Aplicação do algoritmo A* na Unreal Engine

Matheus Fortes dos Santos de Oliveira
Jogos Digitais
UFN
Santa Maria, Brasil
matheusoda88@gmail.com

Guilherme Chagas Kurtz
Jogos Digitais
UFN
Santa Maria, Brasil
guilhermechagaskurtz@gmail.com

Abstract— This work introduces the pathfinding algorithm, focusing on the A* algorithm, describes its details and shows an example of how its path finding technique is used in a game within a 3D environment on Unreal Engine, a game engine from Epic Games . The algorithm will be applied to an entity positioned within this game, having to move through this environment, finding the best possible route, and its operation will be compared with Unreal's navigation system.

Resumo— Este trabalho introduz os algoritmos de *pathfinding*, com foco no algoritmo A*, descreve seus detalhes e mostra um exemplo de como sua técnica de busca de caminhos pode ser utilizada em um jogo dentro de um ambiente 3D na Unreal Engine, motor de jogos da Epic Games. O algoritmo foi aplicado em uma entidade posicionada dentro deste jogo, tendo que se movimentar através desse ambiente, encontrando a melhor rota possível, e seu funcionamento foi comparado com o do sistema de navegação nativo da Unreal.

Keywords: pathfinding, A* algorithm, unreal engine, navmesh

I. Introdução

Pathfinding é um ponto importante a se destacar quando o assunto é inteligência artificial relacionada a jogos, ainda que também tenha importância em outras áreas, como na robótica, tráfego de rotas telefônicas, planejamento de trânsito e solucionamento de labirintos. O crescimento da indústria de jogos impulsionou as pesquisas nessa área da inteligência artificial, e até os dias atuais são pesquisados aprimoramentos para os algoritmos de *pathfinding.*

Pathfinding se refere ao ato de traçar um mapa percorrendo o melhor caminho entre dois pontos, e é uma parte indispensável em diversos domínios, incluindo planejamento de rotas em robôs, sistema de posicionamento global, inteligência artificial e jogos. Nos jogos, *pathfinding* é necessário para que um agente navegue por um caminho até chegar em seu destino [1]. Além disso, essa técnica é uma entre as aplicações mais comuns no meio de pesquisas sobre inteligência artificial [2]. A busca de caminhos é um problema que deve ser resolvido em tempo real, frequentemente sob restrições como limite de memória e CPU, portanto encontrar o caminho ideal se torna um desafio quando é necessário um balanceamento entre precisão e velocidade no momento da busca. Como encontrar um caminho otimizado de forma mais eficiente ainda é uma área de estudos atuais [3].

Com base nesse balanceamento necessário, foi observado o comportamento do algoritmo que foi desenvolvido, e comparado com o sistema de navegação nativo da Unreal Engine em diferentes situações.

II. Referencial teórico

Nesta seção serão apresentados temas relacionados à inteligência artificial nos jogos e aos algoritmos de *pathfinding*.

A. Inteligência Artificial

No livro *Artificial Intelligence: A Modern Approach (1995)*, Russel diz que inteligência artificial é a criação de programas de computador que simulam agir e pensar como seres humanos, assim como agir e pensar racionalmente. Essa definição abrange tanto a visão cognitiva quanto comportamental da inteligência (exigindo simulações de ações e pensamentos). Também inclui, mas separa, noções racionalidade e "humanidade".

A inteligência artificial é a ciência de criar máquinas inteligentes, principalmente programas de computadores inteligentes. Está relacionada à tarefa semelhante de usar computadores para entender a inteligência humana, mas IA não precisa limitar-se a métodos biologicamente observáveis [4].

2.1. Inteligência artificial em jogos

Geralmente, jogos não necessitam de tanta abrangência em relação à noções de inteligência artificial, a inteligência artificial para Jogos é especialmente um código no jogo que faz com que o elemento controlado pelo computador pareça fazer decisões inteligentes quando o jogo tem múltiplas escolhas para uma situação dada, resultando em comportamentos que são relevantes, efetivos ou úteis [5].

O termo *Game AI* (ou inteligência artificial para Jogos) pode ser usado com um rótulo bem amplo, frequentemente usado de forma vaga para se referir a diversas áreas existentes em um jogo: como *pathfinding* (busca de caminhos), controle de personagens, interface do usuário e até mesmo todo sistema de animação. Até certo ponto, esses elementos têm algo a adicionar ao

mundo da IA, mas eles não são o sistema de inteligência artificial primário do jogo [5].

2.2. *Pathfinding*

Fora do ambiente de jogos, pode-se perceber o uso de algoritmos de *Pathfinding* em rotas do Google Maps, serviço que ao receber um pedido de rota dos usuários, devolve o melhor caminho entre os dois pontos de referências, levando em consideração o comprimento das estradas, congestionamento de tráfego, entre outros fatores.

Pathfinding está relacionado ao problema do caminho mínimo, dentro da teoria dos grafos, que examina como identificar o caminho que melhor atende a alguns critérios (mais curto, mais barato, mais rápido, etc) entre dois pontos em uma grande rede [6].

Pathfinding é um elemento de inteligência artificial crítico em diversos gêneros de jogos modernos. Jogos de estratégia em tempo real (RTS) particularmente apresentam grande desafio para os algoritmos de busca de caminho [7]. Portanto, para cada caso em que haja necessidade de utilizar um algoritmo de pathfinder, é necessário estudar o melhor algoritmo para ser usado neste caso.

Algoritmos de *pathfinding*

Algoritmos de *pathfinding* são algoritmos que têm como finalidade solucionar um problema de busca de caminho, para isso ele define um custo mínimo de movimento, podendo ser relacionado à distância, e partir deste custo mínimo e define o melhor caminho, podendo ser o caminho mais curto, o mais barato, ou o que não esteja indisponível, dependendo do contexto de sua aplicação. Existem variados tipos de algoritmos de *pathfinding*, entre eles os mais famosos são:

- Algoritmo de Bellman-Ford — Resolve o problema para grafos com um vértice-fonte e arestas que podem ter pesos negativos.
- Algoritmo A* — um algoritmo heurístico que calcula o caminho mínimo com um vértice-fonte.
- Algoritmo de Floyd-Warshall — Determina a distância entre todos os pares de vértices de um grafo.
- Algoritmo de Johnson — Determina a distância entre todos os pares de vértices de um grafo, pode ser mais veloz que o algoritmo de Floyd-Warshall em grafos esparsos.
- Algoritmo Viterbi — Resolve o menor problema de caminho estocástico com um peso probabilístico adicional em cada nó.
- Algoritmo de Dijkstra — Resolve o problema com um vértice-fonte em grafos cujas arestas tenham peso maior ou igual a zero. Sem reduzir o desempenho, este algoritmo é capaz de determinar o caminho mínimo, partindo de um vértice de início *V* para *todos* os outros vértices do grafo [8].

Entre esses algoritmos o mais utilizado nos jogos é o A*, que por sua vez é uma variação do algoritmo de Dijkstra.

Algoritmo de Dijkstra

O algoritmo de Dijkstra é um algoritmo de busca de caminhos que tem como prioridade encontrar o caminho mais curto entre dois pontos, estando diretamente conectados ou não, para isso é selecionado o ponto inicial e examinado o custo de movimento para todas as posições ligadas a ele, para a partir disso selecionar alguma posição com base no menor custo e repetir esse processo até encontrar o caminho com menor custo até o ponto final. Dijkstra é um algoritmo usado atualmente na conexão de estradas, como por exemplo os serviços de rotas Google Maps.

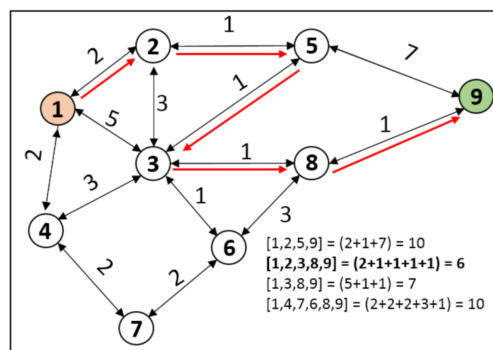

Figura 1. Nessa imagem o algoritmo explora todos os caminhos possíveis, até chegar a conclusão do melhor caminho (1,2,3,8,9) [9].

Na imagem acima o custo de cada movimento é representado pelo número junto a aresta que conecta os pontos. Para dar início a rota, é selecionado o ponto 1 e então é analisado tanto o custo do ponto 1 aos seus vizinhos, quanto dos vizinhos do ponto 1 aos seus respectivos vizinhos. Após escolher o caminho com menor custo de movimento, a posição final será alcançada com o melhor caminho de acordo com Dijkstra.

Algoritmo A*

A* é um algoritmo de busca genérico que pode ser utilizado para solucionar diversos problemas, e a busca de caminhos é apenas um deles [10]. Para o *pathfinding*, o algoritmo A* examina repetidamente a posição inexplorada mais promissora no seu campo de visão. Quando a posição é explorada, o algoritmo é finalizado caso essa posição seja a sua finalidade; caso contrário ele observa todos os vizinhos da sua posição para mais

exploração. A* é provavelmente o algoritmo de busca de caminhos mais popular na inteligência artificial de jogos. Ele pode ser facilmente observado em diversos tipos de jogos com alguma IA básica [11].

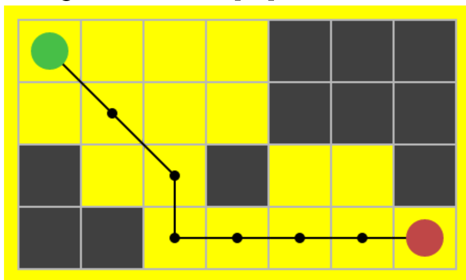

Figura 2. Caminho que seria escolhido pelo algoritmo, considerando os obstáculos presentes no *grid* [12].

O algoritmo A* é um variação do algoritmo de Dijkstra, eles diferem entre si apenas no quesito da busca heurística presente no A*. Enquanto o algoritmo de Dijkstra escaneia todos caminhos possíveis em busca da rota com menor custo, o algoritmo A* prioriza as posições que são mais relevantes, tendo preferência em posições com a menor distância em relação a posição final.

***Pathfinding* em jogos**

Pathfinding tem sido investigado nos jogos por muitos anos. É o problema mais popular, e ao mesmo tempo frustrante inteligência artificial na indústria de jogos. Diversos algoritmos de busca, como o Algoritmo de Dijkstra, que foi o primeiro algoritmo de busca e de profundidade, foram criados para resolver pequenos problemas de busca do menor caminho até o aparecimento do algoritmo A* como uma provável solução otimizada para a busca de caminhos [7].

Dentro dos jogos é possível notar o uso de *pathfinding* mais comumente, é o principal sistema de movimento em jogos de estratégia em tempo real (RTS), que ao mover algum personagem o jogador basta decidir a sua posição final, que a IA encontrará a melhor rota enquanto desvia de obstáculos. Em outros tipos de jogos a maioria das entidades não controladas pelo jogador também terão algum tipo de *pathfinding* em sua implementação.

Um bom exemplo para essa aplicação de *pathfinding* está nas franquias Age of Empires e Total War, que por serem RTS, tem como mecanismo de movimentação principal das entidades o próprio mouse, necessitando apenas da sua posição final para aplicar o movimento.

Figura 3. Exemplo da aplicação do *pathfinding* no jogo Age of Empires.

Na figura 3 o traçado vermelho indica o melhor trajeto para o destino da entidade focada, considerando os obstáculos presentes no caminho. A entidade foco é representada pelo trabalhador destacado com verde, enquanto seu destino é representado pelo círculo vermelho.

2.3 Unreal Engine

A Unreal Engine é um motor de jogos criado pela Epic Games e utilizado pela primeira vez em 1998 no jogo de tiro em primeira pessoa Unreal. Esse motor vem se popularizando pela sua praticidade e poder de processamento desde então. O desenvolvimento na área da programação da Unreal Engine pode ser feito de duas maneiras: através de scripts utilizando a linguagem C++, ou através da programação visual por meio de *blueprints.*

Na criação do projeto existem *presets* disponíveis para a simplificação do desenvolvimento, permitindo que o desenvolvedor foque no diferencial do jogo. Entre os *presets* disponíveis estão: FPS, terceira pessoa, *Top-down*, entre outros.

Além disso, o motor possui diversas bibliotecas que facilitam sua utilização, sendo uma o sistema de navegação denominado *Nav Mesh*, que define áreas do terreno como andáveis ou não.

2.4 Metodologia FDD

A metodologia FDD (*Feature Driven Development*) é uma metodologia de desenvolvimento, concebido originalmente por Jeff de Luca, surgiu em Singapura, entre os anos de 1997 e 1999 com base no método de análise orientada a objetos de Peter Coad, que é uma metodologia de análise orientada a objetos, que tem como foco o estudo de problemas com fundamentos baseados em conceitos palpáveis, e também se baseia em processos interativos.

Esse modelo consiste em desenvolvimento por funcionalidades. Para a funcionalidade básica do modelo FDD alguns processos devem ser desenvolvidos. Primeiramente é necessário o desenvolvimento de um modelo abrangente (Análise orientada por objetos), cujo a

principal finalidade é estudar e analisar o que será desenvolvido.

A segunda etapa é a construção de uma lista de funcionalidades, onde será listado tudo que será desenvolvido, porém separando em etapas com o critério de funcionalidades. Em seguida será feito o planejamento do desenvolvimento com base na lista de funcionalidades [13].

Após isso, a partir da lista de funcionalidades, se inicia a terceira etapa do FDD, na qual é feita uma sequência do desenvolvimento do projeto com base na lista de funcionalidades e suas prioridades. A quarta etapa é executada uma vez para cada funcionalidade, nessa etapa é detalhado e estudado, se necessário, cada funcionalidade existente no projeto.

A quinta etapa é onde se inicia o desenvolvimento das funcionalidades, com base no modelo abrangente e no detalhamento das etapas anteriores.

III. Trabalhos Relacionados

Nessa seção serão apresentados três trabalhos relacionados com o tema e após isso serão dadas as considerações a respeito dos mesmos.

3.1. "A Pathfinding Algorithm in Real-time Strategy Game based on Unity3D"

Este trabalho introduz o algoritmo A*, descreve seus detalhes e mostra um exemplo real de como sua técnica de busca de caminhos é usada em jogos 3D no motor de jogos Unity. Nesse exemplo existem dois scripts essenciais para o seu funcionamento, um para representar o ambiente que usa pathfinding e o outro que deve ser atrelado aos objetos que irão processar os possíveis caminhos. Foi desenvolvido um protótipo na Unity onde uma entidade nomeada "homem virtual" encontra o melhor caminho para diferentes esferas posicionadas em um cenário de labirinto.

3.2. "A comprehensive study on pathfinding techniques for robotics and videos games"

Esta pesquisa fornece uma visão geral de algoritmos e técnicas de *pathfinding* populares com base em problemas de grafos. Focando nos desenvolvimentos e melhorias recentes nas técnicas existentes e examinamos seu impacto na robótica e na indústria de videogames. O objetivo deste artigo é fornecer aos pesquisadores um conhecimento aprofundado sobre o progresso feito nos últimos 10 anos neste campo, resumir as principais técnicas e descrever seus resultados.

Foram classificados diferentes tipos algoritmos de *pathfinding* com base em uma pesquisa de ambiente 2D / 3D. Algoritmos utilizados recentemente foram estudados em diferentes topologias, assim como os diferentes tipos de *grids*, que sendo regulares ou irregulares, foram estudados para extrair o melhor resultado desses algoritmos. A principal preocupação dos pesquisadores é fornecer um caminho ideal em relação ao tempo de processamento e à sobrecarga de memória.

Em relação aos *grids*, a conclusão que os pesquisadores chegaram é de que para a busca de caminho para apenas uma entidade a melhor representação de um terreno plano é dada por um *grid* regular quadriculado enquanto em um terreno irregular a melhor representação é dada por um *grid* irregular triangular. Já para *pathfinding* de múltiplas entidades podem ser utilizadas duas abordagens: desacoplada e acoplada. Na abordagem desacoplada os caminhos são planejados para cada entidade separadamente, nessa abordagem a efetividade dos algoritmos pode não ser garantida por causa do número de execuções do mesmo. Já na abordagem acoplada a busca é feita somente uma vez, como se fosse em apenas um agente visto de uma perspectiva maior, o grande problema dessa abordagem é a colisão entre os agentes.

3.3. "Direction Based Heuristic For Pathfinding In Video Games"

Encontrar os caminhos mais curtos entre dois pontos em um mapa é um tópico importante na pesquisa em matemática e algoritmos. De acordo com os autores, na atual indústria de jogos, o *pathfinding* tem seus próprios requisitos: um deles é a importância de não priorizar o caminho matematicamente melhor, a prioridade é encontrar sempre um caminho curto o suficiente de forma eficiente. Esse artigo revisa as atuais técnicas de *pathfinding* e propõe um novo método de *pathfinding* eficiente para video games, que deve gerar um caminho de maior qualidade usando menos tempo e memória do que outras soluções existentes, por meio da otimização de técnicas de busca, mapeamento efetivo do terreno e o aperfeiçoamento da heurística.

Para a otimização dos algoritmos de *pathfinding,* foi utilizada uma abordagem que usa a direção do destino como heurística em mapas baseados em *grid*, para o funcionamento desta heurística foram utilizadas quatro letras, cada uma simbolizando uma direção diferente, é necessário que ao iniciar a busca pelo melhor caminho também seja calculada a direção do ponto final em relação ao ponto inicial. Essa informação é utilizada para determinar qual é o melhor caminho (ou o caminho mais promissor) de maneira mais rápida.

Foi feita uma comparação dessa técnica com outros algoritmos de busca, entre eles o algoritmo de busca em largura e o A*. A comparação feita com o A* indica que a abordagem do algoritmo criado converge para a direção do destino como o A*, a comparação com os outros algoritmos, demonstra que a nova abordagem também converge para a posição destino, com tempo de

processamento mais rápido que do algoritmo de busca em largura.

3.4 Considerações sobre os trabalhos relacionados

O primeiro trabalho citado proporcionou conhecimentos a respeito do algoritmo A*, e como é dada a sua aplicação em um jogo, dentro de um ambiente 3D simples.

O segundo trabalho demonstrou a diversidade de técnicas existentes para a busca de caminhos, as comparando e, no final, foi apontada a melhor técnica para diferentes cenários, de acordo com os autores.

Já o terceiro trabalho mostra uma abordagem de como pode ser feita a otimização do algoritmo A* para ambientes de escala maior, utilizando a direção do ponto final como parâmetro antes de fazer a busca, após o desenvolvimento foi feita uma comparação com algoritmo A* não adaptado e com o de busca em largura.

IV. Proposta

O objetivo deste trabalho é demonstrar como é a aplicação do algoritmo A* em jogos 3D utilizando o sistema de programação visual da Unreal Engine, denominado *blueprints*. Para isso foi implementado um algoritmo A* e aplicado em um protótipo de uma fase para demonstrar o seu funcionamento, em seguida, na mesma fase, será utilizado o sistema de navegação da Unreal Engine para a mesma finalidade do algoritmo desenvolvido, para que seja feita uma comparação de sua eficiência. Para a realização dessa comparação, foram analisados diferentes cenários e apontados dados como: velocidade da busca, processamento e precisão da busca, em cada um desses cenários.

A imagem acima mostra a criação de um plano para representar o chão, o *grid,* além disso também foi necessário a criação de uma *blueprint*, para a configuração do *grid*, e alguns obstáculos. Cada posição presente no *grid* foi numerada para que seja possível reconhecer as posições das entidades, obstáculos e caminhos livres.

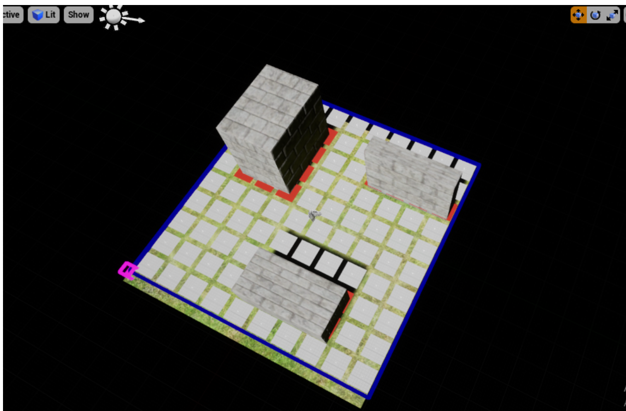

Figura 4. *Grid* com os obstáculos posicionados, e cada um deles transforma a posição do *grid* em que entra em contato em uma posição não explorável.

Com a criação do *grid* e dos obstáculos, foi necessária a criação de um ator que será o ponto inicial do processo de buscar o melhor caminho e responsável por demonstrar as funcionalidades do algoritmos, ou seja, se locomover através do melhor caminho, e assim como os obstáculos, ele deve ter sua posição reconhecida pelo grid.

Nesse momento, cada posição no *grid* precisa conter sua distância em relação à posição da entidade que deseja de mover, assim como a distância em relação à posição a que ele deseja chegar, logo, é necessário estabelecer um custo de movimento padrão entre uma posição do *grid* e suas posições vizinhas, pois com um custo definido é possível definir uma distância entre diferentes posições.

Feita a configuração do ambiente, é necessário que o ator de fato encontre e percorra pelo melhor caminho. Para encontrar o melhor caminho, será feito o cálculo $F(n) = G(n)+H(n)$, sendo H a distância para a posição final, G a distância para a posição atual e F a soma dos dois últimos valores. Com isso é selecionada a posição com menor F e, se a nova posição não for a posição destino, o algoritmo recomeça a busca nos novos vizinhos, caso contrário o ator iniciará o movimento.

A implementação do algoritmo A* e dos elementos que demonstram as suas funcionalidades na Unreal Engine serão feitas através de *blueprints*, o sistema de programação visual desta *engine.*

Feita a implementação desses elementos, será necessária a criação de uma fase para comparar suas funcionalidades com o sistema de navegação da Unreal Engine, para essa comparação serão considerados os parâmetros: tempo de processamento, desempenho e precisão.

Por último, o ambiente 3D desenvolvido será modificado com o intuito de buscar diferentes resultados nas comparações, diferentes cenários serão criados, e para cada cenário, será apontado o método de *pathfinding* recomendado.

V. Metodologia

Nesta seção será demonstrado como a metodologia FDD é aplicada neste trabalho.

5.1. Desenvolvimento de um modelo abrangente

Para o desenvolvimento deste trabalho foi utilizada a metodologia FDD (*Feature Driven Development*), que consiste no desenvolvimento por funcionalidades. Na primeira etapa, que consiste em desenvolver um modelo abrangente, foi elaborado um fluxograma, conforme mostra a figura abaixo.

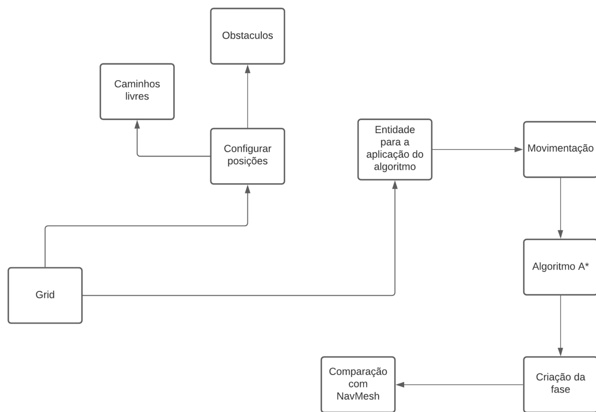

Figura 5. Fluxograma demonstrando a interação entre as funcionalidades.

O fluxograma demonstra a interação que cada funcionalidade tem com as outras, tendo dois pilares essenciais para a demonstração do algoritmo, sendo eles a configuração do *grid* e da entidade que se moverá através do A*.

5.2. Criação de uma lista de funcionalidades

A lista de funcionalidades a ser desenvolvida será dividida entre requisitos funcionais e não-funcionais.

Requisitos funcionais

- Criação da entidade para a aplicação do algoritmo;
- Aplicação do A* na entidade;
- Movimentação da entidade;
- Criação de um fase;
- Comparação do algoritmo com o sistema de navegação da Unreal Engine;
- Adaptação do cenário para novas comparações.

Requisitos não-funcionais

-Criação do *grid;*
-Configuração das posições no *grid;*
-Configurar os caminhos livres e obstáculos.

5.3. Planejamento por funcionalidade

Nessa etapa foi feita uma sequência de desenvolvimento, as funcionalidades a serem desenvolvidas são organizadas em ordem cronológica, tendo em vista que algumas funcionalidades necessitam de outras funcionalidades do projeto para o seu funcionamento. A tabela contendo o tempo de implementação previsto para cada funcionalidade pode ser encontrada no Anexo A.

5.4. Detalhamento por funcionalidade

Essa etapa foi responsável por detalhar as funcionalidades existentes no projeto.

Criação do *grid:* A primeira funcionalidade é puramente visual. O *grid* irá identificar e controlar cada posição presente no mesmo, para que seja possível a configuração das outras entidades, porém nesse primeiro momento o *grid* ainda não terá nenhuma função relevante para o projeto

Configurar as posições no *grid:* Foi nesse momento que o *grid* passou a ter funcionalidade, foram atribuídos valores para cada posição, tanto horizontal, quanto verticalmente. Com isso pode ser feita a identificação das posições presentes no *grid* para a configuração das demais funcionalidades.

Configurar os caminhos livres e obstáculos: Criação de obstáculos que interagem com o *grid*, transformando as posições em que está colidindo em caminhos que não podem ser explorados. Além dos valores para identificar as posições, também é necessário definir cada posição como explorável ou não, as posições não exploráveis serão consideradas obstáculos da cena.

Criação da entidade para a aplicação do algoritmo: Essa etapa consiste em criar uma entidade para aplicar os objetivos do trabalho.

Aplicação do A*: Criada a entidade, foi feita a aplicação do algoritmo de busca A* sobre ela, juntamente com a criação de uma interface visual para demonstrar o seu funcionamento.

Movimentação: Após isso foi feita a aplicação de um sistema de movimento para essa entidade, onde será demonstrado o funcionamento do algoritmo de busca, através da movimentação.

Criação da fase: Em seguida foi criada uma fase nesse ambiente 3D para a demonstração do algoritmo e seu funcionamento em um exemplo real.

Comparação com o sistema de navegação da Unreal Engine: Por último é comparado o funcionamento do algoritmo desenvolvido com o sistema de navegação da Unreal Engine, apontando as vantagens e desvantagens desse sistema, tendo como base para essa comparação os parâmetros apontados na proposta.

VI. Desenvolvimento

Para o desenvolvimento foi seguido a metodologia FDD na etapa de construção por funcionalidade, começando com a criação de uma cena vazia, e a partir dela foi realizada a configuração de um *grid* com diversas posições, que interajam com obstáculos posicionados na cena, que é feito através de duas *blueprint actors* - uma para a configuração do *grid* e o outro para os obstáculos. A *blueprint* dos obstáculos contém um *static mesh*, que é responsável pela aparência, dimensões, colisões e detecção de colisões do objeto, e uma enumeração, que indica qual o tipo do obstáculo atual.

Para a configuração do *grid* foi criada outra *blueprint actor* para representar cada casa presente nesse *grid*. Essa *blueprint* contém apenas uma *static mesh* e valores referentes à sua aparência. Além disso, também foi criada uma estrutura, que é uma variável que pode conter diversas variáveis e de diversos tipos dentro dela, que nesse primeiro momento possui valores do tipo vetor para uma posição no mundo, vetor 2D para o índice do *grid* e a enumeração referente ao obstáculo. Essa estrutura foi acessada pela *blueprint* do *grid*, onde foram criadas duas funções principais. A Figura 6 apresenta as variáveis da estrutura.

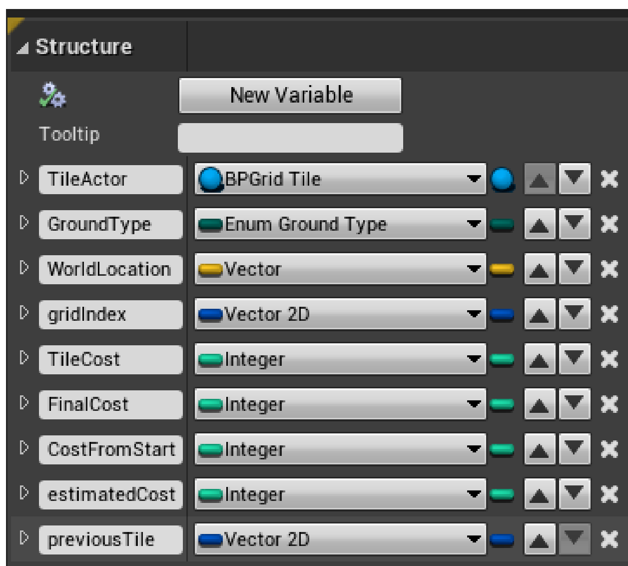

Figura 6. Estrutura de dados e suas variáveis.

Uma das funções é responsável por criar uma matriz onde cada valor presente nela representará uma posição ou "casa" no *grid*. A outra função principal dentro da *blueprint actor* do *grid* gera células interativas em cada posição presente no *grid*, sendo cada uma dessas células uma cópia do *blueprint actor* criada para representá-las. Nesse *blueprint* foi adicionada a interação com o ponteiro do mouse, permitindo que cada casa possa ser selecionada com o clique. Como mostra a Figura 7.

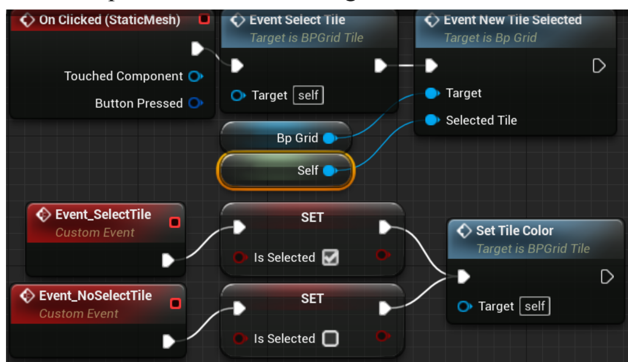

Figura 7. Eventos referentes à seleção de células do *grid*.

A próxima etapa foi adicionar um *widget* ao *blueprint* da casa que mostra uma representação dos valores referentes ao custo de movimento na superfície da casa. Em seguida é iniciada a aplicação do algoritmo A* no projeto. A Figura 8 e 9 demonstram a edição deste *widget*.

Figura 8. Demonstração do design do *widget,* que contém duas caixas de texto.

Figura 9. Conteúdo presente na caixa de texto central do *widget*, referente à variável "GetCostText".

Inicialmente foram adicionados novos valores à estrutura previamente criada, sendo quatro valores do tipo *integer* para representar os custos de movimento, sendo eles o custo padrão de cada casa, custo final, custo a partir do início e custo estimado até o alvo, além de uma variável do tipo *vector 2D* para apontar o índice do casa anterior, e outra variável referente a um objeto do tipo "casa" (*blueprint*/classe criada para representar a casa). Feito isso, foi criada uma função na *blueprint actor* do *grid* responsável por acessar o índice do *grid* na estrutura e identificar os vizinhos da casa selecionada. Em seguida foi criada uma função para calcular o custo estimado até os vizinhos encontrados. Como pode ser notado nas Figuras 10 e 11.

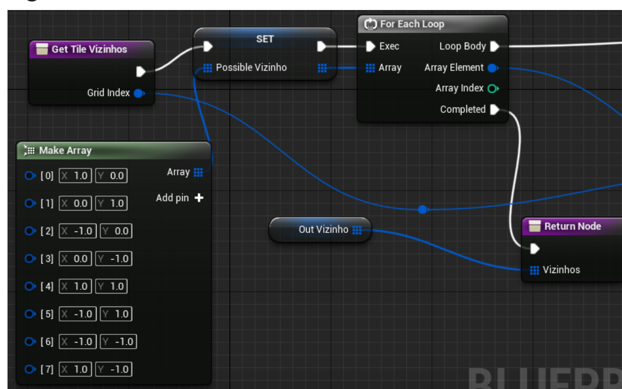

Figura 10. *For Each Loop* percorrendo as células ao redor da célula selecionada.

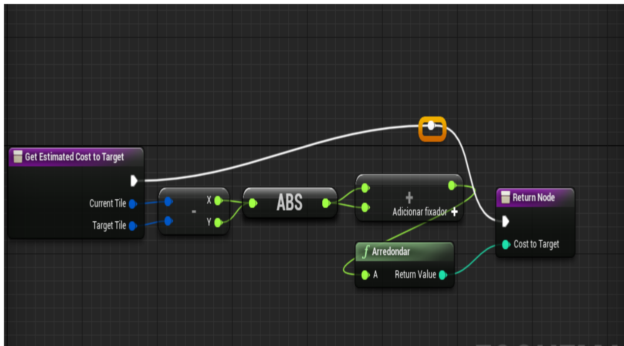

Figura 11. Função que através da subtração de dois vetores 2D e soma dos elementos do vetor 2D resultante retorna um custo estimado para o movimento entre eles.

Logo, foi criada a função para calcular a rota até qualquer casa escolhida no cenário. Para isso, a função passa como parâmetro a posição inicial selecionada através do clique e a posição alvo através do *Overlapping Cursor Event*. Com isso foi criada uma lista de vetores 2D contendo as posições promissoras e definida uma delas como a mais promissora utilizando a heurística do algoritmo A*, sendo o ponto de partida a primeira a ser adicionada a essa lista e definida mais promissora. Como é demonstrado na Figura 12 e na Figura 13.

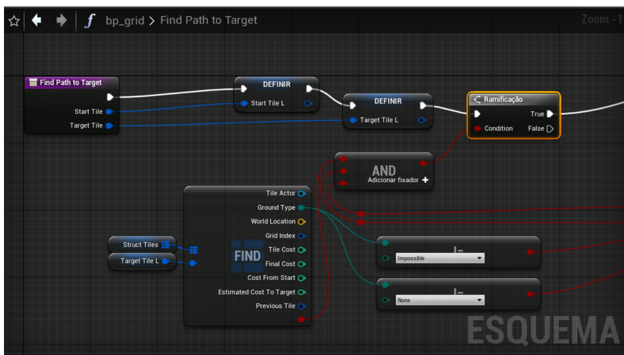

Figura 12. Trecho da função que, considerando que tanto a casa inicial quanto a alvo estão selecionadas, valida a existência de um obstáculo antes do seu andamento.

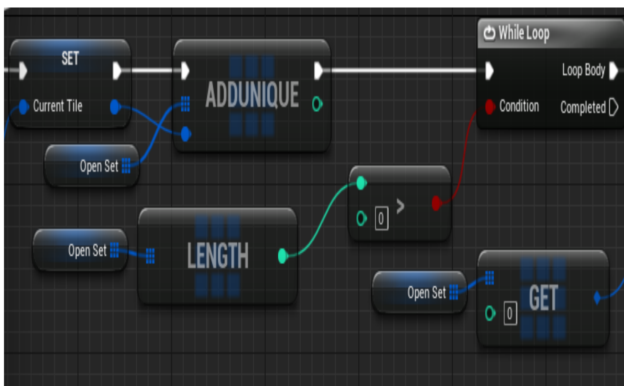

Figura 13. Trecho em que a célula selecionada é adicionada a lista e um *while loop* com a condição da existência de algo nessa lista faz com que a função entre em repetição.

Feito isso, essa lista é percorrida por um *for each loop*, comparando os custos de movimento até cada um dos elementos presentes nela, selecionando assim a posição mais promissora de fato e a transformando na casa selecionada, mesmo que em um primeiro momento a lista só possua um elemento. Como mostra Figura 14.

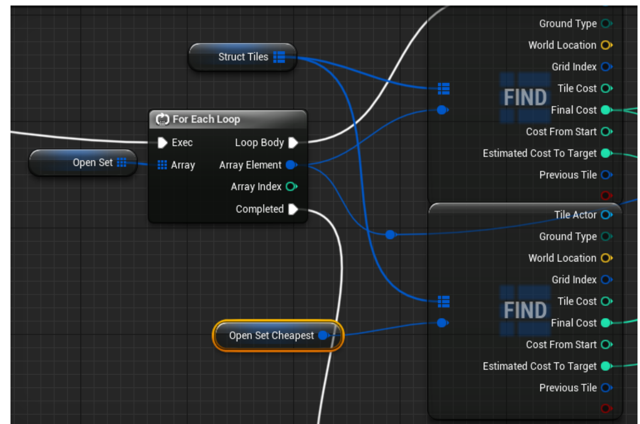

Figura 14. Trecho da *blueprint* que faz a comparação dos valores para definir a célula mais promissora, com um *loop* que ao finalizar a célula mais promissora é selecionada.

O próximo passo é buscar vizinhos da casa selecionada utilizando a função desenvolvida, percorrendo e adicionando-os na lista de promissores. Então, o algoritmo escolhe novamente a casa mais promissora dentro dessa lista, a seleciona e a adiciona a uma lista de posições já visitadas ou analisadas pelo algoritmo e repete o processo até a casa selecionada for igual a posição alvo. Demonstrado pela Figura 15. Cada casa selecionada como parte do caminho pelo algoritmo é adicionada a uma lista, contendo de forma ordenada todas as posições presentes no melhor caminho, para que essas posições sejam acessadas em na função responsável por mover o personagem e para que, através de outra função, durante a seleção de caminhos as posições presentes na lista sejam destacadas com uma cor diferente. Apresentado na Figura 16.

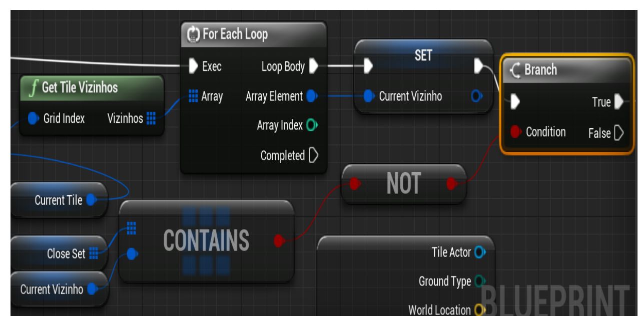

Figura 15. Trecho da função onde os vizinhos da célula selecionada são acessados e validados, para logo adiante os adicionar à lista de caminhos células promissoras.

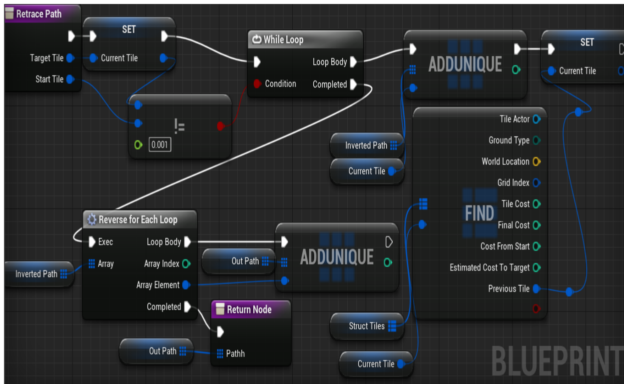

Figura 16. Função responsável por montar a lista referente ao melhor caminho escolhido pelo algoritmo.

A próxima etapa é a movimentação de alguma entidade pela rota. Para isso foi criada uma *blueprint character* para representar a unidade movimentável e um *blueprint controller* para configurar a seleção da mesma através do mouse. Em seguida, foi criada uma função para identificar a célula em que o personagem está posicionado e retornar um vetor 2D referente a essa posição.

Para a funcionalidade do movimento do personagem foi necessário adicionar um *spline component* ao *blueprint do grid*, que é acessado por um evento criado no *blueprint character*, que recebe cada posição presente na lista de posições do melhor caminho e para cada posição presente é criado um *spline point*, como demonstra a Figura 17. Após isso foi criada uma linha do tempo que obtém o comprimento do *spline* e utiliza uma função que acessa a posição e rotação no *spline* de acordo com uma distância recebida, sendo a distância o tempo na *timeline*, ou seja, o tempo zero representaria a posição inicial do *spline*, enquanto o último momento da linha do tempo sua posição final. Com isso, a linha do tempo move o personagem repetidamente para cada posição retornada pela função mencionada e o desenvolvimento do algoritmo é finalizado, demonstrado na Figura 18.

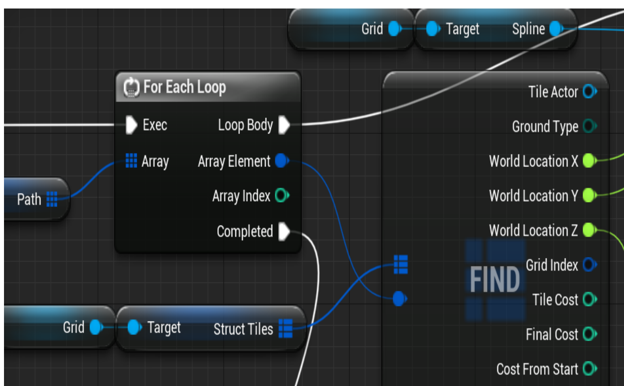

Figura 17. Momento em que a função percorre cada posição na lista do caminho selecionado, coleta suas posições do *transform* e para criar um *spline point* em cada uma delas.

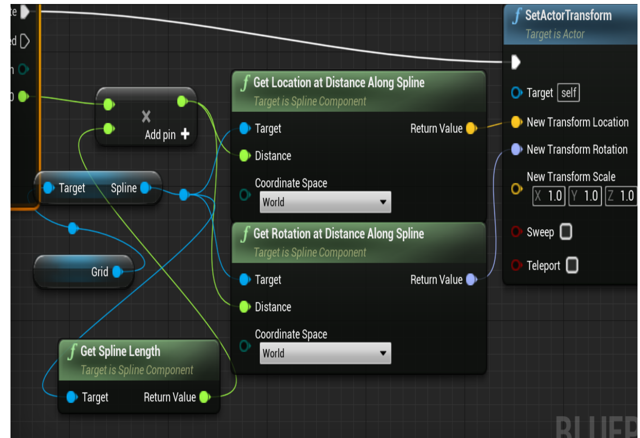

Figura 18. Momento em que a linha do tempo utiliza os *spline points* para executar o movimento do personagem.

Por fim, é criada outra *blueprint character* com uma *static mesh* para representar o personagem a ser movido pelo algoritmo nativo da Unreal Engine e uma *blueprint actor* vazia para representar a posição alvo desse personagem. Com isso, foi acessado a *blueprint* do *level* para configurar o movimento dessa entidade. O movimento funciona de forma simples, sendo feito através de um *event tick* que a todo *frame* move esse *character* em direção do seu alvo, que muda sua posição diversas vezes para que o *character* movido pelo *navmesh* da Unreal Engine percorra diversos caminhos. Sendo assim possível uma comparação entre os dois algoritmos de busca de caminhos. A Figura 19 e 20 demonstram a como é feita a configuração do movimento.

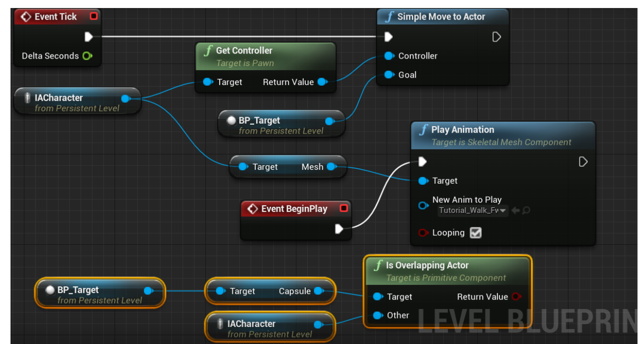

Figura 19. Trecho da *blueprint* que executa o movimento do personagem até o alvo.

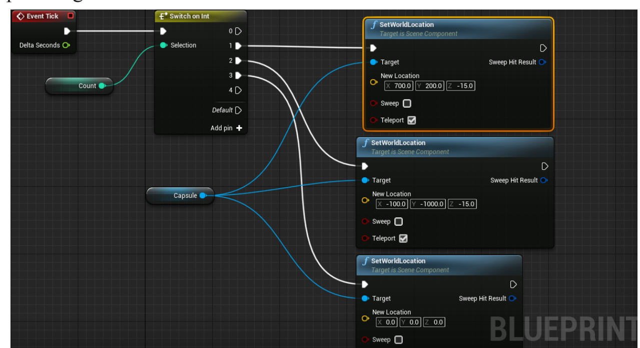

Figura 20. Trecho da *blueprint* onde é alterada a posição do alvo de acordo com o contador.

VII. Resultados

Para coletar os resultados foi criado no ambiente 3D uma representação de um labirinto, posicionando os dois *characters* em uma mesma posição inicial, sendo o movimentado pelo navmesh o primeiro a se movimentar em direção ao seu alvo. Esse labirinto pode ser modificado do como o desenvolvedor desejar. A Figura a seguir mostra uma das variações do cenário.

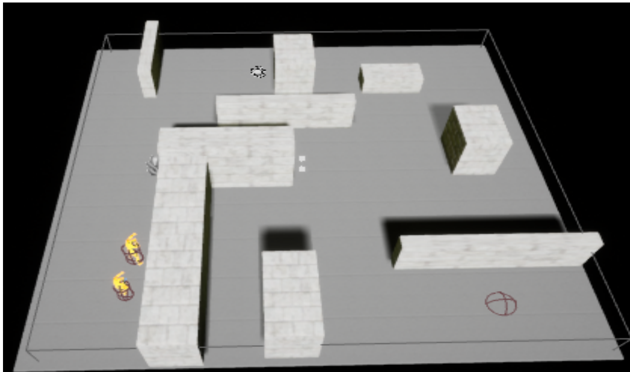

Figura 21. Segundo cenário montado.

Ao chegar ao seu destino é possível movimentar o outro personagem até o mesmo alvo através do mouse, que ao colidir com o alvo, faz com que o mesmo mude de posição, fazendo com que o personagem da *navmesh* percorra outro caminho.

Repetindo esse processo com diversas rotas e em diferentes cenários é possível obter resultados suficientes para fazer a comparação. A demonstração de como é efetuado o primeiro movimento dos personagens no primeiro cenário montado pode ser encontrado no Anexo B.

VIII. Conclusão

Em relação às rotas escolhidas, os algoritmos não apresentaram diferenças, ambos priorizam os mesmos caminhos para chegar ao destino. Já em um ponto de vista voltado para o desempenho, a *navmesh* da Unreal Engine acaba sendo viável em diversas situações, pelo fato do componente interagir diretamente com a *static mesh* para detectar obstáculos e utilizar para dados presente no *transform* para posições no cenário, enquanto o algoritmo desenvolvido necessita de um *grid* para detectar seus obstáculos, e utiliza *loops* que percorrem esse *grid* para entender suas posições.

Na grande maioria dos casos o sistema de navegação nativo da Unreal será a opção mais viável no geral, por conta da sua otimização dentro do motor de jogos e simplicidade na utilização. Porém o algoritmo desenvolvido ainda pode apresentar resultados mais satisfatórios que a *nav mesh* em abordagens de jogos de estratégia que se aproveitem da existência de um *grid.*

Referências

[1] A. N. Sabri, N. H. M. Radzi and A. A. Samah, "A study on Bee algorithm and A* algorithm for pathfinding in games," *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 2018, pp. 224-229, doi: 10.1109/ISCAIE.2018.8405474.

[2] Botea, Adi, et al. "*Pathfinding in games*." Dagstuhl Follow-Ups. Vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[3] Cui, Xiao; Shi, Hao. "DIRECTION ORIENTED PATHFINDING IN VIDEO GAMES". School of Engineering and Science, Victoria University, Melbourne, Australia. 2011.

[4] McCarthy, John. "What is Artificial Intelligence?". Computer Science Department, Stanford University. 2007.

[5] Schwab, Brian. "IA Game Engine Programming Second Edition". 2004.

[6] https://en.wikipedia.org/wiki/Pathfinding

[7] Silver, David. "Cooperative Pathfinding". Department of Computing Science, University of Alberta.

[8] https://pt.wikipedia.org/wiki/Problema_do_caminho_mínimo.

[9] https://www.researchgate.net/figure/Illustration-of-Dijkstras-algorithm_fig1_331484960.

[10] Mathew, Geethu Elizebeth.(2015) "Direction Based Heuristic For Pathfinding In Video Games". K.S.R Institute for Engineering and Technology

[11] Cui, Xiao; Shi, Hao. "A*-based Pathfinding in Modern Computer Games". School of Engineering and Science, Victoria University.

[12] https://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html

[13] Silva, F. G.; Silva, L; Hoentsch, S. C. P. "Uma análise das Metodologias Ágeis FDD e Scrum sob a Perspectiva do Modelo de Qualidade MPS.BR.".Scientia Plena VOL 6. Departamento de Computação, Universidade Federal de Sergipe, São Cristóvão. 2010.

ANEXO A - Tempo de implementação de cada funcionalidade

Funcionalidade	Tempo de implementação
Criação do *grid*	1 dia
Configurar as posições no *grid*	1 dia
Configurar os caminhos livres e obstáculos	2 dias
Criação da entidade para a aplicação do algoritmo	1 dia
Aplicação do A* na entidade	7 dias
Movimentação da entidade	7 dias
Criação de um fase	7 dias
Comparação do algoritmo com o sistema de navegação da Unreal Engine	14 dias
Adaptação do cenário (fase) em busca de diferentes resultados na comparação que será feita	14 dias

ANEXO B - Tempo de implementação de cada funcionalidade

Figura 22. Os dois personagens apresentando a mesma posição inicial.

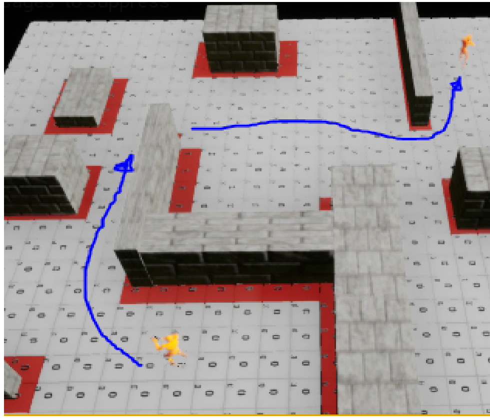
Figura 23. Personagem movimentado pela *nav mesh* se deslocando.
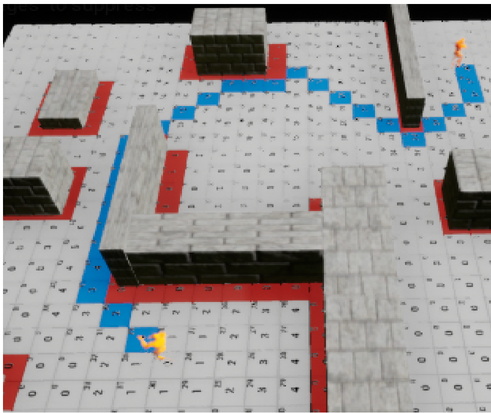
Figura 24. Personagem do algoritmo desenvolvido e posição alvo selecionados e caminho destacado no *grid*.
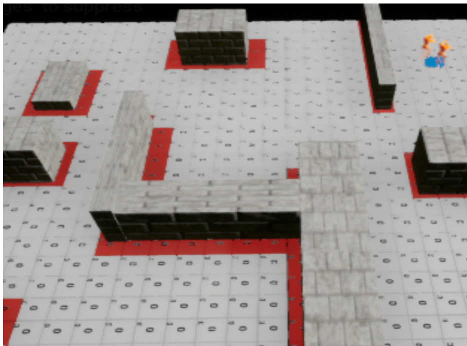
Figura 25. Personagem do algoritmo desenvolvido se deslocando pelo caminho escolhido.