Simulador de trânsito

Jesuino Vieira Filho
Departamento de Engenharias da Mobilidade
Universidade Federal de Santa Catarina - Centro Tecnológico de Joinville
Joinville, Brasil
jesuinovieira_@hotmail.com

Resumo

O intuito deste relatório é apresentar o desenvolvimento de uma simulação de tráfego na linguagem de programação C++, realizado como trabalho da disciplina de Programação III do segundo semestre do ano de 2017. Dividido em quatro seções principais, o documento inicialmente apresenta uma breve introdução ao projeto, tais como um comentário sobre as ferramentas de simulação e o funcionamento dessa aplicação. A segunda parte contém todo o desenvolvimento do programa, expondo a abordagem do problema, a implementação - tanto da simulação quanto da interface gráfica - e as aplicações das ferramentas da orientação ao objeto. O próximo tópico apresenta uma breve discussão sobre o projeto, como por exemplo, algumas dificuldades encontradas durante realização e o esclarecimento de decisões que tiveram de ser tomadas. Finalmente, uma sessão com as conclusões encerra o documento, abordando o ponto de vista do autor sobre o desenvolvimento do projeto, assim como o progresso obtido com a realização.

I. INTRODUÇÃO

A. Simulação

A simulação de trânsito é uma ferramenta de análise computadorizada que permite a avaliação de intervenções antes de serem implementadas no mundo real. As operações são modeladas como uma sequência de eventos discretos no tempo, sendo assim, cada evento ocorre em um determinado instante de tempo e marca uma mudança de estado no sistema. Ferramentas de simulação realizam análises altamente detalhadas de atividades, desde tráfegos rodoviários, transações financeiras até doenças que se propagam através de uma população. Dessa forma, essas aplicações vem sendo muito utilizadas para auxiliar o estudo na execução de projetos relacionados a mobilidade, permitindo avaliar seus resultados e julgar se devida alteração será benéfica e condizente com o esperado. Assim, além de evitar o desperdício de dinheiro e tempo em caso de falha, traz facilidade para a modificação de detalhes e a aplicação de uma nova avaliação. Em síntese, o propósito desse projeto é implementar uma simulação de tráfego em dois cruzamentos por um período de vinte e quatro horas virtuais e, adicionalmente, expor a quantidade de veículos que entram e saem do sistema.

B. Funcionamento

O programa inicia criando automaticamente um carro em cada uma das seis pistas fonte. Para cada criação de um veículo é adicionado um novo evento na fila de eventos, o qual se refere a próxima entrada de carro na pista. Esse tempo é definido por uma distribuição randômica uniforme referente a cada uma delas, possuindo assim, uma frequência pré-determinada:

Pista	Frequencia	Limite Inferior	Limite Superior
O1Leste	10 +/- 2	8	12
S1Norte	30 +/- 7	23	37
S2Norte	60 +/- 15	45	75
L1Oeste	10 +/- 2	8	12
N2Sul	20 +/- 5	15	25
N1Sul	20 +/- 5	15	25

Tabela 01. Faixa de frequência de cada pista fonte.

Além disso, cada pista possui uma velocidade e um vetor de probabilidades, este último é referente a uma distribuição que modela para qual de suas pistas eferentes um carro vai ir - no caso das pistas sumidouro, o veiculo sai do sistema. Tendo em vista que o tempo na simulação é representado em segundos:

Pista	Velocidade (m/s)	Vetor de probabilidades	Pistas Eferentes
O1Leste	22	[80 10 10]	[C1Leste N1Norte S1Sul]
S1Norte	16	[80 10 10]	[C1Leste N1Norte O1Oeste]
S2Norte	11	[40 30 30]	[L1Leste S2Sul C1Oeste]
L1Oeste	08	[40 30 30]	[N2Norte C1Leste S2Sul]
N2Sul	11	[40 30 30]	[L1Leste C1Oeste S2Sul]
N1Sul	16	[80 10 10]	[C1Leste O1Oeste S1Sul]
C1Oeste	16	[40 30 30]	[O1Oeste N1Norte S1Sul]
C1Leste	16	[40 30 30]	[L1Leste N2Norte S2Sul]
O1Oeste	22	[Sumidouro]	[Sumidouro]
S1Sul	16	[Sumidouro]	[Sumidouro]
S2Sul	11	[Sumidouro]	[Sumidouro]
L1Leste	08	[Sumidouro]	[Sumidouro]
N2Norte	11	[Sumidouro]	[Sumidouro]
N1Norte	16	[Sumidouro]	[Sumidouro]

Tabela 02. Velocidade das pistas e faixa de valores de cada pista eferente.

Desse modo, a simulação vai atualizando para cada pista a posição dos veículos e realizando a troca de estado dos semáforos. Quando um carro percorre toda a pista, são realizadas duas verificações: se o semáforo esta aberto e se a pista destino não está cheia. Atendendo essas condições, o veículo segue o seu caminho, caso contrário, permanece na pista causando congestionamento.

Durante a simulação é possível visualizar a quantidade de carros que entraram e saíram do sistema, assim como a quantidade de veículo em cada pista e o tempo total da simulação. Além disso, o usuário tem a possibilidade de pausar e reiniciar a simulação.

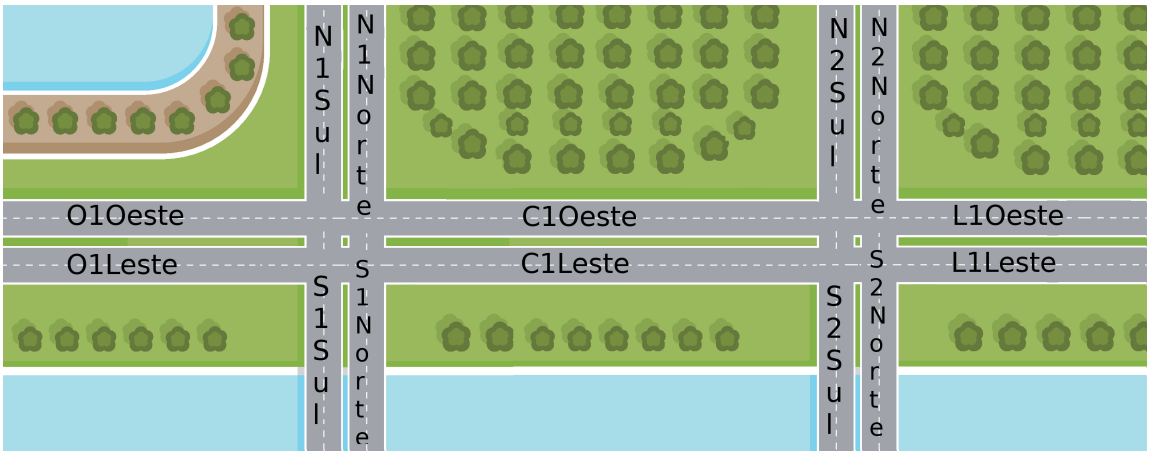

Figura 1. Rodovia.

II. DESENVOLVIMENTO

A. *Estruturação do programa*

Primeiramente, antes de iniciar a implementação do código, a base do projeto foi sendo desenvolvida no papel, com uma breve análise das classes que iriam compor o jogo e seus respectivos relacionamentos. Isso foi de grande importância para desenvolver uma ideia do funcionamento do programa, familiarizar-se com o problema e, consequentemente, auxiliar na codificação. Para a realização desse exame preliminar, cada detalhe teve de ser cuidadosamente considerado, tendo em vista que o projeto viria apresentar uma interface gráfica.

Após criar as classes básicas do sistema - tais como, Evento, Semáforo, Pista e Relógio, foi decidido que o tempo na simulação se passaria em segundos, dando liberdade para o usuário atualizar com maior ou menor frequência o tempo. Uma vez que a simulação representa 24 horas virtuais (86400 segundos de simulação referente ao relógio virtual), seria inviável não fornecer esse controle e deixar um tempo fixo para toda simulação realizada.

Feito isso, a tarefa seguinte foi pensar na representação dos eventos. Como o relógio possui uma fila de eventos, a classe Eventos armazenaria alguns de seus dados, como por exemplo: o tempo de ocorrência, o tipo, a pista em que o evento acontecerá e posteriormente foi adicionado um atributo para representar o tipo de pista, devido a distinção entre as pistas que não recebem carros (chamadas de pistas fonte) e as que recebem (chamadas de pistas receptoras). Esse detalhe poderia ser melhorado aplicando as técnicas de polimorfismo.

Dessa maneira, com a chegada dos veículos nos semáforos, um evento para troca de pista é enfileirado. Caso a pista destino não esteja disponível ou o semáforo esteja fechado, o evento é reenfileirado para o próximo segundo. No caso das pistas sumidouros, os veículos que a percorrem saem do sistema.

No primeiro momento após abrir o programa, o usuário deve iniciar a simulação. Após isso, tem a liberdade para pausar ou reiniciá-la. Estas funções estão dispostas na parte superior esquerda da interface.

Caso a simulação chegue ao fim, uma caixa de diálogo é aberta, informando o usuário.

Figura 2. Em sequência, da esquerda para a direita os botões para: iniciar, pausar e reiniciar a simulação.

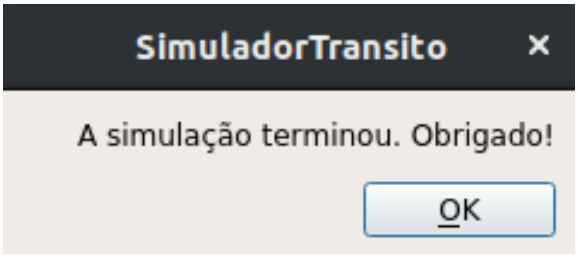

Figura 3. Caixa de diálogo exibida no final da simulação.

B. Implementação

1) Classe MainWindow: esta é a classe fundamental para a interface gráfica. Fornece uma janela principal para a aplicação a qual permite adicionar QWidgets que podem ficar sobre o controle de outras classes. Nesse projeto, a classe é composta principalmente por um ponteiro para Simulacao e três ponteiros para QTimer, responsáveis por: atualizar a tela, atualizar o tempo e atualizar os semáforos. A classe QTimer possui uma propriedade interessante e que foi muito útil na implementação, fornecendo temporizadores repetitivos e de disparos únicos.

2) Classe SimuladorView: através de uma herança publica de QGraphicsView, essa classe ficou responsável por alguns detalhes da interface, lidando desde a inicialização do plano de fundo da tela até a atualização da cena que esta sendo simulada. Basicamente, ela controla o Widget principal da janela, aonde ocorre a simulação. Permite visualizar seu conteúdo em uma janela de exibição rodável, possui também um método sobrescrito que lida com o zoom na tela através de eventos do mouse. Do mesmo modo, a classe é responsável por desenhar os veículos na cena.

3) Classe Semáforo: utilizada para representa os semáforos da simulação. Possui os atributos estado e tempo, e também uma função getter e setter para o estado.

4) Classe Pista: esta é uma das classes principais do programa e serve como classe base para duas outras: PistaEferente e PistaAferente. Possui uma lista de carros, um vetor que contém as posições de cada carro e métodos responsáveis por operações fundamentais na simulação, tais como: a movimentação dos veículos, troca de pista e entrada e saída de veículos em uma determinada pista.

5) Classe Relógio: foi utilizada para controlar o tempo da simulação e monitorar a fila de eventos, a qual é implementada através da priority_queue.

6) Classe Veículo: caracteriza os veículos da simulação e realiza a contagem dos veículos que saíram e entraram na simulação através de um atributo static.

7) Classe Evento: possui as informações necessárias para os eventos da simulação, são elas: tempo, tipo, pista e tipo da pista.

8) Classe Simulacao: baseada no padrão de projeto Fachada, provê uma interface unificada para um conjunto de interfaces de um subsistema. Sendo assim, gerencia as classes Pista, Semaforo e Relogio, de modo a construir a simulação de trânsito. Possui uma struct DadosVeiculo que foi utilizada para comunicar-se com as classes responsáveis por desenhar os carros na cena.

C. Interface gráfica

Aqui, o objetivo foi fornecer uma interface agradável, limpa, simples, atrativa e eficiente para o usuário. No início, a falta de familiaridade com o QtCreator foi uma grande barreira para o seu desenvolvimento. Apesar das dificuldades encontradas, um resultado satisfatório foi obtido, quem desfrutar da simulação deve possuir facilidade para o controle e clareza na visualização dos dados obtidos.

Além das classes MainWindow (criada automaticamente pelo projeto) e SimuladorView, o programa utiliza as seguintes classes para as constantes atualizações da janela principal:

1) Classe CarroImagem: através de uma herança pública de QGraphicsItem, é utilizada para desenhar na tela os carros da simulação.

2) Classe SemaforoImagem: essa classe também herda de QGraphicsItem. Todavia, ao invés de uma imagem, utiliza o método drawEllipse para pintar os semáforos da simulação. Eles são representados por um circulo de cor vermelha para os semáforos fechados ou de cor verde, para os abertos.

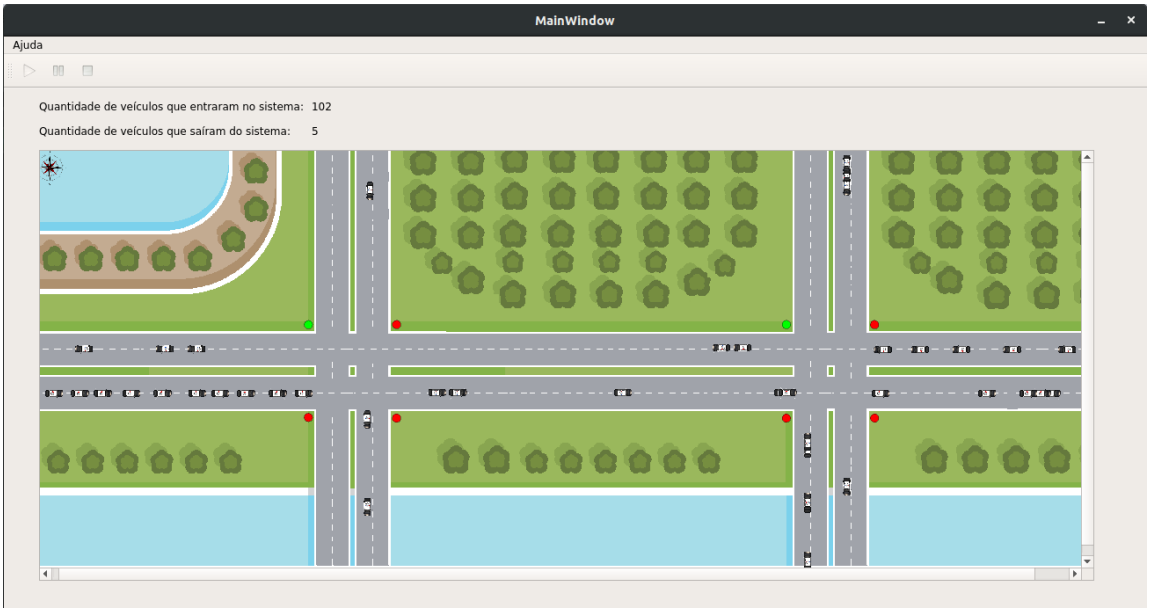

Figura 4. Exemplo de uma simulação em execução.

D. Fundamentos da POO

1) Encapsulamento: como recurso elementar da programação orientada a objeto, o encapsulamente foi respeitado por todas as classes, promovendo maior robustez ao código.

2) Composição: ferramenta fundamental utilizada em quase todas as classes.

3) Herança: ponto chave da POO, forneceu meios de promover a extensibilidade do código, a reutilização e uma maior coerência lógica no modelo de implementação. Além de herdar as classes que o QtCreator fornece, também foi utilizado na classe Pista, a qual serve como base para as classes PistaEferente e PistaAferente. Apesar do tempo não permitir a seguinte realização, a classe Pista poderia ser declarada como abstrata, o que permitiria o uso de polimorfismo na classe Simulador, promovendo uma enorme simplificação do código e ainda, mais elegância.

4) Sobrecarga de operadores: este é outro forte mecanismo da programação orientada a objeto, foi utilizado para fornecer clareza ao código e simplificar algumas operações. A classe Relogio possui a sobrecarga do operador ++, quando utilizado, incrementa um no tempo da simulação. Já na classe Evento, devido ao programa utilizar uma priority_queue, foi necessário implementar a sobrecarga de um operador de comparação, nesse caso, o operador <.

5) Padrão de projeto: como dito anteriormente, foi aplicado o padrão de projeto Fachada na classe Simulação. Também vale a pena ressaltar o poder dos padrões de projetos principalmente, em programas maiores, aonde a numerosa quantidade de classes começam a exigir uma maior organização no código.

III. DISCUSSÃO

O desenvolvimento do programa iniciou-se com a implementação das classes fundamentais. Posteriormente, foi desenvolvido uma lógica para a geração contínua dos eventos, assim como a sua execução. Efetivamente, o maior desafio do programa foi a conexão das classes para realizar a simulação. Isso porque o problema, apesar de não parecer tão complexo em uma primeira abordagem, envolve detalhes que acabam elevando o nível de dificuldade. Dessa forma, dois detalhes deixaram a desejar na interface gráfica: os carros não fazem a curva na hora de trocar de pista, apenas saltam de uma para a outra e também ficam sobrepostos nas vias. Nada obstante, presume-se que o resultado da interface foi satisfatório. Em geral, aplicando as técnicas da orientação a objeto, ficou evidente como a linguagem C++ permite implementar códigos mais robustos e engenhosos do que, por exemplo, a linguagem C. Como o autor também implementou a simulação nessa outra linguagem (exceto a parte relacionada a interface gráfica), notou-se alguns pontos que merecem destaque. Primeiramente, a incrível redução e simplificação de código que ocorre com o uso das classes, tendo em vista que estas fornecem construtores e destrutores que, quando comparados as funções de criação e destruição dos TDA's, são mais simples e elegantes. Além disso, na linguagem C temos procedimentos (ou funções) que são aplicados globalmente em nossa aplicação. No caso da orientação a objetos, temos métodos que são aplicados aos dados de cada objeto, fazendo com que a programação orientada a objetos se torne mais difundida. Essa difusão se dá muito pela questão da reutilização de código e pela capacidade de representação do sistema muito mais perto do que veríamos no mundo real. Em síntese, aplicando os recursos que a POO oferece, desenvolveu-se classes com comportamentos e estados bem definidos, o código tornou-se mais simples, organizado, compacto e sofisticado.

IV. CONCLUSÃO

O projeto apresentado permitiu o desenvolvimento e a familiarização com os recursos da programação orientada a objetos e assim, aprimorou-se as habilidades para o desenvolvimento de softwares. Apesar de ser uma aplicação relativamente "simples"comparada as diversas simulações, o projeto possibilitou a

compreensão dos conceitos básicos, visando o desenvolvimento de um software para ajudar a melhor planejar, projetar e operar sistemas de transporte. A utilização do Qt Creator como ambiente de desenvolvimento integrado também foi de grande importância, além de possuir uma interface agradável e de fácil utilização, fornece recursos para o melhor desenvolvimento dos softwares. Além disso, vale ressaltar que suas ferramentas de depuração agilizaram a resolução de eventuais problemas encontrados pelo caminho. Finalmente, tanto na prática quanto na teoria, a conclusão do projeto foi enriquecedora, incorporando ao autor técnicas e competência para a elaboração de programas cada vez mais sofisticados.