Mini Metro Assistant*

Clarice Saraiva Andrade dos Santos[1]

Abstract— **O jogo Mini Metro possibilita a aplicação de diversos conceitos e problemas clássicos da Ciência da Computação e Inteligência Artificial, como o pathfinding através de buscas informada e não informada. O artigo propõe a implementação de um algoritmo Dijkstra para encontrar o menor caminho entre dois pontos em um grafo construído pelo jogador.**

I. Introdução

Problemas de busca e de rotas são alguns dos problemas mais abordados na Ciência da Computação e áreas correlatas, considerados como problemas clássicos de programação. Dentre eles, alguns dos exemplos mais citados são o problema do caixeiro viajante e o problema de roteirização. Apesar de notoriamente conhecidos, eles também podem ainda apresentar alta complexidade [1]. Os desafios para os sistemas de tráfego urbano servem como motivação para a busca de otimização nas alternativas capazes de apresentar diminuição de custos, bem como a diminuição do tempo de espera, e aumento na qualidade do serviço, considerando as restrições de recursos [2]. Abordagens para a resolução do caos urbano advém de exemplos como do caixeiro-viajante [3] e de roteirização [4].

Além de intrigar pesquisadores, o trânsito urbano é o protagonista em diversos jogos de entretenimento. Dentre eles, o Mini Metro é um jogo de estratégia em que o jogador desenha, e redesenha, uma malha metroviária para transportar passageiros até suas desejadas estações. O objetivo é manter o metrô operacional e transportar o máximo de passageiros, evitando a superlotação nas estações. À medida que o tempo passa, novas estações são inauguradas no mapa e precisam ser conectadas às linhas existentes. Para ajudar seus jogadores, recursos são adicionados, como a disponibilidade de mais uma composição, aumento da capacidade de uma composição ou linha adicional. Os recursos são limitados, portanto devem ser utilizados da melhor maneira possível . É neste contexto que surge a oportunidade de criar um algoritmo de inteligência artificial que auxilia na aplicação destes recursos tão escassos.

As restrições do jogo se dão apenas pelo tempo e os instrumentos fornecidos para tornar a tarefa de manter o serviço operante exequível. A abertura de novas estações é aleatória, bem como o número de passageiros a serem transportados e seus itinerários. Estas características indicam um ambiente dinâmico em que nem todas as informações sobre o estado estão disponíveis.

Existem diversas abordagens acerca de problemas de busca e rota são descritas na literatura. Para escolher um caminho, é necessário compreender o contexto do ambiente e o objetivo perseguido.

A título de exemplo, Ferrari propõe diminuir a congestão no trânsito através de um sistema de compartilhamento de carros privados entre indivíduos de itinerários semelhantes, considerando o algoritmo de Floyd-Warshal para buscar o caminho mais curto como ponto de partida [5]. Outro cenário se dá pelo propósito de criar um modelo de um gerenciador rodoviário otimizador de rotas principais e secundárias, intervalos e tamanho de veículos com base em algoritmos genéticos [2]. As buscas por rotas geográficas eficientes em que os usuários possam adicionar múltiplos destinos e chegar pelo menor percurso relembra o problema do caixeiro-viajante. Entretanto, para restringir e orientar resultados interessantes, e não cair em um problema de baixa performance por alta complexidade, é possível adicionar heurísticas, como a de tempo e de menor custo [3]. Não obstante, a demanda pelos serviços públicos é intrinsecamente uma das variáveis que adiciona complexidade para alcançar eficiência. Diante deste ambiente estocástico, uma das possibilidades se encontra na conjunção de métodos, como o uso de redes neurais, que possam performar soluções baseadas em meta heurísticas [6].

Com o propósito de atuar como um assistente pessoal de controlador de tráfego metroviário, o objetivo geral deste trabalho consiste no desenvolvimento de um pequeno simulador de linhas de metrô capaz de sugerir ações, com o propósito de atuar como um assistente pessoal de controlador de tráfego metroviário. Enquanto objetivos específicos estão i) elaborar uma breve revisão bibliográfica a fim de apontar a abordagem mais oportuna; ii) construir o algoritmo responsável pelo sistema de recomendação de ações; e iii) construir uma interface simples e intuitiva sobre a proposta do jogo, e que permita a conexão de plataformas e receba sugestões sobre melhores conexões.

A metodologia proposta para este trabalho consiste na pesquisa bibliográfica como etapa inicial. A segunda etapa se distingue pela construção do jogo em si, delimitando suas restrições e objetivo principal. Em uma terceira etapa, a interface é desenvolvida para permitir a interação com o jogador. Em sequência, o desenvolvimento do algoritmo de inteligência artificial compõe a etapa mais importante do trabalho. E como etapa final, a análise da performance do jogador ao seguir as ações indicadas pelo assistente inteligente.

[1] Estudante de Engenharia de Software da Escola Superior da Tecnologia da Informação - Instituto INFNET - Rio de Janeiro clarice.santos@al.infnet.edu.br

II. Definição do Problema

O jogo Mini Metro[1] consiste em um simulador estratégico em que, com o crescimento metropolitano, ocorre a expansão da malha metroviária através da abertura de novas estações. O objetivo do jogo se dá pela longevidade do serviço metroviário até que o número de passageiros exceda os padrões de qualidade admitidos pelas autoridades da cidade.

Logo no início, o mapa do metrô da cidade dispõe de três estações que podem ser conectadas por até três linhas disponíveis. A conexão entre as estações, bem como a quantidade de linhas em uso, é estipulada pelo jogador, como na Figura 1. Mediante a passagem de tempo, clientes com seus destinos representados através de formas geométricas surgem nas plataformas e aguardam até que possam embarcar. A chegada de novos passageiros, assim como a abertura de novas estações, é aleatória e limitada a um número previamente determinado. Ao fim do período de sete minutos, melhorias são oferecidas ao jogador para auxiliar nos novos desafios, como a possibilidade de adicionar a capacidade de lotação do vagão, a adição de mais carro ou a disponibilidade de mais uma linha. A velocidade em que as composições se deslocam é uma constante, mas a distância varia conforme o intervalo dos pontos, somados a extensão da linha. O desembarque do passageiro é sempre no destino cujo trajeto é o mais curto.

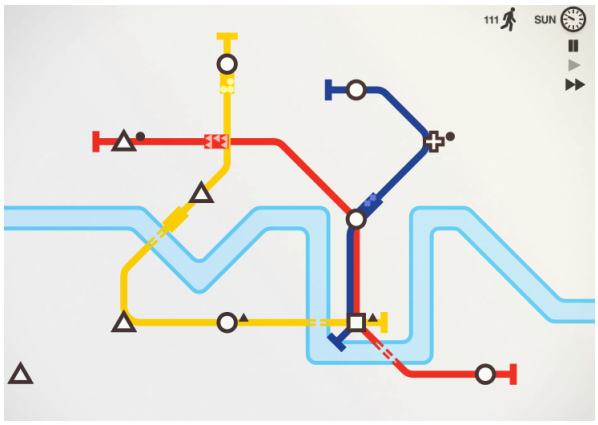

Fig. 1. Estações conectadas no Mini metro

A estação central é apenas uma conexão para a baldeação entre outras linhas. Quando uma estação excede um determinado limite de passageiros, um contador se inicia e apenas é interrompido se este número máximo for respeitado. Caso contrário, o jogo é finalizado e a derrota é anunciada.

As características do jogo compõem parte das restrições que o agente inteligente terá em perceber e atuar no ambiente. Dado que o agente não terá acesso completo às informações do ambiente, a primeira propriedade diz respeito ao seu meio inacessível. A constante mutação pode ser caracterizada como de natureza dinâmica, uma vez que esta independe da deliberação do agente. Outra propriedade importante concerne à independência do estado futuro do atual, implicando em uma propriedade não determinística. Além disso, a agente apenas pode acessar informações em momentos determinados, tornando a experiência em episódica.

Dado a grande quantidade de variáveis, e por consequência de estados, necessários para modelar o ambiente como linha, estação, número de passageiro, destinos, distância entre as estações, contagem regressiva para o fechamento de cada estação torna o problema bastante complexo para o objetivo deste trabalho. Portanto, o escopo precisa ser ajustado para garantir a formulação de um objetivo bem definido assim como a de um problema solucionável. Em vista disso, o trabalho visa implementar uma inteligência artificial capaz de sugerir conexões entre os pontos a fim de identificar o caminho mais curto, sem repetições, quando solicitado pelo jogador. Além de um problema e uma solução, também são indispensáveis parâmetros e métricas para pôr a teste a solução proposta. Estas questões serão detalhadas na seção de metodologia.

III. Referencial Teórico

Alguns dos exemplos clássicos de programação e de Inteligência Artificial envolvem buscas pelo menor caminho e de travessia entre pontos. Assim como em outras disciplinas de Ciência da Computação, a Inteligência Artificial utiliza ferramentas oriundas de outras áreas de conhecimento. A Teoria de Grafos serve para a abstração de problemas que podem ser representados por pontos e percursos. A complexidade algorítmica inerentes às estrutura de dados representáveis por grafos é superior as de tempo polinomial, adicionando mais esforço nas resoluções. Outra contribuição está na representação de grafos em matrizes de adjacências e lista de adjacências, úteis nos procedimentos computacionais [7].

Grafos então úteis na visualização da modelagem de problemas em que se deseja encontrar o caminho entre dois pontos, especialmente quando estes pontos são ponderados, buscando a trajetória de menor peso e consequentemente de menor distância. Um trajeto que atravessa todos os vértices de G é denominado um Caminho Euleriano de G, uma vez que Euler foi o primeiro a perseguir a investigação desse tipo de trajetória nos grafos. Nele, um caminho euleriano é aquele em que é possível tomar um itinerário em que se passa por uma aresta apenas uma vez. Por outro lado, Hamilton identificou um padrão chamado de ciclo hamiltoniano cujo significado se baseia na existência de um ciclo transitável por todos os vértices de um grafo sem repeti-los. Conceitos como o caminho euleriano e o ciclo hamiltoniano são importantes em problemas como o do caixeiro-viajante e de rota. O primeiro tem por objetivo a visita de todas as cidades voltando para a cidade inicial no menor percurso possível enquanto no segundo não há a preocupação de voltar para o ponto de partida. Embora sejam temas bastante recorrentes na literatura, pesquisadores ainda se desdobram para encontrar algoritmos performáticos e determinar soluções ótimas dado a vasta quantidade de aplicações voltadas para estes propósitos [8].

[1]O demo do jogo Mini Metro está disponível em: http://old.dinopoloclub.com/minimetro/

Com a finalidade de propor solução a busca por rotas de menor custo, alguns algoritmos ficaram consagrados. Eles podem ser classificados em buscas sem nenhum tipo de informação conhecidos como busca cega ou busca desinformada, e, em contraposição, aqueles que dispõe de alguma informação norteadora da busca, chamados de busca heurística ou busca informada. Dentre as estratégias para identificar o melhor tipo para um determinado problema estão as características como a completude, complexidade de tempo e de memória, e a optimização [9]. Os algoritmos de Dijkstra e A* são os mais utilizados com o propósito de pathfinding em jogos, sendo que este ganhou a popularidade de uso à medida que dispõe de uma heurística norteadora capaz de encontrar o destino mais eficientemente, sofrendo diversas adaptações para aumento de performance [10].

A. *Busca Cega*

O Algoritmo de Dijkstra é um dos algoritmos que não dispõem de informação e os custos dos nós da árvore expandida crescem de maneira uniforme. Ou seja, à medida em que se procura o caminho de menor custo, os nós folhas de menores valores são avaliados primeiro até que o custo de seguir naquela trajetória seja alto em comparação com outros disponíveis. Este algoritmo é capaz de encontrar o caminho mais curto, mas em termos de performance ele é bem custoso em níveis de profundidade maiores, de complexidade exponencial [11] [8]. A diferença entre os algoritmos de busca em largura e de busca em altura, é que este explora os nós de maneira vertical e aquele de maneira horizontal. Em outras palavras, a busca em profundidade sempre prioriza o nó mais profundo, o que pode levar a resultados não ótimos ou mesmo não encontrar nenhum devido a grande altura, enquanto a busca em largura estende os nós de um mesmo nível antes de descer mais um nível [9].

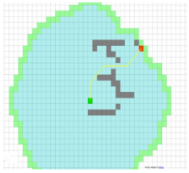

Fig. 2. Busca Cega com Dijkstra [12]

B. *Busca Heurística*

Por outro lado, buscas informadas tem como vantagem alguma informação útil, geralmente provida por meio de uma função de avaliação que estima o custo em direção ao caminho mais próximo do objetivo. A busca gulosa é um algoritmo cujos custos estimados são providos por uma função heurística, relacionada ao problema. No caso de um problema de rota, a distância em linha reta pode ser utilizada como orientação ao destino desejado. Como consequência, os nós mais próximos serão escolhidos como caminho e não haverá retorno a qualquer estado anterior, mesmo que exista um caminho mais curto. Felizmente, a busca A* é uma técnica que combina a completude de busca uniforme com a diminuição dos custos estimados através de uma heurística, resultando em uma função de dupla avaliação f(n) = g(n) + h(n) considerada otimizada.s [10]

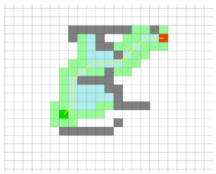

Fig. 3. Busca Informada com A* [12]

IV. Metodologia

Esta seção foca no processo de lapidação do problema e no percurso da implementação do objetivo principal do trabalho e finalidades subadjacentes necessárias.

Ademais, o desenvolvimento desta etapa tem como base a seguinte pergunta norteadora: Como elaborar um algorítimo para auxiliar um jogador de Mini Metro? Os questionamentos mais preponderantes e indispensáveis concernem a conexão das estações e a identificação da estrutura de dados mais adequada; a representação das estações de metrô enquanto vértices em um grafo; a apresentação da sequência de ações para demonstrar a solução; e a mensuração de performance da solução proposta.

A. *Descrições do agente e ambiente*

Novig frisa que a verdadeira arte de solucionar problemas está na decisão do que entra na descrição dos estados e operações e o que é dispensável; processo este conhecido como abstração [9]. Com base nisso, o primeiro passo está pautado na abstração do jogo Mini Metro. Apesar de ter uma interface limpa e propósito bem delineado, o jogo dispõe de múltiplas variáveis que tornam a reprodução e construção de um agente inteligente para lidar com um problema de estados múltiplos, conforme explanado na seção de descrição do problema.

Abreviadamente, se o agente dispusesse do acesso a todas informações do ambiente ele teria de observar a quantidade de estações, a conexão e a distância entre estas, a quantidade de pessoas e os seus destinos, o limite de passageiros por plataforma, as linhas e a capacidade e velocidade máximas de suas composições, o intervalo (parada) nas estações, entre outros. As únicas incertezas neste contexto são a abertura de novas estações e o surgimento de novos passageiros e seus destinos. Desta maneira, o ambiente se configura como majoritariamente acessível, não determinístico, não episódico, dinâmico e contínuo, ou seja, de grande complexidade, como na Figura 4.

A procura de algoritmos simples de pathfinding mais consagrados na literatura. A partir das referências consultadas, as estratégias de busca frequentes podem ser divididos entre aqueles que detém informação do mundo externo e

aqueles que não dispõe de tais instruções: buscas heurísticas e buscas cegas. Ambos podem ser aplicados no contexto deste trabalho, entretanto o algoritmo de busca uniforme de Dijkstra, por ser um dos mais utilizados nas aplicações de jogos e de rotas, foi adotado como método para encontrar o caminho mais curto e ótimo.

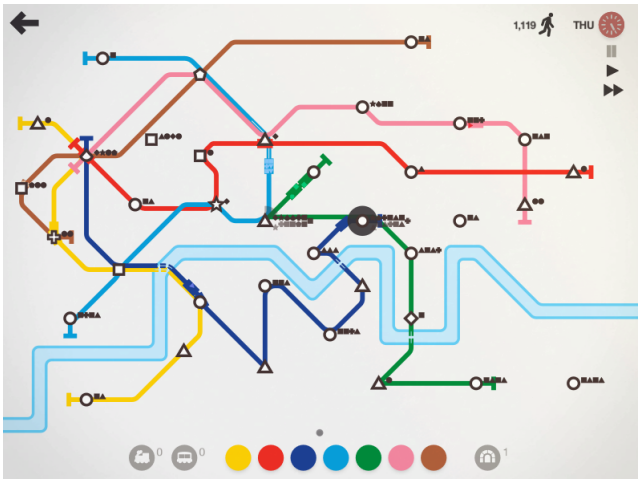

Fig. 4. Exemplo de uma partida já com alta fluxo de passageiros e estações

Enquanto as percepções do agente acerca de seu ambiente, destacam-se os estados de localização no grafo e as ações do agente se limitam à recomendação do próximo estado para chegar no objetivo determinado. Por consequência, o agente necessita checar se chegou no objetivo final, gerar novos estados se necessário, e definir uma ação em uma repetição até que se chegue no destino. O retorno é a sequência de escolhas desde o estado inicial até encontrar o estado final.

A fim de materializar o algoritmo e a interface, algumas tecnologias foram utilizadas. Javascript foi a linguagem de programação escolhida de maneira imperativa devido a compatibilidade com biblioteca para a criação de interface de jogos. A biblioteca kaboomJs [2] consiste em um micro framework, ainda em construção, para facilitar e agilizar a criação de jogos para a web, uma vez que dispõe de funções built-in para a adição e manipulação de objetos e comportamentos. É preciso sublinhar que o jogo original foi desenvolvido na plataforma Unity, e por meio de outra linguagem de programação, mas passível de ser disponibilizado nos browsers por meio de um plugin para a web [3].

B. Detalhes de desenvolvimento

O primeiro passo para a realização de qualquer busca a premissa da existência de um mapa e suas conexões. Existem algumas formas de representar as conexões destes caminhos vistos anteriormente: através de um grafo, de uma matriz de adjacências e de uma lista de adjacências. Apesar de ser uma boa representação visual, como na Figura 5, os grafos precisam ser convertidos para uma estrutura através da qual seja possível o processamento das informações pelo computador.

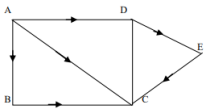

Fig. 5. Um grafo orientado G

A estrutura adotada aqui foi a lista de adjacências em que é a representação de do grafo através de uma lista, conforme a Figura 6.

Node	Adjacency List
A	B, C, D
B	C
C	
D	C, E
E	C

Fig. 6. Lista de adjacências de G

A criação de pontos aleatórios precede a implementação do grafo na interface. Esse procedimento envolveu a construção de uma função para gerar as estações de modo que a distância entre elas não interpolasse um outro ponto previamente adicionado. Sendo assim, inicialmente uma função escolhe um dos formatos geométricos disponíveis e adiciona algumas tags para futura manipulação. Em seguida, uma posição no canvas é escolhida aleatoriamente até que esta passe na condição de distanciamento mínimo. Por fim, a nova estação é renderizada para no browser. Uma vez que a quantidade de objetos máxima foi alcançada, se pode ligar os pontos para que um grafo seja criado de acordo com a vontade do jogador.

Já a geração do grafo necessita das coordenadas dos objetos para verificar a sua identidade, sua distância e a suas ligações. É importante ressaltar que conexão dos objetos leva em consideração os grafos orientados, sendo assim possível determinar apenas uma direção possível entre dois nós. À medida em que os elementos são selecionados, um objeto grafo é criado com base nas informações disponibilizadas, conforme a Figura 7 que demonstra a interface e a estrutura para representá-lo.

Por conseguinte, a partir deste momento se pode aferir o menor caminho entre duas estações. Uma função de é disparada quando se quer identificar o menor caminho e o grafo é processado, guardando o path em uma variável array. Os identificadores do array são comparados com os objetos do grafo a fim de criar um novo objeto com as posições para serem renderizadas no final do método, conforme a Figura 8.

Através do algoritmo de Dijkstra foi possível processar o objeto e retornar o menor caminho. O algoritmo funciona assim: se identifica o nó de menor custo como o nó atual, visitando todos os nós adjacentes. Conforme estes nós são averiguados, há a atualização de suas distâncias desde o nó inicial. Uma vez que o nó atual atualiza o valor dos nós de seus vizinhos e suas distancias se pode marcar o nó

[2]A versão da biblioteca de jogos kaboomJs foi a v.2000.1.6 e está disponível em: https://www.npmjs.com/package/kaboom

[3]Referenciado na documentação do Unity, disponível: https://docs.unity3d.com/2021.2/Documentation/Manual/Quickstart2DArt.html

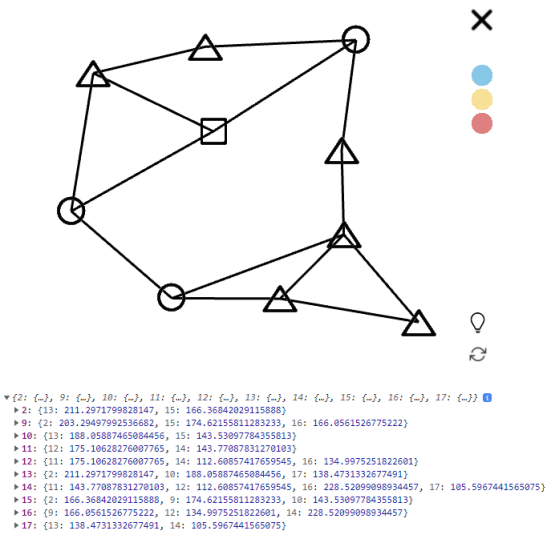

Fig. 7. Grafo e sua estruturação

enquanto visitado. Uma vez que eles são visitados se pode compreender que o custo final foi encontrado. Esta repetição ocorre para todos os nós do grafo até que se visite todos os nós. Ao final temos os custos identificados e os menores valores entre os vizinhos são preferidos, resultando em uma lista sequencial.

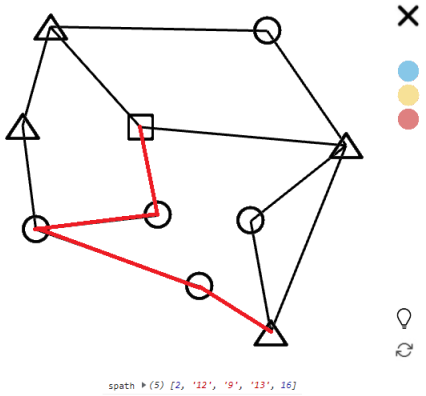

Fig. 8. Path encontrado

V. Simulação e Resultados

Conforme demonstrado na seção anterior, o algoritmo de Disjkstra foi capaz de encontrar os menores caminhos quando requisitado. Este algoritmo sempre acha uma solução, bem como a solução ótima quando implementado corretamente. Os testes executados foram em sua totalidade processados de forma manual, especialmente por possibilitar a averiguação visual.

A. Teste 1

O primeiro teste, conforme a Figura 9 dispõe uma lista de adjacências formada de 17 objetos cujas ligações resultam na sequência [2, ”48”, ”193”, ”13”, 39] através da qual se pode encontrar o menor caminho.

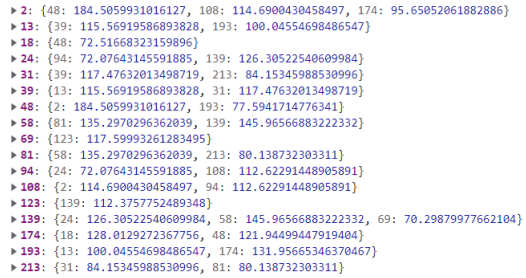

Fig. 9. Primeiro teste

B. Teste 2

Este teste representa o caso em que é apenas possível alcançar uma única solução uma vez que os nós não estão todos conectados, conforme a Figura 10. Em vista disso, a proposta do algoritmo converge com a única resolução alcançável, formada pela sequência: [2, ’9’, ’13’, ’174’, 39].

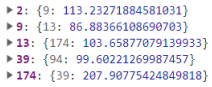

Fig. 10. Segundo teste

C. Teste 3

Já o terceiro teste, conforme a Figura 11 envolve 10 nós, entretanto o problema pode ser resolvido com apenas dois passos - [2, ’48’, 39] - por conta da recíproca direção entre os nós conectados. Este caso aponta que o acesso a lista de adjacências é fundamental para comprovar a eficácia da solução pois a singela verificação visual pode confundir devido a dupla orientação.

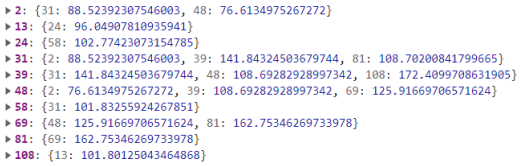

Fig. 11. Terceiro teste

É verdade que em um grafo com valores simples e de quantidade de nós pequenos se pode facilmente identificar um caminho. Entretanto, quando este grafo dispõe de um alto número de nós, distancias e conexões, esta tarefa se torna muito mais difícil para os seres humanos. Para um computador entretanto, esta tarefa é bem simples e depende mais dos recursos de memória do que processamento em si.

VI. Conclusão

A proposta inicial deste trabalho derivou do interesse em desenvolver um algoritmo inteligente que auxiliasse o jogador através de recomendações quanto as conexões entre estações para a construção de linhas de menores distâncias, especialmente para o caso em que o número de passageiros supera o limite estabelecido, como em um plano de contingência. Em termos de tecnologia, é aconselhável a escolha de bibliotecas consolidadas de desenvolvimento de jogos e de boa documentação.

Apesar disto, o problema se mostrou de uma grande complexidade e os objetivos precisaram estar alinhados com expectativas reais de implementação. A despeito da complexidade do problema, ainda foi possível desenvolver o mapa de estações e encontrar os caminhos mais curtos entre dois pontos de um grafo.

Para trabalhos futuros, fica a sugestão os seguintes passos para adicionar features:

i) adoção de heurísticas como norteadores de prioridades, como a não repetição de estações de mesmo tipo ou a de quantidade de passageiros em espera, dado um determinado caminho; ii) implementação de um outro algoritmo de busca informada, como a A*; iii) implementação de testes autônomos para possibilitar a checagem de performance do algoritmo escolhido.

References

[1] D. Kopec, *Classic computer science problems in Python*. Simon and Schuster, 2019.

[2] M. Petrelli, "A transit network design model for urban areas," *WIT Transactions on the Built Environment*, vol. 75, 2004.

[3] Y. Kanza, E. Safra, Y. Sagiv, and Y. Doytsher, "Heuristic algorithms for route-search queries over geographical data," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, 2008, pp. 1–10.

[4] K. Puljić and R. Manger, "Comparison of eight evolutionary crossover operators for the vehicle routing problem," *Mathematical Communications*, vol. 18, no. 2, pp. 359–375, 2013.

[5] E. Ferrari, R. Manzini, A. Pareschi, A. Persona, and A. Regattieri, "The car pooling problem: Heuristic algorithms based on savings functions," *Journal of Advanced Transportation*, vol. 37, no. 3, pp. 243–272, 2003.

[6] C.-s. Ying, A. H. Chow, and K.-S. Chin, "An actor-critic deep reinforcement learning approach for metro train scheduling with rolling stock circulation under stochastic demand," *Transportation Research Part B: Methodological*, vol. 140, pp. 210–235, 2020.

[7] A. Gibbons, *Algorithmic graph theory*. Cambridge university press, 1985.

[8] W. Wallis, "Graph theory with applications (ja bondy and usr murty)," 1979.

[9] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2002.

[10] A. Rafiq, T. A. A. Kadir, and S. N. Ihsan, "Pathfinding algorithms in game development," in *IOP Conference Series: Materials Science and Engineering*, vol. 769, no. 1. IOP Publishing, 2020, p. 012021.

[11] Y. Chao and W. Hongxia, "Developed dijkstra shortest path search algorithm and simulation," in *2010 International Conference On Computer Design and Applications*, vol. 1, 2010, pp. V1–116–V1–119.

[12] A. Noori and F. Moradi, "Simulation and comparison of efficency in pathfinding algorithms in games," *Ciência e Natura*, vol. 37, no. 0, 2015.