PROJETO FINAL DE GRADUAÇÃO 2

Análise de rotas de fuga em ambientes utilizando Aprendizado por Reforço

Anna Carolina Ferreira Rosa

Brasília, Outubro de 2022

UNIVERSIDADE DE BRASÍLIA

DEPARTAMENTO DE ENGENHARIA DE REDES DE COMUNICAÇÃO
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

PROJETO FINAL DE GRADUAÇÃO 2

Análise de rotas de fuga em ambientes utilizando Aprendizado por Reforço

Anna Carolina Ferreira Rosa

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Dr. Vinícius Pereira Gonçalves, _____
ENE/UnB
Orientador

Prof. Dr. Geraldo Pereira Rocha Filho, _____
CIC/UnB
Examinador

Prof. Dra. Mariana Cabral Falqueiro _____
Examinador

“In God we trust, all others bring data.".

W. Edwards Deming

"Não fui eu que ordenei a você? Seja forte e corajoso! Não se apavore nem desanime, pois o Senhor, o seu Deus, estará com você por onde você andar."

Josué 1:9

Agradecimentos

Primeiramente gostaria de agradecer a Deus, por ter me dado forças e ter me sustentado até esse momento. Aos meus pais, Silvia e David e meus avós Izabel e Ladislau, que me ensinaram tudo que sei e que fizeram o possível e o impossível para que eu tivesse a melhor educação. Aos meus irmãos Ryan, Matheus e Sarah por terem participado junto comigo de todos os momentos da minha vida, sendo eles bons ou ruins. A toda família Ferreira que cuidou de mim com todo o carinho do mundo e me ensinou o que é o amor.

Agradeço ao meu orientador, professor Vinícius e o professor Calvin, por terem me apresentado esse tema e aceitado participar comigo desse projeto. A todos os meus amigos e amigas que fiz dentro da graduação, e que junto comigo, trilharam todo esse caminho. A Diretoria de Desenvolvimento Social da Universidade de Brasília que me auxiliou em todos os momentos que precisei e tornaram possível a minha continuidade dentro da Universidade.

Gostaria de agradecer o meu professor Alexandre, que confiou em mim e me deu a oportunidade mais incrível na minha vida, me proporcionando aprendizados que nunca poderei mensurar. A todas as professorar mulheres que tive dentro da minha graduação, que serviram de espelho e referência para que eu pudesse acreditar nos meus sonhos.

A meus amigos que fiz durante toda a vida, que estiveram ao meu lado, em especial o Lucas, que esteve comigo durante toda minha trajetória, me incentivando e me ajudando em todos os momentos que eu precisei.

Dedico esse trabalho a todas as mulheres, dentro e fora da Engenharia, que lutam por espaços e sonham em ser referência e espelho para muitas outras mulheres, para que a sociedade possa entender e reconhecer que somos, sim, capazes de ser o que quisermos.

Anna Carolina Ferreira Rosa

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Com a contínua ocorrência de casos de incêndios em edifícios em todo o mundo, é possível utilizar os avanços da tecnologia para criar uma aplicação para auxiliar a rápida evacuação em casos de emergência, utilizando a planta baixa de um edifício. Esse trabalho apresenta a análise e descoberta da melhor rota de fuga dentro de um edifício, utilizando técnicas de Aprendizado por Reforço.

Palavras-chave: incêndio, evacuação, aprendizado de máquina, aprendizado por reforço.

ABSTRACT

With the continuous occurrence of fire cases in buildings around the world, it is possible to use advances in technology to create an application to assist the rapid evacuation in cases of emergency, using the building floorplan. This work presents the analysis and detection of the best escape route inside a building, using Reinforcement Learning techniques.

Keywords: fire, evacuation, image processing, machine learning.

Capítulo 1

Introdução

1.1 Motivação

Casos de incêndio no Brasil e no mundo continuam a acontecer, em edifícios fechados, o fogo pode se espalhar rapidamente e o tempo de reação a uma ameaça de incêndio a partir do momento que o alarme é acionado, é pequeno, podendo ser de poucos minutos até poucos segundos (57). Quanto mais rápido se identificar a melhor saída do edifício, maior a probabilidade de escapar do acidente sem maiores danos, porém conseguir identificar a melhor saída nesses casos pode ser uma tarefa difícil, tanto pelo poder de espalhamento do fogo, quanto pelo pânico gerado nas vítimas.

Segundo a Administração de Segurança e Saúde Ocupacional dos Estados Unidos (OSHA), as rotas de fuga podem ser definidas como "um caminho contínuo e desobstruído de viagem de saída de qualquer ponto dentro de um local de trabalho para um local de segurança", portanto são rotas que as vítimas podem utilizar durante casos de emergência para saírem de um local perigoso para um local seguro (16). As rotas de fuga são de extrema importância em todos os edifícios, pois é através delas que se consegue ter uma evacuação do local da forma rápida e segura. Além das vítimas do incêndio, bombeiros, socorristas e policiais podem utilizar as rotas de fuga para auxiliar o resgate.

A Norma Brasileira 9077 se refere a saídas de emergência em edifícios e detalha todas as normas necessárias para criar um edifício seguro, desde sinalização de segurança, iluminação de emergência, até escadas como saídas de emergência e dimensionamento das saídas de emergência (1). Todos os edifícios devem ter um plano de fuga para casos de incêndio, é imprescindível que existam pelo menos duas rotas de fuga reconhecíveis para um local seguro, além de sensores de fumaça ligados a centrais de alarme de incêndios para alertar e auxiliar as ações dos bombeiros (10).

Com o avanço da tecnologia, já é possível criar uma aplicação para facilitar a fuga de vítimas, assim como, a atuação de bombeiros, socorristas e policiais em casos de incêndio, através da análise de plantas baixas do edifício, combinado com os sensores de fumaça ligados a centrais de alarme de incêndios (10).

Já existem pesquisas para a rápida identificação de rotas de fuga, a maioria delas é direcionada

pela análise de plantas baixas de edifícios. Analisar e rotular uma planta baixa de forma automática não é uma tarefa fácil, sendo necessário, técnicas precisas de processamento de imagem. Pode-se combinar as técnicas de Aprendizado de Máquina, com as técnicas já definidas de processamento de imagem para uma rápida análise de plantas baixas, como em (26), onde utilizando redes neurais, foi possível converter plantas baixas rasterizadas em representações vetoriais gráficas, com uma alta precisão.

Com representações gráficas da planta baixa de um edifício, pode-se utilizar algoritmos de busca de caminhos como Dijkstra, Bellman-Ford e A* (6) para traçar rotas de fuga. Quanto maior a quantidade de nós de um grafo, mais lenta a busca pela melhor rota de fuga em algoritmos de busca de caminho, por isso uma recente solução para esses casos é o uso de Aprendizado por Reforço.

Aprendizado por Reforço (46) é um método de Aprendizado de Máquina supervisionado baseado em recompensas para comportamentos desejados e punições para comportamentos não desejados. Nessa técnica, existe um agente do Aprendizado por Reforço que consegue interpretar seu ambiente e realizar ações, aprendendo por tentativa e erro, sempre buscando maximizar sua recompensa. Apesar de ser comumente usado para cenários de jogos, o Aprendizado por Reforço pode ser utilizado para busca de melhor caminho em grafos e matrizes.

A aplicação receberá imagens de planta baixa de edifícios, e seu grafo correspondente. Sendo assim é possível calcular as melhores rotas de fuga através de simulações de incêndios para uma análise prévia da planta do edifício. Nesses casos, com as informações dos sensores de fumaça ligados as centrais de incêndio, ou as informações dos cômodos atingidos por fogo, será possível calcular em tempo real as melhores rotas de fuga baseado na localização das vítimas, ajudando na rápida evacuação e no resgate.

1.2 Objetivos

O trabalho visa o estudo de alternativas para a rápida evacuação de vítimas de casos de incêndios, utilizando técnicas atuais, rápidas e precisas. Através da representação da planta baixa de um edifício fechado, é possível encontrar as melhores rotas de fugas em poucos segundos ou minutos.

A alternativa escolhida e desenvolvida foi utilizando Aprendizado por Reforço, um campo de Aprendizado de Máquina, pouco explorado para aplicações em tempo real que necessitam ter um rápido treinamento, bem como uma alta precisão e acurácia nos resultados obtidos na inferência. O objetivo do trabalho não é a análise, rotulação e extração de plantas baixas de edifícios, e sim o estudo do que pode ser feito utilizando a representação em forma de grafo de uma planta baixa de um edifício fechado.

1.2.1 Objetivos Específicos

- Criação de uma interface web para que o usuário possa cadastrar sua planta baixa e seu grafo correspondente e possa encontrar as melhores rotas de fuga do seu edifício.
- Criação de um banco de dados para a persistência das informações da planta baixa e do grafo correspondente.
- Criação de uma API que será responsável pela comunicação entre a aplicação web e o banco de dados criado para armazenar as informações do projeto e a inferência do modelo de Aprendizado por Reforço criado.
- Criação de um modelo de Aprendizado por Reforço Profundo para encontrar os melhores caminhos dentro de um grafo.

1.3 Organização do trabalho

O trabalho é dividido em cinco partes principais: referencial teórico, trabalhos relacionados, metodologia e implementação, resultado e conclusões.

No referencial teórico será exposto a fundamentação teórica dos assuntos que serão importantes para a o tema escolhido. Essa primeira parte será dividida em três tópicos: rotas de fuga em um edifício, aprendizagem por reforço e conceitos básicos da teoria dos grafos. Nos trabalhos relacionados, será abordado estudos e pesquisas já realizados nessa área.

Na metodologia e implementações será exposto as estratégias utilizadas para a resolução do problema de rotas de fuga em casos de incêndios. Essa segunda parte será dividida em três tópicos: representação da imagem de uma planta baixa, algoritmo de *Q-Learning* e aprendizado por reforço profundo.

Nos resultados será documentado os experimentos realizados com três projetos diferentes e os resultados obtidos pelo algoritmo de aprendizado por reforço, levando em consideração algumas métricas como perda e recompensa acumulada.

Na conclusão será realizado uma análise de todo o trabalho.

Capítulo 2

Referencial Teórico

Este capítulo trata dos principais aspectos teóricos relacionados em como representar uma imagem de uma planta baixa em uma forma que possa ser analisado as rotas de fugas do edifício em casos de emergências, utilizando técnicas de Aprendizado por Reforço.

2.1 Aprendizado por Reforço

Desde o nascimento, qualquer ser humano está constantemente aprendendo e interagindo com o ambiente que o cerca, aprendendo sobre a causa e efeito, sobre as consequências de suas ações e como atingir metas desejadas. O aprendizado por interação acontece de várias formas durante a vida, é com essa ideia que surge um campo de Aprendizado de Máquina conhecido como Aprendizado por Reforço.

Aprendizado por Reforço, é um método de tentativa e erro que busca aprender a como maximizar a recompensa em uma tarefa específica. O agente de aprendizagem não será informado em nenhum momento qual melhor ação ele deve tomar, portanto, terá que interagir com o ambiente a fim de descobrir quais ações maximizarão sua recompensa. É importante notar que ao escolher uma ação, o agente não está apenas afetando sua recompensa imediata, mas pode afetar sua próxima recompensa e várias que seguirem.

2.1.1 Aprendizado Supervisionado x Aprendizado não Supervisionado

A maioria dos estudos sobre Aprendizado de Máquina ainda estão concentrados em aprendizado supervisionado, como os métodos de classificação e regressão, onde é necessário rotular a mapear os dados de entradas e saídas do treinamento, para que se aprenda corretamente a extrapolar e generalizar sua resposta para um objetivo final através de resultados pré-definidos, dependendo de uma intervenção humana para a criação e rotulagem desses dados. Porém, para aprendizagem por interação com um ambiente, rotular os dados de treinamento não é a melhor solução, por existir diversas situações distintas. Na Aprendizagem por Reforço se utiliza um tipo de aprendizagem não supervisionada (44).

Na aprendizagem não supervisionada não é necessário a intervenção humana, utilizando uma abordagem de explorar dados desconhecidos, não sendo necessários exemplos de resultados pré-definidos para a aprendizagem, pelo contrário, através da interação com o ambiente e suas experiências, o agente de aprendizagem conseguirá capturar padrões, entender e processar os dados, aprendendo sozinho as melhores ações a se tomar. A aprendizagem não supervisionada combinada com a maximização de recompensas é o Aprendizado por Reforço.

2.1.2 Exploração x Intensificação

Um dos principais conceitos da Aprendizagem por reforço é a diferença entre as políticas de exploração (*exploration*) e intensificação (*exploitation*) (31). Como o agente de aprendizagem busca maximizar sua recompensa final, ele deve usar suas experiências passadas para decidir quais ações tomar, porém, sempre optando por ações que já foram escolhidas, ele não terá conhecimento de todas as ações que poderão ser tomadas dentro do ambiente, sendo assim, é necessário que ele também explore o ambiente, escolhendo ações que não foram selecionadas anteriormente, para comparar com os resultados que já foram obtidos. Intensificação é uma política utilizada pelo agente de aprendizagem para tomar ações com base em suas experiências obtidas e exploração é uma política utilizada pelo agente de aprendizagem para selecionar aleatoriamente ações, visando obter conhecimento de todo o ambiente.

A taxa de exploração deve ser muito bem escolhida, pois uma taxa muito alta pode fazer com que o agente perca suas experiências passadas, comprometendo o aprendizado e consequentemente a recompensa final. Porém, não escolher uma taxa de exploração pode fazer com que o agente não explore todas as ações disponíveis.

2.1.3 Elementos da Aprendizado por Reforço

Existem seis elementos principais para o método de Aprendizado por Reforço: agente, ambiente, estado, política, recompensa e função de valor (45).

O agente de aprendizagem será a entidade responsável por interagir e tomar decisões no ambiente, recebendo recompensas positivas ou negativas de acordo com suas ações, o ambiente é o espaço onde o agente realizará suas ações. Ao realizar uma ação, o agente estará em um estado (s) atual que fornecerá uma recompensa (r). A cada nova ação, o agente se encontrará em um novo estado (45).

A política (π) é a estratégia escolhida para ser aplicada ao agente em determinado momento para ele poder decidir qual será a próxima ação. Geralmente é utilizado o padrão de probabilidade estocástico. Pode-se considerar a política como o núcleo principal do Aprendizado por Reforço, pois é onde se determina o comportamento do agente (45).

A recompensa (r) define o objetivo do Aprendizado por Reforço, pois é através dela que o agente aprenderá se sua ação imediata foi desejada ou não. A cada ação tomada pelo agente de aprendizagem, o ambiente retornará uma recompensa, podendo ser negativa, positiva ou nula.

Como o objetivo do agente é maximizar sua recompensa final, ele começará a aprender as melhores ações a serem tomadas dentro do ambiente, com base nas recompensas adquiridas em cada ação, alterando assim a política utilizada (45).

A recompensa é utilizada para indicar o quão bom será a próxima ação que poderá ser tomada pelo agente, mas ela não consegue definir o quão bom será o percurso a longo prazo do agente começando de um estado específico, a função de valor tem esse objetivo. Se o agente se importa apenas com o próximo estado imediato, ele pode acabar evitando uma ação com uma recompensa baixa, mas que possui altas recompensas nos próximos estados a longo prazo, ou o contrário, tomar ações com recompensas imediatas altas, mas que possui baixas recompensas nos próximos estados a longo prazo, isso pode ser evitado considerando a função de valor (45).

As recompensas são retornadas pelo ambiente a cada ação tomada pelo agente, porém a função de valor deve ser estimada e atualizada ao longo das sequências de observações feitas pelo agente, durante todo seu percurso (45).

2.1.4 Bandidos Multi-Armados (*Multi-armed Bandits*)

O problema mais clássico de Aprendizado por Reforço é o Bandido Multi-Armados. É um problema que ilustra as noções de tomada de decisões de um agente, as recompensas e a função de valor.

No problema de Bandidos k-armados, o agente pode escolher sempre k diferentes opções de ações. É retornado pelo ambiente uma recompensa conforme a ação escolhida, e o agente sempre busca maximizar a recompensa total após um mesmo período de tempo pré-determinado. Com o tempo o agente conseguirá tomar as melhores ações de acordo com o aprendizado adquirido, maximizando assim sua recompensa.

Para melhores explicações e outras variações do problema do Bandido Multi-Armado é recomendado a leitura de (47).

2.1.4.1 Ação e Valores

Utilizaremos a ação tomada em um dado momento de tempo t como A_t, e a recompensa retornada pelo ambiente como R_t. O valor de uma ação aleatória a é representado como mostra a Equação 2.1 (47).

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a] \tag{2.1}$$

Se existem os valores de recompensa de todas as ações que o agente pode escolher no ambiente, fica fácil resolver o problema proposto, pois basta escolher sempre a ação que possui a maior recompensa. Por isso é necessário considerar que o agente não possui o conhecimento de todos os valores de recompensa, mas sim uma estimativa desses valores, representada como $Q_t(a)$, é esperado assim que $Q_t(a)$ seja aproximadamente $q_*(a)$ (47).

Uma maneira simples de definir a estimativa dos valores das ações é como mostra a Equação 2.2, se o denominador for zero, ou seja, a ação ainda não foi escolhida nenhuma vez, $Q_t(a)$ vai ser definido por um valor padrão, como zero, se o denominador tender a infinito então $Q_t(a)$ será aproximadamente $q_*(a)$.

$$Q_t(a) \doteq \frac{\text{soma das recompensas quando a ação (a) é escolhida antes do tempo (t)}}{\text{Numero de vezes em que a ação (a) é escolhida antes do tempo (t))}} \tag{2.2}$$

Se os valores de recompensa forem armazenados, o agente poderá sempre escolher a próxima ação que gera a maior recompensa estimada, como discutido no Subseção 2.1.2, essa técnica é conhecida como intensificação (*exploitation*), e caso o agente escolha uma ação aleatória, não levando em consideração os valores de recompensa armazenados, será utilizado a técnica de exploração (*exploration*). É necessário realizar o balanceamento dessas técnicas, visando sempre maximizar os valores de recompensa obtidos, esse balanceamento é conhecido como algoritmo *epsilon-greedy* (47).

O algoritmo *epsilon-greedy* utilizará um valor de probabilidade ε para decidir se irá utilizar o método de intensificação ou exploração. Caso o método de intensificação seja escolhido, o agente irá escolher a próxima ação com base no máximo valor estimado, como mostra a Equação 2.3, caso contrário ele escolherá uma ação aleatório entre todas as possíveis, independente do valor $Q_t(a)$ estimado (47).

$$A_t \doteq \text{argmax}_a Q_t(a) \tag{2.3}$$

2.1.5 Processo de decisão de Markov (MDP)

O Processo de Decisão de Markov (*MDP -Markov Decision* Process) é um modelo matemático cujo objetivo é modelar a tomada de decisão sequencial em que os resultados são parcialmente aleatórios e parcialmente sob o controle de um tomador de decisão. No MDP não é levado em consideração apenas as recompensas imediatas das ações, mas também as recompensas futuras das próximas ações e estados (48).

2.1.5.1 Interface de Agente-Ambiente

Na Subseção 2.1.4.1, no problema do Bandido Multi-Armado, foi definido $q_*(a)$ como o valor de uma ação aleatória a, porém no MDP será considerado não apenas a ação, mas também o estado s. A interação entre o agente e o ambiente, considerando as ações, estado e recompensas em um tempo t pode ser visualizado na Figura 2.1 (50).

O agente irá interagir com o ambiente em uma sequência de tempos discretos $t = 0, 1, 2, ..., n$, a cada passo de tempo t, o agente estará em algum estado S_t dentro do ambiente, com base em uma ação A_t. Quando o agente selecionar uma nova ação, ele irá mudar para um novo estado S_{t+1} e receberá uma nova recompensa R_{t+1} e então o ciclo será novamente reiniciado para o próximo instante de tempo.

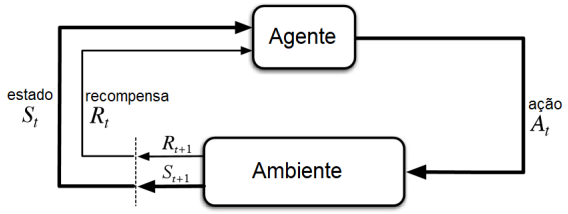

Figura 2.1: Interação entre o agente e o ambiente no MDP

Fonte: Medium (29)

Como as ações $\boldsymbol{A}$, estados $\boldsymbol{S}$ e recompensas $\boldsymbol{R}$ possuem um conjunto finito de números de elementos, pode-se considerar que R_t e S_t possuem uma probabilidade de distribuição discreta, que depende exclusivamente dos estados e das ações.

A probabilidade de transição de um estado s para o outro s', em um tempo t, com recompensa r por escolher uma ação a, pode ser definido conforme a Equação 2.4. Sabe-se que a função de probabilidade $p = \boldsymbol{S} \times \boldsymbol{R} \times \boldsymbol{S} \times \boldsymbol{A} \rightarrow [0, 1]$, pois $s', s \in \boldsymbol{S}$, $r \in \boldsymbol{R}$ e $a \in \boldsymbol{A}$ (50).

$$p(s', r|s, a) \doteq \Pr\left\{S_t = s', R_t = r|S_{t+1} = s, A_{t+1} = a\right\} = \sum_{r \in \boldsymbol{R}} p(s', r|s, a) \tag{2.4}$$

Dado um par de estado e ação, pode-se calcular o valor de recompensa esperado, como mostra a Equação 2.5 e dado o triplete de estado, ação e próximo estado, pode-se calcular o valor da recompensa como mostra a Equação 2.6 (50).

$$r(s, a) \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \boldsymbol{R}} r \sum_{s' \in \boldsymbol{S}} p(s', r|s, a) \tag{2.5}$$

$$r(s, a, s') \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \boldsymbol{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)} \tag{2.6}$$

2.1.5.2 Objetivos e Recompensas

O objetivo final do Aprendizado por Reforço é fazer com que o agente sempre tenha capacidade de aprender como maximizar o número da recompensa acumulada ao longo do tempo, ou seja, não apenas a recompensa R_t obtida a cada passo de tempo, mas sim a soma de todas as recompensas obtidas. Em (51) é definido a hipótese de recompensas como:

"Tudo o que entendemos por objetivos e propósitos pode ser bem pensado como a maximização do valor esperado da soma cumulativa de um sinal escalar (chamado recompensa)."

Por isso é importante que as recompensas retornadas para o agente, realmente auxilie no aprendizado do mesmo, indicando o objetivo final.

2.1.5.3 Retornos e episódios

Um dos conceitos importante no Aprendizado por Reforço é o Retorno (G), ele é definido como alguma função específica da sequência de recompensas, podendo ser a soma das recompensas esperadas (52), como mostra a Equação 2.7, é importante observar que nessa Equação são as recompensas esperadas futuramente, isso explica o primeiro termo da Equação ser t e não *t+1*.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.7}$$

Essa função é valida para aplicações que possuem um final bem definido, onde a interação entre o agente e um ambiente possui uma pausa definida quando se chega o estado final e pode então recomeçar a interação do estado inicial. Essa pausa definida é conhecida como episódio, um episódio começa no estado inicial definido e termina quando o agente consegue atingir o estado final (52).

O estado final não necessariamente é quando o agente termina de uma forma bem sucedida, podendo acabar um qualquer estado do ambiente. Independente da onde o agente termine o episódio anterior, o próximo episódio irá começar do estado inicial estabelecido.

Para aplicações que não possui um estado final bem definido, ou para aplicações específicas, o estado final $T{=}\infty$ e, o retorno G também tenderá a infinito. Para isso é utilizado o retorno com desconto, como mostra a Equação 2.8, onde é adicionado um fator de desconto $0 \leqslant \gamma \leqslant 1$, que será responsável por determinar a importância que terá as recompensas futuras (52).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.8}$$

Analisando essa equação percebe-se que quanto menor o valor de γ, mais relevantes serão as recompensas imediatas.

2.1.6 Políticas e Funções de Valores

A política é uma função de valor que estima o quão bom, considerando as recompensas futuras, é realizar uma determinada ação, estando em um determinado estado. A política mapeia um estado em probabilidade de se selecionar as ações possíveis. A fórmula da política é a probabilidade de no tempo t selecionar uma ação A_t, dado o estado S_t, como demonstra a Equação 2.9 (49).

$$\pi(a|s) = P(A_t = a|S_t = s) \tag{2.9}$$

A política é atualizada com frequência, até que se atinja uma configuração ótima, onde através dessa função de valor, o agente conseguirá escolher as melhores ações.

A função de estado-valor $v_\pi(s)$ de estado da política π, é um valor de retorno esperado ao se iniciar em um estado s e seguir a função π. Essa função considera o retorno, como mostra a

Equação 2.10, onde $\mathbb{E}_\pi$ é o valor esperado dado que o agente segue a política π, no passo de tempo t (49).

$$v_\pi \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s], \text{ para todo s} \in \boldsymbol{S} \tag{2.10}$$

É importante observar que se existir um estado final bem definido, a função de estado-valor da política nesse estado final é sempre nulo.

Existe também a função de ação-valor da política π, sendo esse o valor esperado de se realizar uma ação a, dado o estado s, seguindo a política π. Essa função é definida como mostra a Equação 2.11.

$$q_\pi \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.11}$$

2.1.7 Equação de Bellman

A equação de Bellman é um dos principais elementos do estudo de Aprendizado por Reforço, ela utiliza as funções de valores para definir uma equação funcional sobre a relação entre o valor de um estado e os valores dos possíveis estados sucessores.

Essa equação tem o objetivo de simplificar o cálculo das funções de valores, para encontrar uma solução ótima, calculando a média de todas as possibilidades, ponderando cada uma pela sua probabilidade de ocorrer.

A equação de Bellman afirma que o valor do estado inicial deve ser igual ao valor descontado do próximo estado esperado, mais a recompensa esperada ao longo do caminho e pode ser visualizada na Equação 2.12 (49).

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \text{para todo s} \in \mathbf{S} \tag{2.12}$$

2.1.8 Q-Learning

O algoritmo *Q-Learning*, busca encontrar a melhor ação a ser tomada, considerando seu estado atual, é o algoritmo mais utilizado quando se utiliza Aprendizado por Reforço. É considerado um algoritmo *off-policy*, pois aprende com ações que estão fora da política atual, como ações aleatórias, então não é necessária uma política, isso significa que ele aprende os valores ótimos, independente das ações do agente (14). Esse algoritmo também é *model-free*, pois também não precisa de um modelo.

O Q em *Q-Learning* significa qualidade, ou seja, o quão útil é uma determinada ação para obter alguma recompensa futura. Esse algoritmo é baseado em valor, isso significa que ele atualiza

sua função de valor de acordo com uma equação, geralmente a Equação de Bellman explicada na Subseção 2.1.7.

As funções de valor são um par de ações de estados que indicam o quão boa pode ser uma ação específica, caso se esteja em um estado, ou qual a recompensa esperada para aquela ação, como explicado na Subseção 2.1.6. Como esse algoritmo é *off-policy*, sua função tende a convergir com probabilidade 1 para uma aproximação da função de valor da ação, mesmo quando as ações são selecionadas por exploração ou intensificação (14).

No algoritmo de *Q-Learning*, *Q*(s,a)* será o valor esperado a se tomar uma ação a, estando em um estado s seguindo a política ótima, o algoritmo irá utilizar a técnica não supervisionada de Diferença Temporal (*TD - Temporal Difference*), onde o agente de aprendizagem irá aprender a prever o valor esperado de uma variável que ocorre no final de uma sequência de estados (53). O agente deverá manter uma tabela atualizada Q[S,A] de estados e ações, conhecida como *Q-Table*.

A *Q-Table* é uma tabela que será utilizada para calcular a recompensa máxima esperada de uma ação, dado um estado. O algoritmo de *Q-Learning* é simples, primeiramente é necessário inicializar a *Q-Table* e então para uma interação escolher uma ação a dado um estado s de acordo com a política escolhida, ao realizar essa ação deve-se observar a recompensa r e o novo estado s', com esses valores é possível atualizar a *Q-Table* (14).

Para a atualização da *Q-Table* é utilizado a função de valor conhecida como *Q-function*, como mostra a Equação 2.13, que utiliza os conceitos da Equação de Bellman.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.13}$$

O alfa (α) é um número real entre 0 e 1, que representará a taxa de aprendizagem do agente, quanto menor o valor do alfa, menos o agente aprenderá com novas ações, quanto maior o valor de alfa, o agente tende a ignorar o conhecimento prévio e priorizará as informações mais recentes. Sendo assim, quanto maior o valor de alfa, mais rápido os valores da *Q-Table* irão mudar.

O gama γ é um número real entre 0 e 1 conhecido como fator de desconto, ele definirá a importância da próxima recompensa, quanto menor o gama, menos o agente irá considerar as recompensas futuras, considerando mais as recompensas atuais, por outro lado, quanto maior o valor do gama, o agente irá procurar por altas recompensas a longo prazo.

Um exemplo do algoritmo de *Q-Learnig* será demonstrado na Subseção 4.1, onde será utilizado uma matriz adjacente de um grafo como ambiente.

2.1.9 Aprendizado por Reforço Profundo

O Aprendizado por Reforço Profundo (*Deep Reinforcement Learning*) é um subcampo do Aprendizado por Reforço que combina as técnicas de Aprendizado por Reforço e Aprendizado Profundo, é muitas vezes também conhecido como *Q-Learning* Profundo (*Deep Q-Learning*) por na maioria das vezes utilizar e se basear no algoritmo de *Q-Learning*.

Aprendizado Profundo (*Deep Learning*) é um subcampo do Aprendizado de Máquina que utiliza algoritmos inspirados na estrutura de um cérebro humano chamado de redes neurais (21), é um dos subcampos de Aprendizado de Máquina mais estudado atualmente por se mostrar eficiente em tarefas de classificação e detecção, principalmente no campo das imagens. As redes neurais são técnicas computacionais que, através de um modelo matemático, conseguem simular o funcionamento de um cérebro humano, essas redes aprendem a realizar uma tarefa específica através de exemplos de treinamento.

Essas redes neurais são compostas por camadas de nós de processamento que são densamente interconectados, onde existe uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Um nó pode estar conectado a um ou vários nós de camadas acima, ou abaixo dele. Cada camada é composta por transformações que levam o aprendizado a diferentes níveis de abstração.

As redes neurais possuem pesos, que serão responsáveis por definir o impacto que cada variável de entrada de um nó, tem no valor das variáveis de saída, então um valor é atribuído por um nó para cada conexão de entrada que ele possuir, quando o nó receber um dado diferente de uma de suas conexões, ele irá multiplicar esse peso pelo item de dado recebido e somará os produtos resultantes de todas as conexões em apenas um único número para ser ou não enviado para a próxima camada, dependendo de um valor de limite escolhido. Esses pesos e limites são inicialmente inicializado com valores aleatórios e vão sendo ajustados a medida que ocorre o treinamento (30).

As redes neurais auxiliam o Aprendizado por Reforço quando o tamanho do ambiente e a quantidade de ações que o agente pode tomar são muito grandes, aproximando as funções de valores ou funções de políticas, aprendendo a mapear os pares de estado-ação como feito no algoritmo de *Q-Learning*, substituindo assim a *Q-Table* por uma rede neural.

2.1.9.1 Experience Replay

Experience Replay (Repetição de Experiência) é uma das técnicas mais utilizadas em Aprendizado por Reforço Profundo, aumentando a eficiência e estabilidade do treinamento. Assim como o algoritmo de *Q-Learning* também é *off-policy* (aprende com ações que estão fora da política atual).

A técnica utiliza uma memória ou *buffer* de repetição de tamanho fixo que armazena as transições mais recentes coletadas pela política, ou seja, as experiências mais recentes vividas pelos agente a cada passo de tempo, com isso, esses dados podem chegar a ser utilizados várias vezes no treinamento e não simplesmente descartadas a cada episódio (17).

Esse *buffer* de repetição possui valor fixo, que por padrão é 10^6 (60), o método mais utilizado é o de amostragem uniforme, onde quando esse *buffer* estiver com sua capacidade máxima de armazenamento, dados mais antigos começam a ser descartados para a entrada de novas transições. As transições geralmente são quadruplas de valores (estado, ação, recompensa, final, novo estado). Existem outros tipos de amostragem como a amostragem priorizada (39), onde existe prioridade nas transições, porém é mais complexa e não será utilizada nesse trabalho.

Com o *buffer* de repetição, é possível atualizar a *Q-function* ou treinar a rede neural, amostrando um lote de valor pré-definido de transições do *buffer*.

O algoritmo de *Experience Replay* e o funcionamento do *buffer* de repetição utilizando o algoritmo de *Q-Learning*, pode ser visto no Algoritmo 1, sendo simples de entender e implementar.

Algoritmo 1: Experience Replay

Entrada: *Q-function* - Q, *buffer* de repetição - M, Estado inicial - s

```
1  início
2      repita
3          Selecione uma ação a de acordo com a política escolhida
4          Execute a ação a e receba a recompensa r e o próximo estado s'
5          Armazena a transição (s, a, r, s') em M Amostre um lote de transições B de M
6          Atualize a função Q com B
7          s = s'
8      até s = estado final
9      ;
10 fim
```

2.1.9.2 Rede de Destino

Umas das diferenças entre o algoritmo de *Q-Learning* para o de *Q-Learning* Profundo é que no primeiro existe uma função de valor exata, já no segundo existe uma aproximação dessa função de valor, fazendo uma tentativa de aproximar uma função complexa e não linear, utilizando uma rede neural. Isso pode acabar tornando a rede instável, prejudicando as decisões de próximo estado a se tomar (58).

A função de perda MSE (erro quadrático médio) necessita de dois valores, o valor de Q-previsto (s) e o valor de Q-alvo *(s')*, portanto utilizando o *Q-Learning* Profundo é preciso duas passagem pela Rede Neural, para a descoberta de cada valor. Calcular com a mesma rede esses dois valores pode causar instabilidade, prejudicando o valor de Q-alvo (11).

Para que não exista essa instabilidade, pode ser utilizado uma rede auxiliar, só para estimar o Q-alvo. É com esse objetivo que surge a Rede de Destino (*Target Network*).

A rede de destino é uma cópia da rede neural original, possuindo a mesma arquitetura, mas com seus pesos congelados durante a maior parte do tempo, mas de tempos em tempos atualizando eles com os pesos da rede original, isso faz com que o Q-alvo não mude constantemente a cada interação.

As redes de destino estabilizam o treinamento, pois mantém uma cópia da rede neural original, apenas para o cálculo do valor *Q(s',a')* da equação de Bellman, com isso os valores da rede de Q-alvo são usados como uma retro-propagação e treinamento da rede neural principal. Isso faz com que exista uma redução da correlação entre s e *s'* (58).

2.1.9.3 Tranfer Learning

Transfer Learning significa aprendizagem por transferência, e como o nome já diz, significa aplicar o conhecimento e informações adquiridas por modelos treinados, em outro problema diferente, mas relacionado. É um método de Aprendizado de Máquina popular, mais utilizado em aprendizado supervisionado, que faz com que modelos pré-treinados sejam utilizados como início em outros treinamentos (61). Com o crescimento do Aprendizado por Reforço, estudos já existem sobre a utilização de *transfer learning* nessa área (55).

Nesse trabalho será estudado um método de Aprendizado por Reforço para ser utilizado em tempo real, então a escolha pelo método de *transfer learning* é desejada, visto que esse método é bastante eficaz para acelerar o treinamento de modelos.

2.2 Teoria dos Grafos

A Teoria dos Grafos surgiu em 1736, após Leonhard Euler publicar um artigo sobre o problema das sete pontes de Königsberg. A cidade de Königsberg que ficava no território da antiga Prússia até 1945, tinha um complexo que continha sete pontes para ligar a cidade a duas grandes ilhas, como mostra a Figura 2.2, o problema das sete pontes discutia a possibilidade de realizar um trajeto que atravessasse todas as pontes do complexo, começando e terminando no mesmo local, porém, sem repetir nenhuma ponte. Euler, portanto, provou que era impossível realizar esse trajeto, pois não existia nenhum caminho para isso. Para provar sua teoria, Euler transformou as pontes em pontos e a interseção entre as pontes em linhas, formando assim o primeiro grafo registrado da história (32).

Apesar de Euler ter feito o primeiro registro de um grafo em 1736, apenas em 1936 foi publicado o primeiro livro didático sobre a teoria dos grafos, escrito por Dénes König (12). A teoria dos grafos é um ramo da matemática que estuda estruturas conhecidas como grafos e as relações entre objetos de determinados conjuntos (32).

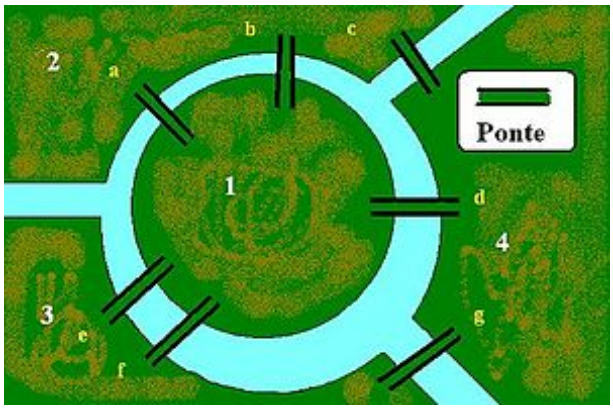

Figura 2.2: Sete pontes de Königsberg

Fonte: Wikipedia (42)

2.2.1 Conceitos básicos

Pode-se representar um grafo simples como $G = (V,A)$, onde V é considerado um conjunto não vazio de objetos conhecido como nós e A um subconjunto de pares não ordenados de V conhecido como arestas, portanto $A \subseteq P(V)$ e $P(V) = \{\{x, y\} : x,y \in V\}$. Cada aresta é composta por um par de nós distintos, sendo assim $x \neq y$ (33).

Dois nós são adjacentes se, e somente se, existir uma aresta x,y que pertença ao subconjunto A, caso um nó não seja adjacente a outro, pode-se considerar que se trata de um nó isolado.

Um grafo é considerado simples quando não possui múltiplas arestas entre o mesmo par de nós e/ou não possui uma aresta que conecta um nó a ele mesmo, conhecida como laço. Um grafo é considerado multigrafo quando possui arestas múltiplas, mas não laço, e pseudografo caso contrário (33).

Um exemplo de um grafo pode ser observado na Figura 2.3.

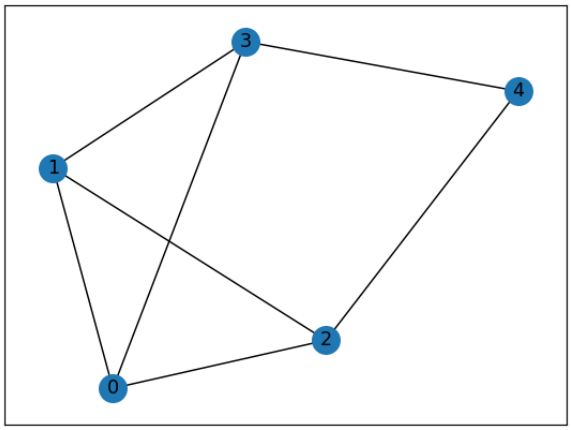

Figura 2.3: Exemplo de um grafo

Fonte: Autor

2.2.2 Matriz Adjacente

Uma forma de representar um grafo é a partir da matriz adjacente, dado um grafo $G = (V,A)$, onde o $|V|=n$ e $|A|=m$, por uma matriz simétrica $M=[m_{i,j}]$, de tamanho $n \times n$, que armazena informações sobre como os nós do grafo estão relacionados. Para essa representação do grafo simples, e não direcionado, é possível definir $m_{i,j}$ como mostra a Equação 2.14. Caso as arestas dos grafos tiverem pesos, quando i for adjacente a j, $m_{i,j} = p$, onde p é o peso entre as arestas (34).

$$m_{i,j} = \begin{cases} 0, \text{se i não é adjacente a j} \\ 1, \text{se i é adjacente a j} \end{cases} \tag{2.14}$$

A matriz de adjacência do grafo da Figura 2.3 pode ser definido como mostra a Equação 2.15.

$$A(G) = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.15}$$

2.2.3 Passeios

Se temos dois nós em um grafo, sendo eles v e w, o passeio entre eles é uma sequência alternada de nós e arestas, que começa em v e termina em w, então o passeio pode ser $v = v_0, a_1, v_1, ..., a_n, v_n = w$, tanto os nós quanto as arestas do passeio podem ser distintos ou não, caso todas as arestas forem distintas, o passeio é chamado de trilha, caso as arestas e os nós forem distintos então é chamado de caminho. Se o nós de origem for igual ao nó de destino, v=w, o passeio é fechado (35).

A teoria do grafo será importante no trabalho, pois as plantas baixas dos edifícios serão representados por grafos bidirecionais, e o algoritmo de Aprendizado por Reforço usará as informações da matriz adjacente para atribuir punições para seu agente.

Capítulo 3

Trabalhos Relacionados

Este capítulo trata dos trabalhos relacionados a utilização do Aprendizado por Reforço para busca de melhor caminho e a representação de uma planta baixa através de uma imagem.

3.1 Rotas de fuga de um edifício

Há muito tempo já vem sendo estudado como automatizar a procura por rotas de fuga em edifícios fechados, principalmente em casos de emergência, para uma rápida evacuação do local. Uma das técnicas que pode ser utilizada para analisar rotas de fuga, seja baseado no menor caminho ou no melhor caminho, considerando informações adicionais como obstáculos, previsão de espalhamento do fogo, ou outros critérios pré-estabelecidos, é o Aprendizado por Reforço, uma vertente do Aprendizado de Máquina supervisionado, onde com apenas poucas informações e um sistema de recompensa e punições é possível ensinar a máquina quais ações são necessárias selecionar para atingir a maior recompensa possível.

Utilizando Aprendizado por Reforço, em (59) foi proposto um sistema capaz de atualizar em tempo real as informações de rota de fuga em cenários de crime, utilizando o processo de decisão de Markov, *Q-learning* e um modelo em grafo, ajudando em casos de perseguição de criminosos, onde em ambientes desconhecidos, as melhores rotas de fuga também são desconhecidas.

Utilizando o motor da Unity game, em (41) foi apresentado uma análise sobre saída de emergências em um centro comercial através de simulações, nesse estudo foram comparados várias saídas de emergência do local, considerando até mesmos os obstáculos. Interessante observar que além da análise do caminho até a saída de emergência, eles conseguiram utilizar o perfil de vários consumidores do local para estimar a capacidade de movimentos dos mesmos, principalmente para as pessoas idosas e deficientes, conseguindo obter resultados e análises muito próximas à realidade.

Já existem estudos para a detecção automática de rotas de fuga em casos de incêndio, podendo ser utilizados algoritmos de busca de caminhos com Aprendizado de Máquina (5) e até mesmo informações do BIM (Building Information Model) e imagens do edifício em tempo real para decisões da melhor rota de fuga (13).

Nesse trabalho a forma escolhida para a representação da planta baixa de edifícios foi através de grafos e matrizes adjacentes, além da utilização do método de *transfer learning*, duas ideias propostas em (43), utilizando Aprendizado por Reforço e *Q-learning*, para a decisão em tempo real da melhor rota de fuga do ambiente. Ao contrário do encontrado em (43), esse trabalho propõe a consideração da distância entre os nós para o cálculo da recompensa retornada para o agente e a utilização do método de *Experience Replay* para o Aprendizado por Reforço Profundo.

3.2 Representação da imagem de uma planta baixa

Uma das formas mais rápidas e eficientes de utilizar o algoritmo de Q-learning para a busca do melhor caminho é utilizar como representação do ambiente um grafo e sua matriz adjacente, porém a maior dificuldade dessa abordagem é representar a imagem de uma planta baixa em um grafo, onde os cômodos serão os nós e as portas serão as arestas.

Umas das primeiras alternativas utilizadas para a análise e rotulagem de uma planta baixa é o processamento de imagem (40). Um dos caminhos é utilizar a análise pixel a pixel da imagem conhecida como bitmap, nessa técnica existe uma matriz de bits que especifica a cor de cada pixel em uma matriz retangular de pixels, o número de bits dedicados a um pixel individual determina o número de cores que podem ser atribuídos a esse pixel, podendo assim fazer o reconhecimento de cada cômodo de uma planta baixa e representar a mesma por um grafo (25).

Pode-se utilizar tecnologias já existentes para facilitar a análise, rotulagem e desenho de uma planta baixa, o mais popular é a utilização da tecnologia CAD (Computer-Aided Design), feita para a documentação técnica de um projeto, sendo muito útil para designers, desenhistas, arquitetos e engenheiros, através da combinação da análise dessa documentação técnica com técnicas de processamento de imagem é possível detectar vários componentes em uma planta baixa, como paredes, portas e janelas, com uma precisão alta e um tempo de processamento baixo, podendo ser utilizada em tempo real (54).

A segunda alternativa que já vem sendo estudada a alguns anos é a utilização de Aprendizado de Máquina para a extração de informações como cômodos, paredes, portas, janelas de uma planta baixa.

É possível transformar a imagem de uma planta baixa em um formato com dados vetoriais confiáveis extraindo informações do modelo de dados abertos para informações espaciais IndoorGML (Indoor Geography Markup Language) e o modelo de representação 3D CityGML (City Geography Markup Language), com com Redes Neurais Convolucionais (22).

Um grande projeto para a geração de grafos é o Graph2Plan, uma estrutura para automação de geração de plantas baixas usando aprendizado profundo. Através de uma interface interativa, o usuário consegue desenhar os limites de uma planta, informando a quantidade de cômodos, a localização e seus cômodos adjacentes, com essas informações é gerado um grafo para a representação dos cômodos, a partir do grafo e dos limites da planta, é possível gerar uma planta baixa completa com os cômodos, feito isso, o usuário consegue fazer ajustes na estrutura, personalizando a planta

como preferir. Após todo esse processo a planta baixa é finalmente vetorizada (20).

Existem dois grandes projetos conhecidos como CubiCasa5k (23) e CubiGraph5k (27), o primeiro consegue a partir de uma imagem de uma planta baixa, separar cada cômodo em polígonos, fazendo também a rotulagem e o segundo consegue a partir das anotações em polígonos gerar o grafo correspondente da planta baixa.

Em (24) foi feito uma análise sobre transformações de imagens escaneadas e rasterizadas para imagens digitalizadas, com base na geometria dos cômodos, utilizando Aprendizado de Máquina. Foram analisados três modelos CNNFloor, CubiCasa5k e Extração de geometria para estimativa de perda de calor, dos modelos o que mostrou maior confiabilidade para a detecção e rotulagem dos cômodos foi justamente o CubiCasa5k.

O CubiGraph5k utilizou a representação em polígonos das plantas baixas que estão no conjunto de dados do CubiCasa5k, para a criação de um algoritmo capaz de gerar representações gráficas dessas representações vetoriais das imagens (27).

Uma pesquisa promissora se baseia na utilização da Rede Neural Convolucional Mask R-CNN (19) para a avaliar imagens bitmap para segmentação de plantas baixas, visando a construção automática 2D de modelagem de informação da construção (BIM). Em seus resultados, com um conjunto de dados de aproximadamente quatro mil imagens, para plantas baixas com facilidade fácil e média, considerando a quantidade de cômodos, o resultado é bom, porém o resultado piora para imagens consideradas difíceis (38). O mesmo princípio é utilizado no pesquisa (15), onde já é utilizado para treinamento o próprio conjunto de dados gerado pelo CubiCasa5k.

Porém, a maioria dos projetos citados para representação das plantas baixas em forma de grafo, são focados em plantas baixas de casas, plantas simples, que possuem poucos nós e arestas. Pesquisas utilizando plantas baixas de edifícios grandes ainda não estão avançadas, pela complexidade do projeto. Por isso, pode-se construir manualmente um grafo para a representação de um edifício caso se tenha todas as informações do mesmo, um processo trabalhoso caso o edifício tenha muitos cômodos e consequentemente uma planta baixa mais complexa.

Capítulo 4

Metodologia e Implementação

Este capítulo trata dos caminhos utilizados para a resolução do problema proposto, bem como as implementações necessárias para o Aprendizado por Reforço Profundo e a representação de uma planta baixa em forma de um grafo.

Os Trechos de Código, utilizando a linguagem de programação *python*, referentes a implementação dos algoritmos de *Q-Learning* e Aprendizado por reforço profundo se encontram na seção Anexos I.1 e um código de exemplo na plataforma *Google Colab* pode ser encontrado em (37).

4.1 Algoritmo de Q-Learning

Com a imagem da planta baixa representada por um grafo e conhecendo todas as saídas disponíveis, é possível construir o primeiro algoritmo de aprendizagem por reforço utilizando o *Q-Learning* como visto na Subseção 2.1.8. Um exemplo do algoritmo de *Q-Learning* construído, pode ser encontrado no Algoritmo 2 (28).

Primeiramente, será necessário construir o grafo com os nós e arestas correspondentes, para que possa ser extraído sua matriz adjacente, que será utilizada como ambiente para o agente de aprendizagem. A matriz adjacente será composta de 0 e 1, conforme a Equação 2.14, pois não será utilizado pesos nas arestas. Após a inicialização do ambiente, é necessário inicializar a *Q-Table* com valores nulos, ela será uma matriz de tamanho número de nós x número de nós.

Algoritmo 2: Algoritmo de Q-Learning

```
   Entrada: Q(s,a)
 1 início
 2     para cada episódio
 3     faça
 4         Inicialize o estado inicial s
 5         repita
 6             Selecione uma ação a de acordo com o método de epsilon-greedy
 7             Execute a ação a e receba a recompensa r e o próximo estado s'
 8             Atualize a função Q(s,a) de acordo com s, s' e r
 9             s = s'
10         até s = estado final
11         ;
12     fim
13 fim
```

Nesse algoritmo é necessário o estado inicial do agente (o nó inicial), o estado final (o nó final) e se existe um ou mais nós que estão sendo atingidos por fogo e devem ser bloqueados, bem como definir o número de episódios ou épocas. Um episódio pode ser definido como uma sequência de estados, ações e recompensas, que termina no estado final definido, então vários episódios acontecerão para que o agente possa aprender o melhor caminho, como explicado na Subseção 2.1.5.3. Os valores de *gamma*, *alpha* e *delta* serão utilizados para o cálculo da Equação de Bellman e a atualização da *Q-table*, já o *epsilon* será utilizado para a decisão de exploração e intensificação.

A cada episódio, o estado inicial do agente será o nó indicado como inicial e o agente tomará ações até que o estado atual dele seja igual ao estado final indicado. Nesse algoritmo é utilizado os conceitos de exploração e intensificação explicados na Subseção 2.1.2, com o auxílio de um algoritmo conhecido como *epsilon-greedy*, explicado na Subseção 2.1.4.1. Caso o método de intensificação for escolhido, o agente irá escolher a próxima ação com base no máximo valor estimado, caso contrário, irá escolher uma ação aleatória entre todas as possíveis ações.

Se o próximo estado escolhido for um nó que está bloqueado (nó atingido por fogo), o agente receberá uma punição severa de -10, caso contrário a recompensa será 0 ou -1, dependendo da adjacência entre o estado atual e o próximo estado. Se o próximo estado for o estado final, o episódio então será encerrado e o agente terá alcançado seu objetivo.

Parar a atualização será utilizado a função de valor *Q-function* demonstrada a Equação 2.13 e devidamente explicada na Subseção 2.1.8.

Com o passar dos episódios, os valores da *Q-Table* irão convergir para os melhores valores possíveis e o agente, portanto, aprenderá o melhor caminho de um nó inicial informado, para um nó final também informado, apenas consultando sua *Q-Table*.

4.2 Aprendizado por Reforço Profundo

Após a construção de um algoritmo de *Q-Learning* para a análise de rotas de fuga utilizando um grafo como ambiente, foi proposto um algoritmo de Aprendizado por Reforço Profundo, podendo ser considerado um algoritmo de *Q-Learning* Profundo, para a otimização dessa busca. Esse algoritmo foi divido em três partes, a rede neural proposta, o ambiente e o agente.

Diferente do algoritmo de *Q-Learning* proposto, a rede neural consegue encontrar a melhor rota de fuga de todos os nós do grafo, para os nós de saída, não sendo obrigatório a informação de um nó inicial.

4.2.1 Rede Neural Proposta

Para esse problema foi implementado uma rede neural totalmente conectada, como mostra a Tabela 4.1, simples e fácil de utilizar, pois a representação dos dados são variáveis numéricas e não imagens ou textos, que necessitam de arquiteturas como Redes Neurais Convolucionais (CNN) ou Redes Neurais Recorrentes (RNN).

Tabela 4.1: Rede Neural Proposta

Name	Type	In Features	Out Features	Activation
F1	Linear	Tamanho de observação	256	ReLu
F2	Linear	256	128	ReLu
F3	Linear	128	64	ReLu
F4	Linear	64	Quantidade de ações	Linear

A arquitetura da rede consiste em 4 camadas, chamadas de camadas totalmente conectadas (*Fully Connected Layer*), essas camadas vão ajudar a alterar a dimensionalidade da saída da camada anterior para que esse modelo possa de forma fácil definir a relação entre os valores dos dados que o modelo está trabalhando.

As redes neurais utilizam transformações matemáticas lineares para processar os dados que recebe, portanto, independente da complexidade da rede neural que se utiliza, só serão utilizadas operações lineares entre as variáveis. Para eliminar essa dificuldade em processar transformações não lineares é utilizado as funções de ativação. Nesse trabalho foi escolhido a função de ativação não linear ReLu (*Rectified Linear Activation Function*), por ser computacionalmente leve, ela acelera o treinamento, pois retorna 0 para todos os valores negativos, como mostra a Equação 4.1, diminuindo a quantidade de neurônios durante o processo de *forward* (processo em que um neurônio transmite informação a outros neurônios da rede neural). Exceto a última camada, todas as outras terão a função de ativação ReLu.

$$f(x) = max(0, x) \tag{4.1}$$

A rede neural proposta para o algoritmo de Aprendizado por Reforço Profundo recebe dois

parâmetros de entrada, a quantidade de ações que o agente pode realizar dentro do ambiente, que será a quantidade de nós que existirem no grafo, e o tamanho de observação do ambiente que será definido como a quantidade de nós * 2, portanto o agente terá um espaço de observação parcial do ambiente, visto que o ambiente utilizado é uma matriz que possui um tamanho de quantidade de nós * quantidade de nós.

O algoritmo de otimização escolhido foi o *Adam*, utilizado para atualizar a rede de forma interativa com base nos dados de treinamento, sendo uma extensão do gradiente descendente estocástico. Já para o cálculo de perda, foi escolhido o método de MSE (Erro quadrático Médio).

A saída da rede será a quantidade de ações possíveis, ou seja, existirá um valor para todas as possíveis ações a se tomar dentro do ambiente e para a escolha da melhor ação deverá ser escolhido a com maior valor, assim como no algoritmo de *Q-Learning*.

4.2.2 Ambiente

A classe de ambiente seguirá a estrutura utilizada no OpenAI Gym (18), facilitando o entendimento do problema e possibilitando mudanças futuras. Essa estrutura consiste em 4 métodos principais, sendo eles *init*, *step*, *render* e *reset*, porém nesse trabalho não será utilizado o método de *render*.

O método de *init* consiste na inicialização das variáveis necessárias para o ambiente, esse método receberá o grafo referente a planta baixa, os nós de saída e os nós atingidos por fogo e a partir do grafo de entrada, será gerado a matriz correspondente, utilizando como peso entre as arestas a distância espacial entre os nós. A quantidade de ações possíveis a serem realizadas será definida como a quantidade de nós do grafo e o tamanho do espaço de observação será definido como a quantidade de ações possíveis * 2, isso indica que o agente não terá conhecimento completo do ambiente de observação, mas apenas de uma parte dele.

O método *reset* reinicializa o ambiente para as condições iniciais definidas. Esse método será responsável por definir o estado inicial do agente no ambiente de forma aleatória ou em um nó pré-definido, caso esse valor tenha sido informado. Ele recebe apenas um parâmetro não obrigatório para indicar o estado inicial. Utilizar o estado inicial de forma aleatória, faz com durante o treinamento o agente aprenda o caminho de todos os nós do grafo até o nós de destino, mas isso também torna o tempo de treinamento consideravelmente maior.

O método *step* será responsável pela parte essencial do algoritmo, definir a próxima ação a ser realizada, o próximo estado e a recompensa. Possui como entrada duas variáveis essenciais, o estado e a ação. Esse método será dividido em 2 partes para melhor entendimento.

A primeira parte do método *step* será responsável por através das variáveis de estado e ação, mapear o nó de próximo estado e o nó atual em que ele se encontra dentro do grafo, isso será necessário para o cálculo da recompensa retornada para o agente.

A segunda parte do *step* é responsável por definir a recompensa retornada para o agente por tomar determinada ação dado um estado atual. Com o conhecimento dos nós correspondentes ao

estado atual e do próximo estado, é simples calcular a recompensa retornada utilizando a matriz adjacente. Como não é informado a distância exata entre um cômodo e outro, os pesos das arestas dentro do grafo é utilizando para dar uma noção disso, calculando a distância espacial entre um nó e outro, no desenho do grafo informado pelo usuário, então quanto maior o valor do peso de uma aresta que conecta dois nós dentro de um grafo, maior a distância espacial entre esses dois nós, isso é levado em consideração pois o melhor caminho entre um nó e os nós de saída e aquele que evita os nós bloqueados e possui menor distância entre os nós.

As recompensas retornadas foram definidas da seguinte forma, caso o próximo estado foi um nó que está na lista de nós atingidos por fogo, o agente receberá uma recompensa severa de -10000, caso o nó atual e o nó do próximo estado não forem vizinhos, ou seja, não forem adjacentes entre si, o agente receberá uma punição de -5000, caso os nós sejam adjacentes, mas o nó do próximo estado não for um nó de saída, o agente receberá uma punição correspondente ao peso da aresta que conecta esses nós e caso o nó do próximo estado estiver na lista de nós de saída, o agente receberá uma recompensa positiva de 10000, indicando assim que ele concluiu seu objetivo com sucesso. Com os valores de recompensa é possível calcular a função de recompensa, podendo ou não ser utilizado algum fator de desconto.

4.2.3 Agente

O agente será responsável por inicializar a rede neural proposta, o ambiente e orquestrar o treinamento.

Para a inicialização do agente, é necessário informar o grafo referente a planta baixa, os nós que possuem saída, os nós que estão sendo atingidos pelo fogo e se será realizado o método de *transfer learning*, se sim, o peso que será utilizado. Algumas variáveis que também serão utilizadas no treinamento, necessitam ser inicializadas, sendo elas o *batchsize*, o tamanho do *buffer* de repetição, a quantidade de episódios, o *gamma*, o *learning rate*, o *beta* e o *epsilon*. Também é necessário a inicialização das variáveis necessárias para os algoritmos de *Experience Replay* e da Rede de Destino.

A primeira coisa a se fazer é a inicialização do ambiente, pois a partir dele poderá ser obtido o tamanho do espaço de observação e da quantidade de ações, variáveis necessárias para então iniciar a rede neural. Após a inicialização do ambiente e da rede neural é definido a inicialização do *buffer* de repetição.

O *buffer* de repetição faz parte da técnica de *Experience Replay*, explicado na Subseção 2.1.9.1, a experiência é representada por uma quádrupla de (estado atual, ação, recompensa, fim, novo estado) e pode ser inicializada antes do treinamento do agente com valores aleatórios de transições, isso significa que não será utilizado a rede neural para escolha das ações a se tomar, será sempre escolhido uma ação aleatória. Essa inicialização aleatória busca auxiliar no treinamento do agente, para que ele possa começar o treinamento com algum conhecimento prévio. Caso escolha inicializar esse *buffer* de repetição antes do treinamento, será definido pelo agente, um valor de quantas experiências prévias serão recolhidas, como 25% do tamanho total do *buffer* de repetição.

Após a definição sobre a inicialização do *buffer* de repetição com valores aleatórios, o ambiente e o agente estão preparados para começar o treinamento, o algoritmo de treinamento pode ser dividido em 2 partes, a primeira sendo a amostragem do ambiente, realizando ações e armazenando as quádruplas do *buffer* de experiência, a segunda onde será selecionado o mini lote do *buffer* de repetição aleatoriamente e a atualização das redes. Um exemplo do treinamento do agente utilizando Aprendizado por Reforço Profundo, *Experience Replay* e Rede de Destino, pode ser visualizado no Algoritmo 3.

Algoritmo 3: Algoritmo do Aprendizado por Reforço Profundo com Experiment Replay e Rede de Destino

Entrada: *D - Buffer de repetição, Q - rede neural, $\hat{Q}$ - Rede de Destino, A - agente*

início
  para *cada episódio*
  faça
    $\epsilon \leftarrow$ calcula um novo epsilon
    Escolha uma ação a dado um estado s usando o método de epsilon-greedy(Q)
    Execute a ação a e receba a recompensa r e o próximo estado s'
    Armazene a transição (s, a, r, s', *final*) no buffer de repetição D
    Amostre um lote de transições aleatórias B de D
    Passe B com os valores de estados (s) pela rede Q para calcular Q(s,a)
    Passe B com os valores dos próximos estados (s') pela rede $\hat{Q}$ para calcular $\hat{Q}(s',a)$
    Calcule a diferença temporal do destino TD-target $\leftarrow r + \gamma * max_a(\hat{Q}(s',a))$
    Calcule a perda MSE com o TD-Target e Q(s,a)
    Atualize a politica utilizada com o otimizador ADAM
    se *o espaço de tempo % tempo de atualização do destino == 0* **então**
      Atualize os pesos de Q e $\hat{Q}$
    fim
  fim
fim

A primeira parte do treinamento é semelhante ao que já foi desenvolvido no algoritmo de *Q-Learning*, é nessa parte que será utilizado os conceitos definidos na Subseção 2.1.2 de exploração e intensificação, a taxa de exploração será definida como o *epsilon* e será responsável por equilibrar às duas técnicas dentro do algoritmo. O *epsilon* é definido como mostra a Equação 4.2, portanto, para os primeiros episódios, a probabilidade de se escolher uma ação pelo método de exploração é muito alta, onde a ação será realizada de forma aleatória, porém vai diminuindo a medida que os episódios vão acontecendo, onde será priorizado o método de intensificação e a ação será realizada com base na rede neural. Essa primeira parte pode ser visualizada no Trecho de Código 8.

$$\varepsilon = exp(\frac{-i}{\frac{\text{numero de episodios}}{2}}) \tag{4.2}$$

A segunda parte do treino se refere ao método de *Experience Replay* e a Rede de Destino descrito nas Subseções 2.1.9.1 e 2.1.9.2. É nessa parte que será utilizado a variável *batchsize*, ela

indicará o tamanho do mini lote de amostras aleatórias do *buffer* de repetição, que será usado para atualizar a rede neural, também é utilizado a variável *gamma* que para o cálculo dos *targets* da Rede de Destino. Em tese, não se deve realizar a aprendizagem da rede enquanto não existir um valor pré-definido de tamanho do *buffer* de repetição, porém como o algoritmo necessita performar da forma mais rápido possível, será realizado a etapa de aprendizagem da rede em todos os episódios.

4.2.4 Inferência

A inferência do modelo de aprendizado de máquina, é uma forma de utilizar o modelo de aprendizado de máquina para processar dados de entrada ao vivo, gerando uma saída desejada, portanto, essa etapa só acontece após o treinamento do modelo.

Após o modelo de Aprendizado por Reforço ter sido treinado, é fácil realizar a inferência do mesmo, onde é necessário apenas o modelo treinado para realizar a inferência dos nós do grafo (exceto os nós de saída), ou de um nó inicial, para descobrir o melhor caminho para as saídas, aprendido pelo agente de aprendizagem.

4.3 Representação da imagem de uma planta baixa

O objetivo desse trabalho é a análise de rotas de fuga utilizando Aprendizado por Reforço, sendo assim, a representação automática da planta baixa de um edifício em forma de grafo, não faz parte do escopo desse trabalho. Portanto, a forma utilizada para a representação de uma planta baixa é manual e feita pelo usuário.

Para facilitar o envio da planta baixa do edifício em forma de um grafo e fazer a inferência do modelo de Aprendizado por Reforço Profundo, foi desenvolvido um interface web, utilizando o *framework Javascript* de código aberto Vue JS (56), uma API (Interface de Programação de Aplicação) utilizando o FastAPI, um *framework web* para desenvolvimento de APIs RESTful em Python (2) e o software de banco de dados orientado a documentos livre MongoDB (4). Essa interface possui duas funcionalidades principais, a criação de um projeto e a inferência de um projeto, como mostra a Figura 4.1.

4.3.1 Criação de um projeto

A criação de um projeto, é uma funcionalidade desenvolvida para que o usuário possa enviar a planta baixa do edifício em formato JPG, JPEG ou PNG e com o auxílio da biblioteca *v-network-graph* (3). É possível que o usuário desenhe os nós e arestas correspondentes aos cômodos e ligações entre os mesmos, sobre a imagem enviada, realizando a transformação da planta baixa do edifício em um grafo, como mostra a Figura 4.2.

Os nós sem saídas, possuem cor azul e indicam que não existe nenhuma saída do edifício naquele cômodo correspondente, os nós com saídas, possuem a cor verde e indicam que existe uma saída do edifício naquele cômodo correspondente. As arestas significam que os dois nós possuem conexão

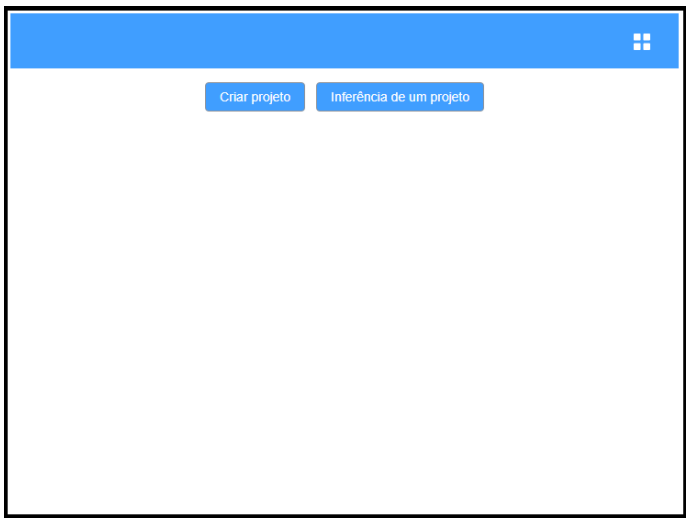

Figura 4.1: Pagina inicial da interface web

Fonte: Autor

direta, por portas, portais ou corredores, portanto, os dois nós serão vizinhos. Não é necessário considerar janelas como passagem, quando não se tem certeza se a mesma pode ser utilizada como uma passagem de um local a outro. É necessário que o nó esteja localizado o mais perto do centro possível, para poder se calcular a distância entre nós vizinhos, utilizando as coordenadas da imagem.

É importante que todas as informações contidas no grafo estejam corretas, como a quantidade de nós, arestas e principalmente os nós que possuem uma saída segura do edifício, pois o algoritmo de Aprendizado por Reforço só utilizará as informações cadastradas, não conseguindo fazer nenhum tipo de verificação para validação das informações.

Após a finalização da criação do grafo correspondente a planta baixa do edifício, é necessário preencher um formulário para cadastro das informações no banco de dados utilizado, indicando o nome do criador do projeto e o nome do projeto, para que ele possa ser localizado posteriormente no banco de dados, como mostra a Figura 4.3. Isso evita que toda vez que for necessário fazer a inferência de um projeto, o usuário tenha que criar novamente o grafo, visto que esse processo é trabalhoso e demorado.

As informações do projeto são salvas em um banco de dados não relacional MongoDB, conforme a Tabela 4.2, com essas informações é possível reconstruir o grafo para um formato em que o algoritmo de Aprendizado por Reforço entenda, bem como, possibilitar que o usuário possa ter acesso a esse projeto futuramente.

4.3.2 Inferência de um projeto

Primeiramente para realizar a inferência de um projeto, é necessário consultar no banco de dados todos os projetos criados por um usuário específico, como mostra a Figura 4.4. É possível selecionar um projeto para continuar o processo e realizar a inferência do mesmo, ou até mesmo

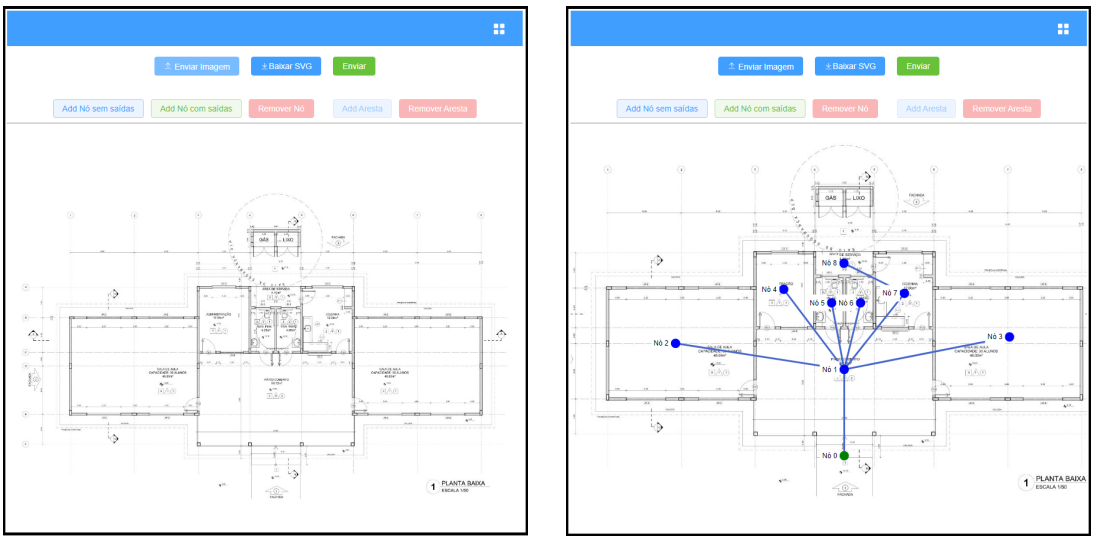

Figura 4.2: Criação de um grafo através de uma planta baixa

Fonte: Autor

Tabela 4.2: Informações salvas do projeto

Chave	Significado
_id	Identificador do documento
createdBy	O nome do criador do projeto
project	Nome do Projeto
nodes	Lista de informações dos nós (nome, tamanho e cor)
edges	Lista de informações das arestas (destino e origem)
graph	Objeto da classe vue-network-graph
fileID	Identificador do arquivo da imagem SVG
createdTime	Data e hora de criação do documento
model	Modelo pré-treinado do projeto

excluir um projeto que não é mais necessário.

Para realizar e inferência do projeto é necessário duas informações cruciais para o modelo de Aprendizado por Reforço, a informações de quais nós estão sendo atingidos por fogo e se é necessário realizar o treinamento considerando algum nó inicial específico. Realizar o treinamento considerando um nó inicial é mais rápido, porém só é retornado o caminho daquele nó específico para saída.

Com isso, como mostra a Figura 4.5, existe uma tabela onde é possível selecionar os nós que estão sendo atingidos por fogo e a necessidade de um nó inicial no treinamento. Após o modelo ter sido treinado, é apresentado em uma tabela o resultado de inferência para todos os nós do grafo, ou para um nó inicial. Essa tabela informa o nó inicial e o nó final, bem como todo o caminho que uma vítima deve percorrer até chegar a uma saída do edifício.

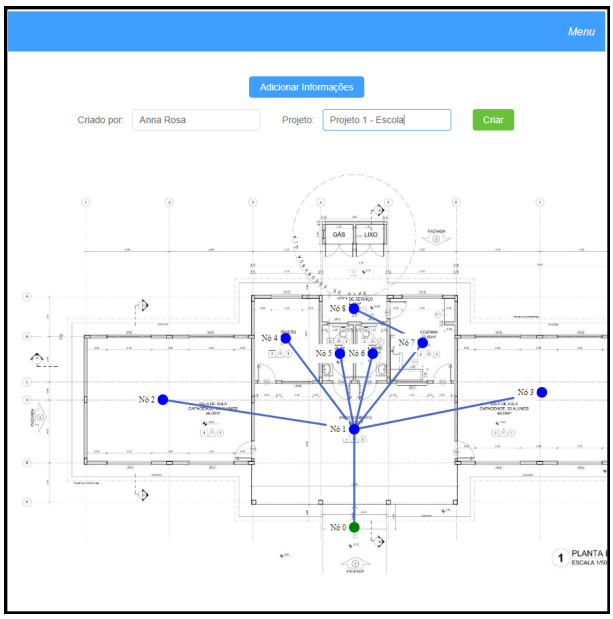

Figura 4.3: Formulário para criação de um projeto

Fonte: Autor

4.3.3 API (Interface de Programação de Aplicação)

Foi desenvolvido uma API (Interface de Programação de Aplicação) *RESTful* utilizando o *framework* FastAPI. *Rest* API é o conjunto de boas práticas utilizadas nas requisições HTTP (Protocolo de Transferência de Hipertexto) realizadas por uma API em uma aplicação web. No contexto do trabalho, a API será responsável por realizar a comunicação entre a interface web e o banco de dados.

Tabela 4.3: Rotas da API

Rota	Dados necessários no corpo da requisição	Significado
/create	nome do criador do projeto, nome do projeto, imagem em svg, nós do grafo, arestas do grafo, grafo	Salvar as informações de um projeto no banco de dados
/list	nome do criador do projeto	Listar projeto de um usuário
/delete/project	nome do criador, nome do projeto, data e hora de criação do projeto	Deletar um projeto específico
/inference	nome do criador, nome do projeto, data e hora de criação do projeto, nós bloqueados, nó de início	Realizar a inferência de um projeto

A API conta com quatro rotas que utilizarão o método de requisição POST. Como mostra a Tabela 4.3, cada rota tem um propósito específico para a comunicação com o banco de dados e para inferência dos projetos, com isso é necessário enviar no corpo da requisição HTTP algumas

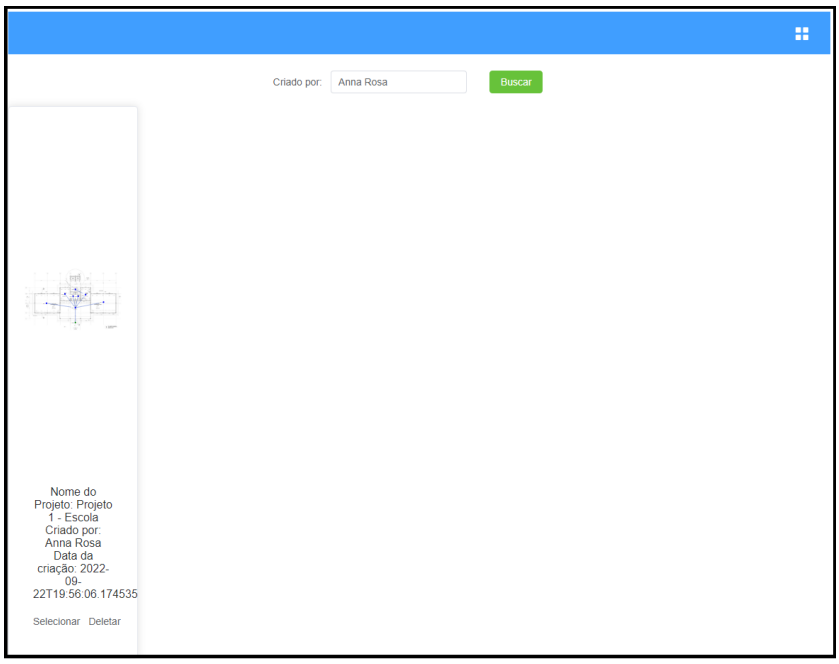

Figura 4.4: Projetos cadastrados para um usuário específico

Fonte: Autor

informações como nome do criador do projeto e nome do projeto.

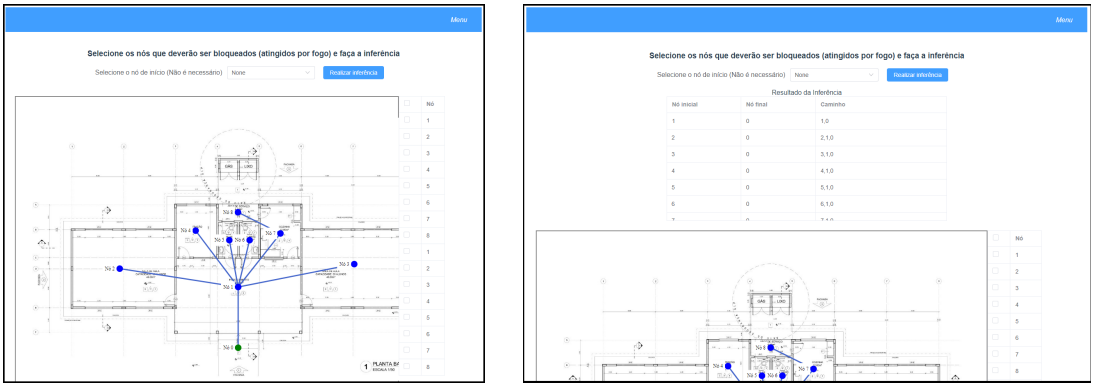

Figura 4.5: Inferência de um projeto

Fonte: Autor

Capítulo 5

Resultados

5.1 Visão Geral

Este capítulo trata dos resultados obtidos nos experimentos feitos com a rede neural construída para a busca do melhor caminho dentro de um grafo, utilizando Aprendizado por Reforço.

Para a análise, foram escolhidas 3 plantas baixas, referentes a escolas públicas, desenvolvidas e disponibilizadas pelo Fundo Nacional de Desenvolvimento da Educação, uma autarquia vinculada ao Ministério da Educação do Brasil (36).

A primeira planta baixa escolhida é apresentada na Figura 5.1, denominada nos experimentos como Projeto 1. Esse projeto se refere a um espaço educativo rural com 2 salas de aulas, destinado para a construção de escolas de um pavimento a serem implementadas em assentamentos ou pequenas comunidades rurais nas diversas regiões do Brasil, com capacidade para até 120 alunos. O projeto conta com 2 salas de aula, 1 pátio coberto, 1 administração, 2 banheiros e uma área de serviço, possuindo apenas uma saída.

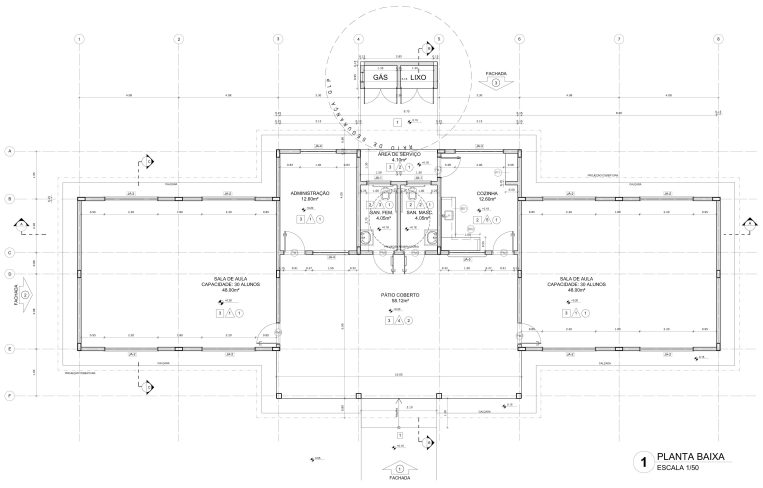

Figura 5.1: Planta Baixa do Projeto 1

Fonte: (7)

A segunda planta baixa escolhida é apresentada na Figura 5.2, denominada nos experimentos como Projeto 2. Esse projeto se refere a um espaço educativo rural com 4 salas de aula e uma quadra esportiva, também destinado a construção de escolas de um pavimento a serem implementadas em diversas regiões do Brasil, porém esse projeto tem capacidade para 240 alunos. O projeto conta com 4 salas de aula, 1 diretoria, 1 secretaria com arquivo, 1 almoxarifado, 1 sala de professores, 1 pátio coberto, 1 vestiário, 1 despensa, 1 cozinha, 1 área de serviço, 1 estacionamento, 1 bicicletário e 6 banheiros, possuindo duas saídas.

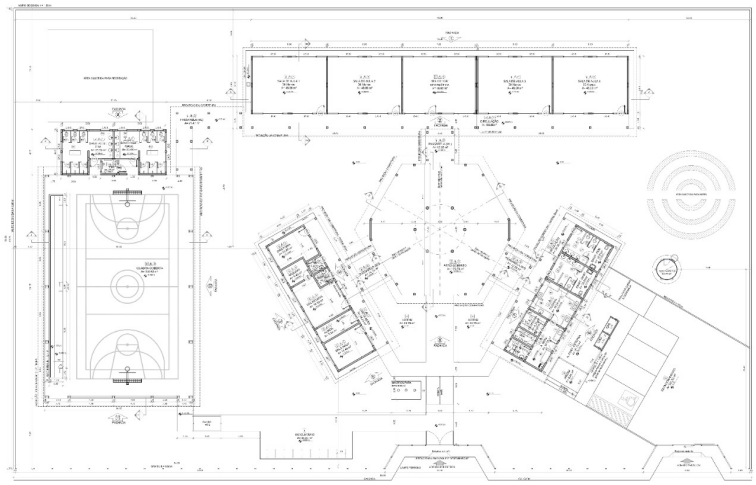

Figura 5.2: Planta Baixa do Projeto 2

Fonte: (8)

A terceira planta baixa é apresentada na Figura 5.3, denominada nos experimentos como Projeto 3. Esse projeto se refere a um espaço educativo rural com 6 salas de aula e uma quadra esportiva, também destinado a construção de escolas de um pavimento a serem implementadas em diversas regiões do Brasil, porém esse projeto tem capacidade para 360 alunos. A única diferença para a planta baixa do Projeto 2, é a adição de 2 salas de aula.

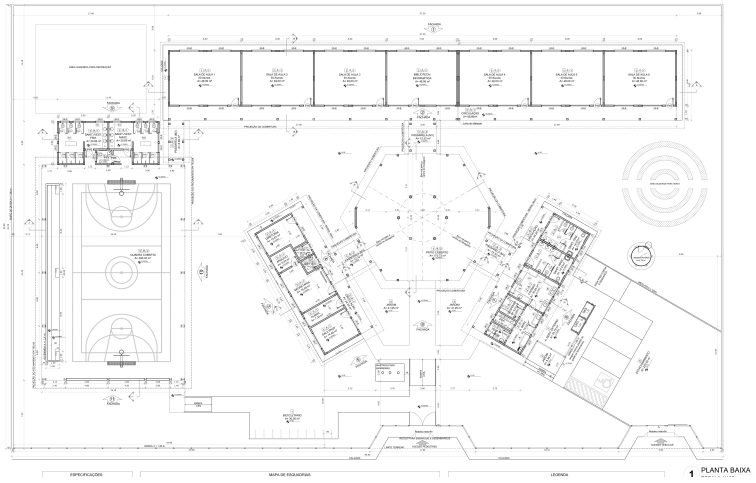

Figura 5.3: Planta Baixa do Projeto 3

Fonte: (9)

A partir das imagens dessas plantas baixas é possível utilizar a interface web desenvolvida para transformar as informações em grafos, podendo assim utilizar o algoritmo de Aprendizado por Reforço Profundo para encontrar as melhores rotas de fuga dos edifícios.

Com as informações dos grafos, é enviado para o algoritmo de Aprendizado por Reforço Profundo apenas os nós, arestas e a localização dos nós. Também é necessário enviar as informação de quais nós são referentes as saídas do edifício e quais nós podem estar sendo atingidos por fogos e, portanto devem ser bloqueados, sendo assim, possível reconstruir o grafo e extrair sua matriz adjacente.

5.1.1 Distância entre dois nós

Uma forma simples encontrada para calcular a distância entre dois nós, para que no melhor caminho até a saída também seja considerado a distância, foi utilizar a distância entre dois pontos em um plano. Como ao desenhar os nós na imagem o que está sendo feito é marcar em um plano a coordenada x e coordenada y de um objeto, podemos considerar essas coordenadas para o cálculo da distância.

Para calcular a distância entre dois pontos, é necessário primeiramente obter as coordenadas desses pontos. Nesse trabalho é simples extrair essas informações, pois ao desenhar o grafo já é retornado as coordenadas de todos os nós. Através dessas coordenadas é possível calcular o comprimento do segmento de reta que liga esses dois pontos, utilizando o Teorema de Pitágoras. Portanto, se existir um nó A e um nó B, onde A(x_a, y_a) e B(x_b, y_b), a distância entre esses nós pode ser calculada conforme a Equação 5.1. É dessa forma que é definida a distância entre os nós, para poder ser retornada a recompensa para o agente.

$$d_{AB} = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} \tag{5.1}$$

5.1.2 Definição das variáveis

A quantidade de episódios deve ser definida conforme os objetivos do problema, para o treinamento do agente e considerando bons resultados, iniciando de um nó específico, é necessário menos episódios do que o treinamento de todos os nós do grafo para os nós de saída. Portanto, a definição da quantidade de episódios necessários foi definido conforme a Tabela 5.1. Um dos fatores mais importantes para a definição dos episódios é a quantidade de arestas existentes no grafo, pois quanto maior a quantidade de caminhos possíveis para serem realizados pelo agente, maior a complexidade do problema. Uma das alternativas que diminui a necessidade de grandes quantidades de episódios é a utilização do método de *transfer learning*, pois a rede já terá pesos pré-treinados, facilitando o aprendizado do agente.

Para todos os experimentos foi definido o tamanho de 10^6 para o *buffer* de repetição, assim como recomendado na literatura. O *beta* foi definido como 1, o *learning rate* como 1e-2, o tamanho da amostra do *buffer* de repetição (*batchsize*) como 240, o *gamma* como 0.99 e o *epsilon* como 0.5.

Tabela 5.1: Definição da quantidade de episódios

Condição	Quantidade de episódios
Existe um nó de início	(Quantidade de arestas) * 100
É utilizado transfer learning	(Quantidade de arestas/3) * 500
A quantidade de arestas é maior que 40	(Quantidade de arestas) * 900
Para os demais casos	(Quantidade de arestas) * 500

Essas variáveis podem ser alteradas de acordo com a necessidade e do problema, mas para fins de comparação foram mantidas constante durante os experimentos.

5.1.3 Avaliação dos treinamentos

Foram escolhidos três métodos para o treinamento do agente, sendo eles, Aprendizado por Reforço com *Experience Replay* com e sem a inicialização do *buffer* de repetição com valores aleatórios e o método de Aprendizado por Reforço com *Experience Replay* sem a inicialização do *buffer* de repetição, mas com *transfer learning*. Esses três métodos estão devidamente explicados na Subseção 2.1.9. Portando os experimentos estarão sempre divididos em três etapas.

Para a avaliação dos treinamentos foi utilizado duas métricas, a média da recompensa acumulada ao longo dos episódios e a perda MSE (Erro quadrático médio), além da observação dos melhores caminhos indicados pelo algoritmo.

5.1.4 Infraestrutura Utilizada

Todos os experimentos detalhados nesse trabalho foram feitos utilizando uma máquina com processador AMD Ryzen 5 3500u, com uma placa de vídeo integrada NVIDIA GeForce MX250 e uma memória RAM DDR4 de 8gb e 2400MHZ.

5.2 Projeto 1

O Projeto 1 possui 9 nós e 8 arestas bidirecionais, como mostra a Figura 5.4, onde só existe uma saída, que está representada pelo Nó 0, o único nó que possui a cor verde no grafo da imagem. Esse projeto pode ser considerado simples, pois com exceção do Nó 8, todos os nós possuem caminhos curtos para o nó de saída. Para grafos simples não seria necessário a utilização de uma rede neural para a análise das rotas de fuga, mas a fim de comparação, será analisado nesse trabalho.

5.2.1 Projeto 1 sem bloqueios

Neste primeiro experimento simulado, não existe nenhum cômodo do edifício sendo atingido pelo fogo, por isso não existe bloqueio entre os nós do grafo e o nó de saída. Esse experimento

será focado apenas em analisar o tempo de treinamento do agente e os métodos propostos para encontrar o caminho de todos os nós do grafo para o Nó 0, o único nó de saída.

Primeiramente é necessário reconstruir o grafo e extrair sua matriz adjacente, isso se torna simples com os nós, arestas e localização informadas e armazenadas pelo usuário, portanto a matriz adjacente pode ser visualizada na Equação 5.2. É importante lembrar que os pesos das arestas correspondem a distância entre um nó ao outro, como mostra a Equação 5.1.

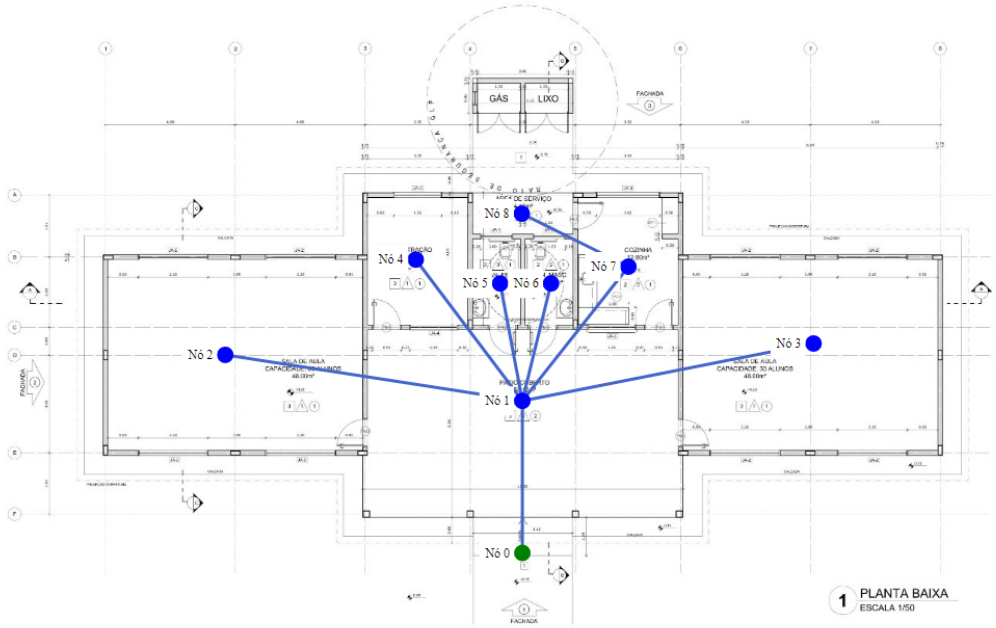

Figura 5.4: Grafo do Projeto 1

Fonte: Autor

$$A(G) = \begin{bmatrix} 0 & 30 & 162 & 175 & 128 & 106 & 106 & 134 & 0 \\ 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 162 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 175 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 128 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 106 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 106 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 134 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 61 & 0 \end{bmatrix} \quad (5.2)$$

Utilizando o Aprendizado por Reforço Profundo com a técnica de *Experience Replay*, é necessário decidir a inicialização ou não do *buffer* de repetição com valores aleatórios. Esse primeiro experimento é uma comparação entre os dois métodos sugeridos. Para os dois experimentos realizados, foi utilizado 4000 episódios.

O tempo de treinamento do agente quando não se inicializa o *buffer* de repetição foi de 33 segundos, e o tempo de resposta da API para a interface web com o resultado foi de 36 segundos. Para o treino com inicialização aleatória do *buffer* de repetição, o tempo de treinamento inicialização do *buffer* foi de 0,7 segundos, o tempo de treinamento do agente foi de 39 segundos e o tempo de resposta da API para a interface web com o resultado foi de 49,35 segundos.

O resultado para a recompensa acumulada, bem como o resultado da perda pode ser visualizado na Figura 5.5. Analisando os resultados, percebe-se que não existe nenhuma diferença significativa entre a inicialização com valores aleatórios ou não do *buffer* de repetição, visto que para os dois experimentos, a recompensa acumulada começa a convergir a partir do episódio 2500. Portanto, para grafos pequenos, a única diferença entre esses dois métodos está no tempo de inferência, visto que a não inicialização do *buffer* com valores aleatórios é mais rápida, apesar de a diferença ser pequena. Nesses experimentos foram encontrados 100% de todas os caminhos possíveis do ambiente.

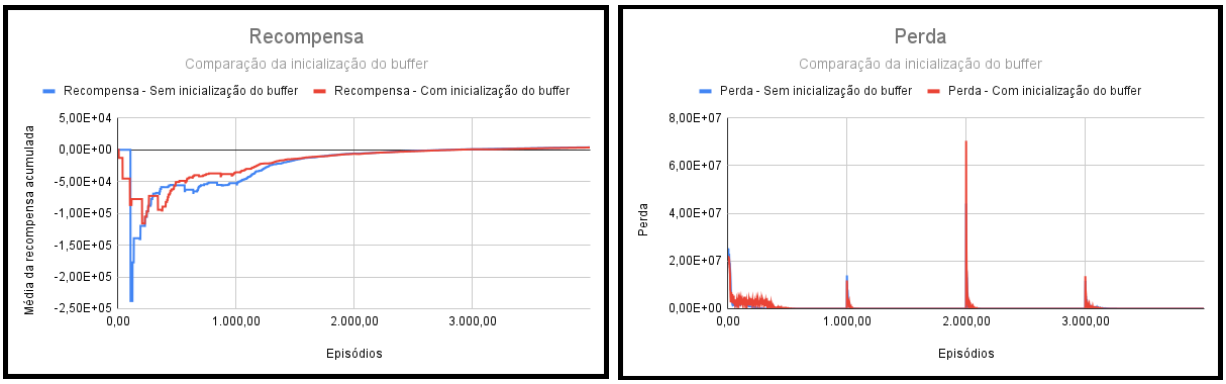

Figura 5.5: Gráfico de Recompensa e Perda para o Projeto 1 - Sem bloqueios
Fonte: Autor

5.2.2 Projeto 1 com bloqueios

Existe apenas um nó do grafo que possui um caminho com tamanho maior que 3 para o nó de saída, o Nó 8, pois o único caminho para o nó de saída será *Nó 8, Nó 7, Nó 1, Nó 0*. Para os demais n nós do grafo sempre existirá um caminho N_n*, Nó 0, Nó 1*. Portanto, caso ocorra um incêndio no *Nó 7*, não existirá caminhos seguros de um nó do grafo para a saída. Foi realizado um experimento a fim de verificar o que o agente aprenderia caso o nó bloqueado fosse o Nó 7.

Os três experimentos foram feitos com o bloqueio do Nó 7, *Experience Replay* sem inicialização aleatória do *buffer* de repetição, *Experience Replay* com inicialização aleatória do *buffer* de repetição e utilizando o método de *Experience Replay* e *transfer learning*. Para o método de *transfer learning* foi utilizado o peso obtido no experimento anterior da Subseção 5.2, sem a inicialização aleatória do *buffer* de repetição, portanto o agente começará utilizando um peso pré-treinado que se assemelha ao cenário atual, exceto pelo bloqueio de apenas um nó.

Apenas para o método de *Experience Replay* com inicialização aleatória do *buffer* de repetição, o agente não consegue aprender os caminhos dos nós para o nó de saída quando o Nó 7 é bloqueado,

possuindo assim uma taxa de precisão de caminhos de 0%. Para os demais casos, o agente consegue aprender o caminho da maioria dos nós para o nó de saída, com exceção do Nó 8, que não possui nenhum caminho seguro para a saída, então não é possível e nem recomendado indicar um possível caminho, portanto foi encontrados 86% do total de caminhos e 100% dos caminhos seguros do ambiente. É interessante notar que é utilizado apenas 1400 episódios para o método de *transfer learning*, que possui um tempo de treinamento de 12 segundos e o tempo de resposta para a interface web de 13 segundos, isso faz com que o tempo de treinamento diminua pela metade, mantendo o resultado obtido treinando o agente sem nenhum peso pré-treinado.

Um último experimento foi realizado para fins de comparação, foi simulado um incêndio no Nó 4 e Nó 5, a fim de verificar o quanto dois bloqueios afetam no treinamento do agente. Para a técnica de *Experience Replay* sem inicialização aleatória do *buffer* de repetição foi necessário 51 segundos para treinamento e 57 segundos para a resposta ser recebida na interface web, já com a inicialização aleatória do *buffer*, foi necessário 1 minuto para o treinamento e 1 minuto e 3 segundos para a resposta ser recebida na interface web, com os dois métodos anteriores utilizando 4000 episódios. Para a técnica de *Experience Replay* sem inicialização aleatória, mas com *transfer learning*, foi utilizado 1500 episódios, com um tempo de treinamento de 16 segundos. A média da recompensa acumulada e da perda para esse experimento se encontram na Figura 5.6, onde em todos os métodos, foi observado uma taxa de precisão dos caminhos do ambiente de 100%.

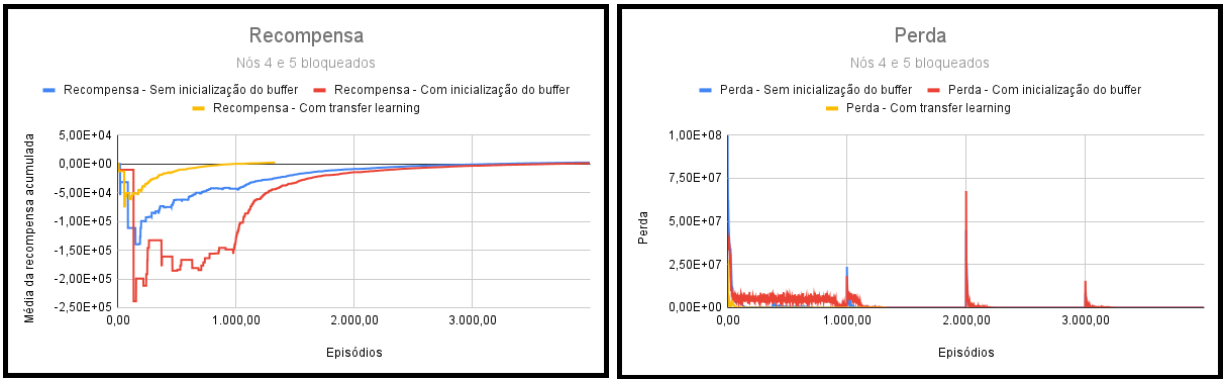

Figura 5.6: Gráfico de Recompensa e Perda para o Projeto 1 - Com bloqueios
Fonte: Autor

Observando apenas a Figura 5.6, fica claro como a técnica utilizando *transfer learning* tem resultados superiores as demais, necessitando de menos episódios para a aprendizagem do agente e com isso o treinamento se torna mais rápido. Fazendo uma análise comparativa entre os experimentos com e sem bloqueio, percebe-se que não existiu diferença significativa para os demais métodos, visto que nos dois cenários, a aprendizagem do agente começa a convergir a partir do episódio 2500.

5.3 Projeto 2

O Projeto 2 possui 31 nós e 34 arestas bidirecionais, como mostra a Figura 5.7, onde existem duas saídas, que são representadas pelos Nó 0 e Nó 19, representados pela cor verde. O grafo do

Projeto 2 já possui maiores dificuldades em relação ao grafo do Projeto 1, por ter uma quantidade significantemente maior de nós e arestas. Nesse projeto, não serão considerados as janelas como passagem.

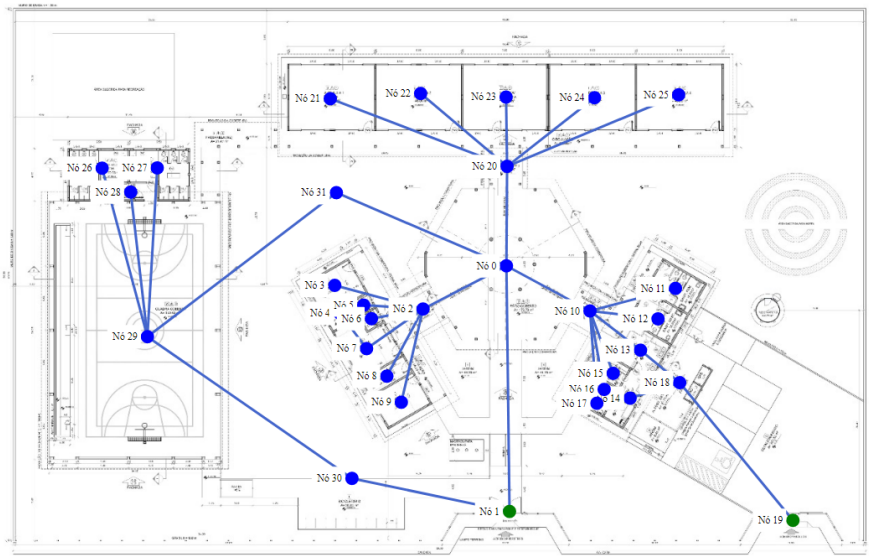

Figura 5.7: Grafo do Projeto 2

Fonte: Autor

5.3.1 Projeto 2 - sem bloqueios

Nesse primeiro experimento simulado para o Projeto 2, não existe nenhum cômodo do edifício sendo atingido pelo fogo, por isso não existe nenhum bloqueio entre os nós do grafo e os nós de saída. Assim como na Subseção 5.2.1, esse primeiro experimento será focado em analisar o tempo de treinamento do agente para encontrar o caminho de todos os nós do grafo para os nós de saída.

Nesse experimento será feito uma comparação entre utilizar a técnica de Aprendizado por Reforço Profundo com *Experience Replay* com a inicialização aleatória do *buffer* de repetição ou sem a inicialização aleatória do *buffer* de repetição. Para os dois métodos, foi utilizado o valor de episódios como 17000.

Para o experimento sem a inicialização com valores aleatórios do *buffer* de repetição, o treinamento durou 2 minutos e 53 segundos, com o tempo de resposta para a interface web de 3 minutos e 10 segundos, já com a inicializando o *buffer* de repetição com valores aleatórios, o tempo para a inicialização do *buffer* foi de 0,6 segundos, o treinamento durou 2 e 39 segundos minutos e o tempo de resposta para interface web foi de 3 minutos. Os gráficos de média de recompensa acumulada e perda podem ser visualizados na Figura 5.8.

Analisando a Figura 5.8, fica claro como a inicialização aleatória do *buffer* de repetição pode acabar prejudicando o aprendizado do agente, pois mesmo com 17000 episódios o agente conseguiu aprender apenas um caminho dentro de todo o grafo, obtendo uma taxa de precisão de caminhos de 0%, enquanto sem a inicialização aleatória, foi necessário apenas 7000 episódios para um bom

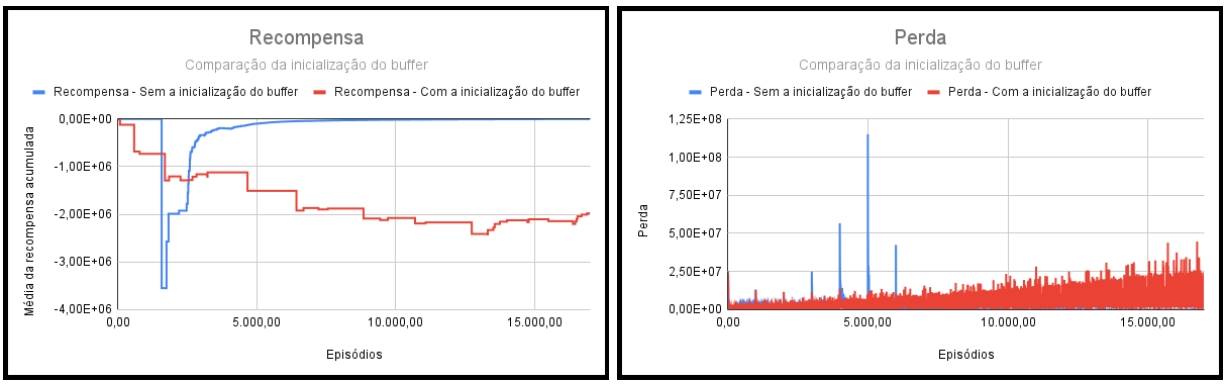

Figura 5.8: Gráfico de Recompensa e Perda para o Projeto 2 - Sem bloqueios

Fonte: Autor

aprendizado do agente, com uma taxa de precisão de 100% dos caminhos possíveis.

A inicialização aleatória pode atrapalhar o aprendizado, pois não existe controle de quais transições são realizadas e ao utilizar as amostras do *buffer* de repetição para a atualização da rede neural, pode ser escolhidas transições ruins, feitas pela inicialização aleatória, confundindo o agente e atrapalhando na aprendizagem. Portando, a inicialização aleatória do *buffer* de repetição, se mostrou ineficaz para o aprendizado por reforço do agente de treinamento.

5.3.2 Projeto 2 - com bloqueios

Assim como feito na Subseção 5.2.2, será analisado qual impacto existe no aprendizado do agente, caso existam nós bloqueados dentro do ambiente, porém, agora em um ambiente maior e mais complexo.

O primeiro experimento a ser feito será bloqueando o Nó 2, pois caso isso aconteça, não existirão caminhos seguros dos Nós 3, 4, 5, 6, 7, 8 e 9 para as saídas. Foram utilizados 17000 episódios para os métodos sem *transfer learning* e 5666 para o método com *transfer learning*. Diferente do resultado obtido no Projeto 1, nenhum dos métodos foi capaz de encontrar as saídas dos nós do grafo para os nós de saída, possuindo uma taxa de precisão de caminhos dentro do ambiente de 0%, isso porque, para muitos nós, a recompensa se torna extremamente baixa, independente do caminho que seja escolhido, prejudicando o aprendizado do agente. Portanto, para esses casos, não devem ser considerados os caminhos previstos pelo algoritmo e se necessário, pode ser utilizado o treinamento do agente sem bloqueios.

Para a análise de bloqueios dentro do grafo, foi feito um experimento, bloqueando os Nós 11, 12, 13, 14, 15, 16 e 17, simulando um incêndio nesses cômodos. Novamente foi utilizado 17000 episódios para os métodos sem *transfer learning* e 5666 para o método com *transfer learning*. Para o método de *Experiment Replay* sem a inicialização aleatória do *buffer de repetição*, foram necessários 2 minutos e 40 segundos para o treinamento do agente e 3 minutos para o recebimento do resultado na interface web, já para o treinamento com a inicialização aleatória, foram necessários 0,6 segundos para a inicialização do *buffer*, 2 minutos e 38 segundos para o treinamento do agente e 3 minutos

para o recebimento do resultado na interface web. Com o método de *Experience Replay* sem a inicialização aleatória do *buffer* de repetição, mas com *transfer learning*, foi necessário apenas 52 segundos para o treinamento do agente, menos da metade do tempo necessário nos outros métodos.

Analisando a Figura 5.9 da média da recompensa acumulada e da perda, quando os Nós 11, 12, 13, 14, 15, 16 e 17 estão bloqueados, percebe-se que assim como no experimento do Projeto 1, a técnica de se inicializar o *buffer* de repetição com valores aleatórios continua retornando resultados ruins e não desejados, com uma taxa de precisão de caminhos de apenas 7%. Novamente a técnica que utiliza *transfer learning* se destaca, precisando de apenas 4000 episódios para convergir, possuindo um tempo de treinamento inferior a 1 segundo e uma precisão de caminhos de 100%.

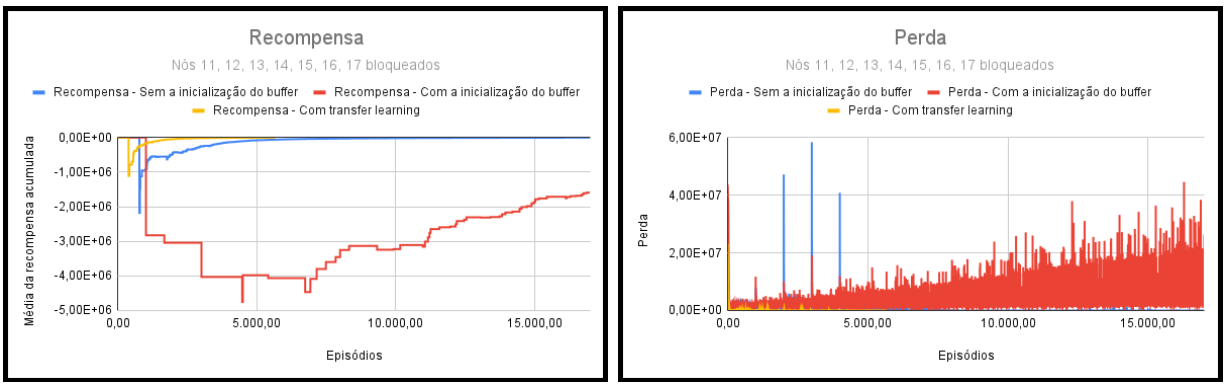

Figura 5.9: Gráfico de Recompensa e Perda para o Projeto 2 - Com bloqueios
Fonte: Autor

Portanto, foi comprovado que a inicialização com valores aleatórios do *buffer* de repetição não é um bom método para esse problema, e será retirada dos experimentos futuros, a fim de entender melhor como a técnica de *transfer learning* pode ser utilizada em Aprendizado por Reforço.

5.4 Projeto 3

O Projeto 3 possui 34 nós e 46 arestas, como mostra a Figura 5.10, diferente do Projeto 2, esse grafo foi desenhado com mais detalhes, considerando algumas janelas como passagem, aumentando consideravelmente o número de arestas no grafo, portanto, apesar do projeto só ter a adição de 2 nós, comparando com o Projeto 2, possui 12 arestas a mais. O Nó 0 e o Nó 21 são os nós de saída do grafo.

5.4.1 Projeto 3 - sem bloqueios

Nesse primeiro experimento simulado para o Projeto 3, não existem nenhum cômodo do edifício sendo atingido por fogo, por isso, não existe nenhum nó bloqueado dentro do ambiente. Como a técnica de inicialização aleatório do *buffer* de repetição não se mostrou eficaz, é necessário apenas treinar o agente para descobrir a saída de todos os nós do grafo, para os nós de saída.

Para isso, foram necessários 41400 episódios para uma precisão de 100% dos caminhos, com

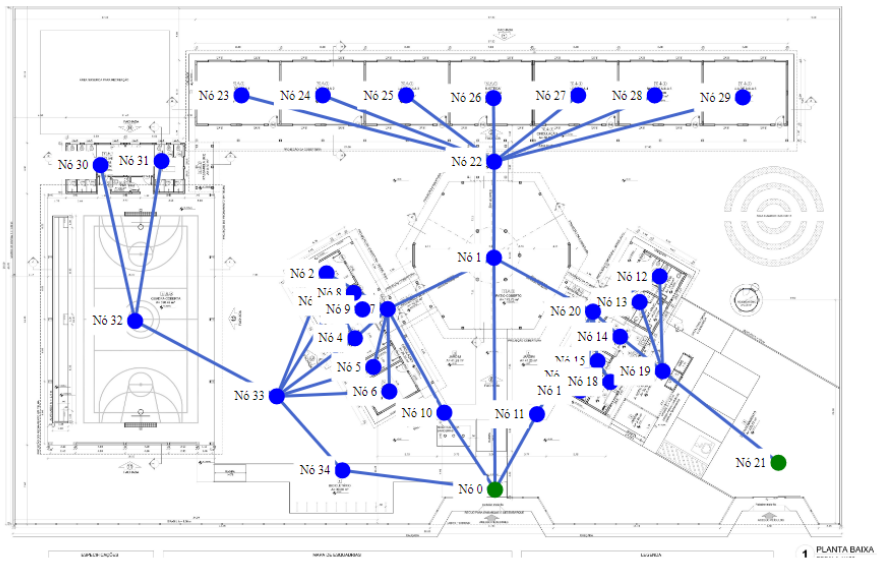

Figura 5.10: Grafo do Projeto 3

Fonte: Autor

uma duração de treinamento de 7 minutos e 14 segundos e 7 minutos e 30 segundos para o resultado ser recebido pela interface web, sendo assim, um acréscimo grande no tempo em comparação ao Projeto 2, para um grafo que possui 2 nós e 14 arestas a mais. Isso deixa claro como a quantidade de arestas e nós influenciam o treinamento do agente, sendo necessário escolher quantidades de episódios grandes, para que o agente consiga aprender todos os caminhos necessários.

Com esse treinamento, pode-se utilizar os pesos obtidos para os treinamentos com *transfer learning*.

5.4.2 Projeto 3 - Com bloqueios

O primeiro experimento será o bloqueio do Nó 32, sendo assim, não existirão caminhos seguros para os Nós 30 e 31. Sem a utilização de *transfer learning*, foi utilizado 41400 episódios, sem sucesso, pois o agente não consegue aprender nenhum caminho dentro do grafo, obtendo uma precisão de caminhos de 0%. Já utilizando o método de *transfer learning*, com 7666 episódios e um tempo de treinamento de 1 minuto, foi possível descobrir o caminho (com exceção dos Nós 30 e 31), de quase todos os nós do grafo, não sendo descoberto o caminho do Nó 18 para a saída, possuindo uma precisão de caminhos de 90%. Isso demonstra que com o método de *transfer learning*, apesar de ter uma precisão pior do que o esperado, ainda assim é o método superior.

Bloqueando o Nó 22, assim como feito no Projeto 2, o método de *transfer learning* continua sendo superior, com exceção dos Nós 23, 24, 25, 26, 27, 28 e 29 que de fato não possuem um caminho seguro para a saída, não foi possível encontrar o caminho dos Nós 11 e 34, possuindo 90% de precisão dos caminhos seguros, enquanto o método sem *transfer learning* não encontra nenhum caminho, possuindo 0% de precisão dos caminhos.

Depois de realizado diversos treinamentos, bloqueando um nó dentro do grafo que faz com que um ou mais nós não possuam caminhos seguros até as saídas, tem-se que o resultado não é confiável de se utilizar, considerando que ocorre uma falha no treinamento do agente, mesmo utilizando o método de *transfer learning*. Podendo ser indicado um nó inicial, para compensar esse erro.

Bloqueando o Nó 22, mas indicando um nó inicial, como o Nó 2, é possível encontrar de forma rápida e precisa o caminho. Para esse experimento (Nó 22 bloqueando e Nó 2 como inicial) foram utilizados 2300 episódios para ambos os métodos. Sem a utilização de *transfer learning*, o treinamento durou 21 segundos e com *transfer learning* o treinamento durou 20 segundos. Observando os gráficos da média da recompensa acumulada e da perda, na Figura 5.11, é a primeira vez nos experimentos que não se observa diferenças significativas entre os métodos com e sem *transfer learning*, pois os dois começam a convergir a partir do episódio 1500, isso acontece pois o modelo pré-treinado foi treinado para todos os nós do grafo e não para um específico.

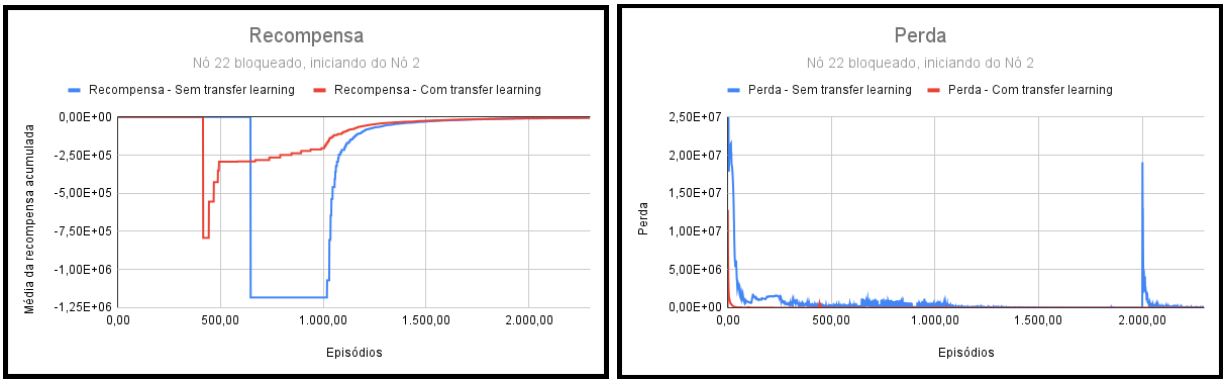

Figura 5.11: Gráfico de Recompensa e Perda para o Projeto 3 - Com um nó inicial

Fonte: Autor

Por fim, a fim de verificar a eficácia da aprendizagem para vários nós bloqueados, foi realizado um experimento bloqueando os Nós 4, 5, 6 e 10. Para o treinamento do agente sem *transfer learning* foi utilizado 41400 episódios, com o tempo de treinamento de 10 minutos e 21 segundos, já com *tranfer learning* foi utilizado 7666 episódios, com o tempo de treinamento de 1 minuto e 48 segundos. Analisando a recompensa acumulada e a perda para esse experimento, como mostra a Figura 5.12, fica bem claro como o método de utilizar *transfer learning* auxilia e acelera o treinamento do agente. Os dois métodos retornaram uma precisão dos caminhos do ambiente de 100%.

5.5 Aplicação Final

Após todo os experimentos, foi possível finalizar a aplicação final, que possui duas principais funcionalidades, a criação de um projeto e a inferência de um projeto, funcionando conforme a Figura 5.13, onde serão utilizados todos os conceitos e códigos desenvolvidos ao longo de todo o trabalho. É possível visualizar a utilização da interface web, da API com o código de Aprendizado por Reforço e o banco de dados MongoDB.

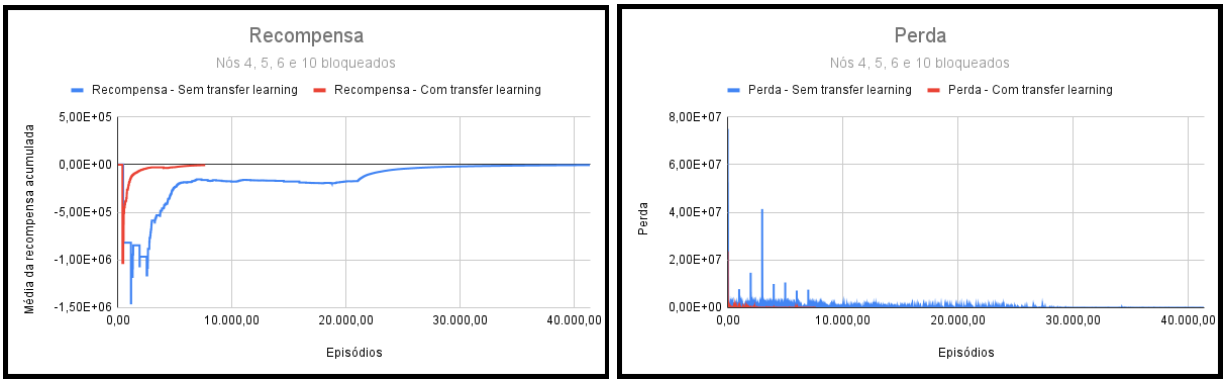

Figura 5.12: Gráfico de Recompensa e Perda para o Projeto 3 - Com bloqueios

Fonte: Autor

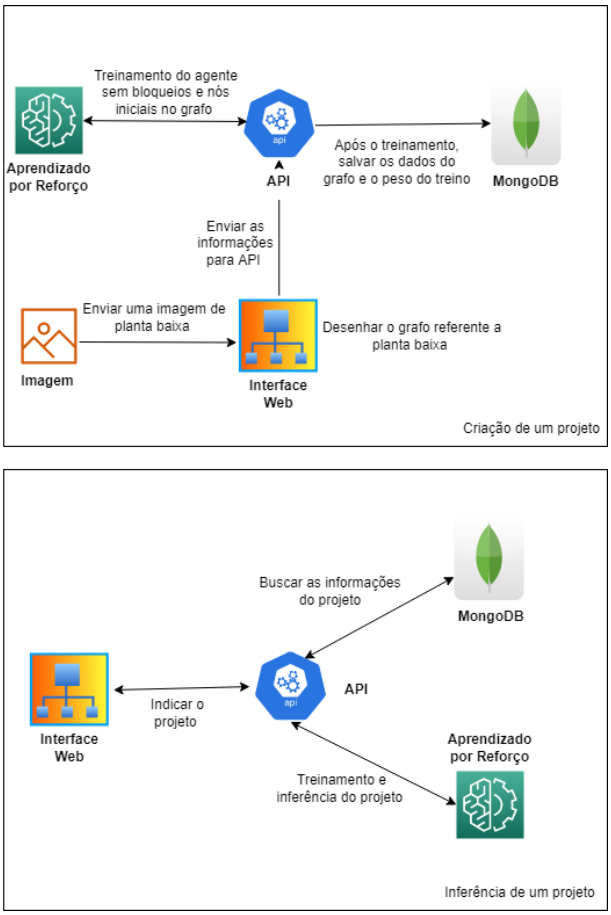

Figura 5.13: Diagrama da aplicação final

Fonte: Autor

Como demonstrado nos experimentos, foi concluído que o melhor método a se utilizar é o Aprendizado por Reforço com *Experience Replay* sem a incialização aleatória do *buffer* de repetição, mas com *transfer learning*. Então, no momento que o usuário criar um projeto pela interface web, já será realizado um treinamento do agente no grafo correspondente, sem nenhum bloqueio ou nó inicial e os pesos desse treino serão armazenados no banco de dados MongoDB, para serem utilizado futuramente. No momento que o usuário escolher fazer a inferência de um projeto existente, será

realizado o treinamento com *transfer learning*, utilizando o peso pré-treinado, armazenado no banco de dados, fazendo com que o treinamento aconteça da forma mais rápida possível.

Uma integração ideal para o projeto, é a utilização de sensores de fumaça inteligentes. É possível utilizar sensores de fumaça inteligentes ligados a internet, para indicar para a aplicação propostas em tempo real, quais cômodos estão sendo atingidos pelo fogo, isso faz com que automaticamente, a cada atualização do ambiente, seja retreinado o agente com novos bloqueios, sempre indicando o caminho ideal e atualizado.

Para essa integração ideal é necessário um servidor externo ao ambiente monitorado para o treinamento do algoritmo de Aprendizado por Reforço proposto. Esse servidor pode centralizar diversas requisições em projetos diferentes. Um diagrama da integração proposta, pode ser visualizada na Figura 5.14

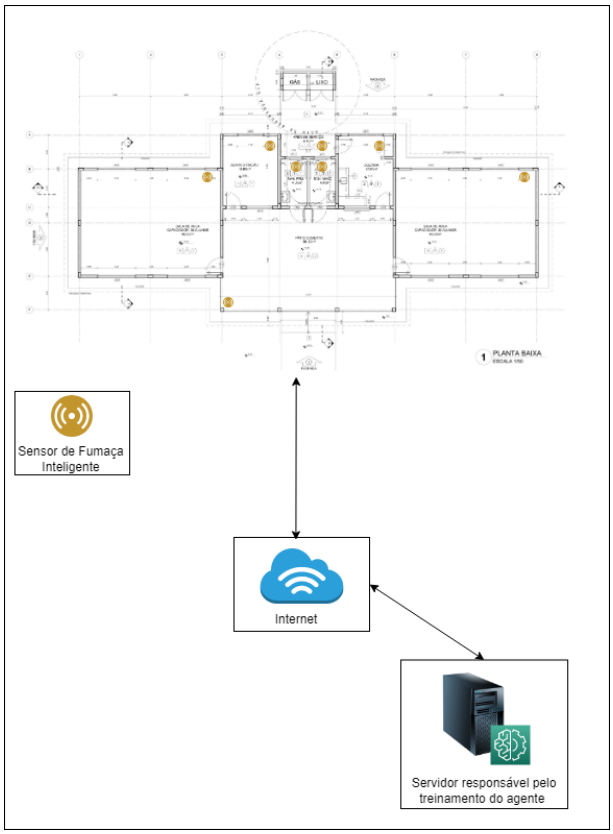

Figura 5.14: Integração ideal com sensores de fumaça inteligentes

Fonte: Autor

Capítulo 6

Conclusão

O Aprendizado de Máquina é um campo muito utilizado atualmente e em constante evolução, apesar disso, o sub-campo de Aprendizado por Reforço ainda tem muito a ser explorado, principalmente quando se refere a sua utilização em tempo real. Isso se deve ao fato de que uma pequena mudança no ambiente de aprendizagem, causa a necessidade de retreinar todo o agente.

Nesse trabalho foi feito um estudo da utilização em tempo real de Aprendizado por Reforço para a descoberta das melhores rotas de fuga em casos de incêndios, isso foi feito através da representação em forma de grafos das plantas baixas de edifícios.

Como apresentado na Subseção 4.3, foi necessário o desenvolvimento de uma interface web interativa, para que pudesse ser cadastrado os grafos referentes as plantas baixas utilizadas, como esse processo é manual e pode demorar, foi utilizado um banco de dados MongoDB para o armazenamento das informações do grafo, tirando a necessidade de ter que re-anotar o grafo toda vez que precisar ser feito a inferência de um projeto.

Como apresentado na Subseção 4.1, foi construído apenas um algoritmo de Aprendizado por Reforço, utilizando *Q-Learning*, para encontrar o melhor caminho de um nó específico até outro dentro de um grafo. Esse algoritmo funciona perfeitamente quando existe a necessidade de encontrar apenas um caminho dentro do grafo, porém como a proposta do trabalho era ter a possibilidade de encontrar o caminho de todos os nós do grafo até os nós indicados como saída, existiu a necessidade de construir um algoritmo de Aprendizado por Reforço utilizando redes neurais.

Foram escolhidos três projetos para realizar os experimentos, todos eles de plantas baixas de escolas públicas. Dentro dos experimentos foram realizadas as comparações de três métodos propostos, o de Aprendizado por Reforço com *Experience Replay* com inicialização aleatória do *buffer* de repetição, Aprendizado por Reforço com *Experience Replay* sem a inicialização aleatória do *buffer* de repetição e Aprendizado por Reforço com *Experience Replay*, sem a inicialização aleatória do *buffer* de repetição, mas com *transfer learning*.

Como apresentado na Subseção 5.2, o Projeto 1 era o mais simples de todos, por possuir poucos nós e poucas arestas em seu grafo correspondente. Foi concluído que para o projeto, quando não existem nós bloqueados, não existiu diferenças significativas entre a utilização do método

de *Experience Replay* com e sem a inicialização aleatória do *buffer* de repetição, sendo utilizado 4000 episódios e com um tempo médio de treinamento de 41 segundos. Porém, quando existem nós bloqueados, sendo possível utilizar o peso pré-treinado do treino sem bloqueio, o método que possui *tranfer learning* se destaca dos demais, precisando apenas de 1500 episódios e 16 segundos para retornar caminhos com alta precisão.

Para o Projeto 2, apresentado na Subseção 5.3.1, fica claro que para grafos que possuem uma maior quantidade de nós e arestas, o método que possui a inicialização aleatória do *buffer* de repetição é ineficaz e atrapalha o aprendizado do agente, sendo necessário descartar esse método e adotar o método sem essa inicialização para o treinamento do agente em um grafo sem bloqueios, para que seu peso possa ser utilizado nos treinos com *transfer learning*, sendo esse o método novamente mais eficaz e rápido.

O Projeto 3, apresentado na Subseção 5.4, comprova que a técnica de *transfer learning* é a mais rápida e eficaz, necessitando de apenas 1 minuto e 28 segundos para retornar de forma precisa todos os caminhos dos nós do grafo para os nós de saída.

Os treinos com bloqueios são mais rápidos do que os realizados sem bloqueios quando o usuário cria o projeto pela interface web, isso porque, os treinos com bloqueios se beneficiam dos pesos pré-treinados. Isso faz com que em casos de emergência, em poucos minutos ou segundos, possa ser de fato calculado os melhores caminhos de todos os cômodos de um edifício até a saída.

Durante os testes foi demostrado como o algoritmo pode falhar em casos de um nó essencial ser atingido por fogo. Se um nó bloqueado, impedir que existam caminhos seguros de um ou mais nós dentro do grafo para a saída, o resultado do treinamento do agente é aleatório, até mesmo com a técnica de *transfer learning*, podendo inferir alguns caminhos dentro do grafo, ou até mesmo nenhum caminho, por isso, é recomendado nesses casos que se utilize o treinamento com um nó inicial. Utilizando um nó inicial dentro do grafo, o aprendizado é rápido, pois o agente só terá que aprender um caminho.

Os experimentos foram realizados utilizando uma infraestrutura local, não ideal para treinamento de Aprendizado de Máquina, portanto, os tempos de treinamento obtidos, podem ser melhorados utilizando uma infraestrutura indicada para isso.

Fazendo a junção de todos os componentes desenvolvidos nesses trabalho, foi possível construir uma aplicação final simples de se utilizar, porém, eficaz para encontrar em tempo real vários caminhos dentro de um grafo.

Referências

1 [S.l.: s.n.]. Disponível em: <https://www.cnmp.mp.br/portal/images/Comissoes/DireitosFundamentais/Acessibilidade/NBR_9077_Sa%C3%ADdas_de_emerg%C3%AAncia_em_edif%C3%ADcios-2001.pdf>.

2 [S.l.: s.n.]. Disponível em: <https://fastapi.tiangolo.com/>.

3 [S.l.: s.n.]. Disponível em: <https://dash14.github.io/v-network-graph/>.

4 A plataforma de aplicação de dados. [S.l.: s.n.]. Disponível em: <https://www.mongodb.com/pt-br>.

5 AGNIHOTRI, Aakanksha et al. Evacuating Routes in Indoor-Fire Scenarios with Selection of Safe Exits on Known and Unknown Buildings Using Machine Learning, p. 1–6, 2018. DOI: 10.1109/SARNOF.2018.8720478.

6 BHATIA, Shaveta. **Survey of shortest path algorithms**. [S.l.: s.n.], nov. 2019. Disponível em: <https://www.internationaljournalssrg.org/IJCSE/2019/Volume6-Issue11/IJCSE-V6I11P107.pdf>.

7 COMUNICAÇÃO SOCIAL DO FNDE COM INFORMAÇÕES DO MINISTÉRIO DA EDUCAÇÃO, Assessoria de. **Projeto Espaço Educativo Rural - 2 salas - portal DO FNDE**. [S.l.: s.n.]. Disponível em: <https://www.fnde.gov.br/index.php/programas/par/eixos-de-atuacao/infraestrutura-fisica-escolar/item/5952-projeto-espa%C3%A7o-educativo-rural-2-salas>.

8 _____. **Projeto ESPAÇO EDUCATIVO URBANO E rural - 4 salas com Quadra - Portal DO FNDE**. [S.l.: s.n.]. Disponível em: <https://www.fnde.gov.br/index.php/programas/par/eixos-de-atuacao/infraestrutura-fisica-escolar/item/5955-projeto-espaco-educativo-urbano-e-rural-4-salas-com-quadra>.

9 _____. **Projeto ESPAÇO EDUCATIVO URBANO E rural - 6 salas com Quadra Coberta - Portal DO FNDE**. [S.l.: s.n.]. Disponível em: <https://www.fnde.gov.br/index.php/programas/par/eixos-de-atuacao/infraestrutura-fisica-escolar/item/5957-projeto-espa%C3%A7o-educativo-urbano-e-rural-6-salas-com-quadra-coberta>.

10 CRISPIM, Calvin Mariano Rêgo. **Proposta de arquitetura segura de centrais de incêndio em nuvem**. 2021. Tese (Doutorado).

11 DEEP Q-learning: An introduction to deep reinforcement learning. [S.l.: s.n.], abr. 2020. Disponível em: <https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>.

12 DÉNES, König; EULER, Leonhard; SACHS, Horst. **Theorie der Endlichen und Unendlichen Graphen**. [S.l.: s.n.], 1986.

13 DENG, Hui et al. BIM and computer vision-based framework for fire emergency evacuation considering local safety performance. en. **Sensors (Basel)**, MDPI AG, v. 21, n. 11, p. 3851, jun. 2021.

14 DSA, Equipe. **Capítulo 68 - algoritmo de agente baseado EM IA com reinforcement learning – Q-learning**. [S.l.: s.n.], mar. 2020. Disponível em: <https://www.deeplearningbook.com.br/algoritmo-de-agente-baseado-em-ia-com-reinforcement-learning-q-learning/>.

15 EKLUND, Anton. **Cascade mask R-CNN and keypoint detection used in floorplan parsing**. 2020. Tese (Doutorado).

16 EMERGENCY Exit Routes. [S.l.: s.n.], 2018. https://www.osha.gov/sites/default/files/publications/emergency-exit-routes-factsheet.pdf. Accessed: 2022-4-22.

17 FEDUS, William et al. Revisiting Fundamentals of Experience Replay. **CoRR**, abs/2007.06700, 2020. arXiv: 2007.06700. Disponível em: <https://arxiv.org/abs/2007.06700>.

18 GYM is a standard API for reinforcement learning, and a diverse collection of reference environments. [S.l.: s.n.]. Disponível em: <https://www.gymlibrary.dev/>.

19 HE, Kaiming et al. Mask R-CNN. **CoRR**, abs/1703.06870, 2017. arXiv: 1703.06870. Disponível em: <http://arxiv.org/abs/1703.06870>.

20 HU, Ruizhen et al. Graph2Plan: Learning Floorplan Generation from Layout Graphs. **CoRR**, abs/2004.13204, 2020. arXiv: 2004.13204. Disponível em: <https://arxiv.org/abs/2004.13204>.

21 IH; Sarker. **Deep learning: A comprehensive overview on techniques, taxonomy, applications and Research Directions**. [S.l.]: U.S. National Library of Medicine. Disponível em: <https://pubmed.ncbi.nlm.nih.gov/34426802/>.

22 JANG, Hanme; YU, Kiyun; YANG, Jonghyeon. Indoor reconstruction from floorplan images with a deep learning approach. en. **ISPRS Int. J. Geoinf.**, MDPI AG, v. 9, n. 2, p. 65, jan. 2020.

23 KALERVO, Ahti et al. CubiCasa5K: A Dataset and an Improved Multi-Task Model for Floorplan Image Analysis. **CoRR**, abs/1904.01920, 2019. arXiv: 1904.01920. Disponível em: <http://arxiv.org/abs/1904.01920>.

24 KÄRKKÄINEN, Leo. **Extraction of geometry information from floor plan images with deep learning**. 2021. f. 46. Master's thesis – Aalto University. School of Science. Disponível em: <http://urn.fi/URN:NBN:fi:aalto-202108298617>.

25 LAM, Obadiah et al. Automated topometric graph generation from floor plan analysis. Edição: H Li e J Kim. Australian Robotics e Automation Association, Australia, p. 1–8, 2015. Disponível em: <https://eprints.qut.edu.au/90385/>.

26 LIU, Chen et al. Raster-to-Vector: Revisiting Floorplan Transformation, p. 2214–2222, 2017. DOI: 10.1109/ICCV.2017.241.

27 LU, Yueheng et al. CubiGraph5K - Organizational Graph Generation for Structured Architectural Floor Plan Dataset, 2021.

28 MUHAMMETBOZKURT. **Muhammetbozkurt/finding-shortest-path-with-reinforcement-learning**. [S.l.: s.n.]. Disponível em: <https://github.com/muhammetbozkurt/Finding-shortest-path-with-reinforcement-learning>.

29 NEVES, Enzo Cardeal. **Aprendizado por reforço 1- Introdução**. [S.l.]: Turing Talks, jun. 2020. Disponível em: <https://medium.com/turing-talks/aprendizado-por-refor%C3%A7o-1-introdu%C3%A7%C3%A3o-7382ebb641ab>.

30 OFFICE, Larry Hardesty | MIT News. **Explained: Neural networks**. [S.l.: s.n.]. Disponível em: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>.

31 PACELLI FERREIRA DIAS JUNIOR, EUGENIO. Aprendizado por reforço sobre o problema de revisitação de páginas web. **Dissertation**, p. 27, 2012. DOI: 10.17771/pucrio.acad.19637.

32 PRESTES, Edson. Capítulo 1 - Fundamentação Teórica. In: INTRODUÇÃO à Teoria dos Grafos. [S.l.: s.n.], 2020. p. 1–30.

33 _____. Capítulo 1.1 - Conceitos Básicos. In: INTRODUÇÃO à Teoria dos Grafos. [S.l.: s.n.], 2020. p. 2–7.

34 _____. Capítulo 1.3 - Matrizes de Adjacência e de Incidência. In: INTRODUÇÃO à Teoria dos Grafos. [S.l.: s.n.], 2020. p. 11–13.

35 _____. Capítulo 1.4 - Passeios e Circuitos. In: INTRODUÇÃO à Teoria dos Grafos. [S.l.: s.n.], 2020. p. 13–18.

36 PROJETOS Arquitetônicos Para Construção - portal do FNDE. [S.l.: s.n.]. Disponível em: <https://www.fnde.gov.br/index.php/programas/par/eixos-de-atuacao/infraestrutura-fisica-escolar?limitstart=0>.

37 ROSA, Anna Carolina. **Google colaboratory - graphDQN**. [S.l.]: Google. Disponível em: <https://colab.research.google.com/drive/1OtWE2lqF_oyCO0o_PG_QCot2C52RfI4J#scrollTo=ZxeMudpTKxBg>.

38 SANDELIN, F. Semantic and Instance Segmentation of Room Features in Floor Plans using Mask R-CNN. **Dissertation**, 2019.

39 SCHAUL, Tom et al. **Prioritized Experience Replay**. [S.l.]: arXiv, 2015. DOI: 10.48550/ARXIV.1511.05952. Disponível em: <https://arxiv.org/abs/1511.05952>.

40 SCHMITT, Simon et al. Fast routing graph extraction from floor plans, p. 1–8, 2017. DOI: 10.1109/IPIN.2017.8115868.

41 SELIN, Jukka; LETONSAARI, Mika; ROSSI, Markku. Emergency exit planning and simulation environment using gamification, artificial intelligence and data analytics. **Procedia Computer Science**, v. 156, p. 283–291, 2019. 8th International Young Scientists Conference on Computational Science, YSC2019, 24-28 June 2019, Heraklion, Greece. ISSN 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.08.204. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050919311238>.

42 SETE Pontes de Königsberg. [S.l.]: Wikimedia Foundation, mai. 2022. Disponível em: <https://pt.wikipedia.org/wiki/Sete_pontes_de_K%C3%B6nigsberg>.

43 SHARMA, Jivitesh et al. Deep Q-Learning With Q-Matrix Transfer Learning for Novel Fire Evacuation Environment. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, v. 51, n. 12, p. 7363–7381, 2021. DOI: 10.1109/TSMC.2020.2967936.

44 SUTTON, Richard S.; BARTO, Andrew G. Chapter 1.1 - Introduction, Reinforcement Learning. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 1–4.

45 _____. Chapter 1.3 - Introduction, Elements of Reinforcement Learning. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 6–7.

46 _____. Chapter 1.7 - Introduction, Early History of Reinforcement Learning. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 13–22.

47 _____. Chapter 2 - Multi-armed Bandits. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 25–42.

48 _____. Chapter 3 - Finite Markov Decision Processes. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 47–.

49 _____. Chapter 3 - Policies and Value Functions. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 58–62.

50 _____. Chapter 3.1 - Finite Markov Decision Processes, The Agent–Environment Interface. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 47–53.

51 _____. Chapter 3.2 - Finite Markov Decision Processes, Goals and Rewards. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 53–54.

52 _____. Chapter 3.3 - Finite Markov Decision Processes, Returns and Episodes. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 53–57.

53 _____. Chapter 6 - Temporal-Dierence Learning. In: REINFORCEMENT learning: An introduction. [S.l.]: The MIT Press, 2018. p. 119–138.

54 TANG, Rui et al. Automatic Structural Scene Digitalization. **CoRR**, abs/1710.01802, 2017. arXiv: 1710.01802. Disponível em: <http://arxiv.org/abs/1710.01802>.

55 TAYLOR, Matthew E.; STONE, Peter. Transfer Learning for Reinforcement Learning Domains: A Survey. **J. Mach. Learn. Res.**, v. 10, p. 1633–1685, 2009.

56 THE progressivejavascript framework. [S.l.: s.n.]. Disponível em: <https://vuejs.org/>.

57 TODAYSHOW. **Newer homes and furniture burn faster, giving you less time to escape a fire**. [S.l.]: TODAY, out. 2017. Disponível em: <https://www.today.com/home/newer-homes-furniture-burn-faster-giving-you-less-time-escape-t65826>.

58 TORRES.AI, Jordi. **Deep Q-Network (DQN)-II**. [S.l.]: Towards Data Science, mai. 2021. Disponível em: <https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c>.

59 WONGSAI, Pakamaj; PAWGASAME, Wichai. A Reinforcement Learning for Criminal's Escape Path Prediction, p. 26–30, 2018. DOI: 10.1109/ACDT.2018.8593191.

60 ZHANG, Shangtong; SUTTON, Richard S. A Deeper Look at Experience Replay. **CoRR**, abs/1712.01275, 2017. arXiv: 1712.01275. Disponível em: <http://arxiv.org/abs/1712.01275>.

61 ZHUANG, Fuzhen et al. A Comprehensive Survey on Transfer Learning. **CoRR**, abs/1911.02685, 2019. arXiv: 1911.02685. Disponível em: <http://arxiv.org/abs/1911.02685>.

ANEXOS

I.1 Trecho de Códigos Relevantes

Todos os Trecho de Códigos anexados são utilizando a linguagem de programação *python*.

I.1.1 Algoritmo de Q-Learning

O Trecho de Código 1 é referente ao algoritmo de *Q-Learning* construído, com um grafo como ambiente e levando em consideração o bloqueio de nós indicados como focos de incêndio.

```
import random
import networkx as nx
import numpy as np

class PathFinder(object):
    def __init__(self,graph: graph: nx.graph):
        self.graph = graph #grafo inicial
        self.adjacent_mat = nx.adjacency_matrix(graph).todense()
        self.num_nodes = len(self.adjacent_mat)
        self.adjacent_mat = nx.adjacency_matrix(graph, nodelist=range(self.num_nodes)).toarray() #matriz adjascente do grafo

    def q_learning(self, start_state: int, aim_state: int, fire: list, num_epoch: int = 200, gamma: float = 0.8, epsilon: float = 0.05, alpha: float = 0.1):
        len_of_paths = []
        q_table = np.zeros((self.num_nodes, self.num_nodes))  # num estados * num acoes

        for _ in range(1, num_epoch + 1): # interacao entre episodios
            current_state = start_state
            path = [current_state]
            len_of_path = 0
            while True:
                next_state = self.epsilon_greedy(current_state, q_table, epsilon=epsilon)
                s_next_next = self.epsilon_greedy(next_state, q_table, epsilon=-0.2)  # epsilon<0, greedy policy
                # update q_table
                if next_state in fire:
                    reward = -10
                else:
                    reward = -self.adjacent_mat[current_state][next_state]

                delta = reward + gamma * q_table[next_state, s_next_next] - q_table[current_state, next_state]

                q_table[current_state, next_state] = q_table[current_state, next_state] + alpha * delta
                # atualizar estado atual
                current_state = next_state
                len_of_path += -reward
```

```
                    path.append(current_state)

                    if current_state in aim_state:
                        break
                len_of_paths.append(len_of_path)

            return path

        def epsilon_greedy(self,s_curr, q, epsilon):#exploracao x intensificacao
            potential_next_states = np.where(np.array(self.adjacent_mat[s_curr])
    > 0)[0]
            if random.random() > epsilon:
                q_of_next_states = q[s_curr][potential_next_states]
                s_next = potential_next_states[np.argmax(q_of_next_states)]
            else:
                s_next = random.choice(potential_next_states)
            return s_next

```

Trecho de código 1: Algoritmo de Q-Learning utilizado

I.1.2 Ambiente do Aprendizado por Reforço Profundo

O Trecho de Código 2 se refere ao método *init* do ambiente, que consiste na inicialização das variáveis necessárias, como o grafo referente a planta baixa, os nós de saída e os nós atingidos por fogo. É nesse método que se extrai a matriz adjacente do grafo.

```
class Environment(object):
  def __init__(self, G: nx.classes.graph.Graph, exist: List, fire:List, beta:
    float) -> None:
    self.graph = G #Grafo
    self.exit = exit #Lista de saidas
    self.fire = fire # Lista de nos bloqeuados
    self.beta = beta #Fator de disconto

    node_list = list(self.graph.nodes)
    self.adj_mat = nx.adjacency_matrix(self.graph, nodelist=node_list).todense()
    #Matriz adjascente

    self.num_nodes = len(self.graph.nodes)
    self.n_actions = self.num_nodes #Numero de acoes
    self.obs_size = self.num_nodes * 2 #Tamanho do espaco de observacao
    self.dict_node = self.create_dict()

  def create_dict(self) -> Dict:
    dict_node = {}

    for idx, node in enumerate(self.graph.nodes):
      dict_node[node] = idx

    return dict_node
```

```

```

Trecho de código 2: Método de *Init*

O Trecho de Código 3 se refere ao método *reset* do ambiente, onde será reinicializado o ambiente para suas condições iniciais.

```

    def next_state(self, action: int) -> int:
        nodes = self.num_nodes #Quantidade de nos

        return action+nodes

    def define_start(self) -> int:
        while True:
          start = random.randint(0, self.num_nodes-1)
          if start not in self.exit: #Estado inicial nao pode ser um no de saida
            break

        return start

    def reset(self, previous_state = None) -> int:
        if previous_state: #Se existe um estado inicial definido
          start = int(previous_state)

        else: #Se o estado inicial for aleatorio
          start = self.define_start()

        state = self.next_state(self.dict_node[start]) #Estado inicial

        return state

```

Trecho de código 3: Método de *Reset*

O Trecho de Código 4 será responsável pelo núcleo de aprendizagem, definindo a próxima ação a ser realizada dentro do ambiente, o próximo estado e a recompensa recebida.

```

    def reward(self, current_node: int, new_node: int) -> Tuple[float, bool]:
     connected = False

     if new_node in self.fire: #O proximo no esta bloqueado
       rw = -10000

     elif new_node in self.graph[current_node]:
       rw = self.adj_mat.item((current_node,new_node))*-1
       connected = True #Nos adjascentes

     else:
       rw = -5000 #Nos nao adjascentes

     return rw, connected

```

```
16
17  def call_reward(self, current_node: int, action: int) -> Tuple[int, bool]:
18    discount = self.beta
19
20    #self.dict = Dicionario de nos
21    current_node = self.dict_node[current_node]
22    new_node = self.dict_node[action]
23    rw, connected = self.reward(current_node, new_node)
24
25    function_reward = rw*discount
26
27    return function_reward, connected
28
29  def current_node_end(self, state: int) -> Tuple[int, int]:
30    nodes = self.num_nodes
31    destination = state % nodes
32    end = (state-destination)/nodes
33
34    return destination, int(end)
35
36  def step(self, state: int, action: int) -> Tuple[int, int, bool]:
37    done = False
38
39    current_node, _ = self.current_node_end(state)
40    new_state = self.next_state(action)
41
42    reward, connected = self.call_reward(current_node, action)
43
44    if not connected:
45      new_state = state #Nos nao sao adjascentes
46
47    elif action in self.exit: #Final do episodio
48      reward = 10000
49      done = True
50
51    return new_state, reward, done
52
```

Trecho de código 4: Método de *Step*

O Trecho de Código 5 se refere a dois métodos auxiliares no ambiente, o primeiro sendo responsáveis por transformar um estado em uma lista de estados e o segundo por transformar uma lista de tensores de novos estados em uma lista de estados. Essas funções serão utilizadas para a atualização das redes utilizadas (incluindo a rede de destino).

```
1     def state_to_vector(self, current_node: int, end_node: int) -> List:
2         n_nodes = len(self.graph.nodes)
3
4         source_list_zeros = [0.] * n_nodes
5         source_list_zeros[current_node] = 1
6
7         end_list_zeros = [0.] * n_nodes
```

```
        end_list_zeros[end_node] = 1.

        vector = source_list_zeros + end_list_zeros

        return vector

    def list_of_vectors(self, new_states_t: T.Tensor) -> List:
        list_new_states_t = new_states_t.tolist()
        list_new_states_t = [int(v) for v in list_new_states_t]

        vector_list = []
        for state in list_new_states_t:
          s, f = self.current_node_end(state)
          vector = self.state_to_vector(s, f)
          vector_list.append(vector)

        return vector_list

```

Trecho de código 5: Funções auxiliares do Ambiente

I.1.3 Agente do Aprendizado por Reforço

O Trecho de Código 6 se refere ao método *init* do agente, sendo responsável por inicializar as variáveis necessárias, o ambiente e a rede neural.

```
    def __init__(self, graph: nx.classes.graph.Graph, fire: List, exit: List,
transfer_learning: str, hyp: Dict) -> None:

        self.device = T.device('cuda' if T.cuda.is_available() else 'cpu')

        # Hiperparametros utilizados
        beta = hyp['beta']
        lr = hyp['lr']
        self.batch_size = hyp['batchsize']
        self.episodes = hyp['episodes']
        self.decay = self.episodes
        self.gamma = hyp['gamma']
        self.epsilon = hyp['epsilon']

        self.env = Environment(graph, exit, fire, beta) # Ambiente
        self.net = Network(self.env.obs_size, self.env.n_actions) #Rede neural

        if transfer_learning:
            self.net.load_state_dict(transfer_learning['model'])
            self.net.eval()

        self.target = Network(self.env.obs_size, self.env.n_actions) #Rede de
Destino
        self.target.load_state_dict(self.net.state_dict())
        self.optimizer = T.optim.Adam(self.net.parameters(), lr=lr) #Otimizador
```

```
24
25          #Variaveis utilizada para o Experience Replay
26          self.reward_buffer = [0]
27          self.episode_reward = 0.0
28          self.buffer_size = hyp['buffer_size']
29          self.min_replay_size = int(self.buffer_size*0.25)
30
31          #Variaveis utilizadas para rede de destino
32          self.target_update_frequency = 1000
33          self.action_list = np.arange(0, len(self.env.graph.nodes)).tolist()
34          self.replay_buffer = deque(maxlen=self.min_replay_size)
35
```

Trecho de código 6: Método de *init* do Agente

O Trecho de Código 7 se refere a inicialização aleatória do *buffer* de repetição com valores aleatórios.

```
1     def init_buffer_replay(self, previous_state=None) -> None:
2         state = self.env.reset(previous_state)
3         for i in tqdm(range(self.min_replay_size)):
4           action = np.random.choice(self.action_list)
5           new_state, rew, done = self.env.step(state, action)
6           transition = (state, action, rew, done, new_state)
7           self.replay_buffer.append(transition)
8           state = new_state
9           if done:
10              state = self.env.reset()
11
```

Trecho de código 7: Inicialização do *Buffer* de repetição

O Trecho de Código 8 se refere a primeira parte do treinamento do agente.

```
1     def train(self, previous_state=None) -> None:
2         state = self.env.reset(previous_state)
3         loss_list = []
4         mean_reward = []
5         number_ep = []
6
7         for i in tqdm(range(self.episodes)):
8           epsilon = np.exp(-i/(self.episodes/2))
9           p = random.random()
10
11          if p <= epsilon:
12            action = np.random.choice(self.action_list)
13
14          else:
15            current_node, end = self.env.current_node_end(state)
16            vector_state = self.env.state_to_vector(current_node, end)
17            tensor_state = T.tensor([vector_state])
18            action = self.net.act(tensor_state)
19
```

```
20          new_state, reward, done = self.env.step(state, action)
21
22          # Experience Replay
23          transition = (state, action, reward, done, new_state)
24          self.replay_buffer.append(transition)
25          state = new_state
26          self.episode_reward += reward
27
28          if done:
29            state = self.env.reset(previous_state)
30            self.reward_buffer.append(self.episode_reward)
31            self.episode_reward = 0.0
32
```

Trecho de código 8: Parte 1 do treinamento do agente

O Trecho de Código 9 se refere a segunda parte do treinamento do agente, onde será utilizado a rede de destino e o método de *Experience Replay*.

```
1       if len(self.replay_buffer) < self.batch_size:
2           transitions = self.replay_buffer
3         else:
4           transitions = random.sample(self.replay_buffer, self.batch_size)
5
6         states = np.asarray([t[0] for t in transitions])
7         actions = np.asarray([t[1] for t in transitions])
8         rewards = np.asarray([t[2] for t in transitions])
9         dones = np.asarray([t[3] for t in transitions])
10        new_states = np.asarray([t[4] for t in transitions])
11
12        states_tensor = T.as_tensor(states, dtype=T.float32).to(self.device)
13        actions_tensor = T.as_tensor(actions, dtype=T.int64).to(self.device).
    unsqueeze(-1)
14        rewards_tensor = T.as_tensor(rewards, dtype=T.float32).to(self.device)
15        dones_tensor = T.as_tensor(dones, dtype=T.float32).to(self.device)
16        new_states_tensor = T.as_tensor(new_states, dtype=T.float32).to(self.
    device)
17
18        #Rede de Destino
19        list_new_states_tensor  = T.tensor(self.env.list_of_vectors(
    new_states_tensor)).to(self.device)
20        target_q_values = self.target(list_new_states_tensor)
21        max_target_q_values = target_q_values.max(dim=1, keepdim=False)[0]
22        targets = rewards_tensor+self.gamma*(1-dones_tensor)*
    max_target_q_values
23        targets = targets.unsqueeze(-1)
24
25        list_states_tensor = T.tensor(self.env.list_of_vectors(states_tensor)).
    to(self.device)
26        q_values = self.net(list_states_tensor)
27        action_q_values = T.gather(input=q_values, dim=1, index=actions_tensor)
28
29        #Perda MSE
```

```
          loss = nn.functional.mse_loss(action_q_values, targets)
          loss_list.append(loss.item())

          self.optimizer.zero_grad()
          loss.backward()
          self.optimizer.step()

          #Recompensa Acumulada
          mean_reward.append(np.mean(self.reward_buffer))

          #Atualizacao da Rede de Destino
          if i % self.target_update_frequency == 0:
              self.target.load_state_dict(self.net.state_dict())

```

Trecho de código 9: Parte 2 do treinamento do agente

O Trecho de Código 10 se refere ao código utilizado para fazer a inferência do modelo treinado.

```
  def inference(self, previous_state=None) -> Dict:
      model = self.net
      if previous_state:
        dict_nodes = {previous_state: previous_state}
      else:
        dict_nodes = self.env.dict_node.copy()
        for node in self.env.exit:
          dict_nodes.pop(node)

      for node in dict_nodes:
        state = self.env.reset(int(node))

        for i in tqdm(range(self.env.num_nodes)):
          current_node, end = self.env.current_node_end(state)
          vector_state = self.env.state_to_vector(current_node, end)
          tensor_state = T.tensor([vector_state])
          action = model.act(tensor_state)

          new_state, reward, done = self.env.step(state, action)

          state = new_state
          self.episode_reward += reward

          if done:
            dict_nodes[node] = path
            break

      result = []
      for i in dict_nodes:
        try:
          init_node = dict_nodes[i][0]
          last_node = dict_nodes[i][-1]
          path = dict_nodes[i]
        except:
```

```
            init_node = dict_nodes[i]
            last_node = dict_nodes[i]
            path = dict_nodes[i]

          dict = {"init_node": init_node, 'last_node': last_node, 'path': path}
          result.append(dict)

        return result

```

Trecho de código 10: Inferência do modelo