GÊNESIS BARROS CAMPOS

ESPECIFICAÇÃO DO COMPORTAMENTO DE AGENTES VIRTUAIS INTELIGENTES POR DEMONSTRAÇÃO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA

MINAS GERAIS - BRASIL

2009

GÊNESIS BAROS CAMPOS

ESPECIFICAÇÃO DO COMPORTAMENTO DE AGENTES VIRTUAIS INTELIGENTES POR DEMOSTRAÇÃO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 08 de julho de 2009.

Alcione de Paiva oliveira (Co-Orientadora)	José Luis Braga (Co-Orientador)
André Gustavo dos Santos	Newton José Vieira

Vladimir Oliveira Di Lorio
(Orientador)

Agradecimentos

Agradeço...

à Deus, em primeiro lugar;

aos meus pais, irmãos e familiares pelo incentivo, apoio, admiração, críticas e cobranças;

ao meu orientador, Vladimir, pelo direcionamento, paciência, amizade e confiança no sucesso do trabalho;

aos amigos da graduação, por despertar o meu interesse no mestrado e mostrar o caminho para alcançar tal objetivo;

à Universidade Federal de Viçosa, em especial ao professores e funcionários do DPI por todas as oportunidades;

aos amigos, novos e velhos, pelos momentos de distração que foram cruciais para o término deste trabalho;

à CAPES e FAPEMIG pelo apoio financeiro.

Esta conquista é de todos vocês!

Muito Obrigado.

Sumário

Lista de Siglas

DPI-UFV	Departamento de Informática da Universidade Federal de Viçosa
EUP	Programação por usuários finais (do inglês *End-User Programming*)
FPSG	jogo de tiro de primeira pessoa (do inglês *First Person Shooter Game*)
IA	Inteligência Artificial
IVA	Agente Virtual Inteligente (do inglês *Intelligent Virtual Agent*)
IVE	Ambiente Virtual Inteligente (do inglês *Intelligent Virtual Environment*)
PBD	Programação por Demonstração (do inglês *Programming by Demostration*)
UT	Unreal Tournament
VE	Ambiente Virtual (do inglês *Virtual Environment*)

Lista de Figuras

Lista de Tabelas

Resumo

CAMPOS, Gênesis Barros, M.Sc., Universidade Federal de Viçosa, Julho de 2009. **Especificação do comportamento de agentes virtuais inteligentes por demonstração**. Orientador: Vladimir Oliveira Di Iorio. Co-orientadores: Alcione de Paiva Oliveira e José Luis Braga.

A possibilidade de testar hipóteses e experimentar a realidade sem as despesas e perigos do mundo real fez com que a utilização de ambientes virtuais atraísse a atenção de diversos profissionais. Tão importante quanto interagir em ambientes realistas é lidar com personagens que transmitam a sensação de vida, seres "inteligentes". A definição da camada de seleção de ações é determinante para o realismo do personagem e pode ser identificada a partir de exemplos dados pelo usuário do comportamento que deve ser executado pelo agente. Diante da não exploração de tal característica pelas abordagens de programação por usuários finais existentes, este trabalho apresenta uma nova proposta para a especificação de agentes virtuais inteligentes. Incorporando pontos fortes dos paradigmas de programação paramétrico, descritivo e imitativo, a solução proposta facilita o desenvolvimento de agentes àqueles que gostariam de realizá-lo e não o fazem por não possuírem conhecimentos de programação. Um sistema foi desenvolvido para validar a possibilidade de se especificar o comportamento de agentes utilizando a nova abordagem de programação para usuários finais. O ambiente realista desenvolvido incorpora os recursos visuais dos ambientes virtuais atuais, e, por si só, motiva tanto desenvolvedores, quanto usuários. Os resultados alcançados neste grau de complexidade levam a crer que a proposta deste trabalho pode ser aplicada a um grande leque de situações, difundindo a utilização deste recurso

Abstract

CAMPOS, Gênesis Barros, M.Sc., Universidade Federal de Viçosa, July of 2009. **Specification of intelligent virtual agents behavior by demonstration**. Adviser: Vladimir Oliveira Di Iorio. Co-advisers: Alcione de Paiva Oliveira and José Luis Braga.

The possibility to test hypotheses and to experience reality without the expenses and dangers of the "real world" have attracted the attention of professionals to the use of virtual environments. As important as interacting in realistic environments is dealing with characters that convey a sense of living, "intelligent" beings. The definition of the layer of action selection is crucial to the realism of the character and it can be identified from the examples given by the users of what the agent must do. Faced with the existent end-user programming approaches not exploring this feature, this paper presents a new proposal for the specification of intelligent virtual agents. The proposed solution incorporates strengths of parametric, descriptive and imitative programming paradigms, and facilitates the development of agents to those who would like to do it but are not able because of lack of programming knowledge. A system was developed to validate the possibility of specifying the agents' behavior using the new end-user programming approach. The environment developed incorporates the realistic visuals of current virtual environments, and by itself, motivates both developers and users. The results achieved at this degree of complexity suggest that the new approach can be applied to a wide range of situations, and that it can spread the use of this resource.

1 Introdução

"*Computer science is no more about computers than astronomy is about telescopes.*"
Edsger Dijkstra

A programação é o meio de extrair todos os recursos disponibilizados por computadores. No entanto, programar, para a maioria das pessoas, é uma tarefa muito difícil. Nos primeiros computadores, era necessário um profundo conhecimento do hardware para explorar o potencial das máquinas, que tinham de ser reconfiguradas fisicamente a fim de mudar suas funcionalidades. Um avanço significativo veio com o advento dos softwares, que permitiam alterações nas tarefas executadas por um computador sem necessariamente exigir um profundo conhecimento de eletrônica. Este avanço permitiu que especialistas de outras áreas desenvolvessem soluções computacionais para seus problemas. O surgimento das linguagens de alto-nível, que aproximaram as sintaxes das primeiras linguagens de programação ao formalismo da linguagem matemática, possibilitou o acesso a programação a um número maior de pessoas. (HAGUE; ROBINSON, 2006)

No início dos anos 60, usuários de computadores eram programadores, ou tinham programas desenvolvidos especificamente para seus propósitos. Desta forma, o usuário sempre era versado em todas as operações necessárias para a realização das tarefas ao seu alcance. Com o advento dos computadores pessoais, o cenário mudou. Ao invés de utilizar programas específicos às suas necessidades, usuários passaram a interagir com aplicações genéricas, e estas definitivamente não atendem às necessidades de todos (CYPHER *et al.*, 1993).

Dar a oportunidade ao usuário de customizar suas aplicações, sem precisar investir tempo e esforço no aprendizado de uma linguagem de programação e posteriormente na estrutura de algum sistema, passou a ser o foco de pesquisas na área de Interação Humano Computador. Esta sub-área, antes conhecida como Psicologia da Programação, hoje recebe o nome de Programação por Usuários Finais[1] (EUP) (MYERS; KO; BURNETT, 2006; CYPHER *et al.*, 1993; NARDI, 1993; LIEBERMAN, 2001). Ao contrário da abordagem tradicional, que tenta fazer com que o usuário aprenda a representação aceita pelo computador, que demanda muito tempo e não atrai a atenção de muitos, a proposta da EUP é tornar o sistema mais próximo do usuário, aceitando representações que não demandem tanta abstração. Os usuários finais, neste contexto, não são computacionalmente leigos, nem programadores em processo de aprendizado. Usuários finais são especialistas em suas respectivas áreas – químicos, físicos, bibliotecários, contadores, entre tantos –, que têm necessidades computacionais e querem fazer uso dos computadores, mas que não estão interessados em tornarem-se programadores profissionais (NARDI, 1993).

O sucesso na utilização de ambientes virtuais[2] (VEs) para pesquisas de Inteligência

[1] *End-User Programming*
[2] *Virtual Environments*

Artificial (IA) e Robótica chamou a atenção de pesquisadores de diversas áreas. A diversidade de possibilidades oferecidas por VEs – simuladores, jogos, entre outros –, sem os riscos e custos inerentes de atividades no mundo real (ANGROS JR. *et al.*, 2002; ANASTASSAKIS; RITCHINGS; PANAYIOTOPOULOS, 2001), cria uma ótima opção para testar hipóteses e visualizar ideias. Mais do que simplesmente olhar para símbolos, fórmulas e conceitos, trabalhar com esses ambientes permite ao usuário experimentar a realidade por trás destas definições.

O interesse de especialistas em diversas áreas em empregar os recursos oferecidos pelos VEs fez com que surgissem ambientes virtuais genéricos, adaptáveis de acordo com as necessidades do usuário e que, consequentemente, atenderiam a várias demandas. A difusão da utilização atraiu os usuários finais à produção dos sistemas e para facilitar o trabalho de criar uma representação de realidade, passou-se, então, a incorporar mecanismos de EUP às aplicações de desenvolvimento de VEs.

O interesse em simplificar o desenvolvimento de ambientes virtuais por usuários finais deu início a uma busca por alternativas às linguagens existentes para a execução da tarefa. Sistemas em linguagens textuais convencionais como MR (SHAW *et al.*, 1993) e Cyberspace (DONOVAN, 1993 apud BIMBO; VICARIO, 1995) eram barreiras para a utilização deste recurso. Segundo Smith, Cypher e Tesler (2000), o problema está na grande lacuna existente entre a representação que o cérebro usa quando pensa em solução e a representação que o computador aceita. Converter experiências empíricas, naturais do comportamento humano, em condicionais, demanda muita abstração (o quão distante se deve começar a frear um automóvel quando nota-se a aproximação de uma pessoa a faixa de pedestres?).

Buscando romper a barreira entre usuários finais e a especificação de sistemas de simulação, várias técnicas de EUP já foram utilizadas. Linguagens textuais de propósito específico, linguagens de programação visual, representações gráficas de fluxos de controle e linguagens baseadas em regras são algumas das abordagens já implementadas. Através da análise dos sistemas existentes, realizada neste trabalho, pôde-se avaliar melhor os pontos fortes e fracos da utilização de cada técnica em relação à fase do desenvolvimento de sistemas de simulação. Percebeu-se que linguagens de programação visual são excelentes para a definição gráfica do ambiente, linguagens baseadas em regras facilitam a especificação da navegação no VE e que linguagens de propósito específico obtiveram melhores resultados que as linguagens convencionais, mas muito aquém das alternativas já citadas.

Os apelos visuais dos ambientes virtuais, por meio de gráficos super realistas, são importantes para que usuário tenha a sensação de estar diante de um objeto ou ambiente real. Entretanto, esses recursos tornam-se pouco interessantes na falta de algo dinâmico e comportamental (AYLETT; CAVAZZA, 2001). Interagir com personagens com atitudes reais dá mais vida a animações, jogos, simuladores e outros VEs, que, quando povoados por tais, passam a ser referenciados como ambientes virtuais inteligentes.

Agentes Virtuais Inteligentes[3] (IVA), também referenciados na literatura como Personagens Autônomos[4] ou Personagens Sintéticos[5], são agentes autônomos com representação física, parte de algum ambiente, e têm sido muito utilizados para dar mais realismo a VEs (REYNOLDS, 1999; BROM, 2006; AYLETT; CAVAZZA, 2001; FRANKLIN; GRAESSER, 1997). IVAs podem ser atores ou tutores em softwares educativos, inimigos e/ou aliados em um jogo ou mesmo personagens em animações computadorizadas (BROM, 2006).

Com o objetivo de permitir que usuários finais pudessem desenvolver VEs mais realistas, técnicas de EUP passaram a ser utilizadas na especificação de IVAs.

1.1 O problema e sua importância

Segundo Brom (2006), a implementação de IVAs está relacionada a (1) definição do avatar (representação do aspecto físico), (2) especificação do comportamento, (3) modelagem das emoções e (4) interação com o usuário, com outros IVAs e com o VE.

As técnicas de EUP exploradas para o desenvolvimento de VEs também foram aplicadas para a facilitar a implementação de IVAs. O estudo da utilização das abordagens existentes em cada fase da implementação, realizado neste trabalho, possibilitou a identificação dos pontos fortes e fracos dos mecanismos. A especificação do comportamento de IVAs apresenta uma particularidade ainda não explorada pelos sistemas existentes. Depois da construção do ambiente, definição do avatar e das formas de interação dos IVAs entre si e com o VE, e representação das emoções do IVA, basta que o desenvolvedor especifique as estratégias de comportamento dos personagens. Este momento pode ser ilustrado como um jogo, onde resta somente que o jogador comande o seu personagem para atingir seu objetivo final. No caso do desenvolvimento de VEs, restaria especificar a estratégia de um personagem autônomo. Desta forma, fica claro que a especificação do comportamento de IVAs pode ser feita utilizando exemplos demonstrados pelo usuário.

[3] *Intelligent Virtual Agents*
[4] *Autonomous Characters*
[5] *Synthetic Characters*

Reynolds (1999) chama a atenção para os diversos significados do termo *comportamento*. Uma ação instintiva, uma sequência óbvia de ações em um simples sistema mecânico, uma decisão complexa em um ambiente caótico e até uma animação em alguma aplicação multimídia são descritos como comportamentos. Para o melhor entendimento, o autor segmenta o conceito em três camadas: (1) seleção de ações, (2) pilotagem e (3) locomoção.

O nível (1) *Seleção de Ações* trata do objetivo do IVA. Caracteriza-se pela decisão do QUE deve ser feito a partir de uma mudança no ambiente, no estado do agente ou mesmo nos objetivos estabelecidos. (2) *Pilotagem*, por sua vez, determina COMO o agente realizará a tarefa. É neste nível que se traçam os pontos intermediários, sub-objetivos, e quais ações simples devem ser executadas para a concretização das atividades por completo. Estes dois níveis juntos definem a estratégia do comportamento do IVA. A camada (3) *Locomoção* é a responsável pela parte física, a representação mecânica da ação é controlada neste nível. Este trabalho concentra-se na camada de seleção de ações. Entende-se, neste, especificação do comportamento como a determinação das ações a serem executadas pelo IVA.

Trabalhar com o comportamento de IVAs facilita a incorporação de mecanismos de inteligência artificial aos VEs especificados por usuários finais. Embora a princípio possa parecer excesso de zelo prezar por IA para usuários finais, esta demanda já foi identificada e explorada por Repenning (2006).

Os conceitos aplicados na facilitação da especificação do comportamento de IVAs, além de agregar valor a definição de VEs mais interativos podem também ser transferidos para o mundo real. A programação de máquinas industriais ou mesmo de robôs domésticos, são exemplos claros de atividades que podem ser beneficiadas pela alternativa proposta neste trabalho.

1.2 Objetivos

O objetivo deste trabalho é facilitar o acesso de usuários finais ao desenvolvimento de ambientes virtuais através da proposição de uma nova abordagem em EUP para a especificação do comportamento de IVAs. Desta forma, pretende-se aumentar a quantidade de pessoas que possam vir a fazer uso deste recurso, e, como consequência, possibilitar a utilização desta tecnologia como ferramenta de aprendizado.

A proposta foi desenvolvida a partir da análise dos pontos fortes e fracos das técnicas

de EUP utilizadas para a execução da tarefa de especificação do comportamento de agentes em sistemas já existentes e do levantamento de outras abordagens, ainda não aplicadas neste contexto, que poderiam ser aproveitadas.

Um sistema foi desenvolvido para validar a possibilidade de se especificar o comportamento de agentes virtuais inteligentes utilizando a nova abordagem de programação para usuários finais proposta no trabalho. A ferramenta consiste basicamente em um ambiente virtual, no qual é inserido um avatar controlado pelo usuário através da interface da aplicação. Atuando via interface, todos os comandos executados são armazenados e a partir deste registro é gerado um IVA, que comporta-se de forma semelhante à demonstrada. Embora o ambiente tenha sido desenvolvido apenas para homologação da nova proposta, não são necessárias muitas adaptações para o seu aproveitamento em outras áreas de pesquisa, ou mesmo em aplicações comerciais. Esta plataforma permitiu ilustrar a discussão dos benefícios da nova solução e compará-la qualitativamente com as abordagens já existentes.

Para a realização da validação, foi solicitado a usuários que atuassem em duas sessões através do ambiente de testes. Na primeira, não lhes foi informado que seu jogo seria gravado. Na sessão seguinte, o propósito da aplicação e as circunstâncias "críticas" da gravação foram apresentados. Tal metodologia foi aplicada esperando que após tomarem ciência da necessidade de demonstrar o comportamento em cada uma das situações pré-determinadas estes buscariam vivenciá-las.

1.3 Organização deste documento

Este trabalho está organizado da seguinte maneira: no Capítulo 2, são apresentados os principais conceitos referentes ao desenvolvimento de VEs e IVAs, e as técnicas de EUP estudadas. O Capítulo 3 apresenta a abordagem de EUP proposta neste trabalho e uma discussão de sua aplicabilidade à tarefa de especificação da estratégia de comportamento de IVAs. No Capítulo 4, é apresentado o ambiente de testes desenvolvido para a validação da proposta do capítulo anterior, e os componentes por ele utilizados. A avaliação e validação da proposta são apresentadas no Capítulo 5, a partir de um estudo de caso realizado no ambiente desenvolvido. Finalmente, o Capítulo 6 apresenta as conclusões e algumas propostas de trabalhos futuros.

2 Referencial Teórico

"If I have seen further it is by standing on the shoulders of giants."

Isaac Newton

Este capítulo apresenta as áreas de pesquisa estudadas para a realização deste trabalho: ambientes virtuais inteligentes e programação por usuários finais. São apresentados também sistemas que implementam a interseção entre essas áreas e uma discussão sobre os problemas e desafios encontrados a partir do estudo dessas aplicações.

2.1 Ambientes Virtuais Inteligentes

A maior disponibilização de capacidade de processamento para a renderização de imagens permitiu que ambientes virtuais explorassem recursos de Realidade Virtual e se tornassem mais interessantes ao usuário. A utilização de técnicas de modelagem de ambientes (geométricas, matriciais e topológicas), visualização 3D (OpenGL[1] , DirectX[2] e Java3D[3]), navegação (perspectivas em primeira e terceira pessoa) e interação fizeram os VEs muito mais realistas.

Todavia, por mais realistas que possam parecer, ambientes estáticos e sem atrativos comportamentais dificultam a sensação de imersão, e consequentemente atraem menos interesse. Ainda que animações possam criar algum dinamismo, o fato de serem pré-estabelecidas vai contra a ideia de interações livres, e só cativam o usuário por um certo período de tempo. (AYLETT; CAVAZZA, 2001)

Assim como permitiu a exploração de um alto grau de realismo virtual, o aumento da capacidade de processamento possibilitou a adição de uma camada de inteligência aos ambientes. A interseção destas tecnologias – VE, Realidade Virtual e IA– resoltou na área de pesquisa conhecida por Ambientes Virtuais Inteligentes [4] (IVEs) (AYLETT; LUCK, 2000).

Anastassakis, Ritchings e Panayiotopoulos (2001) definem um ambiente virtual inteligente como um ambiente virtual que simula um mundo, real ou imaginário, habitado por entidades inteligentes autônomas que se comportam de diversas maneiras. Os autores argumentam que tamanhas são as possibilidades de utilização de IVEs que a estrutura e os componentes destes só dependem da natureza das aplicação alvo e da imaginação de seus desenvolvedores.

IVEs são empregados em diversas áreas, principalmente relacionadas a simulações, entretenimento e educação (ANASTASSAKIS; RITCHINGS; PANAYIOTOPOULOS, 2001). Ambi-

[1] OpenGL http://www.opengl.org
[2] DirectX - http://msdn.microsoft.com/en-us/directx/default.aspx
[3] Java3D - http://java.sun.com/products/java-media/3D/
[4] *Intelligent Virtual Environments*

entes simulados (espaços urbanos, prédios, ruas, etc) auxiliam a produção arquitetônica, de engenharia civil, de controle de tráfego e populações, dentre outros (BRAUN; BODMANN; MUSSE, 2005) (WRIGHT *et al.*, 1998 apud AYLETT; LUCK, 2000). Modelos precisos de equipamentos reais (veículos, aeronaves, etc) não só permitem a execução de testes sem riscos e com custo reduzido, eles possibilitam também atingir resultados mais confiáveis devido às inúmeras situações da vida real que podem ser testadas. Além disso, IVEs definiram novos padrões para o entretenimento digital. Jogos com imensos mundos virtuais (World of Warcraft[5]) capazes de prender a atenção dos jogadores por dias, dramatizações interativas (onde o usuário é parte da estória) (RIEDL; SARETTO; YOUNG, 2003; MAGERKO *et al.*, 2004), e outras áreas da diversão onde imersão e verossimilhança são fatores fundamentais (GRAND; CLIFF, 1998; PAUSCH *et al.*, 1996). Finalmente, IVEs ainda contribuem na área da educação, contribuindo para a criação de tutores virtuais cada vez mais dinâmicos e menos artificiais, tornando o ensino mais estimulante e prazeroso (RICKEL; JOHNSON, 2003).

2.1.1 Agentes Virtuais Inteligentes

Agentes virtuais inteligentes é uma subclasse de Agentes. Portanto, antes de discutir o conceito de IVAs, é necessário definir o termo *agente*. Todavia, não existe uma definição universalmente aceita, nem sequer os atributos que poderiam classificar determinada entidade como um agente são, em suma, compartilhados entre os autores. Essa dificuldade se deve ao fato da definição de *agente* estar estreitamente condicionada ao domínio ao qual se aplica. Entretanto, as definições mais aceitas concordam que agentes são parte de algum ambiente e que devem possuir: autonomia, capacidade sensorial e motora sobre esse ambiente (FRANKLIN; GRAESSER, 1997; PRADA, 2005).

Neste trabalho, utiliza-se a definição de Franklin e Graesser (1997). Optou-se por esta, por ter sido proposta a partir de um levantamento das dez definições mais aceitas na literatura.

> *"An ... agent is a system situated within and a part of an environment that senses that environment and acts on it..."* (FRANKLIN; GRAESSER, 1997)

Agentes virtuais inteligentes são, portanto, agentes com representação física (PRADA, 2005). IVAs fazem a vez dos organismos vivos (humanos, animais, criaturas fictícias, etc)

[5]World of Warcraft (Blizzard Entertainment) - www.worldofwarcraft.com

em IVEs e devem ser críveis física e comportamentalmente (VOSINAKIS; PANAYIOTOPOULOS, 2005).

Um agente virtual inteligente deve ser realista. Sua aparência física deve ser tal que o usuário possa percebê-lo como real, principalmente em suas interações com o ambiente (i.e., o IVA ao segurar determinado objeto deve comportar-se de acordo com o tamanho e formato daquele objeto)(PRADA, 2005). O trabalho de Thalmann (apud PRADA, 2005) apresenta um estudo sobre a verossimilhança de IVAs humanos, que segundo Prada (2005) pode ser transferido para a representação física de outros IVAs (animais, por exemplo). Já a verossimilhança comportamental preza pela coerência das decisões tomadas pelo IVA, que devem estar de acordo com as expectativas do usuário. Esta característica engloba interações sociais e emoções (PRADA, 2005). Segundo Vosinakis e Panayiotopoulos (2005) uma forma de adicionar características mais críveis aos IVAs é fazê-los reproduzirem o comportamento e/ou funcionalidades das entidades que eles representam. Desta forma, um IVA humano ideal é aquele que aparenta comportar-se da forma que um ser humano se comportaria, aquele cujas ações parecem ter propósito e justificativa.

Como dito no Capítulo 1, a implementação de IVAs está relacionada a (1) definição do avatar (representação do aspecto físico), (2) especificação do comportamento, (3) modelagem das emoções e (4) interação com o usuário, com outros IVAs e com o VE (BROM, 2006). Existem trabalhos relacionados em todas as etapas: Torres e Boulanger (2003) propõem ANIMUS, um framework de desenvolvimento de IVAs que engloba todo o processo; Loyall *et al.* (2004), Bates, Loyall e Reilly (1991) apresentam um sistema para criação de IVAs que trata o comportamento, as emoções e as interações; entre outros (AYLETT; CAVAZZA, 2001; KASAP; MAGNENAT-THALMANN, 2007).

Um descrição mais detalhada sobre a especificação do comportamento de IVAs foi apresentada no Capítulo 1. Embora existam muitas questões relacionadas à definição do avatar, modelagem de emoções, interação com usuário, outros IVAs e com o VE, essas não serão discutidas aqui por não pertencerem ao escopo do trabalho. Mais detalhes sobre as outras etapas podem ser encontrados no trabalho de Aylett e Cavazza (2001).

2.2 Programação por usuários finais

A área de programação por usuários finais tem por objetivo permitir que aqueles que possuam alfabetização computacional suficiente para realizarem determinada tarefa no computador, sejam capazes de criar ou adaptar programas para atingir o mesmo objetivo,

não sendo necessário para isso aprender uma linguagem de programação.(MYERS; KO; BURNETT, 2006; CYPHER *et al.*, 1993; NARDI, 1993; LIEBERMAN, 2001)

Usuários finais são pessoas que utilizam o computador em sua vida diária, mas não são programadores ou especialistas em computação. Podem ser usuários experientes em determinadas aplicações, mas não dominam linguagens de programação de propósito geral. Finalmente, podem até programar ocasionalmente, mas esta não é a parte mais importante das atividades por eles exercidas.(MYERS; KO; BURNETT, 2006; REPENNING, 2006)

Com o intuito de permitir que usuários finais possam adaptar, estender, ou mesmo criar aplicações, algumas técnicas de EUP foram desenvolvidas. Embora nem todas envolvam uma atividade de programação propriamente dita, os resultados alcançados só poderiam ser atingidos até então programando-se (CYPHER *et al.*, 1993). As subseções seguintes apresentam os paradigmas de programação: paramétrico, descritivo e imitativo, propostos por Silva (2001). Em cada uma encontram-se as técnicas de EUP associadas.

2.2.1 Paradigma Paramétrico

A seleção e ativação de elementos em um quadro de distribuição caracteriza bem a tarefa de EUP neste paradigma. As técnicas que seguem o paradigma paramétrico oferecem ao usuário, através da arquitetura da aplicação, um conjunto básico de módulos de customização e uma forma de configurar esses módulos. Portanto, existe um conjunto fechado de possibilidades de personalização das tarefas sobre as quais as alternativas operam. (SILVA, 2001)

Configuração de parâmetros, personalização da interface e o uso de layouts são exemplos de técnicas que se enquadram neste paradigma. Cypher *et al.* (1993) encapsula os três mecanismos sob o termo *Preferências*.

Configuração de parâmetros

A configuração de parâmetros é a técnica mais utilizada seguindo o paradigma paramétrico. O mecanismo possibilita ao usuário ativar ou desativar opções que afetam o comportamento da aplicação fazendo uso somente de elementos da interface. Um exemplo comercial da utilização deste é a possibilidade de seleção de elementos para compor a barra de ferramentas de alguns editores de texto.(SILVA, 2001)

Personalização da Interface

Através da alteração da composição e/ou posição de elementos visuais ou léxicos presentes na aplicação, este mecanismo possibilita ao usuário personalizar a interface do sistema. Esta técnica permite, ainda, a adição de novos elementos para a representação de funções já existentes. Todavia, não é possível adicionar novas funcionalidades. A criação de botões personalizados para a barra de tarefa, função disponibilizada por alguns editores de texto, é um exemplo de recurso que implementa este mecanismo.(SILVA, 2001)

Uso de Layouts

O uso de layouts permite ao usuário agrupar conjuntos de parâmetros, denominá-los e empregá-los para customizar o modelo do documento como um todo. A diferenciação e o compartilhamento das alterações entre os usuários são particularidades que merecem destaque, uma vez que o mecanismo não se restringe a alterações globais na aplicação. A criação de estilos, para a formatação de títulos e referências bibliográficas, é um exemplo de funcionalidade que implementa esse mecanismo. (SILVA, 2001)

2.2.2 Paradigma Descritivo

A tarefa de EUP no paradigma descritivo pode ser caracterizada pela construção de um plano de ação por parte do usuário. A arquitetura das aplicações deve disponibilizar um ambiente para que o usuário especifique as sub-tarefas que compõem o plano de ação e um interpretador para a linguagem utilizada. Este é o paradigma que oferece maior possibilidade de extensão, graças ao poder das linguagens utilizadas. Todavia, os mecanismos que o implementam não são tão acessíveis ao usuário quanto os demais. (SILVA, 2001)

Linguagens de script, linguagens visuais de fluxo de controle e linguagens visuais baseadas em regras são exemplos de técnicas reunidas sob esse paradigma.

Linguagens de Script

Linguagens de script são linguagens textuais com menos recursos, muitas vezes, voltadas somente para o propósito de uma dada aplicação. As linguagens de script são comumente derivadas de linguagens de programação tradicionais, possuem sintaxe personalizada para tratar os objetos e ações do domínio e são interpretadas, permitindo assim

respostas imediatas ao usuário.

Linguagens Visuais de Fluxo de Controle

Neste mecanismo, são criadas representações visuais para as principais estruturas de programação como iterações, condicionais e chamadas de processos. Os ícones propostos seguem o padrão dos ícones utilizados nas aplicações, sendo portanto mais amigável ao usuário que linguagens textuais.

Linguagens Visuais baseadas em Regras

Linguagens de programação visuais baseadas em regras são aquelas nas quais o usuário determina as condições, e para cada uma destas, as ações associadas. São um sub-conjunto do mecanismo anterior, uma vez que só existe representação para a estrutura condicional.

2.2.3 Paradigma Imitativo

Nesse paradigma, a tarefa de EUP se caracteriza pela criação da réplica de um processo – uma sequência de ações realizada pelo usuário. Para isso, a arquitetura da aplicação deve disponibilizar mecanismos que permitam ao usuário definir qual sequência de ações ele deseja replicar, e quando ativar a execução da sequência registrada. Por poder replicar sequências que contenham praticamente todo o tipo de ação da aplicação (exceto talvez, ações relacionadas ao mecanismo de replicação que normalmente não podem ser replicadas) o usuário possui um amplo conjunto de possibilidades de personalização da aplicação. (SILVA, 2001)

As técnicas de gravação de macros e programação por demonstração são exemplos de mecanismos de EUP que se enquadram nesse paradigma.

Gravação de Macros

Muito utilizada para a automatização de tarefas repetitivas, essa técnica consiste do registro da sequência de ações realizadas pelo usuário após o acionamento de um gravador de instruções. Feito isso, os comandos gravados podem ser reproduzidos quando necessário. A gravação de macros armazena literalmente os eventos, assumindo como constantes as variáveis de contexto.

Programação por Demonstração

Tal qual a gravação de macros, a programação por demonstração[6] (PBD) faz uso das ações do usuário sobre o ambiente para automatizar determinada tarefa. Todavia, ao invés de registrar literalmente as operações ativadas pelo usuário, criam-se programas generalizados a partir dos exemplos, permitindo aplicá-los em situações diferentes daquela na qual foram realizados.

O programa generalizado é criado a partir de técnicas de inteligência artificial. Algumas implementações também utilizam dessas técnicas para realizar a inferência de comportamentos e minimizar a quantidade de ações a serem exemplificadas.

2.3 Técnicas de EUP para especificação do comportamento de IVAs

Apresenta-se, nesta seção, as técnicas de EUP já empregadas na especificação do comportamento de IVAs. Muitas vezes, a técnica não é utilizada somente para este propósito, sendo útil para a toda e qualquer extensão na aplicação.

2.3.1 Configuração de parâmetros

A criação de *bots* (abreviação de *ro**bots***) – também conhecidos como personagens não controlados por pessoas[7], ou simplesmente NPCs –, no Counter Strike[8], jogo de tiro de primeira pessoa[9] (FPSG), segue essa técnica (COLE; LOUIS; MILES, 2004).

Pode-se julgar o comportamento gerado a partir do número de parâmetros configuráveis e as combinações obtidas. Poucos parâmetros resultam em um baixo potencial de comportamentos, enquanto muitos, comprometem as chances do usuário prever com sucesso qual será o comportamento resultante.

[6] *Programming by Demonstration*
[7] *Non-Player Character*
[8] Counter Strike (Valve Corporation) - www.counter-strike.net
[9] *First Person Shooter Game*

2.3.2 Linguagens de Script

Esta técnica é utilizada em muitos jogos e simuladores como: Unreal Script, linguagem oferecida pelo Unreal Tournament[10] (UT) (COLE; LOUIS; MILES, 2004), um FPSG; Gamebots[11], que encapsula a anterior; e o muito conhecido ambiente Logo (KELLEHER; PAUSCH, 2005).

2.3.3 Linguagens visuais de fluxo de controle

O produto comercial voltado para a programação de robôs Lego Mindstorms[12], o ambiente de programação ToonTalk (KAHN, 1996) e os variantes do Logo: LogoBlocks e Leogo (KELLEHER; PAUSCH, 2005), são exemplos de aplicativos que utilizam essa técnica.

2.3.4 Linguagens visuais baseadas em regras

Embora não passem de condicionais, os mecanismos de adição de expressões a serem avaliadas e simplicidade de manipulação agradam a muitos usuários. O jogo de estratégia de exércitos Warcraft III[13] utiliza essa técnica (EL-NASR; SMITH, 2006).

A incorporação de mecanismos de programação por demonstração, em que o usuário utiliza os elementos visuais da aplicação para a criação dos condicionais é uma variante desta técnica. O fato das regras serem criadas a partir de situações e estados do jogo torna o processo muito mais intuitivo ao usuário. StageCast Creator (SMITH; CYPHER; TESLER, 2000) e KidSim (REPENNING, 2000) são ambientes que implementam essa variante.

2.3.5 Programação por demonstração

O trabalho de Bimbo e Vicario (1995) utiliza a programação por demonstração para determinar o comportamento de IVAs. Os autores criam um conjunto de situações nas quais o usuário deve agir. A partir de então é gerado um IVA que se comporta de forma semelhante à demonstrada pelo usuário.

McDaniel e Myers (1998) também fazem uso dessa abordagem. O sistema por eles proposto, Gamut, permite demonstração do que deve, ou não deve ser feito, através da

[10]Unreal Tournament (Epic Games) - www.unrealtournament3.com
[11]Gamebots (Open Source) - sourceforge.net/projects/gamebots
[12]Lego Mindstorms (Lego & MIT) - mindstorms.lego.com/
[13]Warcraft III (Blizzard Entertainment) - http://www.blizzard.com/us/war3/

manipulação dos objetos do VE. A proposta possibilita, também, destacar elementos do ambiente – ou, segundo os autores, dar dicas – que foram determinantes para os comportamentos demonstrados, evitando inferências errôneas das condições de cada regra.

2.4 Problemas e desafios

O estudo das técnicas de EUP utilizadas para a especificação do comportamento de IVAs revela particularidades que comprometem o desempenho, ou mesmo restringem o acesso dos usuários finais.

Embora poderoso do ponto de vista computacional – vide redes neurais, por exemplo – o emprego da configuração de parâmetros é restrito. O número de parâmetros que se pode esperar que o usuário esteja apto a entender e configurar é bastante pequeno e consequentemente compromete a implementação de comportamentos que atendam às suas necessidades (BARBOSA, 1999).

Linguagens de script, ainda que muito mais simples que as LPs convencionais, necessitam de um mínimo conhecimento de programação que muitas vezes falta ao usuário final. Outro problema levantado por Barbosa (1999) é que as aplicações que oferecem esta técnica geralmente têm de utilizar um novo modo de operação, ou um ambiente distinto para possibilitar extensões. Esta mudança, ainda segundo a autora, desorienta o usuário, que não sabe por onde começar, não sabe como interagir com a "nova aplicação".

O fato de criar ícones para a representação das estruturas básicas de programação proposto pelas linguagens de fluxo de controle utiliza da velha máxima: "uma imagem vale mais que mil palavras" para superar as dificuldades encontradas na assimilação da sintaxe de uma LP. Todavia, persiste a necessidade do entendimento da semântica da estrutura, a princípio isoladamente e posteriormente organizada em forma de programa. Tão importante como saber codificar, é conseguir prever o resultado da execução, e para isso, assim como nas linguagens de script, um mínimo de conhecimento de programação é necessário.

As linguagens visuais baseadas em regras agradam bastante aos usuários. As que incorporam mecanismos de programação por demonstração (SMITH, 2000; REPENNING, 2000; MCDANIEL; MYERS, 1998) conseguiram bons resultados até mesmo com crianças. Todavia, a tarefa de criação de regras é muito maçante. O trabalho de Coura *et al.* (2006) minimiza a quantidade de regras, mas ainda assim, é necessário criá-las basicamente para cada interação do agente com o meio, tornando a técnica inadequada para a especifica-

ção de comportamentos mais elaborados ou quando o agente precisa interagir com um ambiente complexo.

Dentre as técnicas atuais, a programação por demonstração é a que mais facilita a tarefa do usuário. O fato de somente demonstrar as ações é fascinante. Todavia, a proposta de Bimbo e Vicario (1995) e McDaniel e Myers (1998) restringe as demonstrações. O primeiro, limita os comportamentos demonstrados àqueles possíveis de se exibir nas situações pré-definidas pelo designer da aplicação. O segundo, demanda a criação de comportamentos para cada ciclo de execução. Demonstrar comportamentos para cada situação, ou ciclo, pode ser enfadonho ao usuário. Impossibilitá-lo de agir livremente compromete a verossimilhança do IVA gerado, uma vez que, entediado, ele pode não exemplificar todas as ações necessárias para a correta atuação do agente. Não obstante, no trabalho de Bimbo e Vicario (1995), só é possível avaliar o comportamento gerado a partir da observação do agente executando-o, problema também encontrado no trabalho de McDaniel e Myers (1998), e este encapsulamento da solução pode causar novas decepções. Possibilitar a verificação antes da execução torna mais fácil e acessível ao usuário identificar erros, ou realizar melhorias na atuação do IVA. No sistema desenvolvido por Bimbo e Vicario (1995), para realizar essas mudanças, seria necessário que o usuário demonstrasse novamente, o que pode não agradá-lo. O mecanismo de "dicas", implementado por McDaniel e Myers (1998), minimiza as inferências errôneas, mas se estas existirem, assim como na implementação de Bimbo e Vicario (1995), somente uma nova demonstração pode corrigir a atuação do IVA.

Desta forma, vê-se que as técnicas de EUP utilizadas para a especificação do comportamento de IVAs são passíveis de questionamento quanto à real facilitação do trabalho do usuário. Portanto, percebe-se que algo ainda pode ser feito no intuito de minimizar os problemas encontrados.

3 Programação do Comportamento de Agentes por Demonstração

"*The real voyage of discovery consists not in seeking new landscapes but in having new eyes*"
Marcel Proust

Apresenta-se, neste capítulo, a nova abordagem de EUP proposta no trabalho, cujo objetivo é a facilitação da tarefa de especificação do comportamento de agentes virtuais inteligentes por usuários finais. Embora a nova abordagem tenha sido desenvolvida com o foco nessa tarefa, não descarta-se a possibilidade de utilização da mesma em outras áreas. Todavia, este trabalho não entrará no mérito desta questão.

3.1 A união faz a força

O estudo dos paradigmas seguidos pelos mecanismos de programação por usuários finais permitiu a identificação de características relevantes ao contexto do trabalho. As abordagens de EUP para a especificação do comportamento de IVAs que seguem o paradigma paramétrico limitam a tarefa à configuração dos agentes e não realmente a programá-los. O usuário final fica restrito a modificar algumas variáveis, e estas afetam o comportamento do IVA. Alterando as variáveis de forma consciente, e principalmente dosada (alterar todo o conjunto de variáveis de uma só vez compromete o entendimento) o usuário é capaz de associar as modificações feitas com o comportamento executado, sendo possível, portanto, prever a consequência de suas alterações.

Se o paradigma paramétrico peca pela falta, o descritivo tem seu ponto fraco exatamente no excesso de expressividade disponibilizada ao usuário. Comportamentos gerados a partir de mecanismos que seguem o paradigma descritivo podem ser bastante complexos, mas exigem em troca conhecimentos de programação. E este quadro já foi pior. Mesmo nas linguagens visuais, a manipulação de objetos do contexto era feita a partir de variáveis textuais, o que dificultava ainda mais a tarefa do usuário. A incorporação do paradigma imitativo solucionou essa questão.

Mecanismos de EUP que seguem o paradigma imitativo são, por motivos óbvios, os mais adequados ao problema. Especificar comportamentos por demonstração é o mesmo que ensinar pelo exemplo. Todavia, o encapsulamento da solução gerada a partir da demonstração vai em desencontro a usabilidade das abordagens que seguem esse paradigma. Uma vez que o usuário não programa explicitamente, o comportamento inferido pelo sistema a partir da demonstração pode não estar de acordo com aquilo que era esperado, sendo necessário, portanto, realizar modificações na solução proposta. Nos sistemas encontrados na literatura, essas modificações são possíveis somente via nova demonstração. Em sistemas aplicados em outras áreas, planilhas eletrônicas por exemplo, é comum a utilização do paradigma descritivo para solucionar esse problema. Neste caso, os problemas

descritos no parágrafo anterior voltam à tona.

Percebe-se, então, que os paradigmas vistos isoladamente possuem pontos que os qualificam como alternativas para a solução do problema abordado neste trabalho. Contudo, existem também características que comprometem sua utilização. Observa-se, também, que a aplicação de mecanismos de diferentes paradigmas para resolução do problema não é conflitante, e sim complementar. Recursos seguindo o paradigma imitativo resolvem a manipulação de variáveis textuais no paradigma descritivo, que por sua vez, permitem a modificação do comportamento, sem necessidade de nova demonstração.

Como dito anteriormente, não há solução mais intuitiva para a especificação do comportamento de agentes do que alguma que siga o paradigma imitativo. Mais especificamente a programação por demonstração, que soluciona o problema de manipulação de variáveis presente na gravação de macros. Entretanto, o encapsulamento da solução, também já explicitado, é um grande empecilho para a vasta aceitação do mecanismo.

Vista a complementaridade dos mecanismos de diferentes paradigmas, propõe-se a nova abordagem de EUP para a especificação do comportamento de IVAs. A programação visual de regras, com toda a expressividade do paradigma descritivo e seu fácil entendimento e manipulação, é a alternativa utilizada neste trabalho para contornar o encapsulamento das soluções geradas pela programação por demonstração.

Essa aliança caracteriza o caminho reverso da variante da programação visual de regras que utiliza os elementos visuais da aplicação para a criação dos condicionais. Ao invés de permitir que o usuário crie as regras exemplificando as situações que serão vividas no ambiente, nesta nova solução o usuário demonstra, através de um avatar imerso no VE, como o IVA deve se comportar. As ações tomadas pelo usuário diante das situações casualmente enfrentadas, resultam em regras de comportamento que serão seguidas pelo IVA, e nesta proposta podem ser posteriormente editadas sem nova demonstração. A abordagem permite que, por exemplo, um *bot* de um FPSG seja gerado a partir do simples registro do jogo de um usuário.

3.2 Nova solução, novos problemas

A incorporação da programação visual de regras à programação por demonstração traz consigo os problemas existentes neste mecanismo. Uma vez que as regras serão geradas automaticamente, não é necessário preocupar-se com o quão maçante seria criá-las nem com a possibilidade de geração de regras conflitantes. Contudo, sistemas que utilizam este

mecanismo normalmente se atêm à especificação de comportamentos simples, sem muitas variáveis a serem consideradas, posto que uma grande quantidade de regras dificultaria o entendimento do que foi implementado por parte do usuário. A proposta deste trabalho busca atender a todo o tipo de comportamento, simples ou complexo, e sobretudo garantir que usuários finais sejam capazes de utilizá-la. Portanto, uma nova adaptação se fez necessária para assegurar a compreensão do usuário.

O conjunto de variáveis a ser considerado para a especificação de comportamentos complexos é muito extenso. Seguindo a ideia do paradigma paramétrico, este trabalho incorpora ao mecanismo de programação por demonstração os atributos que serão levados em conta para a criação das regras. Para cada atributo, também deve ser definido um conjunto padrão de valores, que pode ser posteriormente editado conforme o interesse do usuário. A coleção de atributos definidos estará disponível ao usuário, e a cada um estará associado seu respectivo elemento visual (se existir) e descrição. Antes de iniciar a gravação o usuário deve definir quais atributos ele deseja que sejam considerados para a criação do comportamento. O que foi demonstrado pode ser utilizado caso, em um segundo momento, ele queira adicionar novos elementos desta coleção. A ideia deste recurso é permitir que o usuário dose, conforme seu grau de entendimento, a quantidade de variáveis a serem manipuladas, permitindo maior predição do comportamento que será executado pelo IVA a partir da análise das regras geradas.

A quantidade de regras é outro fator crítico da solução proposta. Por serem geradas a partir de arranjos de atributos, o número de expressões condicionais resultante da geração automática é imenso. Visto o problema, além das regras foram também adotadas árvores de decisão como forma de representação do conhecimento. Os nós de cada nível representam um atributo e os ramos destes fazem o papel dos valores associados. Acredita-se que a utilização desta técnica pode contribuir muito para o melhor entendimento do usuário, uma vez que ao invés de ser exposta uma lista de regras, ainda que ordenadas, o usuário pode navegar entre os nós da árvore até alcançar aquela que represente a situação que ele deseja analisar. A técnica ainda agrega valor à solução ao impedir que sejam geradas expressões com diferença na quantidade de atributos testados –a cargo do usuário poderiam haver regras avaliando apenas um, ou todos os atributos–, dispensando a necessidade de adoção de políticas de prioridade de execução para comportamentos vinculados a expressões mutuamente verdadeiras. Outro benefício está na avaliação das expressões condicionais. Enquanto regras não organizadas utilizando esta técnica precisam ser testadas uma a uma (O(n) no pior caso, onde n é o número de expressões), algoritmos podem otimizar o posicionamento dos atributos nos níveis da árvore, de forma a definir qual o

comportamento deve ser executado testando a menor quantidade possível de expressões.

3.3 O comportamento

Programar por demonstração é apresentar O QUE deve ser feito. Ao demonstrar, o usuário dá um exemplo da atitude que deve ser tomada naquela dada situação. COMO o IVA executará a mesma ação não é importante, desde que alcance o objetivo pretendido pelo usuário.

Desta forma, a intenção do usuário deve ser o foco da aplicação que implementa a abordagem proposta. As ações das regras condição-ação geradas devem ser apresentadas como representações do propósito do usuário, identificadas a partir da análise dos comandos executados. Extrai-se o QUE a partir do COMO.

Todavia, intenções são sensíveis ao contexto. Enquanto ANDAR PARA FRENTE é um comportamento totalmente aceitável em um VE bidimensional de deslocamento discreto (casas em um tabuleiro, por exemplo), o mesmo não se pode dizer caso o usuário esteja diante de um ambiente 3D, contínuo, como o de um FPSG. Portanto, é fundamental que o nível de abstração do conceito de ação empregado pelo projetista esteja de acordo com as necessidades do problema.

Além de facilitar o entendimento das regras geradas, trabalhar com intenção do usuário evita a determinação de comportamentos como a cópia exata dos comandos demonstrados. Comportamentos especificados a partir da réplica de instruções são suscetíveis a falhas em situações não idênticas àquela na qual houve a demonstração, uma vez que, assim como as macros, tendem a assumir valores das variáveis como constantes.

3.4 Formalização da abordagem

A seleção de ações é o meio pelo do qual um IVA resolve o problema de decidir o que fazer, qual atitude tomar. É a parte executiva da inteligência dos agentes (BRYSON, 2004). Sua análise permite predizer o comportamento de determinado agente sem ser necessário vê-lo agir. Segundo Bryson (2004) a adoção de padrões para a modelagem da camada de seleção de ações simplifica a solução, e consequentemente facilita o entendimento do usuário. Em seu trabalho, a autora apresenta planos reativos básicos[1], planos

[1]Basic Reactive Plans

reativos POSH[2], máquinas de estado finitas e determinismo ambiental[3] como alternativas à implementação.

Planos reativos básicos e POSH, padrões mais complexos, embora permitam especificações mais elaboradas, comprometeriam o propósito da incorporação do paradigma descritivo na programação por demonstração. Conceitos como prioridade e paralelismo podem não ser conhecidos pelo usuário e colocam em risco a compreensão da camada de seleção de ações.

Máquinas de estado finitas e Determinismo ambiental, o padrão adotado neste trabalho, oferecem uma modelagem mais palpável ao usuário. O primeiro é focado nas ações, enumera os estados que o agente pode se encontrar e as causas que motivam a transição entre estes. O segundo orienta-se nos eventos, discretiza o conjunto de situações que o agente pode presenciar e associa a cada uma delas a ação que deve ser tomada. Embora ambos se resumam a condicionais, as máquinas de estado possuem regras diferentes para cada estado, o que poderia dificultar a predição do comportamento que será executado. A simplicidade do conjunto de regras, produto do determinismo ambiental, motivaram a escolha deste trabalho.

Tão importante quanto a escolha da programação por demonstração como estratégia de aprendizado, é a forma em que o sistema representa o conhecimento adquirido. O sucesso reconhecido na utilização das linguagens de programação visuais baseadas em regras justificariam modelagem dos eventos como regras, todavia a vasta quantidade de expressões condicionais motivaram a opção por um esquema híbrido. Este trabalho propõe o modelo formado por regras (intrínsecas ao mecanismo do paradigma descritivo adotado) e árvores de decisão como representação do conhecimento presente na camada de seleção de ações. As justificativas podem ser vistas na Seção 3.2.

Portanto, o comportamento gerado é formado pela mapeamento $A^n \mapsto B$. Onde: A representa o conjunto de atributos selecionados pelo usuário para a criação das regras, $A = \{A_1, A_2, \ldots, A_n\}$; e B, por sua vez, simboliza o conjunto de comportamentos demonstrados, $B = \{B_1, B_2, \ldots, B_k\}$. A é subconjunto de R, biblioteca de atributos que podem ser considerados pelo usuário e definidos previamente pelo projetista da aplicação. Cada atributo pertencente à biblioteca R possui seu conjunto de valores associado $V_{R_i} = \{V_{R_i1}, V_{R_i2}, \ldots, V_{R_im_{R_i}}\}$, que podem ser definidos pelo designer ou editados conforme a necessidade do usuário. Não necessariamente, todos os comportamentos do

[2]Parallel-rooted, Ordered, Slip-stack Hierarchical
[3]Environmental Determinism

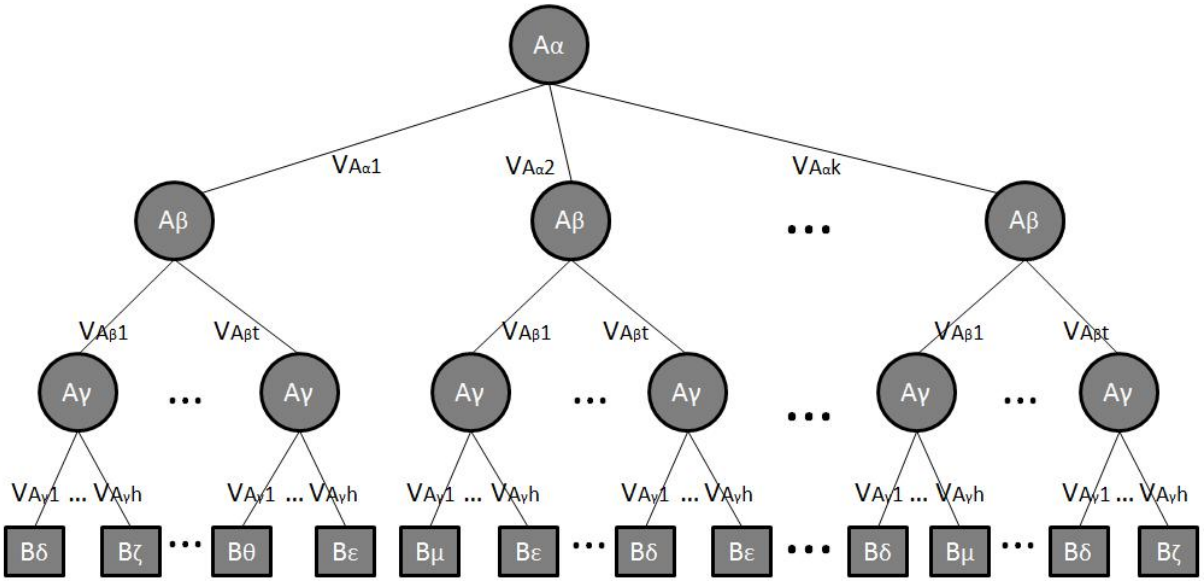


Figura 1: Exemplo da camada de seleção de ações para o comportamento demonstrado em esquema de representação do conhecimento híbrido: regras e árvore de decisão.

conjunto B estarão presentes na árvore de decisão, alguns deles podem estar disponíveis apenas para futura edição do comportamento gerado.

Um exemplo genérico do comportamento gerado pode ser visto na Figura 1. Neste caso, tem-se o conjunto de atributos selecionados pelo usuário sendo $A = \{A_\alpha, A_\beta, A_\gamma\}$ com seus respectivos conjuntos de valores: $V_{A_\alpha} = \{V_{A_\alpha 1}, V_{A_\alpha 2}, \ldots, V_{A_\alpha k}\}$, $V_{A_\beta} = \{V_{A_\beta 1}, \ldots, V_{A_\beta t}\}$ e $V_{A_\gamma} = \{V_{A_\gamma 1}, \ldots, V_{A_\gamma h}\}$. Os comportamentos exibidos na árvore: B_δ, B_ζ, B_θ, B_ϵ e B_μ pertencem ao conjunto B, mas não se pode dizer que B possui somente estes, outros podem estar disponíveis para posterior edição. A leitura de uma das expressões da árvore exemplificada é: Caso A_α tenha valor $V_{A_\alpha 1}$ e A_β igual a $V_{A_\beta 1}$ e A_γ igual a $V_{A_\gamma 1}$, então execute o comportamento B_δ.

3.5 Sugestão de implementação

A extração do conhecimento é o núcleo de um sistema que implementa o mecanismo de programação por demonstração. Identificar a intenção dominante para cada situação, uma vez que podem haver inconsistências, é fundamental para a verossimilhança do IVA.

Ao analisar uma dada demonstração, podem ser identificadas várias intenções para cada estado vivido, seja por imperícia, experiência de insucesso prévia ou distração do usuário final ao executar a tarefa. A função de avaliação (que pode julgar o nível de sucesso ou simplesmente contar as execuções) determina então "pesos" para as intenções

e indica a mais adequada para a situação em questão. A abertura da camada de seleção possibilita que a decisão do comportamento associado não passe somente pela função de avaliação implementada.

Para permitir que o usuário final possa utilizar de forma intuitiva a proposta de edição das regras sem a necessidade de novas demonstrações, sugere-se uma postura de classificação dos comportamentos semelhante à dos atributos, discutida na Seção 3.2. As ações devem ser tão compreensíveis quanto as condições. Portanto, sugere-se que:

- Todos os comportamentos demonstrados pelo usuário estejam associados a termos de fácil interpretação e, se possível for, a elementos visuais presentes na aplicação;
- Os comportamentos demonstrados façam parte de uma biblioteca, que será a fonte de recursos para a edição das regras;
- Se um dado comportamento presente na biblioteca foi demonstrado pelo usuário naquela situação, mesmo que este não seja o julgado pela função de avaliação como mais adequado, essa informação e o "peso" recebido estejam disponíveis em termos intuitivos ou que não fujam aos do contexto da aplicação.

4 Ambiente de testes

"*Man is a tool-using animal.*"

Thomas Carlyle

Este capítulo apresenta o sistema desenvolvido para validar a possibilidade de se especificar o comportamento de agentes virtuais inteligentes utilizando a nova abordagem de programação para usuários finais proposta no trabalho. O ambiente foi desenvolvido como plataforma de testes, todavia não são necessárias muitas adaptações para o seu aproveitamento em outras áreas de pesquisa, ou mesmo em aplicações comerciais.

4.1 A Plataforma

Embora existam diversos sistemas de criação de simulações genéricas, Agentsheets (REPENNING, 1993) e KidSim (CYPHER; SMITH, 1995) por exemplo, a utilização destes para o desenvolvimento da plataforma de testes é impossível, pois seriam necessárias modificações e estes são proprietários. O trabalho de Coura *et al.* (2006) deparou-se com a mesma situação e implementou parte de um sistema aberto para esse propósito. A opção de finalizar o desenvolvimento e fazer as alterações necessárias foi considerada, mas decidiu-se trabalhar com um ambiente mais realista, semelhante aos VEs atuais.

O custo de desenvolvimento de um VE que incorporasse os recursos visuais dos ambientes atuais seria enorme. Entretanto, a modularização do desenvolvimento de jogos de computador permite que estes sejam adaptados para atender às necessidades do trabalho. A adaptação destes jogos é conhecida como *modding*(EL-NASR; SMITH, 2006). Desde o final dos anos 90, entusiastas já alteravam mapas e as formas de se jogar Doom, um dos precursores dos FPSGs. Outro exemplo a ser citado é o Counter Strike, uma modificação de tanto sucesso do Half Life, outro FPSG, que passou a ser distribuída junto da continuação do jogo, Half Life 2. Outros exemplos de jogos que permitem adaptações podem ser vistos no trabalho de El-Nasr e Smith (2006), que apresenta um estudo da abordagem para auxílio pedagógico.

Contudo, as adaptações que podem ser realizadas dependem das possibilidades oferecidas pelo jogo. Disponibilizar ferramentas de *modding* que necessitassem um profundo conhecimento de programação restringiria o acesso dos jogadores ao recurso. Buscando atender não somente um público específico, a indústria de jogos optou por desenvolver linguagens de extensão mais amigáveis, incorporando técnicas de EUP. Destacam-se, entre as abordagens utilizadas, a configuração de parâmetros, programação visual e as linguagens de script. Como já foi dito, as técnicas de EUP facilitam o trabalho de programação ao usuário final, mas, por outro lado, restringem as extensões que podem ser desenvolvidas. Os trabalhos de Yucel, Zupko e El-Nasr (2006) e El-Nasr e Smith (2006) estudaram a

(a) UT original

(b) UT menos violento. Fonte: Kaminka *et al.* (2002)

(c) UT modificado para jogo de corrida, UnWheel.

(d) UT modificado para jogo de guerra, Red Orchestra.

Figura 2: Exemplos da capacidade de customização do Unreal Tournament

utilização de várias linguagens de extensão e a Unreal Script, destacou-se por permitir a customização de praticamente todo o jogo. Alguns exemplos de modificação do UT podem ser vistos na Figura 2, onde destacam-se UnWheel[1] e Red Orchestra[2]. A expressividade da Unreal Script foi determinante para a escolha da plataforma de implementação da ferramenta de validação da solução proposta.

UT é um FPSG desenvolvido pela Epic Games com suporte multiusuário. Os jogadores são representados em um mundo virtual tridimensional com conceitos físicos simulados. Na versão 2004 (UT 2004), existem dez modalidades de jogo pré-definidas das quais se destacam *Death Match* e *Capture The Flag*. Na primeira, o jogador luta pela sobrevivência, e acumula pontos cada vez que mata um inimigo. Já o objetivo do *Capture the Flag* é proteger sua bandeira, enquanto tenta pontuar invadindo a base inimiga e conduzindo a bandeira rival a sua base. Além da linguagem de script embutida, UT 2004 ainda oferece um editor de mapas e permite a utilização de ferramentas de editoração 3D externas para a modificação de personagens e animações.

[1]UnWheel - http://unwheel.beyondunreal.com/
[2]Red Orchestra - http://www.redorchestragame.com/

A extensa capacidade de personalização permitiu que o UT fosse utilizado para pesquisas em diversas áreas, e para diversos públicos (LAIRD *et al.*, 2002; JACOBSON; HWANG, 2002; MAGERKO *et al.*, 2004). Na área de especificação de agentes virtuais inteligentes pode-se citar: Gamebots (KAMINKA *et al.*, 2002), Indigente (MONTEIRO, 2006), UTBots (KIM, 2007) e Pogamut 2 (BURKERT *et al.*, 2007). A compatibilidade da linguagem Unreal Script com as novas versões do jogo assegura o funcionamento de adaptações em ambientes com gráficos cada vez mais realistas e recursos mais motivantes.

Gamebots, segundo Kaminka *et al.* (2002), foi proposto como uma plataforma de realidade virtual que permite a criação e avaliação de agentes inteligentes interagindo com um ambiente 3D contínuo e dinâmico. Desenvolvido como extensão do UT, usufrui de toda a flexibilidade de adaptação do jogo, permitindo testes em inúmeras situações, ao contrário dos sistemas específicos existentes. Gamebots permite o controle de personagens do UT em arquitetura cliente-servidor, enviando informações sensoriais para o cliente (que pode ser um IVA ou um jogador), e repassando ao servidor do jogo quais os comandos foram executados pelos agentes. Embora ofereça recursos para o desenvolvimento de agentes, a comunicação por mensagens, a quantidade de trabalho de baixo nível que precisa ser realizado antes de realmente iniciar a especificação dos IVAs, e a não existência de uma arquitetura de agentes pré-definida são barreiras mesmo para programadores mais experientes.

Os arcabouços propostos por Monteiro (2006), Kim (2007) e Burkert *et al.* (2007) oferecem exatamente a solução para os problemas enfrentados no Gamebots. Todos encapsulam-no, oferecendo uma camada de alto nível para a especificação do IVAs, e uma arquitetura para a modelagem de agentes. Facilitam o trabalho do programador, que só precisa definir qual será o comportamento a ser executado a partir de cada estado do agente modelado e do ambiente, informações disponíveis nos objetos dos arcabouços. Ainda que a ideia seja a mesma, cada um oferece características que merecem destaque. O Indigente Agent Framework (MONTEIRO, 2006) oferece uma sólida base para a implementação de agentes cognitivos, com suporte a Prolog. UTBots (KIM, 2007), por sua vez, foi desenvolvido para facilitar o trabalho de especificação de IVAs por estudantes em aprendizado de programação, e inclui uma biblioteca de comportamentos de exemplo que facilita a tarefa para iniciantes. Outro recurso interessante é a implementação de algoritmos de navegação em tempo real. Pogamut 2 (BURKERT *et al.*, 2007), o mais conhecido, destaca-se por oferecer interface gráfica de desenvolvimento, suporte a diferentes linguagens e ferramentas de depuração.

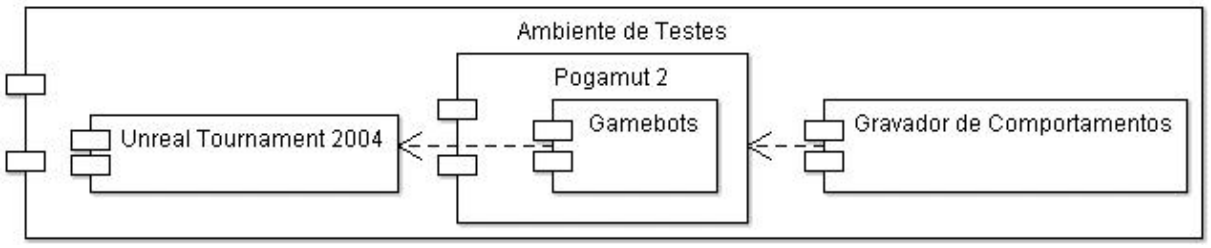


Figura 3: Diagrama de componentes do ambiente de testes

Pogamut 2 continua em desenvolvimento. O foco hoje é incorporar a modelagem de emoções aos agentes. A grande quantidade de informação disponível online, graças à sua vasta rede de usuários, o suporte por parte dos desenvolvedores e a experiência prévia de utilização do arcabouço no Departamento de Informática da Universidade Federal de Viçosa (DPI-UFV), onde também foi realizado este trabalho, justificam sua escolha para a implementação da plataforma de testes.

Portanto, a plataforma escolhida para a implementação do ambiente de testes da nova abordagem de EUP proposta é o jogo Unreal Tournament 2004, devido a sua flexibilidade de adaptação e o arcabouço Pogamut 2, justificado acima. O diagrama de componentes do ambiente de testes é apresentado na Figura 3.

4.2 Vista geral

A ideia básica da aplicação é permitir que o usuário jogue UT 2004 e que a partir do comportamento por ele executado, um IVA seja gerado. Na interface do ambiente de testes, o usuário conecta-se ao servidor de um jogo previamente criado no UT. Neste, é inserido um agente marionete, que será comandado pelo usuário diante das situações vividas no ambiente. Solicita-se, então, a geração das regras a partir do registro da demonstração. A camada de seleção de ações é disponibilizada e o usuário tem a opção de editá-la. Satisfeito com as expressões resultantes, resta conectar o IVA padrão no ambiente, que interpretará as regras geradas e executará um comportamento semelhante ao que foi demonstrado.

4.3 Arquitetura do Sistema

O sistema foi desenvolvido prezando pela manutenibilidade e modularização, uma vez que este pode ser objeto de estudo de mais trabalhos envolvendo o tema. Dividiu-se a aplicação nos módulos de interface, controle, domínio, configuração, logging e inferência.

O módulo de interface é responsável por intermediar a comunicação do usuário com o sistema. As informações obtidas com a interação do usuário são repassadas para os componentes no módulo de controle através das interfaces de comunicação oferecidas, para serem tratados pela aplicação.

A camada de controle possui os componentes que gerenciam toda a lógica da aplicação. Estes componentes são responsáveis por: realizar a conexão/desconexão da aplicação com o UT2004; inserir e remover agentes no ambiente virtual; enviar comandos ao agente já inserido no ambiente; registrar o comportamento demonstrado a cada nova situação no arquivo de saída; disponibilizar a biblioteca de atributos; e identificar as intenções do usuário em cada uma das situações definidas por ele como relevantes.

Na camada de domínio, encontram-se os componentes que representam as principais entidades da aplicação: o agente marionete, através do qual o usuário executará o comportamento, a implementação dos comandos que responderão aos comandos de entrada do usuário e a base para os IVAs que serão gerados. Esse módulo se fez necessário devido ao Pogamut 2 não aceitar entradas do usuário em tempo real para controlar os *bots*.

O módulo de configuração assegura a persistência das preferências do usuário. Desta forma, não é preciso reconfigurar o sistema após cada utilização. Optou-se por armazenar os dados no formato XML devido à portabilidade, organização, simplicidade e legibilidade.

Por sua vez, a camada de logging responde pelo registro das ações executadas pelo IVA marionete junto de seu estado e da situação do ambiente que o cerca. Os dados também são armazenados em XML.

A partir do arquivo gerado pela camada de Logging, o módulo de inferência identifica as situações definidas como relevantes e para cada uma delas infere qual foi a intenção do usuário. As regras geradas são então interpretadas pelo agente base implementado na camada de domínio e este é o IVA gerado.

4.4 Funcionamento do Sistema

O sistema baseia-se em uma interface, que serve de controle remoto a um agente inserido no ambiente virtual. As ações executadas são gravadas e posteriormente analisadas para a geração de regras de execução que determinarão o comportamento a ser executado pelo IVA.

Inicialmente, é necessário executar o Unreal Tournament e iniciar um servidor de

jogo em um dos modos que suportam o controle de agentes via *socket* TCP. Os modos disponibilizados pelo Pogamut 2 ao UT 2004 são: *Remote Bot Capture the Flag*, *Remote Bot DeathMatch*, *Remote Bot Double Domination* e *Remote Bot Team DeathMatch*. O segundo passo é escolher, na biblioteca de atributos, aqueles considerados relevantes. Conecta-se então o sistema ao servidor, e no ambiente adiciona-se o agente marionete.

Abre-se então a janela do servidor do UT em paralelo com a interface do sistema. O papel do agente "host" do servidor deve ser alterado de jogador para espectador. Posiciona-se, então, a visão do agente espectador como a vista em primeira pessoa do agente marionete adicionado e inicia-se a demonstração. Por meio da interface do sistema e utilizando os mesmos controles do jogo, comanda-se o agente marionete diante das situações visualizadas na janela do UT. Não existe tempo máximo ou mínimo para a última tarefa. Deixa-se claro, entretanto, que quanto mais tempo for despendido na exemplificação, menos regras terão de ser definidas manualmente, para a criação de um IVA que se comporte de forma crível.

Depois de demonstrar o comportamento pelo tempo desejado, deve-se parar a gravação, remover o agente marionete do ambiente, desconectar-se do servidor e solicitar o armazenamento em disco do registro realizado. Um arquivo XML será gerado com a listagem dos itens do ambiente e uma coleção de episódios. Cada episódio é formado por um conjunto de cenas. Uma cena representa uma espécie de "foto" do ambiente em um dado instante. Armazena-se em uma cena: o horário que esta ocorreu; a lista de comandos que estava sendo executada naquele momento (i.e. mover-se para frente e atirar); o estado do ambiente, formado pela lista de itens (posição e tipo) e jogadores visíveis(posição, um marcador se é inimigo e arma que está sendo manuseada), e os pontos de navegação acessíveis; e o estado do agente marionete, sendo este composto por pontos de vida, lista de armas e munição de cada uma, e sua localização no mapa. As cenas são capturadas a cada início ou término de comando executado no ambiente. Os episódios são gerados a cada reentrada do agente no ambiente, ou seja, a cada turno do jogo.

O arquivo XML gerado deve ser, então, submetido ao analisador que identificará nele os atributos definidos como relevantes ao usuário e para as conjunções de parâmetros, atributo-valor, realizará uma inferência da intenção do usuário naquele momento. A análise é feita a partir da avaliação da variação de atributos entre cenas sequenciais. Por isso a importância dos episódios, que asseguram que as cenas estejam sempre arranjadas de forma contínua, sem variações bruscas. A não existência destes implicaria em problemas, tal qual uma cena onde o agente marionete estivesse prestes ficar sem pontos de vida

(morrer) ser seguida por outra, na qual o agente está com os pontos de vida completos. A variação nos atributos permite a identificação de eventos, comportamentos ou situações. No próximo capítulo, apresenta-se um estudo de caso, e exemplos concretos da análise serão exibidos. Uma vez que o número de pares de cenas analisados é muito grande, o analisador não realiza tratamento de ruídos. Os possíveis erros de execução do usuário são, por si só, insignificantes no resultado final da análise, não precisando assim serem filtrados.

Uma árvore tal qual a apresentada na Figura 1 será disponibilizada e em cada folha será apresentado o comportamento demonstrado. Caso algum comportamento tenha sido identificado pelo analisador, este possuirá um valor determinado pela função de avaliação. O de maior "peso" estará selecionado, sendo, portanto, o que será executado caso o usuário não realize nenhuma alteração. Uma nova conexão com o servidor deve ser então realizada e o IVA nele inserido. Finalmente, o IVA interpreta o produto do analisador validado pelo usuário, comportando-se segundo as regras estabelecidas. A Figura 4 apresenta o diagrama de sequência das atividades descritas nesta seção.

4.5 Considerações finais

O desenvolvimento da ferramenta deveu-se à necessidade de validar a nova abordagem de EUP proposta neste trabalho. Produzir um ambiente que permita a especificação do comportamento de IVAs por demonstração não é o objetivo do trabalho. A implementação do ambiente de testes deu-se em parceria com um trabalho de iniciação científica conduzido no DPI-UFV, e foi realizado somente até este encontrar-se funcional. Portanto, extensões devem ser realizadas para assegurar a usabilidade necessária para sua utilização de forma intuitiva por usuários finais. No Capítulo 6 são apresentadas sugestões de trabalhos a serem realizados para incorporar novos recursos à ferramenta, possibilitando sua utilização comercial ou para realização de pesquisas em outras áreas. A Figura 5 apresenta duas das interfaces funcionais da ferramenta implementada no trabalho.

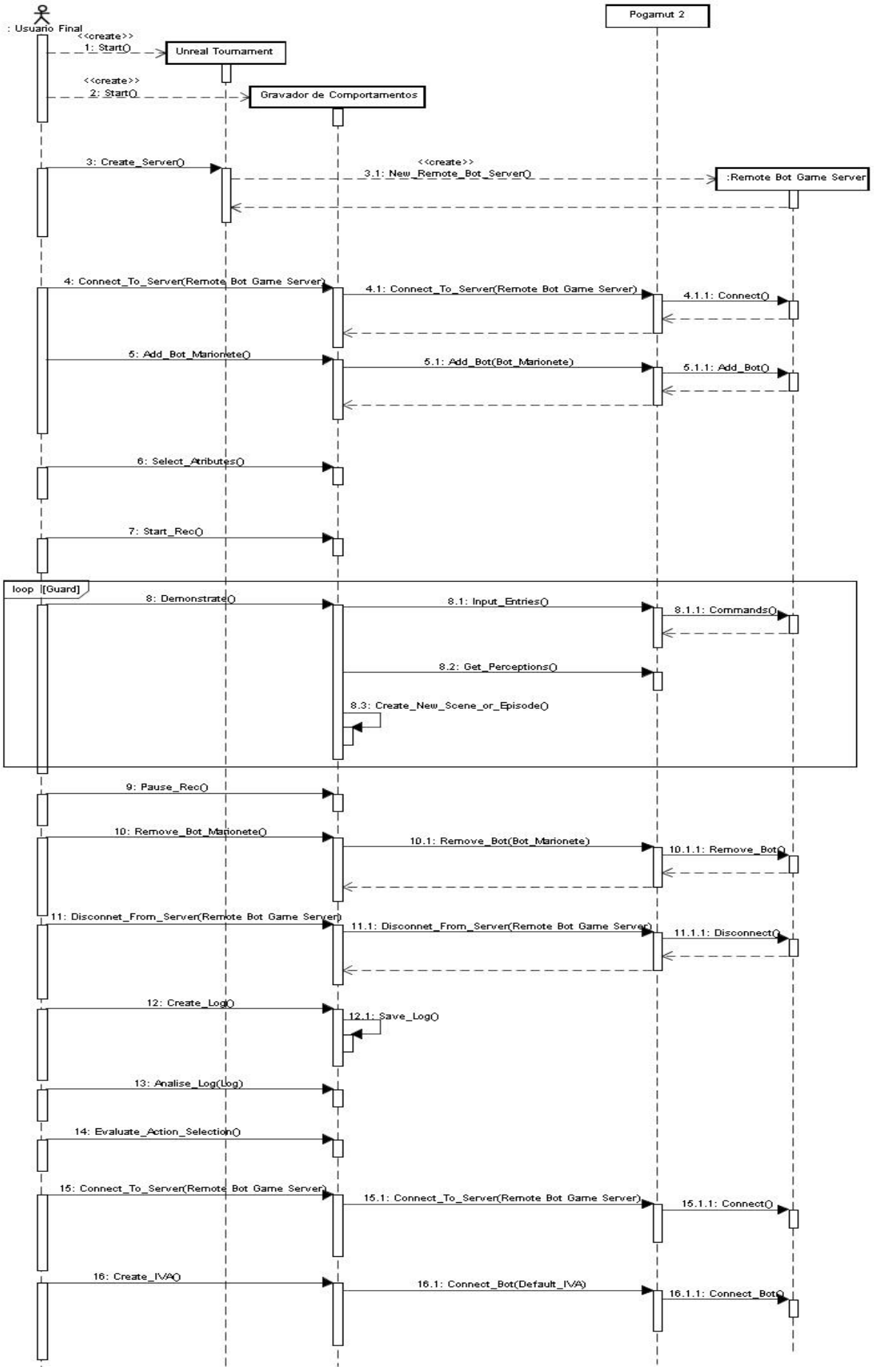
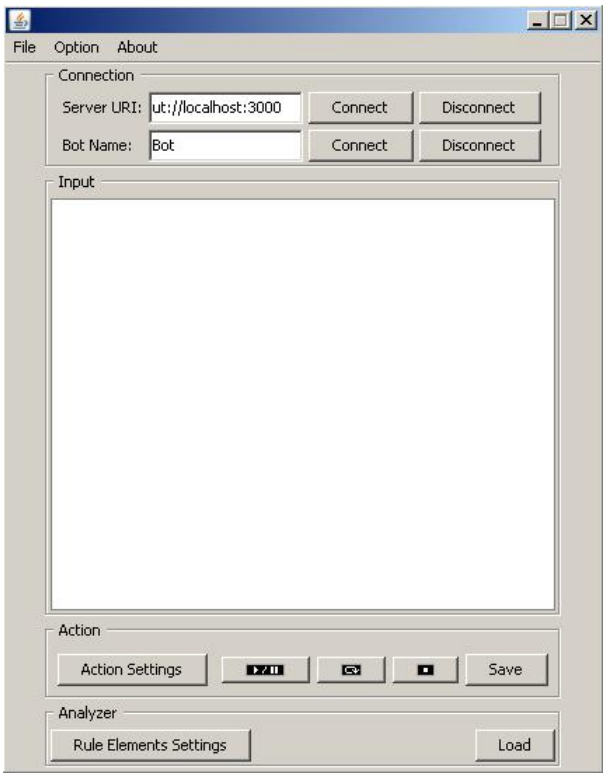
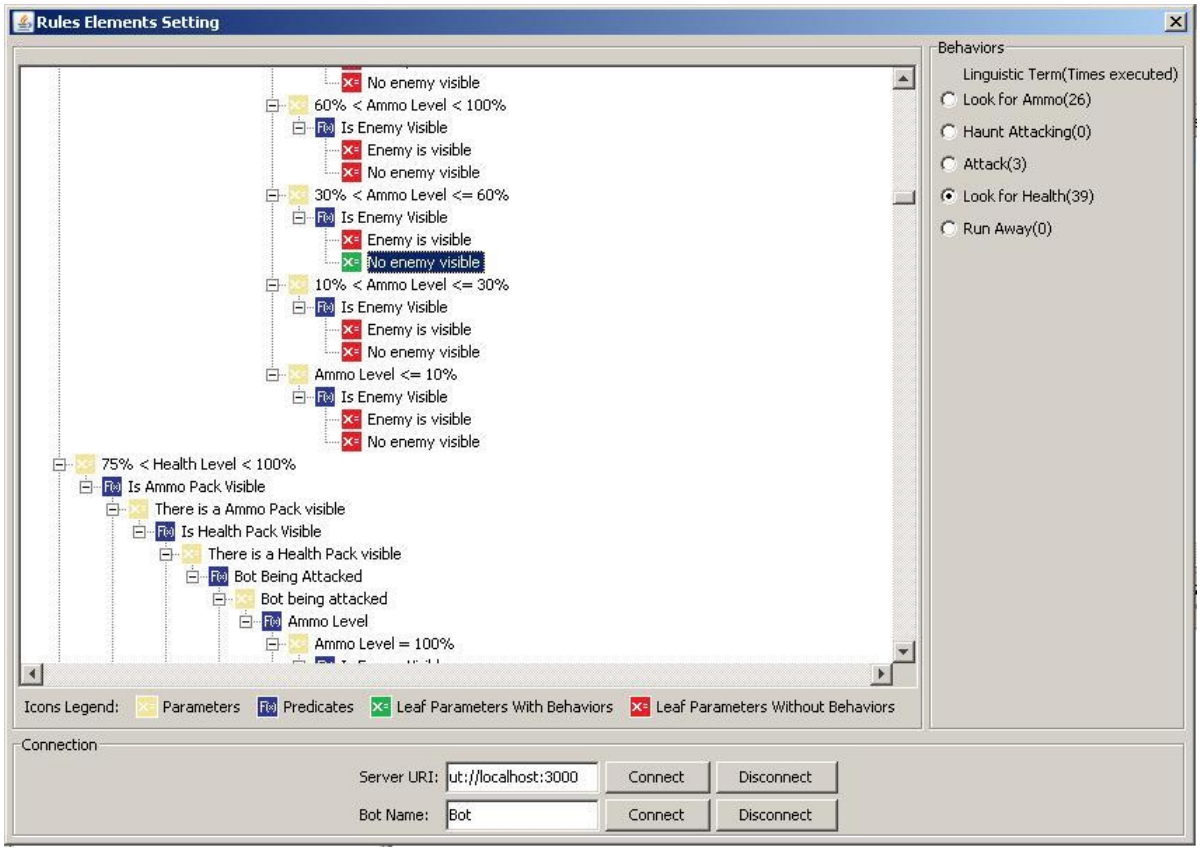


Figura 4: Diagrama de sequência da interação básica com o ambiente de testes.



(a) Interface inicial.



(b) Interface da árvore de decisão.

Figura 5: Interfaces do ambiente de testes.

5 Validação e Avaliação

"*In theory, there is no difference between theory and practice. But, in practice, there is.*"

Jan L. A. van de Snepscheut

Apresenta-se, neste capítulo, o estudo de caso realizado como forma de validar a especificação do comportamento de agentes virtuais inteligentes utilizando a nova proposta de programação para usuários finais, detalhada no Capítulo 3. É apresentada também uma análise qualitativa entre os resultados observados na criação do exemplo e os que seriam teoricamente obtidos caso fossem aplicadas as técnicas de EUP já utilizadas para o mesmo propósito.

5.1 Condições de realização do estudo de caso

A utilização do Unreal Tournament como base de implementação do ambiente de testes torna possível a exploração de ambientes mais realistas, com mais possibilidades a serem exploradas pelos usuários. Todavia, para validar a proposta do trabalho, optou-se por trabalhar em um ambiente mais discreto. Isso foi feito para condizer os comportamentos criados com a pequena quantidade de atributos disponibilizados na biblioteca de comportamentos. Através da utilização de "mutators", objetos de extensão do UT, foram removidos itens (armas, armaduras, estamina, etc...) e recursos (pular e tiros secundários). Desta forma as limitações impostas aos IVAs gerados, a partir da pequena quantidade de atributos existentes, não deixariam a desejar uma vez que também estão restringindo o comportamento demonstrado.

O exemplo criado para validar a proposta deste trabalho baseia-se no modo de jogo *Remote Bot DeathMatch* oferecido pelo Pogamut 2 ao UT 2004. Assim como o modo DeathMatch, descrito anteriormente, o jogador luta pela sobrevivência, e acumula pontos cada vez que mata um inimigo. O diferencial desta variante é suportar o controle de agentes via *socket* TCP. Embora esse modo de jogo permita a entrada de vários jogadores, que combatem entre si, optou-se por trabalhar somente com um único adversário. Ainda que fosse gerado um número maior de regras, conceitualmente o resultado final seria o mesmo.

Adaptou-se um mapa com povoação sugerida de dois usuários para evitar pontos de suicídio, nível de detalhe julgado dispensável. O mapa editado pode ser visto na Figura 6. Foram adicionadas grades de proteção para evitar quedas que levariam à morte imediata do usuário. Itens foram posicionados em cada extremidade do cenário. Em um ponto Bandagens, que recuperam os pontos de vida do usuário, e no outro, Munição para a única arma existente no jogo, já equipada ao usuário no momento de entrada no mapa. Desta forma, força-se o jogador a deslocar-se pelo mapa, aumentando as possibilidades de

Figura 6: Mapa DM-Training Day, editado para o estudo de caso criado.

confrontação.

A jogabilidade do UT também foi alterada, devido aos problemas enfrentados na fidelidade de reprodução na interface do UT dos movimentos do mouse capturados no Ambiente de testes. A sensibilidade era muito comprometida e optou-se por limitar a navegação do agente marionete aos comandos de teclado. Para isso, alterou-se alguns comandos existentes no Unreal. Enquanto no UT as teclas SETA PARA ESQUERDA e SETA PARA DIREITA realizam movimentos laterais no agente comandado, no ambiente de testes estas representam virar a esquerda e a direita. A Figura 7 ilustra a adaptação. Não obstante, a ação de disparar a arma equipada, que no UT também era realizada através do mouse, foi alterada para a tecla *Ctrl*. Embora as modificações comprometam a plena jogabilidade, refinamentos na lógica de programação da troca de mensagens entre o Ambiente de Testes e o UT podem assegurar a sensação de estar-se jogando Unreal Tournament mesmo interagindo por meio da ferramenta desenvolvida. Tais adaptações não foram realizadas por não serem significativas para validação da proposta.

Os atributos disponibilizados na biblioteca do ambiente de testes e seus respectivos conjuntos de valores padrão podem ser vistos na Tabela 1. Embora aparentem ser poucos, estes se mostraram suficientes para a criação de um IVA que se comporta de forma semelhante à demonstração do usuário para a modalidade de jogo escolhida. Todavia, reitera-se que a biblioteca de atributos deve contemplar situações e informações relevantes para toda modalidade de jogo, uma vez que estes não podem ser criados em tempo de execução, e mesmo se pudessem demandariam conhecimento de programação não comumente presente nos usuários finais. A identificação de cada um dos atributos é feita analisando-se os dados

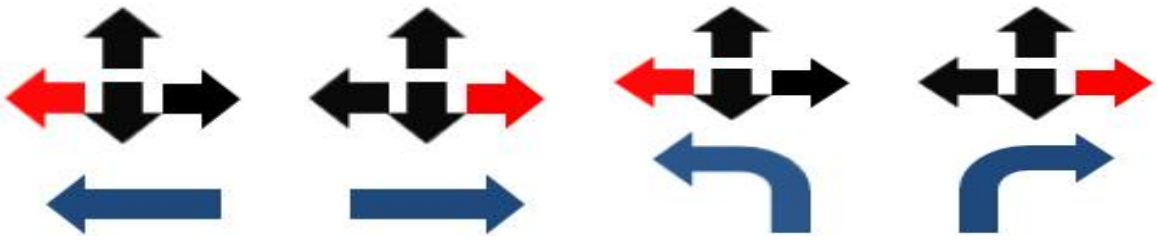
Tabela 1: Tabela com os atributos, e seus respectivos valores, disponibilizados na biblioteca do ambiente de testes para o estudo de caso

Atributos	Valores Textuais	Valores
Nível de Pontos de Vida	Completo	X = 100%
	Bom	75% <= X < 100%
	Razoável	50% <= X < 75%
	Mediano	25% <= X < 50%
	Ruim	X < 25%
Nível de Munição	Completo	X = 100%
	Bom	60% <= X < 100%
	Razoável	30% <= X < 60%
	Mediano	10% <= X < 30%
	Ruim	X < 10%

Atributos	Valores Textuais	Valores
Inimigo no campo de visão	Sim	X = TRUE
	Não	X = FALSE
Munição no campo de visão	Sim	X = TRUE
	Não	X = FALSE
Bandagens no campo de visão	Sim	X = TRUE
	Não	X = FALSE
Está sofrendo ataque	Sim	X = TRUE
	Não	X = FALSE

do registro da demonstração – como no caso do *Nível dos pontos de vida* –, ou a variação destes – i.e. diferença nos pontos de vida entre cenas, para avaliar o atributo *Está sofrendo ataque*. Na execução do estudo de caso, todos os atributos criados foram utilizados.

Para ilustrar a necessidade de simplificação do jogo descrita anteriormente, discute-se a situação hipotética de não se ter restringido o acesso às armas do UT. Se houvesse mais armas, não bastaria tratar a quantidade de munição da forma com que é feita no atributo implementado, sendo necessário avaliar a quantidade de munição para cada uma das armas no inventário. Outro exemplo é a detecção de presença do inimigo, uma vez que este, com uma arma que causa dano superior a que se está equipado é mais nocivo que se estivesse com o mesmo item. Essas considerações poderiam ser levadas em conta pelo usuário ao demonstrar o comportamento, e não seriam identificadas pelo analisador, causando insatisfação diante do IVA gerado.



(a) Movimento para esquerda padrão UT

(b) Movimento para direita padrão UT

(c) Movimento para esquerda adaptado para o Ambiente de Testes

(d) Movimento para direita adaptado para o Ambiente de Testes

Figura 7: Diferenças nos comandos de teclado padrão do UT para o Ambiente de Testes criado. Acima, os comandos do teclado com a tecla em questão em destaque; abaixo, o movimento realizado pelo personagem.

Realizada a simplificação do ambiente e a implementação dos atributos disponibilizados, partiu-se para a identificação das intenções do usuário. Embora existam diversos algoritmos para este propósito baseados em técnicas de IA, o ambiente de testes foi desenvolvido de forma mais simples. Assim como nos atributos, através da análise de cada par de cenas é inferida qual foi a intenção do usuário. Para tanto, foi necessário definir quais comportamentos poderiam ser executados pelo usuário durante a demonstração. Restrito o ambiente e, principalmente, o modo de jogo do estudo de caso, levantou-se os comportamentos condizentes, ou seja, aqueles que um jogador de UT poderia realizar. Finalmente, para cada um destes foram determinadas as variáveis existentes no registro da demonstração que o identificaria. É sabido que tal implementação exige do desenvolvedor da aplicação predizer os possíveis comportamentos a serem demonstrados para o usuário. Restringir as ações implementadas àquelas adequadas à situação vivida é excessivamente simplório. A intenção do usuário pode ser exatamente criar um personagem que destoe de todos aqueles que comumente estão presentes naquela circunstância, e crer que é possível vaticinar todas as necessidades do utilizador é demasiadamente utópico (volta à tona a motivação da programação por usuários finais). Todavia, aos propósitos deste estudo de caso, tal simplificação é satisfatória. Os comportamentos implementados foram: *Atacar*, no qual o usuário dispara contra o inimigo; *Atacar Perseguindo*, o usuário vai em direção ao inimigo investindo contra o mesmo; *Procurar por munição*, aproximar-se do local onde existem cartuchos de munição; *Procurar por bandagens*, idem; e *Fugir*, deslocar-se na direção oposta a do inimigo.

Fica evidente que os comportamentos não são conflitantes. O analisador pode identificar mais de um para cada atributo. Cabe à função de avaliação ponderar qual ação, ou conjunto de ações, é mais adequado para aquela situação vivida. Neste estudo de caso, optou-se por um contador como métrica de decisão. Cada atitude do usuário, através do agente marionete, em determinada circunstância é contabilizada e aquela de maior frequência é a escolhida para ser realizada pelo IVA. Embora simples, essa implementação permite variantes que agregariam mais realismo ao IVA, tal como realizar um sorteio entre todos os comportamentos realizados, sendo que os de maior incidência possuiriam maior chance de serem selecionados. Portanto, o ambiente de testes gera IVAs que diante de determinada situação realizam um, e somente um, dos comportamentos previamente implementados. Todavia, nada impedirá que o usuário decidida qual política deve ser adotada pelo IVA gerado, depois de ponderados os comportamentos identificados, quando este recurso estiver disponível.

5.2 Metodologia

Para a realização da validação, solicitou-se a usuários com conhecimentos da jogabilidade do UT que jogassem duas sessões de seis minutos através do ambiente de testes. Na primeira, não lhes foi informado que seu jogo seria gravado e nem sequer exibidas as situações diante das quais seus comportamentos seriam identificados. Na sessão seguinte, foi-lhes apresentado o propósito da aplicação e as circunstâncias "críticas" da gravação.

Tal metodologia foi aplicada esperando que após tomarem ciência da necessidade de demonstrar o comportamento em cada uma das situações pré-determinadas estes se esforçarão para vivência-las. Os resultados podem ser vistos na próxima seção.

5.3 Resultado

Todos os usuários que participaram do estudo de caso conseguiram criar IVAs, mesmo antes de serem orientados a buscarem as situações relevantes. Eles simplesmente jogaram e obtiveram um IVA que se comporta de forma semelhante à demonstrada. Portanto, a nova proposta de programação por usuários finais desenvolvida neste trabalho é válida para a especificação do comportamento de agentes virtuais inteligentes.

Nos seis minutos de demonstração da primeira seção foram criadas, em média, aproximadamente 36 regras. Na seção realizada após a orientação do propósito da aplicação a média subiu para 41 regras, um aumento de cerca de 12%. Portanto, a hipótese levantada na seção anterior foi comprovada para o período de tempo estabelecido.

Ao se depararem com o IVA gerado sem a tutela do responsável pelo estudo de caso, os usuários não reconheceram o comportamento como correspondente à exibição por eles realizada. Todavia, quando orientados, estes conseguiram identificar as regras por eles demonstradas. Acredita-se que não houve a percepção no primeiro momento devido à expectativa de visualizar um IVA tal qual os *bots* existentes no jogo. Uma vez destacadas as situações para as quais regras foram geradas, percebeu-se a satisfação com o IVA. A possibilidade de alteração das regras sem a necessidade de demonstrar novamente foi também elogiada. Após a orientação recebida esclareceu-se, também, a necessidade de estabelecer regras para todas as conjunções entre os atributos, e a relação direta entre número de regras e a verosimilhança do comportamento do IVA. Novamente, o fato da criação de regras não estar restrito somente a demonstrações foi visto como benefício da ferramenta criada.

As limitações impostas pela ferramenta, principalmente a restrição do uso do mouse, causou incômodo aos usuários. Não foi possível combater o *bot* inimigo da demonstração em igualdade de condições, o que levou o usuário à derrota na grande maioria dos confrontos. Mesmo não sendo relevante ao propósito da aplicação, o excessivo número de derrotas causou aborrecimentos e consequente rejeição ao convite a se realizar novas demonstrações para um conjunto menor de atributos, o que acredita-se facilitar a compreensão das atitudes tomadas pelo IVA gerado.

5.4 Comparativo com outras técnicas

Buscando apresentar os benefícios da proposta deste trabalho em relação a outras técnicas de programação para usuários finais já utilizadas para o mesmo fim, discute-se hipoteticamente a criação do comportamento de IVAs através da configuração de parâmetros, linguagens visuais baseadas em regras e programação por demonstração. Embora as linguagens de script e de fluxo de controle tenham, também, sido apresentadas como opções já disponibilizadas aos usuários finais para a realização da tarefa, estas estão muito aquém da facilitação das demais e, por isso, foram preteridas nesta análise.

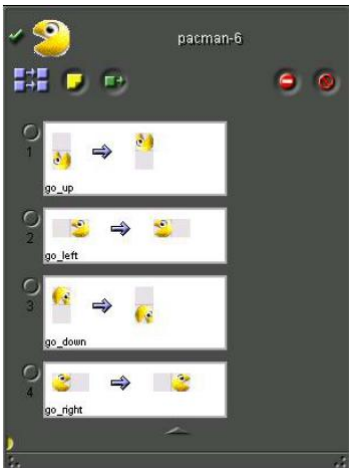
Sistemas que utilizam a configuração de parâmetros para o desenvolvimento de agentes trabalham tradicionalmente com comportamentos reativos. Os projetistas da aplicação preparam um conjunto de regras parametrizadas e estas regem as atitudes do IVA. A Figura 8 ilustra a abordagem. Quanto maior o número de regras mais realista tende a ser o agente e mais tempo será demandado para a implementação. Fica claro que esta alternativa não passa da disponibilização da camada de seleção de ações ao usuário final, também oferecida neste trabalho. Todavia, as customizações se resumem aos valores comparados nas expressões condicionais, destacado na Figura 8. Não é possível alterar a quantidade de regras, o que pode dificultar o entendimento, como já discutido anteriormente. Outro problema é a impossibilidade de criar novas regras com ações distintas, ou mesmo editar as regras pré-estabelecidas. Portanto, a configuração de parâmetros é nada além da possibilidade de alterar o conjunto padrão de valores de cada um dos atributos disponibilizado na biblioteca do estudo de caso, recurso que, embora não implementado, já faz parte da proposta deste trabalho. Logo, percebe-se que, a não necessidade de interferência manual do usuário em todas as regras, evitada pela demonstração, e a possibilidade de alteração das ações associadas às regras são melhorias relevantes da nova abordagem em relação a configuração de parâmetros.

```
if(distanceToEnemy()<= 10)
{
        ATACK_WITH_WEAPON(PISTOL)
}
else if (distanceToEnemy()> 10 AND distanceToEnemy()<= 50)
{
        ATACK_WITH_WEAPON(MACHINE_GUN)
}
else
{
        ATACK_WITH_WEAPON(RIFLE)
}
```

Figura 8: Exemplo de regra de sistemas que utilizam configuração de parâmetros para a especificação do comportamento de IVAs. Em negrito os parâmetros que podem ser modificados.

As linguagens visuais baseadas em regras, principalmente as que incorporam mecanismos de programação por demonstração, são as mais utilizadas na literatura para a especificação do comportamento de agentes. Por basearem-se na interface do VE no qual interagem, são mais intuitivas e consequentemente mais acessíveis aos usuários finais. Tradicionalmente, as linguagens visuais baseadas em regras trabalham com a interação do agente com o meio sem a necessidade de analisar propriedades internas do mesmo, como pode ser visto na Figura 9 (a). No caso da criação de um IVA, tal qual o desenvolvido neste estudo de caso, a análise de atributos é indispensável. Um exemplo desta manipulação na abordagem em questão pode ser vista na Figura 9 (b). Então, a criação das regras envolveria a análise de 7 atributos (os que foram disponibilizados na biblioteca), cada qual com três faixas de valores em média. Na validação realizada foram criadas, em média, 41 regras nos 6 minutos de duração. Suponha que um usuário decida criar tais regras em um sistema que utiliza linguagens visuais baseadas em regras. Ele deveria selecionar cada um dos atributos, as faixas de valores e o operador para compará-los, o que estima-se não levar menos de 30 segundos. Logo, em 6 minutos o usuário teria, possivelmente, especificado 12 regras, nem 30% do resultado obtido através da proposta deste trabalho. Não obstante, a regras estariam organizadas sequencialmente, dificultando o entendimento e também a eficiência da execução. Portanto, graças a possibilidade de demonstrar o comportamento, a técnica desenvolvida neste trabalho é muito mais eficiente, e devido a organização da camada de seleção de ações em árvore, mais compreensível.

Por sua vez, a criação de comportamentos através da programação por demonstração é a que mais se aproxima à abordagem desenvolvida neste trabalho. Em relação às propostas implementadas por Bimbo e Vicario (1995) e McDaniel e Myers (1998), o não ocultamento da camada de seleção de ações, permitindo criação, ou edição, de regras sem necessidade de nova demonstração, e a possibilidade de demonstrar o comportamento



(a) Regras sem análise de atributos. Fonte: Coura (2006).

(b) Regra com análise de atributos (destacado em vermelho). Fonte: Coura (2006) adaptado.

Figura 9: Exemplo de regras de sistemas que utilizam linguagens visuais baseadas em regras para a especificação do comportamento de IVAs.

jogando, e não em situações específicas, ou ciclos, incorporando o fator diversão à tarefa, agregam valor à abordagem deste trabalho. Por outro lado, acredita-se que o mecanismo de "dicas", disponível no trabalho de McDaniel e Myers (1998), poderia facilitar ainda mais o trabalho do usuário final. Todavia, optou-se por não implementá-lo, uma vez que este poderia comprometer o fator diversão presente na nova proposta.

6 Conclusões e Trabalhos Futuros

"*The solution of every problem is another problem.*"

Johann Wolfgang von Goethe

6.1 Conclusões

A possibilidade de testar hipóteses e experimentar a realidade sem as despesas e perigos do mundo real fez com que a utilização de ambientes virtuais se difundisse na comunidade científica. O sucesso da utilização desta tecnologia em áreas como a robótica e inteligência artificial, e a evolução dos recursos gráficos atraiu a atenção de diversos profissionais. Buscando facilitar o acesso destes novos usuários aos sistemas de simulação, técnicas de programação por usuários finais foram adotadas para a criação de aplicações, cada qual com seus pontos fortes e fracos que motivam sua adoção nas etapas do processo de produção de simulações.

Tão importante quanto interagir em ambientes realistas é lidar com personagens que transmitam a sensação de vida, seres "inteligentes". A definição da camada de seleção de ações é determinante para o realismo do personagem e pode ser identificada a partir da exemplificação por parte do usuário do comportamento que deve ser executado pelo agente. Diante da não exploração de tal característica pelas abordagens de EUP existentes, este trabalho apresentou uma nova proposta de programação por usuários finais para a especificação de agentes virtuais inteligentes, que permite a especificação do comportamento de agentes através de demonstração.

Incorporando os pontos fortes dos paradigmas de programação paramétrico, descritivo e imitativo, a solução proposta facilita o desenvolvimento de IVAs àqueles que gostariam de realizá-la e não o fazem por não possuírem conhecimentos de programação. O paradigma imitativo é a base da implementação da nova proposta. Demonstrando o comportamento através de um avatar imerso no ambiente virtual, o usuário poupa esforços na criação massante do vasto conjunto de regras que determinam o comportamento reativo do IVA.

Destoando das implementações de programação por demonstração encontradas na literatura, a solução proposta não encapsula a camada de seleção de ações, permitindo, também, que o usuário crie, ou edite, regras manualmente. A expressividade do paradigma descritivo motivou esta variante. A não necessidade de voltar ao modo de demonstração para fazer pequenas alterações na camada de seleção de ações facilita a realização da tarefa ao usuário, que pode realizar refinamentos no código de forma mais simples. Propõe-se a utilização de regras visuais antes-depois e árvores de decisão como forma de representação do conhecimento. Mesmo trabalhando no estudo de caso com regras textuais, os resultados foram excelentes. Muito desse êxito é creditado à opção pela organização das regras

em forma de árvores de decisão, o que torna a navegação entre as regras mais intuitiva e impede a geração de regras com variações na quantidade de atributos testados, não possibilitando, portanto, a ocorrência de regras mutuamente verdadeiras e dispensando a necessidade de políticas para o tratamento de conflitos. O sucesso alcançado na utilização das regras visuais até mesmo em sistemas voltados ao público infantil, leva a crer que estas, quando implementadas, simplificarão ainda mais a manipulação da camada de seleção de ações.

A adoção de características do paradigma paramétrico contribui para o melhor entendimento da camada de seleção de ações e disponibiliza ao usuário mais poder de customização. A ideia da biblioteca de atributos "dosa" a quantidade de elementos das regras geradas e possivelmente manipuladas pelo usuário, tornando-as mais compreensíveis. Devido ao poder de personalização dos agentes gerados por sistemas que implementam a configuração de parâmetros como forma de especificação do comportamento de IVA, decidiu-se incorporar este recurso à abordagem desenvolvida. Caso deseje, o usuário pode editar os valores padrão dos atributos selecionados, criando *bots* ainda mais críveis.

Embora a especificação do comportamento do IVAs através da demonstração já tenha sido proposta na literatura, o estudo realizado neste trabalho das diversas abordagens de EUP, voltadas, ou não, à tarefa em questão, possibilitou adaptações às técnicas existentes. Estas, supra-citadas, fazem com que a nova proposta, comparada aos mecanismos já utilizados para tal propósito, facilite ainda mais o acesso a sistemas de simulação a profissionais de diversas áreas, antes privados dos benefícios desta tecnologia.

O desenvolvimento do ambiente de testes como modificação do jogo de tiro em primeira pessoa Unreal Tournament comprova a possibilidade de utilização de *modding* como alternativa ao desenvolvimento de ambientes de simulação. A modularização dos jogos torna a tarefa tão simples quanto o desenvolvimento de um ambiente simplório e observou-se que trabalhar com recursos gráficos de última geração e em ambientes super interativos aumenta a sensação de imersão e motiva tanto desenvolvedores quanto usuários. Não obstante, alcançar os objetivos do trabalho em um ambiente realista leva a crer que a proposta deste trabalho pode ser aplicada a um grande leque de situações, ao contrário que poderia ser concluído se o projeto fosse conduzido sobre um VE discreto. Embora as limitações do ambiente de testes desenvolvido tenham sido alvo de críticas por parte dos usuários participantes da validação, sua implementação não está concluída e, com as devidas melhorias, espera-se que este poderá, sem dúvidas, ser utilizado como plataforma de testes de pesquisas em outras áreas, ou mesmo ser transformado em aplicação comercial.

As ferramentas utilizadas, Pogamut 2 e Unreal Tournament, atenderam perfeitamente às necessidades do projeto e os problemas encontrados foram facilmente contornados graças à vasta quantidade de material disponível online e sua ampla rede de usuários.

Lamenta-se o fato de ter-se desenvolvido o ambiente de testes como aplicação à parte da interface de desenvolvimento integrado do Pogamut 2. Sendo parte da IDE, este poderia ter sua utilização difundida e desenvolvimento partilhado junto a comunidade que faz uso do arcabouço, possibilitando maior contribuição do trabalho à comunidade científica. Este fato deveu-se ao reaproveitamento de parte do código previamente desenvolvido por alunos de iniciação científica do DPI-UFV. A migração para o formato da IDE demandaria tempo e esforço que poderia comprometer o andamento do trabalho.

Embora não tenham sido realizados testes formais, a validação da abordagem foi um sucesso. Sem terem sido informados do propósito da ferramenta desenvolvida, todos os usuários participantes foram capazes de criar IVAs a partir da demonstração, sendo que quando instruídos, alcançaram resultados ainda melhores. Percebeu-se que, mesmo utilizando algoritmos simples de extração do conhecimento, os resultados são visíveis, o que leva a crer que implementações mais complexas podem conduzir a melhores proventos. Conclui-se, portanto, que a especificação do comportamento de agentes virtuais inteligentes através da proposta desenvolvida neste trabalho é possível. Conclui-se, também, que para as situações experimentadas, a nova abordagem apresentada torna mais fácil a execução da tarefa a usuários finais.

6.2 Trabalhos Futuros

Embora este trabalho consista na proposta de uma nova abordagem de programação para usuários finais, a sua completa implementação é fundamental para a realização de testes e consequentes análises comparativas quantitativas em relação à satisfação e desempenho do usuário na utilização desta e das outras técnicas já aplicadas à mesma tarefa. A partir da implementação, comparações com variantes da proposta que sigam diferentes padrões de modelagem da camada de seleção de ações poderiam confirmar a esperada dificultação da compreensão desta pelos usuários finais.

O ambiente de testes concentra grande parte das sugestões de trabalhos futuros derivados deste projeto. Desenvolver uma interface intuitiva para a ferramenta e cercar o sistema contra erros de execução é o mínimo que deve ser feito. Posteriormente, buscando complementar as funcionalidades já discutidas, propõe-se finalizar a implementação

da edição de regras, permitir a continuação de demonstrações já iniciadas e implementar o algoritmo de otimização do posicionamento dos atributos nos níveis da árvore – de forma a definir qual o comportamento deve ser executado testando a menor quantidade possível de expressões.

Variantes do algoritmo de extração do conhecimento e funções de avaliação baseadas no sucesso da demonstração (i.e. ações que levaram a vitórias) possibilitariam a verificação das expectativas de melhora no quociente regras por intervalo tempo. O desenvolvimento de valores fuzzy para os atributos, embora comprometa o entendimento, pode gerar IVAs com comportamentos mais próximos dos demonstrados, uma vez que a avaliação humana está longe de ser binária.

Optou-se pela não implementação do mecanismo de "dicas", proposto no trabalho de McDaniel e Myers (1998), por acreditar que este poderia comprometer o fator diversão presente na nova proposta. Todavia, percebe-se agora que a incorporação deste como recurso extra, poderia agregar valor a solução, uma vez que, possibilitaria a criação de regras baseadas em observações "naturais" do usuário (i.e. mover-se para o lado mais escuro do mapa, já que não existem informações de luminosidade nos estados do agente ou do ambiente), e só seriam utilizadas quando o usuário julgasse necessário, ou seja, ele estaria disposto a parar de se divertir para demonstrar um comportamento com uma condição mais elaborada.

Como dito anteriormente, incorporar o ambiente à IDE do Pogamut 2 tende a difundir a utilização da proposta e permitir que outros pesquisadores contribuam com novas ideias ao projeto. Permitir que o usuário escolha entre políticas para a criação das regras que não foram identificadas através da demonstração, tais quais selecionar a ação do nó folha mais próximo ou determinar um comportamento padrão para essas situações, é visto como boa forma de evitar que o IVA gerado permaneça estático a maioria do tempo, indo em desencontro às expectativas do utilizador. Não obstante, a criação de um *wizard* que conduza o usuário de forma amigável durante a criação manual de regras facilitaria, também, a passagem por esta fase do processo.

Referências

ANASTASSAKIS, G.; RITCHINGS, T.; PANAYIOTOPOULOS, T. Multi-agent systems as intelligent virtual environments. In: *KI '01: Proceedings of the Joint German/Austrian Conference on AI*. London, UK: Springer-Verlag, 2001. p. 381–395. ISBN 3-540-42612-4.

ANGROS JR., R. *et al.* Learning domain knowledge for teaching procedural skills. In: *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2002. p. 1372–1378. ISBN 1-58113-480-0.

AYLETT, R.; CAVAZZA, M. Intelligent virtual environments - a state-of-the-art report. In: *Eurographics'01*. [S.l.: s.n.], 2001.

AYLETT, R.; LUCK, M. Applying artificial intelligence to virtual reality: Intelligent virtual environments. *Applied Artificial Intelligence*, v. 14, n. 1, p. 3–32, 2000.

BARBOSA, S. D. J. *Programação via Interface*. Tese (Doutorado) — Departamento de Informática–Pontifícia Universidade Católica do Rio de Janeiro, 1999.

BATES, J.; LOYALL, B.; REILLY, W. S. Broad agents. *SIGART Bull.*, ACM, New York, NY, USA, v. 2, n. 4, p. 38–40, 1991. ISSN 0163-5719.

BIMBO, A. D.; VICARIO, E. Specification by-example of virtual agents behavior. *IEEE Transactions on Visualization and Computer Graphics*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 1, n. 4, p. 350–360, 1995. ISSN 1077-2626.

BRAUN, A.; BODMANN, B. E. J.; MUSSE, S. R. Simulating virtual crowds in emergency situations. In: *VRST '05: Proceedings of the ACM symposium on Virtual reality software and technology*. New York, NY, USA: ACM, 2005. p. 244–252.

BROM, C. *Action Selection for Virtual Humans in Large Environments*. 2006. Doctoral Thesis Abstract – Faculty of Mathematics and Physics – Department of Software and Computer Science Education – Charles University in Prague.

BRYSON, J. J. Action selection and individuation in agent based modelling. In: *Proceedings of Agent 2003: Challenges in Social Simulation*. [S.l.]: Argonne National Laboratory, 2004. p. 317–330.

BURKERT, O. *et al.* Towards fast prototyping of ivas behavior: Pogamut 2. In: *Intelligent Virtual Agents*. [S.l.: s.n.], 2007. p. 362–363.

COLE, N.; LOUIS, S. J.; MILES, C. Using a genetic algorithm to tune first-person shooter bots. In: *Proceedings of the IEEE Congress on Evolutionary Computation*. New York, NY, USA: IEEE, 2004. v. 1, p. 139–145.

COURA, D. P. *Produzindo animações através da programação por demonstração.* Dissertação (Mestrado) — Departamento de Informática–Universidade Federal de Viçosa, 2006.

COURA, D. P. *et al.* Animaçoes através de programação por demonstração. In: *IHC '06: Proceedings of VII Brazilian symposium on Human factors in computing systems.* New York, NY, USA: ACM, 2006. p. 81–90. ISBN 1-59593-432-4.

CYPHER, A. *et al.* (Ed.). *Watch what I do: programming by demonstration.* Cambridge, MA, USA: MIT Press, 1993. ISBN 0-262-03213-9.

CYPHER, A.; SMITH, D. C. Kidsim: end user programming of simulations. In: *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems.* New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995. p. 27–34.

DONOVAN, J. *Cyberspace developer kit concepts and components.* [S.l.], 1993.

EL-NASR, M. S.; SMITH, B. K. Learning through game modding. *Comput. Entertain.*, ACM, New York, NY, USA, v. 4, n. 1, 2006. ISSN 1544-3574.

FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages.* London, UK: Springer-Verlag, 1997. p. 21–35. ISBN 3-540-62507-0.

GRAND, S.; CLIFF, D. Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers, Hingham, MA, USA, v. 1, n. 1, p. 39–57, 1998. ISSN 1387-2532.

HAGUE, R.; ROBINSON, P. End-user programming of reconfigurable systems: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, v. 36, n. 11-12, p. 1285–1306, 2006. ISSN 0038-0644.

JACOBSON, J.; HWANG, Z. Unreal tournament for immersive interactive theater. *Commun. ACM*, ACM, New York, NY, USA, v. 45, n. 1, p. 39–42, 2002. ISSN 0001-0782.

KAHN, K. Toontalk - an animated programming environment for children. *Journal of Visual Languages and Computing*, v. 7, n. 2, p. 197–217, 1996.

KAMINKA, G. A. *et al.* Gamebots: a flexible test bed for multiagent team research. *Commun. ACM*, ACM, New York, NY, USA, v. 45, n. 1, p. 43–45, 2002. ISSN 0001-0782.

KASAP, Z.; MAGNENAT-THALMANN, N. Intelligent virtual humans with autonomy and personality: State-of-the-art. *Intelligent Decision Technologies*, v. 1, n. 1-2, p. 3–15, 2007.

KELLEHER, C.; PAUSCH, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 37, n. 2, p. 83–137, 2005. ISSN 0360-0300.

KIM, I.-C. UTBot: A virtual agent platform for teaching agent system design. *Journal of Multimedia*, Academic Publisher, v. 2, n. 1, p. 48–53, 2007.

LAIRD, J. E. *et al.* A test bed for developing intelligent synthetic characters. In: *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*. [S.l.: s.n.], 2002. p. 52–56.

LIEBERMAN, H. (Ed.). *Your wish is my command: programming by example*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. ISBN 1-55860-688-2.

LOYALL, A. B. *et al.* System for authoring highly interactive, personality-rich interactive characters. In: *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2004. p. 59–68. ISBN 3-905673-14-2.

MAGERKO, B. *et al.* Ai characters and directors for interactive computer games. In: *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference*. [S.l.]: AAAI Press, 2004. p. 877–883.

MCDANIEL, R. G.; MYERS, B. A. Building applications using only demonstration. In: *IUI '98: Proceedings of the 3rd international conference on Intelligent user interfaces*. New York, NY, USA: ACM, 1998. p. 109–116.

MONTEIRO, I. M. *IAF - Indigente Agent Framework: Um Framework para o Desenvolvimento de Agentes Cognitivos em Jogos de Primeira Pessoa*. 2006. Monografia de Graduação – Depto. de Ciência da Computação – Instituto de Matemática – Universidade Federal da Bahia.

MYERS, B. A.; KO, A. J.; BURNETT, M. M. Invited research overview: end-user programming. In: *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2006. p. 75–80. ISBN 1-59593-298-4.

NARDI, B. A. *A small matter of programming: perspectives on end user computing*. Cambridge, MA, USA: MIT Press, 1993. ISBN 0-262-1405305.

PAUSCH, R. *et al.* Disney's aladdin: first steps toward storytelling in virtual reality. In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1996. p. 193–203. ISBN 0-89791-746-4.

PRADA, R. F. F. *Teaming Up Humans and Synthetic Characters*. Tese (Doutorado) — Instituto Superior Técnico–Universidade Técnica de Lisboa, 2005.

REPENNING, A. Agentsheets: a tool for building domain-oriented visual programming environments. In: *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*. New York, NY, USA: ACM, 1993. p. 142–143.

REPENNING, A. Agentsheets: an interactive simulation environment with end-user programmable agents. In: *INTERACT 2000: Proceedings of the IFIP Conference on Human Computer Interaction*. [S.l.: s.n.], 2000. p. 1–8.

REPENNING, A. Excuse me, i need better ai!: employing collaborative diffusion to make game ai child's play. In: *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*. New York, NY, USA: ACM, 2006. p. 169–178. ISBN 1-59593-386-7.

REYNOLDS, C. Steering behaviors for autonomous characters. In: *Proceedings of the Game Developers Conference 1999*. [S.l.]: Miller Freeman Game Group, 1999. p. 763–782.

RICKEL, J.; JOHNSON, W. L. Extending virtual humans to support team training in virtual reality. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 217–238, 2003.

RIEDL, M.; SARETTO, C. J.; YOUNG, R. M. Managing interaction between users and agents in a multi-agent storytelling environment. In: *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2003. p. 741–748. ISBN 1-58113-683-8.

SHAW, C. *et al.* Decoupled simulation in virtual reality with the mr toolkit. *ACM Trans. Inf. Syst.*, ACM, New York, NY, USA, v. 11, n. 3, p. 287–317, 1993. ISSN 1046-8188.

SILVA, S. R. P. da. *Um modelo semiótico para programação por usuários finais*. Tese (Doutorado) — Departamento de Informática–Pontifícia Universidade Católica do Rio de Janeiro, 2001.

SMITH, D. C. Building personal tools by programming. *Commun. ACM*, ACM, New York, NY, USA, v. 43, n. 8, p. 92–95, 2000. ISSN 0001-0782.

SMITH, D. C.; CYPHER, A.; TESLER, L. Programming by example: novice programming comes of age. *Commun. ACM*, ACM, New York, NY, USA, v. 43, n. 3, p. 75–81, 2000. ISSN 0001-0782.

THALMANN, N. *Artificial Life and Virtual Reality*. New York, NY, USA: John Wiley & Sons, Inc., 1994. ISBN 0471951463.

TORRES, D.; BOULANGER, P. The animus project: a framework for the creation of interactive creatures in immersed environments. In: *VRST '03: Proceedings of the ACM symposium on Virtual reality software and technology*. New York, NY, USA: ACM, 2003. p. 91–99. ISBN 1-58113-569-6.

VOSINAKIS, S.; PANAYIOTOPOULOS, T. A tool for constructing 3d environments with virtual agents. *Multimedia Tools Appl.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 25, n. 2, p. 253–279, 2005. ISSN 1380-7501.

WRIGHT, S. *et al.* A framework for supporting intelligent traffic within the leeds driving simulator. In: *Proc. Workshop on Intelligent Virtual Environments of ECAI98*. [S.l.: s.n.], 1998.

YUCEL, I.; ZUPKO, J.; EL-NASR, M. S. It education, girls, and game modding. *International Journal of Interactive Technology and Smart Education Journal: Special Issue on Smarter Use of Technology in Education*, v. 3, n. 2, 2006.