Inteligência Artificial aplicada ao desenvolvimento de controladores autônomos em simuladores de corrida.

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.

O processo de desenvolvimento e de testes de controladores pode se tornar algo extremamente complexo e trabalhoso dependendo do que se espera do mesmo, por isso, qualquer melhoria nesse processo pode facilitar bastante o trabalho dos desenvolvedores.

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.

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.

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:

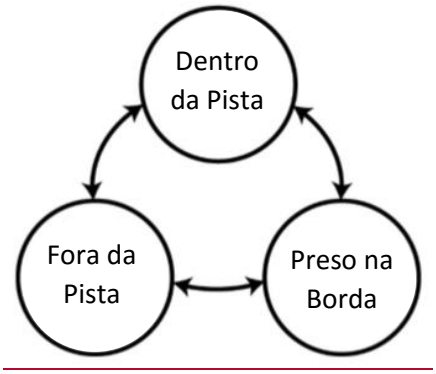
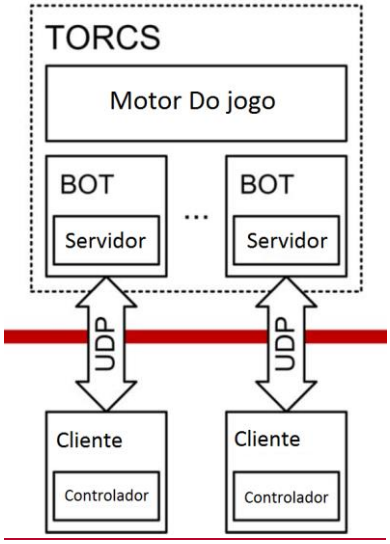


Diagrama de transição de estados

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 totalmente válida pois um algoritmo pode ser implementado em sistemas diferentes, trabalhos como [2] auxiliam nesse processo de experimentação.

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 TORCS fornece as informações dos sensores para o controlador, que as processa e retorna uma ação do veículo para o servidor.



Esquemático do protocolo de comunicação do servidor do TORCS com os clientes

O controlador pode usar comandos básicos, disponibilizados pelo TORCS 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.

O objetivo deste trabalho é contribuir com o trabalho visto em [1], propondo uma nova arquitetura de software, modularizando o cálculo dos atuadores, afim de facilitar o processo de modificação de comportamento do controlador. Como pode ser visto em [3] uma arquitetura modular paramétrica foi desenvolvida em uma situação semelhante para um controlador, e tinha algumas características que permitiam tratar de forma independente cada atuador.

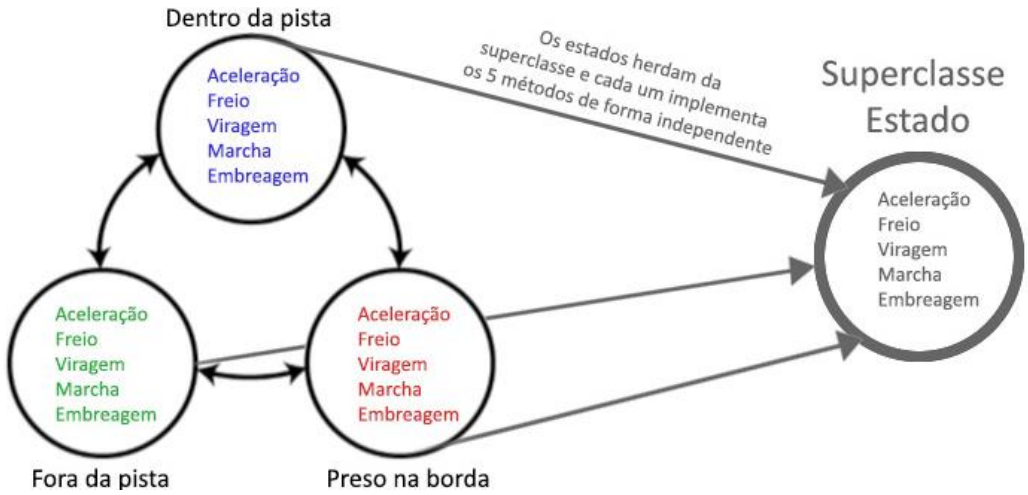
O *FSMDriver3* é um programa independente, escrito em C++ uma linguagem orientada a objetos. O paradigma orientado a objetos disponibiliza alguns recursos que serão utilizados nesse trabalho. Cada estado é representado como uma subclasse, que herda de uma superclasse *Estado*, ou seja, adquire as suas 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 modifique esse método *drive*.

Metodologia

É proposto o *FSMDriver3** (ou *FSMDriver3* modular) que divide a forma de como são calculados os atuadores em módulos independentes, ao invés de termos um método *drive*, agora tem 5 módulos, cada um é um método responsável por realizar o cálculo de cada atuador, e compor o método *drive*, para retornar o *CarControl* com os valores adaptados 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 de uma forma diferente, 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 necessária para cada estado, no caso a situação em que se encontra o carro na corrida.



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

Essa modificação faz com que seja mais fácil realizar modificações de comportamento nos controladores existentes, ou em novos, uma vez que basta apenas fazer com que os estados do controlador herdem da superclasse *Estado*, uma vez feito isso, é obrigatória a implementação dos métodos de cálculo dos atuadores, o que pode ser feito em poucas linhas de código para um novo controlador.

Experimentos

Como o objetivo deste trabalho é apenas contribuir, o comportamento esperado desse controlador modular *FSMDriver3** deve ser o mesmo do original *FSMDriver3*. Baseado nos experimentos de [1], foi montada uma bateria de testes afim de verificar a semelhança dos comportamentos de cada controlador. As pistas escolhidas tiveram a mesma motivação, sendo as mesmas dos experimentos de [1].

(a) *Street 1* (b) *CG Speedway number 1* (c) *D-Speedway* (d) *Dirt 1* (e) *Dirt 3* (f) *Dirt 4*

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 laps					
	Street I	CG Speedway number 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 com FSM3* modular e os em vermelho mostram os que não coincidiram.

Fica claro nesses experimentos que o comportamento dos controladores divergiu 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 eram 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 laps					
	Street I	CG Speedway number 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

Conclusão

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.