Resumo

Q Learning é uma técnica de Aprendizado por Reforço que é bastante difundida e utilizada na área de jogos. Baseado no *Q Learning* é possivel mapear um ambiente de simulação em Estado – Ação – Recompensa pelo método de tentativa e erro, assim o NPC poder aprender a se locomover pele mesmo sem ter conhecimento prévio. Então, alimentando um sistema *Fuzzy* com informações do ambiente e do próprio NPC (níveis e taxa de consumo de energia), é possível enriquecer esta experiência de aprendizado. Podemos então criar comportamento pré-definidos para o NPC, que serão geridos por este sistema *Fuzzy*, com base no estado energético do NPC, havendo assim a influência de um sistema de controle baseado em Lógica *Fuzzy* sobre a tomada de decisão de um sistema de Aprendizado por Reforço.

Abstract

Q learning is a Reinforcement Learning technique well known and used in the game development area. Based on it, we are capable of mapping an environment in State-Action-Reward through the trial and error method, like this the NPC can learn to walk thought this environment, even without knowing it beforehand. So, by feeding a Fuzzy system information taken from the environment, and the NPC itself (energy levels and energy loss rate), is possible to enrich the learning experience. The NPC can now assume pre-defined behaviors that will be managed by the Fuzzy System, based on the NPC energy level, having like this a Reinforcement Learning decision making system influenced by a Fuzzy one.

Sumário

Índice de Figuras

Índice de Tabelas

Índice de Equações

Tabela de Símbolos e Siglas

(Dispostos por ordem de aparição no texto)

FPS – First-Person Shooter

NPC – Non-Player Character

XNA – Plataforma de Jogos da Microsoft

Capítulo 1

Introdução

Os NPCs fazem parte dos jogos eletrônicos, sejam eles de computador ou de videogame. A sigla NPC significa Personagem Não Jogador, do inglês Non-Player Character. Eles são qualquer personagem, entidade ou parte do jogo com que o jogador interage, mas não tem controle sobre ele, ou seja, controlados por inteligência artificial. Devido a complexidade dos jogos de hoje em dia, um NPC pode assumir diversos papéis, tais como:

- Inimigos – devem eliminar o jogador
- Aliados – auxiliam o jogador a cumprir seus objetivos
- Personagens de suporte – personagem neutros que aumentam o realismo e dão continuidade à trama do jogo. Por exemplo, um mercador vende itens ao jogador.

Para que os NPCs exibam nos jogos comportamentos mais humanos, é necessário dá-los capacidade de adaptação e aprendizado, isto é feito por meio de técnicas inteligência artificial. Há várias formas de aplicar inteligência artificial dentro de um jogo. Além de várias formas de implementação, existem também vários níveis diferentes onde a Inteligência Artificial é aplicável, que vão desde a apresentação do personagem do jogo (há jogos que aplicam inteligência artificial nas animações do personagem, para simular reações mais realistas), passando pelo sistema de orientação até as decisões tomadas pelos NPCs. Por exemplo, em determinada situação o NPC pode escolher por fugir ao invés de lutar.

Para tanto os NPCs devem ser capazes de identificar as situações para que possam reagir de acordo. O NPC deve ser capaz de saber como partindo de sua posição atual chegar a outra posição, evitando objetos e obstáculos no caminho. No caso de um NPC hostil, este deve poder identificar o jogador e escolher a melhor forma de enfrentá-lo.

Além de todo este processamento, existem também outros níveis da estrutura de um jogo onde, para simular um comportamento humano, a inteligência artificial deve ser aplicada, abordaremos outros níveis de um NPC mais a frente no Capítulo 2.

Neste contexto, a idéia deste trabalho é simular o comportamento de um NPC dentro de um ambiente de simulação, para analisar a influência de um sistema de controle usando lógica *Fuzzy* sobre as decisões tomadas pelo algoritmo de aprendizado por reforço Q Learning. O ambiente é uma estrutura similar à uma casa, com ao todo 13 cômodos, ou estados, aos qual o NPC tem livre acesso. Dentro de três destes cômodos há uma estação de recarga, onde o NPC deverá caso ele precise recarregar sua energia. Partindo de um cômodo o NPC deve descobrir seu caminho através da casa para um estado final definido pelo usuário. A cada passo o NPC perde energia. Para cada estado foi atribuído uma taxa de perda de energia diferente.

Para montar este sistema, a primeira etapa foi o estudo e a implementação do Q Learning, método que será melhor abordado no Capítulo 3.

Em seguida foram definidos três tipos de comportamentos que podem ser assumidos pelo NPC, são eles:

- Crítico
- Conservador
- Explorador

Cada um deles é uma adaptação do funcionamento padrão do Q Learning, ou seja, cada um deles leva em consideração parâmetros diferentes no momento da tomada de decisão. Com exceção do comportamento crítico, que e a implementação do Q Learning em sua definição.

Em um terceiro momento, foi feito um estudo para a definição do sistema *Fuzzy* e que variável do ambiente ele controlaria. Ficou definido que seria a energia do NPC, principalmente pela facilidade de visualização da perda de energia à medida que o NPC se movimenta no ambiente de simulação.

Com o parâmetro de controle definido, foi implementado um sistema *Fuzzy* de controle que, dependendo da regra disparada, vai influenciar no comportamento do NPC. Os detalhes sobre as regras do sistema *Fuzzy* e sua influência sobre o comportamento do NPC serão abordados no Capítulo 3.

Por fim, foi desenvolvido o ambiente de simulação. Que permite ao usuário o controle e visualização de toda a simulação. Isto foi feito com uso do framework para jogos da Microsoft, o XNA.

1.1 Organização do Trabalho

Este trabalho está organizado em cinco capítulos. Neste Capítulo 1, estão expostos a motivação, escopo, objetivo e estruturação o projeto.

No Capítulo 2 são aprofundados os conceitos envolvendo os NPCs, sua evolução e estado atual, e como o projeto tenta propor uma nova abordagem à questão dos NPCs.

O Capítulo 3 aborda os conceitos fundamentais relativos ao trabalho realizado. É feito um resumo sobre aprendizado de máquina, com foco no aprendizado por reforço. Em seguida são abordados os dois conceitos principais do trabalho *Q Learning* e Lógica Fuzzy.

O Capítulo 4, mostra em detalhes todo o projeto. A simulação envolvendo a tomada de decisão pelo algoritmo de aprendizado por reforço Q Learning sendo influenciada por um sistema Fuzzy de controle. As implementações dos comportamentos, do sistema Fuzzy e discute os resultados encontrados nas simulações.

O Capítulo 5 expõe as conclusões chegadas e discute possibilidades de trabalhos futuros a partir dos resultados encontrados com o estudo realizado nesta monografia.

Capítulo 2

Inteligência Artificial em Jogos e o NPC

É difícil precisar quando o primeiro videogame foi produzido, mas quando se fala da popularização da idéia de jogar jogos em um monitor logo surge o nome Pong. Pong [1] foi um dos grandes responsáveis pela popularização do conceito de videogame.

Lançado em 1972 pela Atari [1], o jogo era baseado no tênis-de-mesa, ou ping-pong como alguns o chamam, daí seu nome. Graficamente o jogo era pobre, como mostra a Figura 2.1, e consistia em apenas um ponto, a bola do jogo, duas barras verticais representando os jogadores, uma linha que dividia os campos e o placar. Este jogo era para ser jogado por duas pessoas, não havia a possibilidade de o jogador desafiar a máquina, ou seja, não havia nenhum NPC presente.

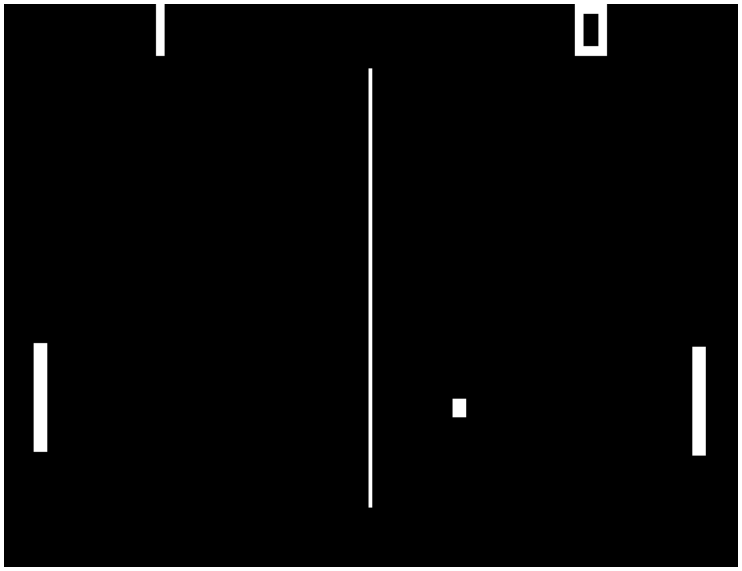


Figura 2.1 Pong, lançado em 1972 pela Atari. O primeiro jogo de computador.

2.1 O Surgimento do NPC

Os NPCs surgiram nos jogos arcade, e eram bem rudimentares comparados aos de hoje em dia. Eram objetos que apresentavam comportamentos pré-programados e sem nenhuma capacidade de adaptação. Como os jogos da época eram também muito rudimentares, geralmente estes NPCs eram objetos que o jogador deveria desviar ou eliminar de alguma forma. Um bom exemplo destes comportamentos padronizados e imutáveis é o jogo Space Invaders, lançado pela Taito em 1978 e ilustrado na Figura 2.2.

Nele o jogador tinha o objetivo de destruir todos os alienígenas que estava vindo em direção ao solo. Estes por sua vez se movimentavam todos juntos e com o mesmo padrão.

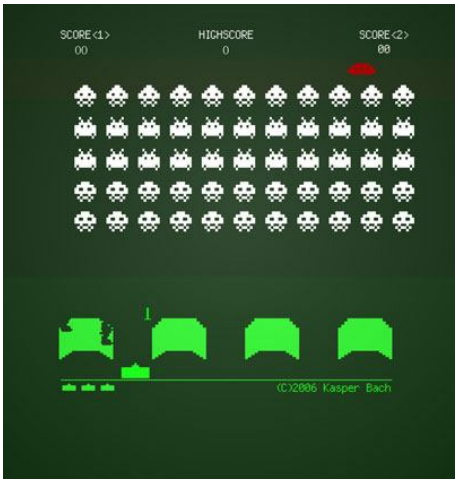


Figura 2.2 Space Invaders lançado em 1978 pela Taito, um exemplo de comportamento padrão dos NPCs

Mesmo com comportamentos pré-definidos e inalteráveis, a adição de NPCs já representou uma melhoria no nível de desafio e entretenimento proporcionado pelos jogos eletrônicos.

Embora ainda com atitudes pré-programadas e sem capacidade de adaptação, os NPCs do jogo Pac-Man introduziram um conceito interessando na indústria dos jogos. A idéia de comportamento.

2.2 Comportamento nos NPCs

Comportamento é um conjunto de regras que define um padrão de ação para um determinado NPC. Dependendo da situação um NPC pode assumir outro comportamento. Mudando de

comportamento um NPC pode fazer coisas que antes, de acordo com o comportamento anterior, ele seria capaz. Isto dá ao NPC uma ilusão de adaptação e inteligência.

Um bom exemplo disto, como já foi dito, é o jogo Pac-Man. O jogo é bem simples e basicamente o que o jogador deve fazer é percorrer um labirinto colhendo pontos na tela e evitando quatro fantasmas diferentes. A Figura 2.3 mostra o jogo em curso.

No jogo existem quarto fantasmas cujo objetivo é alcançar o jogador. Estes fantasmas podem ser distinguidos por sua cor, são eles:

- Vermelho – busca sempre perseguir o jogador, imitando seus movimentos
- Rosa – o mais veloz de todos, porém move-se de maneira aleatória, não toma decisão baseado na posição do jogador
- Azul – este fantasma tenta surpreender o jogador fazendo emboscadas. Move-se erraticamente quando o jogador está longe, porém quando o jogador se aproxima dele ele se torna agressivo e vai ao encontro do jogador
- Laranja – é lento e se move de forma aleatória, o menos perigoso entre os quatro

Embora estes comportamentos ainda fossem *scripts* que não davam ao NPC a capacidade de aprender e se adaptar ao ambiente. O simples fato dos NPCs apresentarem padrões diferentes surpreendia os usuários da época. Simulando por meio de scripts uma adaptação ou aprendizado, imitando comportamento humano. Pode-se dizer então que isto é uma mostra de Inteligência Artificial.

Scripts são muito utilizados para criar comportamentos “inteligentes” em situações geralmente controladas pelo programador. Ou seja,são situações onde o programador deseja controlar exatamente o que acontece no jogo (como no exemplo da Figura 2.6, onde acontece uma conversa entre os personagens), diferente do jogo em si onde quanto mais inesperadas as reações do NPC melhor.

2.3 Inteligência artificial nos NPCs

Como já foi dito no início deste documento, não é nada simples tentar simular o comportamento humano. Dentro de um ambiente 3D dos jogos atuais existem diversas camadas de processamento, a Figura 2.4 ilustra estas camadas, que vão desde a percepção do ambiente à tomada de decisão.

Assim fica claro que o desenvolvimento de NPCs inteligente não é uma tarefa fácil. Na indústria atual de jogos cada projeto é encarado como um desafio novo. Embora o algoritmo A* já se tornou uma plataforma para os problemas relacionados à Pathfinfding, seu uso já é bem difundido e as ferramentas auxiliares ajudam muito. Não existe um padrão para implementação de NPCs.

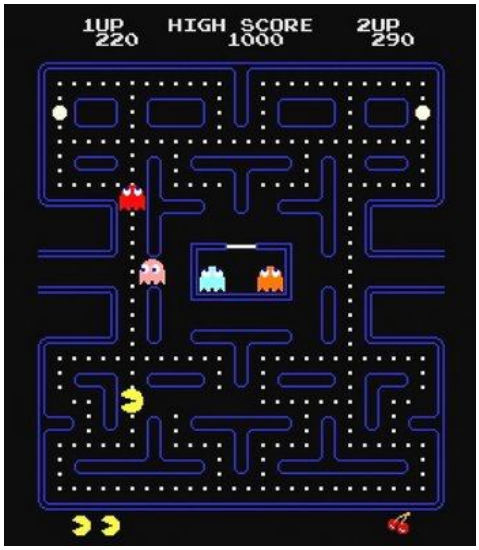


Figura 2.3 Pac-Man, um exemplo de diferentes comportamentos em NPCs

Para simular comportamento inteligente, muitos programadores fazem uso de scripts. Scripts são blocos de código que dizem como o NPC deve reagir em determinada situação.

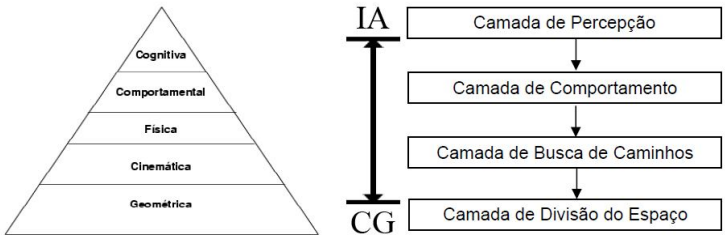


Figura 2.4 Camadas de implementação para IA em jogos

2.4 Ilusão de inteligência

Para jogos onde é possível que o jogador jogue sozinho, os *single-players*, o grande atrativo do jogo vai ser o desafio por ele proporcionado. Neste caso é interessante criar uma IA NPCs que imitem o comportamento humano, porém, mantendo o sistema simples o suficiente para que a

resposta dos NPC seja instantânea ao longo do jogo. Isto é alcançado criando a Ilusão de inteligência.

Geralmente, para que isto aconteça, o NPC é implementado com as seguintes capacidades:

- Evitar o *AI Cheating*
- Criar erros intencionalmente
- Natural Pathfinding
- Comunicação com o cenário
- Percepção mais humana

2.4.1 Evitar o AI Cheating

AI Cheating[8], do inglês roubo para a IA, nada mais é do que dar ao NPC vantagens sobre o jogador. Estas vantagens podem ser em vários níveis, desde fazer com que o NPC saiba onde o jogador se encontra mesmo sem o ter visto. Até conseguir se orientar e fazer a seleção de caminhos em ambientes ainda não explorados por ele.

Também é visto como "roubo", o fato de o NPC ter características que ultrapassam a capacidade humana. Como, por exemplo, uma mira perfeita e tempo de reação menores que os humanos dão ao NPC uma vantagem sobre o jogador em jogos de tiro.

2.4.2 Criar erros intencionalmente

Ter NPCs extremamente inteligentes não é garantia de um jogo divertido[7]. Imagine que, em um jogo de tiro, o jogador é constantemente abatido de surpresa, com um tiro perfeito que um jogador humano não conseguiria executar. Um jogo praticamente impossível de se obter a vitória, não é um desafio satisfatório para os jogadores.

Criar um personagem que possa vencer um humano não é difícil. O difícil é criar um personagem que proporcione uma batalha desafiadora onde os dois, tanto o jogador quanto o NPC têm a possibilidade de vencer. Então, a fim de evitar o que é considerado de "roubo", os programadores de IA para jogos freqüentemente adicionam aos NPCs sistemas que fazem com que eles intencionalmente cometam erros.

- **Mova antes de atirar** – é comum fazer com que os NPCs se movam por trechos do cenário onde o jogador ainda não passou. Durante este tempo, deve-se evitar que os NPCs ataquem o jogador, especialmente em situações onde os NPCs estão escondidos. O

jogador pode levar um tiro e até mesmo morrer sem saber quem o matou e de onde veio o tiro. Uma solução para isso é fazer com que pelo menos um NPC locomova-se no cenário antes de iniciar o ataque para alertar o jogador que a batalha esta por começar.

- **Seja visível**: É meta de todo programador de IA fazer com que o NPC permaneça o mais escondido possível, para que possam fazer emboscadas aos jogadores. Isso ocorre especialmente em cenários com camuflagem. Para isso, deve-se usar uma textura dos NPCs que tenha certo contraste com o ambiente do jogo.
- **Tenha uma péssima mira** – para que o jogador não seja sempre eliminado em frações de segundo, é interessante adicionar ao NPC uma taxa de erro nos seus disparos. Isto também aumenta a emoção do jogo, pois o jogador pode ver disparos em sua direção destruindo o ambiente. Também é comum fazer com que o NPC quase sempre erre o primeiro disparo em direção ao jogador.
- **Ataques individuais** – em ambientes com vários NPCs, o sistema controla os ataques ao jogador para que apenas um, ou alguns, ataquem o jogador de cada vez. Evitando assim que o jogador fique em grande desvantagem numérica.
- **Adicionar vulnerabilidades** – um bom exemplo disto é o caso de um NPC que pode desviar de minas terrestres, mas que eventualmente vai em direção à uma mina.

Para que estas estratégias sejam postas em prática com sucesso, é recomendada uma boa plataforma de testes.

2.4.3 Natural Pathfinding

Consiste em achar caminhos mais suaves entre a posição atual e o destino. A Figura 2.5 ilustra este comportamento em um NPC.

2.4.4 Comunicação com o cenário

É o *feedback* que o NPC recebe do ambiente no qual está inserido. O jogo, como um sistema complexo que é, tem vários sistemas integrados. Podemos imaginar um sistema “Gerente” que trata todas as informações que se passam no jogo. Estas são acessíveis pelos NPCs, que devem tratar as informações que recebem para tomar suas decisões.

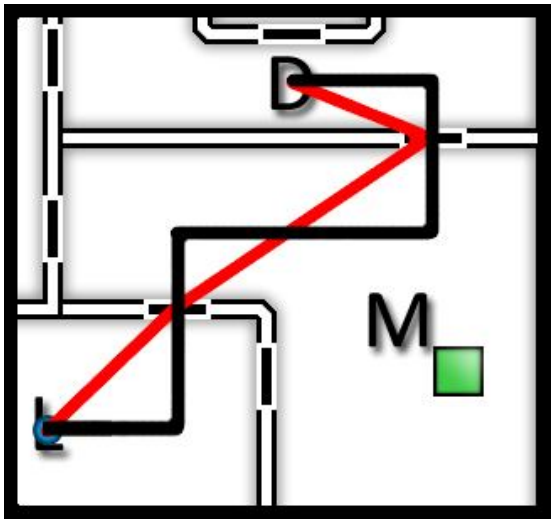


Figura 2.5 Exemplo de Natural Pathfinding em vermelho. Representa o caminho mais suave entre dois pontos

2.4.5 Percepção mais humana

É conseqüência da comunicação com o cenário. São os meios como o NPC se comunica com o ambiente. Fazendo uma analogia com os humanos, este nível representaria os sentidos e as reações que esboçamos ao receber um estímulo.

O NPC ficar atordoado e demorar um pouco mais a reagir ao ser atingido por trás, seria um exemplo de NPC com percepção humana.

2.5 NPCs atuais

Como foi mostrado na Figura 2.4, atualmente os NPCs implementados em camadas. Em um módulo separado que faz interface com o *engine* do jogo.

O *engine* é um software responsável pela criação dos jogos, eles são responsáveis por dispor de ferramentas necessárias para o desenvolvimento de eventos físicos, animações de personagens, edição de som, gerenciamento de memória, entre outras funções. Ou seja, é uma plataforma de desenvolvimento de jogos.

Os NPCs são implementados das mais diversas formas, não há um padrão de projeto de desenvolvimento de IA para jogos. Cada empresa tem sua abordagem característica.

Figura 2.6 Cena do jogo *Call of Duty 4: Modern Warfare*, onde é exibida uma conversa entre dois personagens

Além das técnicas de IA, como arvores de decisão, sistemas baseados em regras e RNAs. Também são utilizados muitos scripts combinados com os artefatos citados acima para dar ao NPC a ilusão de inteligência.

Capítulo 3

Noções de Aprendizado de Máquina para Jogos e Lógica Fuzzy

Este trabalho tem como principal alvo de estudo duas técnicas de Inteligência Computacional, Q Learning e Lógica *Fuzzy*. Para uma melhor compreensão do trabalho é necessário que haja um melhor entendimento destas duas técnicas. Porém, para falar sobre os conceitos abordados neste trabalho, é importante falar antes sobre aprendizado de máquina que constitui uma das áreas de Inteligência Computacional.

3.1 Aprendizado de Máquina

O Aprendizado de Máquina é uma subárea da Inteligência Computacional, e tem como foco o desenvolvimento de técnicas e algoritmos que permitam que os computadores "aprendam". Segundo Mendel e McLaren [6] o processo de aprendizagem pode ser expresso da seguinte maneira:

> "Aprendizagem é o processo pelo qual os parâmetros de um sistema são ajustados de uma forma contínua de estímulo pelo ambiente no qual está operando, sendo o tipo específico de aprendizagem realizada definido pela maneira particular como ocorrem os ajustes realizados nos parâmetros."

Entre os diversos métodos de Aprendizado de Máquina, algumas características são comuns a muitos deles. Assim podemos dividir os tipos de Aprendizado de Máquina em duas grandes áreas:

- Aprendizado Supervisionado
- Aprendizado Não-Supervisionado

Cada um destes tipos tem suas vantagens e desvantagens, não há um paradigma de aprendizado superior a outro. Existem sim, situações onde um tipo de aprendizado é mais adequado. As Figuras 3.2 e 3.1 ilustram respectivamente os aprendizados Supervisionados e Não-Supervisionados.

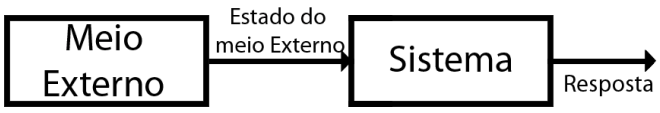


Figura 3.1 Sistema de Aprendizado Não-Supervisionado

Além disto, existem também métodos, ou algoritmos, de aprendizado que são mais adequados em determinadas situações e estruturas. No nosso caso abordaremos mais profundamente o Aprendizado Supervisionado, pois utilizaremos uma técnica de Aprendizado por Reforço que é um caso particular do Aprendizado Supervisionado.

3.2 Aprendizado Supervisionado

É um dos paradigmas mais utilizados, uma grande parcela disto é representada pelos algoritmos de treinamento de RNAs (Redes Neurais Artificiais), especialmente o algoritmo Back-Propagation [5]. O Aprendizado Supervisionado recebe este nome devido ao fato de a entrada e saída do sistema serem previamente conhecidas, pois devem ser fornecidas ao sistema por um supervisor externo, ou professor.

O objetivo do sistema é o ajuste dos parâmetros para que com determinada entrada, o sistema gere a saída desejada pelo professor. Ou seja, o sistema deve conhecer exemplos dos casos reais onde ele deve atuar. Isto é chamado de conjunto de treinamento, e consiste em uma amostra (pares de entrada e saída desejada) do escopo de atuação do sistema. No caso das RNAs os parâmetros que são ajustados são os pesos das conexões, para que com uma entrada qualquer a saída seja igual à esperada pelo professor.

Uma desvantagem do Aprendizado Supervisionado está no fato de que, na ausência de um professor, o sistema não consegue aprender novas estratégias para situações não apresentadas no conjunto de treinamento. Ou seja, um novo treinamento é necessário cada vez que o sistema precise aumentar seu escopo e lidar com novas situações.

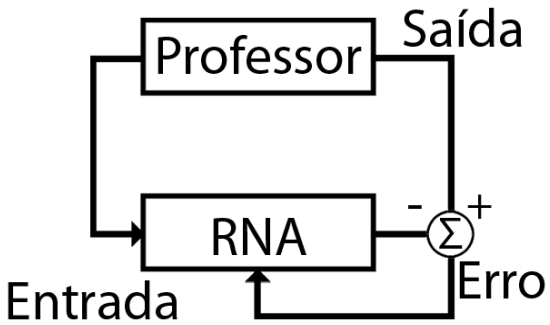


Figura 3.2 Sistema de Aprendizado Supervisionado

Para reduzir o impacto desta limitação, há duas formas de implementar o aprendizado supervisionado:

- Aprendizado Off-Line
- Aprendizado On-Line

No Aprendizado Off-Line, caso novos exemplos precisem ser adicionados, um novo conjunto de treinamento é montado, incluindo também o conjunto de treino anterior. Então um novo treinamento é realizado. O resultado disto é que o sistema adquire o mesmo conhecimento que tinha, com a adição dos novos casos do novo conjunto de treinamento.

No caso do On-Line, o conjunto de treinamento está sempre mudando e o sistema está em constante treinamento. Isto garante que quando o sistema for submetido a alguma alteração, esta já tenha sido exposta ao sistema pelo professor.

3.3 Aprendizado por Reforço

O Aprendizado por Reforço pode ser visto como um caso particular do Aprendizado Supervisionado. A principal diferença entre o Aprendizado Supervisionado clássico e o Aprendizado por Reforço é a medida de desempenho usada em cada um dos sistemas. No Aprendizado Supervisionado, a medida de desempenho é baseada no conjunto de respostas desejadas usando um critério de erro conhecido, enquanto que no por Reforço o desempenho é baseado em qualquer medida que possa ser fornecida ao sistema. No Aprendizado por Reforço a

única informação retornada pelo sistema é se a resposta obtida está correta ou não, isto é, não é fornecida ao sistema a resposta correta. A Figura 3.3 ilustra o aprendizado por reforço.

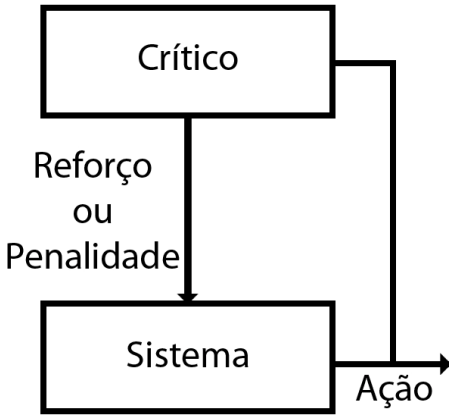


Figura 3.3 Sistema de Aprendizado por Reforço

O Aprendizado por Reforço é uma forma de aprendizado On-Line obtido por um mapeamento de entrada-saída através de um processo de triagem e erro, desenvolvido para maximizar o índice de desempenho escalar chamado de sinal de reforço.

O termo Aprendizado por Reforço tem sua origem em estudos experimentais sobre o aprendizado dos animais, que diz que quanto maior a satisfação obtida por um animal em certa experiência, maiores as chances de ele aprender [2].

Conceito que foi posteriormente reformulado por Sutton [3] que definiu o Aprendizado por Reforço como:

> "Se uma ação tomada pelo sistema de aprendizagem é seguida de estados satisfatórios, então a tendência do sistema de produzir esta ação em particular é reforçada. Se não for seguida de estados satisfatórios, a tendência do sistema de produzir esta ação é enfraquecida."

3.4 Q Learning

O intuito de um sistema de Aprendizado por Reforço, mostrado na Figura 3.3, é de encontrar uma política ideal (menor custo possível) após tentar várias seqüências de ações, e observar os custos associados a cada mudança de estado.

O Q Learning é um método estocástico de programação dinâmica incremental que determina uma política ideal passo-a-passo. É altamente recomendável para a resolução de problemas de

decisão Markovianos sem ter o conhecimento explícito das probabilidades de transição entre os estados. Contudo, seu uso mais adequado é quando podemos assumir que os estados do ambiente são completamente observáveis, isto é, o ambiente sendo uma cadeia de Markov totalmente observável.

A Equação 3.1 mostra a combinação das equações de Otimização de Bellman [4] e do fator Q(i, a) para o par estado-ação (i,a), usando a definição do custo imediato estimado como c(i,a).

$$Q^*(i,a) = \sum_{j=1}^{k} p_{ij}(a)\,(g(i,a,j) + \gamma \min_{b \in A_j} Q^*(j,b)) \;\forall\,(i,a)$$

Equação 3.1 Combinação das equações de Otimização de Bellman e a política de Iterações

Tal equação pode ser vista como uma versão de dois passos da equação de Otimização de Bellman. As soluções para o sistema linear de equações definido na equação 1, definem unicamente os valores Q ideais, Q*(i, a), para todos os pares de ação (i, a) do sistema.

Podemos isolar o sistema de equações em termos dos fatores Q para resolver o sistema linear, representada pela Equação 3.2. Assim, para cada iteração do algoritmo nós temos,

$$Q(i,a) \coloneqq \sum_{j=1}^{N} p_{ij}(a)\,(g(i,a,j) + \gamma \min_{b \in A_j} Q(j,b)) \;\forall\,(i,a)$$

Equação 3.2 Iteração da Equação 3.1

A versão de passo curto desta iteração é descrita por, Equação 3.3:

$$Q(i,a) \coloneqq (1-\mu)Q(i,a) + \mu \sum_{j=1}^{N} p_{ij}(a)(g(i,a,j) + \mu \min_{b \in A_j} Q(j,b)) \;\forall\,(i,a)$$

Equação 3.3 Versão de passo curto da Equação 3.2

Sendo μ a taxa de aprendizado que tem valor definido entre 0 < μ < 1.

Neste estado atual, Equação 3.3, a equação necessita das probabilidades das transições entre os estados. Podemos eliminá-la criando uma versão estocástica da Equação 3.2 Onde é realizado o cálculo da média para cada iteração sobre todos os estados possíveis, isto pode ser substituído por uma única amostra, Equação 3.4. Temos então a equação de atualização do valor Q:

$$Q_{n+1}(i,a) - \left(1 - \mu_n(i,a)\right)Q_n(i,a) + \mu_n(i,a)\left[g(i,a,j) + \gamma J_n(j)\right] para\ (i,a) - (i_n, a_n)$$

Equação 3.4 Atualização do valor Q da ação executada

Onde

$$J_n = \min_{b \in A_j} Q_n(j, b)$$

Equação 3.5 Definição do próximo estado com menor esforço associado

Sendo J o estado anterior, e $\mu_n(i, a)$ é a taxa de aprendizado do par estado-ação (i, a) no instante n. A equação de atualização do valor Q, Equação 3.4, se aplica para o estado atual e altera o valor Q do par estado-ação (i_n, a_n). Para todos os outros pares os valores permanecem inalterados.

Em suma, o Q Learning é uma forma de aproximação estocástica da política de iteração de valores. Atualizando o valor Q para um único par estado-ação a cada iteração do algoritmo. A grande vantagem é que, o algoritmo converge para os valores Q ideais sem a necessidade da criação de um modelo de decisão Markoviano.

3.5 Lógica Fuzzy

A primeira publicação científica sobre Lógica *Fuzzy* veio em 1965 por Lofti A. Zadeh, professor de Ciência da Computação da faculdade da California, a Lógica *Fuzzy* é uma lógica, em oposição à *booleana*, multivalorada. Ela permite que valores intermediários existam entre os extremos representados por *true* e *false*, zero e um, sim e não, etc. A Lógica *Fuzzy* é uma alternativa em relação à noção de conjuntos com origens na filosofia grega.

3.5.1 Conjuntos Fuzzy

Característica básica de um sistema *Fuzzy*, o conjunto *Fuzzy* representa o quanto um elemento pertence a uma classe. São propostos níveis de pertinências às classes. Um elemento pode fazer parte de várias classes ao mesmo tempo sem realmente fazer parte de nenhuma. Por exemplo, a lógica Aristotélica diz que um objeto pertence ou não a um conjunto específico. O conceito de conjunto *Fuzzy* diz que:

> "Ao invés de dar aos elementos do conjunto os valores zero ou um (representando se o objete pertence ou não ao conjunto), vamos atribuir a cada elemento um valor dentro do intervalo (0,1). Assim, agora existem níveis de pertinência de cada elemento em relação a um conjunto específico. Caso o número 1 seja atribuído a um elemento, significa dizer que este com certeza faz parte do conjunto em questão. Se 0 for atribuído, significa dizer que este elemento **NÃO** faz parte do

conjunto. Os valores intermediários representam níveis de pertinência do elemento ao conjunto.[9]"

A função de pertinência é uma regresentação gráfica do nível de participação que cada elemento. Um exemplo de função de pertinência pode ser viso na Figura 3.4, onde x = 3.5 implica dizer que esta relação tem uma pertinência de .5 para o conjunto A.

Vale salientar neste caso, a diferência entre Lógica *Fuzzy* e probabilidades. Ambas operam em um intervalo de valores entre (0,1), zero representando exclusão total e um pertinência total. Entretanto, enquanto que a probabilidade diria que , no exemplo da Figura 3.4, existe uma chance de 50% de o elemento fazer parte do conjunto A, a Lógica *Fuzzy* diz que o elemento está "mais ou menos" adequado ao conjunto A.

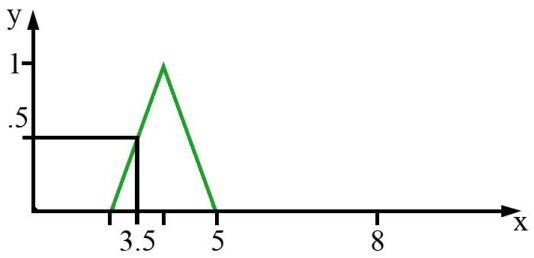


Figura 3.4 Exemplo de Função de Pertinência

3.5.2 Operação com números Fuzzy

Agora com o conceito de conjunto *fuzzy* definido, podemos introduzir oerações básicas entre eles. São elas:

- AND
- OR
- NOT

Em seu primeiro artigo [9] L. A. Zadeh sugeriu um mínimo operador, para intersecção, e um máximo operador, para união de dois conjuntos *fuzzy*. Por exemplo, seja A um intervalo *fuzzy* entre 5 e 8 e B um número *fuzzy* de valor em torno de 4 pode ser visto Figura 3.5. Podemos executar as operações entre estes dois conjuntos, as Figuras 3.5, 3.6 e 3.7 ilustram as operações **AND**, **OR** e **NOT** (esta última apenas para o conjunto A) respectivamente.

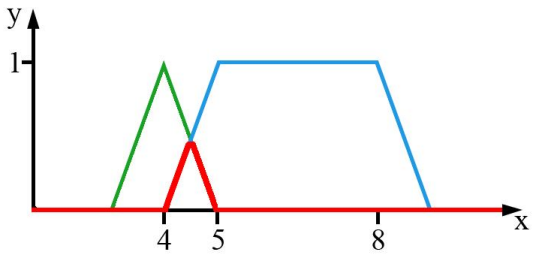


Figura 3.5 Operação *Fuzzy* **AND** entre os conjuntos *fuzzy* A e B

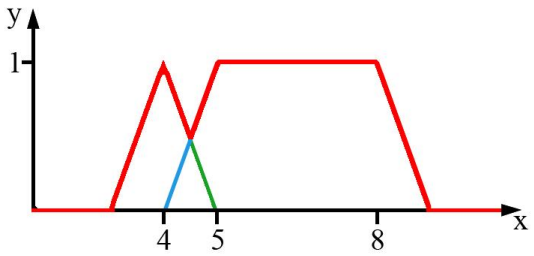


Figura 3.6 Operação *Fuzzy* **OR** entre os conjuntos *fuzzy* A e B

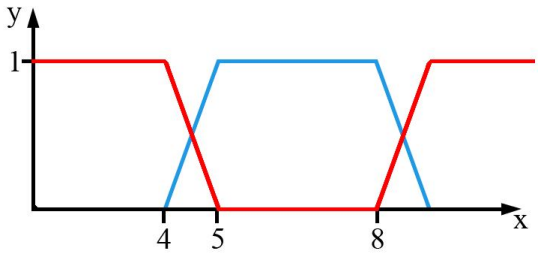


Figura 3.7 Operação *Fuzzy* **NOT** no conjunto *fuzzy* B

3.5.3 Sistemas Fuzzy

Sistemas de controle *fuzzy* são uma aplicação da Lógica *Fuzzy*. Para isto conhecimento especializado é necessário para que as regras do sistema sejam criadas para descrever os conjuntos *fuzzy*, esta descrição é feita através de variáveis linguísticas.

Variáveis Linguísticas

São descrições naturais de uma situação que define uma peroção de um conjunto. Por exemplo, para controlar a temperatura de um ambiente seguintes variáveis linguísticas podem ser criadas:

- Muito Alta
- Alta
- Ideal

- Baixa
- Muito Baixa

Elas representam conjuntos ou sub-conjuntos *fuzzy*. E expressam regras dentro do sistema.

Digamos que este sistema deva manter a temperatura da sala sempre no nível ideal, e que com o decorrer do tempo há uma variação de temperatura. As variáveis linguísticas para a variação seriam:

- Esfriando
- Esquentando

De acordo com estas variáveis linguísticas podemos extrair regras para controlar o sistema. O sistema deve esfriar ou esquentar a sala de acordo com a variação de temperatra da sala no momento x. Então um exemplo de regra do sistema seria:

- **Se** temperatura **ALTA and** variação **ESQUENTANDO** então **ESFRIE**
- Se temperatura **BAIXA and** variação **ESFRIANDO** então **ESQUENTE**

Disparo de regras

Combinando logicamente, com o operador **AND,** as entradas do sistema conseguimos extrair um valor para cada regra. A regra com o maior valor de saída pode ser disparada e a ação definida nela será executada.

Capítulo 4

Ambiente de Simulação de Exploração e Resultados Obtidos

Os experimentos realizados neste trabalho envolvem o uso de uma técnica de aprendizado por reforço para tomada de decisão de um NPC sendo influenciada por um sistema de controle *Fuzzy*, que age sobre a técnica baseado em uma variável do ambiente. O objetivo deste experimento é de avaliar a compatibilidade das duas técnicas mencionadas acima e tentar simular um comportamento mais humano para o NPC no momento da tomada de decisão. Já que a lógica *Fuzzy* trabalha em um nível de abstração similar ao nosso.

O experimento realizado se resume a uma adaptação de um exemplo clássico de aplicação da técnica Q Learning, onde um diagrama de estados é representado como uma casa, pode ser visto na Figura 4.1, onde o NPC deve, partindo de um estado inicial, ser capaz de atingir o estado final. Com a adição da variável energia, pois a cada deslocamento o NPC perde energia, e de pontos de recarga (três ao total, espalhados pelo ambiente onde o NPC pode recarregar suas energias caso necessário).

Assumindo que os cômodos sejam os estados e as portas as transições entre eles, o experimento possui ao todo treze estados ligados entre si de várias maneiras. Também está associado a cada estado, um peso que vai influenciar na perda de energia do NPC.

Foram definidos três comportamentos que o NPC pode assumir, são eles:

- Conservador
- Crítico
- Explorador

O sistema *Fuzzy* age sobre o sistema por meio de diretivas que devem ser seguidas pelo NPC. Dependendo da regra que for disparada o NPC pode ter as diretivas:

- Siga buscando o objetivo
- Recarregue se uma estação de recarga estiver no seu caminho
- Recarregue

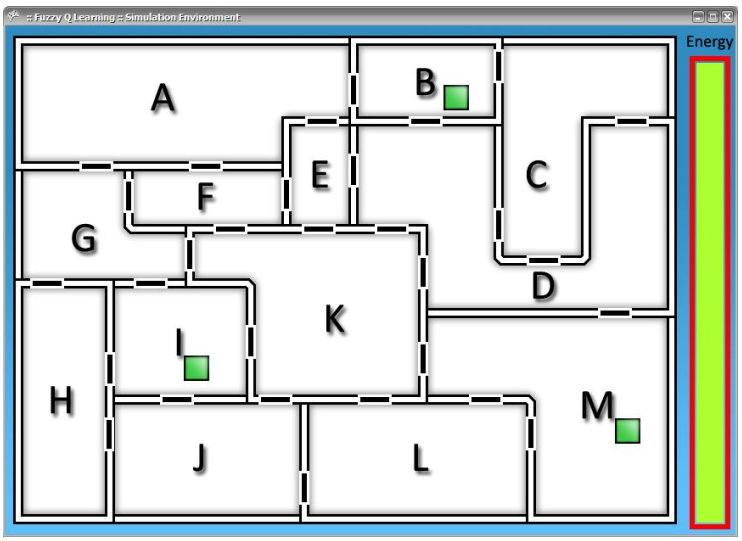


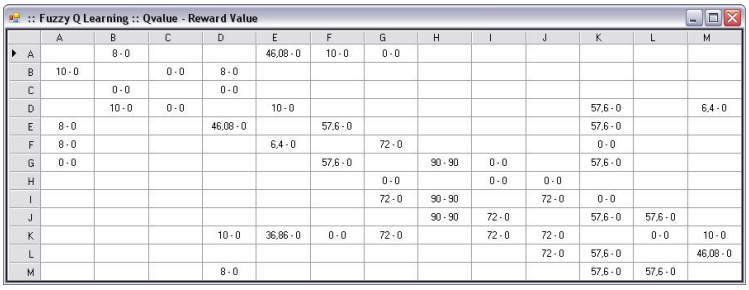
Figura 4.1 Ambiente de simulação do sistema

No caso desta última o que o NPC vai fazer é selecionar um estado contendo uma estação de recarga como seu objetivo atual. Após a recarga o objetivo anterior é retomado.

A interface gráfica do sistema é bastante amigável e permite ao usuário o controle de todos os parâmetros que podem ser ajustados durante a simulação, a janela de configuração principal, Figura 4.3, contém todos os controles necessários para que a simulação aconteça de acordo com o esperado.

Além da janela de configurações existem também mais três janelas, a janela da simulação em si, há também a janela com a grade de valores Q - Recompensa, Figura 4.2, que representa os valores atribuídos pelo algoritmo durante a simulação.

Toda a implementação do sistema foi feita no software Microsoft C# 2008 Express, utilizando o *Singleton design pattern*. E para a simulação foi utilizado também o framework XNA.



	A	B	C	D	E	F	G	H	I	J	K	L	M
A		8 - 0			46,08 - 0	10 - 0	0 - 0						
B	10 - 0		0 - 0	8 - 0									
C		0 - 0		0 - 0									
D		10 - 0	0 - 0		10 - 0						57,6 - 0		6,4 - 0
E	8 - 0			46,08 - 0		57,6 - 0					57,6 - 0		
F	8 - 0				6,4 - 0		72 - 0				0 - 0		
G	0 - 0					57,6 - 0		90 - 90	0 - 0		57,6 - 0		
H							0 - 0		0 - 0	0 - 0			
I							72 - 0	90 - 90		72 - 0	0 - 0		
J								90 - 90	72 - 0		57,6 - 0	57,6 - 0	
K				10 - 0	36,86 - 0	0 - 0	72 - 0		72 - 0	72 - 0		0 - 0	10 - 0
L										72 - 0	57,6 - 0		46,08 - 0
M				8 - 0							57,6 - 0	57,6 - 0	

Figura 4.2 Janela com os valores Q e de recompensa de cada ação

Figura 4.3 Janela de configuração da simulação

Vale ressaltar que pelo fato de a simulação ser feita em um ambiente que pode ser comparado a um diagrama de estados, ou até mesmo um autômato, pode parecer que o NPC deve sempre buscar o caminho mais curto entre os estados inicial e final. Porém, este não é o caso. O objetivo deste trabalho, como já foi dito antes, é analisar a influência da Lógica *Fuzzy* na tomada de decisão e as diferenças apresentadas entre os comportamentos que podem ser assumidos.

4.1 Q Learning

A implementação do Q Learning foi feita utilizando a sua definição formal (FORMULA). Em relação a este projeto em particular, o estado final seria o estado com maior valor de recompensa. Já as ações possíveis são representadas pelas portas, ou transições entre os estados, entre os cômodos do ambiente.

Durante a simulação o NPC apresenta duas fases distintas, são elas:

- Fase de adaptação – Nesta fase o NPC ainda não ajustou os valores Q de forma que ele consiga um caminho que satisfaça o seu objetivo. É nesta fase que as ações serão avaliadas e os valores Q delas serão ajustados.
- Fase de Execução – Nesta fase o NPC já encontrou os valores que satisfazem seu objetivo, pode ser uma ou mais de uma combinação de estados. Caso mais de uma

combinação seja válida a escolha da ação a ser tomada será aleatória entre as ações possíveis, ou seja, em case de empate do valor Q entre duas ou mais ações a escolha é aleatória.

Por exemplo, partindo do estado B para alcançar o estado G, o NPC pode seguir para A, C ou D, como pode ser visto na Figura 4.4, cada possibilidade destas é uma ação que pode ser tomada e terá atribuído o seu valor Q que vai ser proporcional à proximidade do estado final. Neste caso pode-se observar que, partindo de B a maior probabilidade é que ele siga para A, e estando em A, a maior probabilidade é que ele se dirija ao estado G, atingindo assim seu objetivo.

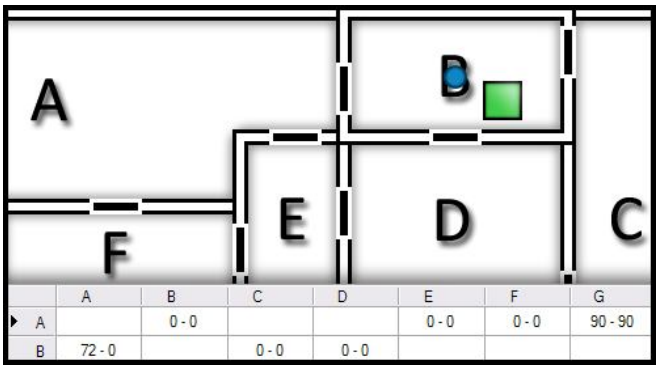


Figura 4.4 Exemplo de tomada de decisão do NPC na fase de Execução

4.2 Os comportamentos do NPC

Como foi dito anteriormente o NPC pode, ao longo da simulação, apresentar três comportamentos distintos, como mostra a Figura 4.5. Cada um deles difere entre si nos parâmetros que serão levados em consideração durante a execução do Q Learning. Vamos ver detalhadamente cada um deles.

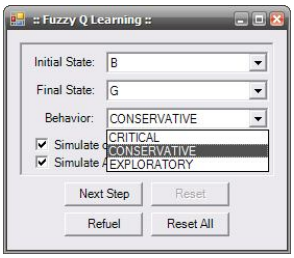


Figura 4.5 Comportamentos disponíveis ao NPC

4.3 Comportamento Crítico

O comportamento crítico é a implementação padrão do algoritmo Q Learning. Sendo assim, é interessante ver seu funcionamento em relação aos outros dois comportamentos. Já que neste caso não há nenhum outro parâmetro levado em consideração na tomada de decisão, apenas o próprio valor Q de cada ação possível.

A execução do algoritmo se dá nesta seqüência:

- Avaliar todas as ações possíveis para o estado no qual o NPC se encontra
- Selecionar as ações que tiverem o valor de recompensa igual ao maior valor de recompensa entre elas
- Retornar as ações que, dentre as previamente selecionadas, apresentar o maior valor Q. No caso de empate a escolha é feita de forma aleatória
- A ação retornada é executada
- Após a execução da ação, seu valor Q é atualizado com base na Equação 3.4

Este fluxo se repete até que haja uma seqüência de ações preferenciais desde o estado inicial até o estado final. Fazendo assim com que o NPC sempre siga este caminho.

Note que, como nesta abordagem apenas o objetivo final vai ter valor de recompensa, as ações irrelevantes para o objetivo em questão não terão valor Q associado, ficando com zero. Esta característica é típica do comportamento Crítico, os outros comportamentos apresentam abordagens próprias para este fator.

4.4 Comportamento Conservador

O comportamento conservador é uma adaptação do comportamento crítico, a diferença entre eles é devido ao que se pode chamar de pré-processamento das ações do estado no qual o NPC se encontra.

A seqüência de execução do algoritmo no comportamento Conservador é idêntica à do comportamento Crítico até o item três da sua seqüência de execução, neste ponto ocorre uma variação no algoritmo.

Neste ponto o sistema tem selecionadas as ações com maior valor Q entre as de maior grau de recompensa. Destas ações a que será selecionada é a que tiver a menor distância a ser percorrida pelo NPC.

O cálculo da distância entre o NPC e a porta que leva ao próximo estado, é a subtração dos vetores de posição da do NPC e da posição da porta correspondente à ação. Pois a posição do NPC e de cada uma das portas é sabida pelo sistema. Isto ocorre devido a uma abstração de outros sistemas, pois em um sistema complexo como um jogo existem vários sistemas integrados. Assumimos então que estes sistemas são capazes de se comunicar e buscar informações. Com isso podemos ter facilmente a distância entre o NPC e qualquer uma das portas possíveis.

Porém, o fato levar em consideração apenas as ações mais próximas do NPC gera o problema ilustrado na Figura 4.6, onde o NPC vai ficar em um loop infinito nos estados H, I e J. Pois como nenhuma das ações tomadas é relevante para o objetivo em questão, chegar ao estado B, o valor Q delas será sempre zero e o NPC continuará decidindo apenas com base na ação mais próxima.

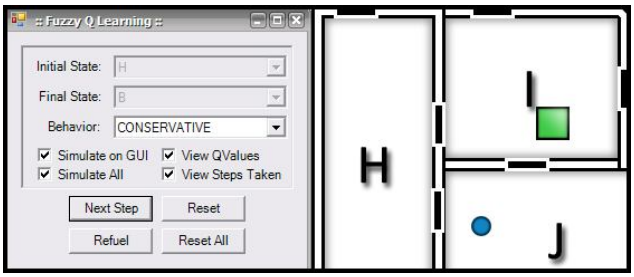


Figura 4.6 *Loop* Infinito causado pela escolha exclusiva da ação mais próxima

Para resolver este problema foi proposta uma alteração na equação de atualização do valor Q. Pois, como já foi mostrado no comportamento Crítico, quando uma ação é irrelevante para o objetivo, o valor Q dela permanece zero. Isto não funciona para o comportamento Conservador, porque se o NPC estiver entre ações irrelevantes, com o valor Q igual a zero, a tomada de decisão sempre vai ser com base na distância. Caracterizando assim um Loop infinito, porque as decisões nunca se alterariam. A mudança proposta esta descrita abaixo:

Se o valor da ação após a atualização é igual a zero, então se atribui -1 a este valor. É uma mudança sutil, porém, o resultado é satisfatório. Por exemplo, no caso ilustrado pela Figura 4.6, o NPC está executado a ação JH, se deslocando de J para H, ao chegar em H o valor Q da ação JH receberá o valor -1. Na próxima vez que o NPC chegar ao estado J, ao final da terceira etapa do algoritmo (Retornar as ações que, dentre as previamente selecionadas, apresentar o maior valor Q) a ação JH não seria mais possível, já que outras ações possuem o valor inicial que é zero. Garantindo assim que não haja uma situação de loop infinito no sistema.

4.5 O comportamento Explorador

O comportamento Explorador, também derivado do comportamento crítico, a busca sempre seguir por caminhos que não tenham sido explorados antes.

Similar ao comportamento Conservador, o Explorador altera o valor Q da ação caso este, após uma atualização, permaneça com zero. Isto ocorre porque a ação selecionada neste caso será sempre a que tem o valor Q igual a zero, isto até que todas as ações do estado em questão sejam exploradas. Após isto ocorrer, a ação selecionada volta a ser a que tem o maior valor Q.

Por conseqüência, o comportamento explorador geralmente encontra caminhos mais curtos (em quantidade de estados percorridos), mas é o que mais demora na fase de adaptação. Por exemplo, partindo de A com destino a C, é aceitável que o comportamento Crítico, execute sempre o caminho A-F-K-D-C ao mesmo tempo em que se o comportamento fosse o Explorador o caminho seria A-B-C.

4.6 O sistema Fuzzy

O sistema *Fuzzy* criado para influenciar na tomada de decisão do Q Learning foi feito para levar em consideração dois parâmetros, são eles:

- Nível de energia atual do NPC
- Taxa de Perda de energia do NPC

Como já foi dito anteriormente, cada cômodo tem um nível de gasto energético diferente. Com base nestes parâmetros é que o sistema *Fuzzy* funcionará para influenciar no NPC. Este terá três opções com base na saída do sistema *Fuzzy*:

- Continuar seguindo o objetivo
- Passar por uma estação de carga, caso haja uma no seu caminho
- Deixar o objetivo de lado, buscar uma estação de carga (utilizando o comportamento crítico). Após a recarga o NPC retoma o objetivo e comportamento anterior

Daqui em diante no texto estas saídas serão caracterizadas pelos termos **continuar**, **passando** e **recarregar** respectivamente.

4.6.1 Variáveis lingüísticas

Foram criadas seis variáveis lingüísticas para este trabalho, três delas se referem ao nível de energia do NPC naquele momento e as outras três se referem à taxa de perda de energia. As que se referem ao nível energético são:

- Satisfatório
- Regular
- Insuficiente

As que se referem à taxa de perda são:

- Alta
- Média
- Baixa

Para passar os valores extraídos do sistema, valores do mundo real, para o sistema *Fuzzy*, é necessária a Fuzzyficação. Nesta etapa vamos definir em que classificação o sistema realmente se encontra. Foram definidos intervalos para os quais os valores extraídos do sistema serão classificados. Com relação ao nível de energia (isto para o comportamento crítico, pois nos outros comportamentos os intervalos variam):

- Satisfatório é quando o NPC tem mais de 60% de energia
- Insuficiente é quando o NPC tem menos de 30% de energia
- Regular é o que não é nem satisfatório nem Insuficiente

Com relação à taxa de perda:

- Alta é quando o NPC perde mais de três pontos de energia a cada passo
- Baixa é quando o NPC perder menos de dois pontos de energia a cada passo
- Média é quando não é nem Alta nem Baixa

4.6.2 Regras do Sistema

As regras Fuzzy vão avaliar os parâmetros de entrada e gerar uma saída. Para este sistema foram criadas regras simples

- Energia satisfatória AND Taxa baixa ENTÃO **continuar**
- Energia satisfatória AND Taxa media ENTÃO **continuar**
- Energia satisfatória AND Taxa alta ENTÃO **passando**
- Energia regular AND Taxa baixa ENTÃO **passando**

- Energia regular AND Taxa media ENTÃO **passando**
- Energia regular AND Taxa alta ENTÃO **recarregar**
- Energia insuficiente ENTÃO **recarregar**

4.6.3 Números Fuzzy

As regras do sistema trabalham com as variáveis lingüísticas, ou seja, valores no universo Fuzzy. Assim, operadores *Fuzzy* devem ser definidos para que a representação numérica continue no universo *Fuzzy*.

4.6.4 Operador AND

Para fazer o cálculo do operador AND, a regra usada foi a seguinte:

$$y = A * B$$

Equação 4.1 Operação AND com operadores Fuzzy

No caso uma simples multiplicação não há o risco de o resultado final ultrapassar o valor um. Pois as duas entradas estarão “fuzzyficadas”, ou seja, entre zero e um.

4.6.5 Operador OR

Se usarmos a abordagem binário neste caso (representar o operador OU como a soma das entradas) corremos o risco de o valor final ultrapassar um. Isto não é permitido no universo *Fuzzy* porque o valor denota pertinência àquela classe.

Para fazer o cálculo do operador OR, a regra usada foi a seguinte:

$$y = (1 - (1 - A)(1 - B))$$

Equação 4.2 Operação OR com operadores Fuzzy

Com isso o problema levantado é resolvido. Por exemplo, se usarmos a SOMA das entradas para o caso 0.63 OR 0.72 o valor seria 1.35, coisa que ao é permitida. Usando a definição proposta o valor encontrado neste mesmo caso seria de 0,8964. Resultado que representa mais fielmente a união dos dois conjuntos, sem quebrar a definição de conjuntos *Fuzzy*.

4.6.6 Condição de Disparo de Regras

Os números *Fuzzy* de saída de cada regra denotam a prioridade daquela regra naquele momento. A cada movimento do NPC é feia uma nova avaliação da sua condição energética, ou seja, as regras estão em contínua mudança de prioridade. O NPC em um determinado momento tem sua energia alta, porém depois de algum tempo a energia esta baixa e a regra que mandava ele se recarregar agora tem uma prioridade que antes ele não possuía.

A regra que vai determinar a influência sobre a decisão do NPC é a regra que obtiver o maior valor de saída.

4.7 Resultados Obtidos

Com as simulações realizadas foi possível fazer um comparação entre os comportamentos, para analisar se realmente cada um teria um resultado final característico. Partindo do estado **A**, e com destino a **J**, foram realizadas vinte simulações para cada um dos três comportamentos.

Foram analisadas as seguntes variáveis:

- Enegia gasta no percurso
- Enegia restante ao final do percurso
- Número de estados visitados

Em cada um destes parâmetros ficou claro que os comprtamentos apresenta resultados direfentes entre si. As Figuras 4.7, 4.8 e 4.9 mostram respectivamente os gráficos de número de estados visitados por simulação, energia restante ao fim de cada simulação e energia gasta a cada simulação.

A Figura 4.7 mostra que o comportamento explorador realmente tende a encontrar um caminho mais curto (em número de estados percorridos), pois tenta todos os caminhos possíveis. Na segunda simulação com o comportamento explorador há um pico no número de estados visitados, assim o NPC fica conhecendo os pesos de praticamente todas as ações dentro do ambiente, podendo assim tomar o caminho mais curto.

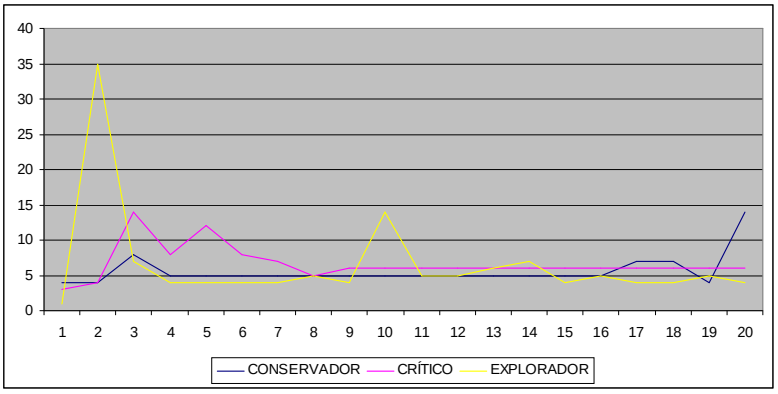


Figura 4.7 Gráfico de número de estados visitados por simulação

Em relação a energia restante após cada simulação é possível perceber que de um modo geral o comportamento Conservador leva uma grande vantagem em relação aos outros comportamentos. Porém, ao final das vinte simulações, percebe-se que o comportamento explorador também tem um bom nível de energia após as simulações. Apenas o comportamento Conservador é mais constante neste quesito, já que está constantemente buscando manter um nível de energia adequado.

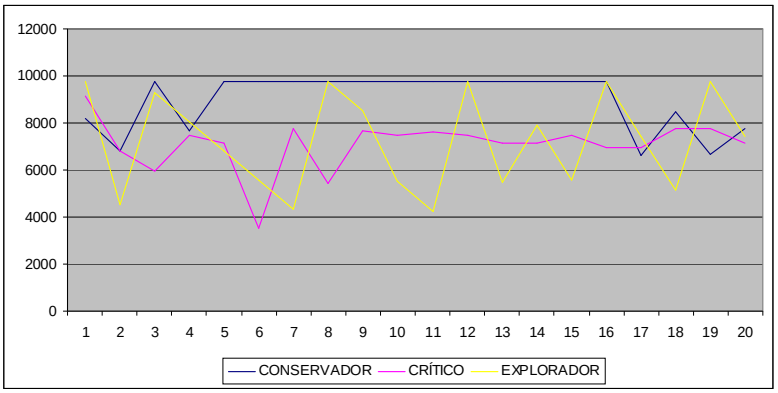


Figura 4.8 Gráfico da energia restante a cada simulação

No que se refere a energia gasta por simulação, é interessante perceber que ao final das vinte simulações o comportamento que mais gasta energia é justamento o Conservador. Mas não é de se admirar, visto que a qualquer momento o NPC se desvia do seu caminho para se recarregar, neste comportamento o importante é manter um nível adequado de energia (e ele o faz, como já foi visto na análise do gráfico de energia restante).

Vale ressaltar a performance do comportamento crítico nas simulações realizadas. O comportamento Crítico é o mais constante de todos, como pode ser visto nos gráficos.

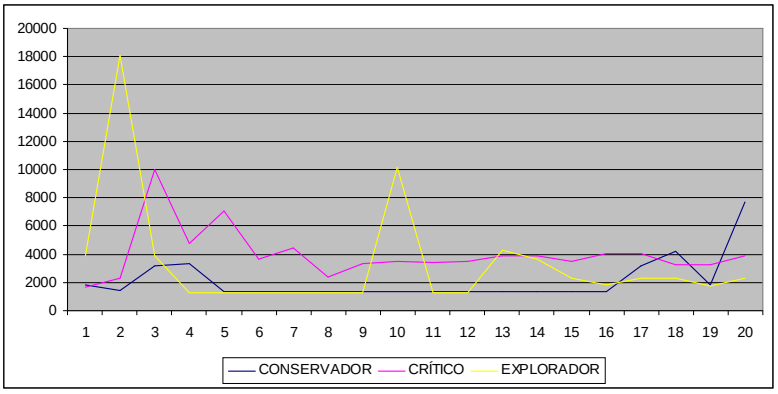


Figura 4.9 Gráfico de energia gasta por simulação

A Tabela 4.1 complementa as informações dos gráficos, mostrango que no geral os comportamentos criados realmente apresentam diferentes maneiras de se movimentar pelo ambiente.

Tabela 4.1 Tabela representando os máximos mínimos e médias das simulações com os três comportamentos

	Nùmero de estados	**Energia gasta**	**Energia sobrando**
COMPORTAMENTO CONSERVADOR			
Máximo	14	7660	9760
Mínimo	4	1316	6596
Média	5,65	2117,6	8952,4
COMPORTAMENTO CRÍTICO			
Máximo	**14**	9972	9144
Mínimo	**3**	1640	3520
Média	**6,65**	3986,6	7095,2
COMPORTAMENTO CONSERVADOR			
Máximo	**35**	18128	9768
Mínimo	**3**	1248	4256
Média	**6,55**	3343,6	7230,6

Capítulo 5

Considerações Finais e Trabalhos Futuros

Prover comportamento humano aos NPCs é uma tarefa complexa e envolve várias áreas da Inteligência Computacional. Novas técnicas e abordagens ao assunto surgem para tentar suprir as deficiências das técnicas disponíveis atualmente. Pelo fato de não ser uma área acadêmica, pois a grande maioria de estudos voltados à Inteligência Artificial para jogos é feita por parte de empresas da indústria de jogos, o tráfego de informação tende a ser mais lento, já que cada empresa tem sua forma de aplicar Inteligência Artificial em seus jogos e abordar os problemas da área.

5.1 Contribuições

Este trabalho procurou avaliar a possibilidade da combinação de técnicas de Inteligência Computacional já difundidas. Especificamente, influenciar a tomada de decisão de um algoritmo de aprendizado por reforço, o Q Learning, por um sistema de controle *Fuzzy* baseado em uma variável específica.

O sistema *Fuzzy* controla o perfil de ação do NPC durante o curso da simulação. As regras implementadas no sistema disparam baseadas no nível energético do NPC e de sua taxa de perda de energia. Isto vai determinar se o NPC pode seguir seu objetivo ou se é necessário que ele recarregue sua energia.

Como cada NPC tem sua própria tabela de valores Q e de recompensa, estrutura proposta neste trabalho proporciona um nível de aprendizado individualizado entre os NPCs do jogo. Seria possível que cada NPC apresentasse um comportamento diferente, devido às diferentes experiências por eles vividas. Coisa que não ocorre na prática em jogos desenvolvidos atualmente.

5.2 Trabalhos Futuros

Para uma melhor avaliação de desempenho do método proposto, além de testes similares em um ambiente maior e mais complexo, seria indicada a simulação com um jogador presente. Uma entidade sobre a qual o usuário teria controle, podendo assim avaliar a resposta dos NPCs em determinadas situações. Este sistema proposto nesta seção está em fase de desenvolvimento de acordo com o que está especificado.

Neste contexto os objetivos dos NPCs não seriam somente ir de um ponto A, a um ponto B (já que isto seria parte do processo de simulação, pois o NPC ainda precisa saber se movimentar pelo ambiente), e sim eliminar o usuário. Sendo assim, o valor de recompensa não seria concentrado em uma única ação como foi feito no trabalho (a recompensa se concentrava totalmente na ação de alcançar o estado final). A eliminação do jogador seria a meta final do NPC, porém, para isto acontecer ele deve ter meios para realizar tal ação.

Para eliminar o jogador o NPC deve reduzir sua energia à zero, para isto ele dispõe de três tipos de armas disponíveis:

- Arma de curto alcance com alto índice de dano
- Arma de médio para longo alcance com baixo índice de dano
- Arma de muito longo alcance com dano máximo, porém muito difícil de acertar

Porém se o NPC tocar no jogador ele próprio é eliminado, então nem sempre é vantajoso o NPC se aproximar demais do jogador.

Como todas estas ações ajudam o NPC a alcançar seu objetivo, elas terão uma recompensa associada atribuída com base na experiência do NPC. E o sistema *Fuzzy*, inicialmente usaria a energia do NPC como parâmetro de controle (ex: se a energia do NPC está baixa, este deve evitar se aproximar do NPC e tentar atacá-lo de longe). Desta forma seria possível avaliar o método proposto em uma situação real de jogo.

Bibliografia

[1] *Videogame history*. Disponível em http://www.atarimuseum.com/ Acesso em: 15 de novembro de 2008

[2] E. L. Thorndike. *Animal Intelligence*. Darien, 1911.

[3] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptative optimal control. In *Proc. Of the American Control Conference*, pages 2143-2146, 1991.

[4] Watkins, C. J. C. H., 1989. *Learning from Delayed Rewards*, Ph.D. Thesys, University of Cambridge, England.

[5] D. E. Rumelhart, G. E. Hinton, and R. J, Williams. Learning Representations by back-propagating errors. *Nature*, 323:533-536, 1986.

[6] J. M. Mendel and E. W. McLaren. *Adaptative, Learning, and Pattern recognition Systems; Theory and Applications*, chapter Reinforcement-Learning control and pattern recognition systems, pages 287-318. New York Academic Press, 70.

[7] Steve Rabin, Ed. **AI game programming Wisdom**. Charles River Media; 1 edition, 2002.

[8] Steve Rabin, Ed. **AI game programming Wisdom 2**. Charles River Media; 2 edition, 2003.

[9] Zadeh, L.A. (1965). "*Fuzzy sets*", *Information and Control* 8 (3): 338-353

[10] S. Korner, "*Laws of thought*," Encyclopedia of Philosophy, Vol. 4, MacMillan, NY: 1967, pp. 414□ 417.