Murilo Cerone do Nascimento
Tássio Naia dos Santos

Arquitetura *BDI* para agentes e *Blackboard*: um estudo de caso de técnicas de IA aplicadas a jogos

São Paulo
2010

Murilo Cerone do Nascimento
Tássio Naia dos Santos

Arquitetura *BDI* para agentes e *Blackboard*: um estudo de caso de técnicas de IA aplicadas a jogos

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

Área de Concentração:
Engenharia de Computação

São Paulo
2010

Murilo Cerone do Nascimento
Tássio Naia dos Santos

Arquitetura *BDI* para agentes e *Blackboard*: um estudo de caso de técnicas de IA aplicadas a jogos

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

Área de Concentração:
Engenharia de Computação

Orientadores:
Ricardo Nakamura
Roberto Cezar Bianchini

São Paulo
2010

. . . no amor e para graduar!

“O resultado de qualquer pesquisa séria só pode ser fazer duas questões crescerem onde antes apenas crescia uma.”

Thorstein Veblen
Sociólogo e economista
noruego-estadunidense
(1853 – 1929)

Resumo

Falar da motivação para uso de técnicas de IA em jogos, e do potencial que o BDI e o quadro-negro têm de agregar ao realismo da experiência dos jogos. (Citar artigos falando do emprego dessas técnicas.)

Falar do intuito do estudo de caso, do que se pretende (e como) quantificar. Mencionar as maiores escolhas e como imaginamos que elas delinearam os resultados. (E instigar o leitor à leitura do restante da tese!)

Palavras-chave: Engenharia. Engenharia da Computação. Inteligência Artificial. Arquitetura BDI. Sistema Blackboard. Jogos.

Abstract

Same text (*ibidem idem* previous section), but now in English.

Keywords: Engineering. Computer Engineering. Artificial Intelligence. BDI Architecture. Blackboard System. Games.

Lista de Figuras

Sumário

Capítulo 1

Introdução

Motivação e referências básicas do trabalho.

Quem são os orientadores, e como viemos a escolher o tema do trabalho. Falar aqui dos cursos de Computação Gráfica, Design e Programação de Games, SBGames.

Capítulo 2

Aspectos Conceituais

2.1 Inteligência Artificial

Um pouco de história, referências básicas sobre o assunto, usos mais comuns, estado da arte.

2.2 Jogos digitais

Uma curta evolução histórica, tendências atuais (puxando um pouco para os problemas de IA — a questão da imprevisibilidade e adaptabilidade dos jogos). Referências.

2.3 Inteligência em jogos

O principal objetivo do uso de técnicas de inteligência artificial em jogos eletrônicos é a diversão.

Vou falar que aqui não interessa muito o como a coisa é feita mas sim o resultado, ou seja, não interessa ao desenvolvedor se preocupar com a teoria de IA mas sim em criar uma situação que forneça ao jogo mais diversão. Um exemplo interessante para deixar claro a diferença entre IA acadêmica e IA em jogos são os shooters. Em jogos de tiros poderíamos fazer (através de técnicas de IA) os NPCs acertarem todos os tiros que disparassem na cabeça do inimigo, no entanto, isso passa longe da realidade humana, fato que poderia tirar toda a diversão de um jogo!

Um pouco de história, motivação do uso, desafios e problemas comuns.

2.3.1 Crença, desejo e intenção — a arquitetura BDI

O modelo BDI (*Beliefs Desires Intentions*) foi originalmente proposto por Bratman como uma teoria filosófica do raciocínio prático, propondo uma análise do comportamento humano que seria baseado em crenças, desejos e intenções. Basicamente supõe-se que as ações são derivadas a partir de um processo chamado raciocínio prático. Este processo é constituído por duas etapas, na primeira, deliberação, o agente seleciona um conjunto de desejos que devem ser alcançados, de acordo com a situação atual das crenças do mesmo. Na segunda etapa ocorre a determinação de como os desejos produzidos no

passo anterior podem ser atingidos através do uso dos meios disponíveis ao agente **[4] Michael Wooldridge, Reasoning about Rational Agents. Cambridge, MA: The M. I. T. Presss 2000.**. A seguir explicaremos melhor o que são crenças, desejos e intenções.

- **Crenças (Beliefs)**: Representam as características do ambiente e são atualizadas constantemente. Podem ser vistas como a componente informativa do sistema.
- **Desejos (Desires)**: Representam os objetivos a serem alcançados. Podem ser vistos como motivações do sistema.
- **Intenções (Intentions)**: Representam o atual plano de ações escolhido.

A partir deste modelo nasceu a arquitetura BDI para agentes. Agentes BDI são sistemas localizados em um ambiente (em nosso caso o ambiente será virtual) sujeito a variações, além disso, percebem o estado deste ambiente constantemente e podem atuar sobre o mesmo para tentar alterar o estado atual. Abaixo vemos o processo de raciocínio prático de um agente BDI.

Figura 2 - Diagrama de uma arquitetura BDI genérica [4]

Como se pode observar existe sete elementos básicos que compõem um agente BDI, são eles **[5] Ingrid Oliveira de Nunes, Implementação do modelo e da arquitetura BDI**:

- Um conjunto de crenças (*Desires*) atuais que representam as informações que o agente tem do ambiente.
- Uma função de revisão de crenças (*Belief Revision Function*), a qual determina um novo conjunto de crenças a partir da percepção da entrada e das crenças do agente.
- Uma função de geração de opções (*Option Generation Function*), a qual determina as opções disponíveis ao agente (seus desejos), com base nas suas crenças sobre seu ambiente e nas suas intenções.
- Um conjunto de opções (*desires*) corrente que representa os possíveis planos de ações disponíveis ao agente.
- Uma função de filtro (*filter*), a qual representa o processo de deliberação do agente, que determina as intenções do agente com base nas suas crenças, desejos e intenções atuais.
- Um conjunto de intenções (*Intentions*) atual, que representa o foco atual do agente, isto é, aqueles estados que o agente está determinado a alcançar.

- Uma função de seleção de ação (*Action Selection Function*), a qual determina uma ação a ser executada com base nas suas intenções atuais.

2.3.2 Quadro-negro — blackboard

A forma mais simples de apresentar o conceito de sistemas blackboard é através de uma metáfora que propõem a seguinte situação **(Daniel D. Corkill, Blackboard Systems. Blackboard Technology Group, Inc.)**: "Imagine um grupo de cientistas reunidos em uma sala trabalhando de forma cooperativa para resolver um problema. Para chegar a uma solução é utilizado um quadro negro (*blackboard*). A resolução do problema começa quando o mesmo é escrito no quadro negro juntamente com informações iniciais. Os cientistas verificam o conteúdo do quadro e aguardam uma oportunidade para aplicar seus conhecimentos visando solucionar o problema. Quando um cientista encontra informações suficientes para fazer uma contribuição, o mesmo coloca sua contribuição no quadro negro, e com sorte, este processo ativará outro cientista e assim por diante até chegarem à solução do problema." Da metáfora acima se pode concluir que um sistema blackboard possui uma base de dados comum a diversos outros sistemas que estão tentando solucionar um problema e utilizam e atualizam as informações existentes nesta base de dados. A seguir apresentaremos algumas características deste tipo de sistema:

- **Independência de conhecimento**: No exemplo acima, supomos que cada cientista adquiriu seus conhecimentos de maneira independente dos outros, ou seja, num sistema blackboard cada sistema que participa da solução de um problema tem seus próprios níveis de conhecimento, independentemente dos outros sistemas.

- **Diversidade nas técnicas de solução de problemas**: Uma das grandes vantagens do sistema blackboard é que não interessa como os sistemas que estão trabalhando na resolução do problema funcionam, ou seja, um sistema pode utilizar redes neurais outro pode utilizar estar utilizando simulações que para o sistema blackboard estas "fontes de conhecimento" são caixas pretas que fazem suas contribuições.

- **Representação da informação é flexível**: Em sistemas blackboard não existe nenhuma definição de como deve ser a informação, dando liberdade a quem faz o sistema de definir como representar os dados.

- **Linguagem comum entre os sistemas**: Ao mesmo tempo em que existe a flexibilidade na representação da informação, é necessário, por outro lado, que todos os sistemas envolvidos sejam capazes de entender tais informações.

- **Liberdade de organização dos dados**: A organização dos dados é livre, porém, deve ser feita de forma eficiente de tal forma que, no caso da base de dados possuir muita informação, seja simples para um sistema encontrar informações específicas de forma rápida e simples.

- **Ativações baseadas em eventos**: Neste tipo de sistema, os sistemas que estão trabalhando na solução do problema não se comunicam diretamente, ao invés disto, todos "observam" a base de dados comum e utilizando as informações da mesma buscam gerar novos dados que são colocados na mesma base, tais dados podem ativar outro sistema que fará a mesma coisa até que o problema seja completamente resolvido.

- **Necessidade de controle**: É preciso haver um "órgão" para controlar todos os sistemas e decidir quem deve e quem não deve poder alterar os dados do quadro negro, assim não existe o risco de mais de um sistema alterar uma mesma área de dados simultaneamente.

- **Geração de solução incremental**: É evidente que a solução de um problema neste tipo de sistema acontece de passo em passo, ou seja, um sistema faz uma contribuição com novos dados, a seguir, outro sistema utiliza tais dados e faz uma nova contribuição e de ciclo em ciclo o problema tende a ser resolvido.

Abaixo temos uma visão geral de como funciona o sistema blackboard. Basicamente existem fontes de conhecimento (sistemas que estão trabalhando na resolução do problema) um sistema de controle que concede permissões às fontes de conhecimento para alterarem os dados do quadro negro, este por sua vez, é uma base de dados que contem informações que ficam disponíveis a todas as fontes de conhecimento.

Figura 1- Visão geral de um sistema blackboard

Neste projeto utilizaremos este tipo de sistema como parte de um jogo computacional que será desenvolvido.

2.3.3 O que avaliar

O que o estudo de caso vai avaliar, e como. Decisões de projeto que validam o estudo (C++, *data-driven* design, prototipação, engenharia de software), isto é, porque o projeto permite extrapolar conclusões para jogos na indústria.

Capítulo 3

Especificação

Nesta seção descrevemos os critérios de aceitação do projeto:

- descrição das telas,
- opções esperadas dos menus,
- comportamento esperado dos NPCs

O Game Design, que é um dos anexos da monografia, complementa a especificação do projeto.

Capítulo 4

Metodologia

O sucesso de um projeto depende em grande medida da capacidade de adaptação dos envolvidos. Nesta seção tratamos do método de planejamento de atividades to trabalho, e de sua evolução no decorrer do projeto. Assim, em certa medida trataremos das mudanças que se operaram nas expectativas e previsões que fizemos ao longo do tempo.

4.1 Evolução

1. Levantamento de atividades e assuntos a pesquisar, cronograma preliminar (de pesquisas), criação de diário de bordo.
2. Decisões de projeto iniciais (condições de contorno): ferramentas e tecnologias de desenvolvimento:
   - escolha de linguagem de presença expressiva na indústria: C++;
   - definição de licensa livre para o projeto;
   - opção por jogo 2D;
   - pesquisa de simuladores BDI;
   - identificação de necessidades de renderização;
   - pesquisa de engines livres.
3. Divisão do projeto em escopos de conhecimento. C++, Java, BDI, AgentSpeak, SDL, Arquitetura de software para jogos.
4. Cronograma refinado (com um mês de folga):
   - escrita do documento de especificação game-design document (GDD);
   - previsão de tempos de implementação, busca de conteúdo artístico, e modelagem;
   - especificação de testes.
5. Metodologia de desenvolvimento: controle de versão, práticas de engenharia de software (referências), prototipação das funcionalidades básicas.

4.1.1 Percalços e reajustes

O que não ocorreu como previsto, e os ajustes que foram feitos. (Falar em termos mais genéricos, o específico será dito na história de evolução do projeto.)

Complexidade da modelagem de agentes (?)

Capítulo 5

Implementação

5.1 Visão global

Como as coisas funcionam. O que faz o quê (macroscopicamente).

Uma explicação sucinta da estrutura do programa, como ele inicia (forking) e como age em regime (responsabilidades java vs C++, e o protocolo).

5.2 Arquitetura

5.3 Protocolo de comunicação

5.4 Estrutura de pastas

O projeto foi desenvolvido com vistas a se prestar à experimentação no design. Assim, algumas ferramentas para a escrita de *scripts* de diálogos ou pequenas cenas foram desenvolvidas.

Descrevemos a seguir a organização dos arquivos do projeto. É importante que essa estrutura seja clara e intuitiva, já que é nossa intenção que alguns desses arquivos (que configuram o comportamento do jogo) sejam alterados primariamente por designers do jogo. Assim, é preciso que haja critério na complexidade que se expõe a esses "usuários".

5.5 Arquivos auxiliares

Uma série de arquivos de texto auxiliares são usados pelo jogo. Eles especificam

- diálogos,
- agentes,
- estados de agentes,
- tipos de estímulos a agentes,

Explicamos a seguir a função de cada um dos tipos de arquivo.

Os *diálogos* são scripts que codificam possíveis dizeres que NPCs ou o jogador podem optar por dizer. Carregam informação sobre o sentimento que expressam ou a impressão que causam. Seguem a gramática descrita no apêndice A.

Agentes são especificados em *AgentSpeak*, a linguagem interpretada pela ferramenta *Jason*. Cada arquivo contém a modelagem de um tipo de agente.

Os três últimos itens são arquivos de interface, usados para leitura pelos processos de renderização e de interpretação do sistema BDI. Optou-se por essa solução porque a comunicação se dá pelo envio de números, que, nesse caso, indicam qual a linha do arquivo de interface em que a mensagem está contida. Obviamente, essa abordagem só é possível porque as mensagens são conhecidas *a priori*.

Para comunicar o estímulo que uma determinada fala no diálogo provoca no agente, a lógica do jogo envia ao sistema BDI um inteiro indicando a linha em que está escrita a string que define o estado. Na prática, a lógica do jogo faz uma chamada para enviar o inteiro correspondente ao estímulo que se deseja comunicar, e uma busca é feita no arquivo que contém os estímulos possíveis para encontrar o número da linha correspondente. Esse número é, por fim, enviado ao sistema BDI, que fará a sua própria busca para identificar qual o estímulo recebido.

No caso de funções, o arquivo de interface deve ser gerado automaticamente pelo código Java, para que seja consultado durante a execução pela lógica do jogo.

Capítulo 6

Testes

Capítulo 7

Conclusão

Sucesso do estudo de caso?

7.1 Trabalhos futuros

Próximo SBGames?

7.2 *Post Mortem*

7.2.1 *"Lessons learned"*

Apêndice A

Gramática dos *scripts* de diálogo

Descrevemos a seguir a gramática da linguagem de especificação de diálogos.