# Navegação autônoma em jogos eletrônicos utilizando algoritmos genéticos

Vitor Borgatto Appolinario
Graduado em Ciência da Computação [Uninove]
Sorocaba – São Paulo [Brasil]
vitorappolinario@gmail.com

Tompson Laureano Pereira
Graduado em Ciência da Computação [Uninove]
São Paulo – São Paulo [Brasil]
tompsonlp@gmail.com

Neste trabalho, explora-se a utilização de algoritmos genéticos em aplicação de entretenimento eletrônico. Com base na teoria da computação evolutiva (Algoritmos Genéticos – AGs), foi desenvolvido um algoritmo para resolver um problema de navegação autônoma de um agente em um cenário virtual. Foram realizados testes para averiguar a viabilidade dos AGs na resolução desse problema.

**Palavras-chave:** Algoritmos genéticos. Inteligência artificial. Jogos de computador.

## 1 Introdução

Na indústria de jogos eletrônicos, a inteligência artificial (IA) sempre foi empregada para prover ao usuário uma experiência mais próxima a de estar competindo com outro ser humano. Dessa forma, as tomadas de decisão dos agentes inteligentes nesse tipo de aplicação, além de serem em tempo real, precisam resultar em ações compatíveis com as de um jogador perspicaz.

Usualmente, em jogos, personagens controlados por computador utilizam algum tipo de sistema baseado em regras para interagir com o jogador. Nesses casos, a interação entre computador e usuário pode ser limitada, pois caso surga uma situação não definida pelas regras, o aplicativo provavelmente não encontrará ação correspondente. Nesse sentido, temos, entre as diversas técnicas de IA, os algoritmos genéticos (AG), que podem ser empregados como solução alternativa em muitos jogos em que se utilizam outros métodos tradicionais como busca em árvores. Isso porque os AGs podem encontrar uma solução viável em ambientes nos quais algumas regras não tenham sido previamente programadas.

Os algoritmos genéticos são métodos de busca e otimização baseados no processo de seleção natural que simulam a evolução. Nesse sentido, os AGs geram indivíduos (cadeias de *bit*s) também chamados de cromossomos, que evoluem em busca da solução de um dado problema (RUSSEL; NORVIG, 2002).

Neste artigo, procura-se estudar o desenvolvimento e a aplicação dos AGs em jogos para os quais haja necessidade de movimentação autônoma de um agente.

## 2 Jogos eletrônicos

Antes da década de 1970, algumas empresas e pessoas já objetivavam a criação de jogos. Uma das principais, a Nintendo, iniciou suas atividades como empresa fabricante de cartas de baralho, em 1889. Segundo a linha de tempo encontrada em Kent (2001), esse é o primeiro marco da história dos jogos eletrônicos (pelo fato de a Nintendo hoje pertencer ao mercado de jogos eletrônicos).

O primeiro jogo eletrônico interativo criado na história foi o Spacewar, em que duas pessoas controlavam dois tipos diferentes de espaçonaves que deveriam combater entre si. Esse jogo foi programado por um estudante do MIT, Steve Russell, em um computador PDP-1, em 1961 (DEMARIA; WILSON, 2004). Uma das versões do jogo se tornou a primeira máquina de fliperama da história, em 1971, com o nome de Computer Space (Figura 1).

Em 1972, foi criada a Atari, famosa pelo jogo Pong (criado pelo engenheiro Al Alcorn). Até o fim daquela década, muitas empresas entraram no mercado de jogos, como a Taito, Midway e Capcom, além da Magnavox, que lançou, em 1972, o computador Odyssey (KENT, 2001).

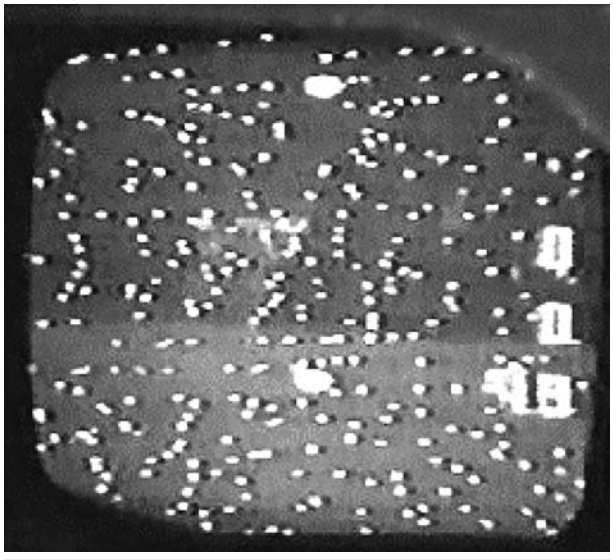

**Figura 1: Tela do jogo Computer Space de 1971**
Fonte: Os autores.

Nos anos 1980, os fliperamas tinham um mercado muito ativo, com vários lançamentos como Donkey Kong, Tron e Q*Bert, ao mesmo

tempo que surgiram os primeiros videogames 8-*bit*: Famicom, da Nintendo, e Master System, da Sega (Service Games).

Na área de jogos para computador, houve também um grande lançamento de jogos e criação de empresas, sendo a *On-line Systems* (atual Sierra *On-line*) uma das pioneiras no setor (DEMARIA; WILSON, 2004).

A década de 1990 foi marcada pelo lançamento de batalhas de videogames de 16-*bit* (Sega Genesis e Super Famicom, da Nintendo), de 32-*bit* (PlayStation, da Sony, e Sega Saturn) e pelo lançamento de videogames como o 3DO, da Panasonic, e o Nintendo64, da Nintendo (esse de 64-*bit*) (KENT, 2001). Um grande marco na história de jogos para computador foi o lançamento do jogo Wolfstein 3D pela *Id Software*, em 1991, o primeiro jogo de tiro em primeira pessoa (KUSHNER, 2003).

No fim da década de 1990 e início de 2000, Sony e Nintendo divulgaram seus novos videogames de 128-*bit* (PlayStation 2 e GameCube, respectivamente). À época, a Microsoft entrou nesse segmento com o seu XBOX. Atualmente, a atenção da mídia e dos jogadores está voltada para os jogos *on-line* multiplayer massivos (centenas a milhares de pessoas jogando ao mesmo tempo, em um mundo virtual, via internet) e para os videogames portáteis Sony PSP e Nintendo DS.

## 3 Algoritmos genéticos

Sucintamente, a teoria da evolução das espécies nos diz que a natureza possuiu, em algum momento da existência deste planeta, os meios para modificar os seres vivos a cada geração. Essas modificações ocorreram por meio de alterações do código genético desses seres, seja por mutações aleatoriamente produzidas por radiações ionizantes, vírus, ou mesmo ação química. Esses seres vivos com código genético alterado se reproduziram, naquele momento, em combinação com os seres vivos típicos de suas espécies, produzindo uma nova geração de seres vivos, que incorporou, de alguma forma, as mutações ocorridas (CARVALHO, 2001).

Os novos seres vivos que, naquele momento, estavam bem adaptados ao meio ambiente do planeta tinham mais chance de sobrevivência e, dessa forma, cresciam e se reproduziam mais do que os seres menos adaptados, passando, assim, suas características de bom ajuste para as gerações futuras. Esse processo é denominado seleção natural do ser vivo (CARVALHO, 2001).

Algoritmos genéticos constituem uma família de modelos computacionais inspirados na teoria da evolução das espécies. Esses algoritmos modelam uma solução para problemas específicos, em uma estrutura de dados como a de um cromossomo, e aplicam operadores que recombinam essas estruturas, preservando informações críticas (SHIRAI, 1988). Constituem poderosa ferramenta com aplicação em problemas complexos, cujos espaços de busca das soluções ótimas podem ser muito grandes para determiná-las com precisão, por meio de um método direto. Essas soluções, em alguns casos, podem nem sequer existir e, muitas vezes, o que procuramos é meramente uma aproximação que nos traga resultado satisfatório para o problema.

A implementação do algoritmo genético começa com uma população de indivíduos (estados de busca) gerados aleatoriamente, chamados de população inicial. Essas estruturas são avaliadas para gerar oportunidades reprodutivas, de forma que os cromossomos que representam uma solução "melhor" tenham mais chances de reprodução do que os que contém uma solução "pior". A definição de uma solução melhor ou pior é tipicamente relacionada à população atual e quantificada pelos resultados da função de *fitness*.

### 3.1 Cromossomos

Geralmente, os cromossomos (Figura 2) são representados como um vetor numérico de possibilidades finitas, na maioria das vezes composto de valores binários. A menor parte representa um alelo.

| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|

**Figura 2: Exemplo de cromossomo. Cada célula representa um alelo**
Fonte: Os autores.

### 3.2 População

O conceito de população aplica-se a um grupo de indivíduos – cada um representado por um cromossomo – que detenham prováveis soluções para o mesmo problema, sendo diferentes entre si.

Podemos chamar de geração cada nova população produzida nas iterações do algoritmo.

### 3.3 Função de avaliação

A função de avaliação de um problema de otimização é construída a partir dos parâmetros que o envolvam. Ela fornece uma medida da proximidade da solução no que se refere a um conjunto de parâmetros que, por sua vez, podem ser conflitantes, ou seja, quando um aumenta, o outro diminui. O objetivo é encontrar o ponto ótimo. Essa função permite o cálculo da aptidão bruta de cada indivíduo, o que fornecerá o valor a ser usado para avaliar sua probabilidade de ser selecionado para reprodução – o *fitness*, que representa o grau de adaptação do indivíduo ao problema. Nesse aspecto, quem possuir o maior valor de *fitness* será considerado uma das melhores soluções.

## 4 Mutação

A mutação gera novos indivíduos (cromossomos) com pequenas modificações arbitrárias de um ou mais *bit*s do cromossomo selecionado (Figura 3), para aumentar a diversidade da população. Uma mutação pontual altera os 1s por 0s ou vice-versa.

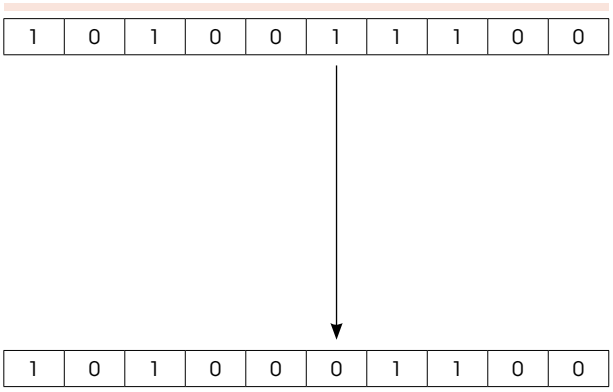


**Figura 3: Exemplo de mutação**
Fonte: Os autores.

O operador de mutação permite ao algoritmo genético explorar novas possibilidades e pode impedi-lo de ficar preso em uma solução local. Ao ser aumentada a probabilidade de mutação, estar-se-á também aumentando a liberdade do algoritmo para buscar novas soluções na região de domínio do parâmetro. Entretanto, o algoritmo genético poderá levar mais tempo para convergir, ou até não chegar a nenhum resultado, uma vez que a mutação também pode destruir uma boa solução.

Tipicamente, a probabilidade de mutação está em torno de 1% a 5% dos *bit*s de uma população. Em alguns AGs "elitistas", existem mecanismos de preservação dos melhores cromossomos, de forma que os impeçam de sofrer mutações, com o objetivo de não perder uma boa solução.

## 5 *Crossover*

O *crossover* representa uma transformação da mais alta ordem. Após algumas iterações do algoritmo genético, espera-se que o melhor indivíduo (o cromossomo com maior *fitness*) represen-

te uma solução quase ótima, ou seja, o programa converge para isso.

O operador de recombinação ou reprodução (*crossover*) permite combinar dois cromossomos de uma mesma população para formar dois descendentes similares. Esse operador pode ser aplicado em posições aleatórias e com certa probabilidade de ocorrência, denominada probabilidade de recombinação. O ponto de *crossover* é escolhido aleatoriamente entre o primeiro e o último *bit* que formam os cromossomos-pais.

A cada iteração do algoritmo, uma nova população é formada por meio da seleção dos indivíduos mais adaptados (cromossomos com maior *fitness*). Alguns membros dessa nova população sofrem alterações, que são implementadas pela utilização dos operadores genéticos (mutação e *crossover*).

## 6 Buscas em árvores

Podem ser utilizadas quando existem várias opções com valores conhecidos, a fim de encontrar a melhor seqüência para determinado problema. A partir do estado inicial é gerada uma árvore, utilizando-o como raiz, e com a expansão dos estados possíveis são criados os próximos nós dessa árvore. Há diversos tipos de estratégias de busca. Para os experimentos demonstrados neste trabalho foi utilizada a busca pela melhor escolha.

### 6.1 Busca pela melhor escolha

Trata-se de uma especialização do algoritmo geral de busca em árvore, no qual um nó é selecionado para expansão com base na função de avaliação (Equação 1). Tradicionalmente, o nó com a avaliação mais baixa é selecionado para a expansão porque essa avaliação mede a distância até o objetivo. A busca pela melhor escolha pode ser implementada em nossa estrutura geral de busca por meio de uma fila de prioridades, uma estrutura de dados que manterá a borda em ordem ascendente de valores da Equação 1 (RUSSEL; NORVIG, 2002). É um método de busca que procura otimizar a solução, considerando todas as informações disponíveis até aquele instante, não apenas as da última expansão. Parte do princípio de que, expandindo o nó da árvore que estiver mais próximo ao destino, será possível encontrar uma solução ideal.

$$f(n) = h(n) \quad (1)$$

em que h(n) é o custo do caminho mais econômico do nó "n" até o nó objetivo. Como a meta ainda não foi atingida, h(n) geralmente não é conhecido, sendo apenas uma estimativa. A busca expande o nó com o menor valor de f(n), e quando f(n) for igual a zero, o nó atual será o nó objetivo, e a busca estará concluída (RUSSEL; NORVIG, 2002).

## 7 Metodologia

Para este trabalho, pesquisou-se a literatura da área. Com base nos dados levantados, foi desenvolvido um jogo com interface gráfica, no qual se deseja que um tanque de guerra (agente) saia de um ponto qualquer do cenário e atinja seu alvo, desviando de obstáculos inseridos, de forma aleatória, no ambiente. Para a tarefa de navegação do agente, um algoritmo genético foi desenvolvido. Além disso, a implementação da busca pela melhor escolha (ARAÚJO; LIBRANTZ; FLÓRIO FILHO, 2006) foi utilizada para realização de um estudo comparativo entre as duas técnicas.

O jogo foi desenvolvido, utilizando-se a linguagem C++ para o núcleo do programa (envolvendo o AG), e a biblioteca Simple Direct Layer (SDL) versão 1.2.8, para o desenvolvimento da

parte visual. O aplicativo destina-se à plataforma Linux e processadores da classe x86.

Ao iniciar o aplicativo, é gerado um cenário (Figura 4), em que são inseridos, de maneira aleatória, alguns obstáculos. Esse cenário, por sua vez, é armazenado em uma matriz numérica $M_{10,10}$. Em seguida, o agente é posicionado na célula 0,0 e o alvo definido em 9,9.

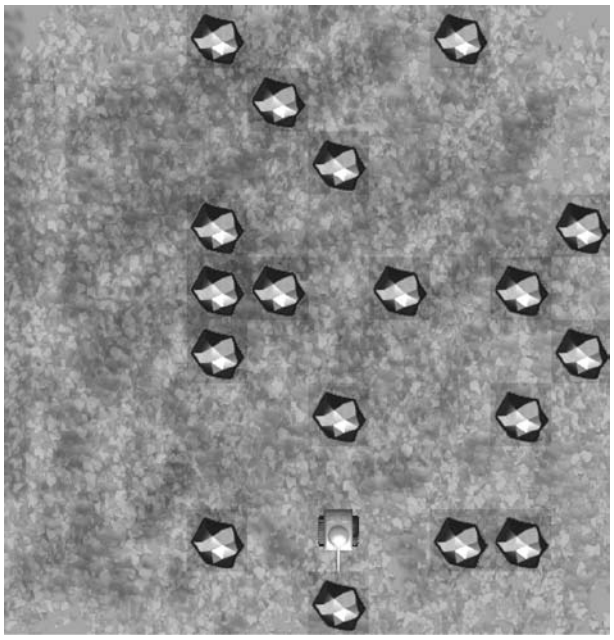

**Figura 4: Tela do aplicativo**
Fonte: Os autores.

Mais adiante, foi experimentado o AG com mapas maiores para avaliar seu desempenho. Nesse caso, a matriz passa a ser $M_{30,30}$ e $M_{50,50}$, e a posição final, 29,29 e 49,49, respectivamente.

Os elementos da matriz podem ser lidos da seguinte maneira: 0 significa espaço livre no qual o agente consegue mover-se; -1 representa obstáculo; -9 significa o ponto de partida do agente, e 9 indica o alvo (local de chegada).

Com base nessa matriz, o AG deve ser executado, gerando uma rota para que o agente alcance seu destino. Em um primeiro instante, o AG cria uma população aleatória, realizando, logo em seguida, a avaliação dessa população.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -9 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | -1 |
| 0 | 0 | 0 | -1 | -1 | 0 | -1 | 0 | -1 | 0 |
| 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | -1 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 9 |

**Figura 5: Matriz do ambiente**
Fonte: Os autores.

Nesse aplicativo, foi definido um cromossomo de tamanho variável entre 60 e 80 alelos, cada um armazenando um valor binário. Esses valores são lidos aos pares a partir do primeiro alelo. Cada par de alelos define um movimento do agente, da seguinte forma: "00" a 0º; "01" a 270º, "10" a 180º e "11" a 90º. Para a avaliação de uma rota, foram determinados os seguintes aspectos:

1- Movimento válido;
2- Seqüência de movimentos válidos;
3- Posição final.

Quanto ao primeiro item, sua validade é observada se o agente que estiver executando o movimento em questão não ultrapassar os limites do cenário ou não alcançar uma célula ocupada por algum obstáculo. Caso ocorra algum desses fatores, esse cromossomo terá seu *fitness* reduzido. Sendo o movimento válido, o *fitness* é pontuado com um valor diferenciado para cada posição assumida no cenário. Esse valor foi definido pelo cálculo da distância entre a posição ocupada e o objetivo. Para esse cálculo, foram experimentadas três fórmulas distintas para distância, a saber: CityBlock,

Chessboard e Euclidiana (ARAÚJO; LIBRANTZ; FLÓRIO FILHO, 2006), cujas equações são dadas em 2, 3 e 4, respectivamente.

$$d(p,q) = |x - u| + |y - v| \quad (2)$$

$$d(p,q) = \max(|x - u|, |y - v|) \quad (3)$$

$$d(p,q) = \sqrt{(x - u)^2 + (y - v)^2} \quad (4)$$

Com essa informação, divide-se o valor máximo possível a ser acrescido no *fitness*, ou seja, o valor que seria passado ao *fitness* – caso o agente estivesse a uma casa de distância do objetivo – pelo resultado da equação. Dessa forma, tem-se a pontuação relativa a essa casa, que será incrementada ao parâmetro *fitness*. É acrescentado também um contador específico do cromossomo, utilizado para indicar quantos passos válidos ele consegue realizar em uma seqüência. Esse valor é somado ao *fitness* no final da avaliação, com o peso 100.

Nessa etapa, se o agente alcançar o objetivo, o *fitness* será acrescido de um valor substancial para destacar-se dos demais indivíduos no momento da seleção. A execução do código não é interrompida, dada a possibilidade de encontrar uma solução melhor.

Durante a fase de desenvolvimento e de testes do jogo, várias combinações dos parâmetros passados ao AG foram estudadas, tais como tamanho da população, quantidade de gerações e tamanho do cromossomo de cada indivíduo.

Para verificar o desempenho do AG na tarefa de navegação do agente, foram realizados 120 testes com cromossomos de 60 a 80 alelos. Variaram-se também a quantidade de indivíduos das populações (entre 80 e 100 indivíduos) e a de gerações necessárias para se encontrar uma rota (entre 80 e 100).

Cada combinação de parâmetros foi experimentada dez vezes. As médias de seus resultados são apresentadas na Tabela 1:

**Tabela 1: Comparativo de mudanças de parâmetros**

| População | Gerações | Cromossomo | Acerto (%) |
|---|---|---|---|
| 100 | 100 | 80 | 50 |
| 100 | 100 | 70 | 70 |
| 100 | 100 | 60 | 60 |
| 100 | 80 | 80 | 50 |
| 100 | 80 | 70 | 80 |
| 100 | 80 | 60 | 30 |
| 80 | 100 | 80 | 60 |
| 80 | 100 | 70 | 100 |
| 80 | 100 | 60 | 60 |
| 80 | 80 | 80 | 70 |
| 80 | 80 | 70 | 80 |
| 80 | 80 | 60 | 80 |
| Média Geral | | | 65,83 |

Fonte: Os autores.

Com base nesses resultados, pode-se verificar que um cromossomo com grande quantidade de alelos não é a melhor alternativa. Observa-se, na Tabela 1, que, em uma situação em que a quantidade de gerações é maior que a quantidade de indivíduos, a taxa de acerto do algoritmo também o é. Constata-se que a combinação de parâmetros com maior taxa de acerto foi: População = 80; Gerações = 100; Cromossomo = 70. Esses mesmos parâmetros foram escolhidos para os testes de variação da quantidade de obstáculos, definidos em 10, 18 e 34 obstruções (Figuras 6, 7, 8, 9, 10 e 11). O tamanho do mapa ficou estabelecido em 10x10. Em um cenário com dez obstáculos, o AG precisou, em média, de 0,21 segundos para ser executado, e a rota teve um custo de 22 passos, em média, para chegar ao destino.

Em um cenário com 18 obstáculos, o AG levou, em média, 0,23 segundos para ser executado, e a rota teve um custo médio de 24 passos para chegar ao destino.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -9 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 |

**Figura 6: Matriz do ambiente com 10 obstáculos**
Fonte: Os autores.

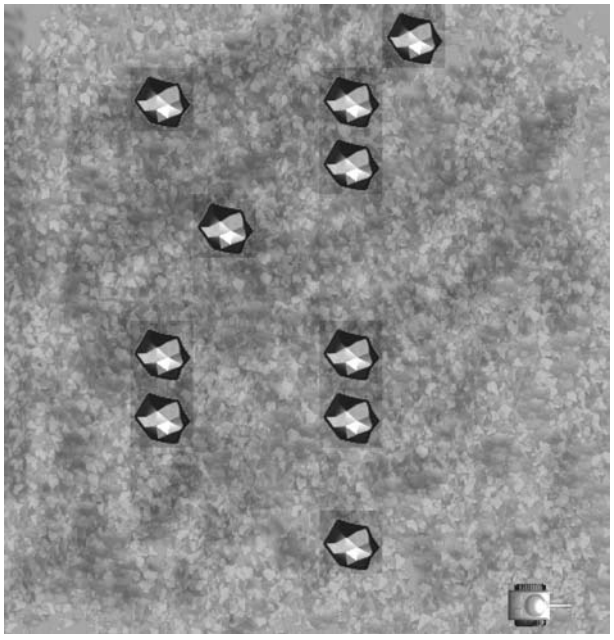

**Figura 7: Tela do aplicativo com 10 obstáculos**
Fonte: Os autores.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -9 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | -1 |
| 0 | 0 | 0 | -1 | -1 | 0 | -1 | 0 | -1 | 0 |
| 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 | -1 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 9 |

**Figura 8: Matriz do ambiente com 18 obstáculos**
Fonte: Os autores.

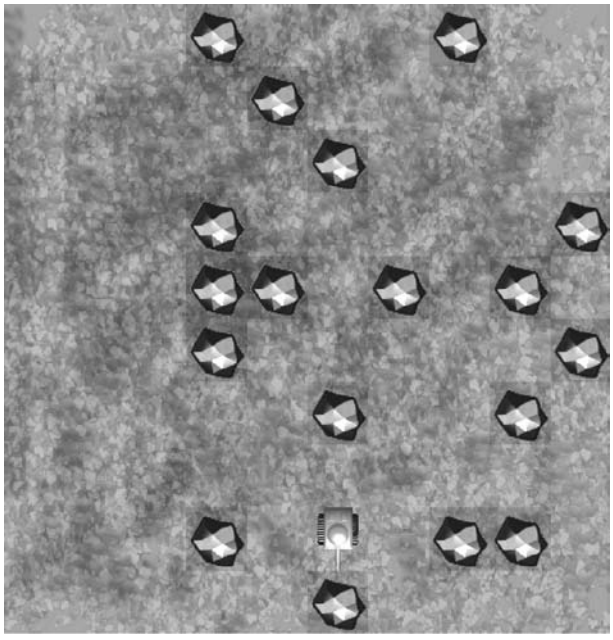

**Figura 9: Tela do aplicativo com 18 obstáculos**
Fonte: Os autores.

Na análise de um cenário com 34 obstáculos, constatou-se que o AG precisou, em média, de 0,22 segundos para ser executado e de 26 passos, em média, para chegar ao destino.

Com esses testes, foi possível observar que o tempo necessário para processar a rota teve pouca variação, aumentando a quantidade de obstáculos. Esse tempo está relacionado à avaliação da população, uma vez que não foram alterados a quantidade de indivíduos e o tamanho do cromossomo.

Já a quantidade de passos aumentou, em razão da necessidade de desviar-se de mais obstáculos.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -9 | 0 | 0 | -1 | -1 | -1 | 0 | 0 | -1 | -1 |
| 0 | 0 | -1 | -1 | 0 | 0 | -1 | -1 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 | -1 | -1 |
| 0 | 0 | -1 | -1 | 0 | -1 | 0 | -1 | 0 | 0 |
| 0 | 0 | -1 | -1 | 0 | 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | -1 |
| 0 | 0 | 0 | -1 | -1 | 0 | -1 | -1 | 0 | 0 |
| 0 | 0 | 0 | -1 | -1 | 0 | 0 | -1 | 0 | -1 |
| 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 9 |

**Figura 10: Matriz do ambiente com 34 obstáculos**
Fonte: Os autores.

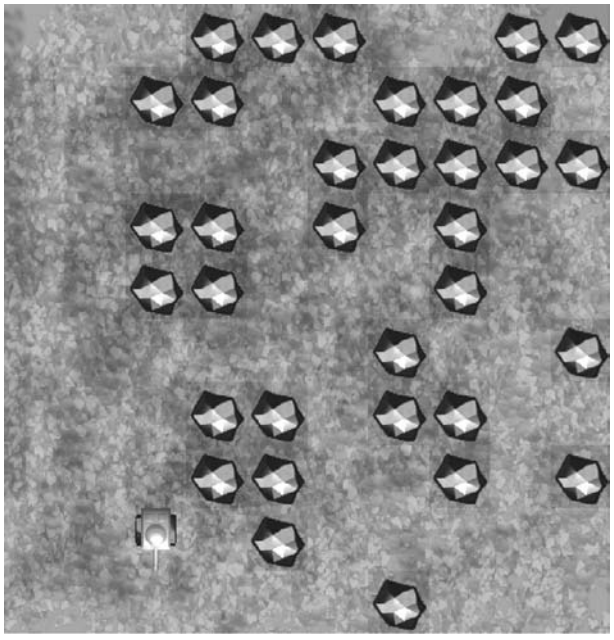

**Figura 11: Tela do aplicativo com 34 obstáculos**
Fonte: Os autores.

## 8 Análise comparativa AG *versus* busca em árvore

Para este estudo comparativo, utilizou-se uma implementação de busca pela melhor escolha (ARAÚJO; LIBRANTZ; FLÓRIO FILHO, 2006), que constrói a rota baseada em uma matriz semelhante à utilizada neste experimento. Após pequena adaptação dos códigos para eliminar essas diferenças de leitura, foram realizados testes para comparar o desempenho das duas técnicas.

Para a fase de testes, foram utilizadas matrizes de diferentes tamanhos, 10x10, 30x30 e 50x50. Com taxa de obstrução de 20% das células que compõem o cenário, cada matriz gerada foi salva em um arquivo-texto, no qual são armazenadas as dimensões da matriz e as posições dos obstáculos. Após a geração desses arquivos, cada programa utilizou essas matrizes como entradas e obteve-se uma rota distinta para cada arquivo. Levantaram-se aspectos como tempo de execução (Figura 12) e número de passos (Tabela 2).

**Tabela 2: Comparativo de desempenho em valores médios (tempo em segundos)**

| Tamanho do cenário | Tempo do AG | Tempo da árvore | Custo AG | Custo árvore |
|---|---|---|---|---|
| 10 x 10 | 0,21 | 0,1 | 20 | 15 |
| 30 x 30 | 0,35 | 0,1 | 151 | 103 |
| 50 x 50 | 0,59 | 0,3 | 428 | 292 |

Fonte: Os autores.

Para averiguar o desempenho dos algoritmos, foram observados o custo de cada rota e o tempo de execução para sua obtenção. O custo foi expresso pelo número de passos de que o agente necessitou para atingir o objetivo.

Em um mapa com dimensões 10x10, o custo médio apresentado pelo AG foi de 20 passos, enquanto a busca em árvore teve custo médio de 15 passos. Mais uma vez houve desvantagem do AG em relação ao tempo de processamento. Média de 0,21 segundos contra 0,1 segundo gasto pelo algoritmo de busca em árvore.

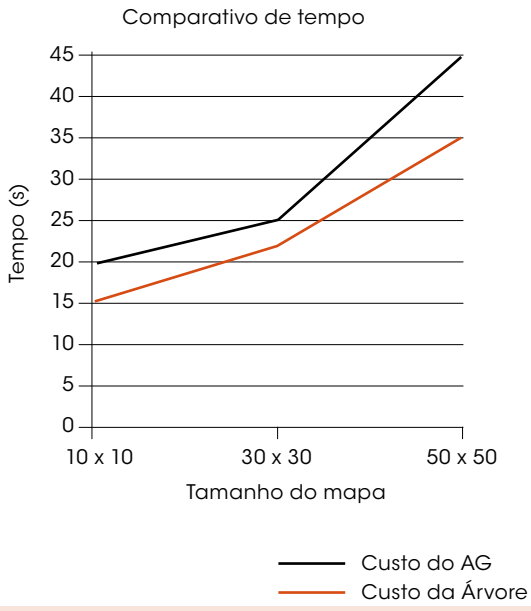


**Figura 12: Gráfico comparando desempenho por tempo de execução**

Fonte: Os autores.

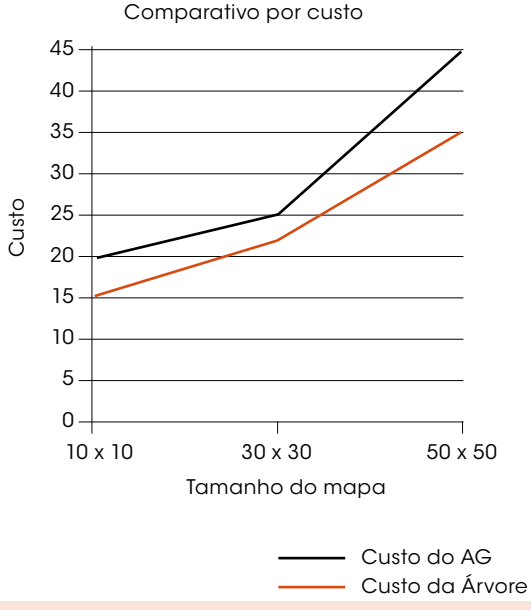


**Figura 13: Gráfico comparando desempenho por custo da rota**

Fonte: Os autores.

Após alteração das dimensões do mapa para 30x30, o AG apresentou um custo médio de 50 passos, enquanto a busca em árvore teve custo médio de 33 passos. Quanto ao tempo de processamento, o AG levou desvantagem novamente: média de 0,35 segundos contra 0,1 segundo gasto pelo algoritmo de busca em árvore.

Já nos testes com o maior mapa utilizado (50x50), o AG apresentou um custo médio de 95 passos, enquanto a busca em árvore teve custo médio de 60 passos. Em relação ao tempo de processamento, houve, mais uma vez, desvantagem do algorítmo: média de 0,59 segundos contra 0,3 segundos gastos pelo algoritmo de busca em árvore.

Outro fato observado foi que as rotas do AG, apesar de maiores na média, não ultrapassaram 36, 270 e 750 passos para cada tamanho de mapa, enquanto o número máximo de passos usando árvore chegou a 44, 287 e 766 passos.

Durante os testes comparativos entre algoritmos genéticos e busca em árvore como implementações de IA em tarefas de navegação autônoma, pôde-se observar que a busca apresentou uma velocidade de processamento muito maior. Além disso, em grande parte dos cenários, foi encontrada uma rota mais curta, porém o AG limitado pela quantidade de alelos obteve rotas com passos sempre abaixo de 36, 270 e 750.

Pôde-se observar também que a rota gerada pela busca em árvore constituiu resposta mais objetiva, ou seja, com base no resultado da busca, o agente percorria um trajeto mais curto, enquanto a rota encontrada pelo AG fornecia trajetos com alguns passos desnecessários, resultado da geração aleatória da população inicial. Talvez seja possível produzir uma população inicial menos dependente dessa aleatoriedade para sanar tal deficiência.

De acordo com as análises, o sistema de busca em árvore para a elaboração da rota nesses cenários apresentou resultados melhores.

Para trabalhos futuros, pretende-se criar uma implementação híbrida, em que o AG possa ser utilizado para otimização da rota encontrada pela busca.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| -9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 | 0 | -1 | 0 | -1 | 0 |
| 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | -1 | 0 | -1 | -1 | 0 | -1 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 | 0 | 0 | -1 | 0 | -1 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | -1 |
| 0 | 0 | 0 | 0 | -1 | -1 | -1 | 0 | -1 | 0 |
| 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | 0 | -1 | 0 | -1 | 0 | -1 | 0 | 0 | 9 |

**Figura 14: Matriz utilizada para comparação dos algoritmos**
Fonte: Os autores.

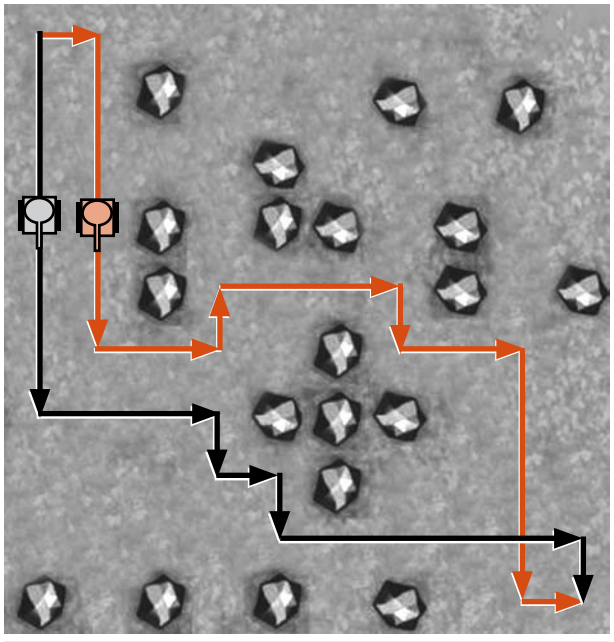

**Figura 15: Rotas geradas pelo AG (laranja) e pela busca em árvore (amarelo)**
Fonte: Os autores.

## 9 Discussão

Para a representação das seqüências de movimento no cromossomo, seria possível a utilização de valores binários ou decimais. Utilizou-se a primeira situação pela facilidade durante a operação de *crossover* e mutação, pois valores decimais envolveriam mais cálculos, o que causaria impacto negativo no desempenho do algoritmo.

A geração da população inicial foi realizada de maneira totalmente aleatória, sem preocupação com a validade dessa rota, ou seja, foram gerados caminhos, que seriam utilizados pelo algoritmo, mesmo que conduzissem o agente ao ambiente externo ao cenário. O AG conseguiu solucionar o problema com a implementação de controles na criação da população inicial, para que as rotas criadas possuíssem caminhos que mantivessem o agente sempre nas dimensões do ambiente.

Para a codificação do operador de *crossover*, foi utilizada, inicialmente, uma rotina com ponto de corte único e fixo, sempre recombinando os cromossomos a partir de sua metade. Durante alguns testes, levantou-se a possibilidade da utilização de um ponto de corte variável (Figura 16), definido da seguinte forma: nas primeiras gerações (5%), o ponto de corte situava-se em um alelo do início de número 10. Nas seguintes, ele avançava pelo cromossomo em saltos de duas posições, para não perder um movimento válido, recombinando o par de *bit*s representativo.

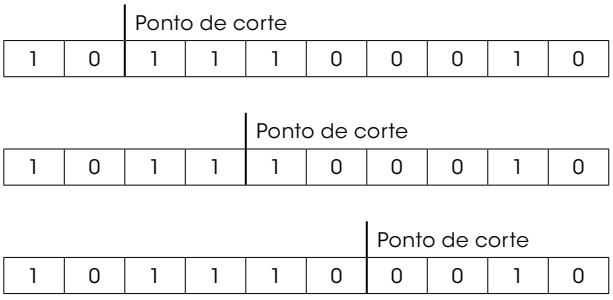


**Figura 16: Mudança do ponto de corte durante execução do AG**
Fonte: Os autores.

Na definição dos parâmetros do AG, principalmente o tamanho da população, o tamanho do cromossomo e a taxa de mutação foram melhorados durante a execução dos testes, confor-

me demonstrado na Tabela 1. Com esses testes, ficou determinado que o cromossomo deveria ter uma quantidade de alelos capaz de armazenar um número de passos de 30% a 35% da quantidade total de casas disponíveis, fornecendo, assim, uma resposta aceitável, ou seja, que não fosse um caminho demasiadamente longo, e também reduzisse a quantidade de cálculos para validar cada indivíduo das populações.

Ao mesmo tempo, a população foi fixada em 80 indivíduos com o intuito de melhorar o desempenho, pois um aumento desse valor não proporcionou melhora significativa no desempenho do AG nem na qualidade da resposta. Isso foi demonstrado na Tabela 3.

**Tabela 3: Quantidade de alelos do cromossomo em relação ao tamanho do cenário**

| Tamanho do Mapa | Tamanho do Cromossomo |
|---|---|
| 10 x 10 | 70 |
| 30 x 30 | 540 |
| 50 x 50 | 1500 |

Fonte: Os autores.

Já a taxa de mutação foi definida em 1,5 % para preservar as soluções encontradas.

Foi implementado um elitismo em que os melhores cromossomos de uma geração – em torno de 3% – são preservados e repassados à população seguinte, maximizando, assim, o aproveitamento de uma possível solução.

As rotas encontradas durante os testes apresentaram pequeno nível de erro, ou seja, alguns passos realizados incorretamente, como no exemplo da Figura 8, na qual o agente poderia ter realizado um trajeto com dois movimentos a menos.

Esse comportamento é observado porque, mesmo com esse erro, o cromossomo escolhido possuía o maior *fitness* dentro de todos os outros da população. Isso mostra que até o cromossomo mais adaptado ao problema pode ainda não sê-lo perfeitamente.

No acompanhamento dos resultados das baterias de testes, observou-se que, para a implementação do AG, a tarefa mais difícil foi elaborar uma função de avaliação que conseguisse analisar, de maneira satisfatória, o problema estudado.

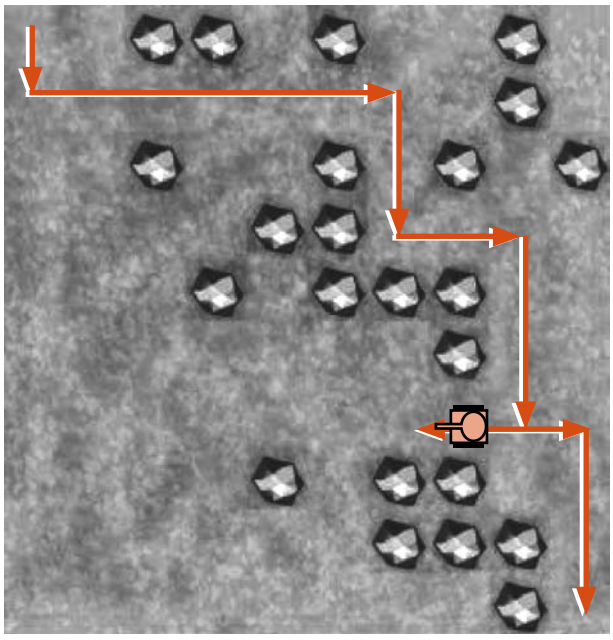

**Figura 17: Exemplo de rota encontrada**
Fonte: Os autores.

A função de avaliação é expressa na equação 5, na qual d(p, q) é a função que calcula a distância entre a posição atual e o destino definido. Essa função foi testada de diversas formas, como pôde ser observado nas equações 2, 3 e 4.

Considera-se E uma variável que o algoritmo define em 1, se o movimento executado é válido; ou 0, caso seja inválido.

$$f(x) = \left(\frac{1000}{d(p,q)} \cdot E\right) - 100 \qquad (5)$$

Essa fórmula é aplicada a cada par de alelos do cromossomo e, ao final de todos eles, o acu-

mulado dessa função, f(x), é somado ao *fitness* do cromossomo.

Dessa maneira, o *fitness* do cromossomo é acrescentado proporcionalmente à quantidade de passos certos e, quanto menor a distância entre a posição do agente em relação ao objetivo, maior será o resultado de f(x).

O movimento é validado a partir da verificação de que o passo a ser executado não conduz o agente nem ao exterior do cenário, nem para cima de um obstáculo, e evita que o agente, ao passar pela mesma casa em movimentos consecutivos, fique preso em algum "beco sem saída". Foi possível ainda constatar que a função responde melhor tendo valores positivos para o *fitness*.

Quando foi realizado o comparativo com as árvores de busca, observou-se a necessidade de reformular a função de avaliação, pois, com a criação de cenários maiores, a quantidade de movimentos errados cresceu drasticamente, levando a valores negativos de *fitness*. A solução encontrada foi modificar os valores da função de avaliação proporcionalmente ao tamanho do cenário. Acrescentaram-se as variáveis L e A, que representam as dimensões do ambiente (Equação 6).

$$f(x) = \left( \frac{10 \cdot L \cdot A}{d(p,q)} \cdot E \right) - 100 \cdot (L/5) \qquad (6)$$

Também foi necessário adaptar a maneira de ler e interpretar a matriz, pois alguns valores escolhidos inicialmente não eram iguais aos do algoritmo adquirido em Araújo, Librantz e Flório Filho (2006).

Os resultados apresentados levaram à escolha do sistema de busca em árvore para a elaboração da rota nesses cenários. Para trabalhos futuros, pretende-se criar uma implementação híbrida, em que o AG será utilizado para otimização da rota encontrada pela busca em árvore.

## 10 Conclusão

Neste trabalho, foi desenvolvida uma aplicação que representa uma situação comum em jogos de computador, usando algoritmos genéticos para prover a navegação autônoma de um agente. De acordo com os resultados obtidos nos testes, o AG demonstrou que pode ser uma solução viável e com um tempo de resposta satisfatório para interagir com um usuário.

Contudo, na comparação com árvores de busca, o AG apresentou desvantagens, principalmente no que diz respeito ao tempo de que o algoritmo necessita para produzir uma rota.

Atualmente, estão sendo feitos os devidos ajustes no AG, visando melhorar seu desempenho. Para trabalhos futuros, pretende-se implementar um algoritmo híbrido, utilizando um método de busca em árvore para a geração da população inicial em vez de ela ser gerada aleatoriamente.

## Agradecimentos

Agradecemos ao artista gráfico Paulo Renato Bonfanti, por ter gentilmente criado e liberado o uso das imagens utilizadas no desenvolvimento deste trabalho, sem qualquer custo para os autores.

Da mesma forma, estendemos os nossos agradecimentos ao Sr. Vagner Loiola, pelo apoio oferecido aos autores durante o processo final de redação deste artigo.

E, principalmente, agradecemos ao Prof. Ms. Eng. Sidnei Alves de Araújo, nosso orientador, pelo apoio, motivação e paciência durante este ano. Sua colaboração foi primordial para a conclusão deste trabalho.

## Autonomous navigation in electronic games using genetics algorithms

In this paper, it is explored the use of genetics algorithms in an electronic entertainment application. Based on the evolutionary computation theory (Genetics Algorithms – GAs), an algorithm was developed to solve an agent's autonomous navigation problem in a virtual scenario. Therefore, tests have been performed to show the viability of AGs for the problem resolution.

**Key words:** Artificial intelligence. Computer games. Genetics algorithms.

## Referências

ARAÚJO, S. A.; LIBRANTZ, A. F. H.; FLÓRIO FILHO, O. *Navegação autônoma de robôs: uma implementação utilizando o Kit Lego Mindstorms*. In: CONGRESSO SUL CATARINENSE DE COMPUTAÇÃO, 2., 2006. Criciúma: Sulcomp, 2006.

CARVALHO, L. A. V. *Datamining: a mineração de dados no marketing, medicina, economia, engenharia e administração*. 1. ed. São Paulo: Érica, 2001.

DEMARIA, R.; WILSON, J. L. *High score*: the illustrated history of electronic games. 2. ed. Emeryville: McGraw-Hill/Osborne. 2004.

KENT, S. L. The ultimate history of video games: from Pong to Pokémon and beyond – The story behind the craze that touched our lives and changed the world. 1. ed. NewYork: Roseville, CA: Prima, 2001.

KUSHNER, D. *Masters of doom*: how two guys created an empire and transformed pop culture. 1. ed. New York: Random House, 2003.

RUSSEL, S. J.; NORVIG, P. *Artificial intelligence*: a modern approach. 2. ed. Englewood Cliffs, New Jersey: Prentice Hall, 2002.

SHIRAI, Y. *Inteligência artificial: conceitos, técnicas e aplicações*. 1. ed. Portugal: Europa-América, 1988.

Recebido em 9 abr. 2007 / aprovado em 12 set. 2007

**Para referenciar este texto**

APPOLINARIO, V. B.; PEREIRA, T. L. Navegação autônoma em jogos eletrônicos utilizando algoritmos genéticos. *Exacta*, São Paulo, v. 5, n. 1, p. 79-92, jan./jun. 2007.