

UNIVERSIDADE FEDERAL DE SANTA CATARINA
Curso Superior de Ciências da Computação

VITOR ANTONIO PELIZZA JR.

ESTUDO DE MÉTODOS PARA FLEXIBILIZAÇÃO DE COMPORTAMENTOS
EM JOGOS DE ESTRATÉGIA EM TEMPO REAL.

Florianópolis - SC, 2008.

VITOR ANTONIO PELIZZA JR.

ESTUDO DE MÉTODOS PARA FLEXIBILIZAÇÃO DE COMPORTAMENTOS
EM JOGOS DE ESTRATÉGIA EM TEMPO REAL.

Trabalho de Conclusão de Curso
apresentado à UFSC como parte dos
requisitos para obter grau de Bacharel em
Ciências da Computação.

Orientador: Mauro Roisenberg.

Coorientador: Dennis Kerr Coelho.

Florianópolis – SC, junho de 2008.

Resumo

A busca pela semelhança de sistemas de inteligência artificial (IA) em jogos com o comportamento dos seres humanos é um dos grandes desafios para desenvolvedores dessa intrigante forma de entretenimento. Objetiva-se com esse trabalho desenvolver um sistema de IA com comportamentos mais flexíveis, tendo como objeto de estudo os jogos RTS (*Real Time Strategy*) – jogos de estratégia em tempo real, assumindo toda complexidade de comportamentos e decisões que esse gênero de jogo abrange. O sistema terá como foco as decisões de alto nível do jogo, ou seja, as decisões que definem o comportamento que deve ser usado por cada equipe jogadora. O propósito é substituir o modo como esses sistemas são implementados na maioria dos jogos, utilizando-se máquina de estados finitos, que faz com que cada comportamento deva ser executado discretamente, deixando visíveis as transições entre os mesmos. Para alcançar tal objetivo o sistema foi desenvolvido baseado no estudo de máquina de estados fuzzy e redes neurais. Tomando-se a segunda como base para o desenvolvimento, busca-se facilitar a definição das transições, já que a mesma é feita através de um conjunto de treinamento com exemplos de situações ideais, dispensando a definição das variáveis necessárias para execução das transições.

Palavras-chave: Jogos RTS. Redes Neurais. Máquina de Estados Fuzzy.

Abstract

The search for the similarities of artificial intelligence (AI) systems of games that simulates the behavior of humans is a major challenge for developers of such intriguing form of entertainment. This research aims to develop an artificial intelligence system with more flexible behaviors, using RTS (Real Time Strategy) games as study object and assuming all the complexity that the behavior and the decisions that this genre of game includes. The system will focus the game's high-level decisions, or, in other words, the decisions that define the behavior that should be used by each game team. The purpose is to replace the way these systems are implemented in most games using finite state machines, in which each behavior is executed in a discreet way, keeping the transitions visible for the user. To achieve this goal, the system was developed based on the study of fuzzy state machines and neural networks. Choosing the second one as the base of the development, the transitions definitions are facilitated, substituting the need of defining the necessary variables for the transition's execution by using a training set with ideal situation examples.

Key-words: RTS Games. Neural Networks. Fuzzy State Machines.

Lista de Ilustrações

Figura 1: Age of empires 3.	15
Figura 2: MEF com 5 estados.	18
Figura 3: O neurônio Artificial.	21
Figura 4: Rede direta.	22
Figura 5: Função sigmoidal.	23
Figura 6: Rede recorrente.	24
Figura 7: Aprendizado supervisionado.	25
Figura 8: Exemplo de ambiente.	26
Figura 9: Grafo de navegação.	27
Figura 10: Desempenho dos tipos de unidades.	30
Figura 11: Relacionamento motor-cena.	33
Figura 12: Funcionamento do motor principal.	33
Figura 13: Personagem.	35
Figura 14: Projétil: flecha.	35
Figura 15: Mina de ouro.	36
Figura 16: Castelo.	36
Figura 17: Mapa e mini-mapa.	38
Figura 18: Integração IA x Modelo Gráfico.	40
Figura 19: MEF da IA de alto-nível.	43
Figura 20: Interface do programa gerador de treinamento da rede neural.	44
Figura 21: Interface do programa treinador da rede neural.	46
Figura 22: Estrutura da rede recorrente utilizada no jogo-exemplo.	48
Figura 23: Conjunto de treinamento da rede neural.	51
Figura 24: Ativação de ataque da equipe 1 em relação aos ciclos do jogo.	52
Figura 25: Ativação de defesa da equipe 1 em relação aos ciclos do jogo.	52
Figura 26: Quantidade de unidades da equipe 1 em relação aos ciclos do jogo.	53
Figura 27: Ativação de ataque da equipe 2 em relação aos ciclos do jogo.	53
Figura 28: Ativação de defesa da equipe 2 em relação aos ciclos do jogo.	53
Figura 29: Quantidade de unidades da equipe 2 em relação aos ciclos do jogo.	54
Figura 30: Estados da equipe gerenciada por IA Simbólica.	55

Figura 31: Níveis de ativação dos estados da equipe gerenciada pela IA baseada em Redes Neurais. 55

Sumário

1 – INTRODUÇÃO	9
1.1 APRESENTAÇÃO DO TEMA	9
1.2 MOTIVAÇÃO DA PESQUISA	11
1.3 OBJETIVOS	12
1.4 METODOLOGIA E PROCEDIMENTOS	13
1.4.1 Pesquisa bibliográfica	13
1.4.2 Desenvolvimento do simulador gráfico	13
1.4.3 Desenvolvimento do módulo de IA	13
1.4.4 Integração	13
1.4.5 Análise dos resultados	14
2 – OS JOGOS “REAL TIME STRATEGY” (RTS).....	15
2.1 UNIDADES	16
2.1.1 UNIDADES ECONÔMICAS	16
2.1.2 EXÉRCITO	16
2.2 CONSTRUÇÃO DA CIDADE	16
2.3 ESTRATÉGIA DE ALTO NÍVEL	17
2.4 BUSCA DE CAMINHOS	17
3 – TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL EM JOGOS	18
3.1 MÁQUINA DE ESTADOS FINITOS (MEF)	18
3.2 MÁQUINA DE ESTADOS FUZZY	19
3.3 REDES NEURAIS ARTIFICIAIS (RNAs)	19
3.3.1 O Neurônio Artificial	20
3.3.2 Redes Diretas	21
3.3.3 Redes Recorrentes	23
3.3.4 Aprendizado da Rede Neural: Retropropagação de Erros	24
3.4 ALGORITMOS DE BUSCA PELO MELHOR CAMINHO	25
3.4.1 Grafos	26
3.4.2 Algoritmos de busca em grafos	27
3.4.3 O algoritmo A*	28
4 – DEFINIÇÃO DO JOGO-EXEMPLO	29
4.1 DEFINIÇÃO DAS CARACTERÍSTICAS DO JOGO	29
4.1.1 Tipos de Unidades	29
4.1.2 Tipos de Construções	31
4.1.5 Tipos Fontes de Recurso	31

4.1.6 Regras do jogo	31
5 – DESENVOLVIMENTO DO MODELO GRÁFICO.....	32
5.1 MOTOR PRINCIPAL E CENA	32
5.2 PERSONAGENS.....	34
5.3 PROJÉTEIS	35
5.4 MINAS DE OURO	36
5.5 CASTELOS.....	36
5.6 MAPA E MINI-MAPA	37
5.7 ESTRUTURAS DE DADOS	38
6 – DESENVOLVIMENTO DO MÓDULO DE INTELIGÊNCIA ARTIFICIAL ...	39
6.1 MODELAGEM DO SISTEMA DE IA.....	39
6.1.1 <i>Análise dos Requisitos e Modelagem Hierárquica</i>	39
6.1.2 <i>Integração com o Modelo Gráfico</i>	40
6.2 IA DE BAIXO NÍVEL	41
6.3 IA DE ALTO NÍVEL.....	42
6.4 IA DE ALTO NÍVEL BASEADA EM REDES NEURAS	43
6.4.1 <i>Gerador de Treinamento</i>	44
6.4.2 <i>Treinador da Rede Neural</i>	45
6.4.3 <i>Modelagem do Sistema Neural</i>	46
6.4.4 <i>Treinamento do Sistema Neural Proposto</i>	50
6.4.5 <i>Análise dos Resultados</i>	51
7 – CONSIDERAÇÕES FINAIS	57
7.1 TRABALHOS FUTUROS	57
8 – REFERÊNCIAS.....	59

1 – INTRODUÇÃO

O desenvolvimento da Inteligência Artificial avança lado a lado com a evolução dos computadores buscando aproximar cada vez mais os conceitos de inteligência a processos computacionais.

O conceito de Inteligência Artificial é alvo de grandes discussões, pelo fato desse conceito ter uma definição muito abstrata. Segundo RUSSEL (1995), por exemplo, a inteligência artificial pode apresentar várias definições, cujo objetivo podem variar da criação de programas de computador que emulem ou os pensamentos ou as ações do ser humano, até desenvolver programas que apresentem ações ou pensamentos racionais.

Os jogos e a Inteligência Artificial (IA) sempre estiveram muito próximos, pois o primeiro vem sendo utilizado como objeto de estudo para observar casos particulares da aplicação da Inteligência Artificial. Segundo SCHWAB (2004), a IA nos jogos é um sistema que faz com que pareça que oponentes controlados por computador (ou elementos cooperativos) tomem soluções inteligentes quando o jogo tem múltiplas escolhas para uma dada situação, resultando em comportamentos que são relevantes, efetivos e úteis. A busca para tornar esses comportamentos e tomada de decisões em algo mais próximo ao comportamento dos seres humanos, motivam cada vez mais o estudo na área da Inteligência Artificial.

Com a evolução dos estudos da Inteligência Artificial na área dos jogos, algumas linhas de pesquisa ganharam destaque, como a IA simbólica e as Redes Neurais. A primeira funciona através de regras pré-estabelecidas, estáticas, onde o sistema infere resultados a partir de valores inseridos no sistema tomando assim suas decisões; a segunda trabalha baseando-se em um histórico de dados para modelar suas decisões, utilizando uma espécie de aprendizado baseado nesse histórico.

1.1 Apresentação do tema

Diferentes técnicas de Inteligência Artificial vem sendo utilizadas na concepção de jogos. Como destaca SCHWAB (2004), no início os sistemas eram somente *scripts* simples, com um conjunto de regras e padrões para controlar o comportamento de objetos nos jogos. Com o avanço desses processos, um

importante passo foi dado: a introdução da MEF (Máquina de Estados Finitos), utilizando regras pré-definidas como controle do sistema de tomada de decisões.

A Máquina de Estados Finitos é uma ferramenta formal simples, porém eficaz, normalmente utilizada para modelagem de sistemas discretos com conjuntos finitos de entradas e saídas, conforme ROISENBERG (2006). É baseada em um conjunto de estados e transições entre esses estados que podem ter ações diferentes. Uma transição é a passagem de um estado a outro, possibilitando a formação de uma seqüência de ações. Então, para cada objeto em um jogo é possível determinar um conjunto de estados. Por exemplo, um soldado pode patrulhar, atacar, fugir e defender-se e para cada um desses estados ele age de maneira diferente, podendo transitar de um estado para outro, conforme forem definidas as ligações entre eles. Essas ligações definem para que estados pode-se transicionar a partir de um outro determinado estado. Segundo SCHWAB (2004), essa simples e poderosa ferramenta ajuda a desmembrar um problema em pequenos problemas mais fáceis de resolver, possibilitando um desenvolvimento com mais flexibilidade e escalabilidade.

Os sistemas que utilizam IA Simbólica se baseiam em regras pré-estabelecidas para inferir resultados. Essas regras são definidas discretamente, fazendo com que o conjunto das decisões também seja discreto.

Atualmente, além das técnicas citadas acima, várias outras mais avançadas são utilizadas, como Redes Neurais, Lógica Fuzzy e Algoritmos Genéticos.

As Redes Neurais são um paradigma da Inteligência Artificial baseado em estudos biológicos do cérebro, que se baseia na simulação de neurônios artificiais conectados, os quais produzem estímulos e reagem de acordo com o estímulo recebido. A rede é treinada com um histórico de informações, “aprendendo” a comportar-se de acordo com um padrão detectado nesse histórico.

A Lógica Fuzzy é uma lógica multivalorada (não somente com valores “verdadeiro” e “falso”), o que possibilita a manipulação de valores não precisos: quase verdade, a maioria, mais ou menos, etc.

Os jogos de estratégia em tempo real (*Real Time Strategy – RTS*) são jogos que possuem como característica principal o gerenciamento de uma equipe de modo eficiente, a fim de derrotar as equipes inimigas. Cada equipe representa uma

sociedade, que geralmente possuem muitas unidades e construções, as quais possuem características específicas como função na sociedade, modos de ataque, etc. Essas características desse estilo de jogo os tornam bastante complexos, visto que podem existir muitos comportamentos e uma grande quantidade de unidades.

Atualmente a MEF ainda é muito utilizada para a modelagem do sistema de IA desses jogos. Isso faz com que esses jogos tenham um comportamento discreto, alternando entre os inúmeros estados pré-determinados pelo desenvolvedor de acordo com regras pré-estabelecidas, deixando bastante aparente para o usuário o ponto da transição entre esses estados. A partir do uso de técnicas mais avançadas como Lógica Fuzzy ou Redes Neurais, pode-se amenizar esse impacto das transições, tornando-as mais suaves, sendo que o resultado das decisões passa a ser de natureza contínua e não mais discreta. Outra vantagem oferecida por essa técnica é a maior facilidade da definição das transições da máquina de estados, pois não é mais necessário defini-las discretamente já que se darão através do conhecimento adquirido pela rede neural através de exemplos.

O módulo de IA dos jogos RTS pode ser dividido basicamente em duas visões:

- Controle da equipe;
- Controle de cada unidade;

Enquanto no primeiro as decisões são tomadas de acordo com a visão do mundo, o segundo é responsável pelo controle individual de cada unidade, o que diminui a complexidade dos comportamentos já que são mais específicos.

Diante desses conceitos, surgem alguns questionamentos: como definir um sistema de inteligência artificial baseado em redes neurais para o módulo de IA responsável pelo controle da equipe de um jogo RTS, visto que suas decisões são as mais significativas para o usuário? Como determinar as entradas e saídas desse sistema? Como integrar esse sistema com o jogo? Qual o impacto sobre o usuário diante da utilização desse tipo de técnica?

1.2 Motivação da pesquisa

Ante uma acelerada evolução de conceitos de Inteligência Artificial e técnicas de desenvolvimento de jogos, e seu crescente mercado mundial, necessita-se um

estudo e análise de novas técnicas a fim de divulgá-las, incentivando a pesquisa nessa área.

Muito há a ser explorado na área de Redes Neurais com relação a jogos. Problemas como a escolha apropriada das entradas para a rede, a necessidade do treinamento da mesma, o consumo de tempo para processamento das informações e as dificuldades na depuração de problemas despertam a grande necessidade de um maior estudo nessa área, que vem revolucionando o mundo dos jogos.

A busca por modelar uma máquina de estados finitos utilizando redes neurais pode obter resultados muito significativos na área dos jogos, sendo que suavizaria o processo de transição entre comportamentos que ainda são muito perceptíveis mesmo nos jogos mais modernos.

Há então a necessidade da pesquisa, análise e comparação da viabilidade do uso dessas técnicas de inteligência artificial, em termos de complexidade, impacto ao usuário, e benefícios em relação ao método mais tradicional (MEF + IA Simbólica).

1.3 Objetivos

Este trabalho tem como principal objetivo a pesquisa e implementação de um sistema de inteligência artificial baseado em redes neurais, para que minimize a visível fronteira entre os comportamentos modelados através de Inteligência Artificial e facilite a definição da mesma, utilizando como ferramenta de estudo os jogos RTS (Real Time Strategy). Para possibilitar a visualização dos resultados será implementado um jogo-exemplo onde será aplicado o módulo de IA desenvolvido.

Seus objetivos específicos compreendem:

- Implementar um sistema de IA utilizando Máquina de Estados Finitos, substituindo o comportamento discreto (IA Simbólica) da tomada de decisões desse sistema por um modelo utilizando Redes Neurais;
- Desenvolver um jogo-exemplo para visualização dos resultados e aplicação do módulo de IA;
- Analisar a diferença do comportamento diante do novo modelo de inteligência artificial adotado.

1.4 Metodologia e procedimentos

A pesquisa em questão possui uma abordagem comparativa.

As etapas de realização do projeto compreendem:

- Pesquisa bibliográfica;
- Desenvolvimento do simulador gráfico;
- Implementação do módulo de IA do jogo, utilizando a técnica de inteligência artificial adotada;
- Integração do módulo de IA com o simulador gráfico;
- Análise dos resultados.

1.4.1 Pesquisa bibliográfica

Esta etapa compreende o levantamento de informações necessárias para a compreensão dos paradigmas de IA Simbólica, Máquina de Estados Finitos e Redes Neurais, assim como o estudo de técnicas necessárias para implementação do módulo de mais baixo nível para o desenvolvimento do simulador gráfico, como algoritmos de busca do melhor caminho (A*).

1.4.2 Desenvolvimento do simulador gráfico

O simulador gráfico é desenvolvido a fim de facilitar o desenvolvimento do módulo de IA, possibilitando a visualização mais concreta do comportamento do mesmo para a detecção de erros ou mesmo melhora do sistema.

1.4.3 Desenvolvimento do módulo de IA

Compreende o desenvolvimento do módulo de IA do jogo a partir das pesquisas teóricas realizadas anteriormente, bem como testes para verificação e validação do sistema.

1.4.4 Integração

São feitas as adaptações necessárias para a integração final do módulo de IA com o simulador, permitindo maior interação do usuário com o mesmo.

1.4.5 Análise dos resultados

Os resultados são analisados e interpretados, para a documentação dos mesmos.

2 – OS JOGOS “REAL TIME STRATEGY” (RTS)

Os jogos de estratégia em tempo real (real time strategy) tem como característica principal a disputa entre duas ou mais civilizações, que devem gerenciar seus recursos para contruir seu exército e dominar o inimigo.

Para cada civilização é necessária uma estratégia que considere a necessidade de ampliar seu exército, evoluir suas habilidades, defender sua integridade e desenvolver sua economia. A economia da civilização é formada a partir da coleta dos recursos disponibilizados pelo ambiente do jogo. Esses recursos podem ser, por exemplo, madeira, ouro e comida.

O exército é ampliado a partir do gasto dos recursos coletados. Assim, a estratégia traçada deve balancear o gasto dos recursos com a coleta dos mesmos e com a velocidade de crescimento do exército.

Um dos jogos mais conhecidos dessa categoria é o “Age of Empires”. Já possui 3 versões, onde cada versão tem suas próprias expansões, incrementando a quantidade de civilizações, mapas, etc. A Figura 1 ilustra a terceira versão do jogo, mostrando um exército atacando sua civilização inimiga.



Figura 1: Age of empires 3.

Segundo SCHWAB (2004), os sistemas de inteligência artificial usados nos jogos RTS são alguns dos mais computacionalmente intensivos, porque envolvem

um grande número de unidades que devem ser gerenciadas e árvores tecnológicas que devem ser navegadas para alcançar os objetivos.

2.1 Unidades

As unidades são quem coletam recursos e compõe o exército. Possuem um módulo de inteligência artificial particular para executar suas tarefas específicas, como andar, evitar obstáculos, atacar, recuar caso não consiga vencer o inimigo entre outras ações que podem ser definidas para o mesmo. Segundo SCHWAB (2004), esse comportamento individual deve ser considerado secundários, dando prioridade às ordens dadas pelo controle central da civilização.

2.1.1 Unidades econômicas

São as unidades que coletam recursos, contribuindo com a economia de sua sociedade. Possuem poder muito baixo de ataque e defesa, o que os torna vulneráveis a inimigos.

Essas unidades devem ser consideradas no plano de defesa da sociedade, pois um ataque do inimigo sobre elas pode desestabilizar a economia e diminuir a velocidade de criação do exército e melhoria das habilidades da mesma.

2.1.2 Exército

O exército tem a função de defender sua civilização e atacar a sociedade inimiga. É desenvolvido de acordo com a disponibilidade de recursos, coletados pelas unidades econômicas, consumindo uma determinada quantia dos mesmos.

O sistema central de controle do time deve considerá-lo em suas decisões para distribuir ou agrupar essas unidades, de acordo com a necessidade de defesa ou ataque. Deve equilibrar seu desenvolvimento com a necessidade do aumento da população das unidades econômicas, para manter o equilíbrio entre a economia e o setor militar da sociedade.

2.2 Construção da cidade

Segundo SCHWAB (2004), a decisão da posição das construções deve considerar vários aspectos, como o agrupamento das construções de função

semelhante – com função econômica por exemplo – ou a distribuição das mesmas, a distribuição dos espaços para ganhar mais visibilidade ou o agrupamento para manter uma estrutura de mais fácil defesa ou mesmo a proximidade das construções militares de uma região de fácil saída da cidade.

2.3 Estratégia de alto nível

Este é o módulo de controle da sociedade. É responsável pelo planejamento da utilização e evolução de toda sua estrutura. Segundo SCHWAB (2004), suas ações envolvem um grande volume de unidades, captando a resposta de cada unidade para ajudar em sua tomada de decisões.

Esse nível de estratégia define o comportamento de cada sociedade, balanceando a economia com ações e ataque, defesa e desenvolvimento de suas tecnologias.

2.4 Busca de caminhos

Segundo SCHWAB (2004), a busca de caminhos é uma das grandes preocupações quanto à processamento dos jogos RTS. Geralmente é necessário controlar um grande número de unidades que podem andar para regiões distintas do mapa, e devem fazê-lo por um caminho válido (evitando obstáculos).

3 – TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL EM JOGOS

Para alcançar o resultado desejado para possibilitar a comparação entre os modelos de Inteligência Artificial foram estudadas técnicas que possibilitassem o desenvolvimento de tais modelos.

O primeiro modelo segue a linha de desenvolvimento das técnicas atuais, utilizando basicamente, no módulo de alto nível da IA, máquinas de estados finitos. O segundo, como objetivo de atingir o propósito descrito, utilizará os conceitos de máquina de estados fuzzy e redes neurais, explicados através do desenvolvimento da pesquisa.

3.1 Máquina de Estados Finitos (MEF)

A máquina de estados finitos é “uma ferramenta formal simples normalmente utilizada para modelagem de sistemas discretos com conjuntos finitos de entrada e de saída” (ROISENBERG, 2006). Suas entradas são os estímulos recebidos do ambiente e suas saídas os estados ou comportamentos. A transição de um estado a outro se dá por meio de uma função que o infere a partir do estado atual mais a entrada lida. Supondo i como entrada, x pertencente ao conjunto de estados possíveis X , em função do tempo t , essa função de transição é descrita como:

$$x(t + 1) = f(x(t), i)$$

A Figura 2 ilustra a estrutura de uma MEF com 5 estados, representados pelos círculos e suas transições, representadas pelas setas.

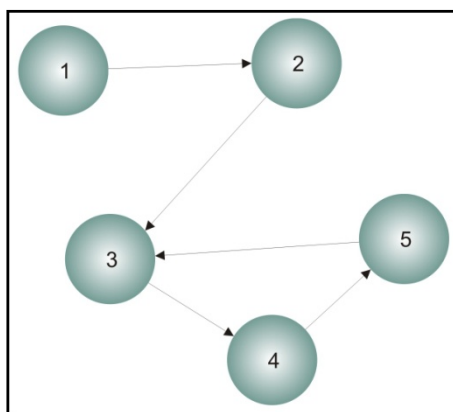


Figura 2: MEF com 5 estados.

3.2 Máquina de Estados Fuzzy

São máquinas de estados baseadas nas noções da lógica fuzzy (lógica nebulosa), comumente definida como um superconjunto da lógica booleana para representar valores de verdade parciais, conforme SCHWAB (2004).

Segundo LOPES (2004), na lógica fuzzy uma sentença pode ser parcialmente verdadeira, associando-se um grau de verdade para a mesma, que é um valor numérico em um intervalo determinado, representando o quanto é verdadeira tal sentença. Os limites inferior e superior desse intervalo representam os valores correspondentes a “totalmente verdadeiro” e “totalmente falso”.

Nas transições, a máquina de estados fuzzy associa um grau de verdade para cada estado, sendo que o estado corrente passa a ser um conjunto de estados, cada um com seu valor de pertinência. Cada estado representa uma situação ideal que, enquanto as MEFs definem exatamente essas situações, esse modelo de máquina pode encontrar valores intermediários (LOPES 2004).

As condições de transição são valores fuzzy, que são valores que se aproximam do raciocínio humano, como *bastante*, *longe*, *perto*... A avaliação dessas condições, que resulta em um valor de verdade a partir dos dados de entrada, é dada a partir dos processos de fuzzificação (transformação dos valores de entrada em valor fuzzy), inferência (cálculo do valor de verdade através da função de pertinência de cada conjunto fuzzy) e defuzzificação (transformação inversa à da entrada).

3.3 Redes Neurais Artificiais (RNAs)

Segundo HAYKIN (1991), o cérebro é um computador extremamente complexo, não-linear e paralelo. A partir de sua estrutura, constituída de neurônios, é capaz de realizar computações como reconhecimento de padrões, percepção e controle motor, muito mais rapidamente do que o mais rápido computador digital convencional existente.

As redes neurais artificiais (RNA) são modelos eletrônicos baseados na estrutura neural do cérebro, sendo composta por neurônios artificiais

interconectados. Segundo HAYKIN (1991), se assemelham ao cérebro em dois aspectos:

1. O conhecimento é adquirido através de um processo de aprendizagem;
2. A força das conexões entre os neurônios, conhecida como peso sináptico, é usada para armazenar o conhecimento adquirido.

Existem várias topologias de redes neurais, porém as principais são as redes diretas e as redes concorrentes. A principal diferença entre as duas é a direção da propagação dos dados: enquanto na rede direta os dados se propagam somente da entrada para a saída, nas redes recorrentes as informações podem se propagar da saída para a entrada o que a torna muito mais complexa. Dentre essas várias topologias de redes neurais, segundo SCHWAB (2004), se destacam na área dos jogos as redes diretas, pois, são mais fáceis de entender, mais fáceis de configurar e menos custosas de executar.

3.3.1 O Neurônio Artificial

Uma rede neural é composta por vários neurônios artificiais, que é uma estrutura que representa o neurônio biológico na rede.

Segundo FREEMAN & SKAPURA (1991), como um neurônio biológico, o neurônio artificial possui várias entradas porém somente uma saída, que pode ser conectada a vários outros neurônios na rede. Cada conexão de entrada é associada a um peso ou força de conexão. O valor de entrada do neurônio é calculado somando-se todas as entradas multiplicadas por seus respectivos pesos, conforme visualizado na Figura 3, onde cada termo E_n representa uma conexão de entrada e cada termo P_n o peso associado à conexão.

O termo Bias ajuda na convergência da rede em sua etapa de treinamento e é considerado como mais um peso do neurônio. Esse termo faz com que os pesos sejam ajustados mais rapidamente pois, adicionado o seu valor à entrada de cada neurônio, acrescenta um estímulo extra à função de ativação. Seu valor é constante e geralmente é assumido como 1 ou -1.

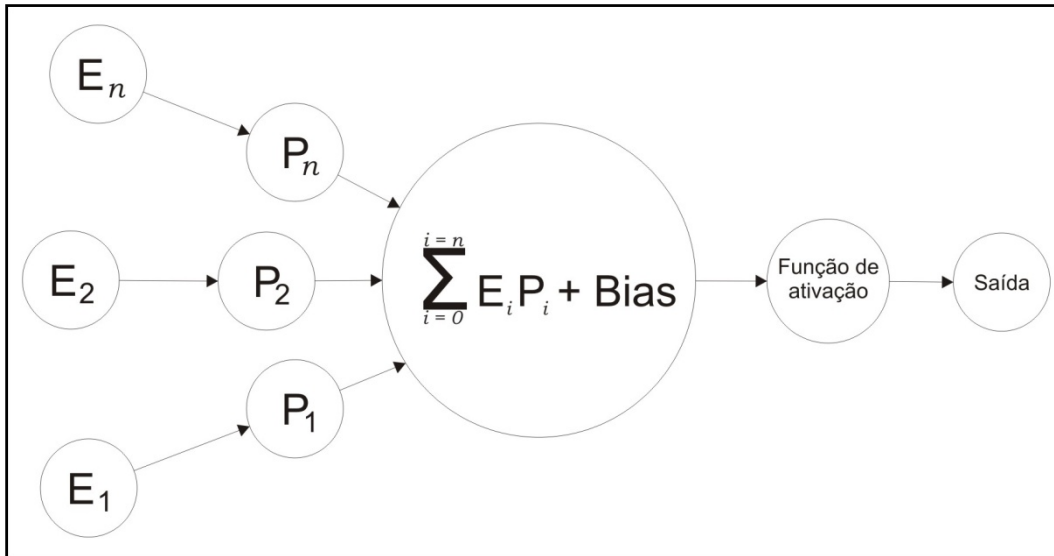


Figura 3: O neurônio Artificial.

Fonte: SCHWAB (2004).

A saída do neurônio é calculada a partir da entrada calculada aplicada à função de ativação. As funções de ativação mais comuns são a função sigmoideal e a função tangente hiperbólica.

3.3.2 Redes Diretas

As redes diretas ou “feedforward” são redes neurais cujo grafo não possui ciclos. É dividida em camadas de neurônios artificiais, onde cada neurônio de uma camada é conectado a todos os neurônios da camada seguinte. A propagação das informações é feita somente em um sentido, de uma camada i para uma camada j , sendo que $i < j$.

As camadas da rede direta são denominadas: camada de entrada, camada intermediária e camada de saída. A Figura 4 mostra um exemplo de rede direta, com 3 neurônios na camada de entrada, 6 na intermediária e 2 na camada de saída.

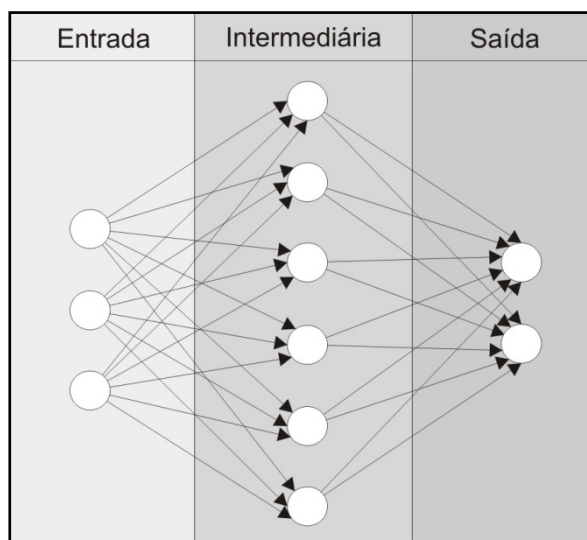


Figura 4: Rede direta.

Fonte: SCHWAB (2004), p. 456.

A rede pode ser composta por uma ou mais camadas intermediárias, porém, segundo SMAGT (1996), somente uma camada intermediária é suficiente para aproximar qualquer função com uma precisão arbitrária, utilizando uma função de ativação não linear para os neurônios. Na maioria das aplicações uma rede direta com somente uma camada intermediária utiliza como função de ativação a função sigmoide, segundo SMAGT (1996). Essa função é representada pela Figura 5, juntamente com seu gráfico cartesiano. Segundo BUCKLAND (2002), o termo a dessa função função, é a ativação do neurônio e p é a velocidade da curva. Para valores maiores de p a curva tende a ficar menos acentuada e para valores menores, mais acentuada.

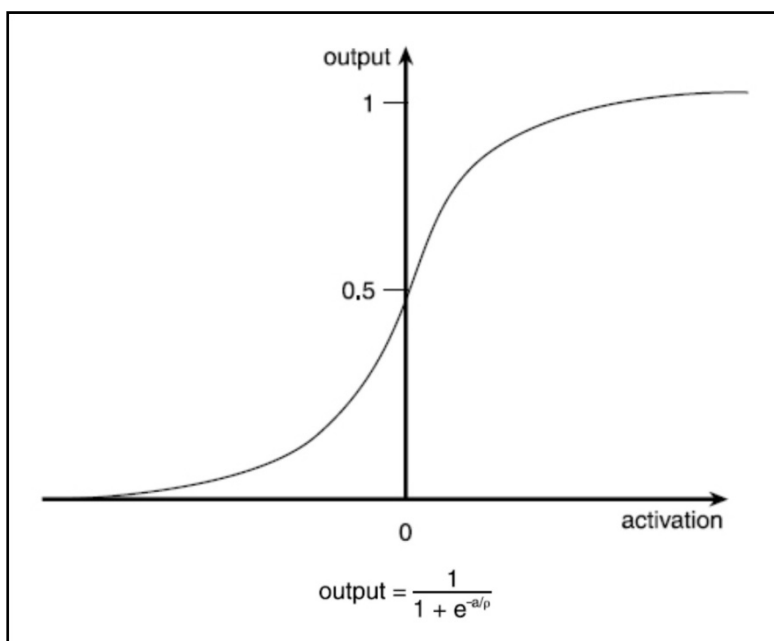


Figura 5: Função sigmoideal.

Fonte: BUCKLAND (2002), p. 246.

Para ajustar os pesos da rede de modo a obter a saída aplica-se um método de aprendizado, cujo objetivo é ajustar os pesos de cada neurônio da rede neural para obter o comportamento desejado. O método mais conhecido de treinamento desse tipo de rede é a retropropagação de erros.

3.3.3 Redes Recorrentes

As redes recorrentes são um tipo de rede desenvolvido para aprender padrões seqüenciais ou variantes de acordo com o tempo. Nesse tipo de rede um grupo específico de unidades recebe sinais de resposta de um passo de tempo anterior, onde essas unidades são classificadas como unidades de contexto, segundo FAUSETT (1994).

A arquitetura desse tipo de rede é parecida com a da rede direta, exceto pela ligação de retro-alimentação das entradas com o sinal da saída do tempo anterior. Essa arquitetura é ilustrada na Figura 6.

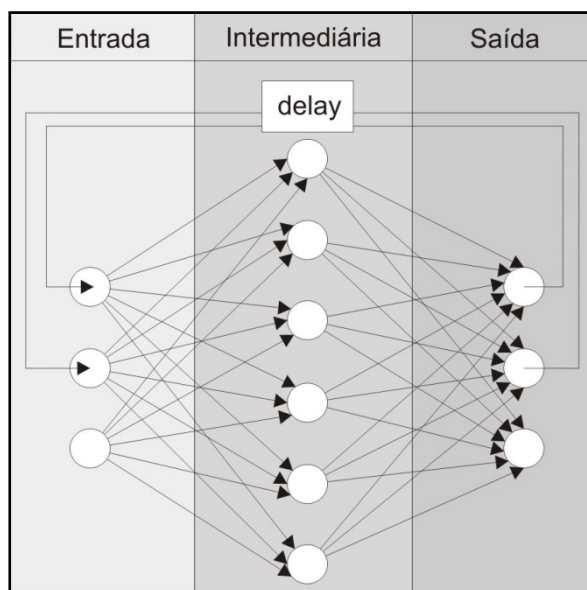


Figura 6: Rede recorrente.

Com essa topologia de rede neural, segundo OMLIN (1998) e UNAL(1994), pode-se codificar uma máquina de estados finitos, onde cada neurônio, tanto das entradas quanto das saídas, ligados a retro-alimentação representa um estado da MEF; as entradas restantes representam os dados necessários para análise da transição do estado.

3.3.4 Aprendizado da Rede Neural: Retropropagação de Erros

A retropropagação de erros (“error backpropagation”) é uma técnica de aprendizado supervisionado que visa propagar os erros da camada de saída da rede para as anteriores, de acordo com a força de ligação de cada neurônio. É classificada como aprendizado supervisionado pois o resultado da saída da rede é analisado por um “professor”, que, a partir de seu conhecimento, toma as medidas necessárias para o ajuste da rede.

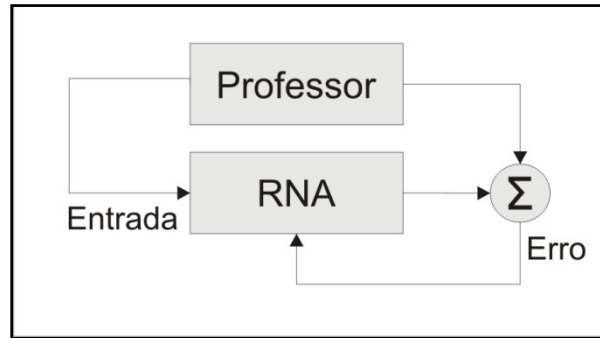


Figura 7: Aprendizado supervisionado.

Para efetuar o aprendizado da rede, é necessário um conjunto de exemplos com entradas e saídas ideais. Assim, para cada exemplo, a entrada é aplicada a rede, mensurando o erro das saídas, comparando-as com as saídas ideais. Calculado o erro, o mesmo é retropropagado para as camadas anteriores. Com o cálculo do erro de cada neurônio, pode-se então ajustar os pesos de cada conexão de forma a minimizar esse erro.

Quando todos os exemplos foram aplicados à rede diz-se que completou-se uma época. O treinamento é encerrado quando foi atingido o número máximo de épocas ou foi encontrado um erro aceitável.

3.4 Algoritmos de busca pelo melhor caminho

Os algoritmos de busca pelo melhor caminho têm grande presença nos jogos RTS. São utilizados em situações onde um objeto necessita se deslocar de um ponto a outro, onde deve seguir um caminho possível de ser traçado e evitar obstáculos.

Segundo SCHWAB (2004), antigamente algoritmos de busca em jogos eram praticamente inexistentes, ao passo que os ambientes eram totalmente abertos, livre de obstáculos. Porém, com o desenvolvimento na área de jogos, os ambientes foram se tornando cada vez mais condizentes com o mundo real, o que exige o uso desses algoritmos.

Para fazer a busca pelo caminho entre dois pontos, segundo BUCKLAND (2005) o ambiente do jogo deve ser particionado em uma estrutura de dados que os algoritmos podem manipular: um grafo de navegação.

3.4.1 Grafos

Os grafos são uma abstração matemática para representação de problemas como, por exemplo, os estados de um personagem de um jogo, ou os pontos possíveis de um ambiente onde o personagem pode se locomover, chamado de grafo de navegação.

Um grafo é formalmente definido por $G(V, A)$, onde:

- V = conjunto não vazio: os vértices ou nodos do grafo;
- A = conjunto de pares ordenados $a=(v,w)$, v e $w \in V$: as arestas do grafo.

Um exemplo de grafo de navegação é dado na Figura 9, onde cada ambiente do mapa, representado na Figura 8, possível de ser acessado é representado por um nodo.

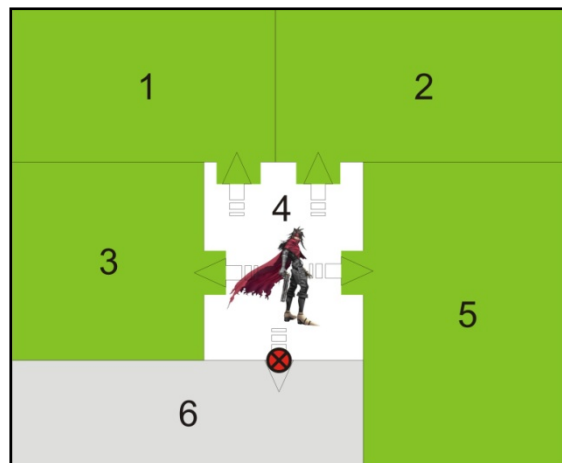


Figura 8: Exemplo de ambiente.

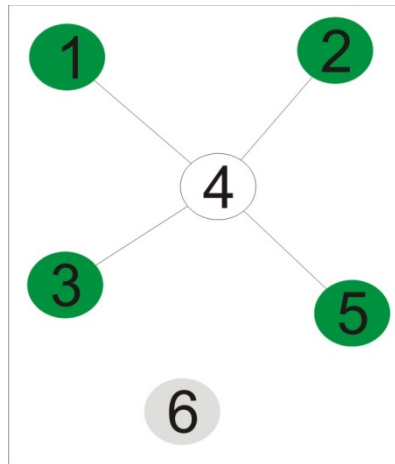


Figura 9: Grafo de navegação.

A partir da representação de ambientes por grafos (grafo de navegação), pode-se manipulá-lo por algoritmos como os algoritmos de busca, muito úteis nos jogos RTS para locomoção dos objetos pelo mapa.

3.4.2 Algoritmos de busca em grafos

Os algoritmos de busca em grafos visam procurar por um caminho entre dois nodos. Dependendo da natureza do problema, esses dois nodos podem representar o problema e a solução, ou então a coordenada atual de um personagem do jogo e o destino da trajetória a ser percorrida.

Existem dois tipos de algoritmos de busca: os não-informados que não possuem informação sobre o problema, a não ser sua definição; e os informados que usam conhecimentos específicos do problema além de sua definição (RUSSEL, 2005).

Os algoritmos de busca não-informados buscam o destino exaustivamente. “Tudo o que eles podem fazer é gerar sucessores e distinguir o estado final de um estado não-final” (RUSSEL, 1995, p.73). Dois exemplos desse tipo de algoritmo são a busca em largura e a busca em profundidade.

A busca em largura expande o todos os sucessores do nodo raiz e depois seus sucessores e assim por diante. A busca em profundidade expande até o nodo mais profundo para cada sucessor.

Os algoritmos de busca informados buscam o destino considerando informações específicas sobre o problema (heurística). Cada nodo é selecionado para expansão baseado em uma função de avaliação $f(n)$ (RUSSEL, 1995).

A heurística ajuda a filtrar somente os melhores nodos a serem expandidos, tornando a busca temporalmente e espacialmente mais eficiente. A função heurística - **$h(n)$ = custo estimado do caminho mais curto do nodo n até a meta** - é então a chave desses algoritmos. Se a heurística nunca superestima o custo para alcançar a meta diz-se que ela é admissível.

Dois exemplos dessa estratégia de busca são a busca do melhor primeiro e a busca A^* (lê-se “A Estrela”). A busca do melhor primeiro avalia os nodos somente a partir da heurística. Logo $f(n) = h(n)$. A busca A^* é mais usada estratégia de busca de caminho em jogos e será abordada no próximo capítulo.

3.4.3 O algoritmo A^*

A busca A^* procura minimizar o custo total estimado da solução. Avalia os nodos a partir da heurística e de sua distância até o início do caminho: $f(n) = g(n) + h(n)$. Sendo assim, **$f(n)$ = custo estimado da solução menos custosa através de n** .

O modo com que o algoritmo A^* avalia os nodos torna essa estratégia mais do que razoável: data que a função heurística $h(n)$ seja admissível, a busca A^* é tanto completa quanto ótima (RUSSEL, 1995). Em outras palavras, nenhum outro algoritmo de busca irá expandir menos nodos na busca pelo caminho menos custoso entre a fonte e o destino (BUCKLAND, 2005).

Em jogos RTS pode-se calcular facilmente a heurística da distância em linha reta entre dois pontos, que é uma ótima heurística admissível. As características do algoritmo A^* e essa facilidade do cálculo dessa heurística fazem desse algoritmo o mais utilizado nesse estilo de jogo.

4 – DEFINIÇÃO DO JOGO-EXEMPLO

Para analisar e comparar as diversas técnicas de IA nos jogos RTS, foi desenvolvido um jogo-exemplo para servir como base para a implementação das mesmas.

O jogo foi criado seguindo um sub-grupo selecionado das características dos jogos RTS, definindo-se regras de jogabilidade as quais devem ser seguidas pelo sistema de IA de cada civilização.

4.1 Definição das Características do Jogo

As características do jogo a ser desenvolvido para ser usado como base para o estudo das técnicas de IA serão divididas em três partes: a definição dos tipos de unidades, de construções e regras do jogo.

4.1.1 Tipos de Unidades

São usadas somente unidades de combate, restringindo assim as decisões do sistema de IA com relação ao tipo de unidade somente às situações de ataque e defesa.

A definição dos tipos de unidades tem importante papel sobre a decisão do sistema de IA. A complexidade dessas decisões aumenta pelo fator da diferença de desempenho de cada tipo de unidade sobre outro. Essa diferença de desempenho se dá pelas características físicas de cada tipo de unidade que serão:

- Velocidade;
- Vida;
- Defesa;
- Distância de ataque;

Os tipos de unidades são definidos abaixo e possuem um comportamento de desempenho conforme a Figura 10:

- **Infantaria:** São unidades que possuem armas de curto alcance. São unidades fortes, com grande poder de ataque e bom desempenho contra cavalaria. Não são muito velozes, o que os põe em grande desvantagem contra artilharia.

- Artilharia: São unidades que possuem arco e flecha como arma, portanto têm ataque de longo alcance. Tem bom desempenho de ataque contra infantaria, pois, além de conseguirem atacar de longa distância, tomam vantagem da baixa velocidade de locomoção dessas unidades. Também não são muito velozes por não dispor de meio de locomoção.
- Cavalaria: Possuem armas de curto alcance, porém são velozes. Não possuem proteção eficiente contra unidades terrestres, o que os deixa fracos contra infantaria; porém possuem bom desempenho contra artilharia, pois conseguem alcançar o inimigo rapidamente e atacar sem levar muito dano de projéteis.

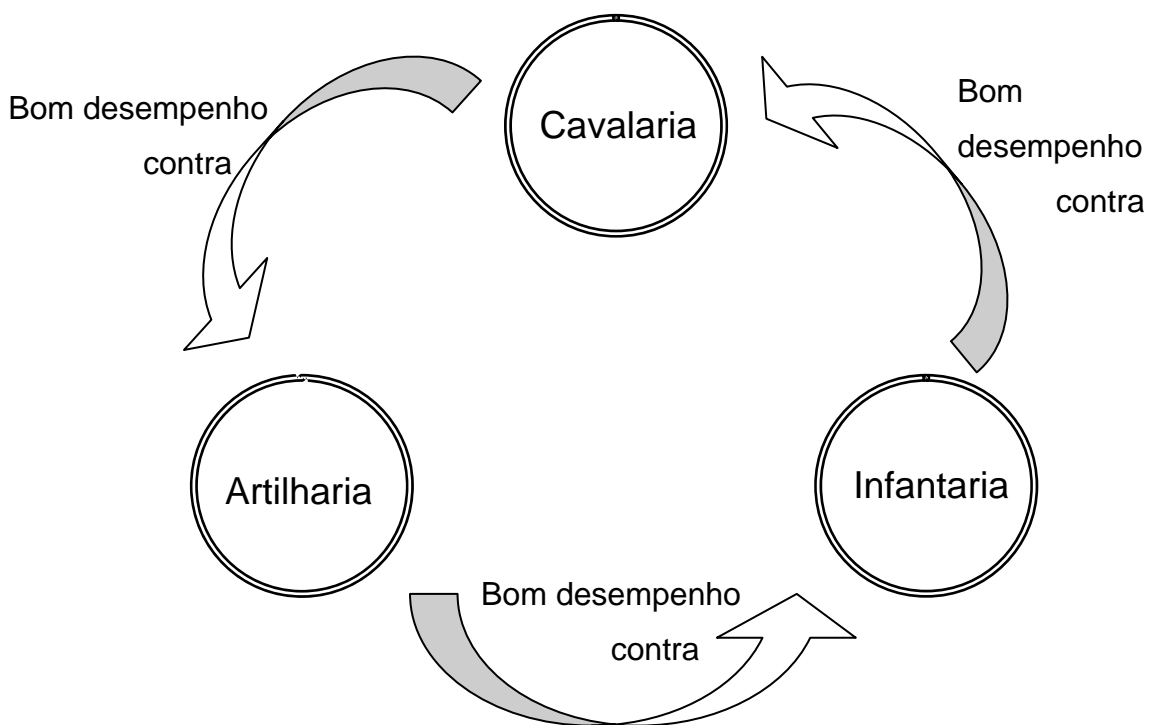


Figura 10: Desempenho dos tipos de unidades.

4.1.2 Tipos de Construções

São utilizados somente castelos no jogo para restringir a complexidade das decisões do módulo de IA. A definição dos tipos de construções têm um papel tão importante quanto a definição dos tipos de unidades, pois devem ser consideradas nas diversas situações do jogo como um ponto estratégico a ser defendido.

O castelo é o símbolo das civilizações, causando a derrota de sua civilização caso seja destruída pelo inimigo. Pode construir novas unidades para aumentar o exército de sua equipe.

4.1.5 Tipos Fontes de Recurso

É utilizado somente ouro como fonte de recurso no jogo. Esses recursos são importantes para a construção de mais unidades de exército; logo, são consideradas pelas decisões de IA caso a necessidade do aumento do exército ou mesmo a defesa do recurso para impedir o crescimento do exército inimigo.

4.1.6 Regras do jogo

O jogo possui duas civilizações rivais lutando pelo objetivo do jogo que é derrotar a civilização inimiga, destruindo seu castelo.

Cada equipe inicia com 1 unidade e pode contruir novas unidades de acordo com sua quantidade de ouro acumulada.

O custo de construção de cada unidade é 200 unidades de ouro. Cada equipe inicia com 100 unidades de ouro.

Para um equipe conquistar uma mina de ouro e usufruir de sua geração de ouro deve ter mais exércitos do que seu inimigo no raio de domínio da mina. A mina gera 10 unidades de ouro por segundo para o dominador da mesma. O raio da mina de ouro é uma área circular em torno da mina de 200 pixels de raio.

5 – DESENVOLVIMENTO DO MODELO GRÁFICO

Este capítulo trata do desenvolvimento do modelo gráfico do jogo-exemplo, baseado nas características definidas no capítulo 4.

O sistema provê a modelagem dos componentes visuais – personagens, construções, recursos naturais – bem como estruturas de dados que centralizam informações referentes ao conhecimento do jogo a ser disponibilizado para o módulo de IA. Engloba também o a estrutura de funcionamento básica composta pela atualização e renderização dos objetos. Com o desenvolvimento dessa estrutura será possível a aplicação do módulo de IA para a visualização de seu comportamento.

Esse módulo não aborda sistemas de interação com usuários humanos, visto que o jogo não possuirá tal tipo de interação, já que o objetivo é somente observar o comportamento das equipes jogadoras diante das técnicas de inteligência artificial implementadas.

5.1 Motor Principal e Cena

O motor principal do jogo é também chamado de “game-loop”. É o responsável pela centralização do processo da constante atualização e renderização dos componentes. O motor executa esse processo repetidamente até que seja sinalizado o fim do jogo. Cada execução desse processo é chamado de ciclo.

Para simplificar o relacionamento entre o motor e os componentes visuais, foi utilizada uma estrutura chamada “cena”, criando assim uma interface com os elementos do jogo a serem gerenciados. A Figura 11 ilustra esse relacionamento.

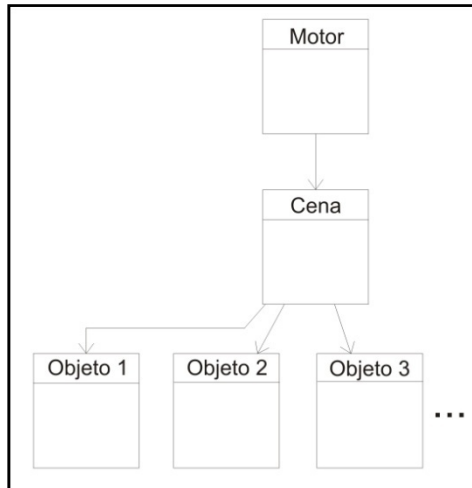


Figura 11: Relacionamento motor-cena.

A cena é uma estrutura que contém os objetos necessários para criar um contexto de interação com o usuário. Ao ser atualizada, pode indicar ao motor do jogo qual a próxima cena a ser manipulada. Por exemplo: um menu pode ser uma cena que possui botões, imagens, animações, textos, etc., onde quando um item é selecionado a mesma retorna para o motor principal do jogo a próxima cena a ser manipulada, que pode ser um sub-menu.

A Figura 12 ilustra o funcionamento do motor, que atualiza e renderiza a cena a cada ciclo executado.

```

enquanto (verdade) {
    novaCena = cenaAtual.atualiza(tempo);
    enquanto (novaCena != cenaAtual) {
        cenaAtual = novaCena;
        novaCena = cenaAtual.atualiza(tempo);
    }
    cenaAtual.renderiza(buffer);
}
  
```

Figura 12: Funcionamento do motor principal.

A rotina de atualização da cena possui como parâmetro a diferença de tempo desde sua última atualização. Isso faz com que a cena que está sendo manipulada tenha noção de tempo conseguindo, por exemplo, coordenar a movimentação de um objeto de acordo com o tempo decorrido.

A rotina de renderização da cena possui como parâmetro *buffer* a ser utilizado para o processo de renderização de seus componentes. Após completa essa rotina o *buffer*, então renderizado, é mostrado na tela. Esse processo é feito através da cópia do *buffer* usado pela cena – chamado de *back-buffer* - para outro *buffer*, que é renderizado na tela – chamado de *front-buffer*.

5.2 Personagens

O personagem é um objeto animado capaz de gerenciar sua própria animação dependendo do estado associado a ele. Possui também uma indicação do tipo de unidade que representa: arqueiro, cavaleiro ou guerreiro.

As animações são compostas por quadros, que são imagens estáticas renderizadas uma após a outra dando a impressão de movimento. Para então compor a animação de cada personagem é associada a ele uma imagem composta de quadros, onde cada linha é uma animação (representa um estado do personagem) e cada coluna representa um quadro da animação. Além da imagem, é necessária a definição do tempo de cada quadro, ou seja o tempo de espera entre a renderização de um quadro e outro.

A Figura 13 mostra o formato de uma imagem que representa a animação de um personagem. Sendo que esta figura possui 6 linhas e 4 colunas, o personagem terá 6 animações com 4 quadro cada uma.

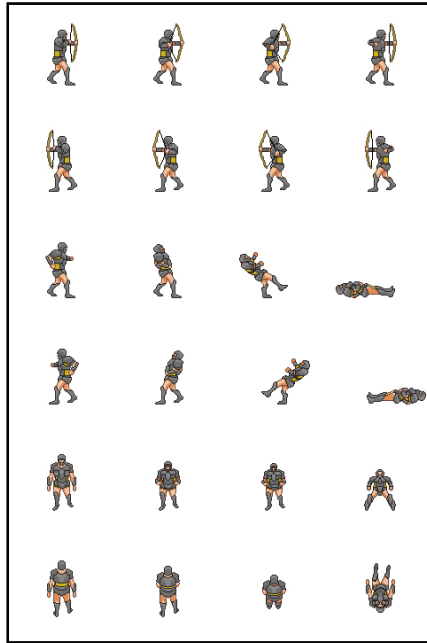


Figura 13: Personagem.

5.3 Projéteis

Os projéteis são artefatos disparados pelos personagens com objetivo de causar dano a um oponente. Existe somente um tipo de projétil: a flecha.

A flecha é um objeto animado, com quatro quadros de imagens, onde cada quadro representa uma direção. É capaz de se locomover de um ponto a outro, e causar dano aos adversários.

É função do sistema de IA gerenciar o ciclo de vida e a movimentação dos projéteis, que são removidos do jogo caso atinjam um inimigo, causando dano a ele, ou cheguem ao seu destino.

A Figura 14 mostra o formato da imagem que compõe a flecha, que possui somente 4 colunas (4 quadros) e 1 linha (1 animação).

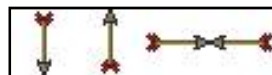


Figura 14: Projétil: flecha.

5.4 Minas de ouro

Esse componente visual é composto por uma imagem estática, ou seja, não é animado. É função de seu sistema de IA distribuir ouro para a equipe que estiver dominando a mesma e saber identificar qual equipe está a dominando, bem como os personagens que estão ao seu redor.

Cada mina de ouro possui um raio que delimita a região de conquista. A equipe que conter mais personagens nessa região ganha o domínio da mina de ouro.



Figura 15: Mina de ouro.

5.5 Castelos

Os castelos são compostos por uma imagem estática e um indicador de vida. Cada castelo possui uma bandeira de cor diferente para identificar a equipe pertencente.

É de sua responsabilidade a criação de novos personagens para sua equipe. O controle dessa ação é dado pelo seu módulo de IA, que o indica quando deve-se criar um novo personagem. Caso sua equipe tenha ouro suficiente o personagem desejado é criado e posto em controle de sua equipe.

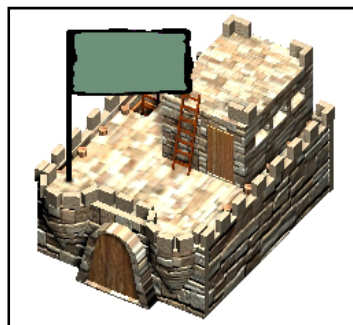


Figura 16: Castelo.

5.6 Mapa e Mini-mapa

A função do mapa é delimitar o espaço de locomoção dos personagens e proporcionar também a visualização parcial do espaço do jogo.

O mapa não possui restrição de tamanho, portanto pode ocupar regiões fora da tela. Para isto possui um sistema de câmera que seleciona a região a ser renderizada. A câmera possui um sistema de movimentação de acordo com a posição do mouse sobre a tela.

O mini-mapa é um componente para visualização total locomoção rápida do mapa. Renderiza um retângulo para simbolizar a posição da câmera no mapa e ícones pré-determinados para simbolizar os objetos, como personagens, minas de ouro ou construções.

Com o mini-mapa pode-se observar eventos – como a conquista de mina de ouro ou encontro de personagens inimigos – que não estão na área visível e locomover a câmera do mapa rapidamente para o local.

A Figura 17 mostra o visual desses componentes. À direita o mapa sendo visualizado parcialmente e no canto inferior esquerdo o mini-mapa indicando a região visualizada e os demais componentes visuais do jogo.



Figura 17: Mapa e mini-mapa.

5.7 Estruturas de dados

Cada civilização possui uma estrutura de dados que centraliza o acesso às informações do jogo, para facilitar o acesso aos objetos tanto para as rotinas de renderização quanto para o módulo de IA.

Fazem parte dessa estrutura de dados:

- Os personagens;
- Os inimigos;
- O castelo;
- O castelo inimigo;
- As minas de ouro não conquistadas;
- As minas de ouro conquistadas;
- As minas de ouro conquistadas pelo inimigo;
- Os projéteis disparados pelos personagens;
- A quantidade acumulada de ouro.

6 – DESENVOLVIMENTO DO MÓDULO DE INTELIGÊNCIA ARTIFICIAL

Este capítulo trata do desenvolvimento do módulo de IA do jogo-exemplo. É apresentada a modelagem do sistema, mostrando como é integrado no modelo gráfico, posteriormente relatando os aspectos específicos do desenvolvimento, tanto do modelo simbólico quanto da nova modelagem neural.

6.1 Modelagem do Sistema de IA

Para modelar o sistema de IA deve-se, inicialmente, prever quais os objetos que devem ser incluídos nesse sistema. A partir dessa análise pode-se definir como o sistema deve ser integrado no jogo.

O desenvolvimento do sistema de IA foi feito utilizando máquina de estados finitos utilizando IA simbólica para o modelo de transições dos estados. Feita a validação dessa implementação, é desenvolvido o mesmo sistema utilizando então Redes Neurais para as decisões das transições de estados, procurando assim minimizar a delimitação de cada estado, deixando as transições entre eles mais suaves e tentando facilitar a definição das regras de transição, visto que serão calculadas através do conhecimento da rede adquirido pelo treinamento através de exemplos.

6.1.1 Análise dos Requisitos e Modelagem Hierárquica

O jogo é composto por duas equipes, as quais possuem suas unidades para seu desenvolvimento e para a conquista da vitória. Definidas essas características principais previamente, pode-se classificar os comportamentos de cada um desses elementos para poder defini-los.

Cada equipe possui um controle central, como um general, para gerenciar seu exército. Pode mandar ações a serem executadas para suas unidades, que, por sua vez, também possuem um controle independente, capaz de responder a essas ações e a estímulos recebidos do ambiente.

Essas características sugerem uma topologia de IA hierárquica, visto que uma das camadas de controle é mais genérica (controle central da equipe), a qual controla a camada mais específica (controle de cada unidade). De acordo com

ROISEMBERG (2006), a arquitetura hierárquica nos possibilita a tradução de comportamentos e tarefas complexas em autômatos simples e de fácil implementação. Dessa forma o módulo de IA será dividido em:

- **IA de alto-nível:** IA de controle da equipe, ao passo que fica no topo da hierarquia;
- **IA de baixo-nível:** IA de controle de cada unidade.

É na IA de alto-nível que será implementada posteriormente a máquina de estados controlada pela Rede Neural, visto que é nessa camada que as decisões mais significativas e perceptíveis são tomadas através dos estímulos recebidos do ambiente.

6.1.2 Integração com o Modelo Gráfico

A Figura 18 ilustra a integração do módulo de IA com o modelo gráfico.

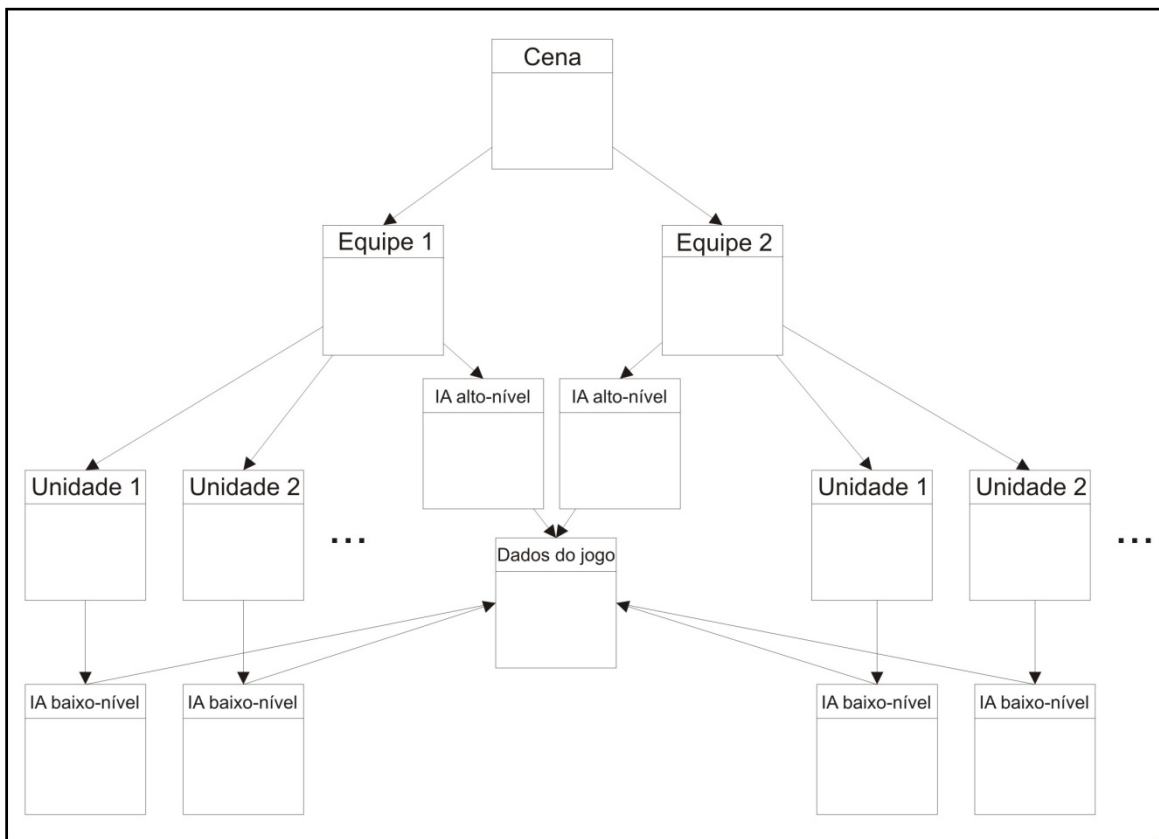


Figura 18: Integração IA x Modelo Gráfico.

A IA de alto-nível faz parte da composição da equipe, e possui referência para a estrutura de dados que possui os dados do jogo, explicada anteriormente no

capítulo 5.7. A IA de baixo-nível faz parte da composição das unidades e também possui referência para os dados do jogo.

A partir da estrutura de dados que possui os dados do jogo, a IA de alto-nível tem acesso à IA de baixo-nível para enviar as ações necessárias para a mesma.

6.2 IA de Baixo Nível

Representa a IA de cada unidade do jogo, sendo que é colocada na posição mais baixa da hierarquia do sistema.

Essa camada do sistema de IA possui várias MEFs, uma para cada ação a qual é designada a executar. Essas ações são:

- Conquistar mina de ouro: move até a região de conquista da mina de ouro e permanece em guarda.
- Atacar inimigo: persegue e ataca o inimigo selecionado, até que seja derrotado.
- Atacar castelo do inimigo: move até o castelo do inimigo e ataca-o, até que seja destruído.
- Defender castelo: move até o castelo de sua equipe e permanece em guarda.

A partir da execução de determinados comportamentos e do recebimento de determinados estímulos do ambiente, a IA de baixo-nível pode executar ações definidas por essas situações e não pela IA de alto-nível. De acordo com a modelagem proposta, a única ação dessa natureza é a *atacar inimigo*, pois é a única ação que pode depender exclusivamente de estímulos externos, como necessidade de defesa. Ao detectar esse comportamento, deve-se guardar a ação atual (imposta pela IA de alto-nível) para que após executado retorne a execução dessa ação anterior, não deixando de obedecer a seu papel na hierarquia do sistema de IA (subordinação à IA de alto-nível).

Para a execução de cada comportamento, a IA de baixo-nível possui ações como:

- Busca do melhor caminho entre dois pontos, utilizando a busca de grafos A*;
- Locomoção;

- Detecção de presença de inimigos em sua linha de visão;
- Perseguição do inimigo selecionado para ser atacado.

6.3 IA de Alto Nível

A IA de alto nível é responsável por controlar e executar os comportamentos do controle central (general) de cada equipe. É a partir das decisões tomadas por essa camada que a IA de baixo nível define seu estado. Para seu desenvolvimento é necessário primeiramente definir quais serão os comportamentos necessários para esse módulo.

Os comportamentos serão definidos baseados nas regras definidas do jogo-exemplo definidas no capítulo 4, e são:

- Evoluir exército: responsável por selecionar uma mina de ouro a ser conquistada e criar, a partir do ouro disponível, uma quantidade de guerreiros suficiente para conquistar a mina selecionada.
- Conquistar mina de ouro: responsável por locomover o exército agrupado para conquistar a mina de ouro selecionada;
- Atacar castelo do inimigo: agrupa soldados necessários para atacar o castelo do inimigo, mantendo a integridade do exército atacante, adicionando ou removendo guerreiros do mesmo.
- Defender castelo: responsável por defender o castelo caso esteja sob ataque, movendo e gerenciando a quantidade necessária de guerreiros para tal.

A partir dessa definição dos comportamentos, pode-se então construir a máquina de estados finitos, definindo-se as transições de cada estado para outro, ilustrada na Figura 19. Suas condições de transição são:

- Condição 1: porcentagem de guerreiros para ataque a mais quantidade de inimigos na mina a ser atacada $> 0,2$.
- Condição 2: ((quantidade de inimigos na mina = 0) e (quantidade de guerreiros na mina > 0)) ou (quantidade de guerreiros na mina = 0).
- Condição 3: porcentagem de guerreiros a mais que o inimigo $> 0,8$.
- Condição 4: porcentagem de guerreiros a menos que o inimigo $> 0,8$.
- Condição 5: quantidade de inimigos atacando castelo = 0.

- Condição 6: quantidade de inimigos atacando castelo > 0.

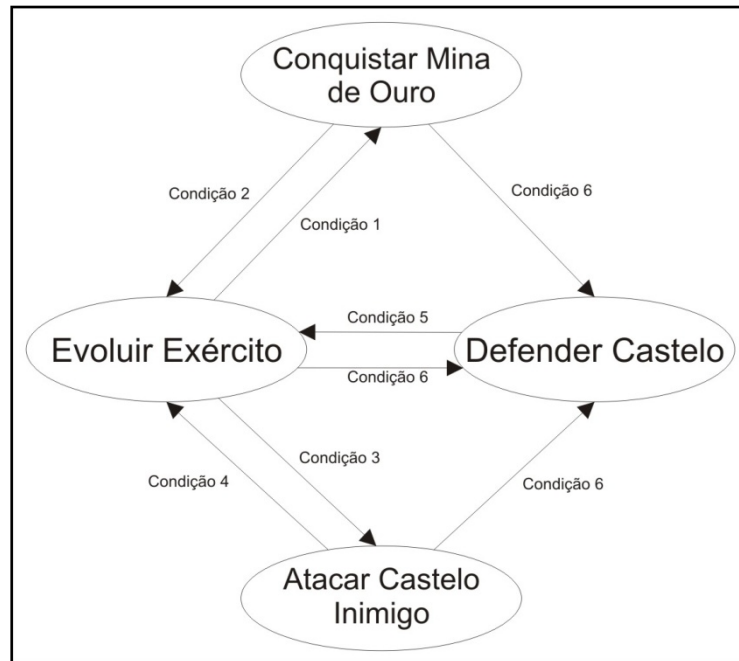


Figura 19: MEF da IA de alto-nível.

Nota-se que todas as condições de transição são discretas, sendo que a implementação é feita utilizando IA simbólica para o controle das transições.

6.4 IA de Alto Nível Baseada em Redes Neurais

Com o objetivo de suavizar as transições da IA de alto nível e facilitar a definição das mesmas, o sistema de IA discutido anteriormente foi desenvolvido também utilizando redes neurais.

A rede neural utilizada é do tipo recorrente, a qual tem a capacidade de simular uma máquina de estados fuzzy. A opção por escolher uma implementação por redes neurais e não por lógica fuzzy se dá pela maior facilidade de desenvolvimento da primeira, pois para a definição da rede é somente necessário definir as variáveis de entrada e saída e um conjunto de treinamento com exemplos de situações ideais, facilitando também a tarefa de manutenção do sistema visto que fica restrita à modificação desse conjunto; em um sistema fuzzy, também é necessária a definição de conjuntos e variáveis fuzzy.

Esse tipo de sistema é adequado para o objetivo em questão pois permite atribuir um conjunto de estados como estado atual, e não mais um estado somente

como na máquina de estados finitos. Isso resulta em uma necessidade de remodelar o sistema já implementado em MEF para adequá-lo à esse modelo.

Para a criação da rede neural utilizada foram desenvolvidas duas ferramentas: um gerador de treinamento, fazendo o treinamento *offline*, e um treinador. Os capítulos seguintes tratam do desenvolvimento dessas ferramentas e de todo o processo de modelagem do sistema neural, para então analisar os resultados obtidos.

6.4.1 Gerador de Treinamento

Para geração do treinamento da rede neural foi implementado um programa gerador de treinamento para a mesma. O usuário do programa deve configurar a quantidade de entradas e saídas da rede e gerar os exemplos que compõem o treinamento.

Após configurado e os exemplos gerado o usuário salva o treinamento em um arquivo *.trn*, podendo ser carregado no treinador da rede. A Figura 20 ilustra a interface do programa sendo utilizado.

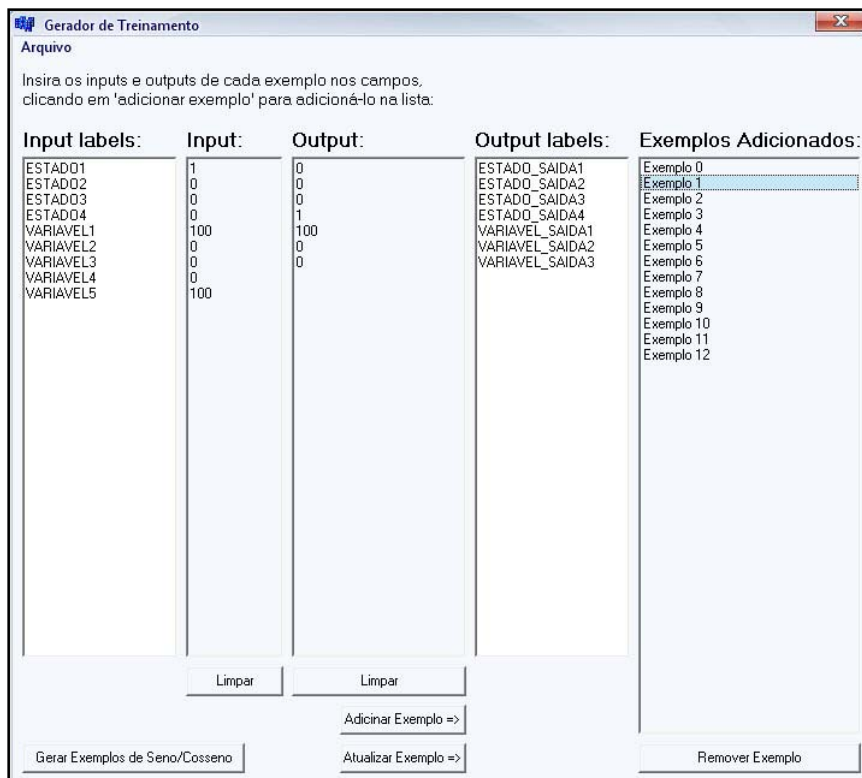


Figura 20: Interface do programa gerador de treinamento da rede neural.

Posteriormente foi implementada a funcionalidade de importação de exemplos a partir de arquivos .csv (arquivo delimitado por vírgulas), que pode ser gerado facilmente por outros programas que trabalham com planilhas (Microsoft Excel, por exemplo), para melhorar a usabilidade do programa.

6.4.2 Treinador da Rede Neural

Para o treinamento off-line da rede neural foi implementado um programa treinador de redes. Esse programa lê o treinamento a partir de um arquivo .trn, comentado no item 6.4.1, e possibilita ao usuário configurar as variáveis envolvidas nesse processo, como taxa de aprendizado, momento, quantidade de neurônios escondidos, erro máximo e quantidade máxima de épocas do treinamento. Também possibilita fazer testes na rede, inserindo valores de entrada e observando a saída.

Após feito o treinamento os pesos da rede são salvos em um arquivo que pode ser carregado pelo módulo de IA do jogo, podendo assim utilizá-la.

A Figura 21 ilustra a interface do programa treinador.

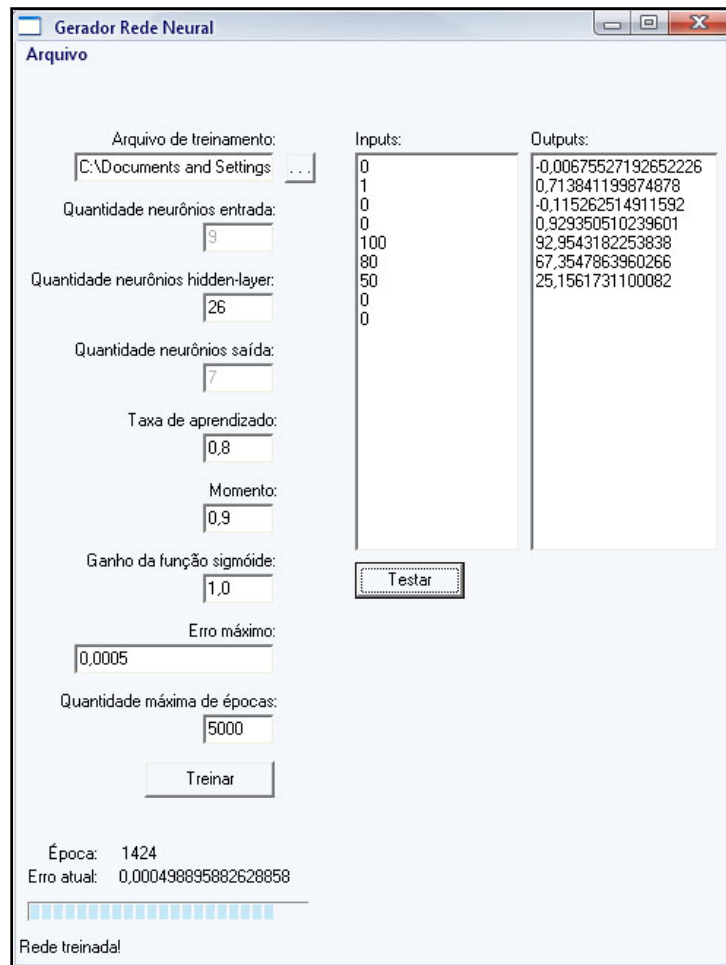


Figura 21: Interface do programa treinador da rede neural.

6.4.3 Modelagem do Sistema Neural

Para iniciar a modelagem do sistema neural deve-se conhecer exatamente que estados e variáveis necessárias para as transições farão parte do sistema e qual o comportamento desejado para o mesmo. Para isso será definida a máquina fuzzy que possui a mesma estrutura e semântica da MEF ilustrada na Figura 19, porém a descrição das condições de transição agora são diferentes, baseada em valores fuzzy:

- Condição 1: vantagem de ataque à mina é *grande*.
- Condição 2: mina de ouro *quase* conquistada.
- Condição 3: vantagem, em número de unidades, sobre o inimigo é *grande*.
- Condição 4: desvantagem sobre o inimigo é *grande*.

- Condição 5: o castelo está *pouco* vulnerável.
- Condição 6: o castelo está *muito* vulnerável.

Definidas as transições, percebem-se as variáveis que devem ser colocadas como entrada da rede, representando os estímulos captados do ambiente do jogo. Essas variáveis são:

1. Vantagem de ataque à mina de ouro;
2. Vantagem do domínio da mina de ouro;
3. Vantagem sobre o inimigo;
4. Vulnerabilidade do castelo;

A escolha dessas estradas não são feitas somente analisando as transições da máquina fuzzy; há a necessidade de considerar a normalização das entradas da rede, escolhendo variáveis que possam ser normalizadas. A normalização é necessária para a rede pois no seu conjunto de treinamento devem conter os valores limites de cada entrada, sendo que na utilização da mesma as entradas não devem estrapar esse valor. A questão da normalização é explicada com mais detalhes no capítulo 3.3.4. Como não há limite de personagens no jogo, a quantidade de personagens não é, por exemplo, uma entrada possível para a rede. Assim as variáveis definidas como entrada são então expressadas na forma de porcentagens, calculadas ao se captar os estímulos do jogo.

Além dessas entradas são colocadas mais quatro, uma para cada estado da máquina. Essas entradas são retro-alimentadas pela saída, criando assim a estrutura da rede recorrente para que possa interpolar resultados de acordo com seu estado interno atual.

As saídas da rede são quatro também, uma para cada estado, finalizando sua estrutura com 8 entradas e 4 saídas. Cada saída representa o nível de ativação de cada estado, ou seja, a porcentagem que cada estado que deve ser executado. Assim, por exemplo, quanto mais vulnerável o castelo, maior a ativação do estado de defesa; quanto maior a vantagem sobre o inimigo, maior a ativação do estado de ataque.

Cada uma das entradas e saídas representa então um neurônio de entrada ou saída da rede, onde a quantidade de neurônios da camada intermediária serão definidos na fase de teste da rede. Sua estrutura é ilustrada na Figura 22.

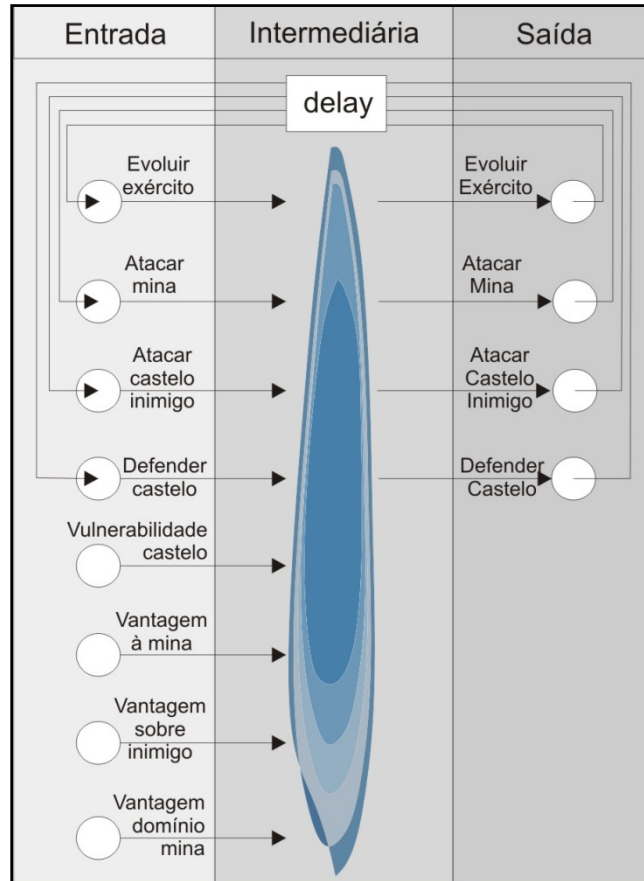


Figura 22: Estrutura da rede recorrente utilizada no jogo-exemplo.

Integrado esse modelo no jogo terá o seguinte funcionamento:

1. Captação dos estímulos do ambiente: os estímulos do ambiente são captados em forma de porcentagem para serem compatíveis com a normalização da rede.
 - a. Vantagem de ataque à mina de ouro: calcula-se a porcentagem da quantidade guerreiros prontos para atacar a mina de ouro sobre a quantidade de inimigos defendendo a mina.
 - b. Vantagem do domínio da mina de ouro: calcula-se a porcentagem da quantidade guerreiros na área de domínio da mina sobre a quantidade de inimigos nessa mesma área.
 - c. Vantagem sobre o inimigo: calcula-se a porcentagem da quantidade de guerreiros sobre a quantidade de inimigos.

- d. Vulnerabilidade do castelo: calcula-se a porcentagem da quantidade de inimigos atacando o castelo sobre a quantidade de guerreiros da equipe.
2. Execução da rede neural: os estímulos captados do ambiente normalizados e as saídas da rede (para retro-alimentação) são colocadas como entrada da rede, executado-a.
3. Extração dos níveis de ativação de cada estado: são analisadas as saídas da rede, desnormalizando sua saída e obtendo assim esses níveis de ativação.
4. Execução dos estados de acordo com seu nível de ativação: cada estado é executado utilizando o nível de ativação interpolado pela rede.

Nem todos os estados são compatíveis com a execução baseada em nível de ativação, pois por natureza não podem ser executados em partes. Isso acontece com o estado *“atacar_mina_de_ouro”*. Para acontecer esse estado deve-se ter uma quantidade suficiente de guerreiros para que a probabilidade de conquistar a mina seja grande. Logo, se fosse modelado para ser executado com nível de ativação, somente um ou alguns guerreiros iriam ser enviados à mina quando esse nível fosse pequeno, o que faria com que fossem exterminados caso houvesse uma grande defesa na mina.

Para adequar o estado *“atacar_mina_de_ouro”* à sua natureza, definindo seu modelo de execução como booleano (executa ou não executa), basta estabelecer uma competição entre seu nível de ativação e o estado precedente ao mesmo, o estado *“evoluir_exército”*. O estado que possuir o maior nível de ativação é então executado.

A execução dos outros estados, considerando o nível de ativação dos mesmos, é feita de modo particular, sendo que cada estado deve saber como utilizar esse valor para alcançar o objetivo esperado: sua execução parcial:

- a. Defender castelo: envia uma quantidade de guerreiros ao castelo para defendê-lo, de acordo com o nível de ativação, selecionando-os pela proximidade com o castelo.

- b. Atacar castelo inimigo: envia uma quantidade de guerreiros ao castelo inimigo para atacá-lo, de acordo com o nível de ativação, selecionando-os pela proximidade com o castelo inimigo.
- c. Evoluir exército: cria novos guerreiros e seleciona novas minas de ouro (de acordo com a facilidade de conquistá-la) para serem conquistadas. Os outros estados podem ocupar seus guerreiros para a execução apropriada dos mesmos.

Pode-se perceber que de acordo com essa modelagem não precisa mais ficar mantendo o grupo de guerreiros atacante ou defensor, pois isso já será feito a partir dos resultados interpolados pela rede neural: o tamanho desse grupo é proporcional ao nível de ativação de cada um desses estados.

Toda a lógica do aumento ou diminuição dos níveis de ativação dos estados de acordo com o estado interno da rede fica a cargo do conhecimento da rede neural adquirido através dos exemplos na etapa do treinamento. Essa etapa é apresentada no próximo tópico.

6.4.4 Treinamento do Sistema Neural Proposto

O treinamento da rede neural é constituído da fase de elaboração do conjunto de treinamento e treinamento da rede *offline*.

O conjunto de treinamento deve englobar situações extremas de cada cenário. Isso fará com que a rede “aprenda” essas situações extremas e interpole os resultados intermediários aos mesmos. Para isso, faz-se necessária a normalização das entradas e saídas, como já comentado no capítulo anterior, as quais serão dadas por porcentagens.

A Figura 23 mostra o conjunto de treinamento utilizado para o aprendizado da rede neural usada no jogo-exemplo.

ENTRADAS								SAÍDAS			
Evoluir exercito	Atacar mina	Atacar castelo inimigo	Defender castelo	Vulnerabilidade e castelo	Vantagem ataque mina	Vantagem sobre inimigo	Vantagem dominio mina	Evoluir exercito	Atacar mina	Atacar castelo inimigo	Defender castelo
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	1	0	0
1	0	0	0	0	0	1	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0
0	1	0	0	1	0	0	0	0	0	0	1
0	1	0	0	0	1	0	1	1	0	0	0
0	1	0	0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0	0	0	1	0
0	0	1	0	1	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	1	1	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0	0	0

Figura 23: Conjunto de treinamento da rede neural.

Cada cenário desse conjunto, representado por uma linha da tabela, é constituído por: estado de entrada, percepção do ambiente e estado de saída ideal. Logo cada cenário representa uma transição onde a condição da transição é dada pelas variáveis de percepção do ambiente, o estado a ser transicionado é o estado de entrada e o estado a ser alcançado é o estado de saída.

O treinamento da rede foi feito utilizando os seguintes parâmetros:

- Taxa de aprendizado: 0.8;
- Momento: 0.9;
- Número máximo de épocas: 5000
- Erro máximo: 0,0005;
- Número de neurônios na camada intermediária: 26;

Foi alcançado um erro de 0,00049 na época 143, finalizando então o treinamento.

6.4.5 Análise dos Resultados

Para a análise dos resultados foi implementado um *logger*. O *logger* é um módulo que fica a disposição do sistema para o armazenamento de informações que desejam ser reportadas, salvando essas informações em um arquivo para posterior análise. O arquivo salvo com o relatório (*log*) possui o formato *.csv* (separado por

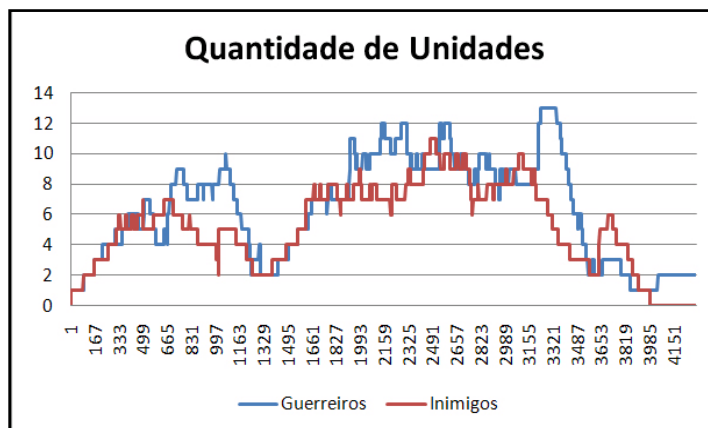


Figura 26: Quantidade de unidades da equipe 1 em relação aos ciclos do jogo.

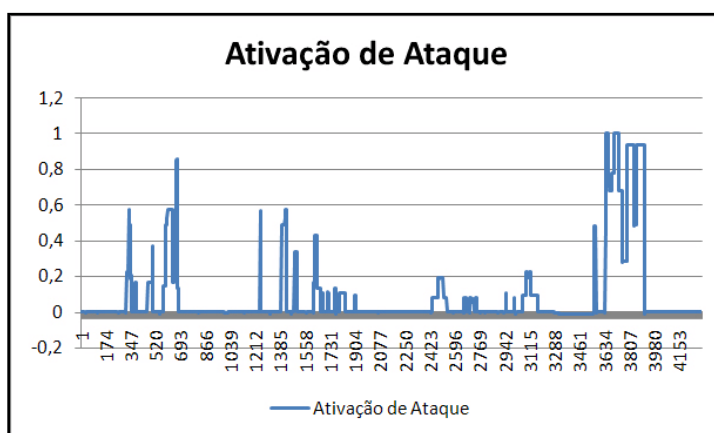


Figura 27: Ativação de ataque da equipe 2 em relação aos ciclos do jogo.

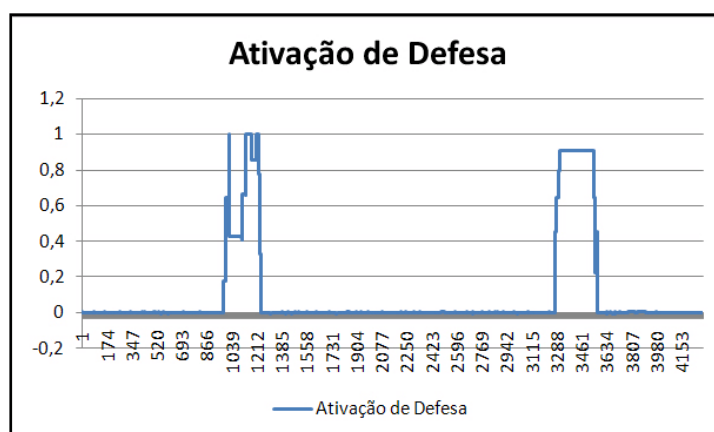


Figura 28: Ativação de defesa da equipe 2 em relação aos ciclos do jogo.

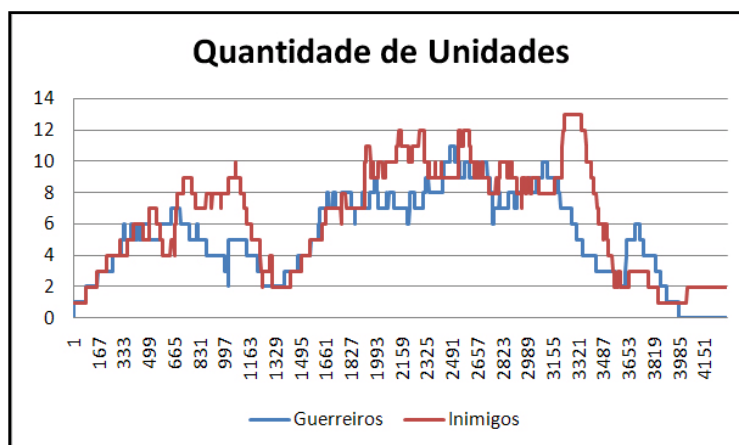


Figura 29: Quantidade de unidades da equipe 2 em relação aos ciclos do jogo.

A partir da análise dos gráficos pode-se perceber que a rede neural interpretou corretamente o antagonismo dos estados de defesa e ataque, sendo que enquanto a ativação de um estava alta a do outro estava baixa.

Também pode-se perceber que os picos da ativação do estado de defesa de uma equipe casam com os picos de ativação do estado de ataque da outra equipe e vice-versa. O número de unidades também aumenta ou diminui de acordo com as ações que cada equipe está tomando.

A análise mais importante a ser feita a partir desses gráficos, é a percepção de momentos em que esses casamentos não acontecem, ou seja, a equipe estava em mais de um estado ao mesmo tempo. Além disso pode-se perceber a transição suave dos estados, uma vez que os picos dos gráficos não são discretos (estando em seu valor máximo ou em seu valor mínimo) mas sim contínuos, como, por exemplo, a Figura 24 cujo gráfico permaneceu variando sua ativação entre valores diferentes de 0 e 1 entre os ciclos 623 e 1212.

A partir da execução de outra partida, agora entre uma equipe utilizando Redes Neurais e outra utilizando IA Simbólica, pode-se contemplar a vitória da equipe gerenciada pela primeira, analisando-se a quantidade final de guerreiros de cada equipe do gráfico da Figura 30, onde os níveis de ativação da equipe inimiga são ilustrados na Figura 31.

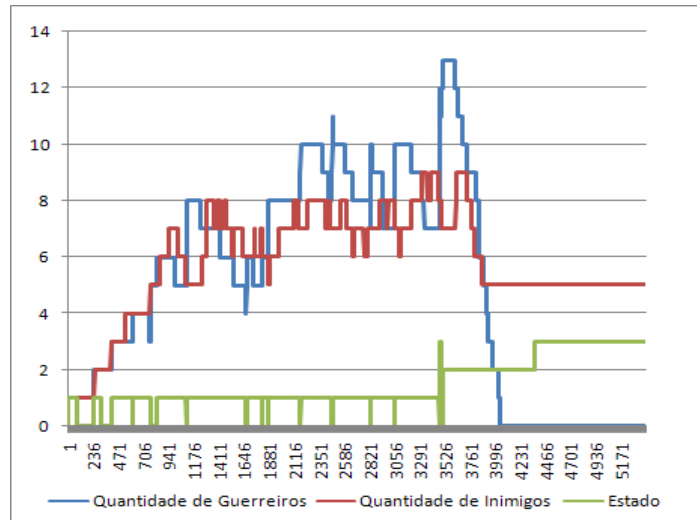


Figura 30: Estados da equipe gerenciada por IA Simbólica.

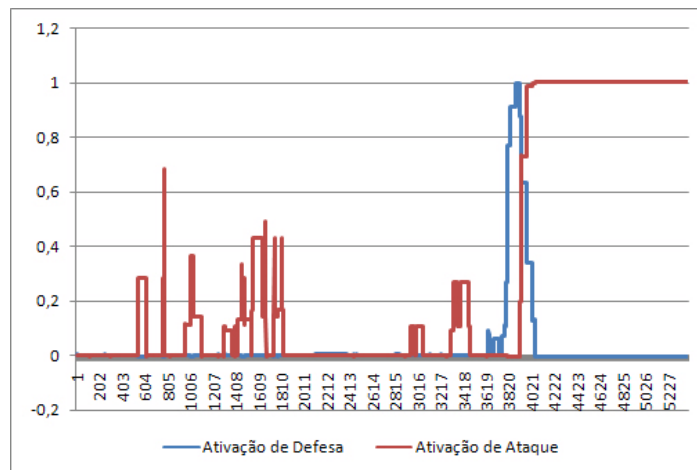


Figura 31: Níveis de ativação dos estados da equipe gerenciada pela IA baseada em Redes Neurais.

É importante notar a diferença entre esses gráficos pois pode-se perceber claramente a diferença da mudança de estados entre os modelos, que passa de discreta para contínua, e da forma de tratamento do estado corrente, que passa a ser um conjunto de estados com um nível de pertinência para cada um e não mais somente um estado.

Essa análise mostra que os objetivos foram alcançados para o jogo-exemplo, suavizando suas transições de estados e facilitando a definição de suas transições que passaram a serem interpoladas pela rede neural através do treinamento com um conjunto de exemplos.

Um ponto negativo analisado ao fazer a análise dos resultados foi o problema da oscilação de estados. Essas oscilações foram percebidas principalmente na transição do estado “evoluir_exército” para “atacar_castelo_inimigo” sendo que foi a transição mais executada na partida analisada. Isso ocorreu porque ao receber certas percepções a rede interpolou um valor de ativação para o estado de ataque, que então começou a ser executado; porém logo ao iniciar essa execução a percepção mudou e essa ativação caiu, voltando ao estado anterior, e assim sucessivamente.

7 – CONSIDERAÇÕES FINAIS

A busca por sistemas cada vez mais inteligentes no mundo dos jogos permanece como um desafio constante para desenvolvedores e criadores. Este trabalho procurou contribuir com esta busca incansável que pode ser estendida do mundo dos jogos à área da robótica ou de qualquer outro tipo de sistema autônomo.

Foi desenvolvido um jogo RTS para aplicação de um módulo de inteligência artificial baseado no estudo de redes neurais e máquinas de estados fuzzy. O objetivo foi amenizar a transição dos estados da máquina de estados responsável pelo controle da IA de alto nível, ou seja, a IA responsável pela coordenação das ações de cada equipe do jogo e a definição da mesma.

Através desses estudos e da experiência prática pode-se observar que foi substituído o modelo da máquina de estados finitos, onde somente um estado por vez é manipulado, por um modelo onde vários estados, ao mesmo tempo porém com grau de pertinência diferente, eram manipulados, alcançando o objetivo da transição suave dos estados. Além disso a definição das transições ficou a cargo da rede neural, facilitando a definição da relação entre os estados. Assim, tornou-se dispensável a definição das variáveis necessárias para o disparo de cada transição como no modelo antigo (utilizando máquina de estados finitos), bastando apenas definir um conjunto de exemplos de situações ideais para treinamento da rede. Assim a mesma se torna capaz de interpolar o grau de pertinência de cada estado para qualquer situação.

7.1 Trabalhos Futuros

Uma proposta de continuidade para essa pesquisa é o estudo em torno da problemática da oscilação de estados. Essa situação ocorre quando uma determinada condição para o aumento do grau de pertinência de um determinado estado é satisfeita e logo esse grau é diminuído novamente pela ocorrência de determinados estímulos do ambiente. Isso faz com que, por exemplo, uma unidade que esteja dominando uma mina seja englobada no estado de ataque (por ter seu grau de pertinência aumentado) e então começa sua locomoção para o castelo inimigo. Logo após o início dessa ação, essa mesma unidade é excluída desse estado (por ter seu grau de pertinência diminuído) voltando a sua posição anterior.

Outra possibilidade é inserção de mais uma camada na hierarquia do sistema de IA: uma camada de controle para grupos de unidades e a aplicação do paradigma utilizado no desenvolvimento do sistema em mais camadas dessa hierarquia. Isso causaria um comportamento mais surpreendente nas decisões do módulo de IA, aumentando bastante sua complexidade a fim de gerenciar esse grupo de unidades.

8 – REFERÊNCIAS

BUCKLAND, Mat. **AI Techniques for Game Programming**. 1. ed. Ohio: Premier Press, 2002.

BUCKLAND, Mat. **Programming Game AI by Example**. 1. ed. Texas: Wordware Publishing, 2005.

FAUSETT, Laurene V. **Fundamentals of Neural Networks: Architectures, Algorithms and Applications**. New Jersey.: Prentice Hall, 1994.

FREEMAN, James A.; SKAPURA, David M. **Neural Networks: Algorithms, Applications and Programming Techniques**. Addison-Wesley Publishing Company, 1991.

HAYKIN, Simon. **Neural Networks: A Comprehensive Foundation**. Prentice Hall, 1999.

LOPES, Gilliard L. dos; **Máquinas de Estados Hierárquicos em Jogos Eletrônicos**. Rio de Janeiro, 2004. 60p. Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

OMLIN, Christian W.; THORNBUR, Karvel K.; GILES, C. Lee. **Fuzzy Finite-State Automata Can Be Deterministically Encoded Into Recurrent Neural Networks**. IEEE Transactions on Fuzzy Systems, p. 76-89, 1998.

ROISENBERG, Mauro; PENATTI, Tiê L. T.; CARVALHO, Sandro R. S. de. **Arquiterura hierárquica de Comportamento de Robôs como Ferramenta de Projeto para Robôs Didaticos**. XXVI Congresso da SBC – III Encontro de Robótica Inteligente. Campo Grande, 2006.

RUSSEL, Stuart J.; NORVIG, Peter. **Artificial Intelligence: A Modern Approach**. 2. ed. New Jersey: Prentice Hall, 1995.

SCHWAB, Brian. **AI Game Engine Programming.** 1. ed. Massachusetts: Charles River Media, 2004.

SMAGT, Patrick van der; KROSE, Ben. **An Introduction to Neural Networks.** 8. ed. Amsterdam: The University of Amsterdam, 1996.

UNAL, Fatih A.; KHAN, Emdad. **A Fuzzy State Machine Implementation Based on a Neural Fuzzy System.** IEEE National Semiconductor, Embedded Systems Division, Santa Clara, CA., 1994.