

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

Alex F. V. Machado Esteban W. Clua *Flavio S. C. da Silva Marcelo da S. Corrêa

Universidade Federal Fluminense (UFF) *Universidade de São Paulo (USP)

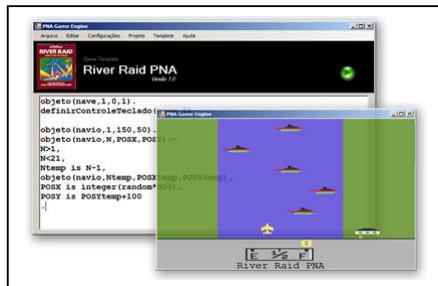


Fig. 1: Interface do PNA Game Engine com a visualização de uma instância do jogo do template River Raid.

Abstract

This work presents a novel *game* oriented tool which can motivate the teaching of declarative languages programming (Fig. 1). It uses the P# compiler [COOK 2003, COOK 2, 2003] as a middleware between 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.

Keywords: Prolog, XNA, teaching

1. Introdução

Conforme apresentado em [SILVA & MELO 2006], 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 melhor caracterizados dessa forma. Já a programação declarativa se baseia no conceito de reapresentação de teorias formais, e muitos problemas difíceis de serem resolvidos imperativamente em linguagens como VB, C++ e Java podem se tornar simples com o uso do paradigma declarativo. Não há muitos esforços registrados na literatura no que se refere a compatibilizar sistemas de programação declarativa com plataformas de desenvolvimento modernas, 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 *Visual Studio*, 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 consequência, o aprendizado de técnicas de programação declarativa pode se tornar mais árduo que o aprendizado de programação imperativa.

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. 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.

Neste trabalho é proposto e documentado um sistema, denominado *PNA Game Engine*, que utiliza a programação em lógica como entrada para a definição do comportamento inicial de um jogo desenvolvido 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.

Esse sistema disponibiliza *templates* para jogos (com modelos 2D e 3D prontos, mas com as classes principais sem instanciação) que podem ser acessados através de uma API batizada de PNA (*Pna is Not an Acronym*) pela interface em PROLOG.

O artigo está organizado da seguinte forma: a seção 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 nesta seção faz-se

uma revisão sobre uma ferramenta similar à proposta do presente trabalho. Na seção 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#. Na seção 4 demonstra-se a aplicação dessas tecnologias no processo de desenvolvimento do *PNA Game Engine* e seus principais componentes e funcionalidades. Nas sessões 5 e 6 são apresentadas algumas conclusões e trabalhos futuros.

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 lingüística 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 a fundamentada em uma modelagem baseada em inferências lógicas.

Nesta seção 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 docentes da área de inteligência artificial, o ensino desta linguagem possui diversas outras vantagens [PALAZZO 1997], tais como:

- Aprendizado mais fácil e natural em comparação com as linguagens imperativas;
- Permite a implementação de sistemas reflexivos;
- Libera o aluno dos problemas associados ao controle de suas rotinas, permitindo-lhe concentrar-se nos aspectos lógicos da situação a representar.
- Exercita fundamentos de linguagens de especificação.
- Facilita a implementação de regras gramaticais de gramáticas livres de contexto.

- Permite o estudo avançado de recursividade através de mecanismos simples como *backtracking*, *cut* e *fail*.

No presente trabalho, destaca-se a contribuição do uso e desenvolvimento de jogos, a partir de técnicas de programação em lógica, para o processo de aprendizagem de PROLOG e programação declarativa.

2.2 Prolog Aplicado a Jogos

O uso de linguagens imperativas muitas vezes inibe o programador de usufruir das vantagens que a programação declarativa pode oferecer ao desenvolvimento de aplicações.

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 da lógica, sistemas especialistas e criação de diálogo.

2.2.1 Controle da lógica

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 linguagens para a programação de computadores.

Trazendo o conceito de lógica e controle para os jogos podemos definir, respectivamente, o ambiente e o comportamento dos elementos. 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 IA de ações dos inimigos usando a estrutura de controle que analisará essa base lógica.

2.2.2 Sistemas especialistas

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.

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 sem-jogador) que pode variar com as ações do *player* (personagem controlado pelo jogador).

2.2.3 Criação de Diálogo

A maior parte dos jogos eletrônicos, e principalmente os do gênero RPG (*Role Playing Game*), necessitam de diálogo entre os NPCs e o *player*. 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.

A criação de diálogos entre *players* e personagens do mundo virtual representa uma das principais áreas de estudo da aplicação de PROLOG para jogos [LEBBINK, WITTEMAN & MEYER 2004].

2.3 Ferramentas de Programação Gráfica para o Ensino de PROLOG

Em [SILVA & SILVA 2006] 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.

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.

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.

3.1 Tecnologias Microsoft

Optou-se pela plataforma *Microsoft Visual Studio .NET* como principal ferramenta de desenvolvimento, por ser uma ferramenta RAD (*Rapid Application Development*) 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 (mas com determinadas configurações extras, herdadas do C++) e por ser a única a trabalhar com XNA.

O XNA (*XNA is Not an Acronym*) *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.

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

O *PNA Game Engine* foi desenvolvido utilizando os formulários do *Visual Studio* e a linguagem XNA. 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 cooperar com a janela normal dos formulários do *Visual Studio*. Isto exige ao desenvolvedor criar 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 nele; ou,

- Criar um novo projeto de jogo do *windows* e referenciar uma diretiva para uso de formulários (opção adotada neste artigo).

As principais etapas para a integração do formulário no *Windows Game Project* no *Visual Studio* são:

1. Preparação do ambiente a partir de um novo *Windows Game*, substituindo a classe principal por um formulário.
2. Criação de um componente para exibição do conteúdo do XNA através de um *User Control*.
3. Criação de um procedimento para atualizar a cada momento o gráfico contido no *User Control*.
4. Criação de um procedimento no 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. Criação da função *RenderToTexture()* para configurar o dispositivo gráfico para usar o *render target* e o *depth buffer*, de forma a esvaziar os *buffers*, desenhar a tela e adquirir o resultado em forma de textura.
6. 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 loop* do XNA no evento *OnIdle()* do formulário. Neste procedimento de repetição é incorporado as funções de *Update()* e *Draw()*, nesta ordem.

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# ele dispõe de diversas classes para o desenvolvedor 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 última escreve-se esse resultado na tela.

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], muito estudo existe neste sentido [SILVA e SILVA 2006]. O *PNA Game Engine* é uma ferramenta que preenche esta lacuna (Fig. 2).

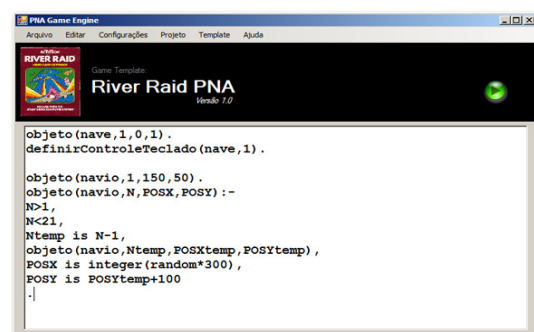


Fig. 2 - Interface do *PNA Game Engine*

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* compostos de bibliotecas de classes, modelos 3D de objetos, modelos 2D de *sprites* e elementos sonoros. O *template* completo desenvolvido nesta primeira versão do programa foi o clássico *River Raid* (Fig. 3), que é um jogo do console Atari, no qual

um avião sobrevoando um rio deve destruir e ultrapassar os navios e helicópteros que são seus obstáculos.



Fig. 3 - Jogo River Raid

Nesta versão o programador está limitado a usar somente os *games templates* pré-definidos, excluindo a possibilidade de criação de um jogo mais personalizado.

4.1 O Engine PNA

O *engine PNA* é o *software* que vai integrar o *template* do jogo com o restante do sistema. Suas principais funcionalidades são (Fig. 4):

- Abrir arquivo PNA – permite abrir um novo código em sua área de edição.
- Definir *template* – permite trocar o *template* atual. Isto 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 seu 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.
- 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.
- Arquivo existente - permite abrir um arquivo PNA desenvolvido pelo usuário.
- Salvar Arquivo PNA – salva o *script* desenvolvido pelo usuário. Somente depois de salvo 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.
- 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 River Raid*.

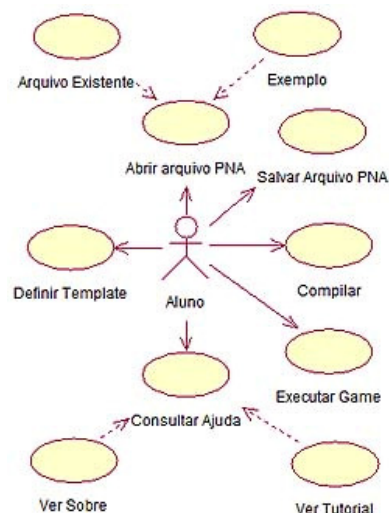


Fig. 4 - Diagrama UML de Caso-de-Uso do PNA Game Engine

4.2 Componentes do Sistema

Os principais componentes do *PNA Game Engine* são (Fig. 5):

- 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 *runtime*.
- 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.

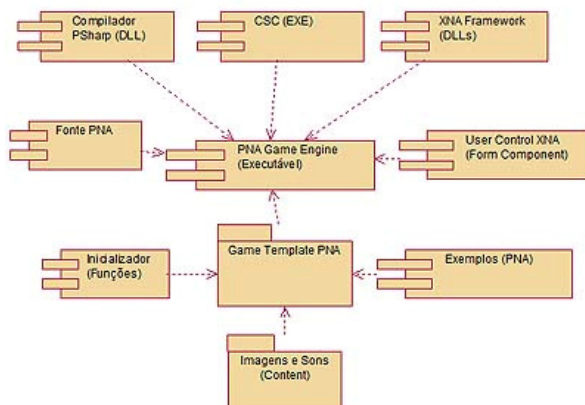


Fig. 5 - Diagrama de componentes do sistema desenvolvido.

4.3 O Código PNA para o *Template River Raid*

O PNA é uma mistura 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.

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), 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 20 navios e de 1 nave, e define o controle do teclado para a nave.

```
objeto(nave,1,0,0).
objeto(navio,1,150,50).
objeto(navio,2,50,150).
objeto(navio,3,250,250).
objeto(navio,4,150,350).
objeto(navio,5,150,450).
objeto(navio,6,250,550).
objeto(navio,7,50,650).
objeto(navio,8,150,750).
objeto(navio,9,150,850).
objeto(navio,10,100,950).
objeto(navio,11,150,1050).
objeto(navio,12,200,1150).
objeto(navio,13,200,1250).
objeto(navio,14,50,1350).
objeto(navio,15,150,1450).
objeto(navio,16,300,1550).
objeto(navio,17,300,1650).
objeto(navio,18,200,1750).
objeto(navio,19,300,1850).
objeto(navio,20,50,1950).
controle(nave,1).
```

Tab. 1 – Exemplo de um código em PNA

O exemplo da Tab. 2 gera um navio com uma posição pré-determinada e outros 19 com posições aleatórias no eixo x enfileirados no eixo y. Ele também insere e cria o controle para uma nave.

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

objeto(navio,1,150,50).
objeto(navio,N,POSX,POSY):-
N>1,
N<21,
Ntemp is N-1,
objeto(navio,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 navios no caminho à frente da nave (Fig. 6). 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. 6- Exemplo de código PNA para o jogo
River Raid

5. 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 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 desta 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].

6. Trabalhos Futuros

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

- 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], é de interesse dos autores documentar atividades práticas em sala de aula para uma avaliação mais criteriosa dos resultados.

Referências Bibliográficas

- BANBARA, M.; TAMURA, N. Translating a Linear Logic Programming Language into Java. ICLP'99 Workshop, 1999.
- CASANOVA, Marco A.; GIORNO, Fernando A. C.; FURTADO, Antonio L.. Programação em Lógica e a Linguagem Prolog. 2006.
- 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.
- 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.
- PALAZZO, Luiz A. M. Introdução à Programação Prolog. Editora da Universidade Católica de Pelotas, 1997
- SILVA, Flávio Soares Corrêa da; SILVA, Filipe Corrêa Lima da. A *Game*-based Animation Tool to Support the Teaching of Formal Reasoning. SBGames 2006.
- SILVA, Flávio Soares Corrêa da; MELO, Ana Cristina Vieira de. Modelos Clássicos de Computação. Thomson, 2006.