Engenharia de Software aplicada ao desenvolvimento de controladores autônomos no simulador TORCS.

Relatório de trabalho do Programa Jovens Talentos para a Ciência

Aluno: Edgar Fabiano de Souza Filho

Professor Orientador: Guilherme Novaes Ramos

Universidade de Brasília

Introdução

Corridas de carros em competições simuladas estão se tornando mais populares no campo da inteligência computacional uma vez que proporcionam uma excelente estrutura para testar características de algoritmos sob investigação como a aprendizagem, adaptabilidade, evolução e raciocínio.

Em um trabalho anterior [1] relacionado ao tema, foi desenvolvido por pesquisadores da Universidade de Brasília um controlador baseado em uma máquina de estados finitos (*Finite State Machine - FSM*), o *FSMDriver3*. Uma máquina de estados finitos é um modelo matemático usado para representar programas de computadores ou circuitos lógicos, é concebido como uma máquina abstrata em que seu estado atual e o próximo estado devem pertencer ao seu conjunto finito de estados.

O [1]*FSMDriver3* é um programa independente, escrito em C++ uma linguagem orientada a objetos. O paradigma orientado a objetos disponibiliza alguns recursos que serão utilizados nesse trabalho. Cada estado é representado como uma subclasse, que herda de uma superclasse *Estado*, ou seja, adquire as suas características e métodos por definição. Um método *drive* é responsável por calcular os valores dos atuadores e retornar os mesmos para o servidor como um objeto *CarControl*, definindo as políticas de condução do controlador. Cada estado tem que implementar um método *drive* diferente de acordo com as suas particularidades de cada estado. Isso funciona, porém se o desenvolvedor deseja realizar uma mudança no cálculo de apenas um atuador, é necessário que ele estenda a classe correspondente ao estado, e altere esse método *drive*.

A ideia por trás desse controlador é observar o comportamento de um piloto, avaliar quais são as situações que ele deve lidar durante a corrida, e separar a condução do veículo em estados distintos, no caso, o controlador que é formado por 3:

- Dentro da pista;
- Fora da pista;
- Preso na borda.

[1] https://github.com/bruno147/fsmdriver

Cada estado representa um tipo de comportamento que o controlador deve ter durante a corrida. Uma função de transição seria responsável por avaliar o ambiente, e baseada nessas informações realizar a troca adequada do estado atual para o seguinte, em que qualquer estado pode levar a qualquer outro, como no diagrama:

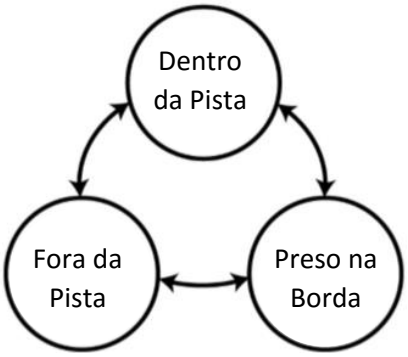

Figura 1: Diagrama de transição de estados (adaptado de [1]*)*

O processo de desenvolvimento de algoritmos de condução de veículos é complicado, pois o teste é uma necessidade frequente e sistemas reais de condução são caros. Para isso, o uso de simuladores realistas de corrida de carros é uma alternativa viável, pois barateiam esse processo de experimentação. Há estudos como o visto em [2] que são esforços para auxiliar o processo de experimentação, em que uma interface de treinamento que agiliza a execução de testes no TORCS. Então todo esforço no sentido de facilitar o processo de desenvolvimento de controladores é válido.

O simulador/jogo TORCS (*The Open Racing Car Simulator*) é uma plataforma que é conhecida pelo seu mecanismo de modelagem física muito robusto e uma interface de fácil utilização, com ambiente muito personalizável para simulações de corrida de carros, tem sido amplamente utilizado em pesquisas de inteligência computacional para o desenvolvimento e comparação soluções. Ele considera fatores tais como colisão, tração, aerodinâmica e o consumo de combustível além de fornecer vários circuitos, veículos e controladores, permitindo simular vários tipos de situações no jogo. Ele foi utilizado para realizar os experimentos por causa dessas características.

O TORCS, é um aplicativo independente em que os controladores são programas escritos em C++ ou Java, compilados como módulos separados, que são carregados na memória principal quando a corrida inicia. A comunicação entre o Simulador e os controladores é feita por meio de um protocolo UDP, em que há uma conexão de servidor/cliente, o [1]SCR cuida desse processo de integração entre os 2 sistemas. O TORCS fornece as informações dos sensores para o controlador, que as processa e retorna uma ação do veículo para o servidor.

O controlador pode usar comandos básicos, disponibilizados pelo SCR para controlar o carro, que são chamados de atuadores, no caso do controlador de máquina de estados finitos com 3 estados (*FSMDriver3*) são utilizados:

- Aceleração;
- Freio;
- Volante;
- Marcha;
- Embreagem.

[1] https://arxiv.org/pdf/1304.1672.pdf

Como pode ser visto no trabalho [3], uma arquitetura modular paramétrica foi desenvolvida em uma situação semelhante para um controlador, que tinha algumas características que permitiam tratar de forma independente cada atuador. Em termos de engenharia de software, um programa modular é mais legível e possui uma manutenção menos custosa, pois ao invés de se ter um único programa cuidando de várias tarefas, é mais viável dividi-lo em vários subprogramas, cada um com uma tarefa específica. Se o mesmo apresentar um erro, basta buscar o módulo originador da sua falha e corrigi-lo.

Por essa motivação, esse trabalho também traz uma proposta de modularizar um controlador, não somente para facilitar a correção de erros, mas para agilizar o processo de modificação do mesmo. Cada um dos 5 atuadores ganhará uma abstração no código do *FSMDriver3* como um módulo, e cada modulo funcionará de forma independente dos outros, sendo responsável somente por retornar os valores para o método *drive*.

Metodologia

É proposto então o [1]*FSMDriver3** (o *FSMDriver3* modular) que divide a forma de como são calculados os atuadores em módulos independentes, ao invés de termos um método *drive*, que lida com o cálculo de todos os atuadores dentro de si, agora tem-se 5 módulos, cada um é um método responsável por realizar o cálculo de cada atuador, e compor o método *drive*, para retornar o *CarControl* (um objeto no programa que cuida dos atuadores) ao servidor com os valores adequados ao estado.

Em sua classe abstrata *Estado*, da qual derivam os estados, foram criados 5 métodos virtuais correspondentes à cada atuador, ou seja, cada um desses métodos deveria ser implementado em cada estado, já que todos herdam da superclasse *Estado*, assumindo seu comportamento em cada situação, dessa forma, cada um dos 3 estados teria obrigatoriamente uma forma independente de calcular os atuadores.

Como pode ser visto no diagrama abaixo, a representação de cada estado (à esquerda) que herda da superclasse abstrata *Estado* (à direita), implementa os 5 módulos da forma que é adequada para cada estado, no caso a situação em que se encontra o carro na corrida.

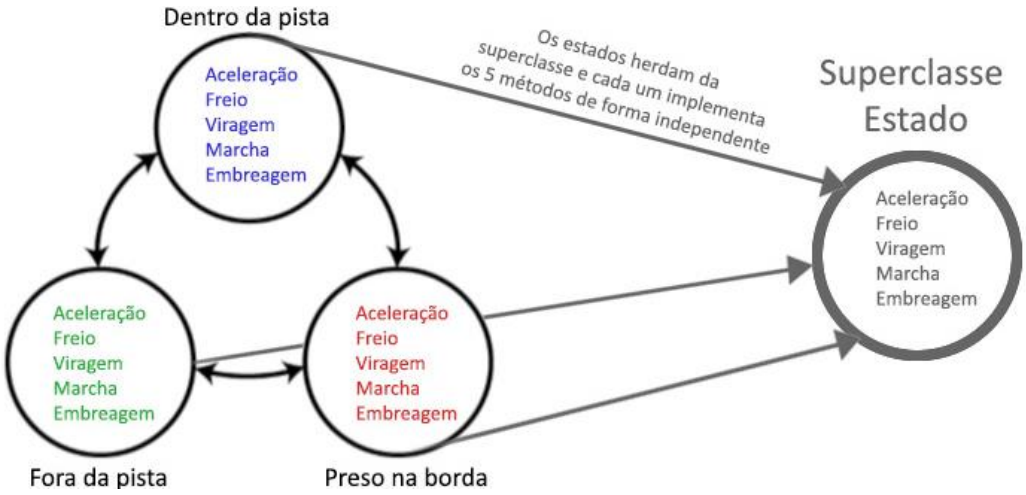

Figura 2: Abstração do esquema de herança entre os estados e a sua superclasse

[1] https://github.com/EdgarFabiano/FSMDriver-Modular

Essa modularização faz com que seja mais fácil implementar modificações de comportamento (dos atuadores) em novos controladores, uma vez que basta apenas fazer com que o novo estado com a modificação herde do estado correspondente no *FSMDriver3** e reimplemente o método desejado, uma vez feito isso, é obrigatória a implementação dos métodos de cálculo dos atuadores, o que pode ser feito em poucas linhas de código no novo controlador.

Inicialmente no *FSMDriver3*, quando era necessário gerar um novo controlador com apenas uma modificação no comportamento de um atuador, era necessário alterar o método *drive* que era responsável por todos os atuadores no estado desejado, na parte responsável pelo atuador desejado. Um esforço de repetir o código do estado com as modificações para o novo controlador era necessário sempre que fosse fazer um novo controlador derivado do *FSMDriver3*.

Experimentos

Para que a proposta seja validada, é necessário que o *FSMDriver3** apresente o mesmo comportamento do *FSMDriver3* não havendo nenhuma alteração nos módulos, ou seja a modularização não pode produzir um controlador diferente, apenas uma arquitetura diferente, mas com o mesmo comportamento. Baseado nos experimentos do trabalho em [1], foi montada uma bateria de testes afim de verificar a semelhança dos comportamentos de cada controlador, os fatores a serem observados eram: o tempo absoluto que eles levam para completar um número fixo de voltas em cada pista, e o comportamento observado de cada um deles. As pistas escolhidas tiveram a mesma motivação, sendo as mesmas dos experimentos de [1].

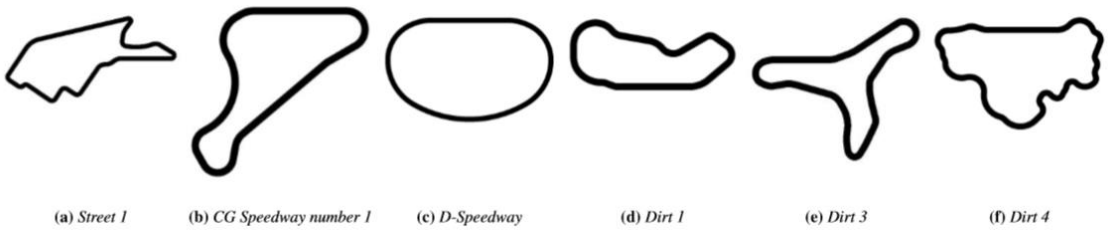

Figura 3: Pistas que foram usadas nos experimentos de [1] e nos deste trabalho

Em cada pista, cada um dos controladores *FSMDriver3** e *FSMDriver3* corria por 1, 2, 3, 5 e 10 voltas para fins estatísticos. Os tempos que levaram para completar 10 voltas em cada uma das 6 pistas foram tomados, e mostrados na tabela abaixo:

	10 voltas					
	Street I	CG Speedway I	D-Speedway	Dirt I	Dirt 3	Dirt 4
FSM3	**11:10:09 (6) †**	**09:04:01**	**07:41:25**	**05:06:56 (6) †**	**17:25:15 (9) †**	**02:23:29 (1) †**
FSM3*	**11:13:47 (6) †**	**08:55:14**	**07:41:25**	**05:26:77 (6) †**	**17:25:35**	**05:11:38 (2) †**

Tempo para completar 10 voltas nas pistas

Em que † representa a situação em que o controlador não conseguiu completar a prova, e o número entre parênteses mostra a quantidades de voltas completas que executou antes de sair da corrida. Os tempos destacados em verde são os que coincidiram do FSM3 (*FSMDriver3*) com FSM3* (*FSMDriver3**) modular e os em vermelho mostram os que não coincidiram.

Fica claro nesses experimentos que o comportamento dos controladores não foi o esperado na maioria dos casos. Então um processo de investigação foi iniciado para tentar descobrir a causa dessas variações. Primeiramente foram observados os comportamentos do *FSM3* e do *FSM3** em segmentos iguais da pista, para observar as principais diferenças. Após isso, as hipóteses foram implementadas com alterações no código do controlador modular, que poderiam levar à solução, então mais testes e simulações eram feitos, se um falhasse, era necessário reiniciar o processo.

Após diversas tentativas, descobriu-se que a causa da divergência de comportamentos entre os controladores *FSM3* e *FSM3** se davam por causa dos métodos *drive* que retornavam um *CarControl* em cada estado, após as alterações para o *FSM3** eles não estavam devidamente estruturados como no *FSM3*, essa investigação também possibilitou a conclusão de que a ordem que os 5 métodos da modularização são chamados dentro de cada método *drive* influencia o resultado final.

A seguir a tabela de resultados dos mesmos experimentos realizados anteriormente, porém com as diferenças implementadas:

	10 voltas					
	Street I	CG Speedway I	D-Speedway	Dirt I	Dirt 3	Dirt 4
FSM3	**11:10:09 (6) †**	**09:04:01**	**07:41:25**	**05:06:56 (6) †**	**17:25:15 (9) †**	**02:23:29 (1) †**
FSM3*	**11:10:09 (6) †**	**09:04:01**	**07:41:25**	**05:06:56 (6) †**	**17:25:15 (9) †**	**02:23:29 (1) †**

Tempo para completar 10 voltas nas pistas após as modificações

Como foi visto então, agora tem-se um controlador modular *FSMDriver3** que tem o mesmo comportamento do da sua versão não modular *FSMDriver3*. Para verificar que realmente implementar uma mudança de comportamento, foi desenvolvido o controlador *FSMDriver3A*, que era derivado do *FSMDriver3** porém com uma alteração apenas no módulo de controle da aceleração no estado "Dentro da pista".

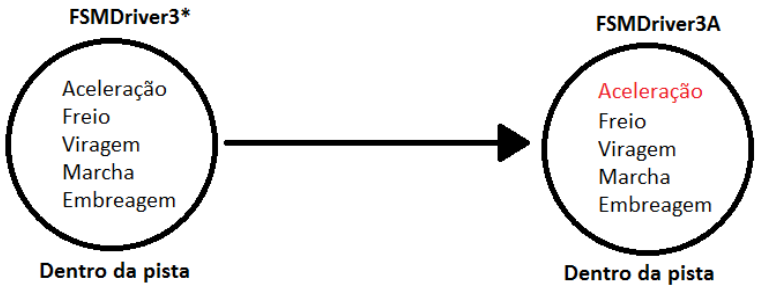

Figura 4: Esquemático do estado "Dentro da pista" do FSMDriver 3 e do FSMDriver3A implementando a alteração somente no módulo de aceleração.*

Com isso é possível desenvolver um controlador que implemente somente a alteração desejada com poucas linhas de código. No exemplo do *FSMDriver3A* foi necessário apenas estender a classe do estado "Dentro da pista" e implementar a alteração na aceleração, ao invés de repetir o código desse mesmo estado no novo controlador e alterar o método *drive* na parte de aceleração. Reduzindo o esforço de repetir algo em torno de 140 linhas de código dependendo do estado para pouco mais de 20 linhas dependendo da implementação.

A seguir os resultados dos mesmos experimentos com o novo controlador *FSMDriver3A* em comparação com o *FSMDriver3**:

	10 voltas					
	Street I	CG Speedway I	D-Speedway	Dirt I	Dirt 3	Dirt 4
FSM3*	**11:10:09 (6) †**	**09:04:01**	**07:41:25**	**05:06:56 (6) †**	**17:25:15 (9) †**	**02:23:29 (1) †**
FSM3A	**17:54:17**	**08:56:46**	**07:25:24**	**06:25:75**	**13:29:31**	**16:40:06**

Tempo para completar 10 voltas nas pistas após a implementação diferente da aceleração.

A única diferença do *FSMDriver3A* para o *FSMDriver3** é a forma com que cada um implementa o cálculo da aceleração no módulo da aceleração do estado "Dentro da pista".

O *FSMDriver3A* foi projetado para que a aceleração fosse proporcional às leituras dos sensores frontais de pista disponíveis no controlador, ou seja, o *FSMDriver3A* acelera mais se estiver longe de uma borda da pista (em uma pista aparentemente reta), e acelera menos se estiver próximo de uma borda (se aproximando ou dentro de uma curva), e também lida com casos excepcionais para acelerar no máximo quando estiver em uma pista reta, e evitar que o mesmo fique preso de frente a uma borda da pista.

Enquanto que a aceleração do *FSMDriver3** é calculada assim: se a velocidade do controlador está abaixo da velocidade desejada para aquele trecho, acelera tudo, se não, não acelera nada.

Com pouco menos do que 30 linhas de código foi possível implementar essas alterações, e os resultados a partir dos experimentos mostraram que o *FSMDriver3A* teve um comportamento diferente do *FSMDriver3**, que por acaso obteve um desempenho melhor nos testes aplicados, conseguindo completar todas as voltas em todas as pistas em menos tempo do que o *FSMDriver3**.

Conclusão

Com esse trabalho foi possível concluir que a modularização de software permitiu implementar mudanças de comportamento nos controladores de uma forma menos custosa do que de um não modular. Além de deixar o código mais legível e fácil de entender.

Bibliografia

[1] G. F. P. Araujo, G. S. Silva, M. C. Crestani, Y. B. Galli, and G. N. Ramos, "Evolving Finite-State Machines Controllers for the Simulated Car Racing Championship," 2014.

[2] G. Caldeira, Clara; Aranha, Claus; Ramos, "TORCS Training Interface : An auxiliary API for developing TORCS drivers," *XII Simpósio Bras. Jogos e Entretenimento Digit.*, vol. 13, pp. 16–19, 2013.

[3] E. Onieva, D. a. Pelta, J. Alonso, V. Milanés, and J. Pérez, "A modular parametric architecture for the TORCS racing engine," *CIG2009 - 2009 IEEE Symp. Comput. Intell. Games*, pp. 256–262, 2009.