PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

HUGGO CAMPOS DA SILVA

CONTROLE INTELIGENTE DE TRÁFEGO TERRESTRE: Uma revisão literária

Goiânia
2023

HUGGO CAMPOS DA SILVA

CONTROLE INTELIGENTE DE TRÁFEGO TERRESTRE: Uma revisão literária

Trabalho de Conclusão de Curso II apresentado à Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Marcelo Antonio Adad de Araújo, M.E E.

Goiânia

2023

HUGGO CAMPOS DA SILVA

CONTROLE INTELIGENTE DE TRÁFEGO TERRESTRE: Uma revisão literária

Este Trabalho de Conclusão de Curso julgado adequado para obtenção do Título de Bacharel em Engenharia de Computação, e aprovado em sua forma final pela Escola Politécnica e de artes, da Pontifícia Universidade Católica de Goiás, em ____/___/_____.

Prof. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca examinadora:

Orientador: Prof. Marcelo Antonio Adad de Araújo, M.E.E

Prof. Carlos Alexandre Ferreira de Lima, M.E.E

Prof. Nilson Cardoso Amaral, Dr.

Goiânia

2023

AGRADECIMENTOS

A Deus primeiramente por me dar saúde e permitir estar presente no exato momento, à minha família que está do meu lado confiando em mim em orações e me incentivando.

Agradeço ao orientador e professor Marcelo Adad Antonio de Araújo que esteve me ajudando do início ao fim deste projeto, nas terças-feiras e sextas-feiras.

Agradeço à minha família por todo apoio e incentivo.

"A vida só pode ser compreendida, olhando-se para trás;
mas só pode ser vivida, olhando-se para frente."

Soren Kierkegaar

RESUMO

O trabalho exposto apresenta o estudo sobre os benefícios da utilização da Inteligência Artificial no controle do tráfego terrestre urbano nas grandes cidades. Nesse contexto, o projeto contempla o percurso a ser percorrido considerando o histórico do controle de tráfego terrestre, os principais recursos de regulagem do semáforo, bem como os benefícios da Inteligência Artificial no controle de tráfego. Assim, trata-se de uma revisão de literatura considerando as iniciativas considerando a aplicação da Inteligência Artificial no controle de tráfego.

Palavras-chave: controlador, Inteligência Artificial, semáforo, tráfego.

ABSTRACT

The exposed work presents the study on the benefits of using Artificial Intelligence in the control of urban land traffic in large cities. In this context, the project contemplates the route to be followed considering the history of terrestrial traffic control, the main features of traffic light regulation, as well as the benefits of Artificial Intelligence in traffic control. Thus, it is a literature review considering the initiatives considering the application of Artificial Intelligence in traffic control.

Keywords: controller, Artificial Intelligence, traffic light, traffic.

LISTA DE FIGURAS

LISTA DE ABREVIATURA

I.A – Inteligência Artificial

LISTA DE ABREVIATURA

ITS – *Sistemas Inteligentes de transportes (Intelligent Transport Systems)*

SUMO - *Simulator of Urban Mobility*

A TraCI - *Traffic Control Interface*

SUMÁRIO

1 INTRODUÇÃO

Na atualidade o quantitativo de carros vem crescendo de forma rápida, dificultando o tráfego nas cidades. Assim, de acordo com o Instituto Brasileiro de Geografia e Estatística (IBGE, 2020), o Brasil conta com uma frota com mais de 107 milhões de veículos nas ruas. Em decorrência da variação no sentido de fluxo de veículos e pedestres nos horários de pico, o uso de semáforos com tempo fixo pode não ser o mais adequado. Nesse contexto, o uso de semáforos inteligentes pode ser uma necessidade.

O semáforo é um potente controlador de trânsito que possibilita a diminuição de colisões entre veículos, aumentando a capacidade de escoamento e organiza o trânsito (ARAÚJO, 2006). As avenidas e ruas, enquanto meio físico de circulação de veículos, devem ser projetadas para adaptar ao volume de tráfego passante diariamente. Assim, em um cruzamento entre duas ou mais vias existem movimentos que não podem ser realizados de forma simultânea, e, no intuito de evitar colisões é necessário estabelecer normas de controle de passagem, tanto de veículos como de pedestres.

Os conflitos entre veículos são resolvidos de forma efetiva pela regra do primeiro a chegar e atravessar em vias de baixo fluxo de tráfego, mas, em ruas movimentadas essas regras não são eficazes. Nesse contexto é essencial estabelecer regras de prioridade entre as aproximações de cruzamento para permitir a travessia da intersecção. Nesse contexto, o semáforo determina a autorização ou proibição de movimentos em uma intersecção (REGO; SEMENTE, 2017).

O semáforo inteligente possui tempo de ciclo não fixo, apresentando alternância de luzes do semáforo de acordo com a quantidade de veículos presentes. Para tanto, um sistema inteligente de controle de tráfego é capaz de garantir a mobilidade urbana, aperfeiçoando as interrupções visando o fluxo de veículos. Esse sistema será eficiente se preparado para a rotina local e, também para situações adversas, de maneira automática, assistida em que um operador será capaz de realizar as modificações necessárias.

1.0 Objetivos

Neste tópico serão apresentados os objetivos de propósito geral e específico do projeto.

1.0.1 Objetivo Geral

Analisar os benefícios da Inteligência Artificial no controle de tráfego terrestre por meio de semáforos nas grandes cidades.

1.0.2 Objetivos Específicos

- Conhecer o histórico do controle de tráfego terrestre e os principais recursos de controle dos e regulagem dos semáforos.
- Identificar os benefícios do uso da Inteligência Artificial no controle de tráfego terrestre.
- Analisar os benefícios da Inteligência Artificial no controle de tráfego terrestre de grandes cidades, considerando as iniciativas em desenvolvimento apresentadas na literatura.

1.0.3 Justificativa

É inegável que o tráfego veicular urbano influencia na vida das pessoas e o quanto um sistema semafórico eficiente reflete em otimização de tempo e consequentemente na qualidade de vida das pessoas, que deixam de perder tempo no trânsito. Assim, o gerenciamento de tráfego é um determinante para a otimização dessa dinâmica, e nesse contexto os semáforos inteligentes se fazem necessários (SERRA, 2004).

Os semáforos inteligentes são um sistema de controle de tráfego de veículos que associam os semáforos tradicionais com uma variedade de sensores de inteligência artificial para direcionar de forma eficiente o tráfego de veículos e pedestres.

Muitas cidades ainda utilizam semáforos de gerações mais antigas que utilizam intervalos especificados considerando os horários de maior movimento do dia, o que não garante uma configuração eficiente, uma vez que não prevê ações mediante eventos inesperados. Assim, o uso de sistemas semafóricos inteligentes permitem coleta de dados que possam ajustar os sinais em tempo real, registrando o uso de detectores e câmeras. As informações coletadas indicam o ajuste do sistema de forma a otimizar o fluxo de tráfego, ajustar a duração do ciclo de sinais e garantir a eficiência do trânsito (MACHADO e SILVA, 2022).

Destarte, o uso da Inteligência Artificial no controle semafórico, proporciona mais rapidez e menos paradas (trânsito fluido), controle de situações imprevistas, aumento da satisfação e redução de reclamações e redução de custos e benefícios para o meio ambiente. Nesse contexto, há a necessidade de se empenhar em pesquisas que sintetizem as últimas atualizações do ramo, considerando os sistemas mais eficientes, eficazes e modernos. Esse tipo de estudo pode estimular a utilização da inteligência artificial no controle semafórico ainda pouco utilizado no contexto brasileiro, expandindo os horizontes para a modernização do controle de tráfego das grandes cidades.

Este estudo apresenta relevância acadêmica por sintetizar os estudos mais recentes sobre controle inteligente de tráfego terrestre, proporcionando um estudo sistematizado e articulado com as produções mais recentes e significativas sobre o tema. Apresenta ainda forte relevância social por apresentar alternativas eficientes para gestão do controle de trânsito de cidades que ainda não adotaram esse sistema, e que poderão utilizar-se desse estudo para pensar em alternativas viáveis.

2 FUNDAMENTAÇÃO TEÓRICA

O transporte rodoviário é o principal sistema logístico de transporte do país, sendo o principal meio de transporte de cargas e passageiros no tráfego viário brasileiro. Essa relevância é constituída historicamente, uma vez que, desde o início da república os governos estabeleceram como prioridade o transporte rodoviário, em detrimento do transporte aéreo, ferroviário e fluvial.

O controle de tráfego é cada vez mais utilizado principalmente nas cidades, por proporcionar o gerenciamento de veículos que circulam pelas ruas e vias nas cidades, e a sua ausência pode gerar uma superlotação e elevação do tráfego diário ou mesmo momentâneo. O monitoramento de tráfego é necessário e pertinente devido a superlotação que pode ocorrer com o aumento do tráfego diário. A arquitetura inadequada pode afetar o tráfego terrestre e provocar congestionamento.

No objetivo de minimizar problemas de tráfego relacionados à circulação de veículos, várias estratégias de controle de tráfego vêm sendo recrutadas, dentre elas, a mais comum é o semáforo, sobretudo considerando o controle de interseções. No entanto, Serra (2004) assegura que a eficiência de operação é um determinante para esse sucesso uma vez que se mal operados podem causar transtornos aos usuários, por atrasos no tempo de viagem, gasto elevado de combustível, poluição ou outros determinantes.

Os semáforos enquanto equipamento de controle de tráfego terrestre devem proporcionar a alternância entre veículos ou pedestres em intersecções entre duas ou mais vias, assegurando os atributos operacionais de fluidez e segurança. Mas esse processo não é novo e pode ser percebido pela evolução do controle de tráfego ocorrido no país nas últimas décadas. Segundo, Bonetti Jr e Pietroantonio (2001) a primeira evolução semafórica ocorreu no Brasil em 1970, em que a Campanha de Engenharia de Tráfego do município de São Paulo, contou com a introdução de controladores multiplanos de tempo fixo, com utilização de planos de controle semafóricos determinados anteriormente por meio de dados históricos do tráfego, e que o plano determinado é resultado de uma programação horária que requer constante levantamento de dados no intuito de atender as demandas do tráfego.

Na década de 1980, houve a inclusão da implantação da centralização de equipamentos com programações de tempo fixo, utilizando controladores eletrônicos

importados da Inglaterra. Esses controladores atuam em subáreas que contam com grupos de interseções semafóricas, em que todos os cruzamentos têm o mesmo ciclo ou múltiplos ciclos, trabalhando em sincronia.

Outra evolução foi a possibilidade de ajuste dos tempos de um ou mais cruzamentos que possam ter imprevistos como acidentes ou veículos quebrados, por meio de técnicos que atuavam em centrais (SERRA, 2004).

Já na década de 1990, foram desenvolvidos controladores eletrônicos mais modernos, os denominados semáforos inteligentes e a implantação de sistemas centralizados de tráfego, com controle de tráfego em tempo real. O foco desses equipamentos está na centralização e a programação dos semáforos é determinada por sistemas dedicados com base em dados de tráfegos coletados nos detectores em campo. A principal característica é a operação de forma isolada ou coordenada, no intuito de determinar dinamicamente o tempo de ciclo semafórico (SERRA, 2001).

A partir dos anos 2000, o controle passou a ser aplicado tanto na forma atuada (tempos semafóricos: verde, vermelho e amarelo) como na seleção de dinâmica de planos (variação de tempos de controle fixos). O tráfego de todo mundo pode ser monitorado, e esse monitoramento fornece informações relevantes, como quais são as horas de pico e como as fases (luzes) podem ser monitoradas de acordo. O sistema computacional é capaz de monitorar o tempo e a mudança de operação conforme o clima, ou ainda em caso de emergência, otimizando o tráfego e a segurança de ruas e estradas.

2.0 Evolução da tecnologia semafórica no contexto brasileiro

A evolução relacionada ao controle de semáforos no Brasil, começou a ser percebida na década de 1970, tendo como referência linhas definidas em outros países, e sendo liderada pela Companhia de Engenharia de Tráfego da cidade de São Paulo. A princípio, quando os equipamentos eram predominantemente eletromecânicos, e só posteriormente foram introduzidos os computadores multiplanos de tempo fixo. Posteriormente, houve a centralização dos equipamentos e programações de tempo fixo recrutando os controladores eletrônicos importados.

O segundo movimento identificado contemplou a implantação da centralização

de equipamentos com programação temporal fixa, com uso de controladores importados. Segundo Yuki e Ferreira (2008), desde essa época até o contexto atual, o controle semafórico em modo atuado não é utilizado significativamente no país, exceto em estágios específicos de travessia para pedestres que pode ser acionado pelo botão. Muito provavelmente, essa organização se dá devido à escassez de recursos tecnológicos avançados no Brasil.

A forma mais recrutada utiliza os controladores com anéis[1] duais, que atuam em duas sequências alternativas paralelas ou ainda como o conhecimento controle volume-densidade[2]. Nesse caso, a lógica estabelecida baseia-se na contagem do número de chegadas durante o período que o sinaleiro permanece vermelho para estabelecer o verde inicial variável e a redução progressiva no intervalo do corte durante o tempo de verde estendido.

Yuki e Ferreira (2008) afirmam que uma terceira técnica utilizada, a mudança por desperdício “waste chance” contempla a lógica de interrupção do verde considerando o acúmulo de intervalos excedentes entre os veículos em relação ao intervalo médio de movimento normal da fila.

Os controladores nacionais apresentam recurso de operação de planos fixo no modo isolado e coordenado, com lógica tradicional e com parâmetros operacionais reduzidos. Em uma perspectiva geral, os controladores nacionais dispõem de recurso de centralização em supervisão e operação, diferenciando-se pelas características de interfaces mais ou menos amigáveis para tanto.

A iniciativa de elaboração de controladores nacionais foi realizada em São Paulo e possui origens inglesas. Aparentemente, a tímida evolução de atuação nos controles eletrônicos nacionais é resultado de causa e consequência desse processo. Nesse contexto, Yuki e Ferreira (2008) afirmam que a dificuldade de implantar os

---

[1] “Anel” é um recurso que possibilita a um controlador operar como se fossem vários controladores independentes. Cada anel é responsável pelo controle de certo número de grupos focais de uma interseção complexa. Outra possibilidade é a operação de um conjunto de até quatro cruzamentos, em que cada um poderá se constituir em um anel, comandados por um único controlador. A programação semafórica de um anel é independente da programação semafórica dos demais anéis do controlador. A grande vantagem de se trabalhar com anéis em cruzamentos complexos é a simplificação da programação, evitando que se tenha que prever todas as combinações de estágios que possam ocorrer simultaneamente em cada sub cruzamento.
(NOTA TÉCNICA 273, 2021, p. 11).

[2] Controle tipo americano a lógica de interrupção de verde considera o acúmulo do excesso dos intervalos entre os veículos em relação ao valor básico de saturação, ou seja, ao intervalo médio no movimento normal da fila (YUKI e FERREIRA, 2008).

sistemas centralizados limitam o desenvolvimento tecnológico do setor.

2.1 Controladores de tráfego inteligentes

Para tornar o sistema semafórico inteligente é necessário integrá-lo à internet. Por essa rede é possível controlar os veículos, o objeto de controle precisa ter capacidade computacional e incluir algum tipo de conexão direta ou indireta com a internet. As tecnologias de Web para os desenvolvedores de programação podem ser utilizadas via Web, que permite o gerenciamento do sistema e o algoritmo de controle do semáforo.

Anterior a essa possibilidade, no ano de 1920, William Potts, Policial de Detroit, Michigan, construiu o primeiro semáforo esquematizando as cores a comandos (verde para seguir, amarelo para atenção e vermelho para parar). Esses semáforos foram instalados na parte superior das torres de trânsito ou fixados em cordoalhas sobre a pista e eram manuseados por policiais ali posicionados. A vantagem percebida no controle manual era a possibilidade de decisão mediante o controle de veículos de acordo com a observação do policial (CORTEZ, 2022).

Posteriormente, os semáforos foram adquirindo mais autonomia e passaram a ser controlados por tempo fixo, predeterminado para cada sinal de cor, dispensando a presença do policial, o que garantiu a expansão numérica dos semáforos. Com o aumento expressivo do trânsito veicular nas cidades, houve o aumento do congestionamento, e um dos fatores determinantes para essa condição foi o semáforo não conseguir dar preferência às vias que tivessem mais veículos.

É nesse contexto que a Inteligência Artificial se faz presente, e mesmo que não notada pelas pessoas é capaz de exercer funções de extrema relevância na tecnologia da informação, permitindo a utilização de algoritmos inteligentes, para elaboração de software e hardware, que permitam a máquina a inteligência e percepção necessária (CORTEZ, 2022).

Sistemas inteligentes de transportes (Intelligent Transport Systems – ITS) tem como finalidade garantir a eficiência do tráfego, sendo utilizados em vários países. Esses sistemas controlam a segurança das vias e reduzem o desperdício de recursos de infraestrutura. Assim, em uma perspectiva ideal, os semáforos teriam o mesmo

tempo, no entanto, nem todas as vias possuem o mesmo volume de tráfego. Mediante a necessidade de organização considerando o tempo e fluxo, câmeras, sensores, e outros são utilizados pela IA para elaborar um algoritmo capaz de prever e informar sinais de trânsito garantindo funcionamento conforme os dados obtidos (MACHADO e SILVA, 2022).

Os sistemas que utilizam IA com o intuito de identificar padrões são capazes de realizar previsões e executar ações de forma mais efetiva. Para esse sistema atingir seu objetivo é essencial que a quantidade e qualidade de dados obtidos por câmeras e sensores de trânsito seja elevada. Conforme Machado e Silva (2022) o aumento de automóveis nas ruas não pode ser contido, mas o congestionamento pode ser gerenciado por meio da IA, e com mudanças estruturais conforme a necessidade. Nessa dinâmica o tempo de congestionamento otimiza a fluidez no trânsito.

Para o efetivo funcionamento do algoritmo, o mesmo necessita de uma coleta de dados contínua ao longo do tempo para posterior análise e extração de informações sobre a via, para que a programação seja eficaz e eficiente. Nesse contexto é necessário sintetizar aqui os sistemas inteligentes de controle semafóricos:

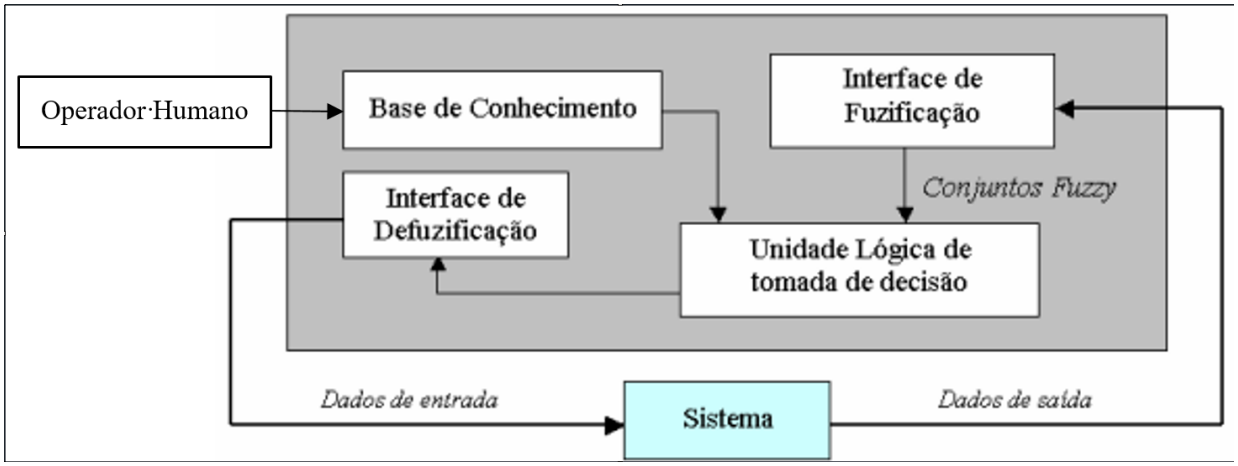
Lógica Fuzzy

Pode ser definida como uma ferramenta capaz de capturar informações vagas, em geral descritas em linguagem natural e convertidas em um formato numérico, que pode ser manipulado computacionalmente. Por fornecer meios de representação e de manipulação de conhecimentos imprecisos e vagos, como “grande” ou “um pouco mais” estabelecendo uma interface entre os dados descritos, por meio de variáveis linguísticas ou numéricas. O sistema Fuzzy permite a implementação de controladores não-lineares, organizados por especialistas e aplicáveis no campo da Engenharia de Transportes e Trânsito e são tema de diversas pesquisas acadêmicas.

A Lógica Fuzzy é baseada em palavras, “valores” expressos linguisticamente, com termos como frio, médio, longe, com modificadores de predicado como, muito, mais, menos, pouco e, ainda, com um conjunto de quantificadores, como por exemplo, pouco, vários, usualmente. Assim, faz ainda uso de probabilidade linguística, por exemplo, provável, improvável, interpretados como números e manipulados aritmeticamente, manuseando todos os valores entre os limites 0 e 1 (ANDRADE, 2004).

Os conjuntos Fuzzy podem ser representados por números reais, por variedade linguística e não apresentam contornos definidos. A principal aplicação da Lógica Fuzzy é o controlador Fuzzy, e a sua vantagem é a habilidade de utilizar o conhecimento especialista na estruturação. Esse conhecimento deve ser expresso com uma base de renda, o controlador fuzzy é composto basicamente pelos métodos de fuzificação, inferência e defuzificação. Conforme o esquema da figura 1:

Figura 1: Diagrama de um controlador Fuzzy



Fonte: Adaptada de (NIITTYMAKI, 1998)

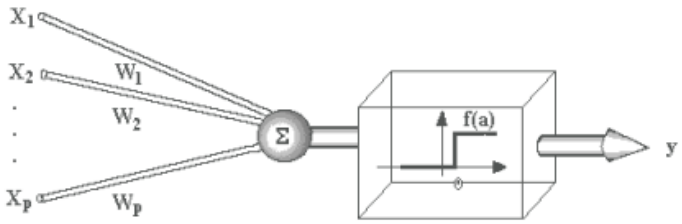
A interface de fuzificação é definida como processo de associação dos valores observados das variáveis de entrada nos universos de discurso correspondentes, aqui são realizadas as avaliações relacionadas ao grau de pertinência do valor medido aos diferentes conjuntos fuzzy da variável linguística considerada. A base de conhecimento do algoritmo fuzzy é o conjunto ordenado de instruções em que a execução reside uma solução aproximada para um problema específico. Suas instruções são expressas pelas regras fuzzy “Se-Então” (ANDRADE, 2004).

Em seguida na lógica para tomada de decisão, o comportamento do controlador fuzzy é caracterizado pela agregação de diferentes regras fuzzy, na qual a lógica resultante é obtida pela utilização de operadores lógicos. Já a interface de defuzificação apresenta após o processamento da variável de entrada do controlador fuzzy são realizadas pelo algoritmo de controle e o resultado obtido na variável de saída é dado no formato fuzzy. Nesse contexto, o processo de defuzificação prevê a seleção de um valor numérico específico que represente o resultado de saída fuzzy, elaborado pelo conjunto de regras fuzzy(ANDRADE, 2004).

Redes Neurais Artificiais

São sistemas paralelos e distribuídos, formados por unidades simples, os neurônios, onde são realizadas funções matemáticas, sendo dispostas em duas ou mais camadas interligadas por conexões. Essas unidades recebem informações as processam e repassam. Abaixo é esquematizado um neurônio artificial:

Figura 2: Neurônio Artificial – Esquema

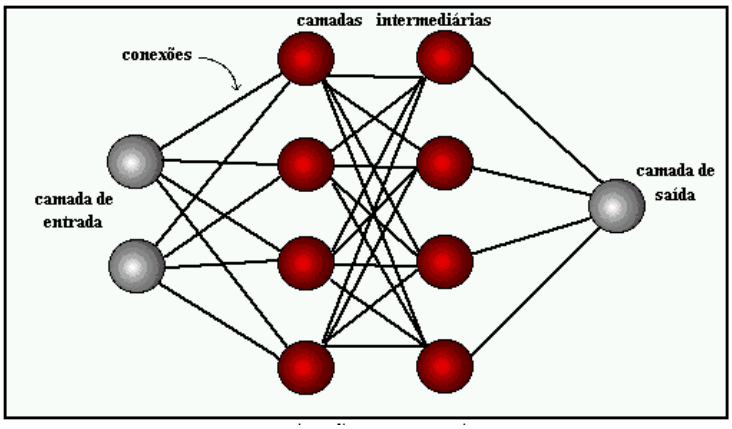


Fonte: (TATIBANA e KAETSU, 2000)

No esquema representado na figura 2, os sinais de entrada (Xi), são multiplicados nos sinais sinápticos (Wi) e em seguida somados, o resultado passa por uma função de ativação f(a) onde deve restringir o sinal de saída (Y) a um valor finito. Assim, a função essencial do neurônio é somar as entradas e retornar uma saída (ANDRADE, 2004).

As redes neurais são esquematizadas basicamente por três camadas, com funções específicas, sendo a camada de entrada responsável pela distribuição das informações para camadas intermediárias, captadas do meio externo. As camadas intermediárias ou ainda ocultas, em que ocorre o processamento e a extração das características. E, por fim, a camada de saída, responsável pela resposta da rede.

Figura 3: Estrutura básica de uma rede neural



(Fonte: https://sites.icmc.usp.br/andre/research/neural/)

A aprendizagem de uma rede é realizada por meio de um algoritmo de aprendizagem ou de treinamento e sua função é modificar os pesos sinápticos da rede de forma ordenada com um objetivo desejado. O algoritmo de aprendizado neural mais utilizado é resultado da diferença entre os dados de saída e os resultados esperados da rede neural. Assim, são características da rede neural: tolerância à falhas, capacidade de aprendizagem, formas de processamento paralelo e distribuído (que permite reconhecer informações incompletas ou ruidosas), capacidade de adaptação aos meios sinápticos ocorridas no ambiente e ainda, capacidade de representação e mapeamentos estatísticos e dinâmicos (ANDRADE, 2004).

As redes neurais podem ser classificadas considerando a disposição dos seus neurônios, os ciclos ou ainda o processo de aprendizagem.

Algorítmos genéticos

Os Algoritmos Genéticos são de busca, que utiliza o processo de seleção natural proposto por Charles Darwin e são bastante utilizados em problemas que dado um conjunto de elementos ou indivíduos se alcance, e que dependendo do problema proposto melhor atenda a certas condições especificadas. Para buscar os melhores resultados os Algoritmos Genéticos executam os seguintes operadores:

> • Seleção: Um dos meios de fazer a seleção de indivíduos para o crossover, e o mais simples, é o método da roleta no qual cada indivíduo da população recebe uma fatia da roleta, essa fatia é determinada pela avaliação (Fitness) do

> indivíduo desta forma quem estiver mais próximo da solução do problema terá mais chances de ser sorteado. Em seguida é feito o sorteio aleatório, seria análogo ao "rodar a roleta", desses indivíduos os selecionados farão parte do operador de crossover e são colocados em uma população auxiliar (...). • Cruzamento ou Crossover: O operador de cruzamento e de mutação são operadores chave do AG para que a busca seja bem sucedida,são os operadores mais X Encontro Anual de Computação - EnAComp 2013 303 importantes do AG [Man et al. 2001]. Existem várias estratégias para esta operação será utilizada neste artigo a estratégia de cruzamento de um ponto, um ponto de cruzamento é escolhido aleatoriamente e a partir dele é feita a troca genética dos indivíduos escolhidos na seleção (país) para a geração dos novos indivíduos (filhos). A informações genéticas anteriores a este ponto de um dos pais é ligado às informações posteriores do outro pai gerando a cada cruzamento dois novos filhos, que são colocados na população auxiliar (...). • Mutação: O operador mutação é necessário para que a população seja diversificada, permite chegar mais rápido ao objetivo da busca e para fugir dos mínimos locais, alterando aleatoriamente um ou mais genes dos indivíduos gerados pelo cruzamento. • Reinserção: Para saber quais indivíduos da população auxiliar será colocado na nova população é utilizado o operador de reinserção. Com a estratégia do elitismo pode haver um crescimento grande de super indivíduo na população, mas usando uma estratégia mais equilibrada parece melhorar o desempenho do AG, essa estratégia é chamada de melhores pais e filhos onde são escolhidos N/2 melhores indivíduos geradores (pais) e N/2 melhores indivíduos criados (Filhos) no qual são inseridos na população gerando uma nova geração (CUNHA et. al. 2013, p.4)

Os algoritmos buscam resolver problemas de otimização de forma interativa, partindo de uma estado inicial, realizando pequenas modificações em solução potencial para o problema até encontrar a solução ideal. Assim os algoritmos genéticos fazem parte do ramo da ciência da computação e procuram simular por meio do algoritmo, partindo de um conjunto inicial de dados gerados em um sistema estocástico, que são avaliados mediante um problema proposto, combinando dados para uma nova solução, mediante a configuração de tempos de semáforo para que haja fluidez de tráfego de acordo com o fluxo de veículos detectados nas vias.

Dessa forma, cada combinação de tempos de semáforos é equivalente a um indivíduo entre uma população de configurações possíveis, assim, para avaliar todas as soluções possíveis é necessário o simulador SUMO[3], que aplicará essa solução dentro do ambiente de simulação de configuração com período predeterminado. No período de simulação o software irá por meio da TraCI API[4] realizar a leitura de

[3] SUMO - Simulator of Urban Mobility é software responsável por simular o comportamento dos tempos de semáforo para que os indivíduos do Algoritmo Genético possam ser avaliados. O simulador foi criado com o objetivo de ser uma plataforma para testes de novos produtos e soluções aplicada à modelagem de trânsito e tem grande aceitação na comunidade de simulação macroscópica e mesoscópica de engenharia de tráfego.(BEHRISCH, 2011).

[4] A TraCI (sigla para Traffic Control Interface) (BEHRISCH, 2011) é uma API escrita na linguagem Python, distribuída junto com o simulador SUMO, que permite integração de scripts Python escritos por desenvolvedores com o simulador. A API fornece métodos que permitem aos desenvolvedores capturar

sensores em execução, e propor através de um algoritmo adaptado do software TRANSYT, atribuindo um valor fitness para cada indivíduo. Após essa simulação, serão realizados cruzamentos entre as soluções durante um limite de gerações, no intuito de encontrar a solução mais adequada para o cenário (CRUZ, 2011).

Figura 4: Estrutura básica de um Algoritmo Genético

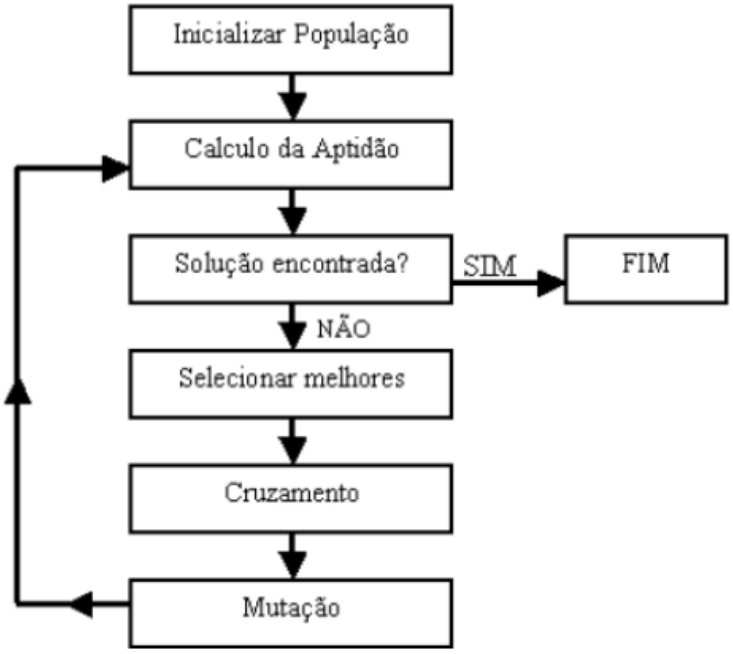


Figura 4: Estrutura básica de um Algoritmo Genético

2.2 Trajeto para implementação de um sistema semafórico

Primeiramente o Detran (2014) orienta a realização de pesquisas iniciais contemplando a contagem classificada de veículos e todas as aproximações da interseção e fazer o levantamento dos dados iniciais necessários. Assim, são considerados: número de faixas de rolamento em casa aproximação; tempo de ciclo de rede (se a interseção estiver inserida e uma via com sinalização semafórica

dados e intervir na simulação que está sendo executada em outro processo, em tempo de execução. Essa comunicação é feita pelo protocolo TCP/IP em uma porta previamente configurada no ambiente de simulação. Entre os métodos disponíveis pela API estão: alterar a configuração das luzes de semáforo em tempo real, efetuar a leitura dos sensores de solo, avançar os passos no ambiente de simulação, entre outros (CRUZ, 2011, p.45).

coordenada); e por fim distância da interseção estudada às interseções semaforizadas.

Em seguida é avaliado se a seguridade do local, essa avaliação se dá por meio da análise do cruzamento e quais características comprometem a segurança dos usuários. Assim, é considerada a geometria da via, as edificações a volta, ou ainda outros elementos que podem comprometer a visibilidade dos usuários ou ainda a própria configuração do cruzamento, que impeça a percepção de qual via é a principal. No caso citado anteriormente, deve-se testar soluções por meio de sinalização horizontal e vertical para se analisar a implantação semafórica.

Outro critério a ser analisado para justificação de uma sinalização semafórica é que o número de ciclos de vazio[5] durante o horário de pico deve ser inferior ao estipulado pelo projetista, sendo menor ou igual a 10% do número de ciclos por hora. Para tanto, Ferreira (2019) estabelece o procedimentos abaixo:

> · PASSO 1: Determinar o tempo de ciclo (C) em segundos, que o semáforo teria caso fosse instalado. O tempo de ciclo operante na rede, caso a mesma opere em modo coordenado, apenas pode ser utilizado caso a distância entre a interseção em estudo para o semáforo mais próximo seja inferior a 500m.
>
> · PASSO 2: Determinação do número de ciclos por hora (NC): NC = 3600/C
>
> · PASSO 3: Determinação do volume total das aproximações da via secundária (FTS) em termos de unidade de carros de passeio (UCP) por hora.
>
> · PASSO 4: Determinação do número médio de veículos por ciclo, em termos de UCP, nas aproximações da via secundária (m). $m = FTS/NC$
>
> · PASSO 5: Determinação do número de ciclos em que não existem veículos na via secundária chegando à interseção (NCV). $NCV = e^{-m} \times NC$ 4.4.5 Efetuar pesquisas de espera; FERREIRA, 2019 p.31)

Outro aspecto a ser considerado é a pesquisa sobre a espera, determinando o tempo total de espera dos veículos da via secundária. E, por fim, deve-se aferir na transversal qual seria o tempo total de espera indicado pelo semáforo, e, caso ele seja inferior a 6000 UCP x segundo, por hora considera-se um atraso médio de 15 segundos sob um volume de 400 UCP/hora na via secundária, descartando motocicletas, o sistema semafórico não poderá ser implantado. Caso o tempo total de espera for superior a 14.000 UCP x segundo, por hora, resultando em um atraso médio de 35 segundos para um volume de 400 UCP/hora na via secundária (desconsiderando as motos), o semáforo deverá ser implantado (DETRAN, 2014).

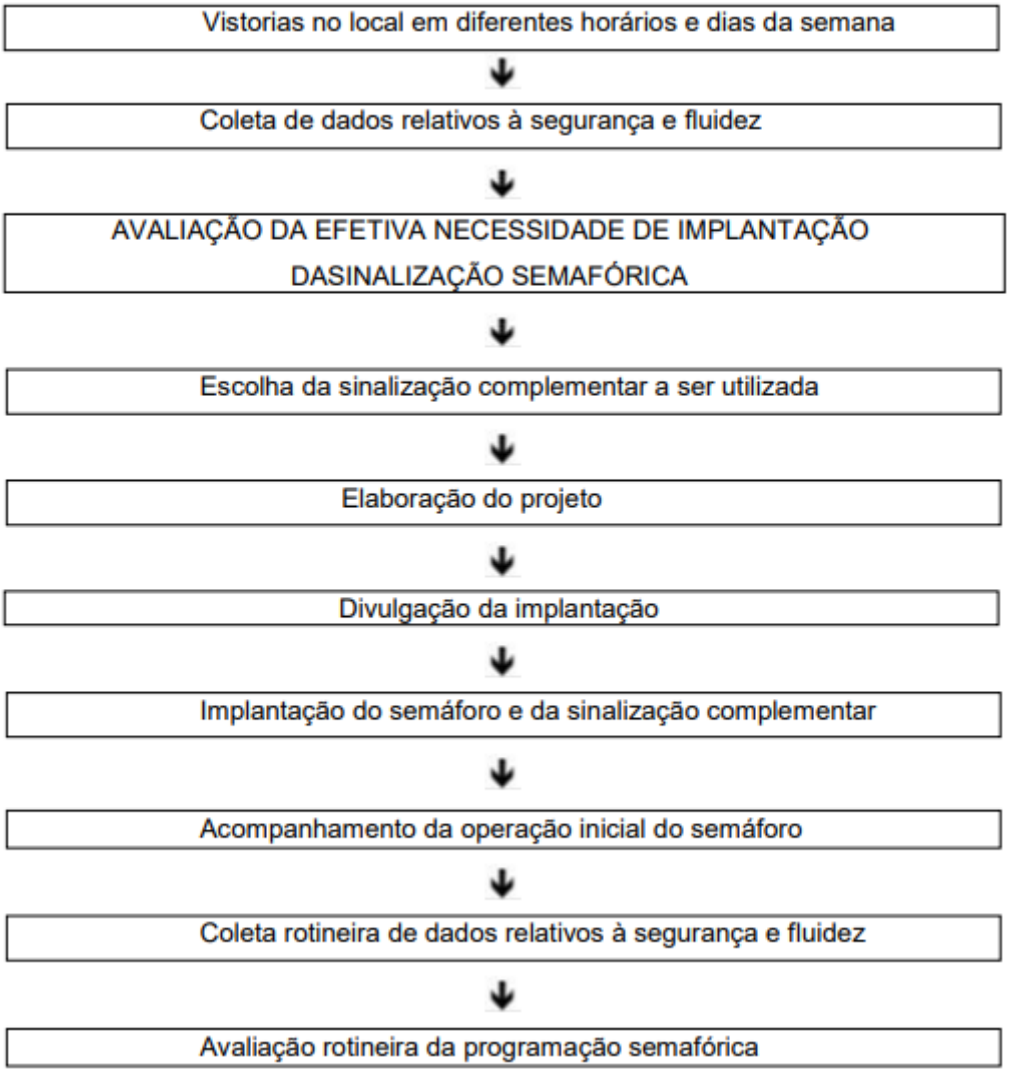
Nesse contexto, a solução não semafórica deve considerar a adoção de outra medida que contemple a segurança dos usuários e fluidez do trânsito, podendo ser

[5] Ciclos vazios são denominados os períodos onde não existiria uma demanda na via secundária dentro do ciclo programado para o semáforo caso ele fosse instalado.

considerada a redução de velocidade nas aproximações, adequação da geometria, a implementação de mini rotatórias, ou ainda mudança no sentido de circulação para eliminação do conflito. Caso a solução seja semafórica, os estudos são ampliados para que haja a definição quanto ao posicionamento e programação do semáforo (FERREIRA, 2019).

No volume V do Manual Brasilerio de Sinalização de Trânsito, no que se refere à sinalização semafórica, é sintetizado um fluxograma dos passos a serem seguidos para realização da implantação semafórica.

Figura 5 - Fluxograma do Procedimentos para implementação semafórica

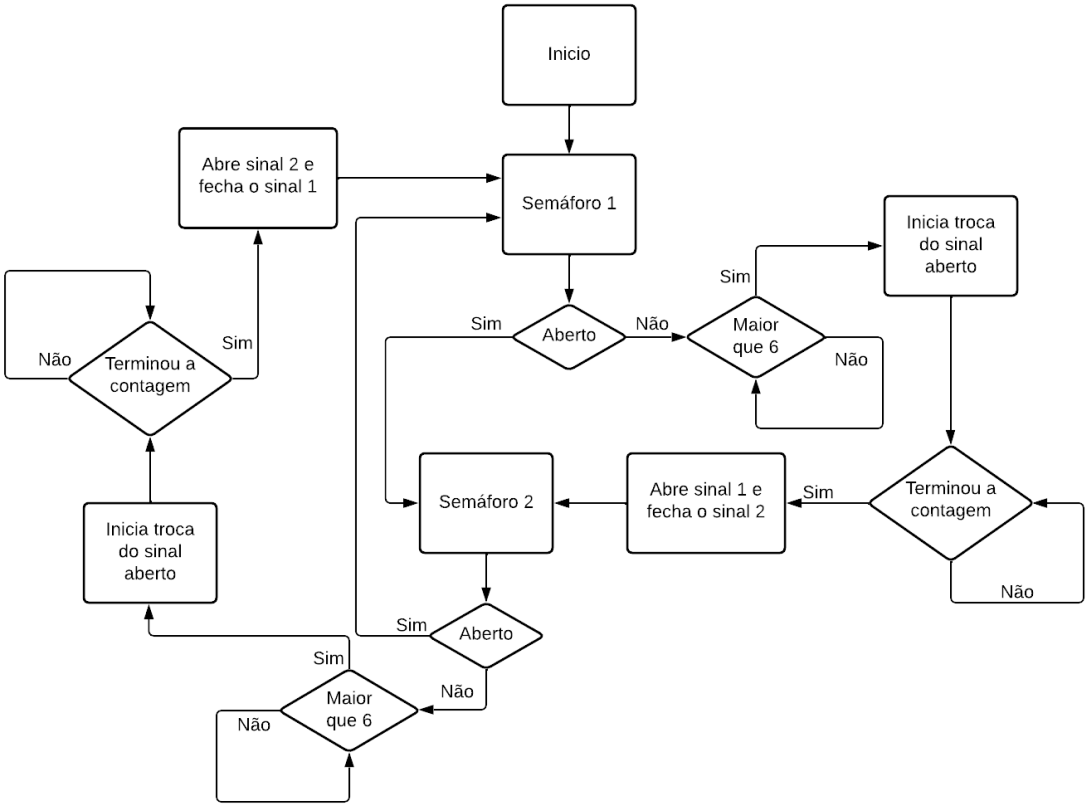


Fonte: Manual de sinalização de trânsito – Volume V – sinalização semafórica

Esse fluxograma apresenta a sistematização dos passos a serem seguidos para implantação semafórica efetiva, considerando as etapas a serem percorridas para que sua efetivação seja qualitativa e gere resultados.

3 RESULTADOS

Figura 6 – Fluxograma do funcionamento do semáforo



Fonte: Autoral

Foi realizada uma simulação em um *Integrated Development Environment* (IDE), chamado *Visual Studio Code* (VS Code) da Microsoft, uma ferramenta de suma importância para desenvolvedores a qual permite realizar todo o ciclo de desenvolvimento em um só lugar.

Considerado um IDE, pode ser utilizado em diversas funcionalidades, sendo essas para editar, criar códigos, implementar aplicativos entre outras. Na opção de desenvolver código, por exemplo, o Visual Studio fornece vários recursos que facilitam o gerenciamento do código com confiança.

Esta simulação foi feita em *Python,* uma linguagem de programação de alto nível, que já possui um ponto positivo, visto que existe uma certa facilidade de programar em linguagens de alto nível, programas escritos por elas tendem a ser mais

diretos, curtos, de fácil entendimento e assim possuem menos probabilidades de erros (KENZIE, 2022).

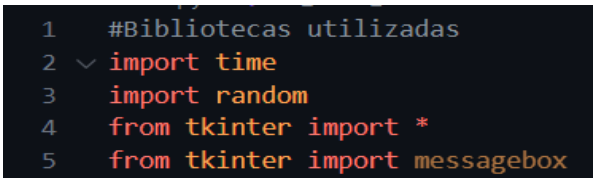
Ademais, mais uma vez ficou no topo de relevância no ranking anual das principais linguagens de programação publicado no dia 23 de agosto de 2022 pelo instituto de engenheiros elétricos e eletrônicos (IEEE). De acordo com Borges (2021) a linguagem criada em 1991 por um programador holandês chamado Guido van Rossum, vem se destacando e crescendo no decorrer dos anos, devido a sua alta quantidade de bibliotecas disponíveis em vários campos da tecnologia, como por exemplo a configuração de nuvens, inteligência artificial e outros.

Não apenas a linguagem de *Python* foi utilizada, mas ainda uma biblioteca chamada *Tkinter* para a criação da interface gráfica simulando dois cruzamentos, onde eles são “interligados” para funcionarem em sincronia, de acordo com o site DEVMEDIA *Tkinter* é uma biblioteca da linguagem python que permite desenvolver interfaces gráficas. Além da biblioteca *Tkinter* foram utilizadas as bibliotecas *“time”* e *“random”,* que servem para contar o tempo e fazer escolhas randômicas respectivamente, para ficar mais dinâmico no contexto da simulação.

Foram desenhados quatro sinais luminosos por cruzamento, um para cada via, onde os mesmos “visualizam” a quantidade de veículos esperando na via, tanto de um lado como do outro, foi determinado uma quantidade “x” de veículos para assim que a câmera do semáforo detectar esta quantidade de veículos, o sinal luminoso que está verde ficar amarelo, depois disso se conta um tempo para o fechamento total do sinal, assim que o tempo chega em zero, o sinal fica vermelho e o que estava vermelho fica verde, assim que o sinal fica verde, os veículos começam a sair, consequentemente outros veículos começam a parar no sentido contrário da via.

A figura 7 são as bibliotecas utilizadas para a criação da simulação conforme citadas acima.

Figura 7 - Bibliotecas utilizadas



Fonte: Autoral

Na figura 8 são mostradas as variáveis que foram utilizadas para simulação, as variáveis (j, k, n, m), são utilizadas para criação dos “carros”, onde é incrementada a posição para fazer o desenho do novo “carro”, as variáveis (l, p, l1, p1) são referentes às vias, e a variável (t) é a de tempo. Nas linhas 420 e 422 é criada, uma tela em branco na simulação.

Figura 8 - Variáveis e tela em branco

```
417  #Função Main
418  if __name__ == '__main__':
419      #Cria a janela da simulação
420      tela = Tk()
421      #Coloca a janela no centro da tela
422      tela.eval('tk::PlaceWindow . center')
423      #Titulo da janela
424      tela.title('Semaforo')
425      #Variaveis utilizadas
426      j = 0
427      k = 0
428      n = 0
429      m = 0
430      l = []
431      p = []
432      l1 = []
433      p1 = []
434      t = 5
435      #Cria um quadrado branco
436      canvas = Canvas(tela, width=1000, height=500, bg="white")
437      canvas.pack()
```

Fonte: Autoral

Na imagem a seguir é a criação dos retângulos para simular a caixa do semáforo do primeiro cruzamento figura 9.

Figura 9 - Criação das caixas do semáforo do primeiro cruzamento

```
439     #X, Y, largura, altura
440     #Primeira Via do cruzamento 1
441     canvas.create_line(190,165,190,0)
442     canvas.create_line(325,165,325,0)
443     #Segunda Via do cruzamento 1
444     canvas.create_line(190,165,0,165)
445     canvas.create_line(190,305,0,305)
446     #Terceira Via do cruzamento 1
447     canvas.create_line(190,500,190,305)
448     canvas.create_line(325,500,325,305)
449     #Quarta Via do cruzamento 1
450     canvas.create_line(325,165,650,165)
451     canvas.create_line(325,305,650,305)
```

Fonte: Autoral

A figura 10 é a criação das formas do sinal (Verde, Vermelho e Amarelo) do primeiro cruzamento.

Figura 10 - Criação das formas do sinal

```
453     #Cria o retangulo do primeiro semáforo do cruzamento 1
454     canvas.create_rectangle(190, 250, 260, 275,fill='black')
455     #Cria as formas do semáforo
456     oval_green_1 = canvas.create_oval(195, 255, 210, 270, fill="green")
457     oval_yellow_1 = canvas.create_rectangle(215,255, 235, 270, fill="grey")
458     oval_red_1 = canvas.create_oval(240, 255, 255, 270, fill="grey")
459     #Cria o retangulo do segundo semáforo do cruzamento 1
460     canvas.create_rectangle(255, 185, 325, 210,fill='black')
461     #Cria as formas do semáforo
462     oval_red_2 = canvas.create_oval(260, 190, 275, 205, fill="grey")
463     oval_yellow_2 = canvas.create_rectangle(280, 190, 300, 205, fill="grey")
464     oval_green_2 = canvas.create_oval(305, 190, 320, 205, fill="green")
465     #Cria o retangulo do terceiro semáforo do cruzamento 1
466     canvas.create_rectangle(210, 165, 235, 235,fill='black')
467     #Cria as formas do semáforo
468     oval_green_3 = canvas.create_oval(215, 170, 230, 185, fill="grey")
469     oval_yellow_3 = canvas.create_rectangle(215, 190, 230, 210, fill="grey")
470     oval_red_3 = canvas.create_oval(215, 215, 230, 230, fill="red")
471     #Cria o retangulo do quarto semáforo do cruzamento 1
472     canvas.create_rectangle(275, 230, 300, 300,fill='black')
473     #Cria as formas do semáforo
474     oval_red_4 = canvas.create_oval(280, 235, 295, 250, fill="red")
475     oval_yellow_4 = canvas.create_rectangle(280, 255, 295, 275, fill="grey")
476     oval_green_4 = canvas.create_oval(280, 280, 295, 295, fill="grey")
```

Fonte: Autoral

A criação das caixas e formas do sinal do segundo cruzamento, é igual ao primeiro cruzamento, mudando apenas a posição.

A criação dos botões é feita conforme a figura 11, o botão é criado e passado o nome que ele vai conter e o comando que ele vai executar quando for pressionado, logo em seguida é configurado o tamanho dele e a cor de fundo, após isso é criado uma janela e escolhido a posição que o mesmo vai ficar e o que essa janela vai conter, no nosso caso o botão. Após a criação dos dois botões, um deles é desativado pelo fato do semáforo estar aberto para aquelas vias, na linha 486 é a condição para a simulação ficar aberta.

Figura 11 - Criação dos botões

```
517     #Cria o botão para simular a camera do semáforo, e adiciona o comando
518     AD_subindo = Button(text = "        AD Subindo", command = ad_carro_subindo, anchor = W)
519     #Configura o botão
520     AD_subindo.configure(width = 15, activebackground = "#33B5E5", relief = FLAT)
521     #Cria uma janela para atrbuir o botão
522     AD_subindo_window = canvas.create_window(350, 350, anchor=NW, window=AD_subindo)
523     #Cria o botão para simular a camera do semáforo, e adiciona o comando
524     AD_atravessando = Button(text = "    AD Atravessando", command = ad_carro_atravessando, anchor = W)
525     #Configura o botão
526     AD_atravessando.configure(width = 15, activebackground = "#33B5E5", relief = FLAT)
527     #Cria uma janela para atrbuir o botão
528     AD_atravessando_window = canvas.create_window(520, 350, anchor=NW, window=AD_atravessando)
529
530     #Bloqueia o botão
531     AD_atravessando.configure(state=DISABLED)
532
533     #Mantem a janela da simulação ativa
534     tela.mainloop()
```

Fonte: Autoral

A imagem abaixo é o trecho do código onde adicionam os carros, onde na linha 12 é feita a escolha randomicamente entre (0, 1 2, 3), que são as vias, logo em seguida na linha 15 é caso a escolha seja “0”, a linha 17 é onde vai ser adicionado a cada vez que essa função for chamada um “carro” na primeira linha da “via” na simulação, caso a posição 1 e 2 seja preenchida, vai para a linha 19 onde se adiciona mais dois “carros” na segunda linha da “via”, a linha 21 faz o mesmo processo caso as duas posições forem preenchidas, da linha 24 até a 31 faz o mesmo processo mas caso a escolha seja “1”, e assim por diante. Já na linha 51 é a condição para o sinal ficar amarelo, caso a quantidade de carros esperando tanto na via de um lado quanto do outro seja maior que “x”, ele vai desabilitar o botão que adiciona os carros subindo, e vai chamar a função “amarelo_1()” para abrir o sinal para os carros que estão subindo ou descendo o cruzamento figura 12 e figura 13.

FIGURA 12 - Adição de carros subindo ou descendo

```
#Função para adicionar carros subindo ou descendo o Cruzamento 1
def ad_carro_subindo():
    #Variaveis globais
    global j
    global k
    global n
    global m
    #Variavel para escolha das vias
    var = [0,1,2,3]
    #Escolha randomica da via
    i = random.choice(var)
    #Se a escolha da via for "0"
    if i == 0:
        #Adiciona passo para a proxima execução
        j = j+35
        #Caso a via tenha menos de 2 carros
        if len(l) <= 1:
            #Cria carros na posições 1 e 2 da via
            l.append(canvas.create_rectangle(160+j, 160, 190+j, 130, fill="black"))
        #Caso a via tenha menos de 4 carros
        elif(len(l) <= 3):
            #Cria carros na posições 1 e 2 da via
            l.append(canvas.create_rectangle(90+j, 90, 120+j, 120, fill="black"))
        #Caso a via tenha menos de 6 carros
        elif(len(l) <= 5):
            #Cria carros na posições 1 e 2 da via
            l.append(canvas.create_rectangle(20+j, 50, 50+j, 80, fill="black"))
    #Se a escolha da via for "1"
    if i == 1:
        #Adiciona passo para a proxima execução
        k = k+35
        #Caso a via tenha menos de 2 carros
        if len(p) <= 1:
            #Cria carros na posições 1 e 2 da via
            p.append(canvas.create_rectangle(220+k, 335, 250+k, 305, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(p) <= 3):
            #Cria carros na posições 1 e 2 da via
            p.append(canvas.create_rectangle(150+k, 370, 180+k, 340, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(p) <= 5):
            #Cria carros na posições 1 e 2 da via
            p.append(canvas.create_rectangle(80+k, 405, 110+k, 375, fill="black"))

```

Fonte: Autoral

FIGURA 13 – Continuação da função adição de carros subindo ou descendo

```
    #Se a escolha da via for "2"
    if i == 2:
        #Adiciona passo para a proxima execução
        n = n+35
        #Caso a via tenha menos de 2 carros
        if len(l1) <= 1:
            #Cria carros na posições 1 e 2 da via
            l1.append(canvas.create_rectangle(620+n, 125, 650+n, 155, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(l1) <= 3):
            #Cria carros na posições 1 e 2 da via
            l1.append(canvas.create_rectangle(550+n, 90, 580+n, 120, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(l1) <= 5):
            #Cria carros na posições 1 e 2 da via
            l1.append(canvas.create_rectangle(480+n, 50, 510+n, 85, fill="black"))
    #Se a escolha da via for "3"
    if i == 3:
        #Adiciona passo para a proxima execução
        m = m+35
        #Caso a via tenha menos de 2 carros
        if len(p1) <= 1:
            #Cria carros na posições 1 e 2 da via
            p1.append(canvas.create_rectangle(680+m, 335, 710+m, 305, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(p1) <= 3):
            #Cria carros na posições 1 e 2 da via
            p1.append(canvas.create_rectangle(610+m, 370, 640+m, 340, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(p1) <= 5):
            #Cria carros na posições 1 e 2 da via
            p1.append(canvas.create_rectangle(540+m, 405, 570+m, 375, fill="black"))
    #Caso a quantidade de carros nas vias seja maior ou igual a 6
    if (len(l) + len(p) >= 6) or (len(l1) + len(p1) >= 6):
        #Atualiza a tela
        tela.update()
        #Desabilita o botão
        AD_subindo.configure(state=DISABLED)
        #Chama a função para deixar o sinal amarelo
        amarelo_1()
```

Fonte: Autoral

A adição de novos “carros” atravessando as vias é feita da mesma forma que os “carros” subindo ou descendo as vias, mudando apenas as posições figura 14 e figura 15.

FIGURA 14 - Função que adiciona os carros atravessando

```
#Função para adicionar carros subindo ou descendo o Cruzamento 2
def ad_carro_atravessando():
    #Variaveis globais
    global j
    global k
    global n
    global m
    #Variavel para escolha das vias
    var = [0,1,2,3]
    #Escolha randomica da via
    i = random.choice(var)
    #Se a escolha da via for "0"
    if i == 0:
        #Adiciona passo para a proxima execução
        j = 35-j
        #Caso a via tenha menos de 2 carros
        if len(l) <= 1:
            #Cria carros na posições 1 e 2 da via
            l.append(canvas.create_rectangle(155, 235+j, 185, 265+j, fill="black"))
        elif(len(l) <= 3):
            #Cria carros na posições 3 e 4 da via
            l.append(canvas.create_rectangle(120, 235+j, 150, 265+j, fill="black"))
        elif(len(l) <= 5):
            #Cria carros na posições 5 e 6 da via
            l.append(canvas.create_rectangle(85, 235+j, 115, 265+j, fill="black"))
    #Se a escolha da via for "0"
    if i == 1:
        #Adiciona passo para a proxima execução
        k = 35-k
        #Caso a via tenha menos de 2 carros
        if len(p) <= 1:
            #Cria carros na posições 1 e 2 da via
            p.append(canvas.create_rectangle(330, 170+k, 360, 200+k, fill="black"))
        elif(len(p) <= 3):
            #Cria carros na posições 3 e 4 da via
            p.append(canvas.create_rectangle(365, 170+k, 395, 200+k, fill="black"))
        elif(len(l) <= 5):
            #Cria carros na posições 5 e 6 da via
            l.append(canvas.create_rectangle(400, 170+k, 430, 200+k, fill="black"))
```

Fonte: Autoral

FIGURA 15 - Continuação da função que adiciona os carros atravessando

```
253     if i == 2:
254         #Adiciona passo para a proxima execução
255         n = 35-n
256         #Caso a via tenha menos de 2 carros
257         if len(l1) <= 1:
258             #Cria carros na posições 1 e 2 da via
259             l1.append(canvas.create_rectangle(610, 235+n, 640, 265+n, fill="black"))
260         elif(len(l1) <= 3):
261             #Cria carros na posições 3 e 4 da via
262             l1.append(canvas.create_rectangle(575, 235+n, 605, 265+n, fill="black"))
263         elif(len(l1) <= 5):
264             #Cria carros na posições 5 e 6 da via
265             l1.append(canvas.create_rectangle(540, 235+n, 570, 265+n, fill="black"))
266     #Se a escolha da via for "0"
267     if i == 3:
268         #Adiciona passo para a proxima execução
269         m = 35-m
270         #Caso a via tenha menos de 2 carros
271         if len(p1) <= 1:
272             #Cria carros na posições 1 e 2 da via
273             p1.append(canvas.create_rectangle(790, 170+m, 820, 200+m, fill="black"))
274         elif(len(p1) <= 3):
275             #Cria carros na posições 3 e 4 da via
276             p1.append(canvas.create_rectangle(825, 170+m, 855, 200+m, fill="black"))
277         elif(len(p1) <= 5):
278             #Cria carros na posições 5 e 6 da via
279             p1.append(canvas.create_rectangle(860, 170+m, 890, 200+m, fill="black"))
280     #Caso a quantidade de carros nas vias seja maior ou igual a 6
281     if (len(l) + len(p) >= 6) or (len(l1) + len(p1) >= 6):
282         #Atualiza a tela
283         tela.update()
284         #Desabilita o botão
285         AD_atravessando.configure(state=DISABLED)
286         #Chama a função para deixar o sinal amarelo
287         amarelo_2()
```

Fonte: Autoral

Quando a função para deixar o semáforo amarelo é chamada, as formas dos semáforos são preenchidas com as respectivas cores tanto na primeira via quanto na segunda via, ver figura 16.

FIGURA 16 - Função para deixar o semáforo amarelo

```
92  #Função para deixar o semáforo amarelo
93  def amarelo_1():
94      #Configuração das cores do primeiro semaforo do cruzamento
95      canvas.itemconfig(oval_red_1, fill="gray")
96      canvas.itemconfig(oval_yellow_1, fill="yellow")
97      canvas.itemconfig(oval_green_1, fill="grey")
98      #Configuração das cores do segundo semaforo do cruzamento
99      canvas.itemconfig(oval_red_2, fill="grey")
100     canvas.itemconfig(oval_yellow_2, fill="yellow")
101     canvas.itemconfig(oval_green_2, fill="grey")
102     #Configuração das cores do terceiro semaforo do cruzamento
103     canvas.itemconfig(oval_red_3, fill="red")
104     canvas.itemconfig(oval_yellow_3, fill="grey")
105     canvas.itemconfig(oval_green_3, fill="grey")
106     #Configuração das cores do quarto semaforo do cruzamento
107     canvas.itemconfig(oval_red_4, fill="red")
108     canvas.itemconfig(oval_yellow_4, fill="grey")
109     canvas.itemconfig(oval_green_4, fill="grey")
110     #Configuração das cores do quinto semaforo do cruzamento
111     canvas.itemconfig(oval_red_5, fill="gray")
112     canvas.itemconfig(oval_yellow_5, fill="yellow")
113     canvas.itemconfig(oval_green_5, fill="grey")
114     #Configuração das cores do sexto semaforo do cruzamento
115     canvas.itemconfig(oval_red_6, fill="grey")
116     canvas.itemconfig(oval_yellow_6, fill="yellow")
117     canvas.itemconfig(oval_green_6, fill="grey")
118     #Configuração das cores do setimo semaforo do cruzamento
119     canvas.itemconfig(oval_red_7, fill="red")
120     canvas.itemconfig(oval_yellow_7, fill="grey")
121     canvas.itemconfig(oval_green_7, fill="grey")
122     #Configuração das cores do oitavo semaforo do cruzamento
123     canvas.itemconfig(oval_red_8, fill="red")
124     canvas.itemconfig(oval_yellow_8, fill="grey")
125     canvas.itemconfig(oval_green_8, fill="grey")
```

Fonte: Autoral

Depois de as cores serem configuradas, é chamada a função “tela.update()” para atualizar a tela com as cores setadas e o “temporizador()” é chamado, ver figura 17.

FIGURA 17 - Chamada da função para contar o tempo

```
126     #Atualiza a tela
127     tela.update()
128     #Chama a função para contar o tempo
129     temporizador(t)
```

Fonte: Autoral

A função Temporizador mostra o tempo na parte amarela do semáforo, e a cada segundo a tela é atualizada para mostrar o tempo real do semáforo figura 18.

FIGURA 18 - Função para contar o tempo

```
401 #Função do temporizador
402 def temporizador(t):
403     #Variavel global
404     global timer
405     #Enquanto o tempo for diferete de "0"
406     while t != -1:
407         #Transforma o tempo em segundos
408         min,sec = divmod(t,60)
409         #Transfere somente os segundos para a variavel "TIMER"
410         timer = '{:02d}'.format(sec)
411         #Configura o sinal amarelo
412         canvas.create_rectangle(215,255, 235, 270,fill='yellow')
413         #Mostra o tempo no sinal
414         canvas.create_text(225, 262, text=timer, fill='black')
415         #Configura o sinal amarelo
416         canvas.create_rectangle(280, 190, 300, 205,fill='yellow')
417         #Mostra o tempo no sinal
418         canvas.create_text(290, 198, text=timer, fill='black')
419         #Configura o sinal amarelo
420         canvas.create_rectangle(675,255, 695, 270,fill='yellow')
421         #Mostra o tempo no sinal
422         canvas.create_text(685, 262, text=timer, fill='black')
423         #Configura o sinal amarelo
424         canvas.create_rectangle(740, 190, 760, 205,fill='yellow')
425         #Mostra o tempo no sinal
426         canvas.create_text(750, 198, text=timer, fill='black')
427         #Atualiza o tempo
428         canvas.itemconfig(timer)
429         #Atualiza a tela
430         tela.update()
431         #Aguarda 1 segundo
432         time.sleep(1)
433         t -= 1
```

Fonte: Autoral

Após o término do tempo que foi definido como 5 segundos, volta para a função “amarelo_1()” para atualizar as cores do semáforo, atualizar a tela e chamar a função que remove os carros das vias onde o sinal estava vermelho figura 19.

FIGURA 19 - Continuação da função para deixar o sinal amarelo

```
130     #Reconfigura o sinal amarelo de todos os semáforos
131     canvas.create_rectangle(215, 255, 235, 270,fill='grey')
132     canvas.create_rectangle(280, 190, 300, 205,fill='grey')
133     canvas.create_rectangle(675, 255, 695, 270,fill='grey')
134     canvas.create_rectangle(740, 190, 760, 205,fill='grey')
135     canvas.itemconfig(timer, fill='grey')
136
137     #Configuração das cores do primeiro semafDoro do cruzamento
138     canvas.itemconfig(oval_red_1, fill="red")
139     canvas.itemconfig(oval_yellow_1, fill="grey")
140     canvas.itemconfig(oval_green_1, fill="grey")
141     #Configuração das cores do segundo semaforo do cruzamento
142     canvas.itemconfig(oval_red_2, fill="red")
143     canvas.itemconfig(oval_yellow_2, fill="grey")
144     canvas.itemconfig(oval_green_2, fill="grey")
145     #Configuração das cores do terceiro semaforo do cruzamento
146     canvas.itemconfig(oval_red_3, fill="grey")
147     canvas.itemconfig(oval_yellow_3, fill="grey")
148     canvas.itemconfig(oval_green_3, fill="green")
149     #Configuração das cores do quarto semaforo do cruzamento
150     canvas.itemconfig(oval_red_4, fill="grey")
151     canvas.itemconfig(oval_yellow_4, fill="grey")
152     canvas.itemconfig(oval_green_4, fill="green")
153     #Configuração das cores do quinto semaforo do cruzamento
154     canvas.itemconfig(oval_red_5, fill="red")
155     canvas.itemconfig(oval_yellow_5, fill="grey")
156     canvas.itemconfig(oval_green_5, fill="grey")
157     #Configuração das cores do sexto semaforo do cruzamento
158     canvas.itemconfig(oval_red_6, fill="red")
159     canvas.itemconfig(oval_yellow_6, fill="grey")
160     canvas.itemconfig(oval_green_6, fill="grey")
161     #Configuração das cores do setimo semaforo do cruzamento
162     canvas.itemconfig(oval_red_7, fill="grey")
163     canvas.itemconfig(oval_yellow_7, fill="grey")
164     canvas.itemconfig(oval_green_7, fill="green")
165     #Configuração das cores do oitavo semaforo do cruzamento
166     canvas.itemconfig(oval_red_8, fill="grey")
167     canvas.itemconfig(oval_yellow_8, fill="grey")
168     canvas.itemconfig(oval_green_8, fill="green")
169     #Atualiza a tela
170     tela.update()
171     #Chama a função para remover os carros
172     rem_carro_subindo()
```

Fonte: Autoral

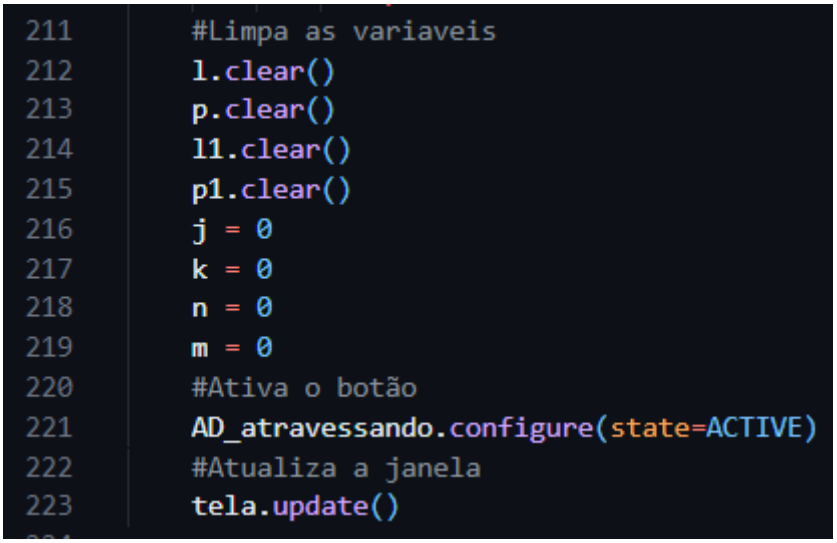
A função para remover os carros é chamada com “rem_carro_subindo()”, para “remover” os carros foi criado um quadrado branco em cima dos “carros”. Limpadas as variáveis para quando os carros forem adicionados novamente, de forma a não quebrar o código da simulação, após isso o botão que estava desativado agora fica ativo figura 20 e figura 21.

FIGURA 20 – Função para remover os carros subindo ou descendo

```
#Função para remover carros subindo ou descendo o Cruzamento 1
def rem_carro_subindo():
    #Variaveis globais
    global j
    global k
    global n
    global m
    #Remove os carros um por um
    for i in range(6):
        #Aguarda um segundo antes de remover o proximo carro
        time.sleep(1)
        #Atualiza a janela
        tela.update()
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(l[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(p[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(l1[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(p1[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
                pass
```

Fonte: Autoral

FIGURA 21 – Continuação da função para remover os carros subindo ou descendo



Fonte: Autoral

A função amarelo_2(), é a mesma da amarelo_1(), mudando apenas a ordem das cores na hora de aparecer no semáforo figura 22 e figura 23.

FIGURA 22 - Função para deixar o sinal amarelo

```
#Função para deixar o semáforo amarelo
def amarelo_2():
    #Configuração das cores do primeiro semaforo do cruzamento
    canvas.itemconfig(oval_red_1, fill="red")
    canvas.itemconfig(oval_yellow_1, fill="grey")
    canvas.itemconfig(oval_green_1, fill="grey")
    #Configuração das cores do segundo semaforo do cruzamento
    canvas.itemconfig(oval_red_2, fill="red")
    canvas.itemconfig(oval_yellow_2, fill="grey")
    canvas.itemconfig(oval_green_2, fill="grey")
    #Configuração das cores do terceiro semaforo do cruzamento
    canvas.itemconfig(oval_red_3, fill="grey")
    canvas.itemconfig(oval_yellow_3, fill="yellow")
    canvas.itemconfig(oval_green_3, fill="grey")
    #Configuração das cores do quarto semaforo do cruzamento
    canvas.itemconfig(oval_red_4, fill="grey")
    canvas.itemconfig(oval_yellow_4, fill="yellow")
    canvas.itemconfig(oval_green_4, fill="grey")
    #Configuração das cores do quinto semaforo do cruzamento
    canvas.itemconfig(oval_red_5, fill="red")
    canvas.itemconfig(oval_yellow_5, fill="grey")
    canvas.itemconfig(oval_green_5, fill="grey")
    #Configuração das cores do sexto semaforo do cruzamento
    canvas.itemconfig(oval_red_6, fill="red")
    canvas.itemconfig(oval_yellow_6, fill="grey")
    canvas.itemconfig(oval_green_6, fill="grey")
    #Configuração das cores do setimo semaforo do cruzamento
    canvas.itemconfig(oval_red_7, fill="grey")
    canvas.itemconfig(oval_yellow_7, fill="yellow")
    canvas.itemconfig(oval_green_7, fill="grey")
    #Configuração das cores do oitavo semaforo do cruzamento
    canvas.itemconfig(oval_red_8, fill="grey")
    canvas.itemconfig(oval_yellow_8, fill="yellow")
    canvas.itemconfig(oval_green_8, fill="grey")
    #Atualiza a tela
    tela.update()
    #Chama a função para contar o tempo
    temporizador_1(t)
```

Fonte: Autoral

FIGURA 23 - Continuação da função para deixar o sinal amarelo

```
340     #Reconfigura o sinal amarelo de todos os semáforos
341     canvas.create_rectangle(215, 190, 230, 210,fill='grey')
342     canvas.create_rectangle(280, 255, 295, 275,fill='grey')
343     canvas.create_rectangle(685, 190, 700, 210,fill='grey')
344     canvas.create_rectangle(740, 255, 755, 275,fill='grey')
345     canvas.itemconfig(timer2, fill='grey')
346     #Configuração das cores do primeiro semaforo do cruzamento
347     canvas.itemconfig(oval_red_1, fill="grey")
348     canvas.itemconfig(oval_yellow_1, fill="grey")
349     canvas.itemconfig(oval_green_1, fill="green")
350     #Configuração das cores do segundo semaforo do cruzamento
351     canvas.itemconfig(oval_red_2, fill="grey")
352     canvas.itemconfig(oval_yellow_2, fill="grey")
353     canvas.itemconfig(oval_green_2, fill="green")
354     #Configuração das cores do terceiro semaforo do cruzamento
355     canvas.itemconfig(oval_red_3, fill="red")
356     canvas.itemconfig(oval_yellow_3, fill="grey")
357     canvas.itemconfig(oval_green_3, fill="grey")
358     #Configuração das cores do quarto semaforo do cruzamento
359     canvas.itemconfig(oval_red_4, fill="red")
360     canvas.itemconfig(oval_yellow_4, fill="grey")
361     canvas.itemconfig(oval_green_4, fill="grey")
362     #Configuração das cores do quinto semaforo do cruzamento
363     canvas.itemconfig(oval_red_5, fill="grey")
364     canvas.itemconfig(oval_yellow_5, fill="grey")
365     canvas.itemconfig(oval_green_5, fill="green")
366     #Configuração das cores do sexto semaforo do cruzamento
367     canvas.itemconfig(oval_red_6, fill="grey")
368     canvas.itemconfig(oval_yellow_6, fill="grey")
369     canvas.itemconfig(oval_green_6, fill="green")
370     #Configuração das cores do setimo semaforo do cruzamento
371     canvas.itemconfig(oval_red_7, fill="red")
372     canvas.itemconfig(oval_yellow_7, fill="grey")
373     canvas.itemconfig(oval_green_7, fill="grey")
374     #Configuração das cores do oitavo semaforo do cruzamento
375     canvas.itemconfig(oval_red_8, fill="red")
376     canvas.itemconfig(oval_yellow_8, fill="grey")
377     canvas.itemconfig(oval_green_8, fill="grey")
378     #Atualiza a tela
379     tela.update()
380     #Chama a função para remover os carros
381     rem_carro_atravessando()
```

Fonte: Autoral

A função temporizador_1(), é a mesma da função temporizador (), mudando apenas onde os quadrados brancos são criados, como mostra a figura 24.

FIGURA 24 - Função para contar o tempo

```
#Função do temporizador
def temporizador_1(t):
    #Variavel global
    global timer2
    #Enquanto o tempo for diferete de "0"
    while t != -1:
        #Transforma o tempo em segundos
        min, sec = divmod(t,60)
        #Transfere somente os segundos para a variavel "TIMER"
        timer2 = '{:02d}'.format(sec)
        #Configura o sinal amarelo
        canvas.create_rectangle(215, 190, 230, 210,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(223, 200, text=timer2, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(280, 255, 295, 275,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(288, 265, text=timer2, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(685, 190, 700, 210,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(693, 200, text=timer2, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(740, 255, 755, 275,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(748, 265, text=timer2, fill='black')
        #Atualiza o tempo
        canvas.itemconfig(timer2)
        #Atualiza a tela
        tela.update()
        #Aguarda 1 segundo
        time.sleep(1)
        t -= 1
```

Fonte: Autoral

A rem_carro_atravessando(), é a mesma da rem_carro_subindo(), mudando apenas a ordem dos carros removidos figura 25 e figura 26.

FIGURA 25 - Função para remover os carros atravessando

```
383   #Função para remover carros subindo ou descendo o Cruzamento 2
384   def rem_carro_atravessando():
385       #Variaveis globais
386       global j
387       global k
388       global n
389       global m
390       #Remove os carros um por um
391       for i in range(6):
392           #Aguarda um segundo antes de remover o proximo carro
393           time.sleep(1)
394           #Atualiza a janela
395           tela.update()
396           #Remove o carro caso tenha
397           try:
398               canvas.itemconfig(l[i], fill="white", outline="white")
399           #Caso não tenha mais carros
400           except:
401               pass
402           #Remove o carro caso tenha
403           try:
404               canvas.itemconfig(p[i], fill="white", outline="white")
405           #Caso não tenha mais carros
406           except:
407               pass
408           #Remove o carro caso tenha
409           try:
410               canvas.itemconfig(l1[i], fill="white", outline="white")
411           #Caso não tenha mais carros
412           except:
413               pass
414           #Remove o carro caso tenha
415           try:
416               canvas.itemconfig(p1[i], fill="white", outline="white")
417           #Caso não tenha mais carros
418           except:
419                   pass
```

Fonte: Autoral

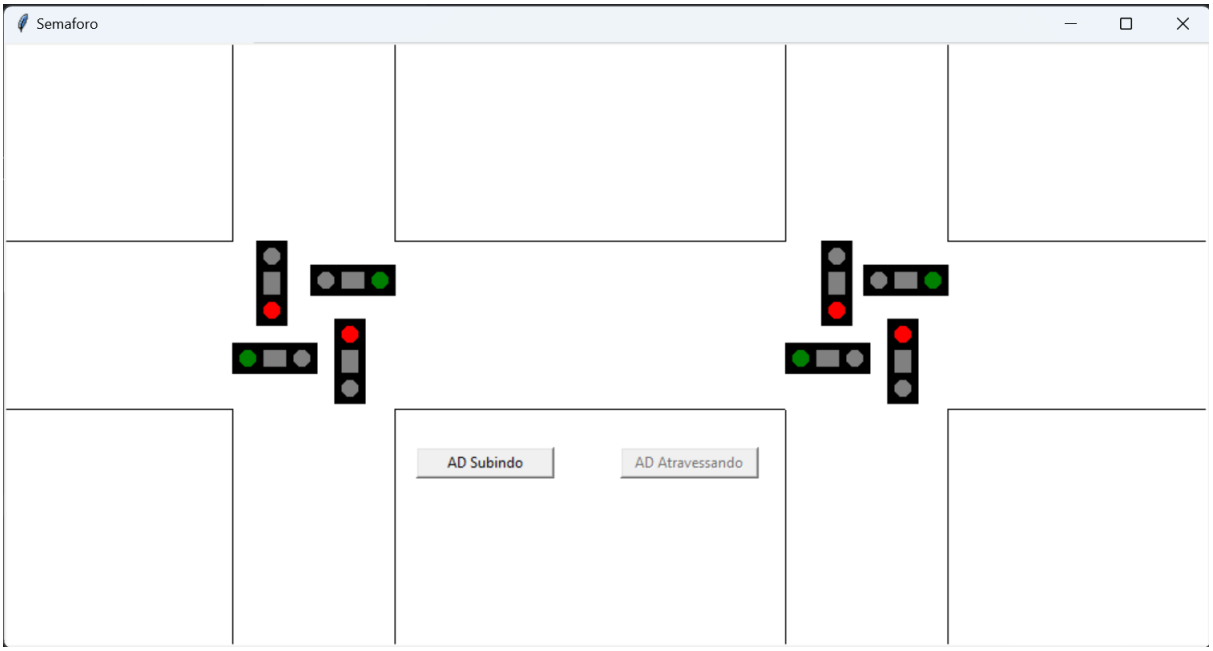
FIGURA 26 - Continuação da função para remover os carros subindo ou descendo

```
421     #Limpa as variaveis
422     l.clear()
423     p.clear()
424     l1.clear()
425     p1.clear()
426     j = 0
427     k = 0
428     n = 0
429     m = 0
430     #Ativa o botão
431     AD_atravessando.configure(state=ACTIVE)
432     #Atualiza a janela
433     tela.update()
```

Fonte: Autoral

Quando executa o código da simulação apresenta-se o resultado na figura 27.

FIGURA 27 - Janela da simulação



Fonte: Autoral

4 CONSIDERAÇÕES FINAIS

O presente projeto é fruto de um esforço em delinear um projeto que contemple o objetivo de analisar os benefícios da Inteligência Artificial no controle de tráfego terrestre por meio de semáforos. Assim, o projeto apresenta as etapas alcançadas no presente semestre, considerando o percurso histórico do controle e regulagem dos semáforos, o uso da inteligência artificial no controle do tráfego terrestre, bem como a análise dos benefícios do uso da IA no controle de tráfego nas grandes cidades e as experiências já publicadas que consideram e aplicam essa possibilidade.

Esse projeto se configura enquanto um esforço inicial de trabalho de final de curso sobre o tema, considerando a história e evolução do controle semafórico, o uso da inteligência artificial no controle semafórico, para os principais sistemas semafóricos; e ainda quais são os passos considerados para se avaliar a viabilidade de implementação de um semáforo e ainda as etapas para que essa implementação ocorra. Desta forma, o trabalho de conclusão de curso é elaborado considerando todas as especificidades relacionadas ao longo de seu desenvolvimento.

A implementação dos semáforos inteligentes nas cidades se faz necessário para melhorar o fluxo dos veículos e prevenir acidentes, conforme apresentado no código da simulação a implementação destes semáforos é algo que pode ser feito a fim de melhorar o fluxo dos veículos conforme as demandas nas vias, já que é algo que não é preciso ficar configurando o tempo de cada ciclo do semáforo, já que quando uma IA é implementada nestes semáforos elas conseguem calcular o tempo de acordo com a quantidade de veículos que estão esperando o sinal abrir do lado contrário da via que está aberta. Não é algo impossível de ser feito, como mostrado neste Trabalho de Conclusão de Curso.

Com os anos a melhorias dos semáforos vem se fazendo presente no dia a dia das pessoas, tanto para melhorar o fluxo dos veículos quanto para melhorar a travessia dos pedestres com segurança em lugares muito movimentados, onde uma passarela não se faz viável e muito menos possível de ser instalada.

Para futuras pesquisas é aconselhável um aprofundamento no tema proposto, os benefícios e a segurança dos semáforos inteligentes no dia a dia das pessoas, tanto motoristas quanto pedestres, as dificuldades de aplicar estes semáforos em grande escala.

No futuro esses semáforos inteligentes, além de melhorar o fluxo do trânsito e diminuir os congestionamentos, irão diminuir consideravelmente a emissão de poluentes na atmosfera, trarão mais segurança para todos, coordenando a travessia de veículos e pedestres, diminuindo significativamente os acidentes.

Termo de autorização de publicação junto a PUC, na página 49 em anexo 1
O Código, se apresenta na página 50 em anexo 2.

5 REFERÊNCIAS

ANDRADE, Michelle. **Estudo para aprimoramento da operação de controladores semafóricos fuzzy.** 150p. Dissertação de Mestrado. Universidade de Brasília: Brasilia, 2004.

ARAÚJO, S. C. **Controlador de Tráfego: Semáforo Inteligente.** Trabalho de Conclusão do Curso de Engenharia de Computação, Faculdade de Ciencias e Tecnologia do Centro Universitário de Brasília, Distrito Federal - DF, 2006.

BONETTI, W; PIETRANTONIO, H. Utilização de Semáforos Atuados pelo Tráfego. **Artigo técnico**, EMDEC, Campinas-SP, Set. 2001.

BORGES, Bruno. **A LINGUAGEM PYTHON COMO FERRAMENTA NO ENSINO BÁSICO**. Tese (Mestrado Profissional em Matemática em Rede Nacional) – Universidade Federal do Triângulo Mineiro, Uberaba, MG, 2021.

CORTEZ, Diogo Eugênio Garcia. **DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE DE TRÁFEGO INTELIGENTE BASEADO EM VISÃO COMPUTACIONAL**. 2022. 110p. Dissertação de Mestrado. PPGTI, UFRGN, Natal RN. Disponível em: https://repositorio.ufrn.br/bitstream/123456789/47158/1/Desenvolvimentosistemacontrole_Cortez_2022.pdf Acesso em: 31 Mar. 2023.

CUNHA, Lucas Costa et al. Lógica para Semáforo Inteligente Baseado na Mineração de Dados por Algoritmo Genético Transgênico. **X Encontro Anual de Computação – EnAComp**, 2013. Disponível em: https://www.enacomp.com.br/2013/anais/pdf/40.pdf Acesso em: 28 Mai. 2023.

CRUZ, Wellington. **Aplicaçao de Algoritmos Genéticos em Semáforos Inteligentes.** 2011. Trabalho de Conclusão de Curso. Universidade Presbiteriana Mackenzie: São Paulo. Disponível em: http://www.sinaldetransito.com.br/artigos/algoritmo_genetico.pdf Acesso em: 28 Mai. 2023.

DENATRAN. **Volume V** – Sinalização Semafórica. Manual Brasileiro de Sinalização de Trânsito, 2014

FERREIRA, João Victor Versari. **Estudo de implantação semafórica na região central da cidade de Campo Mourão**. 2019. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

IBGE. **Frota de veículos no Brasil**, 2020. Disponível em: https://cidades.ibge.gov.br/brasil/pesquisa/22/28120?ano=2020. Acesso em: 14 abr. 2023.

KRINGER, Daniel. **O QUE É PYTHON, PARA QUE SERVE E PORQUE APREDER?**. Kezier, 2022. Disponível em: https://kenzie.com.br/blog/o-que-e-python//. Acesso em: 19 Nov. 2023.

O que é o visual studio. Microsoft, 2023. Disponível em: https://learn.microsoft.com/pt-br/visualstudio/get-started/visual-studio-ide?view=vs-2022. Acesso em: 19 Nov. 2023.

REGO, Rosana Cibely; SEMENTE, Rodrigo. Sistema de controle de semáforo baseado ´ na densidade de trafego. **Anais do Encontro de Computação do Oeste Potiguar ECOP/UFERSA 2017** ISSN 2526-7574 – Pau dos Ferros/RN, v. 1, p. 7-14, jun. 2017.

SERRA, M.R.G. Projeto do Controlador Programável de Semáforo (CPS II) com Tempo Variável. **Monografia de conclusão da graduação**, UFMA, São Luis, Abr. 2001.

SERRA, M.R.G. **APLICAÇÕES DE APRENDIZAGEM POR REFORÇO EM CONTROLE DE TRÁFEGO VEICULAR URBANO.** 2004. 96p. Dissertação de Mestrado. UFSC, Florianópolis. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/87535/206482.pdf?sequence=1&isAllowed=y Acesso em: 21 Mar. 2023.

Tkinter: Interfaces gráficas em python. DEVMEDIA. Disponível em: https://www.devmedia.com.br/tkinter-interfaces-graficas-em-python/33956. Acesso em: 19 Nov. 2023).

YUKI, Hélio Saburo; FERREIRA, Luís Otávio Saraiva. **Projeto de Controlador Inteligente para Semáforo**. Projeto, 26p. Unicamp: São Paulo, 2008.

Anexo 1

PUC GOIÁS

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR
Av. Universitária, 1069 a Setor Universitário
Caixa Postal 86 a CEP 74605-010
Goiânia a Goiás a Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br a reitoria@pucgoias.edu.br

RESOLUÇÃO nº 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante Hugo Campos da Silva
do Curso de Engenharia da Computação, matrícula 201810033017 65,
telefone: (62) 98411-3640 e-mail hugo.campos9@gmail.com,
na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado Controle Inteligente de Tráfego Terrestre: Uma revisão Literária, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 30 de Agosto de 2023.

Assinatura do autor: Hugo Campos da Silva

Nome completo do autor: Hugo Campos da Silva

Assinatura do professor–orientador:

Nome completo do professor-orientador: Marcelo Otávio Cobral de Araújo

Anexo 2

```
#Bibliotecas utilizadas
import time
import random
from tkinter import *
from tkinter import messagebox

#Função para adicionar carros subindo ou descendo o Cruzamento 1
def ad_carro_subindo():
    #Variaveis globais
    global j
    global k
    global n
    global m
    #Variavel para escolha das vias
    var = [0,1,2,3]
    #Escolha randomica da via
    i = random.choice(var)
    #Se a escolha da via for "0"
    if i == 0:
        #Adiciona passo para a proxima execução
        j = j+35
        #Caso a via tenha menos de 2 carros
        if len(l) <= 1:
            #Cria carros na posições 1 e 2 da via
            l.append(canvas.create_rectangle(160+j, 160, 190+j, 130, fill="black"))
        #Caso a via tenha menos de 4 carros
        elif(len(l) <= 3):
            #Cria carros na posições 3 e 4 da via
            l.append(canvas.create_rectangle(90+j, 90, 120+j, 120, fill="black"))
        #Caso a via tenha menos de 6 carros
        elif(len(l) <= 5):
            #Cria carros na posições 5 e 6 da via
            l.append(canvas.create_rectangle(20+j, 50, 50+j, 80, fill="black"))
    #Se a escolha da via for "1"
    if i == 1:
        #Adiciona passo para a proxima execução
        k = k+35
        #Caso a via tenha menos de 2 carros
        if len(p) <= 1:
            #Cria carros na posições 1 e 2 da via
            p.append(canvas.create_rectangle(220+k, 335, 250+k, 305, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(p) <= 3):
            #Cria carros na posições 3 e 4 da via
            p.append(canvas.create_rectangle(150+k, 370, 180+k, 340, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(p) <= 5):
```

```
            #Cria carros na posições 5 e 6 da via
            p.append(canvas.create_rectangle(80+k, 405, 110+k, 375, fill="black"))

    #Se a escolha da via for "2"
    if i == 2:
        #Adiciona passo para a proxima execução
        n = n+35
        #Caso a via tenha menos de 2 carros
        if len(l1) <= 1:
            #Cria carros na posições 1 e 2 da via
            l1.append(canvas.create_rectangle(620+n, 125, 650+n, 155, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(l1) <= 3):
            #Cria carros na posições 3 e 4 da via
            l1.append(canvas.create_rectangle(550+n, 90, 580+n, 120, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(l1) <= 5):
            #Cria carros na posições 5 e 6 da via
            l1.append(canvas.create_rectangle(480+n, 50, 510+n, 85, fill="black"))
    #Se a escolha da via for "3"
    if i == 3:
        #Adiciona passo para a proxima execução
        m = m+35
        #Caso a via tenha menos de 2 carros
        if len(p1) <= 1:
            #Cria carros na posições 1 e 2 da via
            p1.append(canvas.create_rectangle(680+m, 335, 710+m, 305, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(p1) <= 3):
            #Cria carros na posições 3 e 4 da via
            p1.append(canvas.create_rectangle(610+m, 370, 640+m, 340, fill="black"))
        #Caso a via tenha menos de 2 carros
        elif(len(p1) <= 5):
            #Cria carros na posições 5 e 6 da via
            p1.append(canvas.create_rectangle(540+m, 405, 570+m, 375, fill="black"))
    #Caso a quantidade de carros nas vias seja maior ou igual a 6
    if (len(l) + len(p) >= 6) or (len(l1) + len(p1) >= 6):
        #Atualiza a tela
        tela.update()
        #Desabilita o botão
        AD_subindo.configure(state=DISABLED)
        #Chama a função para deixar o sinal amarelo
        amarelo_1()

#Função para deixar o semáforo amarelo
def amarelo_1():
    #Configuração das cores do primeiro semaforo do cruzamento
    canvas.itemconfig(oval_red_1, fill="gray")
    canvas.itemconfig(oval_yellow_1, fill="yellow")
    canvas.itemconfig(oval_green_1, fill="grey")
```

```
#Configuração das cores do segundo semaforo do cruzamento
canvas.itemconfig(oval_red_2, fill="grey")
canvas.itemconfig(oval_yellow_2, fill="yellow")
canvas.itemconfig(oval_green_2, fill="grey")
#Configuração das cores do terceiro semaforo do cruzamento
canvas.itemconfig(oval_red_3, fill="red")
canvas.itemconfig(oval_yellow_3, fill="grey")
canvas.itemconfig(oval_green_3, fill="grey")
#Configuração das cores do quarto semaforo do cruzamento
canvas.itemconfig(oval_red_4, fill="red")
canvas.itemconfig(oval_yellow_4, fill="grey")
canvas.itemconfig(oval_green_4, fill="grey")
#Configuração das cores do quinto semaforo do cruzamento
canvas.itemconfig(oval_red_5, fill="gray")
canvas.itemconfig(oval_yellow_5, fill="yellow")
canvas.itemconfig(oval_green_5, fill="grey")
#Configuração das cores do sexto semaforo do cruzamento
canvas.itemconfig(oval_red_6, fill="grey")
canvas.itemconfig(oval_yellow_6, fill="yellow")
canvas.itemconfig(oval_green_6, fill="grey")
#Configuração das cores do setimo semaforo do cruzamento
canvas.itemconfig(oval_red_7, fill="red")
canvas.itemconfig(oval_yellow_7, fill="grey")
canvas.itemconfig(oval_green_7, fill="grey")
#Configuração das cores do oitavo semaforo do cruzamento
canvas.itemconfig(oval_red_8, fill="red")
canvas.itemconfig(oval_yellow_8, fill="grey")
canvas.itemconfig(oval_green_8, fill="grey")
#Atualiza a tela
tela.update()
#Chama a função para contar o tempo
temporizador(t)
#Reconfigura o sinal amarelo de todos os semáforos
canvas.create_rectangle(215, 255, 235, 270,fill='grey')
canvas.create_rectangle(280, 190, 300, 205,fill='grey')
canvas.create_rectangle(675, 255, 695, 270,fill='grey')
canvas.create_rectangle(740, 190, 760, 205,fill='grey')
canvas.itemconfig(timer, fill='grey')

#Configuração das cores do primeiro semafDoro do cruzamento
canvas.itemconfig(oval_red_1, fill="red")
canvas.itemconfig(oval_yellow_1, fill="grey")
canvas.itemconfig(oval_green_1, fill="grey")
#Configuração das cores do segundo semaforo do cruzamento
canvas.itemconfig(oval_red_2, fill="red")
canvas.itemconfig(oval_yellow_2, fill="grey")
canvas.itemconfig(oval_green_2, fill="grey")
#Configuração das cores do terceiro semaforo do cruzamento
canvas.itemconfig(oval_red_3, fill="grey")
canvas.itemconfig(oval_yellow_3, fill="grey")
```

```
    canvas.itemconfig(oval_green_3, fill="green")
    #Configuração das cores do quarto semaforo do cruzamento
    canvas.itemconfig(oval_red_4, fill="grey")
    canvas.itemconfig(oval_yellow_4, fill="grey")
    canvas.itemconfig(oval_green_4, fill="green")
    #Configuração das cores do quinto semaforo do cruzamento
    canvas.itemconfig(oval_red_5, fill="red")
    canvas.itemconfig(oval_yellow_5, fill="grey")
    canvas.itemconfig(oval_green_5, fill="grey")
    #Configuração das cores do sexto semaforo do cruzamento
    canvas.itemconfig(oval_red_6, fill="red")
    canvas.itemconfig(oval_yellow_6, fill="grey")
    canvas.itemconfig(oval_green_6, fill="grey")
    #Configuração das cores do setimo semaforo do cruzamento
    canvas.itemconfig(oval_red_7, fill="grey")
    canvas.itemconfig(oval_yellow_7, fill="grey")
    canvas.itemconfig(oval_green_7, fill="green")
    #Configuração das cores do oitavo semaforo do cruzamento
    canvas.itemconfig(oval_red_8, fill="grey")
    canvas.itemconfig(oval_yellow_8, fill="grey")
    canvas.itemconfig(oval_green_8, fill="green")
    #Atualiza a tela
    tela.update()
    #Chama a função para remover os carros
    rem_carro_subindo()

#Função para remover carros subindo ou descendo o Cruzamento 1
def rem_carro_subindo():
    #Variaveis globais
    global j
    global k
    global n
    global m
    #Remove os carros um por um
    for i in range(6):
        #Aguarda um segundo antes de remover o proximo carro
        time.sleep(1)
        #Atualiza a janela
        tela.update()
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(l[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(p[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
```

```
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(l1[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(p1[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
                pass
    #Limpa as variaveis
    l.clear()
    p.clear()
    l1.clear()
    p1.clear()
    j = 0
    k = 0
    n = 0
    m = 0
    #Ativa o botão
    AD_atravessando.configure(state=ACTIVE)
    #Atualiza a janela
    tela.update()

#Função para adicionar carros subindo ou descendo o Cruzamento 2
def ad_carro_atravessando():
    #Variaveis globais
    global j
    global k
    global n
    global m
    #Variavel para escolha das vias
    var = [0,1,2,3]
    #Escolha randomica da via
    i = random.choice(var)
    #Se a escolha da via for "0"
    if i == 0:
        #Adiciona passo para a proxima execução
        j = 35-j
        #Caso a via tenha menos de 2 carros
        if len(l) <= 1:
            #Cria carros na posições 1 e 2 da via
            l.append(canvas.create_rectangle(155, 235+j, 185, 265+j, fill="black"))
        elif(len(l) <= 3):
            #Cria carros na posições 3 e 4 da via
            l.append(canvas.create_rectangle(120, 235+j, 150, 265+j, fill="black"))
        elif(len(l) <= 5):
```

```
            #Cria carros na posições 5 e 6 da via
            l.append(canvas.create_rectangle(85, 235+j, 115, 265+j, fill="black"))
    #Se a escolha da via for "0"
    if i == 1:
        #Adiciona passo para a proxima execução
        k = 35-k
        #Caso a via tenha menos de 2 carros
        if len(p) <= 1:
            #Cria carros na posições 1 e 2 da via
            p.append(canvas.create_rectangle(330, 170+k, 360, 200+k, fill="black"))
        elif(len(p) <= 3):
            #Cria carros na posições 3 e 4 da via
            p.append(canvas.create_rectangle(365, 170+k, 395, 200+k, fill="black"))
        elif(len(l) <= 5):
            #Cria carros na posições 5 e 6 da via
            l.append(canvas.create_rectangle(400, 170+k, 430, 200+k, fill="black"))
    #Se a escolha da via for "0"
    if i == 2:
        #Adiciona passo para a proxima execução
        n = 35-n
        #Caso a via tenha menos de 2 carros
        if len(l1) <= 1:
            #Cria carros na posições 1 e 2 da via
            l1.append(canvas.create_rectangle(610, 235+n, 640, 265+n, fill="black"))
        elif(len(l1) <= 3):
            #Cria carros na posições 3 e 4 da via
            l1.append(canvas.create_rectangle(575, 235+n, 605, 265+n, fill="black"))
        elif(len(l1) <= 5):
            #Cria carros na posições 5 e 6 da via
            l1.append(canvas.create_rectangle(540, 235+n, 570, 265+n, fill="black"))
    #Se a escolha da via for "0"
    if i == 3:
        #Adiciona passo para a proxima execução
        m = 35-m
        #Caso a via tenha menos de 2 carros
        if len(p1) <= 1:
            #Cria carros na posições 1 e 2 da via
            p1.append(canvas.create_rectangle(790, 170+m, 820, 200+m, fill="black"))
        elif(len(p1) <= 3):
            #Cria carros na posições 3 e 4 da via
            p1.append(canvas.create_rectangle(825, 170+m, 855, 200+m, fill="black"))
        elif(len(p1) <= 5):
            #Cria carros na posições 5 e 6 da via
            p1.append(canvas.create_rectangle(860, 170+m, 890, 200+m, fill="black"))
    #Caso a quantidade de carros nas vias seja maior ou igual a 6
    if (len(l) + len(p) >= 6) or (len(l1) + len(p1) >= 6):
        #Atualiza a tela
        tela.update()
        #Desabilita o botão
        AD_atravessando.configure(state=DISABLED)
```

```
        #Chama a função para deixar o sinal amarelo
        amarelo_2()

#Função para deixar o semáforo amarelo
def amarelo_2():
    #Configuração das cores do primeiro semaforo do cruzamento
    canvas.itemconfig(oval_red_1, fill="red")
    canvas.itemconfig(oval_yellow_1, fill="grey")
    canvas.itemconfig(oval_green_1, fill="grey")
    #Configuração das cores do segundo semaforo do cruzamento
    canvas.itemconfig(oval_red_2, fill="red")
    canvas.itemconfig(oval_yellow_2, fill="grey")
    canvas.itemconfig(oval_green_2, fill="grey")
    #Configuração das cores do terceiro semaforo do cruzamento
    canvas.itemconfig(oval_red_3, fill="grey")
    canvas.itemconfig(oval_yellow_3, fill="yellow")
    canvas.itemconfig(oval_green_3, fill="grey")
    #Configuração das cores do quarto semaforo do cruzamento
    canvas.itemconfig(oval_red_4, fill="grey")
    canvas.itemconfig(oval_yellow_4, fill="yellow")
    canvas.itemconfig(oval_green_4, fill="grey")
    #Configuração das cores do quinto semaforo do cruzamento
    canvas.itemconfig(oval_red_5, fill="red")
    canvas.itemconfig(oval_yellow_5, fill="grey")
    canvas.itemconfig(oval_green_5, fill="grey")
    #Configuração das cores do sexto semaforo do cruzamento
    canvas.itemconfig(oval_red_6, fill="red")
    canvas.itemconfig(oval_yellow_6, fill="grey")
    canvas.itemconfig(oval_green_6, fill="grey")
    #Configuração das cores do setimo semaforo do cruzamento
    canvas.itemconfig(oval_red_7, fill="grey")
    canvas.itemconfig(oval_yellow_7, fill="yellow")
    canvas.itemconfig(oval_green_7, fill="grey")
    #Configuração das cores do oitavo semaforo do cruzamento
    canvas.itemconfig(oval_red_8, fill="grey")
    canvas.itemconfig(oval_yellow_8, fill="yellow")
    canvas.itemconfig(oval_green_8, fill="grey")
    #Atualiza a tela
    tela.update()
    #Chama a função para contar o tempo
    temporizador_1(t)

    #Reconfigura o sinal amarelo de todos os semáforos
    canvas.create_rectangle(215, 190, 230, 210,fill='grey')
    canvas.create_rectangle(280, 255, 295, 275,fill='grey')
    canvas.create_rectangle(685, 190, 700, 210,fill='grey')
    canvas.create_rectangle(740, 255, 755, 275,fill='grey')
    canvas.itemconfig(timer2, fill='grey')
    #Configuração das cores do primeiro semaforo do cruzamento
    canvas.itemconfig(oval_red_1, fill="grey")
```

```
    canvas.itemconfig(oval_yellow_1, fill="grey")
    canvas.itemconfig(oval_green_1, fill="green")
    #Configuração das cores do segundo semaforo do cruzamento
    canvas.itemconfig(oval_red_2, fill="grey")
    canvas.itemconfig(oval_yellow_2, fill="grey")
    canvas.itemconfig(oval_green_2, fill="green")
    #Configuração das cores do terceiro semaforo do cruzamento
    canvas.itemconfig(oval_red_3, fill="red")
    canvas.itemconfig(oval_yellow_3, fill="grey")
    canvas.itemconfig(oval_green_3, fill="grey")
    #Configuração das cores do quarto semaforo do cruzamento
    canvas.itemconfig(oval_red_4, fill="red")
    canvas.itemconfig(oval_yellow_4, fill="grey")
    canvas.itemconfig(oval_green_4, fill="grey")
    #Configuração das cores do quinto semaforo do cruzamento
    canvas.itemconfig(oval_red_5, fill="grey")
    canvas.itemconfig(oval_yellow_5, fill="grey")
    canvas.itemconfig(oval_green_5, fill="green")
    #Configuração das cores do sexto semaforo do cruzamento
    canvas.itemconfig(oval_red_6, fill="grey")
    canvas.itemconfig(oval_yellow_6, fill="grey")
    canvas.itemconfig(oval_green_6, fill="green")
    #Configuração das cores do setimo semaforo do cruzamento
    canvas.itemconfig(oval_red_7, fill="red")
    canvas.itemconfig(oval_yellow_7, fill="grey")
    canvas.itemconfig(oval_green_7, fill="grey")
    #Configuração das cores do oitavo semaforo do cruzamento
    canvas.itemconfig(oval_red_8, fill="red")
    canvas.itemconfig(oval_yellow_8, fill="grey")
    canvas.itemconfig(oval_green_8, fill="grey")
    #Atualiza a tela
    tela.update()
    #Chama a função para remover os carros
    rem_carro_atravessando()

#Função para remover carros subindo ou descendo o Cruzamento 2
def rem_carro_atravessando():
    #Variaveis globais
    global j
    global k
    global n
    global m
    #Remove os carros um por um
    for i in range(6):
        #Aguarda um segundo antes de remover o proximo carro
        time.sleep(1)
        #Atualiza a janela
        tela.update()
        #Remove o carro caso tenha
        try:
```

```
            canvas.itemconfig(l[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(p[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(l1[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
            pass
        #Remove o carro caso tenha
        try:
            canvas.itemconfig(p1[i], fill="white", outline="white")
        #Caso não tenha mais carros
        except:
                pass

    #Limpa as variaveis
    l.clear()
    p.clear()
    l1.clear()
    p1.clear()
    j = 0
    k = 0
    n = 0
    m = 0
    #Ativa o botão
    AD_atravessando.configure(state=ACTIVE)
    #Atualiza a janela
    tela.update()

#Função do temporizador
def temporizador(t):
    #Variavel global
    global timer
    #Enquanto o tempo for diferete de "0"
    while t != -1:
        #Transforma o tempo em segundos
        min,sec = divmod(t,60)
        #Transfere somente os segundos para a variavel "TIMER"
        timer = '{:02d}'.format(sec)
        #Configura o sinal amarelo
        canvas.create_rectangle(215,255, 235, 270,fill='yellow')
        #Mostra o tempo no sinal
```

```
        canvas.create_text(225, 262, text=timer, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(280, 190, 300, 205,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(290, 198, text=timer, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(675,255, 695, 270,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(685, 262, text=timer, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(740, 190, 760, 205,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(750, 198, text=timer, fill='black')
        #Atualiza o tempo
        canvas.itemconfig(timer)
        #Atualiza a tela
        tela.update()
        #Aguarda 1 segundo
        time.sleep(1)
        t -= 1

#Função do temporizador
def temporizador_1(t):
    #Variavel global
    global timer2
    #Enquanto o tempo for diferete de "0"
    while t != -1:
        #Transforma o tempo em segundos
        min, sec = divmod(t,60)
        #Transfere somente os segundos para a variavel "TIMER"
        timer2 = '{:02d}'.format(sec)
        #Configura o sinal amarelo
        canvas.create_rectangle(215, 190, 230, 210,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(223, 200, text=timer2, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(280, 255, 295, 275,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(288, 265, text=timer2, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(685, 190, 700, 210,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(693, 200, text=timer2, fill='black')
        #Configura o sinal amarelo
        canvas.create_rectangle(740, 255, 755, 275,fill='yellow')
        #Mostra o tempo no sinal
        canvas.create_text(748, 265, text=timer2, fill='black')
        #Atualiza o tempo
        canvas.itemconfig(timer2)
        #Atualiza a tela
```

```
        tela.update()
        #Aguarda 1 segundo
        time.sleep(1)
        t -= 1

#Função Main
if __name__ == '__main__':
    #Cria a janela da simulação
    tela = Tk()
    #Coloca a janela no centro da tela
    tela.eval('tk::PlaceWindow . center')
    #Titulo da janela
    tela.title('Semaforo')
    #Variaveis utilizadas
    j = 0
    k = 0
    n = 0
    m = 0
    l = []
    p = []
    l1 = []
    p1 = []
    t = 5
    #Cria um quadrado branco
    canvas = Canvas(tela, width=1000, height=500, bg="white")
    canvas.pack()

    #X, Y, largura, altura
    #Primeira Via do cruzamento 1
    canvas.create_line(190,165,190,0)
    canvas.create_line(325,165,325,0)
    #Segunda Via do cruzamento 1
    canvas.create_line(190,165,0,165)
    canvas.create_line(190,305,0,305)
    #Terceira Via do cruzamento 1
    canvas.create_line(190,500,190,305)
    canvas.create_line(325,500,325,305)
    #Quarta Via do cruzamento 1
    canvas.create_line(325,165,650,165)
    canvas.create_line(325,305,650,305)

    #Cria o retangulo do primeiro semáforo do cruzamento 1
    canvas.create_rectangle(190, 250, 260, 275,fill='black')
    #Cria as formas do semáforo
    oval_green_1 = canvas.create_oval(195, 255, 210, 270, fill="green")
    oval_yellow_1 = canvas.create_rectangle(215,255, 235, 270, fill="grey")
    oval_red_1 = canvas.create_oval(240, 255, 255, 270, fill="grey")
    #Cria o retangulo do segundo semáforo do cruzamento 1
    canvas.create_rectangle(255, 185, 325, 210,fill='black')
    #Cria as formas do semáforo
```

```
oval_red_2 = canvas.create_oval(260, 190, 275, 205, fill="grey")
oval_yellow_2 = canvas.create_rectangle(280, 190, 300, 205, fill="grey")
oval_green_2 = canvas.create_oval(305, 190, 320, 205, fill="green")
#Cria o retangulo do terceiro semáforo do cruzamento 1
canvas.create_rectangle(210, 165, 235, 235,fill='black')
#Cria as formas do semáforo
oval_green_3 = canvas.create_oval(215, 170, 230, 185, fill="grey")
oval_yellow_3 = canvas.create_rectangle(215, 190, 230, 210, fill="grey")
oval_red_3 = canvas.create_oval(215, 215, 230, 230, fill="red")
#Cria o retangulo do quarto semáforo do cruzamento 1
canvas.create_rectangle(275, 230, 300, 300,fill='black')
#Cria as formas do semáforo
oval_red_4 = canvas.create_oval(280, 235, 295, 250, fill="red")
oval_yellow_4 = canvas.create_rectangle(280, 255, 295, 275, fill="grey")
oval_green_4 = canvas.create_oval(280, 280, 295, 295, fill="grey")

#X, Y, largura, altura
#Primeira Viado cruzamento 1
canvas.create_line(650,165,650,0)
canvas.create_line(785,165,785,0)
#Segunda Viado cruzamento 1
canvas.create_line(650,500,650,305)
canvas.create_line(785,500,785,305)
#Terceira Via do cruzamento 1
canvas.create_line(190,500,190,305)
canvas.create_line(325,500,325,305)
#Quarta Viado cruzamento 1
canvas.create_line(785,165,1000,165)
canvas.create_line(785,305,1000,305)

#Cria o retangulo do quinto semáforo do cruzamento 1
canvas.create_rectangle(650, 250, 720, 275,fill='black')
#Cria as formas do semáforo
oval_green_5 = canvas.create_oval(655, 255, 670, 270, fill="green")
oval_yellow_5 = canvas.create_rectangle(675,255, 695, 270, fill="grey")
oval_red_5 = canvas.create_oval(700, 255, 715, 270, fill="grey")
#Cria o retangulo do sexto semáforo do cruzamento 1
canvas.create_rectangle(715, 185, 785, 210,fill='black')
#Cria as formas do semáforo
oval_red_6 = canvas.create_oval(720, 190, 735, 205, fill="grey")
oval_yellow_6 = canvas.create_rectangle(740, 190, 760, 205, fill="grey")
oval_green_6 = canvas.create_oval(765, 190, 780, 205, fill="green")
#Cria o retangulo do setimo semáforo do cruzamento 1
canvas.create_rectangle(680, 165, 705, 235,fill='black')
#Cria as formas do semáforo
oval_green_7 = canvas.create_oval(685, 170, 700, 185, fill="grey")
oval_yellow_7 = canvas.create_rectangle(685, 190, 700, 210, fill="grey")
oval_red_7 = canvas.create_oval(685, 215, 700, 230, fill="red")
#Cria o retangulo do oitavo semáforo do cruzamento 1
canvas.create_rectangle(735, 230, 760, 300,fill='black')
```

```
    #Cria as formas do semáforo
    oval_red_8 = canvas.create_oval(740, 235, 755, 250, fill="red")
    oval_yellow_8 = canvas.create_rectangle(740, 255, 755, 275, fill="grey")
    oval_green_8 = canvas.create_oval(740, 280, 755, 295, fill="grey")

    #Cria o botão para simular a camera do semáforo, e adiciona o comando
    AD_subindo = Button(text = "        AD Subindo", command = ad_carro_subindo,
anchor = W)
    #Configura o botão
    AD_subindo.configure(width = 15, activebackground = "#33B5E5", relief = FLAT)
    #Cria uma janela para atrbuir o botão
    AD_subindo_window = canvas.create_window(350, 350, anchor=NW,
window=AD_subindo)
    #Cria o botão para simular a camera do semáforo, e adiciona o comando
    AD_atravessando = Button(text = "    AD Atravessando", command =
ad_carro_atravessando, anchor = W)
    #Configura o botão
    AD_atravessando.configure(width = 15, activebackground = "#33B5E5", relief =
FLAT)
    #Cria uma janela para atrbuir o botão
    AD_atravessando_window = canvas.create_window(520, 350, anchor=NW,
window=AD_atravessando)

    #Bloqueia o botão
    AD_atravessando.configure(state=DISABLED)

    #Mantem a janela da simulação ativa
    tela.mainloop()
```