ALGORITMO GENÉTICO ADAPTATIVO PARA TRAVESSIA DE AMBIENTES HOSTIS DINÂMICOS

ADAPTATIVE GENETIC ALGORITHM FOR PATH-FINDING IN DYNAMIC HOSTILE ENVIRONMENT

Tiago Pereira Remédio*

RESUMO

Encontrar uma rota em um ambiente determinado é uma questão constante em diversos tipos de algoritmos. Alguns ambientes possuem situações agravantes como a mudança de seu estado em tempo de execução e pontos que podem causar dano ao agente explorador. Considerando estes problemas, encontrar o caminho mais seguro e mais rápido se torna um problema complexo. Este trabalho tem como objetivo desenvolver e analisar um algoritmo genético, a ser utilizado como suporte à decisão, que encontra o caminho mais seguro em um ambiente hostil cujos obstáculos aparecem no decorrer do tempo. O resultado deste trabalho é uma ferramenta que permite ao usuário, após carregar e mapear um cenário, realizar simulações variando os parâmetros do algoritmo e inserir/remover obstáculos verificando em tempo-real o comportamento dos agentes.

Palavras-chave: Sistema multi-agentes. Ambientes hostis. Busca de caminho. Algoritmo genético.

ABSTRACT

Finding a path considering a determinate environment is a constant issue in many types of algorithms. Some environments have aggravating situations such as changing their state at runtime and points that can cause damage to the explorer. Given these problems, finding the safest and fastest route becomes a complex problem. This work aims to develop and analyze a genetic algorithm, to be used as decision support, which finds the safest path in a hostile environment whose obstacles appear over time. The result of this work is a tool that allows the user, after loading and mapping a scenario, to perform simulations by varying the parameters of the algorithm and adding /removing obstacles, verifying real-time behavior of the agents.

Keywords: Multi-agent system. Hostile environment. Path finding. Genetic Algorithm

Introdução

Encontrar caminhos ótimos em espaços complexos é um problema que a computação tradicional não consegue resolver em tempo aceitável. A travessia de ambientes hostis

* Universidade Estadual Júlio de Mesquita Filho (UNESP) - Câmpus Rio Claro. ensino@tiagoremedio.com.br

dinâmicos consiste em permitir à agentes atravessar cenários que possuem obstáculos (que alteram os valores da função de aptidão caso estes agentes passem por eles) e que estes obstáculos possam ser inseridos/removidos dinamicamente, enquanto o algoritmo ainda está rodando.

Considerando este problema, o presente trabalho de pesquisa visa oferecer uma solução para se obter travessias em ambientes hostis em tempo viável e onde tais ambientes são dinâmicos, ou seja, podem oferecer novos obstáculos enquanto o algoritmo está em execução. A busca por tais caminhos é amplamente pesquisada no meio científico, conforme mostra Saeedvand (2014) e Berger (2013), sendo de muita importância, principalmente para o meio militar, a possibilidade de uma sugestão de escolha não antes percebida. Este trabalho utiliza um algoritmo genético, que é uma heurística, ou seja, um algoritmo não determinístico que encontra uma solução, com um erro aceitável quando comparada ao resultado ótimo, em tempo aceitável.

1 Fundamentação Teórica

O campo da computação natural visa fazer um paralelo entre as mais diversas formas e estruturas biológicas e a computação. Observando a natureza é possível verificar como que alguns problemas são solucionados e, com base em certos pontos, extrapolar métodos e algoritmos computacionais que consigam solucionar problemas que a computação tradicional não consegue. Como por exemplo existem algoritmos, conforme proposto por Castro (2007), simulando colônias de formigas, o funcionamento de colmeias de abelhas e enxame de partículas.

Para que tal campo seja pertinente e funcional, segundo Castro (2007), a distinção de qualquer nova disciplina ou campo de investigação precisa prover soluções possíveis e modelos para problemas antigos sem solução aceitável, e também para novos problemas. Este é apenas um dos benefícios da computação natural que também apresenta uma forma de entender e interagir com a natureza; novos métodos de simulação, emulação e também de perceber fenômenos naturais propostos.

Conforme barreiras para processamento aparecem (como por exemplo energia, frequência, *dark sillicon*) é necessário pensar em um novo paradigma para a computação. Com a computação natural, é possível mudar a forma com que os problemas são tratados ao

invés de se ampliar o poder computacional dos computadores. Para que a computação natural seja realizada, Castro (2007) sugere que as seguintes ramificações sejam abordadas:

- A natureza como inspiração de novas técnicas computacionais.
- Computadores recriando fenômenos naturais.
- Computação através de materiais naturais e/ou biológicos.

1.1 Algoritmo Genético

Os algoritmos genéticos remetem a teoria de evolução de Darwin para solucionarem problemas que levariam um tempo muito grande por métodos computacionais padrões. Assim como na evolução, o algoritmo utiliza de operadores genéticos para fazer uma busca guiada e encontrar melhores soluções. Esses algoritmos têm como essência pegar duas soluções e criar uma nova, possivelmente melhor que as anteriores, fazendo, assim, uma analogia com a reprodução sexuada – onde os filhos possuem propriedades de ambos os pais. Para seu funcionamento, o algoritmo possui uma população com saídas iniciais aleatórias dentro do espaço analisado. Cada indivíduo da população é representado por um alfabeto finito.

Sempre que uma nova geração é criada a partir de suas antecessoras, cada indivíduo dessa geração deve passar por uma função de avaliação, que retornará um valor correspondente. De acordo com Norvig (2004), a função de avaliação deve se comportar de forma crescente, ou seja, retornar valores mais altos para soluções mais próximas ao objetivo. Assim, escolhe-se uma forma de selecionar os melhores indivíduos da geração proporcionalmente à saída da função de fitness (função de aptidão / avaliação) escolhida.

Para a reprodução, é necessário que sejam escolhidos dois indivíduos aleatórios. Após a escolha é chamado o operador de cruzamento (ou *crossover*). Existem diversas maneiras de realizar um crossover, dentre elas a escolha de um ponto de corte no cromossomo do indivíduo. Esta forma de crossover faz com que uma parte do cromossomo de um indivíduo seja interligada com outra parte de outro indivíduo formando um novo indivíduo filho dos indivíduos anteriores. A população inicial gerada aleatoriamente possui uma grande diversidade, que se perde a medida que o algoritmo evolui e os indivíduos tendem a convergir para um estado onde os cromossomos dos indivíduos possuem valores próximos para a função de ativação (NORVIG, 2004). Outro operador genético é o operador de mutação que

modifica algum gene do cromossomo do indivíduo segundo uma regra pré-estabelecida. Ela permite uma variabilidade no decorrer das gerações para os indivíduos, já que possui uma probabilidade de ocorrência e um efeito independente de outros genes ou indivíduos.

Outra questão importante dos algoritmos genéticos é a representação do problema. A escolha da forma de representação impacta diretamente na construção dos operadores e também na solução do problema. Em essência, as vantagens de se utilizar algoritmos genéticos provêm da característica de exploração do espaço de busca de forma paralela, ou seja, cada indivíduo é uma possível solução que será evoluída pelo algoritmo.

1.2 Jogos como Ferramenta de Desenvolvimento

Os jogos eletrônicos e *engines* são interessantes para o desenvolvimento de soluções para problemas computacionais, pois contam com motores gráficos, sonoros e físicos pré-implementados, que permitem que o foco do desenvolvimento seja o problema em questão e não seus meios de representação e visualização.

De acordo com Novak (2010), jogos já foram utilizados por diversas agências governamentais dos Estados Unidos, com o intuito de recrutamento e treinamento. Esses jogos costumam ser simuladores de ambientes e situações adversas que o usuário precisa ter reflexos rápidos e preparação para superá-las. Pilotos militares e da NASA já passaram por estes simuladores para treinarem em novos veículos e também saberem se adaptar a mudanças atmosféricas. Um dos maiores títulos relacionados com recrutamento militar, o *America's Army*[1], fez tanto sucesso em seu lançamento que os servidores de download não conseguiram suprir a demanda de conexões. Ainda segundo Novak (2010), as empresas também usam jogos para reforçar as habilidades de liderança e gestão dos funcionários.

A possibilidade de interagir com o usuário, mascarando o intuito da aplicação com algo divertido e instigante, permite obter informações e resultados de/para os usuários de uma forma mais dinâmica e recompensadora. Assim, Novak (2010) diz que jogos podem melhorar o desempenho de certas profissões conforme treinam as habilidades físicas e mentais dos usuários. Um estudo do *Beth Israel Medical Center de Boston* e do *National Institute on Media and the Family* constatou que cirurgiões que costumavam jogar por três

[1] https://www.americasarmy.com/

horas semanais cometiam 37% menos erros e eram 27% mais rápidos que os médicos que não jogavam.

Dessa forma, é possível concluir que jogos possuem um papel relevante em questões críticas como treinamento militares e preparação de profissionais de diversas profissões. Essa questão foi a principal motivação para o desenvolvimento deste trabalho, já que poderá ser utilizado para esta finalidade, bem como na construção de jogos digitais.

2 Desenvolvimento

A busca do melhor caminho é um desafio que atrai pesquisadores do mundo inteiro, e suas soluções consideram diversos fatores e restrições. Em ambientes militares, os agentes normalmente interagem com ambientes dinâmicos e hostis.

Dessa forma, para atingir objetivos e encontrar resultados eles precisam de cooperação. Conforme proposto por Saeedvand (2014), em um ambiente dinâmico, são tantos os fatores no comportamento dos agentes (como visibilidade, movimentação, comunicação) que isto se torna um problema NP-Completo, justificando, portanto, a utilização de uma heurística para solução. Na maioria dos casos, os agentes também não estão cientes da qualidade do caminho, sendo necessário atravessá-lo para verificar.

Com o intuito de desenvolver uma ferramenta de auxílio para decisão sobre caminhos disponíveis, a solução encontrada foi criar um software sobre uma *engine* de jogos que possibilitaria ao usuário ver em tempo-real a suposta melhor escolha e as possíveis consequências destas escolhas.

A ideia base consiste em mapear um espaço físico e prover *waypoints* onde o agente poderia passar até seu destino. A necessidade de se criar *waypoints* deve-se ao fato que a menor distância entre dois pontos é sempre uma linha reta, então o caminho traçado por agentes deste tipo seria muito previsível. Em um sistema mapeado com *n waypoints* é possível escolher caminhos diferentes e também otimizar a escolha destes caminhos baseando-se em funções de aptidão (*fitness*) diferentes.

Assim, decidiu-se por um algoritmo genético para escolher qual a melhor sequência de *waypoints* que um agente deveria tomar para chegar ao seu destino. A representação escolhida para os cromossomos do algoritmo genético foi a representação binária. Dessa forma, uma fileira de *waypoints* precisa ser uma potência de 2 (dois) – necessário para se

criar o cromossomo – e podem existir quantas fileiras forem necessárias. Cada fileira será concatenada formando o cromossomo final.

Tabela 1 – Esquema de mapeamento do cenário

Fileira	**Comprimento**	***Bits* na *Bitstring***
[A B C D E F G H]	8	3
[A' B' C' D']	4	2
[A'' B'' C'' D'' E'' F'' G'' H'']	8	3
[A''' B''']	2	1
[A'''' B'''' C'''' D'''']	4	2

No exemplo de mapeamento da Tabela 1, existem cinco fileiras disponíveis, onde obrigatoriamente um único ponto de cada fileira precisa ser escolhido sequencialmente até o final. O cromossomo final, neste caso, é composto por onze bits e os cálculos do algoritmo serão realizados sobre esta cadeia. Um problema pode surgir onde o agente pode ir de um extremo da fileira anterior para o oposto da próxima (no exemplo, de A para D'), mas as sucessivas gerações do algoritmo combinada com a função de avaliação levando em conta a distância percorrida conseguem solucioná-lo.

Uma forma de se obrigar um agente a passar por um espaço físico no cenário é criar uma fileira mínima, de comprimento 2 (1 bit no cromossomo) e, no mapeamento, colocar os dois *waypoints* sobrepostos. A cada geração, os agentes podem ter seus *waypoints* escolhidos (a rota final) alterados internamente pelo algoritmo, considerando a taxa de crossover e a taxa de mutação presentes.

2.1 Ferramenta

A ferramenta desenvolvida para o trabalho foi um jogo sobre a *engine* Unity3D[2] onde é possível carregar um cenário 3D e, sobre este, mapear inúmeros *waypoints* (seguindo o padrão de fileiras) que representam as possibilidades de destinos temporários dos personagens 3D (agentes). Toda a programação do sistema foi feita utilizando a linguagem C#.

[2] https://unity3d.com/

Cada fileira precisa ser percorrida em sequência (sequência esta estipulada pelo desenvolvedor da ferramenta – para cada cenário deve-se realizar este processo). A simulação da trajetória do personagem entre os *waypoints* escolhidos é dada por uma ferramenta interna da *engine*, chamada de *navmesh*[3] (onde seu algoritmo de movimentação é uma variação do A*[4]).

Dinamicamente, após o cenário ter sido mapeado, o usuário pode iniciar as simulações e, para cada simulação, escolher os parâmetros de funcionamento do algoritmo genético, sendo eles: taxa de crossover, taxa de mutação, número de elementos na população e quantidade máxima de gerações. Internamente, a característica de elitismo (operador genético que sempre mantém, ao decorrer das gerações, o melhor indivíduo da geração antiga na posição do pior indivíduo da geração futura) sempre é utilizada.

A interface também permite ao usuário adicionar ou remover inimigos (obstáculos dinâmicos) no decorrer da simulação – estes obstáculos irão alterar o resultado da função de aptidão do personagem se o mesmo estiver dentro de seu alcance (marcado por um círculo em vermelho).

Na parte da simulação é possível ativar a câmera dos personagens, mostrando uma visão em terceira pessoa do trajeto escolhido – isto permite ao usuário avaliar se realmente aquele é um bom caminho ou se algum obstáculo existe e que não foi considerado pelo algoritmo.

2.2 Aplicação

As Figuras 1 e 2 mostram telas do cenário carregado na ferramenta desenvolvida. Na figura 2, além do cenário ter sido todo mapeado, as fileiras estão bem definidas e expostas em diferentes cores (apenas para ajuda visual, o que define os limites das fileiras é um objeto da classe de mapeamento). Assim, neste exemplo desenvolvido, um agente precisa sair do círculo verde inicial, passar por um *waypoint* laranja, amarelo, azul, roxo, rosa, verde-claro, vermelho, verde, e chegar no alvo vermelho final, precisamente nesta ordem.

[3] https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html
[4] http://web.mit.edu/eranki/www/tutorials/search/

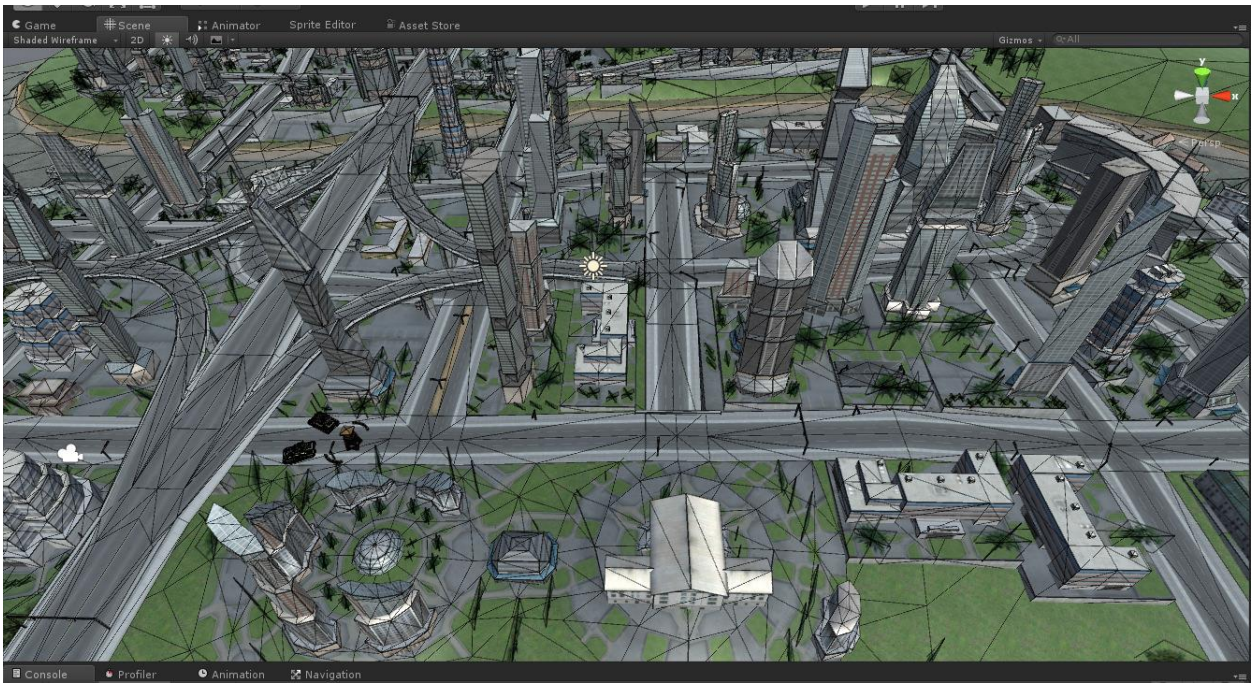

Figura 1 – Cenário carregado na *engine.*

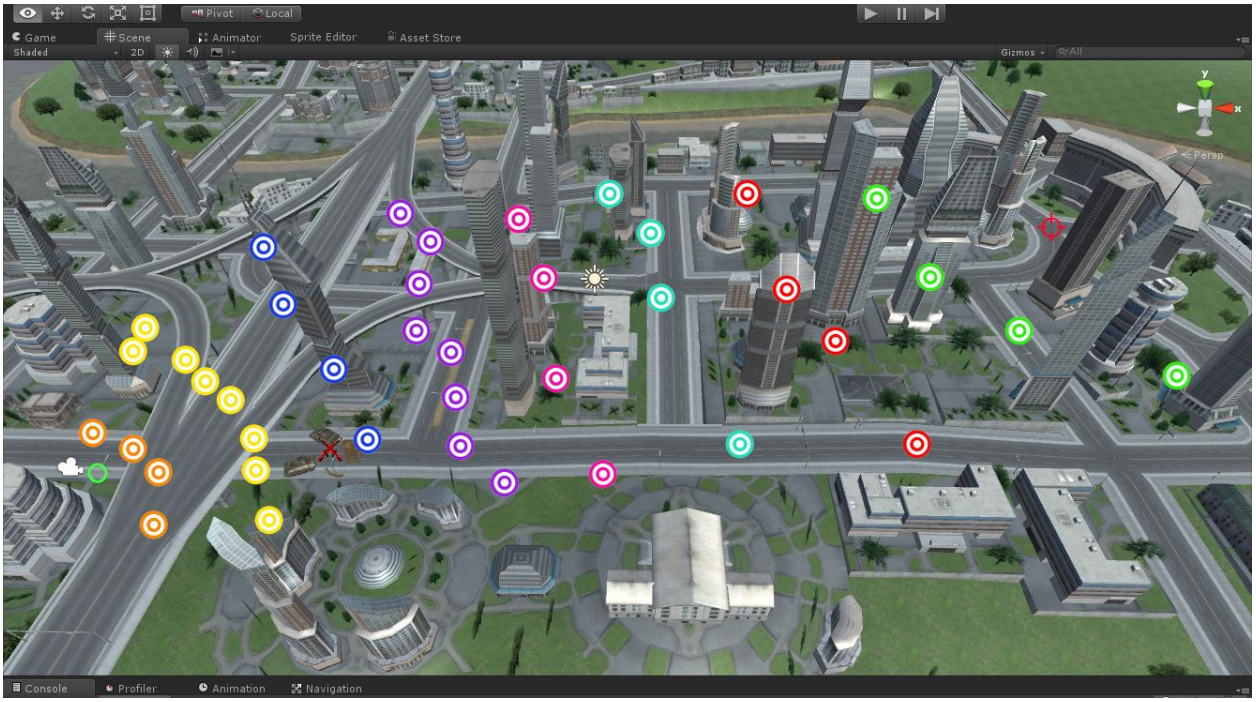

Figura 2 – Cenário com *waypoints* mapeados.

A figura 3 detalha como é dentro da *engine*, o modo de inserção dos *waypoints* que os agentes terão conhecimento nas travessias.

O funcionamento do algoritmo genético prevê que após um número *n* de gerações, os agentes tendem a seguir as melhores rotas (que possuíram melhor resultado na função de aptidão) e toda variação futura se dará pela taxa de mutação escolhida. Se o tempo disponível ao usuário for alto, então essa taxa de mutação e o crossover serão suficientes para compensar os obstáculos dinâmicos introduzidos após o algoritmo convergir em um mesmo valor de genes para os cromossomos. Mas, como o cromossomo pode ser muito grande (número de sequências de *waypoints* pode ser alta) então uma mudança na base do algoritmo foi feita.

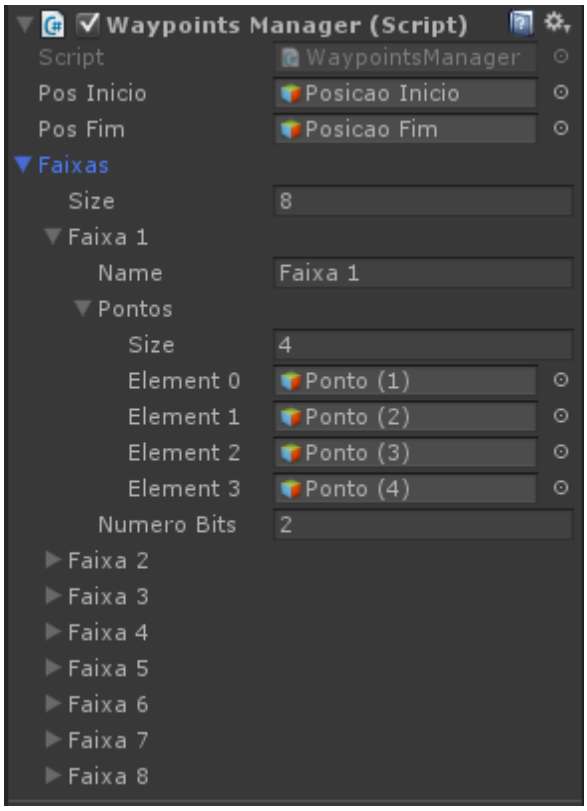

Figura 3 – Gerenciador de *waypoints*, mostra como separar cada fileira, especificar o comprimento da fileira e relacionar os pontos do cenário para cada ponto da fileira em questão. Também permite especificar a posição inicial e final do cenário.

Quando um novo obstáculo é adicionado, a parte do cromossomo referente à sequência de *waypoints* mais próxima ao obstáculo é calculada novamente na próxima geração. Ou seja, os *waypoints* daquela fileira perdem seu peso e passam a ter a mesma chance de serem escolhidos – dessa forma, após a geração acabar, um novo cromossomo pode ser escolhido desviando o recém criado obstáculo, e a taxa de mutação continua a valer para as seguintes gerações (a taxa de mutação é presente em todos os bits do cromossomo).

Durante a visualização também é possível aumentar a velocidade com que o tempo passa (útil quando o cenário representado é muito grande) – não é possível fazer todos os cálculos do algoritmo internamente, pois a função de avaliação necessita de dados como o tempo despendido (obstáculos causam um dano pré-definido por segundo quando o agente está em sua área de atuação), a distância e a vida final do agente. As Figuras 4, 5 e 6 mostram detalhes do funcionamento inicial da ferramenta, bem como as opções de inicialização e visualização do andamento.

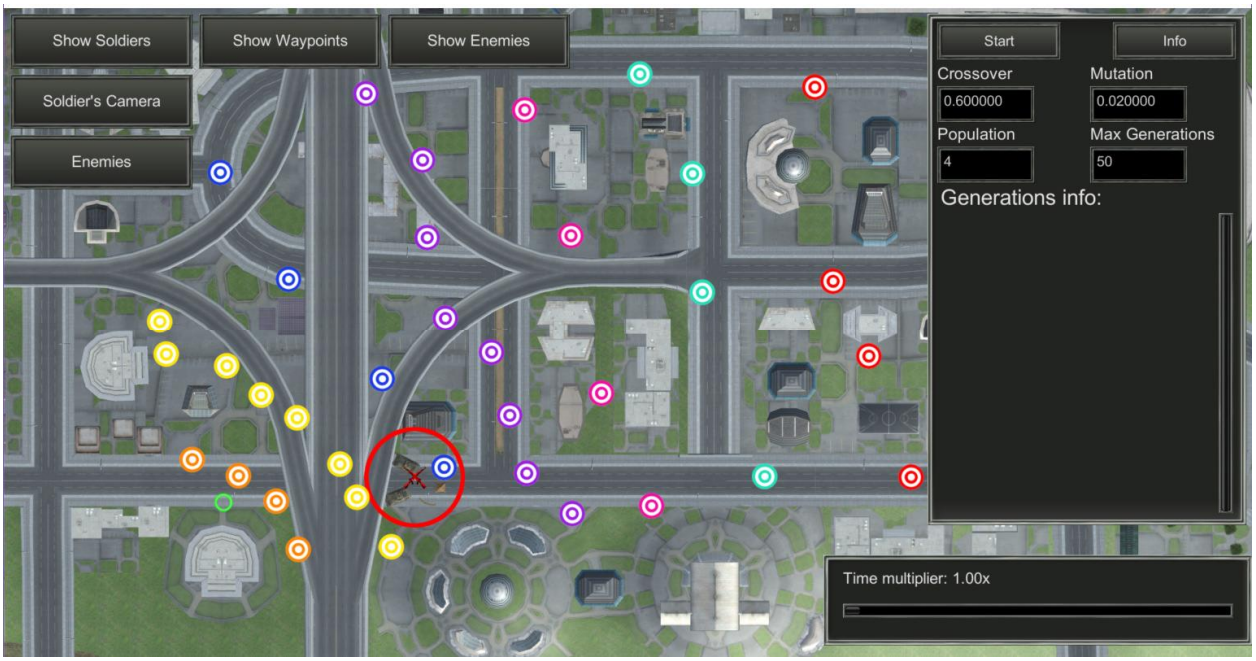

Figura 4 – Tela inicial com o menu de configurações do algoritmo à direita e opções de funcionamento do sistema à esquerda.

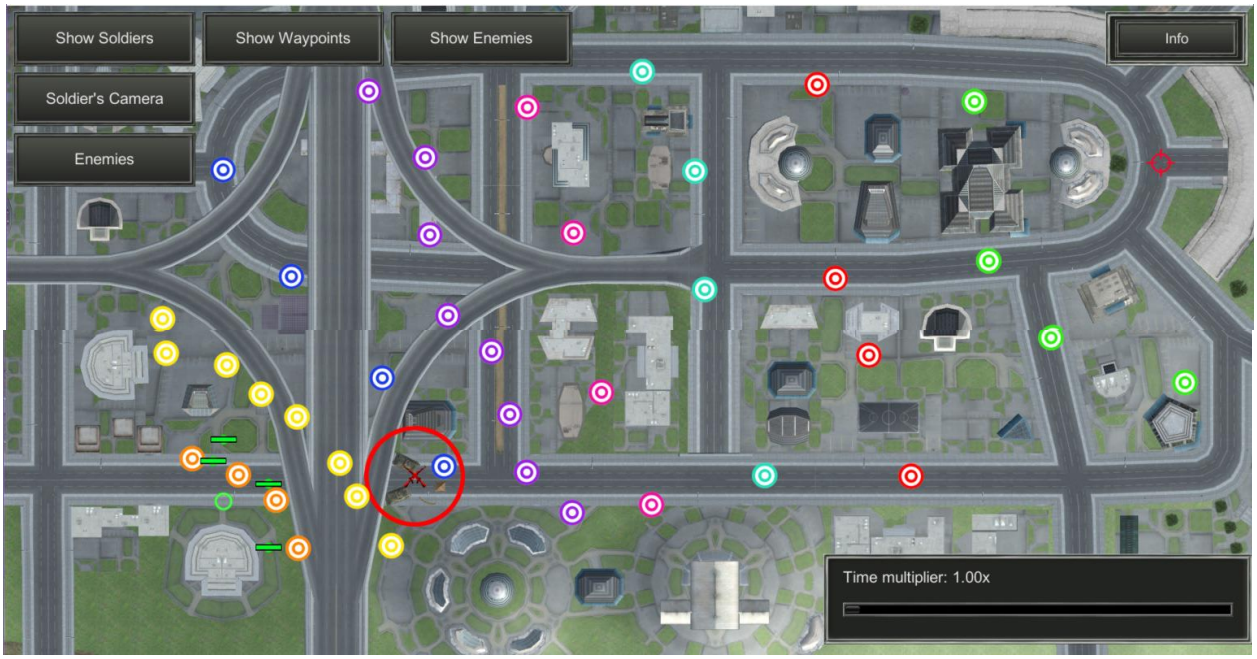

Figura 5 – Personagens (agentes) movendo através do cenário após a ferramenta ser sido inicializada

Figura 6 – Visão do soldado percorrendo o cenário

Na figura 7 é possível verificar um caso crítico onde, apesar do algoritmo tratar um caminho como bom e não alterar os valores da função de aptidão do soldado (por este não estar dentro da distância estipulada), o usuário percebe que o agente está próximo demais de

um obstáculo, com linha visual limpa, gerando uma situação onde não é interessante a escolha deste caminho.

Figura 7 – Visão do soldado ao encontrar um obstáculo – escolha ruim de *waypoint*, que será tratada em futuras gerações

Na Figura 8 é possível visualizar agentes em um cenário onde obstáculos diversos foram adicionados e as barras de vida de cada agente (barra de vida representando uma parte integrante da função de aptidão) são alteradas conforme o tempo que se passa dentro da área de atuação do obstáculo.

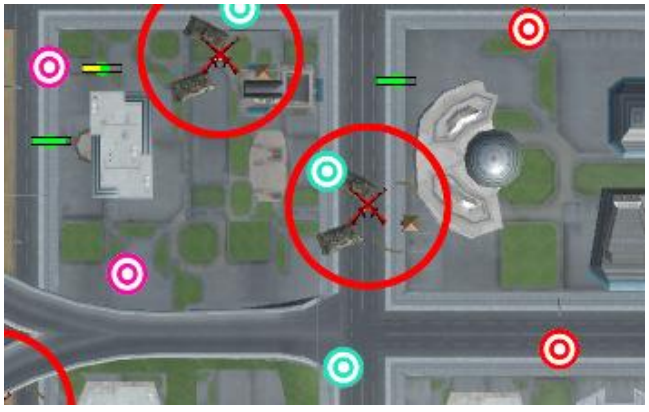

Figura 8 – Personagens percorrendo o cenário. Novos obstáculos foram adicionados e é possível verificar personagens com a vida diminuída (barras de vida sobre seus referenciais)

A Figura 9 mostra a forma de resposta de algumas iterações do algoritmo. A janela de informações lateral informa os detalhes das rotas traçadas bem como a melhor rota até o momento.

Figura 9 – Tela de informações após algumas gerações do algoritmo, mostrando todos os cromossomos escolhidos, qual teve a melhor saída na função de avaliação, e os dois parâmetros de avaliação desta saída.

Neste sistema, a função de avaliação pensada leva em conta a trajetória final do personagem (trajetória simulada pela ferramenta) e também a "vida" no destino. A vida do personagem pode ser pensada analogamente ao número de soldados que conseguiram chegar ao destino ou à quantidade de dano que um veículo sofreu. Se um personagem morrer (vida zerar no percurso) implica em uma função de avaliação nula – o caminho escolhido é extremamente ruim.

Os testes e execuções do programa serão realizados em três etapas, a primeira sem nenhum obstáculo, a segunda já com um número de obstáculos iniciais definidos e a terceira com obstáculos sendo introduzidos após um número *n* de gerações. Também serão avaliadas variações nas configurações do algoritmo genético, como número de indivíduos na população, taxa de crossover (sempre com um ponto de crossover) e taxa de mutação. A saída analisada será o número de gerações que o algoritmo leva para convergir de acordo com as configurações de execução, bem como os valores de distância e vida da melhor saída encontrada.

A função de avaliação considerou como parâmetro principal a vida do personagem que é inversamente proporcional à distância percorrida. A fórmula inicial para teste foi a mais simples, cuja saída é a razão da vida pela distância. Assim que as melhores configurações internas do algoritmo forem levantadas, uma outra função de aptidão, que enfatiza mais a vida que a distância, será testada. O algoritmo foi avaliado durante cem gerações, número este suficiente para a convergência dentro do pequeno cenário mapeado para testes.

Sendo assim, os parâmetros avaliados foram:

- Obstáculos
    - Sem obstáculo
    - 10 obstáculos iniciais
    - 10 obstáculos inseridos após 50 gerações do algoritmo.
- Taxa de *crossover*
    - 35%
    - 60%
    - 85%
- Taxa de mutação
    - 2%
    - 5%
- Quantidade de Indivíduos
    - 4 por geração
    - 8 por geração
- Cálculo do Fitness
    - F(x) = vida / distância (1)
    - F(x) = vida³ / distância (2)

3 Resultados

Os resultados mostrados na Tabela 2 a seguir foram feitos através das configurações previamente mencionadas e utilizando um total de cem gerações. Esses resultados foram sobre a função de aptidão (1), pois sem obstáculos a vida se torna constante e apenas a distância é levada em conta (inversamente proporcional à saída).

Analisando os valores da tabela 2, nota-se que taxas pequenas de crossover não convergem em valores de distância satisfatórios, já valores muito altos tendem a seguir uma única resposta muito rapidamente (em uma geração inicial), o que faz com que não se explore tanto o espaço. Assim, uma taxa média de crossover é ideal, pois faz com que o algoritmo atinja bons resultados e consiga sempre ir convergindo e melhorando seus resultados no decorrer das gerações. A figura 10 concentra todas as configurações e saídas levantadas.

Tabela 2 – Resultados sem Obstáculos

Quantidade indivíduos	**Taxa de *crossover***	**Taxa de mutação**	**Melhor distância (Qual geração)**
4	35%	2%	675,33 (31)
8	35%	2%	568,12 (30)
4	35%	5%	541,31 (73)
8	35%	5%	578,76 (89)
4	60%	2%	550,55 (47)
8	60%	2%	567,07 (77)
4	60%	5%	559,09 (46)
8	60%	5%	532,14 (97)
4	85%	2%	629,49 (22)
8	85%	2%	575,61 (63)
4	85%	5%	634,96 (98)
8	85%	5%	559,09 (58)
8	60%	50%	547,00 (75)

A taxa de mutação é responsável pela verificação aleatória de caminhos, uma vez que o algoritmo tenha convergido em alguma rota. Ela é importante para se explorar caminhos uma vez já esquecidos. É interessante manter essa taxa pequena. Na última linha da tabela apresentou-se um caso com uma taxa alta, mas taxas altas assim transformam o algoritmo em algo aleatório, e não garante convergência, seria como uma busca por tentativa e erro – em algoritmos de espaço de busca pequenos pode funcionar, mas não garante o bom funcionamento do algoritmo genético.

A quantidade de indivíduos representa quantas rotas diferentes é possível buscar paralelamente a cada geração. Normalmente quanto maior este número, melhor. No entanto, números altos gastam mais memória e torna-se mais complexo a comunicação entre agentes.

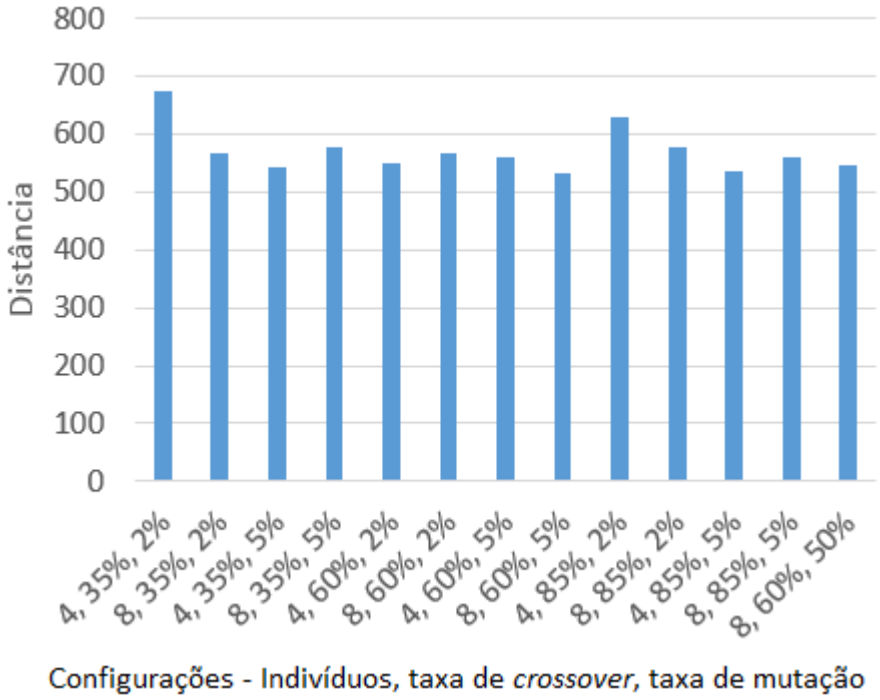

Figura 10 – Distâncias para ambientes sem obstáculos

Com os resultados previamente levantados, as configurações base escolhidas para os testes variando os obstáculos foram: taxa de crossover de *60*%, taxa de mutação de *5*% e uma população de 8 indivíduos. Além disso, com as configurações agora estipuladas como padrão, deixou-se o algoritmo rodando por *1000* gerações para se verificar a melhor saída de distância e o resultado foi de *506,34*, na geração *967*. A Tabela *3* mostra os resultados para as diferentes funções de aptidão utilizando as configurações padrões citadas anteriormente.

Tabela 3 – Resultados com Obstáculos

Função de aptidão	Obstáculos	Distância / Vida (Qual geração)
(1)	10 iniciais	597.65 / 95.8 (63)
(2)	10 iniciais	546.01 / 100.0 (31)
(1)	10 após 50 gerações	500.91 / 100.0 (47) – A 964.21 / 22.31 (51) – B 653.57 / 100.0 (96) – C
(2)	10 após 50 gerações	602.52 / 100.0 (42) – A 1053.57 / 78.14 (51) – B 509.32 / 100.0 (68) – C

Na primeira execução, um ponto interessante é que a fórmula escolhida preferiu a saída *597.65 / 95.8* (aptidão de *0.16*) ao invés de *648.58 / 100.0* (aptidão de *0.15*), sendo ambas gerações encontradas no decorrer da mesma execução do algoritmo. Esse é um exemplo do porque a função (2) é mais interessante de ser utilizada. Ao se utilizar a função (2), com os mesmos resultados obtidos, se obteria os valores de aptidão *1471* para a primeira saída e *1541* para a segunda (que, agora, é escolhida como a melhor saída).

Assim, de acordo com os dados obtidos, a escolha da função (2) se deve por ela dar mais peso às variações de vida (opção de projeto, poderia ser escolhido enfatizar mais a distância). Ainda na Tabela 3, as linhas que contém obstáculos inseridos após o início possuem três saídas diferentes, “A”, “B” e “C”. A saída “A” é a melhor saída quando ainda não haviam obstáculos, a saída “B” representa a primeira geração após os obstáculos terem sido inseridos, e a saída “C” mostra a segunda convergência do algoritmo até o final de sua execução.

Considerações finais

Avaliando o programa desenvolvido, percebe-se que o algoritmo genético consegue atingir bons resultados conforme gerações vão passando, mesmo com obstáculos sendo adicionados no decorrer do tempo. O tempo para convergência em um bom resultado depende de quão complexo é o mapeamento do cenário – que ocasiona um aumento no número de genes do cromossomo de análise. Quanto maior essa cadeia, mais difícil de se encontrar o melhor caminho.

A taxa de mutação é muito importante neste caso, pois como o sistema trabalha com o operador de elitismo, a melhor saída sempre será passada para a próxima geração, e a mutação contribuirá para uma procura mais ampla no espaço de análise. Diversos métodos e abordagens existem para encontrar um bom caminho dinamicamente e a escolha deste precisa levar em conta tantas variáveis que sua complexidade é muito alta para avaliar todos os caminhos possíveis e encontrar o melhor caminho absoluto.

A utilização de algoritmos genéticos para a solução do problema se mostrou eficiente, porque consegue prover caminhos diferentes e que tendem ao melhor caminho (de acordo com a função de aptidão). A ferramenta utilizada mostrou robustez suficiente para ser considerada em simuladores, fornecendo novos caminhos e alternativas que, em momentos de alto nível de hostilidades, podem passar despercebidos. A eficiência da resposta do algoritmo depende do bom mapeamento do cenário e da configuração correta dos parâmetros dispostos, onde estes podem levar à resultados errôneos ou simplesmente não convergir (por se tratar de um algoritmo estocástico, ele não garante que chegará, ou sempre chegará, na melhor solução possível e nem se a solução que chegou já é a melhor – visto que na maioria dos problemas não se conhece a melhor solução).

Referências

BERGER, J.; BARKAOUI, M. Genetic Algorithm for In-Theatre Military Logistics Search-and-Delivery Path Planning. **World Academy of Science, Engineering and Technology**, v. 7, n. 11, p. 1475-1483, 2013.

CASTRO, L. N. **Fundamentals of Natural Computing**. Boca Raton, EUA: Chapman & Hall/CRC (Taylor & Francis Group), 2007.

CEKMEZ, U. et al. UAV Path Planning with Parallel Genetic Algorithms on CUDA Architecture. In: WORLD CONGRESS ON ENGINEERING, **Proceedings...** 2014. V. 1.

KANG, M.; JHA, M. K.; KARRI, G. Determination of Robot Drop Location for Military Path Planning Using GIS Application. Recent Advances in Computer Engineering and Applications. In: CEA'10, WSEAS INTERNATIONAL CONFERENCE ON COMPUTER ENGINNERING AND APPLICATION, 4., **Proceedings...**
Cambridge, USA, 2010. p. 194-200.

LEENEN, L.; TERLUNEN, A.; LE ROUX, H. A Constraint Programming Solution for the Military Unit Path Finding Problem. Mobile Intelligent Autonomous Systems, 2012. **Proceedings**

NOVAK, J. **Desenvolvimento de games**. 2. ed. São Paulo: Cengage Learning, 2010.

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**. 2. ed. Rio de Janeiro: Editora Campus, 2004.

SAEEDVAND, S; RAZAVI, S. N.; ANSAROUDI, F. Path-finding in Multi- Agent, unexplorer And Dynamic Military Environment Using Genetic Algorithm. International **Journal of Computer Networks and Communications Security**, v. 2, n. 9, p. 285-291, set. 2014.