# Uma Arquitetura Modular para Desenvolvimento de Agentes Cognitivos em Jogos de Primeira e Terceira Pessoa

Ivan Medeiros Monteiro

Universidade Federal da Bahia – Instituto de Matemática
fehler@im.ufba.br

**Resumo**

*Em jogos de primeira e terceira pessoa é freqüentemente necessária a existência de agentes capazes de emular comportamentos humanos. Tais agentes são denominados BOTs. No presente trabalho, introduzimos uma nova arquitetura para o desenvolvimento de agentes autônomos cognitivos. Esta arquitetura visa simplificar o processo de modelagem, implementação e reutilização de BOTs.*

***Palavras Chave:*** *Inteligência Artificial, Agentes, Arquitetura de Agentes.*

## 1 Introdução

Jogos desempenham um importante papel nos campos relacionados à inteligência artificial. Isto porque os jogos podem ser utilizados como laboratórios para testar tanto teorias sobre o raciocínio humano quanto métodos de resolução de problema[1]. Este é o caso do desenvolvimento de BOTs, foco deste trabalho. Em jogos, os BOTs devem atuar como jogadores virtuais dotados de inteligência artificial, tentando, em especial, emular seqüências de comportamentos dos jogadores humanos.

Cada jogo possui características específicas que requerem uma implementação única dos componentes inteligentes. Entretanto, é possível identificar características comuns que são compartilhadas por diferentes gêneros de jogos [15], em especial jogos de primeira e terceira pessoa. A identificação de similaridades encontradas em jogos de primeira e terceira pessoa é um dos fatores motivadores para a criação de modelos genéricos capazes de suportar as necessidades de jogos em ambos os gêneros.

Além da identificação de similaridades, tais modelos genéricos devem contemplar a necessidade de realismo para o comportamento dos BOTs nos jogos atuais. De fato, tais jogos requerem uma maior aproximação do comportamento do BOTs com o raciocínio e as ações humanas. Assim, faz-se necessária uma arquitetura de agentes híbridos para conquistar tal aproximação, visto que tais arquiteturas oferecem o potencial esperado[4].

Este trabalho introduz uma nova arquitetura para agentes autônomos cognitivos, otimizada para o desenvolvimento de BOTs empregados em jogos de primeira e terceira pessoa. Tal arquitetura, combina elementos existentes em diversas outras arquiteturas[1][2][3][4][5][7][10], focalizando o caráter modular.

A arquitetura introduzida neste trabalho vem sendo utilizada, com sucesso, como base para o desenvolvimento do subsistema de inteligência artificial do InGE[11], um motor para jogos 3D desenvolvido pelo grupo Indigente[12].

A seção 2 deste artigo comenta sobre trabalhos relacionados. Na seção 3 é feita a apresentação da arquitetura proposta. Um estudo de caso através de um exemplo simples é visto na seção 4. A seção 5 levanta algumas questões sobre implementação e na seção 6 são feitas algumas considerações.

## 2 Trabalhos Relacionados

Uma arquitetura bastante utilizada para implementação de BOT é a Arquitetura de Subsunção[7], que se baseia em máquinas de estado finitos para montagem de controladores reativos. O fato da Arquitetura de Subsunção modelar um agente puramente reativo, a torna

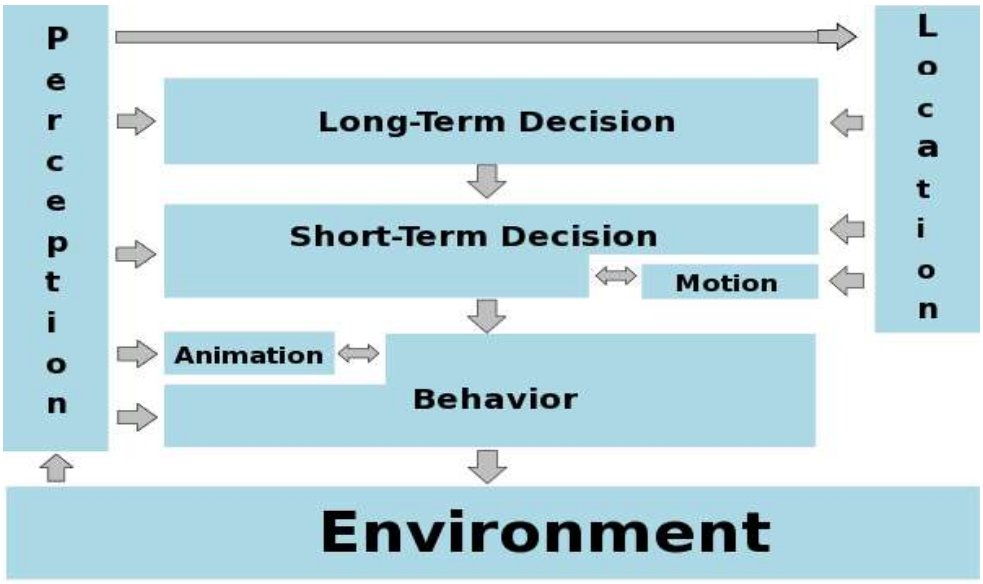


Figura 1: A arquitetura

imprópria para agentes deliberativos ou híbridos. Mesmo assim, existem trabalhos, como o de Yiskis [10], que extendem a Arquitetura de Subsunção dentro do contexto de jogos.

Algumas arquiteturas, como a apresentada por O'Brien[3], enfatizam o planejamento em detrimento a reatividade. Estas arquiteturas, apesar de oferecerem bons mecanismos para decisões estratégicas, não são adequadas para implementação de jogos de primeira e terceira pessoa, que requerem um alto nível de reatividade.

Assim, a necessidade de implementação de BOTs convincentes, que atuem no ambiente igualmente aos jogadores humanos, focaliza este trabalho em arquiteturas híbridas, que mesclam reação e deliberação[4][5][1]. Entretanto, o problema que envolve algumas dessas arquiteturas híbridas[4][5] é o fato delas não estarem definidas dentro do contexto de jogos. A desvinculação com a aplicação, apesar remeter a maior flexibilidade e generalização, na prática, impõe dificuldades na modelagem e no desenvolvimento de um BOT. Isto porque estas arquiteturas deixam de fazer uso de diversas otimizações possíveis, oferecidas pelo contexto do jogo. Desconsiderar tais otimizações torna os processos de desenvolvimento do BOT bastante árduo.

Uma arquitetura que atende bem a utilização de otimizações que o jogo pode oferecer é a arquitetura utilizada nos BOTs do jogo QuakeIII [1]. Entretanto, esta arquitetura possui deficiência no planejamento, que é a peça fundamental para que um BOT emule seqüências de comportamentos dos jogadores humanos.

Este trabalho, introduz uma arquitetura híbrida que consegue tirar proveito de uma série de características que são comuns a jogos de primeira e terceira pessoa através da criação de módulos especializados para o contexto de jogos. Esta especialização torna a implementação destes módulos bastante reutilizáveis, uma vez que as funcionalidades implementadas são comuns para vários BOTs.

## 3 A arquitetura

A arquitetura de agente cognitivo apresentada aqui é composta por sete módulos:

- *Perception* - Módulo de eventos responsável pela filtragem dos sensores do agente.
- *Location* - Centraliza informações sobre o modelo do ambiente para o agente.
- *Long-Term Decision* - Responsável pelo planejamento.
- *Short-Term Decision* - Responsável pela execução do plano atual.
- *Motion* - Manipula aspectos de roteamento, colisão e desvios de obstáculos.
- *Animation* - Determina a animação a ser exibida.
- *Behavior* - Atua sobre o ambiente de forma reativa.

Dos sete módulos apresentados, cinco são responsáveis pelo processo de decisões do agente, são eles: *Long-Term Decision*, *Short-Term Decision*, *Motion*, *Animation*, *Behavior*. Os módulos de decisão estão agrupados em 3 camadas:

- Camada Deliberativa - Responsável por gerar soluções globais para tarefas complexas, composta pelo módulo *Long-Term Decision*.
- Camada Executiva - Responsável pelo sequenciamento do plano passado pela camada deliberativa, composta pelos módulos *Short-Term Decision* e *Motion*.
- Camada Reativa - Responsável por responder as percepções do ambiente, composta pelos módulos *Animation* e *Behavior*.

O fluxo de informações através dos módulos é indicado na Figura 1. Os dados captados do ambiente seguem para o módulo *Perception* que os filtra e os enviam para os módulos responsáveis. *Perception*, após a filtragem, envia informações para *Location*, *Behavior*, *Animation*, *Short-Term Decision* e *Long-Term Decision*. Com a informação recebida de *Perception*, *Location* atualiza o modelo de ambiente do agente e disponibiliza, de forma síncrona com Perception, consultas a este modelo de ambiente para os módulos, *Long-Term Decision*, *Short-Term Decision* e *Motion*. Baseado no modelo atualizado do ambiente e nas percepções, *Long-Term Decision* utiliza sua representação interna de conhecimento para gerar planos que atendam aos objetivos do agente. Estes planos são passado para o *Short-Term Decision* que fica responsável pela execução dos mesmos. Para isto, *Short-Term Decision* utiliza as percepções e o modelo atualizado do ambiente juntamente com o módulo *Motion*, para decidir como executar cada tarefa de um dado plano. A execução das tarefas é feita trocando o comportamento reativo atual do módulo *Behavior*. Assim, *Behavior* atua sobre o ambiente baseado em suas percepções e o comportamento ativo atual. *Animation* e *Motion* são sub-módulos especializados de *Behavior* e *Short-Term Decision,* respectivamente. *Animation* comunica-se com o *Behavior* para decidir qual animação exibir e pode também receber percepções para gerar novas animações. Já o *Motion* é usado para decidir sobre a navegação do agente no ambiente.

### 3.1 Perception

Todos os dados captados por sensores do agente são tratados como percepções. Estes sensores podem ser dos mais variados tipos, imitando alguns dos sentidos humanos, sensores de visão, audição, tato e olfato. Cada percepção recebida deve ser repassada para os módulos apropriados. Assim, *Perception* centraliza o recebimento de percepções, cria um modelo de representação interna dessas percepções e filtra o envio dessas para cada módulo.

Os sensores visuais são responsáveis por captar os estímulos visuais junto com suas informações. Estas informações devem conter necessariamente uma representação geográfica do objeto causador do estimulo e adicionalmente características deste objeto. Uma representação geográfica para o estimulo visual, tanto poderia ser as coordenadas esféricas do objeto, em uma representação local, como poderia ser as coordenas cartesianas, em uma representação global. A quantidade de informação sobre o objeto causador do estimulo pode ainda ser controlada com base na distância que o objeto está do sensor.

Os sensores auditivos são responsáveis por captar os sons do ambiente. Os sons podem agregar informações semelhantes aos sensores visuais. Entretanto, a representação geográfica de um som, apesar de importante em diversas situações, pode ser dispensada. As demais informações sobre o som, como o tipo de som, o que pode ter causado este som, são elementos importantes na representação do som. Para sensores que possuem representação geográfica do som, a oclusão de sons também pode ser implementada. Assim, um sensor que esteja próximo a algo que produza bastante ruido, pode não conseguir captar devidamente os demais sons produzidos no ambiente.

Os sensores de tato são responsáveis por captar tudo o que se sente através de contato. Assim, a informação obtida através de um sensor de toque pode ser desde uma simples percepção de colisão com uma parede do cenário, até a percepção de danos causados por projéteis e

explosões.

Os sensores de olfato seguem a mesma idéia do sensores auditivos. Eles oferecem uma via alternativa para captar percepções. Um problema existente da implementação de sensores de olfato é que atualmente este representa um desbalanceamento em comparação com um jogador humano. Isto porque, até a data de escrita deste material, dispositivos de saída que emite odores não são comuns entre jogadores.

O módulo *Perception* agrupa cada sensor, de forma a gerenciar a captação de percepções. Este gerenciamento, consiste em gerar informação a partir dos dados captados pelos sensores e também filtrar e encaminhar as percepções para os módulos devido. Um exemplo simples é que, o som ambiente de que está ventando muito gera uma percepção auditiva que apesar de não ser muito útil para *Behavior*, pode ser interessante para *Short-Term Decision* e/ou *Long-Term Decision*.

### 3.2 Location

É comum que a representação do ambiente, para agentes híbridos de três camadas, esteja agregada ou a camada deliberativa ou a camada executiva. Este trabalho separa a representação do ambiente dos módulos de decisão, por entender que ele é transversal a outros módulos e também por facilidade na reutilização com outros agentes.

O módulo *Location* possui uma representação do ambiente de modo a facilitar questões como determinação de rotas, navegação, desvio de obstáculos e localização de objetos no ambiente. Em jogos, é comum gerar essa representação de maneira pré-processada, diminuindo assim, o processamento em tempo de execução. Com isto, para o agente, *Location* atua como um grande oráculo sobre o ambiente, sendo atualizado diretamente por conjuntos de percepções.

Dentre as representações utilizadas em *Location*, um sistema de *waypoint* é o mais comumente adotado para o propósito de roteamento e navegação em ambientes 3D. O estudo de caso deste trabalho também leva em consideração um sistema de *waypoint* para representar *Location*. Isto porque este sistema resulta em um grafo conexo dos locais acessíveis. Assim, é possível viajar de um *waypoint* para qualquer outro se existe ligação entre eles.

O fato de separar a representação do ambiente em um módulo a parte simplifica o desenvolvimento do agente, fazendo com que o problema da representação seja resolvido em um único local e não fique espalhado por vários módulos de decisão do agente.

### 3.3 Long-Term Decision

Ganhar o jogo é o mais importante objetivo para um BOT. Atingir esse objetivo significa tomar uma série de decisões que conduza o jogo ao estado de vitória para este BOT. *Long-Term Decision* é o módulo responsável pelas decisões de mais alto nível para o agente. Este módulo utiliza modelos de tomada de decisão para gerar soluções globais para tarefas complexas. A principal atribuição para este nível é o planejamento.

O agente pode utilizar diversos modelos para representar o conhecimento que irá compor sua base de conhecimento. Esta base é utilizada pelo agente para definir novas metas do agente. A atualização desta base de conhecimento é feita tanto pelas percepções vindas de *Perception* quanto pelas informações do ambiente vindas de *Location*.

Cada meta do agente está associada com um conjunto de planos. Um conjunto de planos, agrupa vários planos possíveis para chegar a uma meta. Cada plano define uma seqüência de tarefas para se atingir um determinado objetivo. Assim, dada uma meta e seu conjunto de planos correspondente, qualquer plano dentro desse conjunto fornece uma seqüência para atingir o objetivo. Entretanto, apenas um dos planos do conjunto deve ser executado.

A decisão de qual melhor plano, dentro de um conjunto de planos, deve levar em consideração o estado atual do jogo. Por exemplo, em um jogo FPS de *Capture The Flag*, se o objetivo é recuperar a bandeira que acabou de ser roubada pelo adversário, um plano que leve em consideração uma emboscada na base adversária, pode ser preferível que uma perseguição direta ao inimigo, se o BOT não possui pontos de vida o suficiente para combate direto.

Assim, *Long-Term Decision* fica responsável por decidir qual a meta atual e o melhor plano para

atingir a esta meta. Este plano é enviado para *Short-Term Decision*, que fica encarregado de executar tal plano. Com isto, *Long-Term Decision* concentra-se em tipos de soluções estratégicas, delegando a execução dos planos para a camada inferior.

### 3.4 Short-Term Decision

*Short-Term Decision* atua como integrador das decisões tomadas pelo *Long-Term Decision* e as ações executadas pelo *Behavior*. Dado um plano passado para *Short-Term Decision*, este módulo fica responsável por decidir como executar o plano recebido. Este módulo também possui a liberdade de acrescentar ao plano, tarefas que otimizem o estado do agente e não desvie do objetivo do plano. A execução de tarefa dentro de um plano é feita selecionando os comportamentos dentro de *Behavior*.

O agente pode acrescentar tarefas em um dado plano que esteja executando. Ainda no exemplo de resgate da bandeira, um BOT poderia incluir como uma tarefa pegar p*owerup* ao perceber que está passando próximo a algum. Situações como essa permite ao BOT otimizar seu estado atual, sem se desviar do objetivo do plano.

*Short-Term Decision* tem acesso direto a *Motion* porque todas as informações de navegação, roteamento e desvio de obstáculos necessitam ser manipuladas pelo próprio *Short-Term Decision*. As definições de melhor caminho a ser tomado e o seqüenciamento de cada passo são feitas no momento da execução das tarefas que utilizam a navegação.

Dada uma seqüência de tarefas a ser executada, *Short-Term Decision* decide qual o comportamento ativar para a execução de cada uma destas tarefas. Uma tarefa pode conter um ou mais comportamentos seqüenciados. Dessa forma, *Short-Term Decision* atua sobre *Behavior* como um seletor do comportamento atual.

### 3.5 Motion

*Motion* é o responsável por entender como o agente deve se movimentar no mundo. Ele recebe informação de *Short-Term Decision* indicando para onde deve se movimentar e então decide a forma apropriada de mover para o destino. Este módulo não executa a movimentação, apenas determina como executar. Toda decisão tomada por este módulo é computada com base nas estruturas de *Location*. *Motion* acha uma série de pontos que juntos compreendem o caminho da origem para o destino.

*Motion* é utilizado quando *Short-Term Decision* necessita de informações sobre navegação, roteamento e desvio de obstáculos. Assim, *Motion* trabalha em série com a execução das tarefa de *Short-Term Decision*. Se existe uma tarefa em que é necessário seguir um personagem, cada ciclo de execução da tarefa precisa consultar *Motion*, e assim determinar a melhor forma do BOT acompanhar o personagem alvo.

Como *Location* comumente utiliza sistema de *waypoint*, este módulo torna possível uma representação através de grafos. Tal representação possibilita a utilização de diversos algoritmos clássicos sobre este sistema de *waypoint*. Assim, quando é preciso atravessar diversos *waypoints* para chegar ao destino, é possível utilizar qualquer algoritmo para achar caminho em um grafo, por exemplo o A* , executando assim o roteamento.

A navegação pode ser local ou global. Navegaçào local visa a movimentação simples dentro de um mesmo *waypoint* e a global utiliza o roteamento de *waypoint* para reduzir cada etapa em uma navegação local. Volumes convexos são freqüentemente utilizados para definir um *waypoint*. Neste caso a navegação local se resume a livre movimentação dentro de um volume convexo. Na navegação global, a rota é definida, e a navegação local é utilizada para mover de um *waypoint* para outro, visto que esses volumes convexos são adjacentes.

Ambientes que possuem muitos elementos dinâmicos carregam consigo um outro problema, o de desvio de obstáculos dinâmicos[8]. A execução de tarefa que requer navegação deve executar o roteamento no início da tarefa, e a cada a ciclo desta tarefa executar a navegação local. Isto torna possível a previsão de objetos dinâmicos no percurso durante a execução da navegação local.

### 3.6 Animation

*Animation* é responsável controlar o corpo do BOT decidindo qual animação será apresentada para

demonstrar o estado atual de ação deste BOT. Estas animações são muitas vezes pré-geradas ou por animadores profissionais ou por captura de movimento, entretanto é possível gerar animação em tempo de execução[6]. Um exemplo de geração de animação em tempo de execução é a utilização de cinemática inversa[13].

Um modelo que possua apenas animações pré-definidas, baseadas em *keyframes*[14], é o modelo mais simples de ser tratado por *Animation*. Neste caso, o módulo *Animation* deve apenas selecionar qual animação pré-definida deve ser executada, com base nas ações de *Behavior*. Apesar da facilidade de manipulação e o baixíssimo custo computacional em tempo de execução, esta abordagem apresenta uma inflexibilidade nas animações, já que todas as animações são pré-definidas, e também alto consumo de memória, uma vez que todos os quadros de animação são carregados.

Um modelo que permita a geração dinâmica da animações é um pouco mais complexo de se tratar, pois além de levar em consideração as ações do BOT, deve levar em consideração também as percepções que este recebe. Este modelo possibilita animações como apontar para um alvo e pegar um elemento em cima de uma mesa. Cinemática inversa é bastante utilizada para este tipo de animação. Esta abordagem aumenta a complexidade de manipulação da animação pois agora não é uma simples seleção que deve ser feita, e também aumenta o custo computacional em tempo de execução, pois a cada ciclo é preciso gerar uma nova animação. Entretanto, o consumo de memória é bem mais baixo, já que não é necessário carregar todas as animações, e as animações são bem flexíveis uma vez que elas são geradas dinamicamente[14].

É possível também utilizar uma abordagem hibrida, com parte da animação pré-definida e uma outra parte sendo gerada dinamicamente. Isto ajuda a balancear o processamento em tempo de execução e o uso de memória, fornecendo flexibilidade na animação quando necessário e a facilidade de seleção para animações padrões.

### 3.7 Behavior

*Behavior* é o módulo responsável por decidir qual ação executar sobre o ambiente. Dado que *Short-Term Decision* selecionou um comportamento, este passa a ser o comportamento reativo atual. O comportamento reativo selecionado capta as percepções do ambiente e executa ações sobre o mesmo ambiente. O conjunto de ações do agente é similar as ações executadas pelo jogador humano através dos dispositivos de entrada.

Embora a interface das ações executadas pelo agente seja bastante similar às executadas pelo jogador humano, existe diferença. Um jogador humano e o dispositivo de entrada utilizado por ele, possuem limitações físicas para a mudança do ângulo de visão. As ações executadas pelo agente não possuem tais limitações. Assim, essas restrições devem ser simuladas explicitamente no agente para que ele possua igualdade de condições com o jogador humano.

## 4 Estudo de caso

O estudo de caso apresentado neste trabalho é um exemplo de um BOT simplificado e seus estados dentro do jogo, visando ilustrar algumas das características descrita pela arquitetura acima.

A implementação dos módulos desse BOT fica da seguinte maneira:

- *Perception* utilizando um sistemas baseado em mensagens, o qual filtra percepções para os demais módulos.
- *Location* utilizando um sistema de *waypoint* para representar o ambiente.
- *Long-Term Decision* utilizando um sistema baseado em conhecimento para definir metas e planos
- *Short-Term Decision* utilizando um sistema baseado em conhecimento para decidir sobre a execução das tarefas.
- *Motion* utilizando o algoritmo A* para o roteamento e interagindo com o sistema de verificação de colisão.
- *Behavior* utilizando máquina de estado finito dirigida a dados para representar um comportamento.
- *Animation* utilizando uma máquina de estado finito para definir a animação atual.

O exemplo observado é um jogo clássico do gênero FPS, o *Capture The Flag*. Neste jogo, são apresentados os possíveis estados do BOT após ter

sua bandeira roubada. Assim, a meta atual do agente é "Recuperar bandeira roubada".

Para essa meta, existem dois planos no conjunto de planos associado:

*Plano1*

- Seguir o adversário com a bandeira;
- Entrar em combate com adversário;
- Pegar a bandeira.

*Plano2*

- Seguir para a base adversária;
- Procurar uma posição estratégica;
- Esperar adversário;
- Entrar em combate com adversário,
- Pegar bandeira.

Os comportamentos possíveis são os apresentados abaixo nas figuras 2,3,4.

Suponha que a percepção acaba de alimentar a base de conhecimento com a informação de que o adversário está próximo e que existe informação de que o BOT tem pontos de vida suficiente para entrar em combate direto com o inimigo. *Long-Term Decision* escolhe o *Plano1* e o passa para *Short-Term Decision*.

A execução da primeira tarefa do *Plano1* começa utilizando a percepção visual, com informação do jogador que acabou de roubar a bandeira, para decidir como seguir o adversário. Ele passa as informações chegadas da percepção visual para o *Motion*, que lhe retorna o ponto o qual ele deve se mover naquele momento. Assim, *Short-Term Decision* seleciona o MoverPara (X,Y,Z) como comportamento reativo atual, passando como parâmetro o ponto retornado do *Motion*. Esta iteração se repete até *Short-Term Decision* verificar que o adversário parou de fugir.

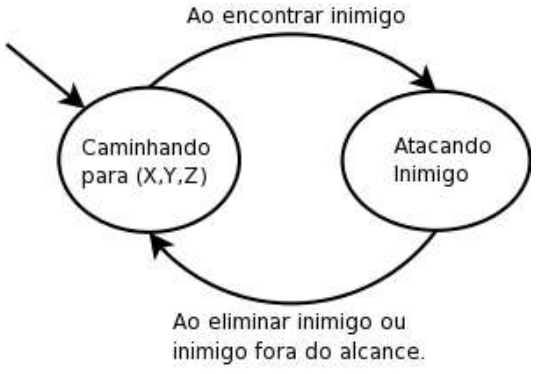


Figura 2: Comportamento MoverPara(X,Y,Z)


Figura 3: Comportamento PegarItem(X)


Figura 4: Comportamento Combate1

Para executar a tarefa seguinte, *Short-Term Decision* ainda utiliza *Motion* para passar parâmetro para o novo comportamento reativo o Combate1. Após a eliminação do inimigo, o comportamento PegarItem(X) passa a ser ativo. Ao pegar a bandeira, *Long-Term Decision* percebe que o objetivo atual foi atingido e decide o próximo objetivo, que poderia, como exemplo, ser "Capturar a bandeira adversária".

## 5 Questões sobre implementação

O caráter modular da arquitetura aqui descrita tem simplificado a implementação das técnicas de inteligência artificial dentro do InGE. pois tem tornado possível a realização de testes e depuração dessas técnicas de forma independente. Este é o caso de implementação da máquina de estado finito que pôde ser testada simulando módulos como o *Animation* e *Behavior*.

Apesar da forma de implementar cada módulo ser peculiar ao contexto de cada jogo, alguns

módulos apresentam tendências para algumas técnicas de inteligência artificial em detrimento a outras. *Long-Term Decision* é um forte candidato a um sistema baseado em conhecimento, *Perception* tende a ser um sistema baseado em mensagens, e *Behavior* a ser um sistema baseado em estados, em especial máquinas de estado dirigida a dados[9], sejam elas máquinas de estado finito ou máquinas de estado fuzzy.

## 6 Conclusão

A arquitetura introduzida neste trabalho explora características pertencentes a jogos de primeira e terceira pessoa, juntamente com elementos presentes em outras arquiteturas, para simplificar o desenvolvimento de agentes autônomos cognitivos para jogos em ambos os gêneros. Esta arquitetura se baseia no modelo híbrido de agentes cognitivos, por tal modelo oferecer melhor aproximação com as seqüências de ações e com o raciocínio humano. A divisão da arquitetura em sete módulos quebra o processo de desenvolvimento do agente em partes menores, simplificando o gerenciamento de cada parte e reduzindo a complexidade total do problema. Além disso, a especialização no contexto de jogos é o que permite que esta arquitetura de agente cognitivo obtenha melhores resultados em relação a outras arquiteturas híbridas. A implementação de estudos de caso e o desenvolvimento do subsistema de inteligência artificial no InGE tem mostrado a viabilidade de se utilizar tal arquitetura. É possível assim, observar os importantes avanços obtidos com este trabalho para o desenvolvimento de BOTs em jogos de primeira e terceira pessoa.

## 7 Referências


[1] Waveren, J.M.P. van. *The Quake III Arena Bot*, Master of Science thesis Delft University of Technology, June 2001.

[2] Tozour, P. *First-Person Shooter AI Architecture*, AI Game Programming Wisdom, pp. 387-396, 2002.

[3] O'Brien, J. *A Flexible Goal-Based Planning Architecture*, AI Game Programming Wisdom, pp. 375-383, 2002.

[4] Sloman, A. *What sort of architecture is required for a human-like agent?* In Proceedings of AAAI-96, August 1996.

[5] Bittencourt, G. *In the Quest of the Missing Link.* In Proceedings of IJCAI 15, pp. 310-315, August 1997.

[6] Grünvogel,S. M. *Dynamic Character Animation*, IJIGS, Vol 2, No 1, February 2003.

[7] Brooks, R. A. *A Robust Layered Control System for a Mobile Robot.* IEEE Journal of Robotics and Automation,2, pp. 14-23, 1986.

[8] Johnson, G. *Avoiding Dynamic Obstacles and Hazards.* AI Games Programming Wisdon 2, pp. 161-170, Charles River Media, Hingham, Massachusetts, 2004.

[9] Rosado, G. *Implementing a Data-Driven Finite-State Machine.* AI Games Programming Wisdon 2, pp. 307-317, Charles River Media, Hingham, Massachusetts, 2004.

[10] Yiskis, E. *A Subsumption Architecture for Character-Based Games.* AI Games Programming Wisdon 2, pp. 329-337, Charles River Media, Hingham, Massachusetts, 2004.

[11] InGE. *Indigente Game Engine*, http://indigente.org/inge (15/09/2005).

[12] Indigente: *Interactive Digital Entertainment*, http://indigente.org (15/09/2005).

[13] Eberly David H. *3D Game Engine Design*, Morgan Kaufmann Publishers, Silicon Valley, 2000.

[14] Pipho, E. *Focus On 3D Models*, Premier Press , Cincinnati, Ohio, 2003

[15] Schwab, B. *AI Game Engine Programming*, Charles River Media, Hingham, Massachusetts, 2004.