

Propostas de Técnicas para Otimização em Algoritmos de Inteligência Artificial nos Jogos

Fernando Mendel¹, Ricardo Dias¹

¹Bacharelado em Ciência da Computação Universidade do Vale do Rio dos Sinos
(Unisinos)

CEP 91.501-970 São Leopoldo RS Brazil

fernando18@gmail.com, rickd113@hotmail.com

Resumo. Este artigo tem como objetivo analisar dois jogos antigos (*Prince of Persia* e *Dangerous Dave*), desenvolvidos entre 1980-1990 e dois jogos atuais (*Tomb Raider* e *Deus Ex 2: Invisible War*), criados a partir de 2004. Mostrando as similaridades e descrevendo os pontos fortes de cada jogo, objetivando perceber as tecnologias gráficas e de Inteligência Artificial que se desenvolveram de modo significativa nesse período e os problemas que não tiveram um desenvolvimento tão acentuado, bem como tentar identificar técnicas para o melhorar a velocidade do processamento de algoritmos de IA.

1. Introdução

Grande parte da evolução em Computação Gráfica se deve aos jogos, por causa deles os hardwares gráficos são relativamente baratos e estão em crescente melhora, devido a alta demanda de compra por usuários comuns. Esse mercado cresce de maneira espantosa, enquanto em 2000 o faturamento da indústria mundial de jogos computadorizados não ultrapassou os US\$ 10 bilhões, o ano de 2003 terminou com um faturamento equivalente a US\$ 28,8 bilhões, o que equivale a quase três vezes a receita gerada pela indústria cinematográfica[14].

Abordamos também a inteligência artificial, que é empregada de maneira simplificada nos jogos para torná-los mais interessantes, proporcionando uma maior imersão do usuário.[15]

Foram escolhidos quatro jogos de estilo similar para facilitar a comparação. Sendo os dois antigos em 2D e os novos em 3D.

2. Jogos Antigos

2.1. *Dangerous Dave*

Este jogo foi desenvolvido em 1990 por John Romero. *Dave* é um caçador de tesouros, seu objetivo é conseguir pegar uma taça de ouro para poder passar pela porta para o próximo nível. O jogo consiste em dez níveis normais e quatro áreas de bônus (*Warp Zones*). [1]

A forma de visualização é lateral, e ele pode ser controlado através do teclado ou do mouse. Deve-se ressaltar que esse jogo possui um menu de opções muito bom, o que

não era comum para os jogos dessa época. Este menu pode ser acessado a qualquer momento no jogo através da tecla F2.

Nas configurações padrão, *Dave* é movimentado pelas setas direcionais, a tecla Alt ativa o *JetPack* (permite voar por um curto período de tempo) e Ctrl atira. Sendo que a Arma e o *JetPack* duram somente na fase onde *Dave* está, devendo consegui-los novamente no próximo nível. Quando *Dave* atira ele dispara uma bala cinza em linha reta (veja fig. 1), e se atingir algum inimigo ocorre uma explosão. As únicas formas para o inimigo explodir são: ou *Dave* o acerta com um tiro, ou ele encosta no inimigo (Os dois explodem). Veja na fig. 2 um exemplo de explosão.

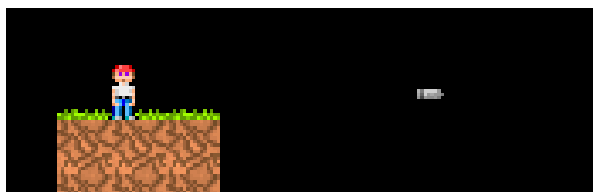


Fig. 1 - Disparo

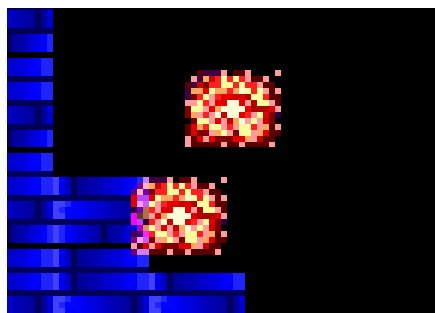


Fig. 2 - Explosão

O jogo é em duas dimensões e utiliza uma paleta de 256 cores no modo VGA (nas opções do jogo podem ser selecionados outros modos como CGA, que possuem menos cores).

O personagem pisca ao se movimentar, pular, ou ao usar o *JetPack* para voar, principalmente quando se aproxima da parte superior da tela. Isso parece ser devido ao baixo número de quadros por segundo (FPS). O som do jogo é feito em beeps do gabinete e é muito repetitivo. O jogo deveria ter uma opção para salvar o progresso do jogador.

2.2. *Prince of Persia*

Prince of Persia foi desenvolvido em 1990 por Jordan Mechner e revolucionou o mundo dos jogos. Até então, nenhum jogo havia criado lutas com espadas e feito movimentos de um personagem tão reais. Segundo Wikipédia [2], Mechner usou um processo no qual ele estudou por muitas horas filmes de seu irmão mais novo, David, correndo e pulando com roupas brancas, para assegurar que todos os movimentos parecessem corretos. O som do jogo é muito bom para a época, e os sons não se repetem tanto como no jogo anterior (*Dangerous Dave*).

O jogo utiliza uma paleta de 256 cores, sendo muito bem utilizada, pois os inimigos e os ambientes têm um bom contraste de cores. Assim como no jogo anterior, a forma de visualização é lateral, a diferença é que o personagem não pisca

constantemente como no outro jogo. O movimento se faz por meio das teclas direcionais e a tecla Shift ativa a espada (depois que o príncipe a consegue).

O objetivo do jogo é salvar a princesa, o príncipe tem uma hora até que ela se case com o vizir. Cada nível tem uma porta para o próximo nível, que precisa ser aberta por algum mecanismo para ele poder subir as escadas para o próximo nível.

Um fato notável é a dificuldade do jogo, as fases são complexas com vários quebra cabeças. E a inteligência artificial dos inimigos aumenta bastante no decorrer das fases. No entanto, essa inteligência artificial é bem previsível, percebe-se que a abordagem de IA utilizada foi a mesma que Cunha[8] descreve no seu estudo de caso: A IA não apresenta mecanismos mais adequados para que os adversários não se revelem previsíveis após um certo tempo de jogo. Dessa forma, o desafio proposto ao jogador fica limitado à descoberta do padrão de comportamento do inimigo.

A animação do personagem principal é boa, os movimentos tentam imitar o ser humano. Um exemplo disso é que quando o personagem vira, ele mexe os braços, veja na fig. 3 o príncipe mexendo os braços ao se virar. Outro exemplo que demonstra bem isso é a figura 4, que mostra o príncipe pulando.



Fig. 3 Movimentos dos braços

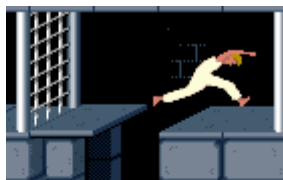


Fig. 4 - Pulando

3. Jogos Atuais

3.1 *Tomb Raider Anniversary*

Lançado oficialmente em 1 de julho de 2007 nos EUA, para *Playstation 2*, *Xbox*, *PC*. *Tomb Raider Anniversary* é um *remake* do primeiro jogo da série da musa Lara Croft.

A *engine* do *game* baseia-se na do título anterior, o qual foi totalmente reestruturado com amplas paisagens, detalhismo impecável, e segundo Uol Jogos [3] há efeitos de luz nos quais é possível até enxergar feixes luminosos que entram pelas frestas, tornando evidentes partículas suspensas no ar, como folhas.

A visão é em terceira pessoa, e o controle tem uma resposta muito rápida. Lara pode fazer saltos para todas as direções, correr, atirar, rastejar, nadar. O interessante é que nesse episódio, na versão para *PC*, a mira pode ser controlada pelo mouse, sendo possível atirar com precisão em objetos que não sejam necessariamente alvos. Outra característica interessante foi a implantação do *bullet time*, ou seja, sempre que um

inimigo vier contra Lara é possível desviar e atirar em câmera lenta e se acertar o ponto crítico, uma grande porcentagem de energia é tirada dele.

É muito comum Lara ou o inimigo ficarem presos em lugares que delimitam os cenários. Alguns objetos não possuem sombras e outros a sombra da Lara atravessa. Em alguns cenários onde há itens, como munições, que apesar de estarem na tela são impossíveis de pegar, veja na Fig. 5.

Sobreposição de imagens é muito comum, é só mirar a arma perto de alguma parede que e as mãos ultrapassam ela. E quando Lara usa uma corda, às vezes, o corpo dela entra no chão como mostra a Fig 6.



Fig. 5 O item está na tela, mais não é possível pegar.



Fig. 6 Lara está até os joelhos abaixo do chão

3.2 *Deus Ex: Invisible War*

Lançado no começo do 2004, *Deus Ex: Invisible War* é um RPG em primeira pessoa, onde o personagem principal é um agente cibernético que pode ser aperfeiçoado em várias habilidades através de chips, conseguidos no decorrer do jogo, através da experiência. Segundo Terra Games [4] *Deus Ex - Invisible War* é um grande jogo com grandes defeitos.

O game teve grande impacto nos jogadores pelo fato de oferecer dezenas de maneiras para concluir as missões e vários finais diferentes. Existem grupos com interesses diferentes, sendo possível escolher ser o mocinho ou o bandido. Um ponto interessante é que o game não economiza em catástrofes, um bom exemplo é logo no começo, onde *Chicago* é destruída em segundos.

Os gráficos são pouco detalhados e existem algumas travas de *frames* que as vezes incomodam, isso se dá pelo fato do game usar um baixo número de frames por segundo. É possível ter uma interação com praticamente todos os objetos dos cenários, porém, alguns respondem de maneira estranha, por exemplo, quando o personagem encosta em alguns objetos no chão os mesmos são jogados a uma distância muito grande.

Outro grande problema é a inteligência artificial do game, é muito fácil matar os inimigos. Se, por exemplo, Alex matar algum guarda a distância com um rifle *sniper*, os outros guardas nem irão se preocupar com o corpo no chão.

4. Comparações

A primeira diferença encontrada entre os jogos novos e antigos é a diferença no tempo de resposta dos controles. Os jogos antigos demoram muito mais tempo para responder. No jogo *Prince of Persia*, por exemplo, é preciso ficar quase um segundo com a tecla pressionada pra cima enquanto o personagem corre para ele pular.

Outra coisa óbvia é a diferença nos sons e gráficos entre os jogos antigos e novos. Por mais criatividade e talento que pudessem ter, pessoas como Jordan Mechner (criador do *Prince of Persia*), não conseguiriam e nem poderiam superar os sons e gráficos dos jogos atuais, depois da grande evolução em recursos multimídia existentes nos computadores atuais. Um bom exemplo de evolução multimídia é na trilha sonora de Tomb Raider anniversary não foi usado uma orquestra como parece, e sim recursos tecnológicos [9].

Segundo Foley[5], os gráficos eram produzidos através de bibliotecas bem mais simples e com um nível de abstração bem menor, como a biblioteca SRGP (*Simple Raster Graphics Package*), PHIGS (*Programmer s Hierarchical Interactive Graphics System*), GKS (*Graphical Kernel System*), ou através de programação em baixo nível diretamente no *framebuffer*. E estas bibliotecas prestavam serviço somente na parte gráfica. Hoje, temos bibliotecas como a SDL (*Simple DirectMedia Layer*), segundo SDL[6] é uma biblioteca que provê acesso de baixo nível ao áudio, teclado, mouse, joystick. E conforme Wikipédia SDL [7] oferece suporte a *threads*, rede, tempo e processamento de objetos compartilhados. O que permite criar aplicativos multimídia portáteis e multiplataforma.

Quanto a realidade virtual a diferença é muito grande também, enquanto nos jogos atuais vemos inimigos andando pelos ambientes, com uma certa autonomia, procurando o personagem principal em todos lugares, nos antigos vemos os inimigos fazendo movimentos em ciclos sendo possível prever onde estarão e o que farão. E segundo Andrade[10], isso se deve à evolução da capacidade do hardware, que liberou poder de processamento dos módulos gráficos para os módulos de inteligência, com isso, foi possível inovar nas técnicas de programação usadas para determinar o comportamento dos agentes e do ambiente, evoluindo dos scripts de programação para técnicas como redes neurais, algoritmos genéticos, lógica nebulosa, raciocínio baseado em casos.

Um fato que chama a atenção no *Dangerous Dave* e em alguns jogos deste gênero, é que geralmente a IA é muito simples, sendo que os inimigos fazem sempre o mesmo percurso. Essa é uma questão que poderia ser aprimorada.

5. Motivação

Cada vez mais atenção foi dada à Inteligência Artificial nos jogos, o motivo disto pode ser visto pela pesquisa realizada[14] em jovens com faixas de idade entre 10 e 17 anos (grande parte do público alvo dos jogos) foi feita a seguinte pergunta:

“Porquê você gosta de jogos para computadores?”

A maioria das pessoas respondeu que sentem atração devido ao desafio imposto pelo mesmo. Uma parcela considerável também respondeu que gosta dos jogos devido à história e à qualidade gráfica do mesmo.

Esse desafio, citado pela pesquisa, em grande parte é criado pela inteligência artificial dos inimigos. Por isso, no final dos anos noventa houve uma mudança na visão dos projetistas de jogos, que antes não demonstravam grande interesse em agregar inteligência artificial em seus produtos[15]. Mas depois dessa mudança de visão nos projetos de jogos houve um aumento substancial nos desenvolvedores dedicados exclusivamente à programação da IA podemos confirmar isso na Fig 7[15].

	'97 GDC	'98 GDC	'99 GDC
Developers dedicated to AI	24%	46%	60%
% of the overall game CPU reserved for AI processing	5%	10%	10%

Fig 7 - Recursos voltados à IA

Problema: *Como melhorar a inteligência artificial dos jogos sem prejudicar no desempenho?*

Enquanto nos jogos o que importa é o comportamento dos personagens, independente de sua maneira de pensar, podendo estes serem encarados como caixas pretas, na vertente mais acadêmica temos que o raciocínio é o foco das atenções e pesquisas. Por isso, muitas das técnicas da inteligência artificial acadêmica são empregadas de maneira simplificada nos jogos.[15]

O jogo não pode utilizar as mesmas técnicas de IA para todos os computadores, visto que nem todos são iguais e nem tem o mesmo poder de processamento, lembrando que um jogo é uma aplicação de tempo real. Ele deve identificar como está a velocidade do processamento e colocar algoritmos de IA apropriados, de acordo com o desempenho da máquina.

Pontos Chave:

1. Há algum hardware disponível para acelerar o processamento da IA?

Um dos hardwares que talvez tenha tido uma maior evolução desde o começo da informática é a placa de vídeo. A velocidade do processamento gráfico, em placas

gráficas com processador dedicado aumentou espantosamente. Veja esta tabela retirada do Clube do Hardware [12].

Chip Gráfico	Clock	Clock da Memória	Memória	Taxa de Transf. Memória	Pixels por Clock	DirectX
GeForce 4 MX 440 AGP 8x	275 MHz	512 MHz	128 bits	8,1 GB/s	2	7
GeForce 7300 GT (TC)	350 MHz	667 MHz	128 bits	10,6 GB/s	8	9.0c
GeForce 8400 GS	450 MHz / 900 MHz	800 MHz	64 bits	6,4 GB/s	16	10
GeForce 8600 GT	540 MHz / 1,18 GHz	1,4 GHz	128 bits	22,4 GB/s	32	10
GeForce 8800 GTS	500 MHz / 1,2 GHz	1,6 GHz	320 bits	64 GB/s	96	10
GeForce 8800 GTX	575 MHz / 1,35 GHz	1,8 GHz	384 bits	86,4 GB/s	128	10
GeForce 8800 Ultra	612 MHz / 1,5 GHz	2,16 GHz	384 bits	103,6 GB/s	128	10

Tabela 1 – Evolução das GPUs nos últimos anos

A GeForce 4 MX440 foi lançada em 2002 [13] e sua taxa de transferência é de 8,1 GB/s e processa 2 pixels por clock, enquanto o modelo mais atual até o momento, a GeForce 8800 Ultra, transfere 103,6 GB/s e processa 128 pixels por clock. Um aumento gigante em apenas 5 anos.

Utilizar o tempo de processamento ocioso da placa de vídeo para executar algoritmos complexos de inteligência artificial seria uma alternativa interessante, mas seria complicado, pois os dados precisariam ser transformados por um programa e depois de processado os resultados precisariam ser coletados, o que degradaria o desempenho das placas de vídeo.

Uma alternativa interessante seria o processador Intia, que a empresa AISeek vem desenvolvendo para realizar processamento de IA, segundo ela, o chip funciona 200x mais rápido que o processador aritmético para cálculos de caminho mais curto, simulação sensorial, detecção, análise de terreno[16].

Porém, uma aquisição de um novo módulo para aumentar o desempenho de um jogo é indesejado para muitas pessoas em função do custo. As empresas de jogos vêm tentando achar maneiras diferentes para ter uma IA mais aprimorada com um menor custo.

2. É possível modificar a IA de acordo com o nível de CPU utilizada?

Uma alternativa possível seria ter vários algoritmos de IA armazenados em cache, e utilizar uma verificação em tempo de execução para verificar a taxa de *frames* do jogo, e se estiver muito acima de 24 frames por segundo, carregar uma IA mais

sofisticada, e se a taxa de frames estiver muito baixa, carregar uma IA menos sofisticada para os NPC (*Non-Player Character*).

No exemplo da figura 8 foram definidos em pseudo código, 5 níveis de inteligência e essa IA é iniciada no nível 3, um nível intermediário. E alterada dinamicamente em tempo de execução.

```
IA_atual=3
Enquanto (gameLoop)
inicio
    se (framesPorSegundo<24 E IA_atual>1) então
        carregaIA(--IA_atual)
    se (framesPorSegundo>28 E IA_atual<5) então
        carregaIA(++IA_atual)
fim
```

Fig 8 – Pseudo código para a verificação do nível de IA

No entanto, é possível que com essa alteração de nível de inteligência possam ocorrer algumas falhas, como por exemplo, se algum NPC com sua IA selecionada para o nível um se trancar em algum lugar do cenário em função da baixa inteligência, a probabilidade dele não sair do lugar quando uma AI mais sofisticada for carregada é muito grande. Outro ponto relevante é que enchendo a memória cache com algoritmos de IA pode causar uma perda significativa na desempenho do jogo, pois como a memória é limitada, deixar armazenados muitos dados relativos a IA não convém já que existem prioridades maiores como a própria *engine* do jogo.

Uma alternativa interessante é a da empresa kynogon[17], que tem desenvolvido várias soluções de IA utilizando *C++* e *Lua Script*. Ela traz uma eficaz resolução ao problema da movimentação dos NPCs pelos cenários utilizando *Dynamic Path Finding* que é um dos tópicos favoritos para os desenvolvedores de jogos, pois é muito comum ver NPCs perdidos pelo cenário, correndo na direção das paredes, caindo em buracos, incapazes de abrir uma porta, etc. Segundo a empresa, a proposta de solução tem baixo consumo de processamento e maior otimização nas tomadas de decisão. Recentemente jogos de grandes desenvolvedoras tem utilizado a linguagem Lua script para a implementação da IA por ser rápida, portátil e embutível[18], um bom exemplo de sua utilização em IA é o jogo *Crysis* que contém algoritmos complexos para o comportamento dos NPCs [19].

Se for utilizado o processador proposto pela AISeek descrito no primeiro ponto-chave juntamente com a solução de *Dynamic Path Finding* apresentado pela empresa kynogon é possível que o ganho na performance dos jogos que necessitam de uma IA mais sofisticada seja ainda maior.

3. Qual é a necessidade de que os NPC que estejam muito longe sejam inteligentes?

Há uma variedade de maneiras diferentes dos inimigos reagirem com o personagem principal, mas se o inimigo estiver muito longe, não há possibilidade disso acontecer. Por isso, os inimigos que estejam longe deveriam ter uma IA bem simples, e

quando forem se aproximando do personagem o nível de complexidade da IA irá aumentando.

4. Aproveitar momentos de ociosidade do jogo.

Caso algum algoritmo de IA com alto custo computacional necessite ser aperfeiçoado, essa mudança poderia ser realizada em momentos ociosos do jogo, como quando o personagem ficar parado por muito tempo, ou o jogo for pausado.

5. Existe alguma técnica de IA boa, e que tenha um custo computacional relativamente baixo?

Uma técnica que melhoraria muito o desempenho dos jogos seria a utilização de redes neurais já treinadas, ou treinadas em momentos ociosos do jogo, como explicado no ponto-chave 4. Isso poderia ser feito utilizando o programa SNNS (*Stuttgart Neural Network Simulator*)[11], escolhendo a rede com o melhor desempenho (menor erro) e então gerando o código executável dela em C através do utilitário `snns2c.exe`, inclusive com o código do pacote do SNNS.

O tempo de execução seria bem rápido, já que a rede já estaria treinada, seria apenas executada. No próximo capítulo escrevemos uma síntese sobre Redes Neurais.

6. Redes Neurais

O conceito de redes neurais artificiais surgiu na década de 40 e visa criar um modelo matemático que simule a estrutura neural do cérebro e permita a aquisição de conhecimento através do estudo de informações.

Assim como no cérebro, o elemento principal de uma rede neural artificial é o neurônio. Na estrutura artificial, os dendritos foram substituídos por entradas cujas ligações com o corpo celular são realizadas através de elementos chamados de peso que simulam a sinapse. Os estímulos captados pelas entradas são processados pela função de soma, e o disparo do neurônio biológico foi substituído pela função de transferência.

Uma rede neural artificial é formada por uma série de neurônios classificados em "entrada", "escondidos" e "saída", que se conectam simulando o contato dos dendritos que ocorre com neurônios biológicos, mas do modo artificial em um número consideravelmente menor.

Os neurônios de entrada recebem os valores, a camada escondida, os processa e envia para os neurônios de saída. A partir dos valores resultantes nos neurônios de saída o algoritmo de treinamento ajusta os pesos das conexões entre os neurônios e o valor de BIAS para que a rede fique com um erro menor.

A propriedade mais importante das redes neurais é a habilidade de aprender e melhorar seu desempenho. Isso é feito através de um processo iterativo de ajustes em seus pesos, que é chamado de "treinamento" da rede. O aprendizado ocorre quando a rede neural atinge o conhecimento necessário para dar uma resposta condizente com a realidade. Veja um modelo de RNA criada no programa JavaNNS (figura 9).

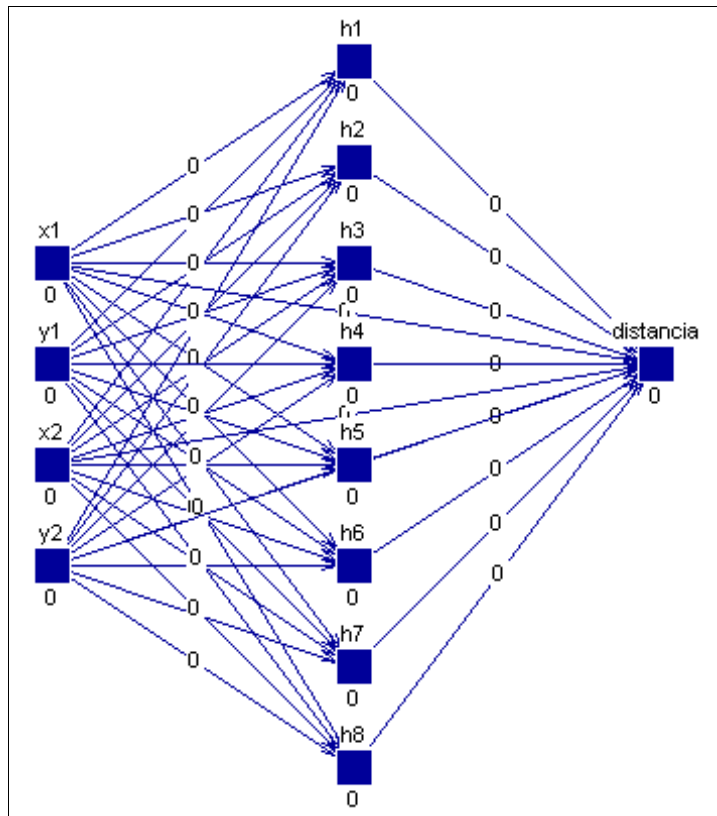


Figura 9 - Modelo de uma rede neural.

7. Hipótese

Provar que uma rede neural já treinada é mais rápida do que realizar cálculos complexos. Para isso, utilizamos a fórmula da distância, descrita por:

$$Distância_{a,b} = \sqrt{((x_b - x_a)^2 + (y_b - y_a)^2)}$$

Figura 10 – Distância entre dois pontos

Esta é uma fórmula relativamente simples, provavelmente para número pequenos o cálculo pelo co-processador aritmético será mais rápido do que a utilização da rede neural, as redes neurais se destacariam em cálculos mais complexos como em Equações Diferenciais e Sistemas Dinâmicos. Mas o nosso objetivo é mostrar que redes neurais apresentam um desempenho similar ao de fórmulas complexas para uma determinada faixa de valores com um custo computacional baixo e constante.

8. Geração dos Dados

Para geração dos dados utilizamos o programa R[20], com o seguinte script (figura 11):

```

distancia<-NULL
for (x1 in c(2,4,6)){
  for (y1 in c(2,4,6)){
    for (x2 in c(3,5,7)) {
      for (y2 in c(3,5,7)){
        distancia<-rbind(distancia,c(x1,x2,y1,y2,sqrt((x2
-x1)^2+(y2-y1)^2) ))
      }
    }
  }
}

```

Figura 10 – Script R para geração dos dados

O script para geração dos dados percorre possibilidades de x_1 e y_1 de 2 até 6 de 2 em 2. E percorre x_2 e y_2 de 3 até 7 de 2 em 2, totalizando 81 dados. Estes serão os dados de entrada da Rede Neural. A saída será a fórmula da distância, mostrada anteriormente. Destes dados foram selecionados aleatoriamente 1/3 dos dados para o conjunto de testes e o restante para o treinamento.

9. Treinamento

Para treinamento e teste da rede foi utilizado o programa SNNS. Variamos a inicialização de 1 a 10, neurônios na camada escondida (0,1,2,4,8,16) e épocas (1000,2000,5000). Os dados foram normalizados com o objetivo de obter um melhor desempenho da rede neural.

10. Resultados

De todas as rede testadas, a que obteve o menor erro foi a com 16 neurônios na camada escondida, na 5 inicialização e com 5000 épocas de treino. O erro (SSE) foi de $8.863706e^{-02}$.

Depois de desnormalizados os dados, plotamos um gráfico (figura 11) com os valores das saídas. Os dados em preto diz respeito aos dados que deveriam ter sido a saída da Rede Neural (dados de *pattern*). E os dados em vermelho são as saídas da Rede Neural.

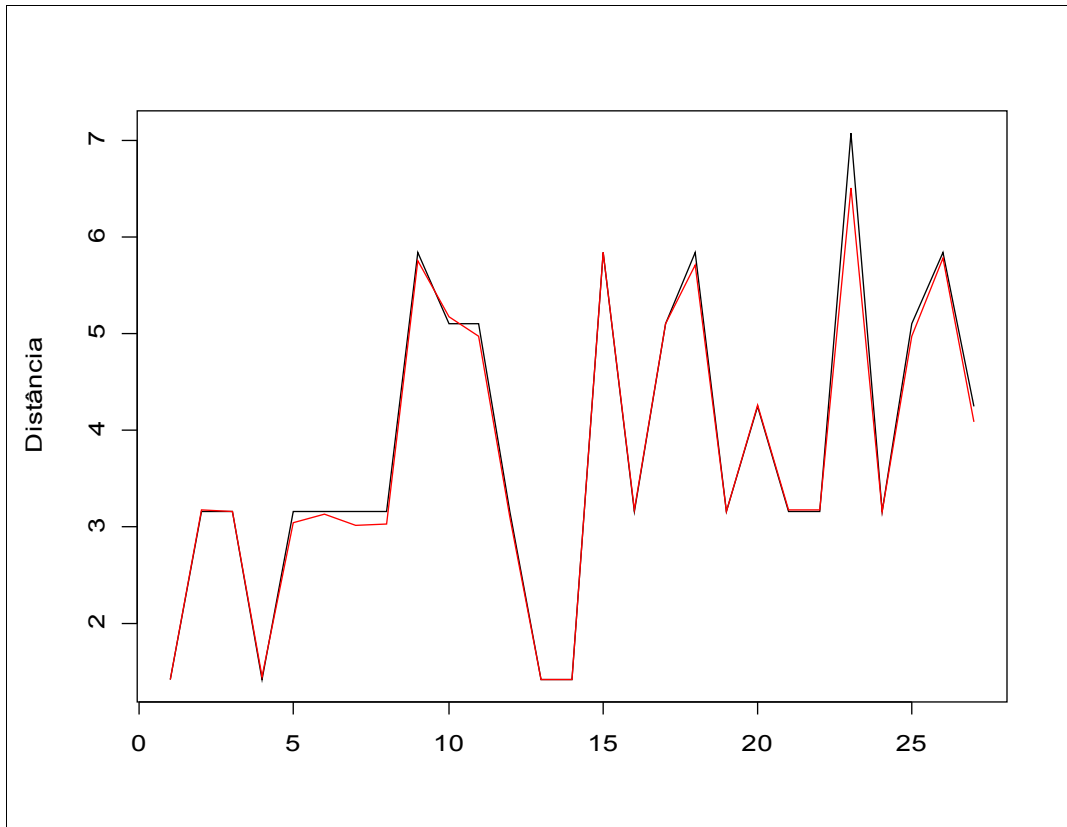


Figura 11 – Saídas da Rede Neural

Como podemos perceber, os dois gráficos ficaram semelhantes, plotamos na Figura 12 um gráfico com as saídas da rede em função das saídas que deveriam ser as respostas corretas da RNA. Quanto mais pontos ficarem dentro da diagonal principal menor será o erro da rede.

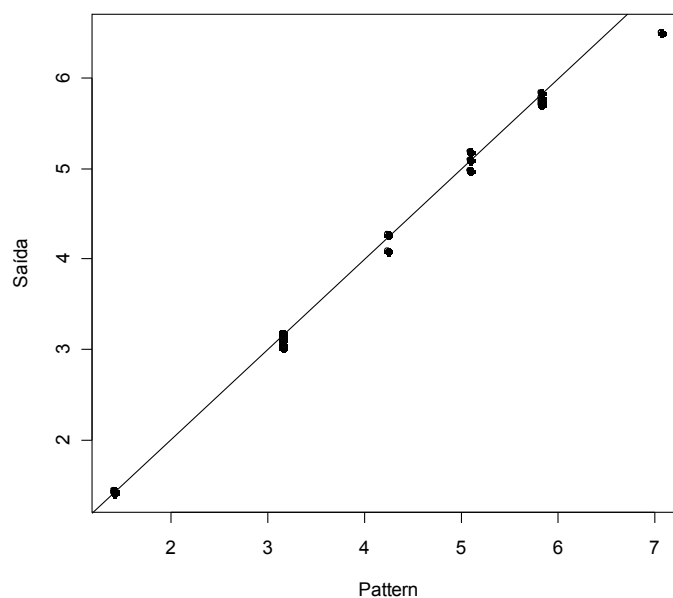


Figura 12 – Comparação entre a Saída da Rede e a Saída Pattern

11. Comparação entre métodos

Depois de termos escolhido a rede neural com menor erro, geramos o código C dela através do utilitário `sns2c.exe`, após isto implementamos um programa em C que executava 10 milhões de vezes o código da Rede Neural e depois disso executava 10 milhões de vezes o código da fórmula da distância.

O resultado que obtivemos foi que por instrução a RNA demorava $7.3e^{-06}$ segundos e a fórmula da distância demorava $3e^{-07}$. Ou seja, para cada chamada da Rede Neural, a fórmula da distância executava 24 vezes. Depois de analisado o código em C gerado automaticamente pelo SNNS, percebemos que essa diferença nos tempos de execução se deve principalmente à função sigmóide que é utilizada nos neurônios da camada escondida na rede neural. A função sigmóide é representada por:

$$f(x) = \frac{1}{1 + e^{-z}}$$

Figura 13 – Função Sigmóide

Essa função é mais complexa que a fórmula da distância, pois exige que o processador calcule um exponencial, potencialmente grande, de números fracionários.

Para provar que essa fórmula é o gargalo das Redes Neurais alteramos o código em C gerado automaticamente pela RNA e substituímos o cálculo da função sigmóide por uma pesquisa em um array, este array poderia conter valores pré-calculados da função para otimizar o desempenho da RNA.

Após as novas comparações, podemos constatar que a Rede Neural sem a função sigmóide na camada oculta teve um desempenho 340% superior à Rede com a função sigmóide ativa (Figura 14).

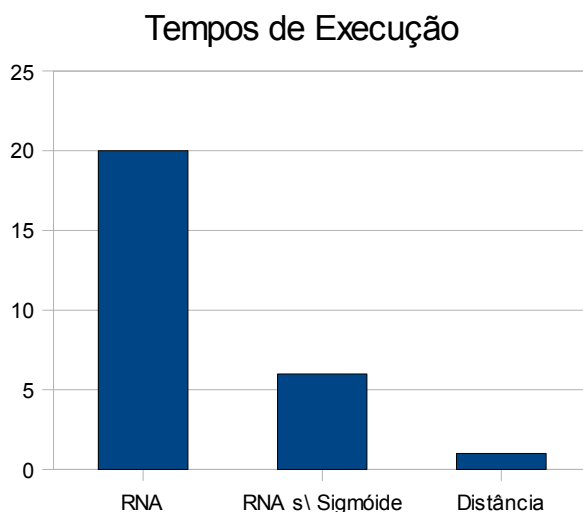


Figura 14 – Comparação dos Tempos de Execução

Obviamente, este método de utilizar um array com valores pré-calculados também poderia ser utilizado na função da distância para calcular a raiz quadrada.

Quanto mais se busca otimização e redução da complexidade de algoritmos, menor será a sua precisão.

12. Conclusão

Um grande fator que tornou possível a evolução dos jogos foi a criação de bibliotecas gráficas poderosas, como a SDL, OpenGL e DirectX, que permitiram que algumas tarefas árduas e em mais baixo nível dos programadores fossem mais abstraídas e automatizadas, tornando o desenvolvimento de jogos mais rápido e focado no desenvolvimento do jogo propriamente dito, e não tanto nos recursos módulos de que ele necessitaria para funcionar.

A quantidade de CPU reservada a IA continua baixa e estável em 10%. Isso nos motivou a escrever este artigo para tentar otimizar as técnicas atuais e não sobrecarregar a CPU. Porém métodos muito sofisticados só poderão ser utilizados se for disponibilizado um hardware específico para isso. Uma solução poderia ser o uso de Redes Neurais, como demonstramos anteriormente, elas tem um desempenho constante e um erro baixo quando os dados não tem um intervalo muito grande de valores. Porém, como demonstrado, ela não é eficiente em cálculos relativamente simples. Ela seria útil em Sistemas Dinâmicos, onde os cálculos tem uma complexidade maior do que a execução da Rede Neural.

Cada projeto de jogo deve analisar qual é o público alvo do seu aplicativo e criar um projeto apropriado, quanto maior for a precisão de Inteligência Artificial utilizada, maior será o custo computacional. Tentamos neste trabalho demonstrar algumas técnicas que poderiam ser aplicadas para reduzir o custo de alguns algoritmos necessários em jogos, porém, quanto mais otimizada for a inteligência, menos precisa ela será. Cabe aos projetistas analisarem qual o melhor custo/benefício e desenvolver uma inteligência artificial adequada.

A alternativa do processador de IA Intia, citada no ponto chave 1, pareceu ser outra alternativa bem interessante. Quem sabe num futuro próximo, esse processador de IA já não venha integrado na placa de vídeo. Assim, os jogos se tornarão cada vez mais atrativos surpreendendo e desenvolvendo cada vez mais a curiosidade e fantasia do jogador

13. Referências Bibliográficas

- [1] *Cheat Book*. Disponível em: <<http://www.cheatbook.de/wfiles/danger1.htm>> Acesso em: 20 de agosto de 2007.
- [2] *Wikipédia*. Disponível em: <http://pt.wikipedia.org/wiki/Prince_of_Persia> Acesso em: 21 de agosto de 2007.
- [3] *Uol Jogos*. Disponível em: <<http://jogos.uol.com.br/analises/pc/ult398u328.jhtm>> Acesso em: 22 de agosto de 2007.
- [4] *Terra Games*. Disponível em: <<http://games.terra.com.br/interna/0,,OI950348-EI6524,00.html>> Acesso em: 22 de agosto de 2007.
- [5] Foley, J.D. et al. *Computer graphics: principles and practice*. Reading: Addison-Wesley, 1990.

- [6] *SDL*. Disponível em: <<http://www.libsdl.org/index.php>> Acesso em: 8 de setembro de 2007.
- [7] *Wikipédia SDL*. Disponível em: <[http://pt.wikipedia.org/wiki/SDL_\(biblioteca\)](http://pt.wikipedia.org/wiki/SDL_(biblioteca))> Acesso em: 8 de setembro de 2007.
- [8] ANDRADE, Fernando, ANDRADE, Gustavo, LEITÃO, André, FURTADO, André, RAMALHO, Geber. *Knock'em: Um Estudo de Caso de Processamento Gráfico e Inteligência Artificial para Jogos de Luta*. II Workshop Brasileiro de Jogos e Entretenimento Digital, 2003.
- [9] *Tomb Raider Forums*. Disponível em: <<http://www.tombraiderforums.com/archive/index.php/t-95136.html>> Acesso em: 11/09/2007
- [10] ANDRADE, Gustavo D. *Aprendizagem por Reforço e Adaptação ao Usuário em Jogos Eletrônicos*. 2004. 43 f. Dissertação (Graduação em Ciência da Computação) Centro de Informática, Universidade Federal de Pernambuco, [2004].
- [11] *SNNS*. Disponível em <<http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html>> Acesso em: 08/10/2007
- [12] *Clube do Hardware - Tabela comparativa dos chips da nVidia*. Disponível em: <<http://www.clubedohardware.com.br/artigos/519>> Acesso em: 09/10/2007
- [13] *Boa dica Nvidia*. Disponível em: <<http://www.boadica.com.br/layoutdica.asp?codigo=167>> Acesso em: 09/10/2007
- [14] BITTENCOURT, João Ricardo, GONZALEZ Esteban Walter C. *Uma Nova Concepção para a Criação de Jogos Educativos*. Simpósio Brasileiro de Informática na Educação, 2004.
- [15] YAMAMOTO, Flávio S. et al. *Inteligência Artificial em Jogos Eletrônicos Interativos*. Escola Politécnica Universidade de São Paulo (EPUSP).
- [16] *AiSeek WhitePaper*. Disponível em: <<http://www.aiseek.com/AiSeek%20-%20Intelligence%20for%20New%20Worlds.pdf>> Acesso em: 10/10/2007
- [17] *Kynogon Pathfinding is not A star* Disponível em: <<http://www.kynogon.com/images-blog/Documents/WPG2Pathfinding.pdf>> Acesso em: 21/10/2007
- [18] *Lua A linguagem de programação* Disponível em: <<http://www.lua.org/portugues.html>> Acesso em: 23/10/2007
- [19] *Crytek cryengine 2* Disponível em: <http://www.crytek.com/fileadmin/user_upload/cryengine2/CryENGINE2Features.pdf> Acesso em: 23/10/2007
- [20] *R Project*. Disponível em: <<http://www.r-project.org>> Acesso em 19/11/2007.
- [21] *ComparaExecução* Disponível em: <<https://sourceforge.net/projects/distancia/>> Acesso em: 21/11/2007