2.2 Utilização de PROLOG no Desenvolvimento de Jogos

O desenvolvimento de um *engine* que permita visualizar os resultados de uma rotina declarativa sem a necessidade de implementação das funções imperativas fundamentais presentes em um jogo (como gerenciamento dos dispositivos gráficos, controle do teclado e carga dos arquivos de imagem), pode aumentar a motivação do aluno para o estudo de linguagens e ambientes de programação que permitam obter tais facilidades.

Podem-se destacar três grandes áreas para uso de PROLOG no desenvolvimento de jogos: controle do ambiente e do comportamento dos agentes, sistemas especialistas e criação de diálogo.

2.2.1 Controle do Ambiente e do Comportamento dos Agentes

A programação em lógica é constituída por dois elementos principais: a lógica e o controle [PALAZZO 1997]. O componente lógico corresponde à definição (especificação) do que deve ser solucionado, enquanto que o componente de controle estabelece como a solução pode ser obtida. A estrutura lógica é responsável por gerar a base de conhecimento e a estrutura de controle coordena o entendimento sobre a mesma. É necessário somente descrever o componente lógico de um programa, deixando o controle da execução ser exercido pelo sistema de programação em lógica que se está utilizando. Portanto a tarefa do desenvolvedor passa a ser simplesmente a especificação do problema que deve ser solucionado, razão pela qual as linguagens lógicas podem ser vistas simultaneamente como linguagens para especificação e para a programação de computadores.

Trazendo o conceito de lógica e controle para os jogos pode-se definir, respectivamente, o ambiente e o comportamento dos agentes. Por exemplo, em um jogo do estilo plataforma os obstáculos poderiam ser gerados seguindo um princípio lógico na base de dados de informação de posições e definir a inteligência artificial de ações dos inimigos (agentes inteligentes) usando a estrutura de controle que analisará essa base lógica.

2.2.2 Sistemas especialistas

Um sistema especialista (SE) é uma forma de Sistema Baseado em Conhecimento (SBC) especialmente projetado para emular a especialização humana em algum domínio específico. Tipicamente um SE irá possuir uma base de conhecimento formada de fatos, regras e heurísticas sobre o domínio, juntamente com a capacidade de estabelecer comunicação interativa com seus usuários, de modo muito próximo ao que um especialista humano faria. Portanto, um sistema especialista resolve problemas que normalmente são solucionados por pessoas especializadas. Ele recebe uma entrada do usuário, analisa as possíveis respostas da base de conhecimento e,

dependendo da resposta, pode exigir uma nova entrada do usuário, repetindo este processo até uma solução ser obtida.

Um sistema especialista de xadrez poderia, por exemplo, concluir qual seria a melhor jogada em determinado momento. Para provar tal proposição, seriam apresentados alguns casos passados e seus respectivos resultados. Isto faz com que o usuário conheça a lógica ou a razão para aquela tomada de decisão, a qual ainda pode ser modificada pelo próprio usuário, caso este a julgue incorreta.

O estudo e a criação de sistemas especialistas representam uma das subdivisões mais importantes das linguagens de programação em lógica, bem como dos jogos com inteligência artificial. Pode-se criar, por exemplo, as possíveis ações de um NPC (*non-player character*, ou personagem não-jogador, que representa um agente inteligente autônomo) que pode variar com as ações do *player* (personagem controlado pelo jogador).

2.2.3 Criação de Diálogo

As propriedades do PROLOG vão ao encontro das necessidades que surgem quando da programação de sistemas processadores de linguagens naturais. Suas principais características inerentes a esta aplicação são [COVINGTON 1994]:

- Facilidade de modificação de estruturas de dados grandes e complexas: Esta peculiaridade é de vital importância para o armazenamento de estruturas sintáticas, estruturas semânticas bem como de entradas léxicas, elementos presentes em programas que manipulam qualquer linguagem natural.
- Capacidade de auto-análise e auto-modificação de programas: O suporte à metaprogramação é muito forte em PROLOG, o que possibilita a adoção de modelos abstratos de programação.
- Algoritmo Busca em Profundidade (*depth-first)* embutido: Para a busca das informações concernentes a um programa PROLOG (fatos e regras), o método de busca *depth-first* é internamente utilizado em sistemas PROLOG. Este

algoritmo é de extrema utilidade para a implementação de analisadores sintáticos, sendo adotado pela maciça maioria.

- Incorporação de DCG (*Definite Clause Grammar*): DCG é um formalismo que estende as gramáticas livres de contexto, possibilitando a identificação e utilização de inter-relacionamentos entre os componentes de uma regra gramatical. Além de cobrir a principal carência das gramáticas livres de contexto, as DCGs não oneram o processamento de sentenças de tamanho considerável, como o fazem as gramáticas sensíveis ao contexto. Deste modo, com as DCGs tem-se um formalismo de grande capacidade de expressão, aliada à eficiência. As gramáticas sensíveis ao contexto, quando empregadas no processamento de sentenças, denotam uma complexidade de tempo exponencial, em função do tamanho das entradas.

A maior parte dos jogos eletrônicos, e principalmente os do gênero RPG (*Role Playing Game*), necessitam de diálogo entre os NPCs e o *player* [LEBBINK, WITTEMAN & MEYER 2004]. Em [LEBBINK, WITTEMAN & MEYER 2004] é apresentado um sistema multi-agentes de diálogo para jogos que permite a análise semântica das sentenças através de um motor escrito em PROLOG. Nele a semântica e o raciocínio são definidos através da formulação de regras de utilização classificadas como pré-condições (conjunto de critérios pré-definidos para o agente) e pós-condições (novos critérios atualizados a partir de decisões tomadas sobre novos estados).

A criação de diálogos entre *players* e personagens do mundo virtual representa uma das principais áreas de estudo da aplicação de PROLOG para jogos.

2.3 Ferramentas Gráficas para o Ensino de Programação

O ensino de programação pode ser facilitado por ferramentas que deem um retorno gráfico, como jogos, por exemplo.

Um exemplo é o Alice [CONWAY, 1997]. Desenvolvido na Universidade de Carnegie Mellon, é um programa grátis e aberto implementado em Java e *multi* plataforma, feito com o objetivo de apoiar o inicio da aprendizagem da programação.

Alice estimula o aprendizado à programação, utilizando elementos 3D. Fundamentos da programação orientada a objetos são explorados pelo *software* como propriedades e métodos. Cada objeto é representado por uma figura 3D, como carros, pessoas, animais, entre outros.

Utilizado por mais de 10% das escolas e universidades norte americanas [SILVA & SILVA 2006], Alice possibilita escolher um cenário, adicionar objetos e interagir com eles. Por exemplo, você pode adicionar uma patinadora em um lago de gelo e determinar movimentos para ela, como girar, levantar a perna, mexer o braço, entre outros. Cada objeto tem seus métodos e propriedades próprias, assim como na programação orientada a objetos. Desta forma o programador pode absorver muitos conceitos de programação de uma forma divertida.

Os principais objetivos desta ferramenta, no que tange o ensino de programação para iniciantes, podem ser resumidos em:

- Fazer com que o estado dos objetos sejam tão visíveis ao aluno quanto for possível.
- Animar todas as mudanças de estado dos objetos.
- Não permitir que programas sintaticamente errados sejam escritos.
- Realçar a noção de Objeto (focando na programação orientada a objetos).
- Usar ambientes tridimensionais para motivar os alunos.

Uma das ferramentas mais conhecidas para apoiar o ensino de programação através do desenvolvimento de jogos é o Robocode [HARTNESS 2004]. Ele permite o exercício de conteúdos teóricos de forma prática em aulas de inteligência artificial. O Robocode é uma arena onde vários tanques de guerra batalham entre sí. O tanque que prevalecer até o final será o vencedor. Os alunos são responsáveis por escrever a

lógica do tanque em Java e o Robocode faz a simulação da batalha utilizando esse código.

Em [HARTNESS 2004] demonstra-se que os alunos destas aulas foram capazes de compreender melhor a teoria e adquiriram maior confiança para implementar seus códigos. Outro motivo que tornou o uso do Robocode um sucesso foi a promoção de um ambiente competitivo em sala de aula, onde os alunos disputavam para ver quem conseguia escrever a melhor lógica.

Entre outras ferramentas, em [SILVA & SILVA 2006] [SILVA & MELO 2007] foi apresentado um sistema com um ambiente virtual 3D, animações e interação com múltiplos agentes, para permitir que alunos de graduação exercitem a programação lógica.

Para tanto foram utilizados:

- Uma interface para a interação entre programas em PROLOG e programas em C++;
- O *engine* 3D Ogre para a visualização de ambientes virtuais tridimensionais, controlados por programas em C++;
- O interpretador SWI PROLOG para a construção e execução de programas em PROLOG.

Com esta ferramenta, o aluno é dispensado de escrever toda a camada de apresentação de seu algoritmo, porque ela já se encarrega disso para os cenários apresentados. Assim, o aluno só precisa se concentrar na lógica para resolver o problema.

O trabalho mostra, através de experimentos em sala, que o desenvolvimento de programas em PROLOG com visualização gráfica do comportamento dos mesmos representa uma valiosa ferramenta para docentes que desejam motivar os alunos no aprendizado de inteligência artificial, lógica formal e programação declarativa. Esta ferramenta foi utilizada em 2 cursos do Instituto de Matemática e Estatística da Universidade de São Paulo, IME/USP: Métodos Formais em Programação e Laboratório

de Inteligência Artificial. O objetivo não foi verificar o aprendizado, mas uma fazer uma avaliação empírica da reação dos alunos a essa abordagem.

2.3.1 Análise das Ferramentas Gráficas de Apoio ao Ensino de Programação Declarativa

Para o efetivo desenvolvimento de uma ferramenta que apóie o ensino de programação declarativa através do desenvolvimento de jogos foram observadas as seguintes vantagens das ferramentas citadas no capítulo anterior:

- De acordo com os autores das três ferramentas, a motivação proporcionada pelo uso da ferramenta justificou sua abordagem em sala.
- Todas as ferramentas restringiram os alunos à programação do comportamento dos elementos gráficos fornecidos pelos mesmos, dispensando a programação em mais baixo nível (leitura do teclado, inserção dinâmica das imagens...).
- Todas demonstraram ser possível o desenvolvimento de uma engine/ferramenta atavés de softwares gratuitos.
- Enquanto o Robocode e o Alice enfatizaram o paradigma Orientado a Objetos, a ferramenta Odin demonstrou que é possível desenvolver um ambiente agradável para o ensino de programação declarativa.

Capítulo 3

Tecnologias para a Criação da Ferramenta Proposta

Toda tecnologia utilizada na ferramenta proposta é gratuita, de forma a facilitar seu uso em fins acadêmicos. Além de programas e linguagens da plataforma .Net, incluem-se rotinas para integração do XNA em formulários, DLL do compilador P# e a própria linguagem PROLOG. O código completo do sistema desenvolvido pode ser conferido nos apêndices A.1 e A.2.

3.1 Tecnologias Microsoft

Optou-se pela plataforma *Microsoft Visual Studio .NET* como principal ferramenta de desenvolvimento, por ser uma ferramenta RAD e permitir a interoperabilidade entre múltiplas linguagens [LIBERTY 2001].

Dentre as principais linguagens suportadas por esta plataforma, escolheu-se C# por ser uma linguagem orientada a objetos similar ao JAVA e por ser a única a trabalhar com XNA.

O C# é, de certa forma, a linguagem de programação que mais diretamente reflete a plataforma .NET sobre a qual todos os programas .NET executam. C# está de tal forma ligada a esta plataforma que não existe o conceito de código não-gerenciado (*unmanaged code*). Suas estruturas de dados primitivas são objetos que correspondem

a tipos em .NET. A desalocação automática de memória por *garbage colletor* além de várias de suas abstrações tais como classes, interfaces, delegados e exceções são nada mais que a exposição explicita recursos do ambiente .NET.

Quando comparada com C e C++, a linguagem é restrita e melhorada de várias formas incluindo:

- Ponteiros e aritmética sem checagem só podem ser utilizados em uma modalidade especial chamada modo inseguro (*unsafe mode*). Normalmente os acessos a objetos são realizados através de referências seguras, as quais não podem ser invalidadas e normalmente as operações aritméticas são checadas contra sobrecarga (*overflow*).
- Objetos não são liberados explicitamente, mas através de um processo de coleta de lixo (*garbage collector*) quando não há referências aos mesmos, prevenindo assim referências inválidas.
- Destrutores não existem. O equivalente mais próximo é a interface *Disposable*, que juntamente com a construção *using block* permitem que recursos alocados por um objeto sejam liberados prontamente. Também existem finalizadores, mas como em Java sua execução não é imediata.
- Como no Java, não é permitida herança múltipla, mas uma classe pode implementar várias interfaces abstratas. O objetivo principal é simplificar a implementação do ambiente de execução.
- C# é mais seguro no tratamento de tipos do que C++. As únicas conversões implícitas por *default* são seguras, tais como ampliação de inteiros e conversões de um tipo derivado para um tipo base. Não existem conversões implícitas entre inteiros e variáveis lógicas ou enumerações. Não existem ponteiros nulos (*void pointers*) (apesar de referências para *Object* serem parecidas). E qualquer conversão implícita definida pelo usuário deve ser marcada explicitamente, diferentemente dos construtores de cópia de C++.
- Algumas sintaxes são diferentes, como para a declaração de vetores ("*int[] a = new int[5]*" ao invés de "*int a[5]*").

- Membros de enumeração são colocados em seu próprio espaço de nomes (*namespace*).
- C# não possui modelos (*templates*), mas C# 2.0 possui genéricos (*generics*).
- Propriedades estão disponíveis, as quais permitem que métodos sejam chamados com a mesma sintaxe de acesso a membros de dados.
- Recursos de reflexão completos estão disponíveis.

O XNA *Game Studio Express* é uma API da plataforma .Net que permite fácil acesso aos periféricos (como o teclado), ao hardware gráfico, controle de áudio e armazenamento de informações (em arquivos ou banco de dados) [CREATORS CLUB 2008]. Essa API pode também ser usada como base para o desenvolvimento de jogos para o console XBox 360.

O XNA, por ser uma plataforma de desenvolvimento, é formado por alguns componentes descritos a seguir:

- XNA *Game Studio*: IDE de desenvolvimento, baseado no Visual C# Studio. Sua versão gratuita (*Express Edition*) foi desenvolvido para ser usada por estudantes e pequenos grupos de desenvolvimento (*indie developers*), com o intuito de permitir que os usuários desenvolvessem seus próprios jogos;
- XNA *Framework*: conjunto de classes necessárias para a execução de um jogo XNA. Funciona sobre o .NET *Framework* (*games* para PCs) ou sobre o .NET *Compact Framework* for *Xbox* 360 (*games* para *Xbox* 360);
- XNA *Content Pipeline*: componente de gerência de conteúdo e artefatos do projeto, tais como imagens (JPG, PNG, BMP, etc), modelos 3D (X, FBX, etc), sonoros (WAV, etc) e dados (XML, etc). Ele transforma os arquivos automaticamente no momento do *build* em um formato que será entendido pela aplicação em tempo de execução. O *Content Pipeline* é extensível e permite que o desenvolvedor escreva um conversor para um formato especial ou desconhecido com flexibilidade;
- XACT (*Audio Authoring Tool*): ferramenta para áudio *designers* organizarem seus arquivos de áudio e efeitos sonoros.

Outra ferramenta importante do *framework* da Microsoft é o CSC (*C-Sharp Compiler*), aplicativo *stand alone* que é chamado para compilar classes .cs em arquivos executáveis (.exe) ou DLL's através de uma linha de comando.

3.2 XNA em Formulários

Estes formulários permitem, de forma fácil e rápida, a criação de menus de configuração e de uma interface amigável para o aluno. Entretanto, a união dessas duas tecnologias não é um procedimento nativo da plataforma. A dificuldade reside no fato do gerenciador de dispositivos gráficos do XNA criar sua própria janela e não fornecer integração com a janela normal dos formulários do *Visual Studio*. Isto exige que o desenvolvedor crie seu próprio código de gerência de dispositivo gráfico.

Existem dois caminhos para se criar um projeto do *Visual Studio* que use formulários da plataforma. Net e o *XNA framework*:

- Criar um novo projeto de aplicação do *windows* com formulário e referenciar as DLL's do XNA a partir dele. Esta é a forma mais complexa, porque o .csproj (arquivo com informações do projeto) terá que ser modificado manualmente para ter o acesso ao *Content* Pipeline; ou,
- Criar um novo projeto de jogo do *windows* e referenciar uma diretiva para uso de formulários. Opção adotada neste trabalho.

As principais etapas para a integração do formulário do *Windows Game Project* no *Visual Studio* podem ser conferidas no diagrama da Fig. 2 e suas descrições são detalhadas em seguida.

Fig. 2: Etapas para a intetração do XNA com o Visual Studio

1. Preparação do ambiente - Iniciar um novo *Windows Game*, remover a classe pré-criada *Game1.cs* e adicionar um *Windows Form*. A partir de agora, o corpo do game ficará neste formulário, e não mais no Game1.cs. Esta adição será a responsável por inserir toda referência de suporte para a execução de um formulário. É necessário também modificar a classe *Main* do arquivo Program.cs para usar a diretiva *System.Windows.Forms* e permitir a geração do formulário criado.
2. Criação de um componente para exibição do conteúdo do XNA - deve-se adicionar um *User Control* no projeto (este ficará disponível na *Toolbox*). O componente *User Control* permite agrupar um conjunto de códigos e/ou outros componentes para ficarem disponíveis para o uso em um ou mais formulários. Portanto ele funciona como um recurso de modularização para diminuir o trabalho de codificação e manutenção do sistema.
3. Implementação do *User Control* - este componente deverá atualizar a cada momento o gráfico contido nele. Deve-se criar um procedimento para

substituir o método *OnPaint* herdado pelo *System.Windows.Forms.UserControl*. O código de sua implementação pode ser conferido no apêndice A.1.

4. Criação de um dispositivo gráfico – Foi desenvolvido um código, no corpo da *Main Class* do formulário para gerenciar o dispositivo gráfico. Ele possui duas funções principais, o *Draw()* para renderizar o conteúdo e o *Blit()* para aumentar a eficiência, pois este limitará o código de visualização para ser executado somente quando exigido uma vez que armazena todo resultado de cada renderização em uma textura para evitar que a próxima renderização do frame comece do zero.
5. Usando o *render target* no *looping* do jogo – *Render target* é uma área linear da memória de exibição que precisa ser recriada sempre que o dispositivo é chamado. Ela contém a informação da cor de uma imagem renderizada. Para esta etapa é criada a método *RenderToTexture()* para configurar o dispositivo gráfico para usar o *render target* e o *depth buffer*, para esvaziar os *buffers*, desenhar a tela e pegar o resultado em forma de textura.
6. Criação do método *Update()* – criação do método principal do controle do comportamento do jogo em uma linha do tempo. Para tanto é criado um conjunto de funções para usar do *game looping* do XNA no evento *OnIdle()* do formulário. Neste procedimento de repetição é incorporado as funções de *Update()* e *Draw()*.

Portanto estas etapas têm por objetivo criar um componente capaz de simular o ambiente de desenvolvimento do XNA (por implementar os métodos *Draw()* e *Update()* entre outros) integrado ao código do formulário do *Visual Studio*.

3.3 O Compilador P#

O compilador P# [COOK 2003] foi desenvolvido a partir do projeto do PROLOG Café [BANBARA & TAMURA 1999] para produzir C# ao invés de Java a partir de um *script* PROLOG. Ele pode ser usado de forma *stand alone* para testar, executar ou gerar aplicações a partir de códigos PROLOG, ou como um módulo (uma única DLL) para

compilar arquivos PROLOG em classes C#. Quando um código PROLOG é compilado no P#, ele gera uma classe para cada predicado definido. A convenção usada no nome do arquivo gerado é:

NomeDoPredicado_NumeroDeArgumentos.cs
(Ex.: Pai_2.cs)

Quando o *assembly* do módulo P# é usado em uma aplicação C# ela dispõe de diversas classes para o desenvolvedor se comunicar com os predicados e configurações do lado ambiente do PROLOG (classes geradas a partir do código PROLOG). Um exemplo de *script* é:

1. *VariableTerm pai = new VariableTerm( );*
2. *PROLOGInterface sharp = new PROLOGInterface( );*
3. *sharp.SetPredicate(new Pai_2(pai,SymbolTerm.MakeSymbol("Zé"),new ReturnCs(sharp)));*
4. *sharp.Call();*
5. *Console.WriteLine("O pai é: {0}", pai.Dereference());*

Na linha 1 é instanciado um termo PROLOG (variável PROLOG) com o nome de pai; na linha 2 cria-se a interface para comunicação com o código PROLOG, *sharp*; na linha 3 realiza-se uma consulta através do método *SetPredicate* em uma classe denominada *Pai_2* (que é um arquivo .cs gerado a partir de um *script* PROLOG convencional) armazenando no termo pai todos os resultados cujo segundo termo é Zé; na linha 4 chama-se o primeiro resultado; e na ultima escreve-se esse resultado na tela.

Capítulo 4

Desenvolvimento do Sistema

Nenhum *engine* de desenvolvimento de jogos atual (como o XNA ou o *3D Game Studio*) utiliza qualquer linguagem de programação em lógica em seu módulo de *script*. Mas devido às suas inúmeras aplicações na área de Inteligência Artificial [CASANOVA 2006, GIORNO E FURTADO 2006], muito estudo existe neste sentido [SILVA e SILVA 2006]. Neste trabalho é desenvolvido o *PNA Game Engine*, uma ferramenta que pretende preencher esta lacuna (Fig. 3).

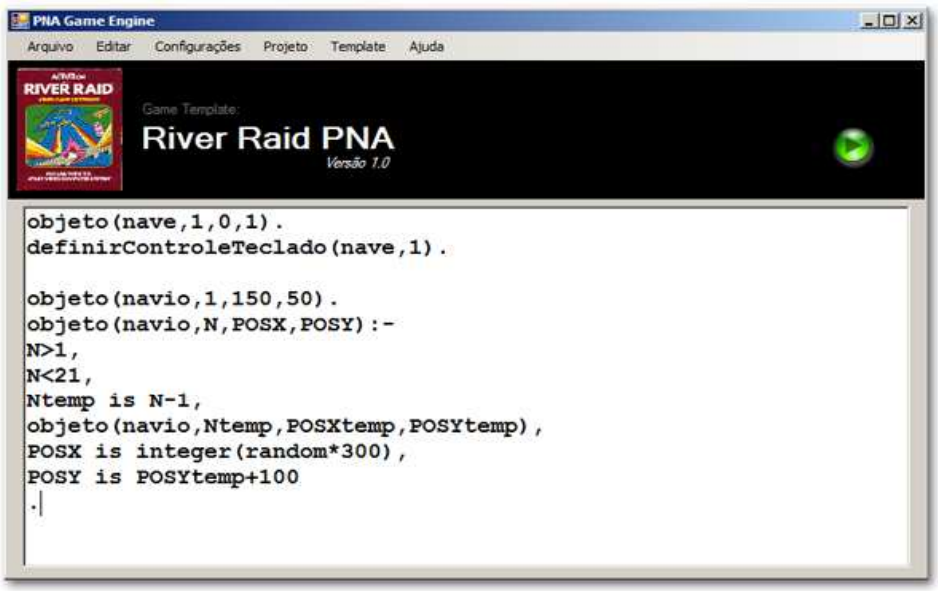

Fig. 3: Interface do PNA Game Engine

Com base no compilador P# e na biblioteca de códigos gráficos para desenvolvimento de jogos da Plataforma .Net, a XNA, são criados jogos "vazios" ou *game templates* (capítulo 4.4) compostos de bibliotecas de classes, modelos 3D de objetos, modelos 2D de *sprites* e elementos sonoros (os principais componentes deste sistema podem ser conferidos no capítulo 4.2).

O *template* completo desenvolvido nesta primeira versão do programa foi o clássico *River Raid* (Fig. 4), que é um jogo do console Atari, no qual um avião sobrevoando um rio deve destruir e ultrapassar os barcos e helicópteros que são seus obstáculos.

Fig. 4: Jogo River Raid

Nesta primeira versão, o programador está limitado a usar somente os *game templates* pré-definidos, excluindo a possibilidade de incorporação de novos.

4.1 O Engine PNA

O *engine PNA* é o *software* que vai integrar o *template* do jogo com o restante do sistema. Seus principais requisitos podem ser demonstrados no seguinte diagrama (Fig. 5):

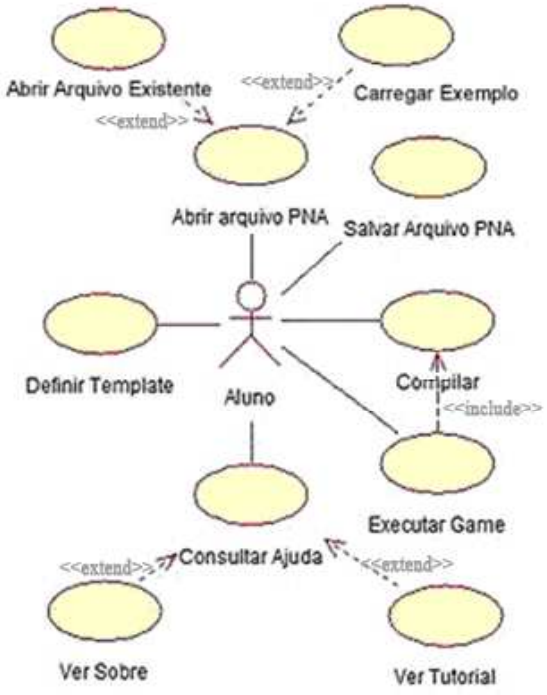

Fig. 5: Diagrama UML do PNA Game Engine

- Abrir arquivo PNA – permite abrir um novo código em sua área de edição.
- Definir *template* - permite trocar o *template* atual. Esta tarefa carrega as bibliotecas de código e altera todas as opções de configuração e compilação para este determinado modelo. Sempre que o programa for aberto, será exibida uma tela para a definição do *template* inicial. Durante o desenvolvimento de um programa, se o modelo for alterado, a área de edição será reiniciada (para não criar uma confusão de predicados, pois eles são diferentes para cada *template*) e será dado início a um novo arquivo PNA.
- Carregar Exemplo – permite abrir arquivos PNAs pré-definidos (ver tabelas Tab. 1 e Tab. 2). Esses arquivos têm a extensão .pl, porque embora possuam predicados pré-estipulados para o *game template* definido ainda assim não deixam de representar *scripts* do PROLOG.
- Abrir Arquivo existente - permite abrir um arquivo PNA desenvolvido pelo usuário.
- Salvar Arquivo PNA – salva o *script* desenvolvido pelo usuário. Somente depois de salvo é que esse *script* pode ser compilado.

- Compilar – realiza as etapas: verifica o ultimo código PNA salvo e exibe mensagem de erro em console caso exista (P#); traduz o código para arquivos .cs, sendo uma classe para cada predicado (P#); e, transforma cada arquivo gerado em DLL (CSC).
- Executar jogo – carrega os parâmetros definidos nas DLLs e exibe as novas configurações do jogo na tela. Nele está incluído o caso Compilar.
- Consultar ajuda – permite ver opções de ajuda para auxiliar aos novos usuários.
- Ver Sobre – exibe informações dos autores e do *software*.
- Ver Tutorial – exibe instruções gerais para o uso do PNA e os predicados específicos do *template sendo usado*.

4.2 Componentes do Sistema

O sistema está organizado em agrupamentos de arquivos físicos contendo códigos, bibliotecas, DLL's, imagens, entre outros. Muitos desses arquivos podem possuir dependências ou associações com troca de mensagens. Os principais componentes do *PNA Game Engine* são (Fig. 6):

- Fonte PROLOG – código inteiramente em PROLOG acrescido de predicados especiais que facilitam na comunicação com o C#.
- Compilador P# - Traduz o código PROLOG em arquivos de classe C#. Posteriormente ele será usado para comunicar com as DLLs geradas pelo CSC.
- CSC – *C# Compiler* é um executável da .Net *Framework* responsável por transformar as classes .cs em DLLs para permitir a exibição do jogo final em tempo de execução.
- *XNA Framework* – fornece todo suporte para a execução de rotinas em XNA.
- *User Control XNA* – componente criado no *Visual Studio* para permitir a integração de formulários com XNA sem problema de compatibilidade entre os gerenciadores de dispositivos gráficos.
- *Game Template PNA* – pacote que integra os arquivos necessários para a execução do respectivo *template*.

- Inicializador – representa um componente vital do sistema. Possui um conjunto de funções que utilizam a DLL do compilador P# para interpretar os predicados das DLLs das classes criadas pelo CSC e gerar o comportamento do jogo do *template*.
- Imagens e Sons – pacote que armazena os sons e imagens dos *sprites* necessários neste *template*.
- Exemplos – Códigos PNA de exemplo para o *template* carregado.

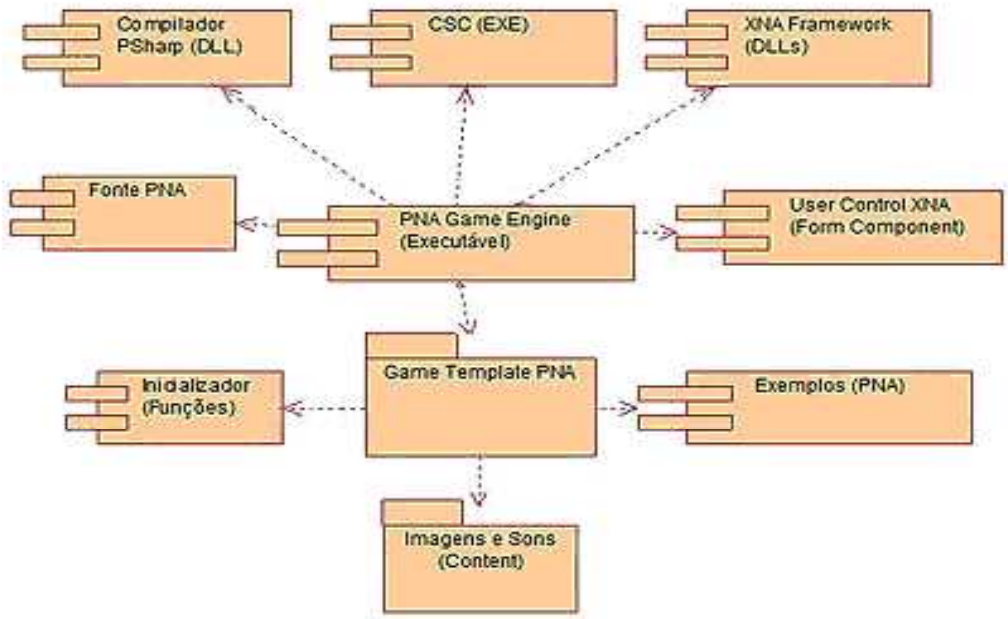

Fig. 6: Diagrama de componentes do sistema desenvolvido.

4.3 Sequência de Execução

Com base nos requisitos resumidos no diagrama UML (capítulo 4.1) e os componentes desenvolvidos (capítulo 4.2) é possível definir o fluxo sequencial de execução das principais rotinas do sistema. O diagrama de sequência dá ênfase à ordenação temporal em que as mensagens são trocadas entre os objetos de um sistema. Entende-se por mensagens os serviços solicitados de um objeto a outro, e as respostas desenvolvidas para as solicitações. No seguinte Diagrama de Seqüência de Análise (Fig. 7) projeta-se a interação dos componentes do sistema necessários para transformar o código PNA em uma aplicação renderizada no dispositivo gráfico.

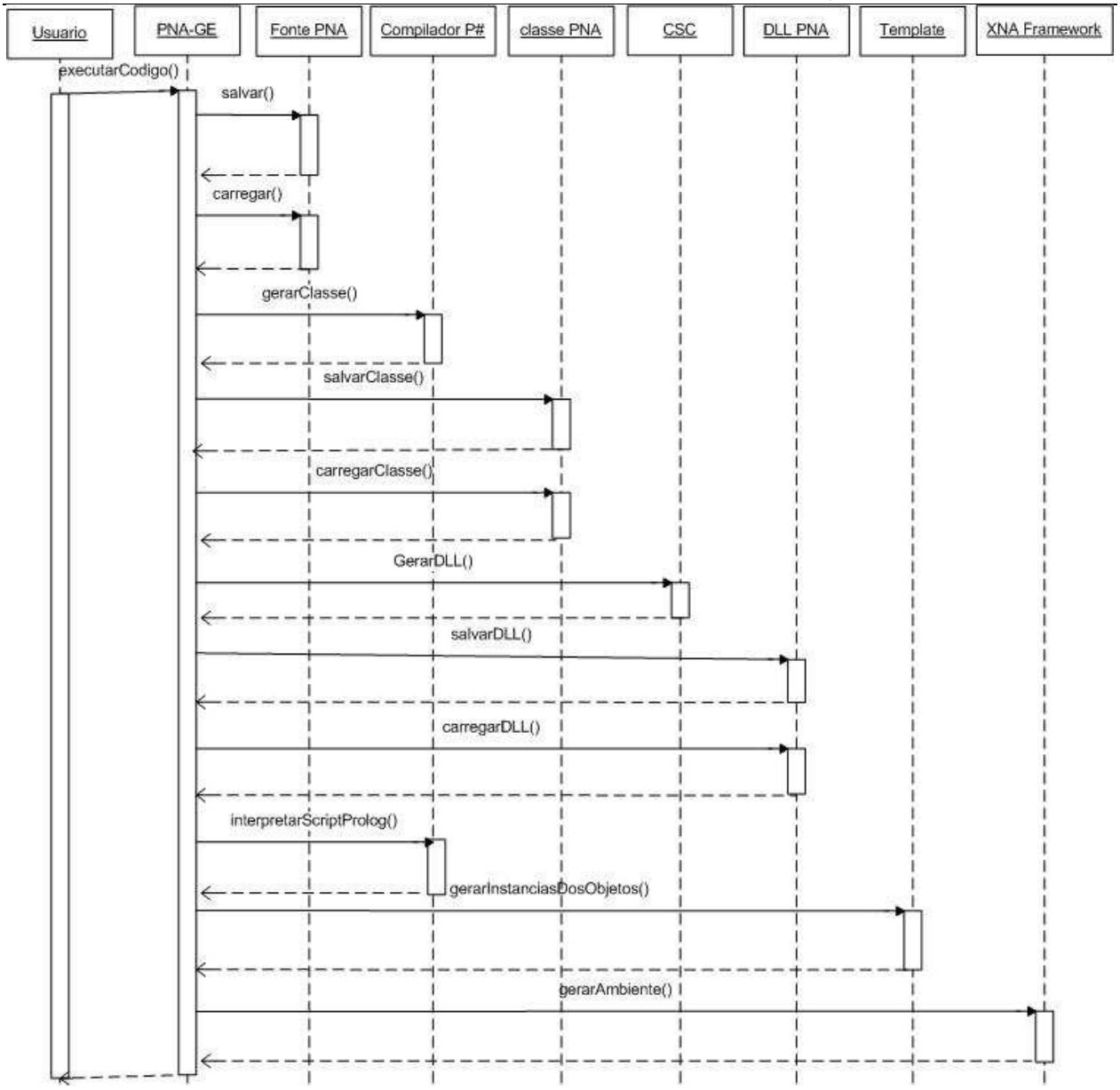

Fig. 7: Diagrama de Sequência de Análise do sistema desenvolvido.

Retrata-se, para tanto, a ação Executar Game. Neste diagrama podemos observar que o PNA *Game Engine* (*PNA-GE*) tem o controle de todo o sistema. O Usuário (no caso o aluno) ativará o comando *executarCodigo()* e receberá como retorno o jogo desenvolvido com seu *script*. O funcionamento de cada objeto do sistema pode ser definido como:

- Usuário – representa o ator do sistema. Considerando que o *script* PNA já tenha sido digitado, este diagrama ilustra uma ordem de compilação deste código (método *executarCodigo()*) e exibição do cenário do jogo gerado.

- PNA-GE (PNA *Game Engine*) - é o objeto do sistema responsável por sua gerência, por este motivo é o que possui a linha do tempo com maior interação (troca de mensagens).

- Fonte PNA – é o *script* PNA digitado pelo usuário e armazenado através do comando *salvar()* do objeto PNA-GE. É carregado para a memória através da método *carregar()*
- Compilador P#- neste ponto ele irá traduzir (através da método *gerarClasse()*) o código anteriormente salvo em classes. Uma para cada predicado definido pelo usuário. Logo salvará também estas classes.
- Classe PNA – Representa as classes salvas, mas que não serão usadas em tempo real. O objeto PNA-GE faz a requisição do carregamento destas classes (método *carregarClasse()*).
- CSC – É necessário que as classes estejam disponíveis em tempo real para a execução do jogo. Para tanto elas são transformadas em DLL pelo CSC. Esta compilação é feita pela método *GerarDLL()*.
- DLL PNA – Cada classe salva é transformada em DLL (método *salvarDLL()*). A partir deste ponto estas DLLs são carregadas e interpretadas (método *interpretarScriptProlog()*) pelo Compilador P# que será o responsável por gerar a instância dos objetos (método *gerarInstanciasDosObjetos()*).
- *Template* – Estas instâncias representam gráficos, sons e comportamentos, entre outros elementos do jogo definido pelo objeto *Template*. Este terá somente códigos XNA enviados (método *gerarAmbiente()*) para o XNA *Framework*.
- XNA *Framework* – este interpreta o código nativo da plataforma e devolve o retorno visual em forma do jogo do *template*.

4.4 Especificações do Game Template PNA

Em PROLOG para se obter as soluções para a consulta dos nomes dos filhos de um determinado pai, através de um predicado, poderia se fazer:

filhos (Lista_Dos_Nomes, 'José')

Da mesma forma a biblioteca P# fornece a seguinte interface para realizar a mesma consulta em C#:

VariableTerm Nome = new VariableTerm();
PROLOGInterface sharp = new PROLOGInterface();
sharp.SetPredicate(new filhos_2(Nome, SymbolTerm.MakeSymbol("José"), new ReturnCs(sharp)));

Onde *SetPredicate* é um método definido pelo P#, o qual exige os parâmetros variáveis para o retorno da consulta *(Nome)*, critério de pesquisa *(SymbolTerm.MakeSymbol("José"))* e o objeto instanciado da classe *PROLOGInterface*.

Com base neste princípio e utilizando a DLL da classe compilada do script PROLOG (código PNA) definido pelo usuário, estipularam-se nomes de predicados que seriam varridos pelo código C# para a definição de instanciações, posições e comportamentos em um dado *template*, como pode ser observado no seguinte código:

```
118 VariableTerm posX = new VariableTerm();
119 VariableTerm posY = new VariableTerm();
120 PrologInterface sharp = new PrologInterface();
121
122 for (int cont = 1; cont <= 100; cont++)            {
123    IntegerTerm i = new IntegerTerm(cont);
124    sharp.SetPredicate(new Objeto_4(
125    SymbolTerm.MakeSymbol("nave"), i, posX, posY, new ReturnCs(sharp)));
126    for (bool r = sharp.Call(); r; r = sharp.Redo()){
127       spaceShipTexture[cont] = contentMgr.Load<Texture2D>("nave");
128       spaceShipLocation[cont] = new Vector2(
129       float.Parse(posX.Dereference().ToString()),
130       float.Parse(posY.Dereference().ToString()));
131    }
132 }
```

Onde podemos destacar:

- Nas linhas 118 e 119 são definidos termos que acessarão os predicados do código PROLOG;
- Em 122 é definido um laço para varrer no máximo 100 predicados (e portanto gerar no máximo 100 “naves”);
- Em 124, no método SetPredicate, estão sendo dados dois critérios: deseja-se localizar o objeto cujo predicado seja “objeto”, com quatro termos, onde o primeiro seja “nave” e o segundo um índice entre 1 e 100;
- Em 126 faz-se a busca no script PROLOG para a consulta pré-definida;
- Em 127 e 128, caso exista a “nave” especificada, inicia-se o processo de carregamento e posicionamento para a posterior renderização da mesma na tela, através de diretivas XNA.

4.5 O Código PNA para o Template River Raid

O PNA é uma mescla de PROLOG com predicados pré-definidos do *template* específico carregado. Nele é possível inserir objetos 3D ou 2D (dependendo do *template*), alterar o visual, fazer o *game design*, controlar o fluxo dos dados, gerenciar o comportamento, animar, etc.

Foi implementado, nessa primeira versão do programa, somente o comportamento inicial do jogo. No método *Update()* estará pré-configurada a animação do movimento dos objetos simulando o vôo de uma nave.

Cada template possui seus predicados reservados pré-definidos, os quais podem ser conferidos na tela Ver Tutorial. Para auxiliar ainda o usuário (aluno) os exemplos pré-desenvolvidos de cada *template* utilizam todos os predicados e valores chaves. *Como* exemplo, na Tab. 1 e na Tab. 2 são criados códigos para o *template River Raid* com os seguintes predicados PNA:

- objeto – permite inserir um *sprite* 2D. Possui quatro parâmetros: nome do objeto (restrito aos objetos existentes no jogo: nave e barco), índice

(identificador do objeto, deve ser único para cada de objeto), posição X e posição Y.

- controle – permite definir os controles básicos (movimentos e tiros) de evento do teclado para um determinado objeto. Os dois parâmetros permitem identificar o objeto a ser manipulado: nome e índice.

O exemplo da Tab. 1 cria a posição inicial de 4 barcos e de 1 nave, e define o controle do teclado para a nave.

```
objeto(nave,1,0,0).
objeto(barco,1,150,50).
objeto(barco,2,50,150).
objeto(barco,3,250,250).
objeto(barco,4,150,350).
controle(nave,1).
```

Tab. 1: Exemplo de um código em PNA

O exemplo da Tab. 2 gera um barco com uma posição pré-determinada e outros 4 com posições aleatórias no eixo x e enfileirados no eixo y. Ele também insere e cria o controle para uma nave.

```
objeto(nave,1,0,0).
controle(nave,1).

objeto(barco,1,150,50).
objeto(barco,N,POSX,POSY):-
N>1,
N<6,
Ntemp is N-1,
objeto(barco,Ntemp,POSXtemp,POSYtemp),
POSX is integer(random*300),
POSY is POSYtemp+100
.
```

Tab. 2: Mesmo código da Tab. 1 criado a partir de recursão.

Portanto, estes códigos das tabelas anteriores são semelhantes, pois distribuem barcos no caminho à frente da nave (Fig. 8). Entretanto, eles exercitam técnicas distintas de PROLOG: o primeiro mostra claramente o uso de predicados e atributos formando um conjunto de fatos; no segundo é criado uma cláusula com o uso de recursividade e todo princípio definido no primeiro exemplo.

Fig. 8: Exemplo de código PNA para o jogo River Raid

Os exemplos da Tab. 1 e Tab. 2 estão disponíveis no PNA Game Engine com os nomes Exemplo Simples e Exemplo com Recursão, respectivamente. Estes exemplos podem ser carregados a qualquer momento para servir como base para o desenvolvimento de códigos mais complexos. Além destes dois exemplos, esta engine ainda disponibiliza um terceiro exemplo denominado Exemplo de Operações (Tab. 3).

```
controle(nave,1).
objeto(nave,1,250,350).

objeto(barco,1,POSX,50):-
POSX is integer(200),
POSX>100
.
```

Tab. 3: Exemplo de um código PNA com operações básicas

Neste terceiro exemplo o barco de índice 1 somente é instanciado se a variável POSX for maior que 100. Para a resolução deste problema é demonstrado o uso de operação de atribuição (is) e comparação (>). O cenário do jogo da Tab. 3 pode ser conferido na Fig. 9.

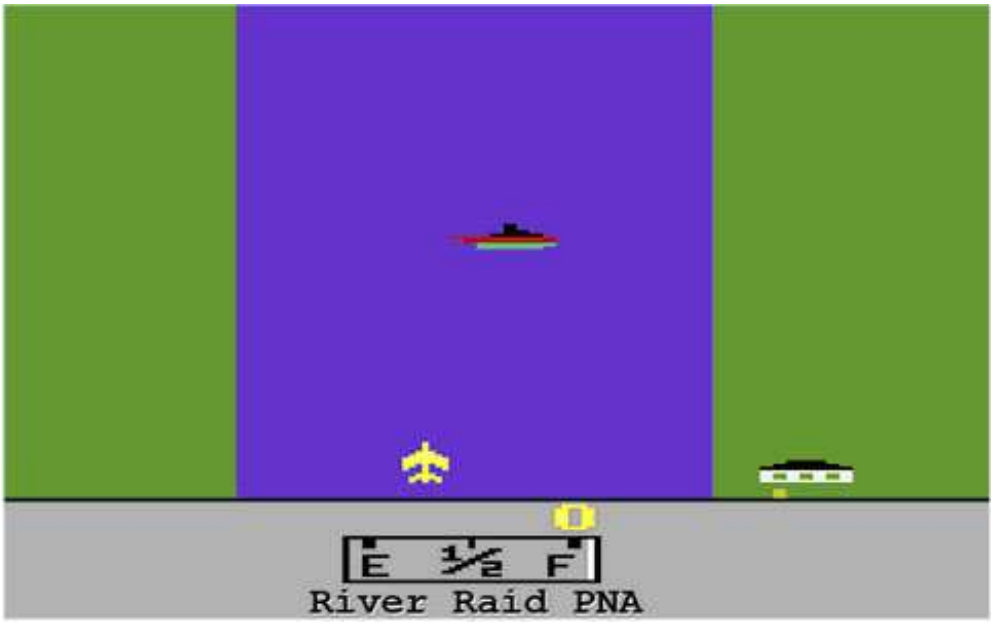

Fig. 9: Exemplo de um cenário do jogo para o exemplo de operações básicas

4.6 Protótipos e Implementações

A implementação do software final descrito neste trabalho foi feito através de prototipagem. Um protótipo pode ser considerado como uma implementação concreta, embora parcial, de um programa. Estes protótipos foram criados para explorar múltiplas questões durante o desenvolvimento do software.

O P# possui duas versões: uma standalone e outra encapsulada em uma DLL. Testou-se primeiramente o GUI (*Graphical User Interface*) *standalone* do *software* P# *Compiler* (Fig. 10 – a). Nesta primeira etapa criaram-se predicados simples, como testar se um número é impar ou não. Nesta versão é possível compilar programas PROLOG em arquivos C#. Esta GUI possui todas as funcionalidades da biblioteca de comandos encapsulada na DLL usada neste sistema. Nestes testes foi possível observar uma satisfatória avaliação de requisitos como performance (velocidade de tradução do código), eficiência (se todos os predicados PROLOG foram transformados em classes) e segurança (no sentido de uma conversão consistente e correta dos códigos).

Como um dos requisitos do sistema seria a integração do formulário do Visual Studio com o XNA, foi desenvolvido um protótipo de integração dos mesmos (Fig. 10 – b). Este possibilitaria o teste das principais bibliotecas de códigos e a verificação da performance e interoperabilidade. Para este fim prototipou-se um exemplo simples de controle de uma figura geométrica (triângulo) através de um componente de alto nível do *Visual Studio* (*ScrollBar*).

A partir do código de verificação de número ímpar, foi criado um código para testar a incorporação do código PROLOG no C# através da interpretação pela DLL do P# (ilustrada na Fig. 10 – c).

O próximo passo foi o desenvolvimento de uma ferramenta que integrasse todos os módulos anteriores e outras funções básicas (Fig. 10 – d). Esta representa o ultimo protótipo antes do desenvolvimento final. Nela foram encapsuladas funções importantes em módulos separados para verificação do funcionamento de cada uma, onde podemos destacar:

- Salvar Arquivo .pl – consiste em gravar um arquivo físico do *script* digitado pelo usuário no componente *RichTextBox* no mesmo diretório do programa.
- Compilar em .cs – salva um arquivo de classe .cs para cada predicado definido pelo usuário.
- Compilar em DLL – utilizando o CSC é feito a conversão de cada arquivo .cs em DLL.
- Atualizar – executa o jogo baseado na DLL compilada.

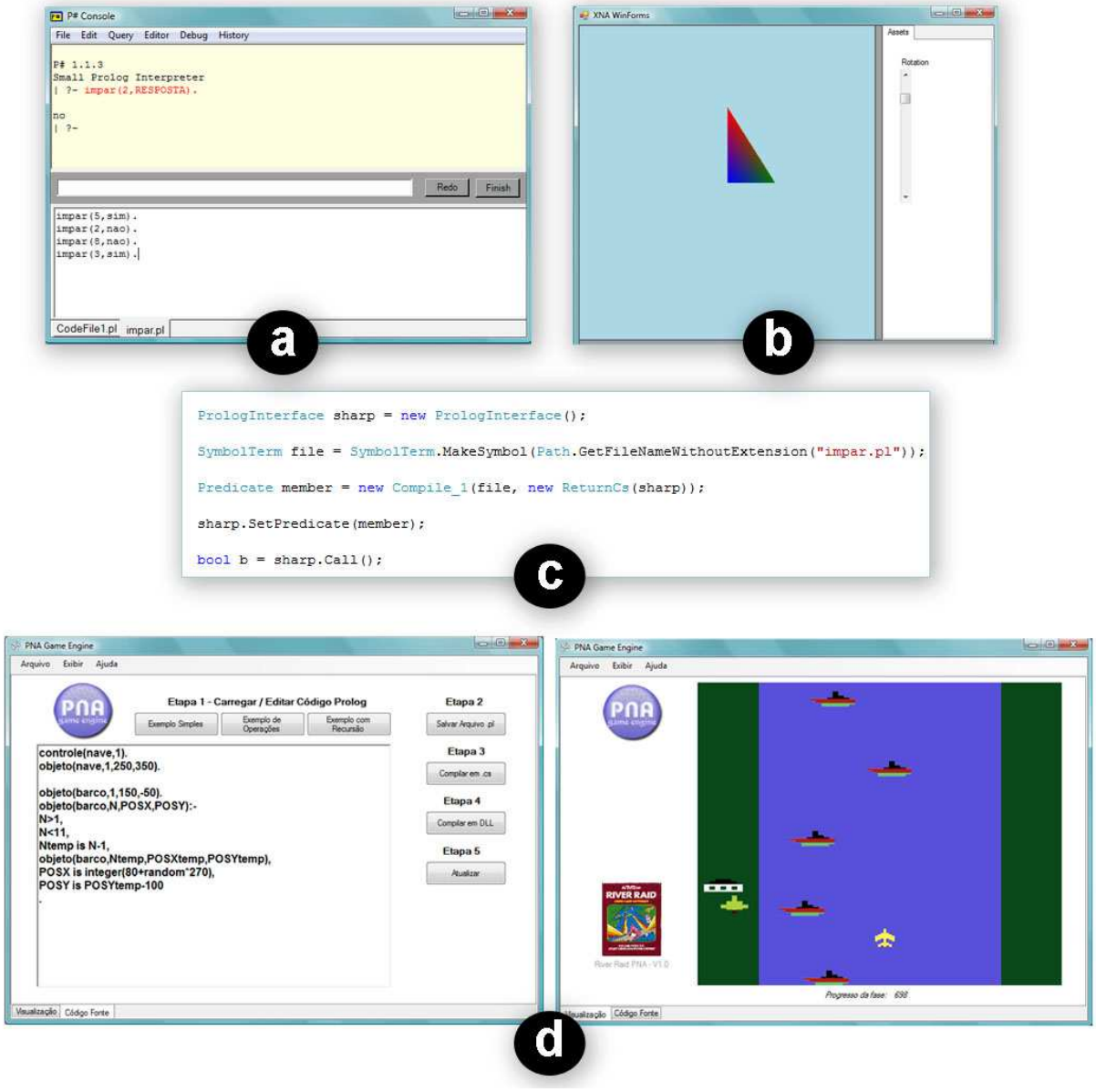

Fig. 10: Principais implementações e protótipos desenvolvidos

O *software* PNA *Game Engine* (Fig. 1) foi desenvolvido a partir da otimização do código deste ultimo protótipo.

Capítulo 5

O PNA Game Engine Aplicado ao Ensino de Computação

A educação atual passa pela utilização de novos meios e tecnologias disponíveis no mercado [ARAÚJO 2006]. Assim sendo, garantir formas de aprendizado que motivem e que facilitem a aquisição de conhecimentos, de forma que se obtenham resultados satisfatórios, é também uma maneira de incentivar a aprendizagem de um aluno na escola.

A tecnologia moderna e a rapidez com que ela se atualiza contribuíram para essa mudança nas escolas dos dias de hoje. Alguns autores colocam que os métodos de ensino de algumas décadas do passado requeriam um aprendizado fortemente baseado na memorização e repetição da matéria [BESSA 2008], que consiste numa abordagem oposta ao construtivismo, defendido neste trabalho.

Assim como diversas áreas de conhecimento da humanidade, a pedagogia também foi beneficiada com a ajuda da tecnologia. Através de meios multimídia, como jogos eletrônicos, os alunos possuem maior facilidade em aprender e compreender a matéria.

Porém, apesar de toda a tecnologia disponível, ainda há o problema de que o professor muitas vezes desconhece esses meios, ou, quando conhece, não sabe usá-los, perdendo, assim, a chance de obter uma maior motivação dos alunos pela matéria, e, como conseqüência, a melhora no desempenho escolar.

Neste trabalho utiliza-se uma metodologia pedagógica baseada na Sequência Fedathi [NETO & SANTANA 2001]. Esta proposta é capaz de intermediar a abordagem da lógica declarativa através da ferramenta lúdica descrita nos capítulos anteriores.

5.1 Sequência Fedathi

A Sequência Fedathi é uma metodologia pedagógica baseada na aprendizagem por resolução de problemas explorados.

Nesta teoria, são categorizados os níveis de desenvolvimento do pensamento lógico que uma pessoa utiliza quando é solicitada a resolver um problema [NETO e BORGESI 2006]. A Sequência é composta pelas etapas:

- Tomada de posição: Quando uma situação é apresentada a um indivíduo, ele faz uma tomada de posição, ou seja, ele recorre a uma base de conhecimentos específicos, que são mais eficazes para a resolução da referida situação;
- Maturação: Onde a situação é amadurecida, associada, organizada e comparada com outras já conhecidas, em busca de uma solução para o problema;
- Solução: Depois do relaxamento de encontrar uma solução, parte-se em busca da prova, que reúna a solução mais otimizada, mais elaborada, sem redundâncias;
- Prova: Esta ultima solução propicia partir-se para generalizações, para elaborações de modelos teóricos.

O ensino por meio de multimídia pode ser aplicado na tomada de posição. De acordo com [NETO & SANTANA 2001], cabe ao professor elaborar problemas que estejam devidamente contextualizados em relação ao saber acadêmico. Para tanto, é necessário viabilizar os elementos necessários à imersão cultural do aluno, processo este que é essencial ao desenvolvimento da fase seguinte.

5.2 Workflow Pedagógico Proposto

O processo de ensino, ou workflow pedagógico, para a utilização de desenvolvimento de jogos em sala através da Seqüência Fedathi pode ser modelada nas seguintes 4 etapas (Fig. 11):

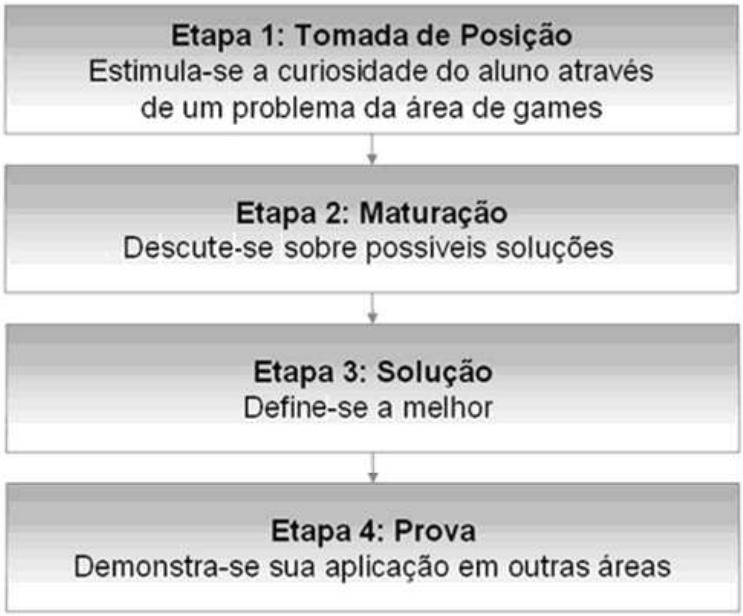

Fig. 11: Workflow pedagógico baseado na Sequência Fedathi.

- Etapa 1: Atividade prática do aluno com o recurso de multimídia interativo. Representa uma transposição didática de um problema determinado. Caracteriza a tomada de posição, pois o aluno emerge no ambiente do mundo virtual, conhece as regras que o regem e pode recorrer a lembranças de outros conteúdos ensinados pelo professor, por exemplo. Neste momento, também é possível diagnosticar as dificuldades em relação à aprendizagem do conteúdo em questão. A postura didática do professor é a de não-intervenção para que o aluno possa pensar e indagar-se.
- Etapa 2: O professor deve iniciar as discussões com os alunos sobre o problema em questão e propor que este desenvolva seus raciocínios e argumentos. É um momento de discussão em grupo. Ciente da situação transmitida pelo desenho, o aluno busca por conta própria deduzir soluções para novos problemas relacionados. Representa a maturação.

- Etapa 3: Etapa em que o professor formaliza e confronta as possíveis soluções ou observações definidas pelos alunos. O professor deve valorizar todas as soluções debatidas independentemente de estarem corretas ou não. A confrontação normalmente requer o uso de contra-exemplos locais e globais do professor, que serve como ferramenta para direcionar o aluno ao caminho certo sem definir limites em seu raciocínio. Representa a etapa de solução da seqüência Fedathi.
- Etapa 4: Parte teórica em que o professor conecta o desenho animado, as soluções propostas pelos alunos e o modelo cientifico correto. Neste momento, a solução é formalizada e as idéias são mais uma vez revisadas. Podem ser usados exemplos ainda não explorados para demonstrar a aplicação do conteúdo em outras áreas ou situações.

Este workflow pedagógico tem como objetivo nortear o docente na prática do ensino utilizando-se de uma ferramenta lúdica como apoio, e não como instrumento principal da transposição didática do conteúdo.

5.3 O PNA Game Engine como Meio de Ensino

Para o ensino de conteúdos específicos presentes dentro da lógica declarativa, através do PNA *Game Engine* e o *template River Raid* utilizando-se o workflow proposto, o docente poderia apresentar aulas específicas como segue:

Conteúdo	**Etapa 1** Problema a ser lançado para os alunos e resolvido através do PNA-GE.	**Etapa 2** Ainda no PNA-GE, o professor orienta os alunos em busca de uma solução que seja mais próxima da correta.	**Etapa 3** De forma teórica, faz-se a formalização da melhor solução para o problema.	**Etapa 4** Generalização desta solução.
Notação Clausal e fatos	Distribuir 20 navios no rio, com a restrição dos navios não coincidirem na mesma posição.	Criação de um conjunto de cláusulas para a inserção dos navios observando para não se repetir coordenadas X e Y. Princípio: Tab. 1.	Define-se que todo fato é composto por um predicado que estabelece uma relação entre seus argumentos e encerrado por um ponto.	A linguagem de especificação é entendida pela máquina e é,por si só, uma linguagem de programação. O refinamento de especificações é mais efetivo do que o refinamento de programas. Não há distinção entre o programa e os dados.
Regras	Criar regras para o desenho de 5 naves, as quais só serão desenhadas se sua posição Y for superior a 100 e inferior a 300.	Criação de um conjunto de cláusulas para a inserção dos navios através de implicações. Princípio: Tab. 3.	Define-se que são condições que devem ser satisfeitas para que uma certa declaração seja considerada verdadeira e executada.	Neste ponto, observa-se que um fato é sempre verdadeiro (como visto no exercício anterior) e regras especificam algo que “pode ser verdadeiro se algumas condições forem satisfeitas”.

Recursividade	O mesmo proplema da Notação Clausal, mas utilizando recursividade. Sugerir a criação de um nível de um nível de dificuldade (exemplo: quanto maior o valor de Y, maior será a freqüência de navios distribuídos).	Ultilizando notação clausau e regras, definir uma relação em termos de si própria. Princípio: Tab. 2.	Na definição de qualquer função recursiva é necessário realizar uma análise sobre o argumento da função que deve ser o valor inicial do argumento, e neste caso a definição especificará o valor da função aplicado para aquele argumento.	A recursão, em Prolog, é a forma natural de ver e representar dados e programas. Substituindo laços do tipo "for" ou "while".

Considerando o fato que o aluno desenvolverá uma fase do jogo e logo fará testes e apresentará a seus amigos, estes três exercícios buscam de forma lúdica ensinar conteúdos específicos da programação declarativa apoiado na ferramenta PNA Game Engine.

Capítulo 6

Conclusão

Neste trabalho foi apresentado o *PNA Game Engine*, uma ferramenta com a entrada de dados em PROLOG e o retorno visual de um jogo desenvolvido em XNA. Para este fim foi desenvolvida a API PNA *Game Engine* capaz de controlar o comportamento do *game template* através do paradigma de programação declarativo.

Esta *engine* é capaz de gerar elementos visuais e comportamentos de objetos em jogos a partir de programas declarativos simples. Assim como em [HARTNESS 2004], este aplicativo busca motivar o aprendizado de técnicas de programação, mais especificamente de programação em lógica, utilizando a linguagem PROLOG. Além deste objetivo pedagógico, demonstrou-se que a criação de um *engine* voltado para a programação de jogos através de uma interface em PROLOG pode auxiliar na exploração científica da área de *games*.

Todos os componentes do sistema criado utilizam tecnologia *Microsoft* e foram desenvolvidos na versão *freeware* da plataforma .Net.

Na engenharia deste processo destacam-se, devido à interoperabilidade proporcionada, as rotinas de integração dos formulários do *Visual Studio* com o XNA e

comunicação da linguagem C# com o PROLOG através do uso do compilador P# [COOK 2003].

Capítulo 7

Trabalhos Futuros

A partir da análise dos jogos de RPG, é possível definir questões a serem consideradas no *game design* que posteriormente serão desenvolvidas para um conjunto mais concreto de ferramentas para a utilização na narrativa interativa em jogos digitais (*Interactive Storytelling*). Entre estas questões é possível se destacar a criação de diálogos. O PNA *Game Engine* pode auxiliar no desenvolvimento de novas técnicas que podem repercutir em diálogos mais dinâmicos, proporcionando assim um *game* de RPG mais realista. Assim sendo, deseja-se criar um novo *template* orientado a criação das interações entre agentes no que tange a diálogos.

Outra continuidade natural deste trabalho é a criação de novos *game templates* em ambientes virtuais 3D, dentre os quais destacamos:

- Jogos no estilo de “Mundo de Wumpus” - jogo de tabuleiro em que um caçador deve andar entre as casas a procura de um tesouro. Estas casas podem possuir elementos nocivos ao caçador (fogueira e o monstro Wumpus) e dicas para evitar estes elementos. O Mundo de Wumpus tem sido amplamente utilizado no ensino de conceitos fundamentais de Inteligência Artificial.

- Jogo do gênero *Racing* – jogo de corrida de carros com obstáculos e disputas de velocidade. Jogos de corrida possibilitam o aprendizado de técnicas avançadas de programação, como por exemplo, programação *multithreaded*.

Embora existam experimentos preliminares com resultados positivos de aplicação em aulas de inteligência artificial, especificamente para o ensino de PROLOG, de ferramentas gráficas [SILVA & SILVA 2006] [SILVA & MELO 2007], é de interesse dos autores documentarem atividades práticas em sala de aula, através do workflow proposto baseado na Sequência Fedathi, para uma avaliação mais criteriosa dos resultados.

Este trabalho atual não define um padrão de projeto para a criação de novos game templates. Portanto, possui a desvantagem de baixa (ou difícil) escalabilidade com relação à incorporação de novos modelos de jogos. Pretende-se, então, atualizar esta engine criando ferramentas para importação de game *templates* e estabelecer o padrão de projeto que os mesmos devem ter.

Apêndice

A.1 Código da Classe XnaView

Abaixo é mostrado o código da classe XnaView, desenvolvido sob um formulário com o componente UserControl. O XnaView é a base da integração do C# com o XNA.

```
using System.Windows.Forms;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace XnaInForm
{
    public partial class XnaView : UserControl
    {
        #region Members & Properties

        public delegate void PaintFunction(XnaView ctrl);
        private PaintFunction paintMe;

        private Vector4 clearColor;
        public Vector4 ClearColor
        {
            get { return clearColor; }
        }

        private RenderTarget2D renderTarget;
        public RenderTarget2D RenderTarget
        {
            get { return renderTarget; }
        }

        private DepthStencilBuffer depthStencil;
        public DepthStencilBuffer DepthStencil
        {
            get { return depthStencil; }
        }
```

```
        private Texture2D texture;
        public Texture2D Texture
        {
            get { return texture; }
            set { texture = value; }
        }

        public Rectangle ClientArea
        {
            get { return new Rectangle(0, 0, Width, Height); }
        }

        #endregion

        #region Code

        public XnaView()
        {
            InitializeComponent();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            if(paintMe != null)
                paintMe(this);
        }

        protected override void OnPaintBackground(PaintEventArgs e)
        {
        }

        public void Initialize(GraphicsDevice gfxDevice, PaintFunction
paintFct, Vector4 clearColor)
        {
            this.paintMe = paintFct;
            this.clearColor = clearColor;

            renderTarget = new RenderTarget2D(gfxDevice, Width, Height, 1,
SurfaceFormat.Color);
            depthStencil = new DepthStencilBuffer(gfxDevice, Width, Height,
DepthFormat.Depth24);
        }

        #endregion

        private void XnaView_Load(object sender, System.EventArgs e)
        {

        }
    }
}
```

A.2 Código da Classe XnaForm

Abaixo é mostrado o código da classe XnaForm. É a classe principal do sistema, desenvolvida sobre um formulário e agrupando todas as funções do sistema.

```
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Text;
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Content;
using JJC.Psharp.Lang;
using JJC.Psharp.Predicates;
using JJC.Psharp.Resources;
using System.Diagnostics;

namespace XnaInForm{
public partial class XnaForm : Form{
    int[] barcomorto = new int[1000];
    int progresso = 0;
    private float _time;
    private int _curFrame;
    Texture2D explo;
    int temploexplosao = 10;
    Vector2 posicaoexplosao = new Vector2();
    Texture2D marTexture;
    Vector2 marLocation = new Vector2();
    Texture2D tiroTexture;
    Vector2 tiroLocation = new Vector2();
    Texture2D casaTexture1;
    Vector2 casaLocation1 = new Vector2();
    Texture2D casaTexture2;
    Vector2 casaLocation2 = new Vector2();
    Texture2D casaTexture3;
    Vector2 casaLocation3 = new Vector2();
    Texture2D[] spaceShipTexture = new Texture2D[1000];
    Vector2[] spaceShipLocation = new Vector2[1000];
    Vector2[] spaceShipLocationN = new Vector2[1000];
    Texture2D[] spaceShipTexture2 = new Texture2D[1000];
    Vector2[] spaceShipLocation2 = new Vector2[1000];
    Vector2[] spaceShipLocationN2 = new Vector2[1000];
    int ncontrolado;
    int numero = 0;
    int numero2 = 0;
    int cont2;
    int cont3;
    float velocidade = 1.5f;
```

```
    class GfxService : IGraphicsDeviceService
    {
        GraphicsDevice gfxDevice;
        public GfxService(GraphicsDevice gfxDevice)
        {
            this.gfxDevice = gfxDevice;
            DeviceCreated = new EventHandler(DoNothing);
            DeviceDisposing = new EventHandler(DoNothing);
            DeviceReset = new EventHandler(DoNothing);
            DeviceResetting = new EventHandler(DoNothing);
        }

        public GraphicsDevice GraphicsDevice
        { get { return gfxDevice; } }

        public event EventHandler DeviceCreated;
        public event EventHandler DeviceDisposing;
        public event EventHandler DeviceReset;
        public event EventHandler DeviceResetting;

        void DoNothing(object o, EventArgs args)
        {
        }
    }

    GraphicsDevice gfxDevice;
    ContentManager contentMgr;
    DepthStencilBuffer defaultDepthStencil;
    SpriteBatch copySprite;
    SpriteBatch smallViewSprite;
    long lastTimeCount = 0;

    public XnaForm()
    {
        InitializeComponent();

        CreateDevice();
        copySprite = new SpriteBatch(gfxDevice);
        defaultDepthStencil = gfxDevice.DepthStencilBuffer;

        smallView.Initialize(gfxDevice, new XnaView.PaintFunction(Blit), new
Vector4(0.5f, 0.5f, 0.5f, 1.0f));

        GfxService gfxService = new GfxService(gfxDevice);
        GameServiceContainer services = new GameServiceContainer();
        services.AddService(typeof(IGraphicsDeviceService), gfxService);
        contentMgr = new ContentManager(services);

        explo = contentMgr.Load<Texture2D>("explos");
        posicaoexplosao = new Vector2(-1000.0f, -1000.0f);

        marTexture = contentMgr.Load<Texture2D>("mar2");
        marLocation = new Vector2(0.0f, 0.0f);

        tiroTexture = contentMgr.Load<Texture2D>("tiro");
        tiroLocation = new Vector2(-100.0f, -100.0f);

        casaTexture1 = contentMgr.Load<Texture2D>("casas");
        casaLocation1 = new Vector2(5.0f, -50.0f);
        casaTexture2 = contentMgr.Load<Texture2D>("casas");
```

```
        casaLocation2 = new Vector2(5.0f, -600.0f);
        casaTexture3 = contentMgr.Load<Texture2D>("casas");
        casaLocation3 = new Vector2(450.0f, -400.0f);
        spaceShipTexture = new Texture2D[1000];
        spaceShipLocation = new Vector2[1000];
        spaceShipLocationN = new Vector2[1000];
        spaceShipTexture2 = new Texture2D[1000];
        spaceShipLocation2 = new Vector2[1000];
        spaceShipLocationN2 = new Vector2[1000];

        VariableTerm person1 = new VariableTerm();
        VariableTerm person2 = new VariableTerm();
        VariableTerm person3 = new VariableTerm();
        PrologInterface sharp = new PrologInterface();
        numero = 0;
        numero2 = 0;

        for (int cont = 1; cont <= 100; cont++)
        {
            IntegerTerm i = new IntegerTerm(cont);
            sharp.SetPredicate(new Objeto_4(SymbolTerm.MakeSymbol("barco"),
i, person2, person3, new ReturnCs(sharp)));
            for (bool r = sharp.Call(); r; r = sharp.Redo())
            {
                spaceShipTexture[cont] = contentMgr.Load<Texture2D>("barco");
                spaceShipLocation[cont] = new
Vector2(float.Parse(person2.Dereference().ToString()),
float.Parse(person3.Dereference().ToString()));
                spaceShipLocationN[cont] = new
Vector2(float.Parse(person2.Dereference().ToString()),
float.Parse(person3.Dereference().ToString()));
                if (person2.Dereference().ToString() != null)
                {
                    numero++;
                }
            }
        }

        person1 = new VariableTerm();
        person2 = new VariableTerm();
        person3 = new VariableTerm();
        sharp = new PrologInterface();

        for (int cont = 1; cont <= 100; cont++)
        {
            IntegerTerm i = new IntegerTerm(cont);
            sharp.SetPredicate(new Objeto_4(SymbolTerm.MakeSymbol("nave"), i,
person2, person3, new ReturnCs(sharp)));
            for (bool r = sharp.Call(); r; r = sharp.Redo())
            {
                spaceShipTexture2[cont] = contentMgr.Load<Texture2D>("nave");
                spaceShipLocation2[cont] = new
Vector2(float.Parse(person2.Dereference().ToString()),
float.Parse(person3.Dereference().ToString()));
                spaceShipLocationN2[cont] = new
Vector2(float.Parse(person2.Dereference().ToString()),
float.Parse(person3.Dereference().ToString()));
                if (person2.Dereference().ToString() != null)
                {
                    numero2++;
```

```
                }

            }
        }

        VariableTerm n = new VariableTerm();
        sharp.SetPredicate(new Controle_2(SymbolTerm.MakeSymbol("nave"), n,
new ReturnCs(sharp)));
        sharp.Call();

        ncontrolado = int.Parse(n.Dereference().ToString());

        smallViewSprite = new SpriteBatch(gfxDevice);

        Application.Idle += new EventHandler(Application_Idle);
        lastTimeCount = PerformanceCounter.QueryPerformanceCounter();
    }

    private void CreateDevice()
    {
        PresentationParameters presentation = new PresentationParameters();
        presentation.AutoDepthStencilFormat = DepthFormat.Depth24;
        presentation.BackBufferCount = 1;
        presentation.BackBufferFormat = SurfaceFormat.Color;
        presentation.BackBufferWidth = 640;
        presentation.BackBufferHeight = 480;
        presentation.DeviceWindowHandle = this.Handle;
        presentation.EnableAutoDepthStencil = true;
        presentation.FullScreenRefreshRateInHz = 0;
        presentation.IsFullScreen = false;
        presentation.MultiSampleQuality = 0;
        presentation.MultiSampleType = MultiSampleType.None;
        presentation.PresentationInterval = PresentInterval.One;
        presentation.PresentOptions = PresentOptions.None;
        presentation.SwapEffect = SwapEffect.Discard;

        gfxDevice = new GraphicsDevice(GraphicsAdapter.DefaultAdapter,
DeviceType.Hardware, this.Handle,
            CreateOptions.HardwareVertexProcessing, presentation);
    }

    private void Blit(XnaView viewCtrl)
    {
        if (viewCtrl.Texture == null)
        {
            ClearOptions options = ClearOptions.Target |
ClearOptions.DepthBuffer;
            Vector4 clearColor = new Vector4(1, 0, 0, 1);
            float depth = 1;
            int stencil = 128;
            gfxDevice.Clear(options, clearColor, depth, stencil);
        }

        else
        {
            gfxDevice.SetRenderTarget(0, null);
            gfxDevice.DepthStencilBuffer = defaultDepthStencil;
            gfxDevice.RenderState.FillMode = FillMode.Solid;
            copySprite.Begin();
```

```
            copySprite.Draw(viewCtrl.Texture, Vector2.Zero,
Microsoft.Xna.Framework.Graphics.Color.White);
            copySprite.End();
            gfxDevice.SetRenderTarget(0, viewCtrl.RenderTarget);
            gfxDevice.DepthStencilBuffer = viewCtrl.DepthStencil;
        }

        gfxDevice.Present(viewCtrl.ClientArea, null, viewCtrl.Handle);
    }

    private delegate void DrawView();

    private void RenderToTexture(XnaView viewCtrl, DrawView drawFunction)
    {
        gfxDevice.SetRenderTarget(0, viewCtrl.RenderTarget);
        gfxDevice.DepthStencilBuffer = viewCtrl.DepthStencil;

        ClearOptions options = ClearOptions.Target |
ClearOptions.DepthBuffer;
        Vector4 clearColor = viewCtrl.ClearColor;
        float depth = 1;
        int stencil = 128;
        gfxDevice.Clear(options, clearColor, depth, stencil);
        drawFunction();
        gfxDevice.ResolveRenderTarget(0);
        viewCtrl.Texture = viewCtrl.RenderTarget.GetTexture();
    }


    private void DrawSmallView()
    {
        smallViewSprite.Begin();
        smallViewSprite.Draw(marTexture, marLocation, null, Color.White,
0.0f, new Vector2(0.0f, 0.0f), 1.0f, SpriteEffects.None, 0.0f);

        smallViewSprite.Draw(tiroTexture, tiroLocation, null, Color.White,
0.0f, new Vector2(0.0f, 0.0f), 1.0f, SpriteEffects.None, 0.0f);

        smallViewSprite.End();

        smallViewSprite.Begin();
        for (cont3 = 1; cont3 <= numero; cont3++)
        {
            if (barcomorto[cont3] == 1)
            {
                spaceShipLocation[cont3].X = 1000.0f;
                spaceShipLocation[cont3].Y = 0.0f;
            }

            smallViewSprite.Draw(spaceShipTexture[cont3],
spaceShipLocation[cont3], null, Color.White, 0.0f, new Vector2(0.0f, 0.0f),
1.0f, SpriteEffects.None, 0.0f);
        }
        for (cont3 = 1; cont3 <= numero2; cont3++)
        {
            smallViewSprite.Draw(spaceShipTexture2[cont3],
spaceShipLocation2[cont3], null, Color.White, 0.0f, new Vector2(0.0f, 0.0f),
1.0f, SpriteEffects.None, 0.0f);
        }
```

```
        smallViewSprite.Draw(casaTexture1, casaLocation1, null, Color.White,
0.0f, new Vector2(0.0f, 0.0f), 1.0f, SpriteEffects.None, 0.0f);
        smallViewSprite.Draw(casaTexture2, casaLocation2, null, Color.White,
0.0f, new Vector2(0.0f, 0.0f), 1.0f, SpriteEffects.None, 0.0f);
        smallViewSprite.Draw(casaTexture3, casaLocation3, null, Color.White,
0.0f, new Vector2(0.0f, 0.0f), 1.0f, SpriteEffects.None, 0.0f);
        smallViewSprite.End();
        for (cont3 = 1; cont3 <= numero; cont3++)
        {
            if ((spaceShipLocation[cont3].X < tiroLocation.X) &&
(spaceShipLocation[cont3].X + 70 > tiroLocation.X) &&
(spaceShipLocation[cont3].Y - 10 < tiroLocation.Y) &&
(spaceShipLocation[cont3].Y + 30 > tiroLocation.Y))
            {
                _curFrame = 0;
                tiroLocation.X += 10000;

                barcomorto[cont3] = 1;

                temploexplosao = 20;
                posicaoexplosao.X = spaceShipLocation[cont3].X;
                posicaoexplosao.Y = spaceShipLocation[cont3].Y;
            }
        }

        if (temploexplosao > 0)
        {
            temploexplosao--;
            smallViewSprite.Begin(SpriteBlendMode.Additive);
            smallViewSprite.Draw(explo,
            new Vector2(posicaoexplosao.X - 30f, posicaoexplosao.Y - 40f),
            new Rectangle(_curFrame * 100, 0, 100, 100),
            Color.White,
            0,
            new Vector2(0, 0),
            new Vector2(1.5f, 1.5f),
            SpriteEffects.None,
            0
            );
            smallViewSprite.End();
        }
    }

    private void Draw()
    {
        gfxDevice.RenderState.CullMode = CullMode.CullCounterClockwiseFace;
        gfxDevice.RenderState.DepthBufferEnable = true;
        gfxDevice.RenderState.DepthBufferFunction =
CompareFunction.LessEqual;
        gfxDevice.RenderState.DepthBufferWriteEnable = true;
        RenderToTexture(smallView, DrawSmallView);
        Blit(smallView);
    }

    private void Update(float deltaTime)
    {
        _time += 1;
        if (_time > 0.1f)
        {
            _curFrame++;
```

```
            if (_curFrame == 26)
            {
                _curFrame = 0;
            }
            _time = 0.0f;

        }
        for (cont2 = 1; cont2 <= numero; cont2++)
        {
            spaceShipLocation[cont2].Y += velocidade;
        }

        casaLocation1.Y += velocidade * 0.7f;
        casaLocation2.Y += velocidade * 0.7f;
        casaLocation3.Y += velocidade * 0.7f;
        tiroLocation.Y -= 8;
        progresso++;

        if (progresso > 1000)
        {
            progresso = 0;
            for (cont2 = 1; cont2 <= numero; cont2++)
            {
                spaceShipLocation[cont2].Y = spaceShipLocationN[cont2].Y;
                spaceShipLocation[cont2].X = spaceShipLocationN[cont2].X;
            }
            casaLocation1.Y = -50.0f;
            casaLocation2.Y = -600.0f;
            casaLocation3.Y = -400.0f;
        }
        label1.Text = (1000 - progresso).ToString();
        KeyboardState keyState = Keyboard.GetState();
        if (keyState.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.Up)) {
spaceShipLocation2[ncontrolado].Y -= velocidade * 2; }
        if (keyState.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.Down)) {
spaceShipLocation2[ncontrolado].Y += velocidade * 2; }
        if (keyState.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.Left)) {
spaceShipLocation2[ncontrolado].X -= velocidade * 2; }
        if (keyState.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.Right)) {
spaceShipLocation2[ncontrolado].X += velocidade * 2; }
        if (keyState.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.Space))
        {
            tiroLocation.X = spaceShipLocation2[ncontrolado].X + 17;
            tiroLocation.Y = spaceShipLocation2[ncontrolado].Y + 10;

        }
        for (cont3 = 1; cont3 <= numero; cont3++)
        {

            if (((spaceShipLocation[cont3].X - 30 <
spaceShipLocation2[ncontrolado].X) && (spaceShipLocation[cont3].X + 60 >
spaceShipLocation2[ncontrolado].X) && (spaceShipLocation[cont3].Y - 10 <
spaceShipLocation2[ncontrolado].Y) && (spaceShipLocation[cont3].Y + 30 >
spaceShipLocation2[ncontrolado].Y)) || spaceShipLocation2[ncontrolado].X < 80
|| spaceShipLocation2[ncontrolado].X > 410)
            {
                tiroLocation.X += 10000;
                _curFrame = 0;
                temploexplosao = 20;
                posicaoexplosao.X = spaceShipLocation2[ncontrolado].X - 20;
```

```
                posicaoexplosao.Y = spaceShipLocation2[ncontrolado].Y - 20;
                progresso = 1000;
                spaceShipLocation2[ncontrolado].Y =
spaceShipLocationN2[ncontrolado].Y;
                spaceShipLocation2[ncontrolado].X =
spaceShipLocationN2[ncontrolado].X;
            }
        }
        smallView.Focus();

        KeyboardState kbState = Keyboard.GetState();
        if (kbState.IsKeyDown(Microsoft.Xna.Framework.Input.Keys.Escape))
            Application.Exit();
    }

    [StructLayout(LayoutKind.Sequential)]
    public struct Message
    {
        public IntPtr hWnd;
        public Int32 msg;
        public IntPtr wParam;
        public IntPtr lParam;
        public uint time;
        public System.Drawing.Point p;
    }

    [System.Security.SuppressUnmanagedCodeSecurity]
    [DllImport("User32.dll", CharSet = CharSet.Auto)]
    public static extern bool PeekMessage(out Message msg, IntPtr hWnd, uint
messageFilterMin, uint messageFilterMax, uint flags);

    void Application_Idle(object sender, EventArgs e)
    {
        while (AppStillIdle)
        {
            long newCount = PerformanceCounter.QueryPerformanceCounter();
            long elapsedCount = newCount - lastTimeCount;
            double elapsedSeconds = (double)elapsedCount /
PerformanceCounter.QueryPerformanceFrequency();
            lastTimeCount = newCount;

            Update((float)elapsedSeconds);
            Draw();
        }
    }

    protected bool AppStillIdle
    {
        get
        {
            Message msg;
            return !PeekMessage(out msg, IntPtr.Zero, 0, 0, 0);
        }
    }

    private void someDialog_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Jogo paralizado!");
        lastTimeCount = PerformanceCounter.QueryPerformanceCounter();
    }
```

```
    private void XnaForm_Load(object sender, EventArgs e)
    {
        string line;
        string arquivo = "temp.pl";
        StreamReader valor2 = new StreamReader(arquivo);
        line = valor2.ReadLine();
        richTextBox1.Text = "";
        while (line != null)
        {
            richTextBox1.Text = richTextBox1.Text + line + "\n";
            line = valor2.ReadLine();
        }
        valor2.Close();
    }

    private void someDialog_Click_1(object sender, EventArgs e)
    {
        MessageBox.Show("    Jogo paralizado!    ", "Mensagem do Sistema");
        lastTimeCount = PerformanceCounter.QueryPerformanceCounter();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        numero++;
        spaceShipTexture[numero] = contentMgr.Load<Texture2D>("barco");
        spaceShipLocation[numero] = new Vector2(smallView.Width / 2,
smallView.Height);
    }

    private void button3_Click_1(object sender, EventArgs e)
    {
        Application.Restart();
    }


    private void button2_Click_1(object sender, EventArgs e)
    {
        string line;
        string arquivo = "exemplos/simples.pl";
        StreamReader valor2 = new StreamReader(arquivo);
        line = valor2.ReadLine();
        richTextBox1.Text = "";
        while (line != null)
        {
            richTextBox1.Text = richTextBox1.Text + line + "\n";
            line = valor2.ReadLine();
        }
        valor2.Close();
    }

    private void button4_Click_1(object sender, EventArgs e)
    {
        StreamWriter valor = new StreamWriter("temp.pl", false,
Encoding.ASCII);
        valor.Write(richTextBox1.Text);
        valor.Close();
        MessageBox.Show("Arquivo Salvo com Sucesso.", "Mensagem");
    }
```

```
    private void button6_Click_1(object sender, EventArgs e)
    {
        FileInfo arquivo2 = new FileInfo("Objeto_4.cs"); arquivo2.Delete();
        FileInfo arquivo1 = new FileInfo("Controle_2.cs"); arquivo1.Delete();
        PrologInterface sharp = new PrologInterface();
        SymbolTerm file =
SymbolTerm.MakeSymbol(Path.GetFileNameWithoutExtension("temp.pl"));
        Predicate member = new Compile_1(file, new ReturnCs(sharp));
        sharp.SetPredicate(member);
        bool b = sharp.Call();
        MessageBox.Show("Arquivos .cs criados com sucesso.", "Mensagem");
    }

    private void button7_Click_1(object sender, EventArgs e)
    {
        MessageBox.Show("DLL criada com sucesso.", "Mensagem");

System.Diagnostics.Process.Start("C:/Windows/Microsoft.NET/Framework/v2.0.507
27/csc.exe", "/target:library /out:tempo/prolog_objetox.dll
/reference:Psharp.dll Objeto_4.cs");

System.Diagnostics.Process.Start("C:/Windows/Microsoft.NET/Framework/v2.0.507
27/csc.exe", "/target:library /out:tempo/prolog_controlex.dll
/reference:Psharp.dll controle_2.cs");
    }

    private void XnaForm_FormClosed(object sender, FormClosedEventArgs e)
    {
        Application.Exit();
    }

    private void button5_Click_2(object sender, EventArgs e)
    {
        string line;
        string arquivo = "exemplos/recursao.pl";
        StreamReader valor2 = new StreamReader(arquivo);
        line = valor2.ReadLine();
        richTextBox1.Text = "";
        while (line != null)
        {
            richTextBox1.Text = richTextBox1.Text + line + "\n";
            line = valor2.ReadLine();
        }
        valor2.Close();
    }

    private void button1_Click_1(object sender, EventArgs e)
    {
        string line;
        string arquivo = "exemplos/operacao.pl";
        StreamReader valor2 = new StreamReader(arquivo);
        line = valor2.ReadLine();
        richTextBox1.Text = "";
        while (line != null)
        {
            richTextBox1.Text = richTextBox1.Text + line + "\n";
            line = valor2.ReadLine();
        }
        valor2.Close();
    }
```

```
    private void pNAGameEngineToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        pnaengine s = new pnaengine();
        s.ShowDialog();
    }

    private void sairToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }

    private void sobreOGameToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        sobre s = new sobre();
        s.ShowDialog();
    }

    private void tutorialToolStripMenuItem_Click(object sender, EventArgs e)
    {
        tutorial s = new tutorial();
        s.ShowDialog();
    }

    private void códigoFonteToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        tabControl1.SelectTab(tabPage1);
    }

    private void vizualizaçãoToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        tabControl1.SelectTab(tabPage2);
    }

}
}
```

Referências Bibliográficas

BANBARA, M.; TAMURA, N. Translating a Linear Logic Programming Language into Java. ICLP'99 Workshop, 1999.

BESSA, L., 2008. Professores, assim não. Disponível em: http://www.jornaldenegocios.pt/index.php?template=SHOWNEWS&id=312955 [Acesso em 21 de julho de 2008]

CASANOVA, Marco A.; GIORNO, Fernando A. C.; FURTADO, Antonio L.. Programação em Lógica e a Linguagem PROLOG. 2006.

CONWAY, Matthew J. Alice: Easy-to-Learn 3D Scripting for Novices. Faculty of the School of Engineering and Applied Science at the University of Virginia. Tese de Doutorado, 1997

COOK, Jonathan. P#: Using PROLOG within the .NET Framework. Laboratory for Foundations of Computer Science, University of Edinburgh. 2003.

COOK, Jonathan. P# Manual (version 1.1.3). Manual do programa. 2003.

COVINGTON, M. Natural Language Processing for Prolog Programmers. Prentice-Hall, 1994.

CREATORS CLUB. XNA definition. Disponível em:

http://creators.xna.com/ . Acessado em Agosto/2008.

HARTNESS, Ken. Robocode: using games to teach artificial intelligence. Journal of Computing Sciences in Colleges archive. Volume 19. 2004

LEBBINK, Henk-Jan. WITTEMAN, Cilia. MEYER, John-Jules. A Dialogue *Game* Approach to Multi-Agent System Programming. Belgium-Netherlands Conference on Artificial Intelligence, 2004

LIBERTY, J. Programming C#. O'Reilly, 2001.

NETO H. B. & BORGESI, S. M. C., 2006. *O papel da informática educativa no Desenvolvimento do raciocínio lógico*. FACED/UFC.

NETO H. B. & SANTANA, J. R., 2001. *Fundamentos Epistemológicos da Teoria de Fedathi no Ensino de Matemática*. FACED/UFC.

PALAZZO, Luiz A. M. Introdução à Programação PROLOG. Editora da Universidade Católica de Pelotas, 1997

SILVA, Flávio Soares Corrêa da; MELO, Ana Cristina Vieira de. Modelos Clássicos de Computação. Thomson, 2006.

SILVA, Flávio Soares Corrêa da; SILVA, Filipe Corrêa Lima da. A *Game*-based Animation Tool to Support the Teaching of Formal Reasoning. SB*Game*s 2006.

SILVA, Flávio Soares Corrêa da; SILVA, Filipe Corrêa Lima da. Uma ferramenta para o ensino de inteligência artificial usando jogos de computador. Dissertação de Mestrado. USP, 2007.