TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL PARA JOGOS

COMPUTAÇÃO EVOLUTIVA

FACULDADE DE TECNOLOGIA DE AMERICANA

Jader Artur Costa

Thiago Guerino

Victoria Andressa Santos Macedo de Faria

Bruno Cardoso Ambrosio

Fernando Muniz

12/12/2018

Tradicionalmente, desenvolvedores de jogos digitais fazem uso sempre de um mesmo conjunto de técnicas simples na implementação das funcionalidades de IA(Inteligência Artificial) em jogos, diversas foram utilizadas em jogos eletrônicos, como Máquina de Estados Finita, Sistemas Baseados em Regras, Algoritmos de busca, etc. Atualmente, o mercado tem buscados técnicas mais clássicas para o desenvolvimento dos NPC's(non-player character, em português personagem não jogador), como Redes Neurais, Algoritmos Genéticos e Lógica Fuzzy. Além delas, um breve relato sobre Máquina de Estados Finita pode ser feito, pois foi uma técnica muito utilizada e que até hoje é recomendada para ser usada em qualquer jogo, pelo poder que possui e fácil programação da mesma.

Uma das técnicas mais impressionantes da IA é a RNA (Rede Neural Artificial), pois é uma técnica que busca simular o cérebro humano.

A parte do cérebro que as RNA's simulam são os neurônios, que na maioria das implementações são divididos computacionalmente em camadas. Esses neurônios são ligados entre si, assim como os neurônios naturais. Particularmente aplicada a jogos eletrônicos, a RNA, mesmo não sendo muito usada até hoje, é uma técnica muito interessante, pois como trata de aprendizado, os NPC's podem adquirir novos conhecimentos ao longo do jogo, e até mesmo reconhecer padrões. Adquirir conhecimento é fundamental, pois dessa maneira um NPC aprende alguma coisa que viveu no jogo, ou seja, alguma ação que o jogador efetuou. Quando esse jogador efetuar uma ação parecida, o NPC estará preparado, pois aprendeu com a experiência. Associado a isso, reconhecer padrões torna-se fundamental, como, por exemplo, padrões de movimento do jogador.

Diferente de outras técnicas, o NPC que usa RNA não é pré-programado com padrões de movimento. Ele espera a ação do jogador para somente assim decidir qual movimento fará.

Um exemplo clássico pode ser observado em jogos de corrida, o NPC ao colidir em uma curva aprende que deve fazer uma curva a esquerda ao invés da direita ou reto.

Para jogos, como o foco é a implementação de NPC's, que são inseridos em jogos eletrônicos, o Backpropagation é o algoritmo RNA recomendado, pois treina a rede exaustivamente até que ela assuma um estado perfeito, o que é como estar treinando um NPC de forma que ele busque o estado mais perfeito possível que possa chegar no jogo. Além do Backpropagation, outro algoritmo que pode ser recomendado para o uso em jogos eletrônicos é o Perceptron citados em breve.

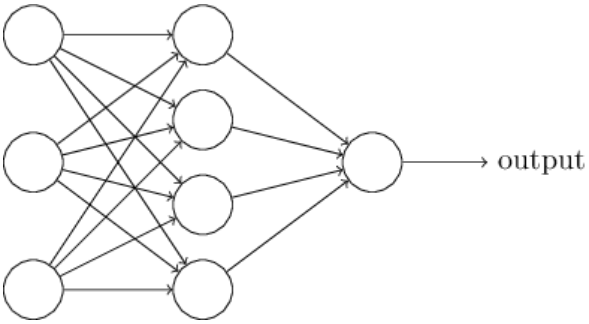
A evolução de redes neurais é uma forma fundamental de adaptação além do aprendizado, métodos baseados em computação evolutiva têm sido proposto como uma alternativa eficiente e robusta para os projetos de RNA's. Podendo ser usado para optimizar diversos aspectos como pesos, número de camadas escondidas, número de neurônios em cada camada escondida, conexões entre neurônios. Além disso são utilizados para diversas tarefas como: treinamento de pesos, design da arquitetura, adaptação da regra de aprendizagem, seleção de características (reconhecimento de imagens), extração de regras da rede, etc.

Uma característica das RNE (Redes Neurais Evolutivas) é a adaptabilidade à um ambiente dinâmico. Em um sentido mais amplo, RNE's podem ser consideradas uma generalização de sistemas adaptativos sem a intervenção humana.

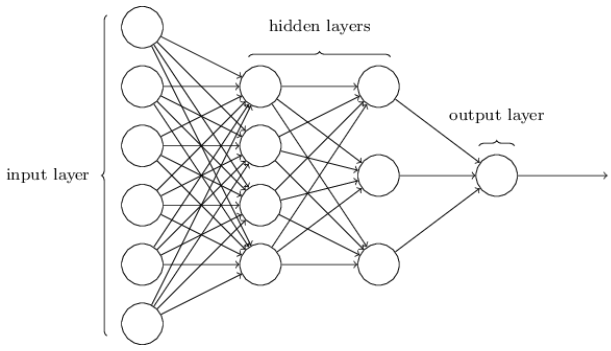
Métodos de redes neurais artificiais sem computação evolutiva podem requerer um enorme processamento de dados em projetos grandes o que causa a não viabilidade devido ao custo e tempo.

REDES NEURAIS ARTIFICIAIS

Existem diversas arquiteturas e topologias de RNA, assim como as suas formas de treinamento. Quando nos referimos à arquitetura de uma rede neural estamos nos referindo sobre a disposição dos neurônios, um em relação ao outro. Já a topologia da rede se refere às diferentes composições estruturais possíveis com diferentes quantidades de neurônios nas camadas de entrada, intermediária e de saída da rede, observe a imagem:



A camada mais à esquerda nesta rede é chamada de camada de entrada e os neurônios dentro da camada são chamados de neurônios de entrada. A camada mais à direita ou a saída contém os neurônios de saída ou, como neste caso, um único neurônio de saída. A camada do meio é chamada de camada oculta, já que os neurônios nessa camada não são entradas ou saídas. O termo “oculto” não significa nada mais do que “uma camada que não é entrada ou saída”. A rede acima tem apenas uma única camada oculta, mas algumas redes possuem múltiplas camadas ocultas. Por exemplo, a seguinte rede de quatro camadas tem duas camadas ocultas:



Tais redes de camadas múltiplas são chamados de Perceptrons Multicamadas ou MLPs (Multilayer Perceptrons), ou seja, uma rede neural formada por Perceptrons.

Em RNA onde a saída de uma camada é usada como entrada para a próxima camada são chamadas de redes neurais feedforward (alimentadas para frente). Isso significa que não há loops na redes, as informações sempre são alimentadas para a frente, nunca são enviadas de volta. Se tivéssemos loops, acabaríamos com situações em que a entrada para a função σ dependeria da saída.

No entanto, existem outros modelos de redes neurais artificiais em que os circuitos de feedback (Alimentação para trás) são possíveis. Esses modelos são chamados de redes neurais recorrentes. A ideia nestes modelos é ter neurônios que disparem por algum período de tempo limitado. Disparar pode estimular outros neurônios, que podem disparar um pouco mais tarde, também por uma duração limitada. Isso faz com que ainda mais neurônios disparem e, ao longo do tempo, conseguimos uma cascata de disparos de neurônios. Loops não causam problemas em tal modelo, uma vez que a saída de um neurônio afeta apenas sua entrada em algum momento posterior, não instantaneamente.

Geralmente, as arquiteturas de redes neurais podem ser colocadas em três categorias específicas, redes neurais feed-forward, redes recorrentese e redes conectadas simetricamente. Dentre estas categorias podemos citar 10 arquiteturas principais, Redes Multilayer Perceptron, Redes Neurais Convolucionais, Redes Neurais Recorrentes, Long Short-Term Memory (LSTM), Redes de Hopfield, Máquinas de Boltzmann, Deep Belief Network, Deep Auto-Encoders, Generative Adversarial Network e Deep Neural Network Capsules (este é um tipo completamente novo de rede neural, lançado no final de 2017).

A camada de entrada é responsável pelo recebimento dos dados/sinais/amostras a serem analisados, assim como a correspondente associação com os pesos de entrada. A camada intermediária ou escondida tem por finalidade extrair as informações associadas ao sistema inferido, sendo também responsável pela maior parte do processamento destes dados. Já a camada de saída agrega os dados das camadas anteriores e ativa uma resposta adequada.

Um exemplo simples de implementação feita em python de uma “Step Function” ou seja, a saida só pode ser 0 ou 1 (imagem abaixo), pode ser encontrada no link “https://techja.com.br/2018/12/10/redes-neurais-artificiais-step-function/” (by Jader).

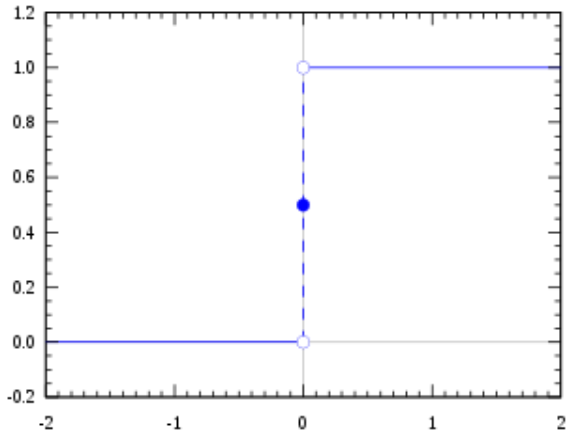


Figura 1: Gráfico de saida de uma Step Function

Entre as dez arquiteturas as quatro mais usadas em RNA são:

FeedForward de Camada Simples: Uma camada de entrada diretamente associada a um ou mais neurônios que vão gerar a resposta de saída. Observe que o fluxo de dados segue sempre em direção à camada de saída. São empregadas em problemas de classificação de padrões e filtragem. Tipos: Perceptron e Adaline.

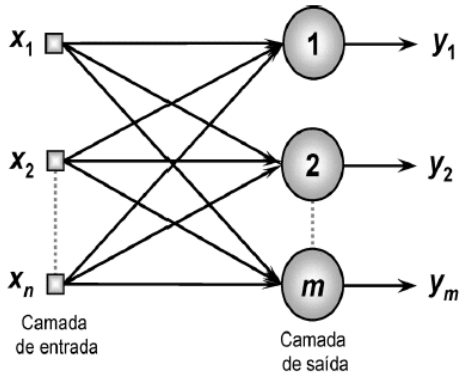


Figura 2: Arquitetura FeedForward de Camada Simples

FeedForward de Camadas Múltiplas: Constituída por uma ou mais camadas escondidas de neurônios. São empregadas em problemas de aproximação de funções, classificação de padrões, identificação de sistemas, otimização, robótica e controle de processos. Tipos: Perceptron Multicamadas, Redes de Base Radial.

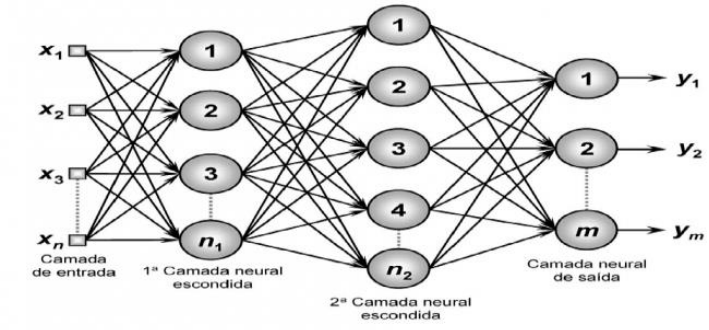


Figura 3: Arquitetura FeedForward de Camadas Múltiplas

Recorrente ou Realimentada: Suas saídas são realimentadas como sinais de entrada para outros neurônios, sendo assim empregadas para o processamento de sistemas variantes no tempo. São empregadas em previsões de séries temporais, otimização, identificação de sistemas e controle de processos. Tipos: Hopfield e Perceptron Multicamadas com Realimentação.

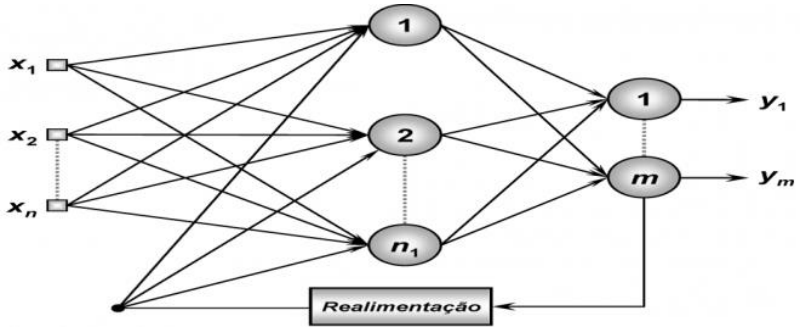


Figura 4: Arquitetura Recorrente ou Realimentada

Estrutura Reticulada: Consideram a disposição espacial dos neurônios com o propósito da extração de características do sistema, ou seja, sua localização espacial serve para ajuste de

seus pesos e limiares. São empregadas em problemas de agrupamento, reconhecimento de padrões, otimização de sistemas, etc. Tipo: Kohonen.

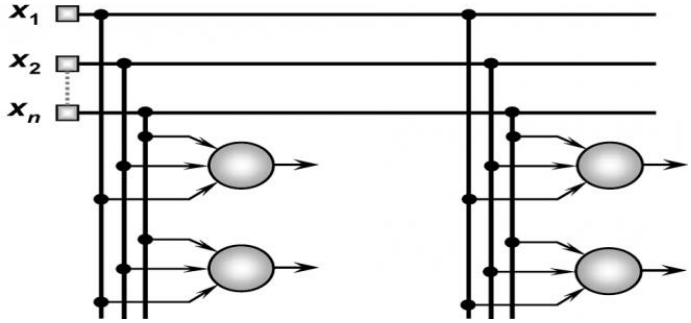


Figura 5: Arquitetura de Estrutura Reticulada

Treinamento da Redes Neurais Artificiais

Para que uma RNA atenda seus objetivos ela deverá ser treinada. O termo treinamento de uma rede neural consiste basicamente em fazê-la adotar valores de pesos e limiares baseados em suas amostras de entrada, de modo que qualquer outra amostra futura apresentada a ela seja corretamente classificada. Isto é, apresentaremos nossas amostras/dados na entrada da rede e direcionaremos uma resposta desejada fazendo com que a rede seja obrigada a modificar seus pesos e limiares. Assim qualquer outro valor apresentado a ela depois que a rede estiver corretamente treinada produzirá uma saída ou resultado igual ou próximo do desejado.

Supervisionado: Deve-se dispor das amostras e das respectivas saídas desejadas para que os pesos e limiares sejam ajustados continuamente pelo algoritmo de aprendizagem.

Não Supervisionado: A saída deve se auto organizar em relação às particularidades do conjunto de amostras e assim identificar subconjuntos similares. Seus pesos e limiares são ajustados pelo algoritmo de aprendizagem de modo a refletir estas particularidades.

Com Reforço: Similar ao treinamento supervisionado, contudo seu algoritmo visa ajustar os pesos e limiares baseando-se em informações procedentes da interação com o sistema mapeado, visando reforçar as respostas satisfatórias.

Lote de Padrões (Off-Line): Apresenta-se primeiramente todo o conjunto de amostras para então se ajustar os pesos e limiares da rede.

Padrão por Padrão (On-Line): Os pesos e limiares são ajustados após a apresentação de cada amostra permitindo que a mesma seja descartada posteriormente. Este tipo de treinamento é empregado quando o sistema a ser mapeado sofre variações ao longo do tempo. Observe que devido às características do sistema variante, este tipo de treinamento somente fornecerá respostas precisas após a apresentação de um número significativo de amostras.

Algoritmos Evolutivos

Algoritmos evolutivos referem-se a uma classe de métodos de busca populacionais estocásticos inspirados nas idéias e princípios da evolução natural, incluem: algoritmos genéticos, estratégias evolutivas, programação evolutiva, programação genética e sistemas classificadores.

Uma característica importante destes algoritmos é sua entratégia de busca por métodos populacionais. Indivíduos em uma população competem e trocam informações entre si a fim de realizar alguma tarefa.

Para os jogos eletrônicos, uma técnica muito importante e talvez a que venha a ser mais utilizada, principalmente em jogos de RPG (Role Playing Game), são os algoritmos genéticos, muito conhecidos também, ao menos o conceito, como computação evolucionária. Trata-se de uma técnica para simular o sistema genético, ou seja, da mesma maneira que um ser humano consegue evoluir, se adaptando aos diferentes meios que tem para sobreviver, o personagem no jogo irá evoluir e se adaptar ao ambiente do jogo e a sua interação com o jogador.

Analisando mais especificamente um jogo eletrônico, o jogador irá enfrentar NPC’s, que são normalmente no início, mais fracos, e então é mais fácil do jogador conseguir derrotá-los. Dessa forma, novos NPC’s vão surgindo assim que outros são derrotados, e devido aos genes conseguirem evoluir, os NPC's ficam mais fortes e mais difíceis para o jogador conseguir derrotá-los. Na teoria da evolução ocorre o mesmo, onde os herdeiros de determinados pais pegam a carga genética desses pais, e se adaptam das diferentes maneiras, assim conseguindo o desenvolvimento.

Algoritmos genéticos podem ser utilizados para a criação de populações inteiras em um jogo, ou até mesmo a mutação e evolução de personagens. Jogos de RPG usam, portanto, esse tipo de algoritmo pela própria definição do mesmo: um jogo onde o ambiente e os personagens nele contidos se desenvolvem ao longo do jogo. É notável como algoritmo genético é o mais recomendado para RPG, pois faz, por definição, o que RPG’s precisam.

O processo de evolução de um AG (Algoritmo Genético) ocorre quando há o cruzamento de indivíduos da mesma espécie, ou seja, são criados personagens com cromossomos pré-definidos e esses personagens são colocados no ambiente de maneira com que se cruzem com outros. Os cromossomos podem ser representados computacionalmente como um vetor de bits. Quando há um cruzamento, um novo personagem é criado, e a lei do mais forte prevalece. Personagens melhores sobrevivem, enquanto os mais fracos são retirados do ambiente. A cada iteração, esses personagens podem ou não receber mutação, que é a outra maneira dos cromossomos mudarem. Em jogos eletrônicos, os personagens mais fracos são logo eliminados, pois o próprio jogador irá conseguir derrotá-lo.

Um AG não deve ser utilizado caso haja um método específico de se resolver um problema, mas no caso de jogos eletrônicos, sempre existirá um jogador, e os seres humanos sempre pensam de maneira diferente e jogam de maneira diferente. Dessa forma, é altamente recomendado o seu uso, já que o objetivo é criar cada NPC de forma que pareça muito com um ser humano, não apenas fisicamente (parte gráfica), mas também quanto ao comportamento, maneira como age no ambiente, ou seja, como se movimenta e como interage com o personagem que o jogador controla.

LÓGICA FUZZY EM JOGOS

A Lógica Fuzzy é outra técnica de IA usada em diversos ramos da computação, e que pode ser perfeitamente utilizada em jogos eletrônicos. Trata-se de uma técnica que consegue tomar decisões parecidas com a de seres humanos. Um ser humano toma uma decisão muitas vezes baseado no que ele acha, e não em fatos comprovados por experimento. Não é sempre que uma decisão é tomada estando totalmente experimentada, testada e com embasamento teórico e prático suficientes. Na verdade, na maioria das vezes as pessoas respondem aquilo que elas acham, e o achar não é relacionado somente a fatos, mas opinião dentre outras coisas influencia.

Durante bastante tempo, técnicas para implementar IA em NPC foram baseadas em FSM (Finite-State Machine) e Sistemas Baseados em regras. Esse tipo de técnica necessita de respostas exatas, ou seja, um NPC irá decidir o que fazer pensando no que pode fazer e resolver sim ou não. Por exemplo, em um jogo de confronto com armas, se o NPC quiser saber se vai atirar ou não no personagem do jogador naquele momento ele irá analisar e decidirá como na lógica clássica, verdadeiro ou falso, ou seja, sim ou não, irá atirar naquele momento ou não. Sistemas Baseados em Regras será dessa forma, pois todas as regras já estão pré-definidas. FSM também, pois um NPC decidirá se irá ou não mudar de estado, decisão sim ou não. Já na Lógica Fuzzy acontece diferente, uma decisão de um NPC não será necessariamente sim ou não, ele irá analisar outras coisas, o que ele acha que deve ou não fazer.

O fato de uma NPC achar algo já o torna inteligente, pois achar já é um ato de pensar, e se ele consegue pensar, ele possui certo nível de inteligência. O que irá influenciar a decisão de um NPC baseado no que ele acha são fatores do ambiente que ele está envolvido, e até mesmo no que o jogador pensa ou no que os outros NPC's estão fazendo naquele momento. É interessante ressaltar que um NPC trabalha em conjunto com outros NPC's, e juntos formam o que chamamos de estratégia.

Imagine um jogo de corrida, onde a velocidade de um carro guiado por um NPC deve ser sempre levada em consideração, para ele fazer curvas, por exemplo, ele deve reduzir se sua velocidade for alta. Diferente da lógica booleana, a lógica Fuzzy irá tratar a velocidade de modo diferente. Uma pessoa pode considerar 100 km/h rápido, já outra pode considerar lento, mas é quase certo que elas consideram 10 km/h lento, e 140 km/h veloz.

Para o uso específico em jogos eletrônicos, o mais interessante é a chamada FFSM (Fuzzy Finite-State Machine), que é uma evolução da FSM (Finite-State Machine). Essa última técnica trata-se de uma implementação em que o NPC irá possuir estados diferentes. Por exemplo, em um jogo de batalhas, um NPC pode estar no estado de perseguir um personagem do jogador ou no estado de correr do jogador, dependendo da situação. Esse estado pode portanto variar. A FFSM é uma técnica que desenvolve a Máquina de Estados Finita, acrescentando a mesma a lógica Fuzzy. A FFSM é interessante para ser usada, pois muitos jogos utilizam as FSM.

MÁQUINA DE ESTADOS FINITA EM JOGOS

Uma técnica que muitas vezes não é colocada como IA e outras vezes sim, é a Máquina de Estados Finita, ou FSM. As FSM's foram utilizadas em muitos jogos na história, e diferentes de técnicas como RNA, AG e Fuzzy, são bem simples de serem implementadas.

Por definição, essa técnica trata de um algoritmo onde o estado do NPC pode alterar. O estado é nada mais do que o que ele sente no momento. Por exemplo, em determinado momento do jogo o NPC está em um estado de aceleração, em outro em estado de desaceleração e assim por diante. Não existe muita dificuldade em programar um algoritmo de FSM. Apenas devem

existir funções que validam o estado do personagem, ou sejam, mudam seu estado de acordo com o que está acontecendo. Esse tipo de função trabalha através de flags, que podem ser variáveis que guardam o estado do NPC. Dependendo do parâmetro que essa função recebe e da flag no momento, essa flag poderá ser alterada.

CONCLUSÃO

A utilização de RNA, AG e Lógica Fuzzy pode (e é recomendado) serem utilizadas em conjunto, para que possa existir um jogo eletrônico mais aprimorado. Não foi discutido aqui o poder computacional (em termos de hardware) necessários para isso, dependendo do tamanho do jogo.

A melhor forma de garantir aprendizado é com uma RNA, e por isso é aconselhável utilizá-la para que os NPC's do jogo tenham um cérebro e possam agir de forma inteligente. Junto a esse cérebro, é interessante colocar algoritmos de lógica Fuzzy, pois são fundamentais na hora de tomada de decisão, e foram criados justamente para tomar decisões, para que o NPC possa pensar. Para pensar nada mais aceitável que ele tenha um cérebro e seja capaz de aprender e de interpretar certos fatos com base no que ele acha que acontecerá ao ter determinada escolha.

O AG é fundamental para a estratégia, e praticamente todo jogo possui uma estratégia, tanto de combate, corrida, esporte, etc. Essa estratégia é garantida com o trabalho conjunto dos NPC's. Como vários NPC's trabalham em equipe consideramos que são uma população, e AG é realmente feito para o desenvolvimento e evolução de populações. Combinar AG com RNA e lógica Fuzzy, uma forma de fazer com que os indivíduos sejam inteligentes do ponto de vista cerebral, ou seja, pensar e guardar informações de experiências que viveram, e conseguir interagir da melhor forma possível com o ambiente à sua volta. A FSM pode entrar em qualquer momento do jogo, pois irá apenas praticamente garantir o estado do NPC naquele momento.

REFERÊNCIAS BIBLIOGRÁFICAS:

BORGES, Deise Miranda, BARREIRA, Rafael Gonçalves, SOUZA, Jackson Gomes de Souza. Comportamento de personagens em jogos de computador. Palmas: Centro Universitário Luterano de Palmas, 2009.

GALDINO, Carlos Henrique Silva. Inteligência artificial aplicada no desenvolvimento de jogos de computador. Itajubá: Universidade Federal de Itajubá, Instituto de Ciências Exatas, 2007.

KISHIMOTO, André. Inteligência Artificial em Jogos Eletrônicos. Copyright © 2004, André Kishimoto, 2004.

LUZ, Mairlo Hideyoshi Guibo Carneiro da. Desenvolvimento de Jogos de Computador. Itajubá – MG: Universidade Federal de Itajubá – Unifei, Departamento de Matemática e Computação – DMC, 2004.

MILLINGTON, Ian, FUNGE, John. Artificial Intelligence for Games. 2ª Edição. Editora Elsevier, 2009.

RUSSEL, Stuart, NORVIG, Peter. Inteligência Artificial: Uma abordagem moderna. Editora Campus, 2003.

SANTANA, Roberto Tengan. I.A. Em Jogos: a busca competitiva entre o homem e a máquina. Praia Grande: Faculdade de Tecnologia de Praia Grande, 2006.

SCHWAB, Brian. AI Game Engine Programming. 1.ed. Hingham, Massachussetts: Charles River Media, Inc. 2004.

TATAI, Victor Kazuo. Técnicas de Sistemas Inteligentes Aplicadas ao Desenvolvimento de Jogos de Computador . Campinas: Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação , 2003.

TRÉ, Daniel. Redes Neurais Artificiais Aplicadas ao Jogo da Velha 3D em Pinos.

Silva, Ivan N.; Spatti, Danilo H.; Flauzino, Rogério A; Redes Neurais Artificiais para engenharia e ciências aplicadas, Artliber, 2010.

Walter, Igor F. Algoritmos Evolutivos e Redes Neurais Artificiais Revisão Bibliográfica, Universidade Estadual de Campinas, 2002.

A Arquitetura das redes neurais. Deep Learning Book, São Paulo, 20 de jun. de 2018. Disponível em: <http://deeplearningbook.com.br/a-arquitetura-das-redes-neurais/>. Acesso em: 12 de dez. de 2018.