Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

TORCS Training Interface: Uma ferramenta auxiliar ao desenvolvimento de pilotos do TORCS

Clara Marques Caldeira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Guilherme Novaes Ramos

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenadora: Prof.ª Dr.ª Maristela Holanda

Banca examinadora composta por:

Prof. Dr. Guilherme Novaes Ramos (Orientador) — CIC/UnB
Prof.ª Dr.ª Carla Denise Castanho — CIC/UnB
Prof. Dr. Rodrigo Bonifácio de Almeida — CIC/UnB

CIP — Catalogação Internacional na Publicação

Caldeira, Clara Marques.

TORCS Training Interface: Uma ferramenta auxiliar ao desenvolvimento de pilotos do TORCS / Clara Marques Caldeira. Brasília : UnB, 2013.
115 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. TORCS, 2. inteligência artificial, 3. jogos digitais

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

TORCS Training Interface: Uma ferramenta auxiliar ao desenvolvimento de pilotos do TORCS

Clara Marques Caldeira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Guilherme Novaes Ramos (Orientador)
CIC/UnB

Prof.ª Dr.ª Carla Denise Castanho
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
CIC/UnB

Prof.ª Dr.ª Maristela Holanda
Coordenadora do Bacharelado em Ciência da Computação

Brasília, 13 de dezembro de 2013

Dedicatória

Dedico à minha família e amigos, tão frequentemente negligenciados em favor dos meus estudos, mas que ainda assim mantiveram o tão necessário e motivador apoio ao longo dos últimos anos.

Agradecimentos

A todos os grandes professores e professoras que contribuíram para a minha formação e para o meu crescimento intelectual e profissional, a minha grande admiração e um muito obrigada. Agradeço especialmente o professor Guilherme Ramos, pela orientação e incentivo para ir além do esperado ou estritamente necessário, ao professor Claus Aranha e a Bernhard Wymann, pelas valiosas contribuições a este trabalho.

Resumo

A ineficiente maneira como pilotos são testados e desenvolvidos para jogo e simulador de corrida TORCS é um problema relevante por conta das limitações impostas sobre trabalhos de desenvolvimento de pilotos, i.e., algoritmos que determinam o comportamento dos carros não controlados por jogadores humanos. Porque este *software* tem um papel de plataforma para *benchmark* de diferentes abordagens de Inteligência Artificial, é importante que se procure mitigar tal problema.

Aqui desenvolveu-se a *TORCS Training Interface*, uma ferramenta que oferece automatizações para melhorar a eficiência das chamadas de simulações e retornar dados mais completos - ambos fatores importantes para as necessárias avaliações que têm como objetivo estimar habilidades de pilotos. Os resultados dos testes comparativos realizados indicam que o uso da ferramenta é uma alternativa viável às abordagens observadas na literatura, apresentando vantagens que podem torná-la a forma mais adequada para processos similares aos considerados neste trabalho.

Palavras-chave: TORCS, inteligência artificial, jogos digitais

Abstract

The inefficient manner in which drivers are tested and developed for the racing game and simulator TORCS is a relevant problem because of the limitations imposed over projects of development of drivers, i.e., algorithms that determine the behavior of cars that are not controlled by human players. Because this software has a role of benchmark for different techniques of Artificial Intelligence, it is important to work on mitigating this problem.

The TORCS Training Interface was developed, a tool that offers automatizations in order to improve the efficiency of simulation calls and return more complete data - both of which are important for the necessary evaluations that have as a goal estimating the fitness of drivers. Results of the comparative tests performed indicate that the use of the tool is a viable alternative to the approaches seen in the literature, presenting advantages that can make it the most fitting to processes that are similar to the ones considered here.

Keywords: TORCS, artificial intelligence, digital games

Sumário

Lista de Figuras

Capítulo 1

Introdução

Ao passo que são desenvolvidas novas tecnologias na área de Inteligência Artificial (IA), tornam-se necessárias formas de se estimar os avanços feitos, observar como estes se comparam com o que existia até então e que novas possibilidades trazem. Para tanto, pode-se utilizar como base de comparação, ou meta a ser alcançada, o desempenho observado em seres humanos, tomados como referência no quesito de inteligência. O jogo de xadrez é frequentemente usado como ferramenta de *benchmark* para IAs. A habilidade de jogá-lo bem é ligada à inteligência em humanos [8] e, nesse caso, uma análise dos resultados de partidas nas quais a máquina joga contra pessoas com diferentes níveis de habilidade proporciona informações a respeito da habilidade da própria IA. Para que seja possível realizar efetivamente tais comparações com intuitos avaliativos, é essencial que exista um ambiente no qual diferentes IAs e humanos possam atuar em condições semelhantes.

Jogos e sistemas de simulação têm o potencial de oferecer uma plataforma adequada para esses processos. Embora suas regras apresentem grandes variações, eles têm comum a necessidade de frequentes tomadas de decisões que afetam o resultado final e quão bom o agente inteligente (algoritmo participante do jogo no lugar de um indivíduo) ou jogador é considerado. Nesse sentido, uma IA procuraria maximizar a pontuação obtida, ou vencer a disputa, dependendo de cada situação específica, considerando-se que o seu objetivo é obter o melhor resultado possível.

A visão de jogos como ferramentas para experimentação e estudo de Inteligência Artificial não é recente. Uma parte significativa dos trabalhos acadêmicos na área têm foco em jogos de tabuleiro tradicionais, como damas e xadrez, nos quais o desafio é superar os melhores jogadores humanos. Embora esse enfoque ainda seja base de muitas pesquisas, existe uma área crescente referente a aplicações de IA em jogos digitais diversos, nos quais muitas técnicas preexistentes são passíveis de serem utilizadas, desenvolvidas e testadas, independentemente do gênero [15].

Algoritmos evolucionários, baseados no processo de aprendizado coletivo em populações, exibem resultados robustos em processos que procuram otimizar parâmetros [1]. Cada indivíduo representa um ponto no espaço de soluções em potencial para um dado problema e a população é normalmente inicializada arbitrariamente, evoluindo na direção de regiões melhores do espaço de busca através de seleções, mutações e recombinações [1]. Avaliações da qualidade dos indivíduos são realizadas a cada etapa, o que pode limitar o processo se essas forem caras, como é o caso do jogo usado neste trabalho. A ferramenta

proposta procura reduzir os custos de avaliação de pilotos, beneficiando o processo de desenvolvimento dos seus algoritmos como um todo.

1.1 Inteligência Artificial em jogos

Jogos que incluem agentes ou personagens não controlados por um jogador (NPCs) podem fazer uso de métodos de Inteligência Artificial com o intuito de controlá-los, de forma que jogadores humanos possam ter a máquina como oponente hábil e desafiador, caso esses agentes tenham esse papel, e possam perceber o ambiente virtual de forma mais convincente e interessante. Outra aplicação relevante de técnicas de IA é a geração de conteúdo diversificado direcionado a jogos digitais. A habilidade de gerar novas fases automaticamente torna o processo de criação mais barato e proporciona a possibilidade de oferecer ao jogador maior quantidade e variedade sem sacrificar a qualidade, como aconteceria se utilizada uma abordagem pseudo-aleatória simples. De fato, técnicas de Inteligência Artificial podem ser aplicadas à maioria dos aspectos de desenvolvimento e *design* de jogos, incluindo também a animação procedural, iluminação adaptativa e controle inteligente de câmera [15, 16].

Jogos de tabuleiro tradicionais são definidos por suas regras, que determinam as ações permitidas, pontuações e resultados finais em termos de ganhadores e perdedores. As combinações de possíveis jogadas rapidamente produzem árvores de jogo extensas contendo todas as possíveis ações em cada passo, até o nível onde não há mais ações permitidas (i.e., fim de jogo), ponto no qual se pode identificar aquelas decisões que levam aos melhores resultados esperados, permitindo escolhas com informação. Implementações bem sucedidas como *Deep Blue* para o xadrez e *Chinook* para damas gastam seus esforços com essas buscas em árvores [16], e por esta razão são referenciadas como abordagens de força bruta.

Entretanto, estratégias como esta têm aplicabilidade extremamente limitada quando o jogo em questão tem como característica um largo fator de ramificação, i.e., muitas ações possíveis em cada situação, fazendo com que a árvore, que cresce em ordem exponencial, torne-se rapidamente impraticável de se percorrer. Essas técnicas não funcionam bem para jogos como *Go*, onde a grande quantidade de possíveis ações produz espaços de busca além do que se consegue percorrer atualmente, e são ainda mais inaplicáveis em jogos digitais dotados de características contínuas [16], cujos conjuntos de ações podem ter dimensões virtualmente infinitas.

A crescente complexidade dos mundos virtuais de jogos digitais traz problemas semelhantes, são necessárias abordagens alternativas para tomar decisões em ambientes cujas possíveis configurações são inúmeras. Torna-se cada vez mais difícil desenvolver personagens não jogáveis cujos comportamentos sejam suficientemente interessantes e críveis a ponto de acompanhar os avanços da computação gráfica. O chamado efeito Vale da Estranheza refere-se ao fato de que, ao passo que imagens se tornam mais realistas, os jogadores se interessam menos pelos personagens, a não ser que eles apresentem um comportamento apropriado complementar à boa aparência [16].

Técnicas atuais de controle envolvem principalmente máquinas de estados finitos, uso de *scripts* e buscas [15]. A maior parte do comportamento é programado manualmente, exigindo muito esforço humano direcionado a desenvolvimento e testes [15]. As técnicas utilizadas muitas vezes não se equiparam ao estado da arte em aprendizagem computa-

cional, mas argumenta-se que o uso de tais tecnologias poderia enriquecer jogos, adicionando à experiência de um jogador se aplicadas à geração automatizada de conteúdo ou ao comportamento de personagens não jogáveis [16].

NPCs podem se relacionar com humanos de formas diversas, como guiá-los em relação ao funcionamento do jogo, cooperar com eles, serem seus adversários ou até mesmo estarem presentes apenas para proporcionar contexto, de forma a enriquecer a narrativa. A aplicação de técnicas de Inteligência Artificial proporciona a possibilidade de comportamentos dinâmicos e adaptados à habilidade do jogador, além da criação dinâmica de conteúdo, por exemplo cenários ou fases. Adicionalmente, personagens que aprendem e evoluem não têm um comportamento tão previsível quanto o daqueles cujas decisões são determinísticas, fato do qual pode se aproveitar um indivíduo que esteja familiarizado com elas, de forma a tornar o jogo mais interessante e desafiante. Especula-se que a próxima geração de IA aplicada a jogos trará personagens inteligentes e empáticos, que irão conhecer o jogador e trabalhar para otimizar sua experiência em tempo real [16].

Observa-se, assim, que o emprego de Inteligência Artificial em jogos digitais é possível de diferentes formas, podendo oferecer vantagens no processo de desenvolvimento e enriquecer o produto final.

1.2 Jogos em Inteligência Artificial

Similarmente, a pesquisa em Inteligência Artificial pode se beneficiar bastante dos ambientes complexos desenvolvidos para jogos digitais. Esses podem ser utilizados para aplicar e estender algoritmos, além de potencialmente apresentar desafios que incentivam a criação de novas abordagens ou adaptação de técnicas existentes para outras aplicações, consequentemente incentivando o avanço do estado da arte na área.

Os jogos que oferecem simulações ainda podem ser usados em trabalhos direcionados à aplicação de IA em situações reais. Simuladores de corrida, por exemplo, oferecem a possibilidade de testar algoritmos desenvolvidos para o controle de veículos. Esta é uma área de pesquisa extremamente relevante atualmente por seu potencial de amenizar problemas crescentes do transporte urbano, como engarrafamentos e colisões [6, 20]. Os testes baratos e rápidos que podem ser realizados em simuladores, embora não substituam por completo os testes conduzidos com os próprios robôs ou máquinas, são de grande utilidade para o estudo e avanço de projetos.

Ao mesmo tempo, jogos de corrida, que fornecem ambientes envolvendo múltiplas variáveis interrelacionadas, representam uma plataforma adequada para a comparação de diferentes abordagens para problemas complexos, como provado pelas competições organizadas acerca desse tópico nos últimos anos [11]. Esse gênero encontra-se entre os mais populares por poder prover experiências realísticas de direção em cenários vívidos, além da disponibilidade de periféricos que adicionam à experiência do *gameplay*. Estes também estão sob um constante processo de evolução, ficando cada vez mais caros de se desenvolver. De fato, os NPCs deste gênero têm de lidar com motores de simulação física incrementalmente realistas e com uma larga variedade de situações complexas.

1.3 *Simulated Car Racing Championship*

The Open Racing Car Simulator (TORCS) é um simulador de corrida bastante utilizado atualmente em pesquisas na área de Inteligência Artificial [11]. Ser multi-plataforma e oferecer um motor sofisticado e a possibilidade de alterações variadas são fatores que abrem a oportunidade para estudos variados, como mostra a literatura.

Utiliza-se o TORCS como ferramenta para a realização das corridas e qualificações em competições entre IAs [13]. Este simulador é o estado da arte e proporciona um ambiente altamente customizável, gráficos 3D sofisticados, uma plataforma poderosa de simulações físicas que leva em consideração aspectos importantes como colisões, tração, aerodinâmica e consumo de combustível, além de fornecer uma variedade de pistas, veículos e pilotos que proporcionam um extenso conjunto de situações possíveis dentro do jogo [11, 14].

Desde 2007 acontece anualmente o *Simulated Car Racing Championship* (SCR), campeonato de simulação de corrida automobilística entre pilotos de carros submetidos por pesquisadores de todo o mundo [11]. A partir de 2009 este campeonato foi divido em três etapas, cada uma ocorrendo em uma conferência internacional (*IEEE Congress on Evolutionary Computation*, *ACM Genetic and Evolutionary Computation Conference* e *IEEE Symposium on Computational Inteligence and Games*, todas do mesmo ano). Tratam-se de três "Grandes Prêmios", divididos em duas etapas em uma mesma pista previamente desconhecida pelos competidores: qualificação seguida de corrida de 5 voltas com os oito melhores qualificados. Ao final de cada evento, os competidores são pontuados de acordo com as regras da Fórmula 1. O ganhador do campeonato é o time que, após a última etapa, totalizar o maior número de pontos [14].

Entre os pilotos submetidos existem aplicações neuroevolução, lógica difusa, campos potenciais e algoritmos genéticos, além de aplicações de redes neurais que utilizam diferentes abordagens para este problema, sejam treinadas para fazer o menor tempo em uma pista específica, para ter bom desempenho em diferentes situações ou mesmo para exibir um comportamento similar ao de um ser humano [11].

Uma competição como essa proporciona à comunidade científica uma medida padrão dos resultados de diferentes algoritmos de aprendizagem computacional em uma situação complexa, com múltiplas variáveis contínuas, e de fácil visualização [14], além de oferecer um bom incentivo a trabalhos nesse campo, proveniente não só do gosto por competições mas pelo automobilismo em geral, por jogos digitais e simulações. Dentre tais trabalhos podem estar aqueles direcionados à pesquisa em autonomia veicular, tema extremamente relevante na atualidade que faz uso de princípios similares aos daqueles presentes no desenvolvimento dos pilotos em jogos de corrida. De fato, os formatos de dados fornecidos pelo TORCS aos pilotos participantes do SCR assemelham-se às abstrações dos dados coletados pelos sensores do robô *Stanley* [22], vencedor do desafio *DARPA*[1] de 2007.

1.4 Definição do Problema

Existem diversas formas de abordar o problema de desenvolvimento de IA, como algoritmos de busca em árvores e de busca local [19], mas um ponto em comum é que todos

[1] Competição entre veículos autônomos realizada nos Estados Unidos (http://archive.darpa.mil/grandchallenge)

precisam de uma forma de avaliar o desempenho dos agentes. No caso do TORCS, essa métrica precisa estar relacionada com os aspectos considerados na estimação da habilidade de pilotos, como o tempo necessário para se percorrer um circuito e a quantidade de colisões que este sofreu ou provocou nesse intervalo. A realização anual do SCR, uma forma de *benchmark* entre abordagens de Inteligência Artificial, incentiva a realização de novos trabalhos voltados ao desenvolvimento de pilotos para o simulador, e todos os anos diferentes grupos de pesquisa aceitam esse desafio.

Considerando que a maior parte dos sensores de um veículo no TORCS pode assumir valores em um intervalo contínuo, atingir uma configuração ótima de parâmetros em um algoritmo de piloto é uma tarefa complexa, que deve envolver uma grande quantidade de avaliações. Quando o objetivo do trabalho é chegar a um piloto que atinge um tempo competitivo em meio a outros carros, que sofre pouco ou nenhum dano e que tem resultados satisfatórios em pistas de diversos tipos, também é importante fazer medições em situações que, em conjunto, levem todos esses critérios em consideração. Além disso, cada corrida admite o número máximo de dez carros, o que exige que qualquer processo com uma população maior realize mais do que uma etapa de avaliação em cada ciclo de execução. Observa-se que tornar avaliações mais ricas normalmente aumenta a quantidade de vezes que uma instância de simulação é executada, como é o caso de usar um conjunto de circuitos com características diferentes para cada avaliação.

Durante o estudo sobre desenvolvimento de pilotos para o simulador, notou-se que a execução de tais simulações repetidamente é ineficiente a ponto de prejudicar o processo de desenvolvimento e, consequentemente, os seus resultados. Além da simulação em si ser bastante demorada, as formas atuais de se realizar testes no jogo são bastante limitadas, o que dificulta o desenvolvimento de pilotos. A repetição de testes exige muitas execuções manuais, que são cansativas, ou o desenvolvimento de *scripts* que forneçam alguma automatização, que é feito redundantemente por não ser disponibilizado para a comunidade. Os resultados obtidos são insuficientes, apenas é fornecido o tempo de um dentre todos os pilotos na corrida, enquanto dados sobre o quanto o carro saiu da pista durante a simulação, que podem ser extraídos, ofereceriam mais informações sobre o comportamento do piloto, enriquecendo a sua avaliação.

Usando a interface de texto fornecida pela versão mais recente do TORCS, observa-se que completar uma volta leva um tempo da ordem de segundos. Embora essa seja uma melhora bastante significativa sobre a simulação em tempo real, que leva um tempo da ordem de minutos, a repetição desse processo continua resultando em etapas demasiadamente demoradas.

Adicionalmente, quaisquer mudanças nas configurações da corrida em termos de quantidade de voltas, pista selecionada, quantidade ou ordenação dos carros precisam ser modificadas manualmente em um arquivo específico em formato `XML` ou na interface gráfica do jogo. Essa restrição faz com que diversificações desejadas em relação à forma como as avaliações são feitas exijam intervenções manuais.

A necessidade de intervenções manuais e a reduzida flexibilidade em termos de testes repetitivos envolvidos no desenvolvimento de pilotos para o TORCS, resultantes da carência em automatizações nas simulações, elevam os esforços envolvidos no desenvolvimento de pilotos inteligentes. Entende-se que prover serviços que resolvam ou ao menos amenizem tais pontos levará a um aumento da proporção de tempo gasto com inovações em comparação com o tempo gasto em contornar dificuldades e limitações, além de incentivar

a realização de mais trabalhos na área. Por isso, propõe-se a *TORCS Training Interface* (TTI), com o propósito de oferecer utilidades a processos de desenvolvimento de pilotos para o simulador TORCS.

No Capítulo 2, discute-se o funcionamento do TORCS e das simulações, além dos fatores importantes para o desenvolvimento de pilotos, relevantes para o projeto da ferramenta proposta, já que seu objetivo é auxilar esse tipo de trabalho. Em seguida, no Capítulo 3, explica-se como se propõe lidar com os problemas mencionados e como a TTI foi projetada e implementada. O Capítulo 4 fornece uma comparação do uso da TTI com outras formas de testes. Por fim, no Capítulo 5, apresenta-se a conclusão e se discute de que formas a solução desenvolvida pode ser aprimorada.

Capítulo 2

Fundamentação teórica

2.1 O simulador TORCS

The Open Racing Car Simulator (Figura 2.1) é um jogo e simulador de corrida de código aberto, desenvolvido na linguagem C++, que está disponível para plataformas Windows, Linux, FreeBSD e MacOS. Ele é usado tanto como jogo de corrida comum quanto como plataforma de pesquisas para simulação da interação e análise do comportamento de diferentes Inteligências Artificiais.

TORCS foi criado por Eric Espié e Christophe Guionneau na década de 1990, mas partes substanciais foram adicionadas por outros contribuintes como Bernhard Wymann e Christos Dimitrakakis[1]. O projeto é atualmente comandado por Wymann e usa uma Licença Pública Geral GNU (GPL) versão 2 (discutida na Seção 2.3). A maior parte de sua arte está registrada sob a Licença da Arte Livre[2].

Há uma variedade de carros, circuitos e oponentes e pode-se jogar com *joystick* ou volante, se o periférico for aceito pela plataforma. Também é possível usar *mouse* e teclado. Os recursos gráficos incluem iluminação, fumaça, marcas de derrapagem e brilho dos discos de freio. A simulação inclui um modelo simples de danos, colisões, propriedades das rodas (e.g. molas, amortecedores, rigidez), aerodinâmica (e.g. efeito solo, aerofólios), e muitos outros. O jogo permite diferentes tipos de corrida, desde simples treino até um campeonato composto de uma série de circuitos. Há também um modo com até quatro jogadores simultâneos [11].

TORCS é um programa de código aberto e permite não só a criação de novos controladores, mas também a fácil extensão daqueles já incluídos na distribuição. No campeonato SCR é utilizada uma arquitetura cliente-servidor (Figura 2.2), onde os controladores são executados em processos separados que se comunicam com o servidor através de uma conexão UDP. A simulação funciona como um relógio, onde a cada iteração (ou tique, que equivale a aproximadamente 20ms de tempo de simulação) o servidor envia para cada cliente as informações correspondentes aos seus sensores e espera a resposta para os seus respectivos atuadores [13].

As informações disponíveis para cada piloto incluem tanto aquelas importantes para o controle da direção, como posicionamento em relação à pista, distância dos oponentes em sua proximidade, velocidade e marcha atual, quanto as que são úteis para a avaliação

[1] http://torcs.sourceforge.net/index.php?name=Sections&op=viewarticle&artid=19
[2] http://artlibre.org/licence/lal/pt

Figura 2.1: Captura de tela de uma corrida realizada no *TORCS* [11].

do desempenho (e, consequentemente, para estratégias de Inteligência Artificial) como distância percorrida, tempo decorrido em relação ao início da corrida, posição e dano causado ao veículo.

Graças a sua licença, modularidade, extensividade, e, principalmente, qualidade de simulação, TORCS foi adotado como base para muitos projetos de pesquisa. Além do desenvolvimento de controladores, já se utilizou este simulador em trabalhos de Inteligência Artificial direcionados a diferentes desafios em jogos e simulações de corrida, como otimização das configurações físicas de um veículo [10], determinação do caminho ótimo em um circuito [5], geração de novos circuitos [12] e, especificamente, ultrapassagens e estratégias para lidar com adversários [17]. Outras competições realizadas utilizando o simulador incluem: uma voltada para a geração de uma configuração ótima para um veículo [3], outra focada em resiliência [17], além daquelas informais, realizadas por entusiastas na área.

O simulador lida com uma série de configurações de corrida durante a sua execução. Além dos diferentes modos listados na seção 2.1.4, os arquivos de configuração de corrida contêm um conjunto de detalhes como a sua duração, o circuito selecionado, os controladores participantes e a ordem de partida. A duração pode ser expressa em voltas a serem completadas ou em uma distância em metros a ser percorrida. Neste último caso, o cálculo do número de voltas é feito com base no comprimento da pista, usando-se o teto do valor obtido na divisão, de forma que os carros percorrem a distância determinada e mais o que for preciso para completar uma última volta.

Um arquivo de configuração de corrida é fornecido ao TORCS para a sua execução em linha de comando. Nesse caso, o jogo imprime no terminal, a cada volta completada pelo primeiro colocado, o tempo cumulativo da simulação até então. Essa execução não é feita em tempo real, a duração de uma volta é da ordem de segundos. Ao final, o processo é encerrado. Para a execução tradicional em interface gráfica, as características das corridas são especificadas na própria interface, através de botões. O processo só é encerrado quando o usuário solicita, e as simulações são feitas em tempo real, da ordem de segundos, mas podem ser aceleradas em até 128 vezes.

O TORCS fornece uma forma prática de se desenvolver robôs, os agentes do próprio jogo. Porque seu código é aberto, é possível estudar e modificar os que estão incluídos na distribuição, além de criar novos módulos de pilotos fazendo uso da mesma estrutura. Estes são compilados em bibliotecas compartilhadas, de forma que não é necessário recompilar todo o jogo após mudanças no código dos pilotos.

2.1.1 Terminologia

A terminologia usada neste trabalho envolve principalmente os pilotos (também referenciados como controladores), agentes que controlam veículos e produzem uma resposta (comandos para os atuadores) a partir de um conjunto de estímulos (sensores). Um conjunto de controladores diferentes que são desenvolvidos e testados juntamente é denominado uma população de indivíduos. Denomina-se cliente o módulo composto pela classe de controlador juntamente ao código que gerencia a comunicação com o servidor.

O simulador, jogo ou servidor, quando se faz referência ao funcionamento em arquitetura cliente-servidor do SCR, é a plataforma usada para a execução de corridas, o TORCS. Os processos de desenvolvimento de controladores mencionados são trabalhos que usam técnicas de Inteligência Artificial que procuram otimizar parâmetros dos algoritmos dos pilotos. Para tanto, eles usam diferentes estratégias para gerar novos indivíduos, com novas combinações de parâmetros, buscando assim a melhor possível. Para julgar a qualidade de cada um desses indivíduos, é preciso determinar uma forma de avaliação para estes. No caso do TORCS, por se procurar pilotos que se saem bem em corridas é necessário inserí-los nessa situação para que se possa averiguar os seus resultados.

As funções de avaliação de desempenho descrevem as informações escolhidas para quantificar a qualidade de um controlador, traduzindo-a em um número usado para classificação, utilizando para tanto pesos associados a cada fator considerado e uma fórmula que os relaciona para a determinação do valor final. As referências a custos envolvidos tratam do custo computacional em termos de tempo exigido para completar um processo em *hardware* e *software* equivalente aos das máquinas comerciais da atualidade.

2.1.2 Os controladores

Os pilotos são os personagens não jogáveis em um jogo de corrida. Eles competem contra os jogadores e, como eles, podem ter diferentes níveis de habilidade em diferentes situações. Desenvolvê-los não é uma tarefa trivial devido à grande quantidade de fatores envolvidos em uma simulação sofisticada como é o caso do TORCS. Atualmente o simulador estudado permite a criação de controladores novos de duas formas.

A primeira é a criação de um módulo local de pilotos junto àqueles que vêm na própria distribuição do TORCS. Nesse caso, compilam-se os arquivos-fonte em uma biblioteca compartilhada que será utilizada pelo programa principal quando o controlador for selecionado para participar de uma corrida. Cada módulo é limitado a um número máximo de dez controladores, que podem, ou não, ter comportamentos distintos uns dos outros. Existem recursos para a especificação de configurações únicas para cara controlador, em cada pista e modo de corrida diferente.

Além disso, pode-se alterar as propriedades visuais dos veículos, do *pit stop* e a bandeira da equipe. Bernhard Wymann, atualmente o principal desenvolvedor do *software* de

simulação, disponibiliza tutoriais em sua página da *internet*[3] que ilustram detalhadamente a exploração dos recursos oferecidos, além de um módulo base já configurado para compilação e integração com o simulador (arquivo `Makefile`). Adicionalmente, os arquivos-fonte de todos os outros módulos estão disponíveis para serem estudados e alterados livremente.

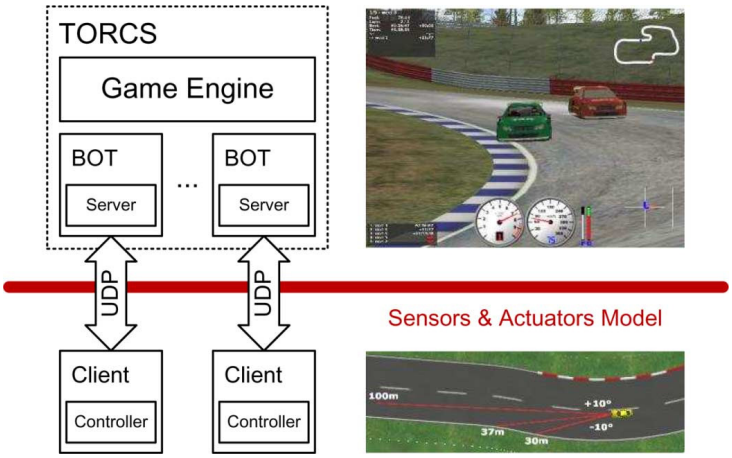

Figura 2.2: Arquitetura cliente-servidor do simulador *TORCS* [13].

Os controladores pertencentes a esses módulos usam, para determinar as suas decisões, das mesmas informações sobre a pista e sobre os seus adversários que simulador durante a execução de uma corrida. Assim, eles têm acesso, a todo o tempo, a informações sobre o formato da pista e sobre o posicionamento e a velocidade de todos os outros carros participantes da corrida. O simulador ainda precisa esperar, a cada passo, que todos os controladores retornem os valores correspondentes às suas ações para continuar o processo. A simulação, nesse caso, fica vulnerável a problemas causados pelos pilotos, já que caso um deles demore muito para retornar ou cause falha, toda a execução é prejudicada, podendo apresentar atrasos ou ser interrompida abruptamente no caso da ocorrências de exceções não tratadas.

Uma outra forma de funcionamento é a conexão em protocolo UDP entre servidor e clientes (Figura 2.2). Esse modo funciona com o uso de um módulo especial adicional, desenvolvido especialmente para o SCR, cujos dez controladores se comunicam usando as portas 3001 a 3010. Dessa forma, é possível selecionar qualquer combinação entre controladores originais, modificados e os que utilizam essa forma de conexão, observando apenas o limite de dez veículos por corrida. Essa opção proporciona a liberdade de se utilizar qualquer linguagem capaz de trabalhar com esse protocolo para desenvolver e testar controladores, não apenas C/C++ como no caso de controladores acoplados, assim como a possibilidade da realização de simulações remotamente.

Durante a simulação, a cada tique o servidor envia para cada cliente as suas informações de entrada específicas e espera 10ms para receber uma ação. Caso não receba, repete a última ação recebida, o que permite que a simulação não seja prejudicada por um controlador demasiadamente demorado ou que falha durante a corrida. Suas vantagens também incluem a independência entre as camadas, já que o simulador não precisa esperar a resposta dos pilotos para continuar com o funcionamento, e a limitação das

[3] http://www.berniw.org/tutorials/robot/tutorial.html

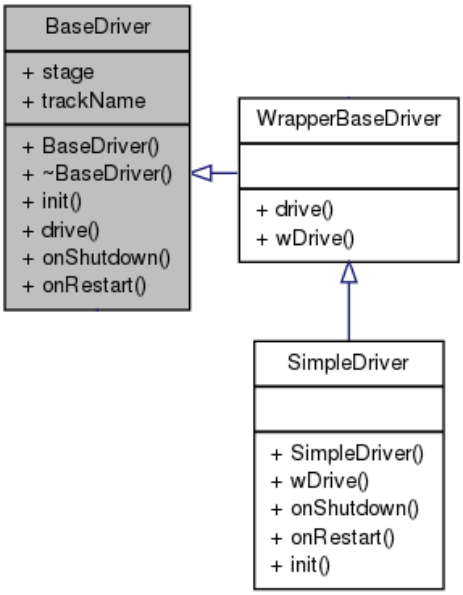

Figura 2.3: Hierarquia das classes *BaseDriver*, *WrapperBaseDriver* e *SimpleDriver* (atributos de *SimpleDriver*, retorno e argumentos dos métodos omitidos)

informações disponíveis para os clientes [13], de forma que os dados trocados em sensores e atuadores assemelham-se aos que poderiam ser acoplados a um veículo real - não se tem a informação exata do formato do circuito ou dos outros carros na pista. Também é fornecido livremente um controlador básico a ser estendido, nas linguagens C++, Java e Python, que operam com essa arquitetura.

O cliente disponibilizado em C++ traz facilidades para lidar com a arquitetura cliente-servidor. As classes *CarControl* e *CarState* são abstrações dos atuadores e sensores, respectivamente. Seus atributos são os dados trocados na conexão e ambas têm métodos para conversão de seus dados em *string* e para a determinação dos valores a partir de uma *string*, nos dois casos no mesmo formato em que são trocadas as mensagens entre clientes e servidor. A classe abstrata *WrapperBaseDriver*, que deve ser estendida pelos controladores em si (Figura 2.3), possui um método *drive* que recebe como parâmetro uma *string*, referente às informações dos sensores, e retorna uma *string* referente aos atuadores. Assim, o algoritmo do controlador deve apenas interpretar essas informações e retornar comandos para os atuadores.

A arquitetura cliente-servidor funciona com executáveis separados para cada cliente, cada um inclui apenas um piloto e encerra seu funcionamento após o final de uma corrida. Dessa forma, realizar k corridas, cada uma com n controladores, implica na chamada de executáveis $k \times n$ vezes. Se essas execuções forem feitas com o TORCS em linha de comando, este também precisa ser chamado novamente para cada execução, o que leva a $k \times (n + 1)$ chamadas. Nota-se que fazer muitas simulações, o que é necessário para um processo de desenvolvimento de pilotos, rapidamente torna-se trabalhoso e cansativo, podendo chegar a ser impraticável para altos valores de k e n.

2.1.3 Sensores e atuadores

Enquanto os pilotos acoplados ao *software* principal têm acesso a informações referentes à estrutura de todo circuito e ao posicionamento e velocidade de todos os outros veículos, os controladores que funcionam na arquitetura cliente-servidor têm as informações organizadas de forma que cada veículo capta um conjunto restrito de dados sobre o ambiente em sua volta, com alcance limitado. Esta observabilidade parcial é mais próxima à situação de um robô real controlador de um veículo automotor, sendo necessária a interpretação da informação dos sensores para determinar que ações serão tomadas a cada instante.

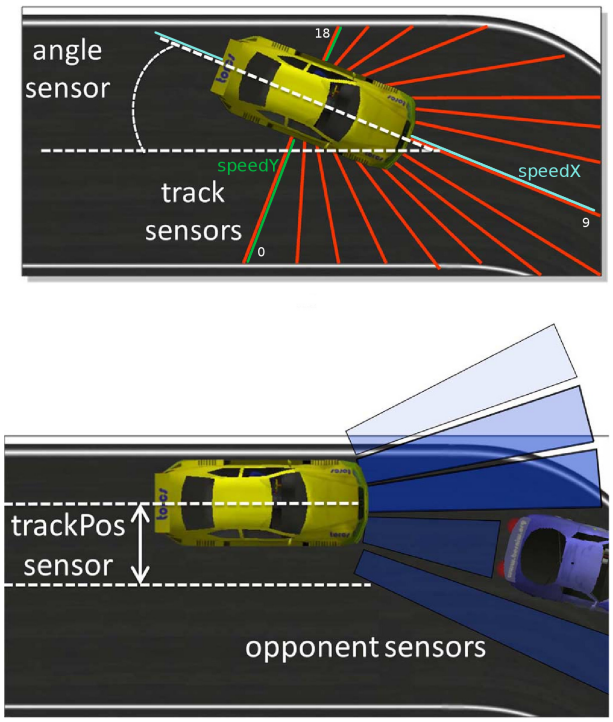

Figura 2.4: Ilustração dos sensores *angle*, *track* (configuração padrão), *trackPos*, *opponents* e das direções relativas aos sensores *speedX* e *speedY*. Adaptado de [14].

São ao total 19 sensores que fornecem dados relativos à situação atual do carro, como seu posicionamento em relação à pista, velocidade e distância de oponentes próximos, além de seu desempenho na corrida, como tempo da última volta, posição em relação aos outros carros e pontos de dano sofridos. Os atuadores são os mesmos de um veículo real, como aceleração, freio, posição do volante e marcha.

A Figura 2.4 ilustra os sensores *angle*, que informa ângulo do carro em relação ao eixo longitudinal do trecho atual da pista, *track*, que fornece a distância da borda da pista a partir de diferentes ângulos partindo do centro do carro, *trackPos*, que informa a posição do veículo em relação à largura da pista, e *opponents*, que informa a distância do adversário mais próximo em intervalos angulares, fundamentais para estimar a situação atual do veículo na pista e, consequentemente, para a realização de ações durante uma corrida.

2.1.4 Circuitos e modos de corrida

As pistas inclusas no TORCS são diversas em suas características, o que deve ser levado em consideração no desenvolvimento de controladores. Estratégias mais ousadas, que atingem maiores velocidades e não desaceleram tanto para fazer curvas, devem apresentar melhores resultados em pistas largas e com curvas amplas, pois estas permitem atingir e manter velocidades mais altas, enquanto comportamentos mais conservadores, que percorrem curvas mais lentamente, são mais adequados para pistas estreitas que contêm curvas fechadas.

Os circuitos incluídos na distribuição do simulador, ilustrados no Anexo A, são divididos em três categorias. *Oval* inclui pistas largas de alta velocidade com curvas suaves. Estas são mais curtas do que a média e se assemelham a circuitos típicos de corridas de *Stock car*. Esse tipo de circuito exige do controlador a habilidade de manter-se em velocidade alta constantemente e a ultrapassagem é facilitada pelo amplo espaço lateral disponível.

As pistas da categoria *road* têm sempre pelo menos um trecho relativamente lento, que exige uma significativa diminuição da velocidade do veículo. Seus formatos são bem mais diversos e irregulares em comparação com a categoria anterior, assim como seus comprimentos, entre 2 e 22km.

A última categoria chama-se *dirt*, cujas pistas têm superfícies irregulares e são, completa ou parcialmente, de terra. Esse tipo de terreno afeta a visualização do piloto e também o desempenho do carro, de acordo com seus atributos físicos.

O jogo ainda permite a adição de novos circuitos, que podem tanto ser usados por jogadores humanos quanto para a realização de testes de controladores. Para alcançar pilotos que se saem bem em situações diversas e desconhecidas, é desejável realizar testes em pistas com características distintas, de forma que se avalie como o indivíduo de uma forma abrangente.

Os tipos de corrida disponíveis são:

- *Quick race*: corrida única, seleciona-se um circuito, um conjunto de controladores e determina-se a distância em quilômetros ou em voltas. Este modo proporciona a opção de visualizar os resultados apenas, não necessariamente exibir toda a corrida, e os dados são fornecidos para cada controlador: tempo de cada volta, menor tempo e as velocidades máxima e mínima registradas durante a corrida. A ordenação é determinada nas configurações da corrida.

- *Practice*: similar à *quick race*, porém limitada a um único controlador.

- *Non-championship race*: corrida de uma qualificação para determinar a ordem de partida dos veículos. Em seguida, realiza-se outra corrida com comprimento de 150 quilômetros.

- *Endurance race*: semelhante à anterior, porém realiza a corrida com comprimento de 500 quilômetros. Em ambos os casos, a quantidade de voltas é calculada de acordo com o tamanho da pista.

- *Championship*: são realizadas doze etapas constituídas, cada uma, de qualificação e corrida subsequente de 180 quilômetros. Os circuitos utilizados são predeterminados (g-track-1, e-track-6, g-track-3, street-1, wheel-2, ruudskogen, aalborg, alpine-1,

dirt-3, michigan, forza e alpine-2, ver Anexo A). Estes são bastante diferentes uns dos outros, todos pertencem à categoria *road*, com exceção de dirt-3 (categoria *dirt*) e michigan (categoria *oval*) . Assim, espera-se chegar a um resultado condizente com a qualidade dos controladores participantes. Essa opção permite salvar e carregar um campeonato parcialmente realizado, já que sua duração é longa.

- *Challenge race*: é realizada uma qualificação, uma corrida de 50 quilômetros e uma subsequente de 180 quilômetros.

2.1.5 Formas de execução: GUI e CL

Até o lançamento da versão 1.3.3, o TORCS oferecia apenas o funcionamento em interface gráfica. Esta precisa de uma configuração prévia da corrida a ser executada, especificando o circuito, tipo de corrida, duração e pilotos participantes. A execução em tempo real é muito lenta para a realização repetida de testes, mas permite uma observação do comportamento dos controladores que pode contribuir para o processo de desenvolvimento, mais informações são obtidas dessa forma do que apenas a partir de dados como o tempo gasto para completar uma ou várias voltas.

A possibilidade de execução em linha de comando permite uma maior agilidade na execução das simulações (da ordem de segundos), mas as informações obtidas são limitadas. A chamada de uma instância em linha de comando do TORCS imprime, durante a execução, uma sequência de linhas no seguinte formato:

```
Sim Time:    t [s], Leader Laps:    i, Leader Distance:    d [km]
```

Onde t é o valor do tempo e d é a distância coberta pelo líder até então, ambos cumulativos. O número i indica a volta correspondente e é fornecida uma linha por volta. O valor de t é exibido com precisão de duas casas decimais e, d, de três. Observa-se que esses valores não oferecem informações suficientes para situações onde se avalia mais de um controlador, já que não são informados valores específicos individuais e não se sabe a qual deles os dados dizem respeito. Estes ainda podem se referir a controladores diferentes já que, devido a ultrapassagens, aquele que está na primeira posição pode mudar ao longo da corrida.

Quando executado em linha de comando em ambiente Linux, o simulador registra informações mais completas em um arquivo salvo em */pasta-pessoal/.torcs/results/config-corrida/results-hora.xml*, onde *pasta-pessoal* refere-se à pasta do usuário, */home/nome-de-usuario* ou */root* no caso de superusuário, *config-corrida* é o nome do arquivo de configurações fornecido e *hora* contém a data e a hora na qual foi realizada a simulação, e.g. 2011-03-15-21-35-50.

Para que o controlador crie este arquivo, a pasta deve já existir e ter o mesmo nome do arquivo `XML` fornecido com as configurações da corrida. O formato do arquivo de resultados depende do modo de corrida selecionado, mas as informações contidas nele ainda são limitadas. Essas informações são, para cada volta: pontos de dano, maior velocidade atingida, menor velocidade atingida e melhor tempo de volta até então. Nesse caso, as informações são identificadas a um controlador, mas estas só são reportadas para um deles. Dessa forma, a não ser que se esteja interessado apenas nos dados referentes a um dos controladores, realizar avaliações com múltiplos carros exige obtenção dos resultados de maneira alternativa.

2.2 Estado da arte no desenvolvimento de pilotos

A forma de avaliação de indivíduos é extremamente importante em qualquer processo que procura encontrar uma configuração ótima para uma solução para um problema. É preciso fazer comparações constantemente de forma a selecionar os melhores indivíduos em cada etapa, pois entende-se que estes estão mais próximos do resultado desejado em processos evolutivos, majoritários dentre os pilotos que apresentam bons resultados. Dessa forma, descartam-se os restantes e trabalha-se em cima daqueles que foram considerados superiores. Existem diversos fatores que influenciam nesses processos, que podem variar bastante dependendo da estratégia empregada. Esses fatores precisam ser levados em consideração no planejamento de ferramentas como a TTI, pois fazem parte de decisões tomadas pelos desenvolvedores, que devem ter a liberdade de fazê-las da forma que considerarem melhor.

2.2.1 Definições iniciais

Quando varrer todas as combinações possíveis é inviável, começar com uma qualquer, aleatória, e ir caminhando na direção mais promissora é uma alternativa para lidar com o problema. Determinar qual é essa direção depende da forma de avaliação e classificação utilizada. É preciso chegar a uma forma de quantificar a qualidade de um indivíduo que permita estimar a sua habilidade. Também se pode estabelecer um critério de parada da busca com base nessa avaliação, i.e., um valor mínimo a ser alcançado pelo processo antes que ele encerre a sua execução.

Considera-se aqui que o objetivo de um trabalho de desenvolvimento de piloto para o SCR seja chegar a um controlador que atinja tempos competitivos em corridas, sejam individuais ou junto a outros veículos. Adicionalmente, deseja-se a habilidade de fazê-lo nos mais variados tipos de circuitos, pois aqueles usados no campeonato não são previamente conhecidos pelos competidores.

A avaliação natural é tomar o tempo gasto para percorrer um número de voltas ou uma distância pré-estabelecida e trabalhar com uma relação inversa, de forma que os mais rápidos (i.e., que atigem os menores tempos) obtenham uma melhor (ou maior, quantitativamente) avaliação. Outra alternativa é considerar um intervalo de tempo (ou de tiques do jogo) fixo e utilizar a distância total alcançada pelo indivíduo, também considerando a maior velocidade média no intervalo de teste. Entretanto, existem diversos fatores importantes adicionais que precisam ser considerados.

O simulador não insere qualquer tipo de aleatoriedade nas corridas, de forma que, contanto que os controladores também não o façam, todos os resultados são determinísticos. Pode haver variações, entretanto, quando ocorrem perdas durante a comunicação, e estas nem sempre podem ser previstas, evitadas ou tratadas a ponto de não influenciarem nos resultados. Ainda há muito o que se considerar acerca de como realizar as corridas. É necessário determinar qual circuito, ou conjunto de circuitos, utilizar. A quantidade de veículos e a ordem da partida em cada teste são outras decisões importantes que afetam significantemente as informações coletadas.

2.2.2 A escolha dos circuitos

Selecionar circuitos adequados para a avaliação é crucial para todo o processo de desenvolvimento de controladores. A avaliação de desempenho deve levar em conta todas as qualidades procuradas nos indivíduos, ou seja, os circuitos devem cobrir uma grande variedade de curvas de diferentes dificuldades. Utilizar uma pista com retas e curvas amplas provavelmente levará a um controlador que acelera bastante e se sai bem em pistas rápidas, mas não consegue manter-se dentro dos limites da pista quando encontra curvas fechadas. Já o uso de um circuito mais lento deve levar a um conservadorismo que não permite o alcance de velocidades suficientemente altas em trechos mais velozes. Ambos os casos são ruins e dificilmente conseguirão competir em situações desconhecidas com robôs treinados em ambientes mais diversos, que reconhecem essas diferentes situações têm habilidade para lidar com elas.

Os desenvolvedores do controlador Mr.Racer [18] mostraram que nem sempre os dados recebidos pelos sensores do veículo permitem ao piloto a extração de informações suficientes sobre a curvatura do trecho à frente do carro (Figura 2.5) e, dessa forma, não é trivial encontrar a melhor decisão a se tomar baseando-se somente nelas. A solução proposta foi tomar proveito da etapa de aquecimento do campeonado, que permite a cada controlador percorrer o circuito da corrida sozinho e extrair informações a serem utilizadas posteriormente, de forma a se mapear todo o trajeto. É importante notar que, porque as informações podem ser semelhantes para curvas com características extremamente diferentes, fazer testes dentro de apenas um intervalo limitado muito provavelmente levará à seleção de indivíduos que exibem um comportamento adequado para aquele intervalo apenas e acabe tendo um desempenho medíocre quando se depara com curvas que possuem características suficientemente diferentes para exigir decisões distintas.

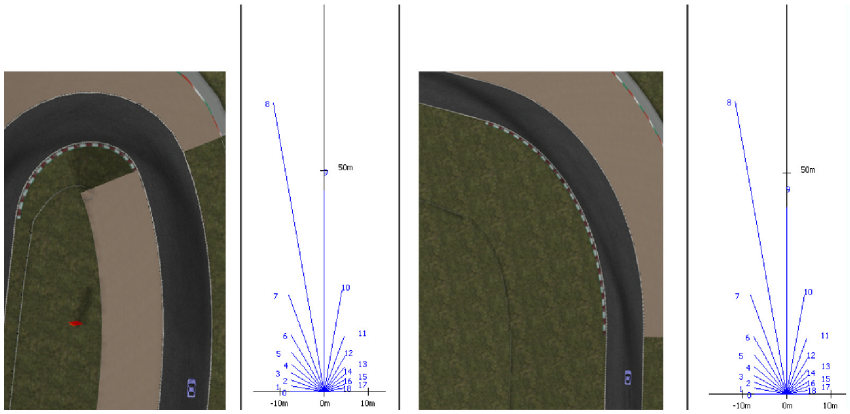

Figura 2.5: Informação enganosa do sensor *track*, quase idêntica para situações que exigem ações diferentes [18].

A questão da variedade de curvas em pistas foi estudada juntamente a um trabalho de geração de novos circuitos com o uso de computação evolucionária [12]. Foram propostas duas avaliações que descrevem características diferentes. A primeira, baseada em um perfil de curvatura, realiza uma análise do formato do circuito, dividindo-o em trechos e analisando a curvatura de cada um. Já o segundo, baseado em um perfil de velocidade, faz a análise da velocidade média atingida por um grupo de controladores em cada trecho. Assim, calculando a entropia das duas distribuições, pode-se chegar a uma avaliação

do quão diversa a pista é. Essa geração de circuitos foi feita com o objetivo de criar aqueles que seriam usados no campeonato de 2010 e procurava-se chegar a indivíduos com alta entropia, já que se considera que há uma relação direta entre a entropia e o quão desafiador um circuito é. Além disso, a realização da competição utilizando circuitos com alta diversidade de curvas e velocidades permite a avaliação abrangente de como os controladores se adequam às mais variadas situações, necessitando-se, para tanto, de uma única corrida. Essa vantagem pode e deve ser explorada também durante os treinamentos, e espera-se, portanto, que seja benéfico que o processo de avaliação escolha pistas com essas características. Curiosamente, o treinamento do Mr.Racer [18] foi feito na *Wheel-2* (Figura 2.6), a pista original do jogo que teve a maior entropia dentre as exemplificadas.

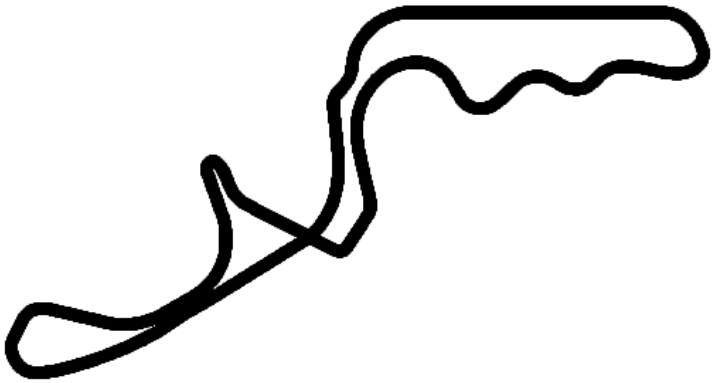

Figura 2.6: Wheel-2, um circuito com alta diversidade de curvaturas e velocidades [12]

2.2.3 Ordenação e intervalo considerado

A ordenação dos veículos na partida pode afetar bastante os resultados. Um controlador que não lida bem com ultrapassagens ou qualquer adversário à sua frente pode obter bons e enganosos resultados se estiver posicionado em primeiro lugar desde o início. Outra possibilidade é a de que um controlador tenha seu desempenho avaliado significantemente reduzido por, em algum ponto, estar atrás de um veículo que esteja parado, se sofrer colisões ou encontrar-se em qualquer outra situação adversa. É verdade que se espera que o controlador lide com tais dificuldades e as supere, mas nesse caso normalmente se compara esse resultado, alterado, com o de outros indivíduos que não se depararam com os mesmos problemas. Tentativas de minimização dessas distorções envolvem rodízio das posições, de forma que todos realizem a largada em cada uma das colocações exatamente uma vez, ou mesmo avaliar cada combinação possível.

Também é importante considerar quantas voltas cada avaliação vai durar, ou, alternativamente, quanto tempo o controlador tem para cada avaliação. Pode-se ignorar o tempo da primeira volta, ou considerá-lo separadamente, já que esta tem a particularidade de iniciar com todos os veículos parados e normalmente resulta em uma volta mais demorada do que as demais. Espera-se que, quanto maior a quantidade de voltas (ou tempo) disponibilizado para o controlador, melhor se estime a sua qualidade.

2.2.4 Informações auxiliares

Levar os pontos de dano do veículo em consideração é mais um fator de extrema importância. Proveniente de colisões com adversários, com os limites do cenário ou mesmo da saída dos limites de pista, danos afetam o desempenho do carro e podem distorcer os resultados encontrados e, consequentemente, a avaliação calculada. É possível desativar a consideração dos danos, eliminando o problema, porém essa prática priva a avaliação da informação da quantidade de colisões sofridas, ou causadas, pelo controlador. Em algumas situações, essas dificuldades podem não ser culpa do próprio indivíduo, mas uma das qualidades que se espera é a minimização de colisões que o são.

O treino realizado sem levar danos em consideração, quando sua avaliação final o faz, muito provavelmente resultará em uma discrepância entre a qualidade estimada ao fim do treinamento e a observada na própria simulação, além de poder levar à retirada do controlador da competição após um certo intervalo de tempo. Pontos de dano ainda são critérios de desempate no campeonato SCR, o que torna ainda mais importante levá-los em consideração durante o desenvolvimento de controladores.

Ser capaz de visualizar o funcionamento do carro faz bastante diferença, principalmente em estratégias assistidas por humanos. Apenas a informação do tempo necessário para que o controlador complete uma volta não fornece informações sobre quais características problemáticas, ou passíveis de melhora, ele está exibindo. Algumas dessas podem ser amenizadas junto à configuração das características físicas do carro. Por exemplo, um veículo que se prejudica e sai da pista quando encontra certas curvas porque está derrapando pode ter seu desempenho melhorado através de um acerto nas rodas e nos pneus. Esse tipo de ajuste, que é facilmente captado a partir da simples observação do comportamento do controlador, pode levar a uma economia de muitos ajustes em termos de comportamento e até mesmo ajudar a encontrar indivíduos mais aptos ao final.

2.2.5 Influência sobre os resultados

Observamos a relevância da estratégia de avaliação utilizada e a sua influência em relação aos resultados obtidos no trabalho de desenvolvimento do controlador Kamikaze [7]. O seu nome advém da estratégia do indivíduo resultante, que mantém uma velocidade alta sempre e, ao se aproximar de uma curva, apenas ajusta a sua direção suficientemente para que, assim que colide com os limites da pista, o carro já fique alinhado na direção certa e precise apenas continuar acelerando. O controlador resultante conseguia atingir bons tempos, porém a sua estratégia de aproveitar batidas para realizar curvas e manter a velocidade alta não era estável o suficiente para continuar funcionando bem após dez ou mais voltas. Nesse caso, a única informação utilizada como avaliação foi a posição conquistada por cada indivíduo em cada corrida, não foi levado em consideração o dano resultante e cada etapa de avaliação consistia de apenas uma única volta. Além disso, a decisão de realizar todo o treino em duas das pistas mais difíceis levou a um piloto cujo desempenho em pistas fáceis (da categoria *oval*) era precário.

O trabalho que desenvolveu controladores de forma *online* usando neuroevolução [4] estabeleceu intervalos de avaliação dentro de uma volta, medidos em tiques do jogo. Os intervalos foram testados em diferentes comprimentos e encontrou-se uma relação significativa entre o comportamento da população, em termos de qualidade do melhor indivíduo, e o tamanho do intervalo escolhido. Observou-se que intervalos curtos (500

tiques) levavam a um aprendizado veloz no início mas resultavam em um desempenho final bem pior após a mesma quantidade de voltas, enquanto intervalos longos (em torno de 6000 tiques, o suficiente para completar duas voltas no circuito) resultavam em um aprendizado muito lento. Intervalos de 1000 e 2000 tiques levaram ao melhor resultado, com poucas diferenças entre o comportamento dos dois. A avaliação, nesse caso, envolveu computar o número de tiques nos quais o carro estava fora da pista e a distância percorrida no intervalo.

Diferenças em resultados podem ser causadas pelas estratégias utilizadas, que podem divergir bastante. Alguns algoritmos podem envolver aleatoriedade, ao contrário de outros. As possíveis configurações também podem depender de como funciona o algoritmo de evolução, e essas exercem influência sobre os indivíduos resultantes.

2.3 *Software* Livre e a licença GPL

O termo "Livre" em *Software* Livre refere-se à liberdade do usuário para copiar, distribuir, estudar, modificar e incrementar um programa. Mais precisamente, são definidas quatro liberdades [21]:

1. A liberdade para executar o programa para qualquer propósito.
2. A liberdade para estudar como o programa funciona e para adaptá-lo às próprias necessidades.
3. A liberdade para distribuir cópias.
4. A liberdade para melhorar o programa e disponibilizá-lo para o público de forma que toda a comunidade seja beneficiada.

Um programa é considerado *Software* Livre se seus usuários têm todas essas liberdades, que significam que não é necessário pedir permissão, o usuário é livre para executar qualquer das quatro ações. Também deve estar assegurado o direito de fazer modificações e usá-las de forma privada sem precisar anunciar que estas existem [21].

As liberdades 2 e 4 exigem acesso ao código-fonte do programa, de forma que essa é uma condição essencial para que um programa seja realmente livre, pois somente quem tem acesso ao código pode estudar o seu funcionamento e fazer modificações. Essa possibilidade promove cooperação na comunidade, contribuições podem ser feitas por diversos indivíduos para adicionar funcionalidades ou remover *bugs*.

É possível estabelecer regras sobre esses programas contanto que não existam conflitos com as quatro liberdades. Essa é a função das licenças de *Software* Livre, regular o uso e distribuição, assegurando que versões futuras do produto não restrinjam nenhuma delas [21]. A *GNU General Public License*[4] (GPL) é uma das licenças de *Software* Livre disponíveis, e esta é usada tanto pelo TORCS quanto pelo *patch* do SCR.

A possibilidade de qualquer usuário contribuir para o programa promove um desenvolvimento colaborativo, de forma que indivíduos com diferenciadas perspectivas, criatividades e experiências enriqueçam o projeto, além de corrigir problemas não identificados

[4] Tradução não oficial para o português disponível em http://creativecommons.org/licenses/GPL/2.0/legalcode.pt

pelos próprios autores. Isso beneficia toda a comunidade, pois as pessoas podem ter acesso a versões mais completas e robustas do programa. É fundamental a manutenção de uma documentação de qualidade, de forma a facilitar essas contribuições.

A ferramenta aqui proposta usa também a licença GPL, possibilitando esse desenvolvimento conjunto de membros da comunidade que têm experiência no desenvolvimento de controladores. Assim, pode-se enriquecê-la, de forma a adicionar mais funcionalidades a ela, ou a torná-la mais útil para diferentes projetos.

Capítulo 3

TORCS Training Interface

3.1 Proposta

O objetivo deste trabalho é construir a *TORCS Training Interface*, uma ferramenta que atue como interface entre o servidor e um algoritmo de evolução computacional, além de outras abordagens que necessitem executar um processo de avaliação, que ofereça funcionalidades úteis e facilidades para lidar com o TORCS. Procura-se oferecer uma seleção rápida, flexível e simples das configurações de corridas e uma melhoria da eficiência do processo de avaliação. Pretende-se, adicionalmente, oferecer valores de retorno significativos, úteis para uma classificação dos indivíduos de uma população.

Por ser uma plataforma bem definida, implementada e mantida, dotada de reconhecimento internacional, o TORCS é uma ferramenta usada em diversas pesquisas científicas [11]. Espera-se que disponibilização de uma ferramenta que forneça facilidades nesse sentido leve a uma economia de esforços gastos em processos manuais e repetitivos, além de evitar o desenvolvimento desnecessário de *scripts* específicos para tais propósitos, direcionados a cada trabalho, quando boa parte desse código poderia ser reaproveitada externamente pela comunidade científica. O próprio simulador já vem adicionando recursos direcionados a facilitar esse processo, como a execução por meio de linha de comando e a possibilidade de solicitação de reinício de uma corrida, que só recentemente foram implementados[1]. Entretanto, acredita-se que ainda há espaço para contribuições sobre os recursos disponíveis, para as quais este trabalho está direcionado.

A maior parte dos fatores discutidos têm soluções simples, porém estas costumam elevar o custo de cada etapa de um processo iterativo de desenvolvimento. Trabalhar com um conjunto amplo de circuitos, repetir os testes para todas as combinações possíveis, executar um número grande de voltas e observar as simulações no modo de interface gráfica multiplicam a duração de uma simulação simples, que, de início, já é relativamente alto (da ordem de segundos).

Foi utilizado neste trabalho a última versão do TORCS disponível, 1.3.4, em ambiente Unix (Ubuntu versão 12.10). Trabalha-se, aqui, com o funcionamento em arquitetura cliente-servidor do TORCS, usada no campeonato SCR.

A fim de aproveitar as facilidades oferecidas e exigir o mínimo de alterações adicionais, procurou-se ao máximo fazer uso dos recursos já fornecidos no cliente disponibilizado, adaptando-o de forma a atender às novas necessidades de coletas de informações. Isso

[1] v1.3.3 de 17/02/2012

só foi possível porque tanto o TORCS quanto o *patch* do SCR usam licença de *Software* Livre, permitindo estudo e modificação do código. Dessa forma, é reduzida a necessidade de alterações em controladores já em processo de desenvolvimento e é feita mais natural e intuitiva a familiarização de novos usuários com a ferramenta. Também se fez uso de recursos do próprio simulador, sabendo-se que este necessariamente estaria disponível em uma máquina que usaria faria tais simulações.

A *TORCS Training Interface* é aqui proposta como ferramenta para auxiliar desenvolvimento de controladores para o SCR, oferecendo automatizações não anteriormente disponibilizadas, assim como permitindo configurações para adequação a diferentes projetos. Ela não interfere nas corridas e portanto não influencia o funcionamento do simulador, mas sim oferece uma abstração sobre todo o processo de simulação.

A ferramenta administra as chamadas de processos do simulador para a execução de uma corrida e de um cliente para cada controlador participante, todos devendo funcionar paralelamente. Essa funcionalidade procura amenizar o problema da necessidade de constantes intervenções manuais, automatizando as chamadas que precisam ser feitas. A TTI também faz um processamento dos resultados da corrida para cada controlador de forma independente da sua classe, retornando então dados mais completos do que os que são impressos pelo simulador no terminal quando executado em linha de comando. Ambos os fatores são benéficos e importantes para o desenvolvimento de pilotos, de forma que se espera que a TTI seja uma alternativa viável e potencialmente vantajosa em comparação com as formas de testes observadas na literatura.

3.2 Estrutura da TTI

A ferramenta desenvolvida faz uso de boa parte do código do módulo cliente do SCR, que contém algoritmos que gerenciam a conexão e provêem mapeamento dos sensores e atuadores em objetos *CarState* e *CarControl*. São usadas todas as classes, exceto a própria classe do piloto, como a *SimpleDriver*, já que essa deve ser fornecida para o treinamento. Esse código foi estendido de forma a atender às funcionalidades implementadas na TTI.

Foram criadas duas classes principais, *TTI* e *RaceSet*. A primeira gerencia as chamadas do simulador, feitas através de chamadas *pipe*, que permitem a execução de comandos do terminal e a leitura da saída impressa do programa, e as chamadas dos controladores, cada um em uma *thread* própria, de forma a oferecer paralelidade, já que durante uma corrida todos os controladores devem funcionar o tempo todo e dessa forma eles não se prejudicam entre si. A segunda lida com entrada e saída de configurações e com a organização das corridas solicitadas.

A criação de uma instância da classe *TTI* requer o fornecimento de uma coleção de referências para controladores que devem herdar da classe *TTIWrapperDriver*, além de configurações para a TTI, fornecidas em um arquivo `XML`. A classe *TTIWrapperDriver* herda da *WrapperBaseDriver*, da qual originalmente as classes de cotroladores deveriam herdar. *TTIWrapperDriver* tem acesso aos dados dos sensores e atuadores a cada tique do jogo e faz computações necessárias para o retorno dos resultados dos testes ao final da corrida. Dessa forma, uma classe de controlador que é executada com o módulo cliente precisa apenas passar a herdar de *TTIWrapperDriver*, nenhuma outra alteração é necessária e o mesmo vale para a relação inversa.

O arquivo de configurações em `XML` é lido durante a construção do objeto *RaceSet*. São retiradas deste uma coleção de corridas e suas especificações de circuito e duração. O uso de `XML` permite mudanças nos testes sem a necessidade de alterar o código fonte e recompilá-lo. Porque o TORCS trabalha com arquivos `XML` de configuração, que já deveriam ser lidos pela TTI, decidiu-se que esse formato era uma boa opção para essa definição de parâmetros.

O objeto da classe *RaceSet* guarda o conjunto de corridas e é responsável pela geração dos arquivos de configuração de cada uma delas, além da determinação de qual será a próxima a ser executada. Seu funcionamento é circular, executando-as em ordem e voltando à primeira depois da última, mas é fornecido um método que permite modificar o próximo índice na sequência de corridas a serem simuladas, permitindo um maior controle sobre a ordenação das corridas.

3.3 Configurações

Como exposto no Capítulo 2, existem vários fatores a serem considerados em relação à simulação, mais especificamente sobre que detalhes considerar, que compromissos fazer a fim de se encontrar uma configuração adequada ao trabalho sendo desenvolvido. É importante deixar essas decisões nas mãos dos responsáveis pela IA. É o caso, aqui, apenas disponibilizar o máximo de informações relevantes para serem usadas conforme a estratégia definida.

3.3.1 Circuitos

Com a TTI, o usuário tem a liberdade de escolher quais circuitos serão usados e a sua ordem, e não há checagem ou limites de quantidade, de forma que o usuário pode usar com a TTI circuitos adicionais, diferentes dos disponíveis na distribuição do TORCS, e da forma e ordem que preferir. Entretanto, erros de nomenclatura não são tratados. Fica a cargo do usuário identificá-los e consertá-los.

3.3.2 Duração das corridas

Aqui, tomou-se proveito do fato de que os arquivos de configuração lidos pelo servidor têm campos de duração em termos de distância, em voltas ou em metros. Dessa forma, o arquivo de configurações da TTI recebe esses dois tipos, verifica a validade e reflete os mesmos valores para as especificações do arquivo gerado.

3.3.3 Término adiantado

Estabelecer um desempenho mínimo a ser alcançado em pontos intermediários da simulação e descartar o controlador, economizando o custo de executar sua avaliação até o final, caso este não o faça é uma forma de aumentar a eficiência de testes demorados que precisam ser executados para vários indivíduos.

Para tanto, pode-se escolher uma distância mínima percorrida em cada subintervalo de avaliação. Essa prática permite o rápido descarte de indivíduos que não saem do lugar ou andam com velocidade muito baixa e transforma o tempo que seria gasto esperando o seu

período de avaliação terminar em tempo gasto obtendo informações realmente úteis. Os resultados de um trabalho por Haasdijk et al. [9] indicam que, nos casos onde o processo de avaliação é caro, essa interrupção é benéfica.

Essa opção está disponível nas configurações da TTI, mas não é obrigatória. Para fazer uso dela, precisam ser determinados dois parâmetros, *mindist* e *maxticks*. Então, a cada *maxticks* passos da simulação, verifica-se a distância percorrida no último intervalo. Caso seja menor do que *mindist*, encerra-se a sua simulação. Embora estabelecer verificações mais frequentes possa adicionar em eficiência, fazendo com que indivíduos pobres sejam detectados e descartados com maior rapidez, isso pode fazer com que veículos que encontraram problemas, possivelmente não causados pelos mesmos, possam ser erroneamente considerados como de má qualidade. A determinação desses valores fica a cargo do usuário.

A classe *RaceSet* lê de um arquivo `XML` fornecido, contendo informações sobre as corridas que devem ser realizadas. Os dados lidos são um *id* para o teste, usado para isolar os arquivos das suas corridas, configurações para o opcional término adiantado, e, para cada corrida, a pista e a duração, em voltas ou em metros, da simulação. Um exemplo desse arquivo está no Anexo B.

3.3.4 Configurações implícitas

Algumas configurações, embora não lidas de arquivo específico, são definidas por meio do próprio código. É o caso da quantidade de controladores nas simulações. Um conjunto de referências objetos (vetor) é fornecido à instância da classe *TTI* no momento da construção, contendo entre 1 e 10 elementos, inclusive (uma restrição do próprio TORCS). Cada corrida envolverá necessariamente todos estes indivíduos na ordem em que são fornecidos. É possível trocar o conteúdo de duas referências entre chamadas, dessa forma a próxima simulação será executada com os controladores na nova ordem.

Uma possibilidade para obter ainda mais flexibilidade do uso da TTI é manter várias instâncias da classe, cada uma com diferentes configurações ou diferente conjunto de controladores. Como exemplo, para se realizar testes entre dois controladores diferentes, tanto isoladamente quanto juntos, pode-se usar três objetos *TTI*, cada um para cada combinação desejada. Uma quarta poderia ser usada para se realizar testes com as duas possíveis ordenações dos dois testados conjuntamente.

Disposição da largada

A questão da ordenação também é bastante relevante e pode ter grande impacto sobre os custos de um ciclo de avaliação. Embora utilizar uma única ordem seja definitivamente mais ágil, parece benéfico, quando possível, incluir mais opções. Uma possibilidade é a realização de um rodízio entre os controladores, de forma que cada um fique em cada posição de largada uma única vez. Pode-se incrementar essa opção com a realização de uma corrida para cada controlador, sozinho, já que se deseja que estes tenham um bom desempenho também quando não estão lidando com adversários.

Embora não tenha sido implementada uma gerência automatizada da ordenação dos controladores em uma corrida, o usuário tem toda a liberdade de fazê-lo da forma como determinar melhor. Porque a TTI recebe como parâmetro uma coleção de referências para objetos de controladores, a troca dos conteúdos de duas dessas referências implicaria

na troca da ordem em que estas partem na corrida. Precisar-se-ia, nesse caso, considerar que os resultados também seriam retornados nas novas posições.

Interface gráfica

A possibilidade de se executar o servidor em modo de interface gráfica permite a observação do comportamento dos controladores, importante para a realização de *debug* durante o desenvolvimento a partir dessa observação, principalmente na fase inicial, onde se experimenta com o algoritmo para encontrar um comportamento satisfatório. Entretanto, essa opção limita a utilidade da interface, não permite uma configuração automática do circuito ou repetições automáticas. O jogo permite, nesse caso, acelerar a simulação (de potências de 2 entre 1 e 128 vezes), o que possibilita avaliações mais velozes. Entretanto, quanto mais veloz, menor é a visibilidade que se tem dos movimentos realizados pelo veículo, e perde-se proporcionalmente a vantagem de se assistir as corridas. A opção de não iniciar o TORCS está presente na TTI, e nesse caso, as corridas precisam ser configuradas na interface gráfica do jogo e inicializadas manualmente.

3.3.5 Geração dos arquivos XML

Após a leitura das configurações, são gerados os arquivos específicos para cada corrida, que posteriormente serão fornecidos ao simulador especificando a corrida a ser simulada a cada vez que este é chamado. O *id* lido é usado para dar nome a um diretório no qual serão guardados os arquivos de configuração gerados. Isso permite o uso de múltiplas instâncias da TTI, evitando que os arquivos de configuração de uma sobrescrevam os da outra. Dessa forma também se pode escapar de criações excessivas de arquivos que aconteceriam no caso de se gerar um identificador único automaticamente a cada execução. É necessário que o usuário tome o cuidado de especificar um *id* próprio para cada instância da TTI criada para evitar conflitos na execução.

Aplica-se ao *id* toda e qualquer restrição de nomes de diretórios. O diretório é criado, caso não exista. Cada arquivo `XML` gerado é salvo nesse diretório e nomeado `ri.xml`, onde $i \in [0, n-1]$ e n é a quantidade de corridas definidas nas configurações.

Essa geração é feita a partir de modelos mantidos pela TTI. Os arquivos `XML` de configuração lidos pelo TORCS na interface de texto contém uma grande quantidade de dados que especificam a corrida a ser simulada, e suas estruturas mudam conforme o modo de corrida utilizado e a quantidade de carros. Por isso, é mais simples e eficiente a cópia de arquivos prontos, apenas fazendo as modificações necessárias. Para cada corrida especificada nas configurações da TTI, faz-se uma cópia do modelo, modificando os valores de circuito e duração. Os modelos são todos referentes a corridas no modo *quickrace*, escolhida por ser a mais simples dentre as que permitem corridas com múltiplos carros.

Toda a manipulação de arquivos nesse formato é feita através de bibliotecas usadas pelo próprio TORCS para o mesmo propósito de gerar, alterar e ler dados `XML`. Assim, utiliza-se código maduro para realizar essas operações, código este que necessariamente já está presente na máquina que contém uma instalação do servidor, necessária para a utilização da TTI.

3.4 Simulação

A coleção de controladores usada pela TTI é formada por referências para objetos da classe abstrata *TTIWrapperDriver*. Esta é uma subclasse da *WrapperBaseDriver* fornecida dentro do pacote de controlador simples no modo cliente-servidor disponibilizado (Figura 2.3). As modificações foram feitas no sentido de possibilitar ao controlador o registro, durante uma corrida, de dados relevantes para análise posterior, sem precisar fazê-lo no código do próprio controlador, isso fica a cargo da *TTIWrapperDriver*. Manter o último conjunto de dados recebidos dos sensores (abstração fornecida pela classe *CarState*, também incluída no cliente padrão), permite o acesso a informações úteis para a avaliação, como as dos sensores *damage*, referente aos pontos de dano do carro, *distRaced*, que fornece a distância total percorrida, e *racePos*, a posição no *ranking* da corrida.

Também é feito o registro em um vetor dos tempos para completar cada volta. O sensor *lastLapTime* fornece uma maneira simples de fazê-lo. Seu valor inicial é zero, sendo atualizado cada vez que o carro completa uma volta durante a execução. Assim, pode-se monitorá-lo e adicionar seu valor ao vetor de tempos cada vez que é alterado, i.e., cada vez que uma nova volta é completada. Porque a simulação somente é encerrada alguns tiques após o final da última volta, o último valor lido em *lastLapTime* corresponde à duração da última volta, de forma que apenas essa checagem já é suficiente para se capturar todos os tempos em uma corrida. Entretanto, se o controlador não completar todas as voltas, seja por ter sido retirado por exceder o limite de tempo, de danos ou se seu combustível for esgotado, esse vetor não vai conter o número total de voltas solicitadas. É importante que o usuário esteja ciente desse fato e lide com ele, caso ocorra. Esse detalhe é explicitado na documentação da classe *TTI*.

Também foi implementada uma checagem do posicionamento do carro em relação aos limites da pista, considerando se este está ou não fora dela a cada instante. Caso esteja, uma variável, inicializada em zero, é incrementada. Esta informação é oferecida pelo sensor *trackPos*. É preciso verificar, apenas, se o seu valor está dentro do intervalo $[-1, 1]$, pois este é normalizado de forma que valores -1 e 1 indicam os limites laterais do circuito. Esse é um dos parâmetros que podem ser usados para estimar a qualidade do controlador e por isso é registrado durante a execução e retornado junto aos resultados.

3.4.1 Interação com o servidor

A Figura 3.1 ilustra como a TTI interage com o algoritmo de treinamento e com o TORCS. Para invocar uma instância do servidor, é preciso executar um comando de terminal no seguinte formato:

```
$ torcs -r /caminho/absoluto/para/config-corrida.xml
```

Onde o arquivo `XML` fornecido determina as configurações da corrida que será realizada. A TTI o faz através da criação de um processo novo. O caminho para o arquivo de configuração é determinado pela classe *RaceSet*, que gerencia o conjunto de corridas e determina qual deve ser a próxima na fila de simulação.

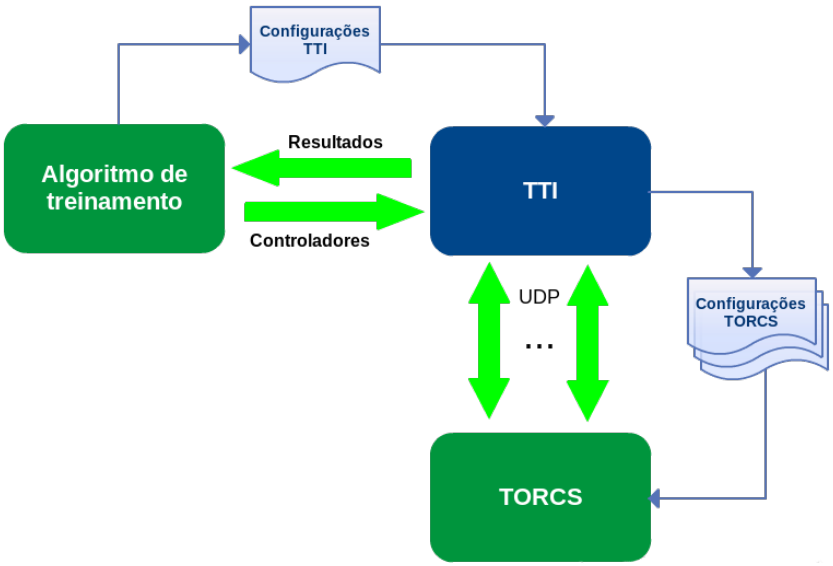

Figura 3.1: Interações da TTI com o simulador TORCS e com o programa que a utiliza

3.4.2 Execução dos clientes

Os controladores no modo cliente-servidor são executados em *threads* individuais, de forma que cada uma estabelece a sua própria conexão, de acordo com a porta e o nome de hospedeiro fornecidos. Trabalha-se com *localhost* por padrão, porém é possível estabelecer conexões remotas, se assim for desejado. O funcionamento é realizado através de uma versão adaptada do código no *client.cpp* disponibilizado pelo SCR. Essa recebe como parâmetro a referência para o controlador correspondente à própria *thread*.

3.4.3 Interface gráfica

Em algumas situações é possível que os usuários desejem a possibilidade de observação do comportamento dos controladores na simulação, como em fases iniciais do desenvolvimento onde se procura alterar o algoritmo para remediar atitudes inadequadas tomadas pelo controlador. Por esse motivo, está presente a opção de utilizar o servidor em modo de interface gráfica. Entretanto, nesse caso, não é fornecido um arquivo que informa as configurações da corrida a ser realizada, esta deve ser configurada de forma manual. O início das corridas também deve ser selecionado manualmente, exceto no caso de reinícios, que podem ser solicitados pelos próprios controladores a partir de um atuador especial (*meta*). Nesse caso o *software* desenvolvido gerencia, assim, apenas a execução dos clientes.

Quando se utiliza o modo de interface gráfica, não é criado pelo TORCS um arquivo contendo informações sobre a corrida ao final da sua execução. Há a vantagem de se poder observar o comportamento dos controladores em testes, o registro de dados de resultado pode ser feito de forma manual, anotando-se observações sobre o comportamento do controlador nas situações observadas. A TTI retorna as informações normalmente, e estas podem ser usadas de forma complementar junto às anotações.

3.4.4 Dados retornados

Uma execução em linha de comando do TORCS fornece resultados de duas formas. A primeira é a impressão de tempos no próprio terminal (ver Seção 2.1.5), onde a cada volta finalizada pelo líder, é impresso o tempo cumulativo até o momento. Esse resultado pode ser útil e informativo em simulações feitas com um controlador individualmente, mas em outras situações não se sabe quem está em primeiro lugar, situação que pode mudar ao longo da corrida. Não só não é informado a qual controlador aqueles resultados se referem, não há garantia que todos os tempos informados sejam relativos ao desempenho do mesmo piloto.

Algumas informações referentes à corrida executada são registradas em um arquivo `XML`, criado pelo TORCS ao final da execução. Esse contém dados mais completos e informativos, explicitando a qual se referem, mas ainda são limitados. Embora sempre sejam inclusos os resultados do líder, em corridas com múltiplos carros nem sempre isso é verdade para todos eles. Os dados informados incluem a posição no *ranking*, o tempo médio para completar uma volta e os pontos de dano ao final. Todos esses resultados e mais são obtidos pela TTI, incluindo todos os pilotos participantes, e retornados em uma estrutura de dados após o final da execução, fornecendo acesso mais eficiente do que leitura e *parsing* do arquivo gerado a cada simulação.

Usando-se a TTI, ao final da execução de cada corrida é retornado um conjunto de resultados. Para cada piloto, são informados:

- O tempo total da corrida.
- Uma lista de tempos correspondentes a cada volta.
- Média de tempo por volta.
- Pontos de dano do veículo.
- Quantidade de tiques nos quais o veículo estava fora da pista.

Retornar o tempo é crucial para a procura de pilotos mais velozes. Enquanto o tempo total oferece uma informação mais direta e geral, ter acesso a todos os tempos permite observar o padrão de comportamento - é possível que uma volta tenha sido consideravelmente mais demorada do que as demais por causa de alguma complicação que não aconteceu nas outras.

A média de tempo é a princípio uma medida mais adequada do que o tempo total pois o carro pode não ter completado todas as voltas, tendo atingido um dos critérios do simulador de retirada da corrida, como a ultrapassagem de um número de pontos de dano limite. Assim, é possível que um dos controladores tenha um menor tempo total mas uma média de tempos maior do que um outro, por ter completado menos voltas. Os pontos de dano e o quanto o carro sai da pista são critérios adicionais usados para a avaliação em [4], e provêem mais informação sobre o comportamento do carro, além de contarem pontos no SCR, sendo portanto dados importantes para a avaliação.

Todos esses valores são computados a partir dos dados informados pelos sensores ao longo da corrida, sem tomar proveito dos resultados fornecidos pelo TORCS. Essa escolha foi feita em virtude dos resultados impressos pelo simulador serem bastante limitados, especialmente no caso de corridas com mais de um piloto.

Os objetos dos controladores ainda guardam o último conjunto de dados de sensores recebidos, o que permite a extração de informações como a posição no *ranking* (sensor *racePos*).

3.5 Implementação

Por causa da necessidade de lidar com arquivos `XML` para atender os objetivos traçados da ferramenta desenvolvida, e porque o TORCS já continha ferramentas para fazê-lo, procurou-se uma forma de aproveitá-las na TTI. O simulador oferece uma forma de desenvolver novos módulos de controladores que são compilados em bibliotecas compartilhadas, usando um *Makefile* especial, e os módulos podem fazer uso de métodos do próprio TORCS em seu código. Dessa forma, percebeu-se a oportunidade de adaptar o *Makefile* para a compilação da TTI, incluindo os módulos de leitura e escrita de arquivos `XML`. Indica-se que o código-fonte da TTI seja copiado para um diretório próprio junto a outros arquivos-fonte do jogo, como é feito para os módulos de controladores, e a sua compilação é feita com um simples comando `make install`.

Tanto o TORCS quanto os módulos são desenvolvidos em C++, de forma que foi natural a decisão de implementar a ferramenta nessa linguagem. Essa também é amplamente usada no desenvolvimento de pilotos para o SCR, cujo cliente é disponibilizado em C++, Java e Python. Porque a TTI interage com objetos, é necessário que os pilotos testados com ela sejam desenvolvidos na mesma linguagem.

O código da TTI está documentado no padrão *Doxygen*, incluindo exemplos de instalação, uso e resolução de problemas comuns. Este está disponível publicamente na plataforma *GitHub*[2] e, assim como o TORCS e o código do SCR, usa a licença de *Software* Livre GPL versão 2.0. Dessa forma, a ferramenta está disponível para uso livre da comunidade e para contribuições ao seu código, possivelmente feitas por indivíduos com experiência no desenvolvimento de pilotos que podem enriquecê-la, adicionando às suas funcionalidades.

3.6 Como utilizar

Para fazer uso da ferramenta desenvolvida, um programa deve criar um vetor de controladores e um arquivo de configurações, e então usá-los para construir uma instância da TTI. Após esse passo, cada chamada do método *race()* executa a simulação da próxima corrida, retornando um vetor de resultados, onde cada elemento corresponde ao controlador de mesma posição. O código a seguir exemplifica esse funcionamento.

A coleção de resultados obtida a cada corrida pode então ser usada para se fazer uma estimação da habilidade de cada piloto em teste. A alteração de valores dos atributos de objetos contidos no vetor *conjunto* e subsequente execução de uma nova corrida retornará resultados referentes a essas novas , pois os objetos são fornecidos à TTI por referência. Também é possível fazer trocas dos objetos apontados por essas referências, obtendo então resultados da simulação nessa nova ordem.

[2] https://github.com/clarac/tti

```
#include "TTI.h"
#include "TTITestDriver.h"

int main(){

    // criando objeto
    TTITestDriver *piloto = new TTITestDriver();

    // vetor de controladores
    std::vector<TTIWrapperDriver *> conjunto;

    // usando apenas um como exemplo
    conjunto.push_back(piloto);

    // criando o objeto TTI computa configuracoes iniciais
    TTI tti(conjunto, "/caminho/para/configuracoes.xml");

    // obter resultados da simulacao com uma simples chamada de metodo
    // raceData eh uma estrutura que guarda os resultados retornados
    std::vector<raceData> resultados = tti.race();

    return 0;
}
```

Capítulo 4

Testes

Procura-se, aqui, verificar a influência dos fatores abordados pela interface desenvolvida e evidenciar a sua eficácia em oferecer vantagens ao processo de testes, e, consequentemente, de desenvolvimento de IAs destinadas ao controle de veículos no TORCS. Para se comparar as diferenças envolvidas em trabalhar em otimizar esses parâmetros usando-se ou não a TTI, foi realizado um conjunto de testes abordando os mesmos problemas de três diferentes formas.

A primeira, forma mais imediata, é a manual. A segunda é uma abordagem adaptada da observada em [4], onde em uma mesma corrida periodicamente muda-se o algoritmo em controle do veículo, registrando os resultados do anterior e, assim, testando múltiplos pilotos em uma única execução. Por fim, faz-se o mesmo procedimento usando a *TORCS Training Interface*, e analisam-se as vantagens e desvantagens de cada forma em termos de tempo de realização, resultados, esforço envolvido e flexibilidade.

Para que se possa avaliar a TTI, são necessários:

- um piloto a ser testado;
- configurações específicas dos testes;
- uma métrica de desempenho.

A forma como esses fatores foram determinados é detalhada neste capítulo, seguido-se uma discussão a respeito dos resultados observados.

4.1 Controlador usado

Trabalhou-se com um algoritmo de controlador adaptado daquele disponibilizado pelos mantenedores do TORCS (*SimpleDriver*). Este controlador é composto por métodos que determinam os valores dos atuadores do veículo, a cada instante, a partir dos valores dos sensores que recebem. Para tanto, um conjunto de constantes são utilizadas. Neste piloto, o cálculo do valor do atuador correspondente à aceleração, um valor entre 0 e 1, é feito a partir da diferença entre uma velocidade alvo e a velocidade atual. A velocidade alvo é estimada com base nas características da curva. Caso o trecho atual seja julgado como reta ou curva ampla, usa-se a velocidade alvo máxima, ou uma fração desta, caso contrário.

Para estimar se o veículo se encontra em uma reta ou em curva aberta o suficiente para se procurar estar na velocidade máxima, observa-se o valor do sensor *track* paralelo ao eixo de direção do carro, que informa a distância em linha reta da frente do carro até o ponto onde essa reta encontra um dos limites laterais da pista. Quanto maior este valor é, mais amplo se considera o trecho onde se encontra o veículo. Assim, faz-se uso de uma distância limite, a partir da qual sempre se usa a velocidade máxima como alvo.

Os parâmetros de velocidade máxima e de distância máxima têm seus valores iniciais estabelecidos de forma empírica. Procurar um conjunto de valores ótimos para estes e outros tantos parâmetros do controlador que são usados para o cálculo da direção, freio e outros atuadores, como já discutido, é complicado não só por seus extensos conjuntos de valores, mas por conta da enorme quantidade de situações onde este controlador pode se encontrar, considerando possíveis configurações do carro, circuitos e oponentes.

4.2 Descrição dos testes

Os três testes fazem uma varredura dos valores de duas das constantes do controlador, *maxSpeed* e *maxSpeedDist*, respectivamente a velocidade máxima do carro e a distância mínima da frente do carro em linha reta até a borda da pista para que se considere o trecho como uma reta. Como outros fatores, seus valores ótimos dependem das propriedades da pista em termos de formato e de relevo. Velocidades mais altas seriam mais adequadas para pistas largas com curvas amplas, enquanto pistas estreitas com curvas fecharas exigiriam estratégias bem mais conservadoras.

A velocidade máxima por si só é aquela que o carro procura manter quando percebe estar em uma reta. Esta não se refere aos limites de velicidade do carro, além do fato que não pode sair desse intervalo. Assim, a sua aceleração ou freio é determinada em função da diferença entre a velocidade alvo e a velocidade atual. Valores muito baixos fazem com que o carro tenha desempenho pobre, bastante inferior ao seu potencial, enquanto valores muito altos podem levar a derrapagens, giros do veículo e saídas dos limites da pista.

A distância de velocidade máxima trata do mínimo valor do sensor *track* paralelo ao eixo longitudinal do carro para que se considere o trecho à frente como reta ou curva aberta o suficiente para manter-se em velocidade máxima. De forma similar, valores muito altos podem significar subdesempenho por lentidão e valores muito baixos podem levar a instabilidades. Seu valor, em metros, não deve ultrapassar 200, máxima distância informada pelo sensor *track*.

Os dois fatores estão intimamente ligados e influenciam um ao outro - uma velocidade máxima mais alta pode exigir que a distância mínima seja maior para o alcance do mesmo desempenho. Aqui, modificam-se as duas em intervalos fixos a partir dos seus valores iniciais. São usados dois circuitos com características diferentes e mede-se o tempo para completar duas voltas como medida de comparação das combinações.

Os valores de velocidade máxima e distância de velocidade máxima são testados em intervalos relativos aos seus valores iniciais, 150 e 70, respectivamente em quilômetros por hora e em metros. Então, são feitos testes para as combinações dos dois com variação entre -15% e $+15\%$, em intervalos de 5%. Dessa forma, velocidade máxima é testada para os valores $\{127.5,135,142.5,150,157.5,165,172.5\}$ e a distância é testada com os valores $\{59.5,63,66.5,70,73.5,77,80.5\}$. Assim, para cada caso, são feitos 49 testes por pista, um para cada combinação possível, totalizando 98 testes. Escolheu-se testes neste formato por

fazerem uma varredura local das duas constantes em uma quantidade viável de se fazer manualmente, que é uma das formas de testes consideradas. Assim também é ilustrado o caso de testes em múltiplas pistas, levando em consideração a forma com que isso pode ser feito em cada método.

Foram usadas as pistas Michigan, da categoria *oval*, e G-track-3, da categoria *road* (ver Anexo A). A primeira é uma pista veloz, larga e com curvas abertas. A segunda é composta tanto por curvas fechadas quanto por trechos rápidos. O uso das duas oferece maior diversidade nos testes, de forma a avaliar melhor o desempenho de cada controlador. Cada teste foi feito em duas voltas, um intervalo de simulação condizente com aqueles observados na literatura, com o controlador sozinho na pista. Por este motivo, não há preocupação com ordenação.

4.2.1 Teste manual

Testes manuais foram feitos de forma simples, a partir apenas dos códigos já disponíveis de cliente e de servidor. Nesse caso, primeiramente são configurados dois arquivos `XML`, cada um referente a uma corrida. Para cada iteração, o valor é modificado no código, que é recompilado e executado. Também a cada etapa o simulador TORCS é iniciado manualmente em interface de texto.

Uma planilha é preparada previamente com os valores dos parâmetros e, a cada iteração, o valor do tempo para percorrer as das duas voltas, impresso no terminal, é usado para preencher o campo adequado. Ao final, soma-se o tempo total de cada uma das 49 configurações para as duas pistas e ordenam-se os registros para, assim, determinar o melhor conjunto observado no experimento.

A medição do tempo gasto nesse processo foi feito com auxílio de um cronômetro, iniciado imediatamente antes do início da execução e parado assim que o último valor foi inserido na planilha. Dessa forma, não se inclui no tempo registrado preparações iniciais ou a posterior ordenação dos dados. Os testes com as duas pistas foram feitos de forma sequencial, sem interrupções para evitar distorções nos resultados, e intercalada entre uma pista e outra, para que se compilasse apenas uma vez cada configuração.

4.2.2 Teste *online*

Este teste é uma adaptação da estratégia de treino observada em [4], um trabalho de desenvolvimento de pilotos usando neuroevolução feito pelos próprios organizadores do SCR. Neste artigo, o controlador em teste tem posse do carro durante um intervalo fixo de tempo. Neste caso, para melhor possibilidade de comparação com os outros testes, o intervalo foi definido como duas voltas. Assim, a cada duas voltas, altera-se os parâmetros escolhidos para os próximos valores e se registra o tempo. O simulador é executado apenas uma vez por pista, numa corrida de 98 voltas (2×49).

Para tanto, foi necessário fazer modificações no código do cliente de forma que, a cada duas voltas, os valores dos parâmetros fossem alterados conforme determinado e o tempo para percorrer as duas últimas voltas fosse impresso na tela. Ao final da execução, os valores impressos no terminal são inseridos em uma planilha eletrônica, onde se somam os resultados das duas pistas.

O tempo gasto foi medido no próprio código. Para cada uma das duas execuções, o controlador registra o tempo em seu método *init* e também no método *shutdown*, chamados respectivamente no início e no final da simulação.

4.2.3 Teste com a TTI

Para uso da TTI, criou-se inicialmente um `XML` de configurações contendo os dois circuitos escolhidos, ambos com duas voltas, de acordo com o formato de testes definido. Então, um segundo programa foi desenvolvido, contendo dois laços aninhados que iteram sobre os valores estabelecidos de *maxSpeed* e *maxSpeedDistance*, chamando a simulação para cada uma das duas pistas e guardando a menor soma. Todos os valores são impressos e, ao final, é informado o menor tempo e a configuração que o apresentou.

O tempo, nesse caso, foi registrado antes da criação do objeto da TTI e após o término do laço de repetição.

4.3 Resultados dos testes

Cada método utilizado apresentou características que são discutidas nessa seção. Nota-se que os testes realizados, compostos por uma varredura de intervalo em duas dimensões, funcionam de forma bastante diferente de estratégias de aprendizado de máquina, e, assim, podem não refletir precisamente as relações entre as diferentes abordagens nesses casos. Entretanto, estes incluem a realização de vários testes em diferentes indivíduos, característica de tais abordagens, e são, dessa forma, válidos a título de avaliação comparativa da ferramenta proposta.

4.3.1 Tempo gasto

A duração medida de cada teste está ilustrada na Figura 4.1. O teste *online* apresentou o resultado mais veloz, com um total de 1536 segundos (25min36s), dos quais 466 para a simulação na pista Michigan e 1070 para a G-Track-3. Não se levou em consideração o tempo gasto em se iniciar a execução de cada instância ou em se guardar os resultados em planilha eletrônica, como foi o caso nos testes manuais. Nesses, foram gastos 2268 segundos (37min48s) do início ao final dos testes.

Os testes com a TTI apresentaram custos próximos aos do teste *online*, totalizando 1560 segundos (26min), um aumento de apenas 1.6%. Em comparação com o processo manual, houve redução de 31.2% no tempo total.

4.3.2 Esforços envolvidos

Embora os testes manuais exijam perto de nenhuma preparação inicial, estes envolveram extenso trabalho humano ao longo do processo. Para cada uma das 98 corridas, precisa-se executar os programas cliente e servidor, além de copiar os resultados para um local apropriado. A modificação dos valores também foi manual, levando a 49 compilações no total.

O teste *online* apresentou um meio termo, foram executadas apenas duas corridas, mas foi necessário alterar o código-fonte do controlador de forma significativa, além de testá-lo.

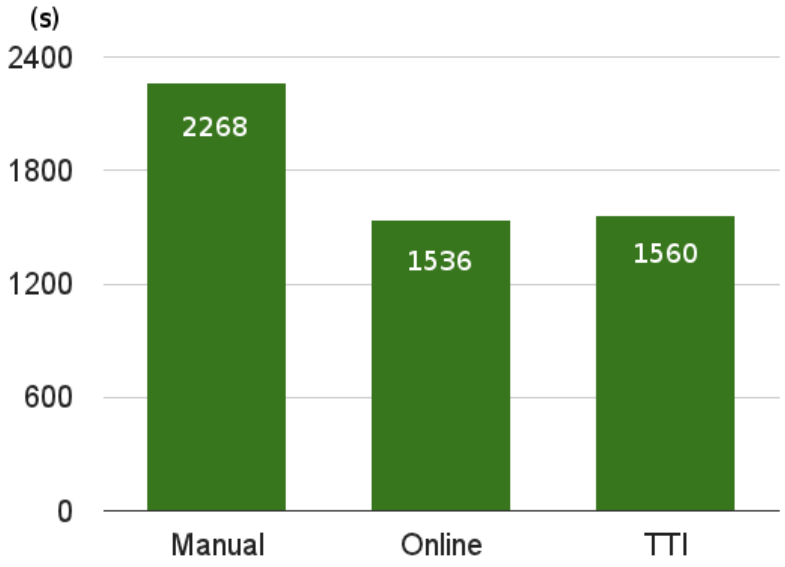

Figura 4.1: Tempo gasto por cada forma de testes (em segundos).

Observa-se que a classe do controlador não é local adequado para esse código adicional, relacionado com a execução dos testes mas não com o funcionamento do controlador em si, e que o uso do controlador em simulações após os testes exigiria que essas modificações fossem desfeitas, adicionando ao esforço de se utilizar essa abordagem.

A TTI mostrou-se a mais prática das formas usadas. A configuração de `XML` é bastante rápida de se fazer, precisando apenas da definição das duas corridas, assim como o desenvolvimento de um programa que solicita as execuções de corridas e guarda o melhor valor. Diferentemente das outras duas abordagens, o uso da TTI necessitou de uma única execução, enquanto a manual envolveu 196 execuções (programas cliente e servidor para cada uma das 98 corridas) e para a *online* foram feitas 4 (programas cliente e servidor para as 2 corridas).

4.3.3 Tempos retornados

Tanto os testes manuais quanto os com a TTI retornaram os mesmos resultados em termos de tempos de corrida para cada configuração, já que em se tratando de simulação, foi feito o mesmo procedimento que é determinístico.

O teste em *online*, entretanto, por consistir de uma única corrida, mostrou resultados diferentes. Com exceção das duas primeiras voltas, os tempos retornados apresentaram entre três e cinco segundos a menos do que os outros, resultado que é explicado pelo fato de que nesse caso, no intervalo de teste o carro tem uma velocidade inicial alta e tem posição e orientação na pista resultante das voltas anteriores.

4.4 Discussão

Os testes manuais mostraram-se mais limitados em relação aos outros dois em termos de esforço envolvido e tempo gasto no processo de testes. Isso é extremamente significante,

já que a exigência de um indivíduo trabalhando nos testes constantemente é cansativo e usa de tempo que poderia ser gasto de forma mais produtiva, além de ser bastante suscetível a erros humanos.

Já a estratégia *online*, embora tenha resultado em um tempo comparável com a TTI, apresenta problemas nos outros dois critérios avaliados, a flexibilidade e o esforço humano exigido. O esforço nesse caso não foi tão extenso, mas pode chegar a ser dependendo do algoritmo utilizado. Por exemplo, um teste que é realizado repetidamente para diferentes grupos de controladores é similar ao manual, já que novas instâncias precisam ser executadas para cada grupo ou pista diferente. Se a cada vez que o grupo em trabalho for ser avaliado, for necessário chamar uma nova instância de servidor, precisar-se-ia, então, de trabalho de supervisão realizado por uma pessoa. Fazer uso dessa estratégia para testes de diferentes classes ainda exigiria que o código adicional fosse incluído em todas elas, aumentando o esforço necessário.

A diferença de tempo entre o uso da TTI e o teste *online* pode ser explicado pela necessidade de se executar um novo processo do servidor a cada vez que uma nova corrida é iniciada, pois quando executado em linha de comando o TORCS termina o seu processo ao final da corrida solicitada. Caso fosse possível realizar múltiplas corridas em uma única execução, a TTI poderia tomar proveito desse fato e ter resultados melhores em termos de custo computacional, mas essa é uma limitação imposta pelo TORCS.

Em termos de resultados, a dependência das avaliações anteriores que existe no teste *online* pode levar a grandes distorções nos dados obtidos. Como exemplo, um controlador que inicia seu funcionamento em condições difíceis, como fora da pista ou andando na contra-mão, precisa de um esforço significativo para voltar ao funcionamento correto, esforço esse que alteraria o seus resultados.

Esse tipo de problema acontece porque, nesse caso, cada indivíduo toma controle do veículo durante a corrida, na posição, orientação e velocidade deixada pelo controlador em teste anteriormente. Essa forma também leva a corridas longas, o dano sofrido pelo carro é acumulado e afeta o seu desempenho, além de poder atingir o limite estipulado, o que leva a encerrar a corrida prematuramente. Também há um limite de tempo mínimo para percorrer uma volta, que, embora extenso, esgota-se caso o carro fique preso, levando também à finalização da simulação. Em ambos esses casos, todos os indivíduos que seriam testados posteriormente deixam de ser, exigindo, no mínimo, que o TORCS seja executado novamente.

A TTI executa diversas simulações com diferentes configurações e é prática de se usar. É flexível em relação às outras formas testadas por conta das possibilidades de configuração e por conta de liberdade de alterá-la da forma que melhor atender as necessidades de um projeto específico com um mínimo esforço adicional. Não há um limite definido sobre quantas corridas podem ser especificadas, é preciso apenas adicionar uma nova entrada nas configurações da TTI, em contraste com a criação e manutenção de um arquivo para cada corrida que precisa ser realizada manualmente quando não há uma ferramenta que automatiza essas tarefas. Após criado o arquivo de configuração e instanciado um objeto da classe *TTI*, o teste de um conjunto de controladores é feito com uma única chamada de método, fornecendo uma abstração sobre todo o processo de simulação.

A ferramenta atende aos seus propósitos de oferecer funcionalidades úteis ao desenvolvimento de controladores para o SCR, permitindo testes de forma mais eficiente, reduzindo as limitações presentes nos outros métodos observados e provendo dados detalha-

dos e úteis para a avaliação de controladores. Estima-se que o tempo e esforço gasto para instalá-la e começar a utilizá-la para treinamentos, conforme o crescimento da quantidade de simulações realizadas, rapidamente torne-se mais vantajoso do que o uso de outras abordagens.

4.5 Limitações

A TTI está atualmente limitada a um subconjunto de projetos desenvolvidos em C ou C++ em ambientes Linux. Esse é um problema relevante já que o simulador com o qual trabalha está disponível para diversas plataformas, incluindo os sistemas operacionais Windows e MacOS. A arquitetura cliente-servidor é ainda mais flexível, considerando que a interação é feita somente por conexão UDP. Assim, pode-se usar qualquer linguagem com a capacidade de estabelecer essa conexão e se comunicar através dela.

Embora a linguagem C++ seja comumente utilizada em tais projetos e embora seja a mesma linguagem na qual o próprio simulador foi desenvolvido, vê-se que essa limitação é bastante restritiva. Aproximadamente 50% dos controladores submetidos para a competição SCR são desenvolvidos em C++, com o restante em Java, mas com a recente disponibilização de software cliente em Python já surgem projetos realizados para o SCR nessa linguagem. Observa-se, entretanto, que estas outras linguagens não permitiriam a passagem de objetos por referência, exigindo que se adote outras estratégias para que seja possível fazer alterações sobre as instâncias de pilotos entre simulações.

A segunda principal limitação dessa interface refere-se ao próprio encapsulamento do código modificado ao código do controlador e possíveis outros módulos adicionais específicos ao projeto. Observou-se que vários dos controladores participantes do campeonato SCR exibiram modificações relevantes ao funcionamento do controlador em outros arquivos, como no código específico que lida com a comunicação com o servidor. Embora seja possível fazer essas modificações diretamente com a TTI, caso essas façam parte do algoritmo de funcionamento, elas teriam de ser refeitas no cliente próprio para execução normal posterior.

4.6 Observações

Bernhard Wymann, principal mantenedor atual do TORCS, expressou opiniões positivas a respeito da TTI. A comunicação com ele é mostrada no Anexo C (em inglês). Ele agradeceu pela contribuição para o TORCS, leu o código e ofereceu sugestões, elogiando a iniciativa e o trabalho. Ele também sugeriu que a TTI fosse estendida afim de poder ser usada também para controladores que não funcionam na arquitetura cliente-servidor.

Foi escrito um artigo curto [2] a respeito da TTI, que foi aceito no SBGames de 2013.

Capítulo 5

Conclusão

A ineficiente maneira como controladores são testados e desenvolvidos no TORCS é um problema relevante por conta das limitações impostas sobre trabalhos de desenvolvimento de IAs para esse simulador. Por ter um papel de plataforma para *benchmark* de diferentes abordagens de IA, a existência de uma solução para teste mais eficientes é de grande valia. Estratégias atuais envolvem processos manuais, que são problemáticos não só pelo extenso trabalho envolvido, mas pela incidência de erros humanos e pela sua necessidade de operação manual.

A TTI, ou *TORCS Training Interface*, foi proposta para lidar com os problemas observados, auxiliando os processos que precisam fazer testes de simulação repetidamente. A TTI foi implementada como biblioteca compartilhada com o propósito de servir como uma camada de abstração sobre a interação entre cliente e servidor usada para testes com o simulador TORCS. Esta oferece funcionalidades facilmente usadas e bastante úteis, como o uso de diferentes circuitos e a terminação adiantada, ambos com potencial de adicionar ao processo em termos de qualidade ou de eficiência.

Adicionalmente, argumenta-se que o uso da TTI envolve consideravelmente menos esforço do que outras estratégias observadas, considerando-se modificações de código e intervenções manuais.

Fazer uso da TTI reduz a preocupação com a comunicação com o servidor, que antes precisaria ser cuidadosamente considerada e tratada no desenvolvimento de controladores. Nesse caso, os desenvolvedores têm a opção de incrementar o código de acordo com o que precisarem, mas destaca-se o fato de ser uma opção - usar a interface torna esse processo automatizado e secundário, permitindo com que se preocupe, nesses casos, com o foco real do trabalho, o projeto e desenvolvimento do controlador.

Além disso, a sua licença permite não só complementos específicos às necessidades observadas em um processo para uso próprio quanto ajustes compartilhados com a comunidade, promovendo ainda mais o reuso de código de testes que, de outra forma, precisaria ser desenvolvido redundantemente para cada projeto.

Os testes comparativos realizados entre diferentes formas de avaliação interativa de pilotos mostraram que o uso da TTI de fato é vantajosa. Em comparação com testes manuais, reduziu-se o tempo do processo, além do trabalho envolvido ativando testes repetidamente. Também se reduziu a probabilidade de erros humanos, já que nesse caso as chamadas são automatizadas.

Em comparação com o teste *online*, embora tenha apresentado um tempo ligeiramente mais lento, a TTI ainda mostra vantagens em termos de possibilidades e de variedade, além de comparabilidade de resultados, já que os testes de cada controlador são realizados em situações idênticas.

Promove-se o reuso de código, evitando o desenvolvimento de *scripts* para execução automatizada de testes. O usuário tem a liberdade de modificar a ferramenta desenvolvida para adequá-la aos seus propósitos, mas reduz-se em muito a necessidade de se preocupar com a forma de realização dos testes. A TTI funciona como uma camada de abstração que gerencia as simulações e lida com vários dos desafios discutidos, de forma que o tempo dos desenvolvedores possa ser gasto de maneira mais produtiva no piloto em si.

A TTI é compatível com o software cliente original, no sentido de que controladores treinados com ela podem ser usados e devem ter desempenho tão bom quando quando acopladas a ele, necessitando para tanto apenas que seja alterada a classe da qual este herda.

É importante notar que diferentes algoritmos podem favorecer um funcionamento ou outro. A TTI não necessariamente será a melhor opção para todos os casos, mas, quando não for, poder-se-ia procurar incrementá-la afim de atender esta demanda específica.

5.1 Trabalhos futuros

Claramente, o software desenvolvido tem vários pontos passíveis a melhorias. A princípio, trabalhar nas limitações mencionadas seria de grande valia, pois implicaria na ampliação de possíveis aplicações da ferramenta. Para tanto, seria preciso adaptar seu código para outros sistemas, adequando as chamadas de sistema realizadas, além de desenvolver seus algoritmos em outras linguagens como Java e Python, por serem usadas com frequência no desenvolvimento de pilotos para o SCR. Uma alternativa para esta segunda atividade seria a criação de uma interface universal, como a usada na arquitetura cliente-servidor, para torná-la tão flexível quanto em termos de linguagem utilizada.

Dentre outros possíveis aprimoramentos estão a adição de novas funcionalidades, opções e configurações. Embora altamente possível de ser realizada pelo programador, por causa da forma como foi implementada a TTI, não foi criado um mecanismo específico para organizar a ordem de largada dos controladores em teste. Esta ordenação é importante para adicionar informação e variedade aos testes, e poderia ser incorporada como uma opção da própria ferramenta.

Uma outra funcionalidade importante seria a possibilidade de avaliação comparativa com um piloto modelo, seja em termos de tempo ou mais detalhadamente, em termos das decisões tomadas para uma sequência de entradas, ou comportamento desejado bem definido, como trajeto ótimo em cada pista. Essa poderia vir junto a uma saída adicional, além dos dados de tempo e dano, um traçado do percurso do veículo ao longo da corrida. Essa opção daria aos desenvolvedores informações relevantes sobre o comportamento do controlador de forma clara, objetiva e mais eficiente do que a observação da corrida em interface gráfica.

A escolha de quais circuitos serão utilizados pode ser feita pelo usuário, mas, quando não for o caso, pode-se predeterminar um conjunto de circuitos de teste, observando os cuidados necessários discutidos em termos de variedade e entropia. Uma alternativa é, de porte de informações precalculadas das pistas do jogo, escolher um conjunto de

circuitos, de tal forma que a probabilidade de um deles ser escolhido seja proporcional à sua avaliação em termos de diversidade.

Também estimada como útil é a opção de definir os intervalos de simulação em tiques de jogo, além da possibilidade de selecionar quais resultados seriam retornados e procurar otimizações específicas para diferentes necessidades.

Finalmente, usuários de técnicas específicas de IA podem levantar recursos úteis especificamente para aquele tipo de abordagem e trabalhar para incorporá-los no código da ferramenta aqui desenvolvida. Esta tem bastante espaço e possibilidades de crescimento, além de poder ser discutida e incrementada de forma colaborativa por aqueles que estão mais intimamente ligados ao TORCS e ao campeonato de pilotos.

5.1.1 Comparação com modelos de referência

Uma funcionalidade potencialmente útil que poderia ser oferecida pela TTI é uma avaliação prévia, feita através da comparação do comportamento do controlador com algum modelo preexistente.

É possível estimar uma trajetória ótima para a pista, previamente, e utilizar como avaliação o quanto o controlador adere a ela. Essa trajetória costuma ser um compromisso entre aquela de menor comprimento e a que minimiza as curvaturas executadas, i.e., maximiza o raio das curvas do trajeto percorrido. O caminho de menor extensão é aquele que é percorrido mais rapidamente por um corpo de velocidade constante, enquanto o de menor curvatura é, de forma simplificada, o que permite ao carro maximizar sua velocidade média. Sabe-se que o tempo gasto para percorrer um caminho é inversamente proporcional à velocidade média e diretamente proporcional ao seu comprimento. Minimizar o tempo, dessa forma, pode ser entendido como maximizar a velocidade e minimizar o comprimento. Como os dois levam a caminhos diferentes (Figura 5.1), procura-se uma trajetória intermediária que ofereça a melhor combinação.

Existe uma grande quantidade de estudos nessa área, seus resultados são de interesse tanto para jogos quanto para aplicações reais de corrida. Em jogos comerciais, as IAs que controlam os adversários do jogador costumam ser programadas para seguir a trajetória ideal, que é traçada por especialistas e posteriormente incrementada através de testes dentro do próprio jogo. Esse é um processo custoso e demorado, mas existem abordagens que procuram realizá-lo de forma automatizada [5]. Uma comparação da trajetória do veículo com a ótima proporciona um bom complemento à medida de tempo e velocidade alcançada pelo controlador, já que provê informações simples mas extremamente relevantes, que descrevem bem o seu comportamento. Esse é um modelo simplista, o trajeto ótimo real leva em consideração as propriedades físicas do carro como seu alcance de aceleração e eficiência da frenagem, além da interação com oponentes em cada ponto, e.g. ultrapassagens. Ainda assim, pode ser muito útil como ferramenta, mesmo que complementar, para estimações de qualidade.

Uma outra forma simples de avaliação é a comparação dos resultados com os de um piloto já existente. Os próprios módulos incluídos na distribuição do simulador podem servir de base para esse tipo de métrica. Quando comparado com os controladores que participaram do campeonato de 2009, o melhor deles atingiu tempos competitivos e, em algumas situações, superiores [14]. Normalizar os valores ainda traz uma informação relevante, a de quão perto os indivíduos chegam, em termos de qualidade, dos melhores dentre

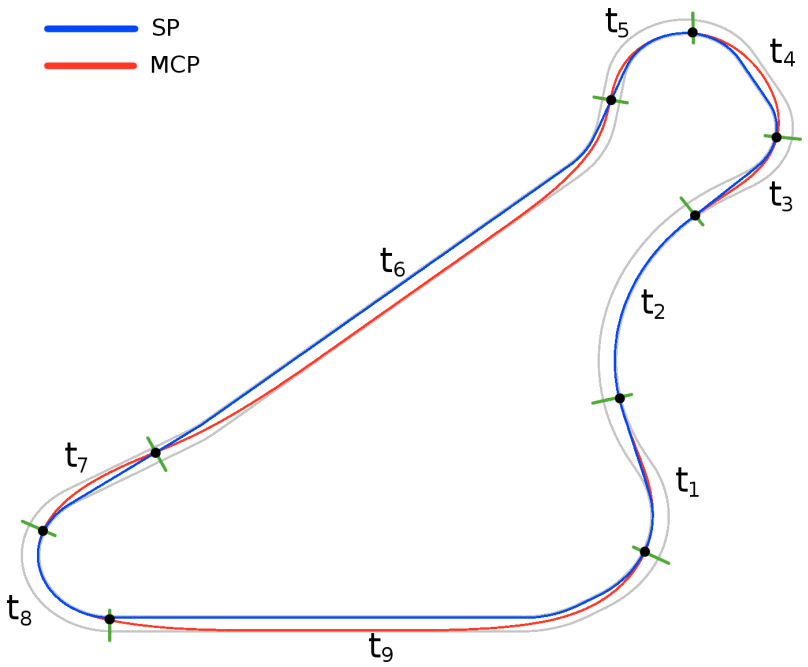

Figura 5.1: Ilustração das trajetórias calculadas: a de menor comprimento em azul (*shortest path*) e a de menor curvatura em vermelho (*minimum curvature path*) [5]

os já desenvolvidos. Valores acima de 100% indicariam uma sobresaliência, mostrando que o piloto em questão poderia ser considerado melhor do que aquele tomado como referência.

Essa estratégia pode considerar simplesmente o tempo resultante ou escolher uma forma mais sofisticada de análise e relacionamento entre as decisões tomadas pelos dois controladores. Esta última é especialmente útil quando se precisa avaliar comportamentos específicos complexos, como o de ultrapassagem [11]. É desejável que o controlador resultante consiga passar à frente de seus adversários, que o faça sem perder muito tempo, no ponto certo e que não perca muito em desempenho para fazê-lo. Assim como a escolha de impossibilitar o tanto quanto possível que os adversários o ultrapassem, que se trata de outro comportamento complexo de se definir ou avaliar, pode ser mais simples e mais rápido fazer uma comparação entre as decisões do indivíduo sob avaliação e as de um outro que seja considerado hábil nesse quesito.

Referências

[1] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993. 1

[2] Clara Caldeira, Claus Aranha, and Guilherme Ramos. Torcs training interface: An auxiliary api for developing torcs drivers. In *XII Simpósio Brasileiro de Jogos e Entretenimento Digital*, pages 13–16, 2013. 37

[3] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Car setup optimization. competition software manual. *Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy*, 2010. 8

[4] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Learning to drive in the open racing car simulator using online neuroevolution. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(3):176–190, 2010. 18, 28, 31, 33

[5] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli. Searching for the optimal racing line using genetic algorithms. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 388–394. IEEE, 2010. vii, 8, 40, 41

[6] Marcos Cintra. Os custos dos congestionamentos na capital paulista. *Conjuntura Econômica*, 61:30–33, 2008. 3

[7] Anca Dumitrache, Jacob Kooijman, and Andra Pascale. Implementing kamikaze driver. Technical report, Department of Computer Science, Vrije Universiteit Amsterdam, 2011. 18

[8] Roland Grabner, Elsbeth Stern, and Aljoscha Neubauer. Individual differences in chess expertise: A psychometric investigation. *Acta psychologica*, 124(3):398–420, 2007. 1

[9] Evert Haasdijk, Arif Atta-ul Qayyum, and Agoston Endre Eiben. Racing to improve on-line, on-board evolutionary robotics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 187–194. ACM, 2011. 24

[10] Markus Kemmerling and Mike Preuss. Automatic adaptation to generated content via car setup optimization in torcs. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 131–138, 2010. 8

[11] Daniele Loiacono. Learning, evolution and adaptation in racing games. In *Proceedings of the 9th conference on Computing Frontiers*, pages 277–284. ACM, 2012. vii, 3, 4, 7, 8, 21, 41

[12] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Automatic track generation for high-end racing games using evolutionary computation. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):245–259, 2011. vii, 8, 16, 17

[13] Daniele Loiacono, Luigi Cardamone, and Pier Luca. Lanzi. Simulated car racing championship: Competition software manual. *ArXiv e-prints*, April 2013. vii, 4, 7, 10, 11

[14] Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A Pelta, Martin V Butz, Thies D Lönneker, Luigi Cardamone, Diego Perez, Yago Sáez, et al. The 2009 simulated car racing championship. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):131–147, 2010. vii, 4, 12, 40

[15] Simon Lucas. Computational intelligence and games: Challenges and opportunities. *International Journal of Automation and Computing*, 5(1):45–57, 2008. 1, 2

[16] Simon Lucas. Computational intelligence and ai in games: a new ieee transactions. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):1–3, 2009. 2, 3

[17] Enrique Onieva, Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Overtaking opponents with blocking strategies using fuzzy logic. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 123–130. IEEE, 2010. 8

[18] Jan Quadflieg, Mike Preuss, Oliver Kramer, and Günter Rudolph. Learning the track and planning ahead in a car racing controller. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 395–402. IEEE, 2010. vii, 16, 17

[19] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, 1995. 4

[20] Gary Silberg and Richard Wallace. Self-driving cars: The next revolution. *KPMG and Center for Automotive Research*, pages 10–15, 2012. 3

[21] Richard Stallman and Joshua Gay. *Free software, free society: Selected essays of Richard M. Stallman*. CreateSpace, 2009. 19

[22] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006. 4

Anexo A

Relação de circuitos do *TORCS*

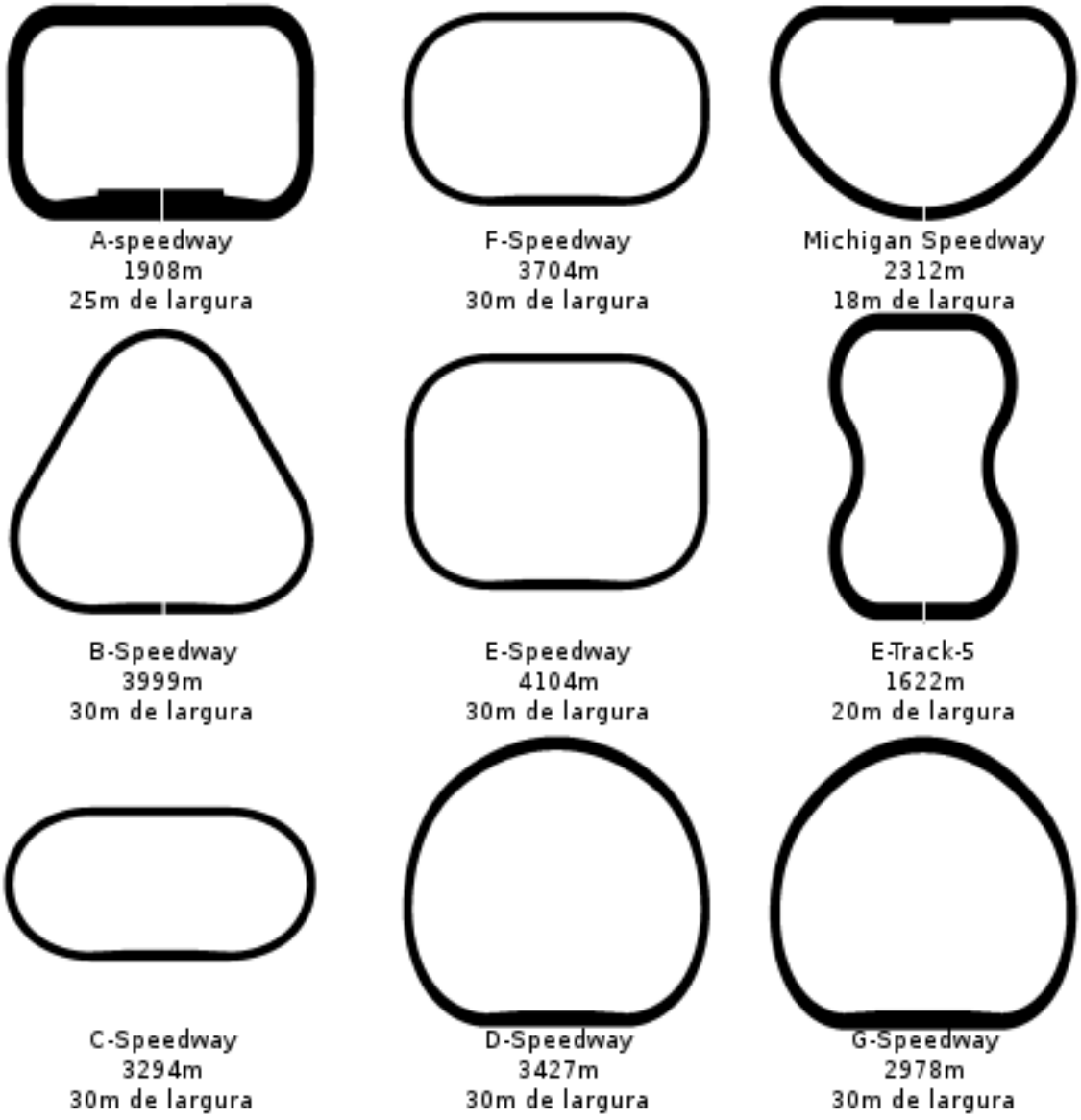

Figura A.1: Categoria *oval*

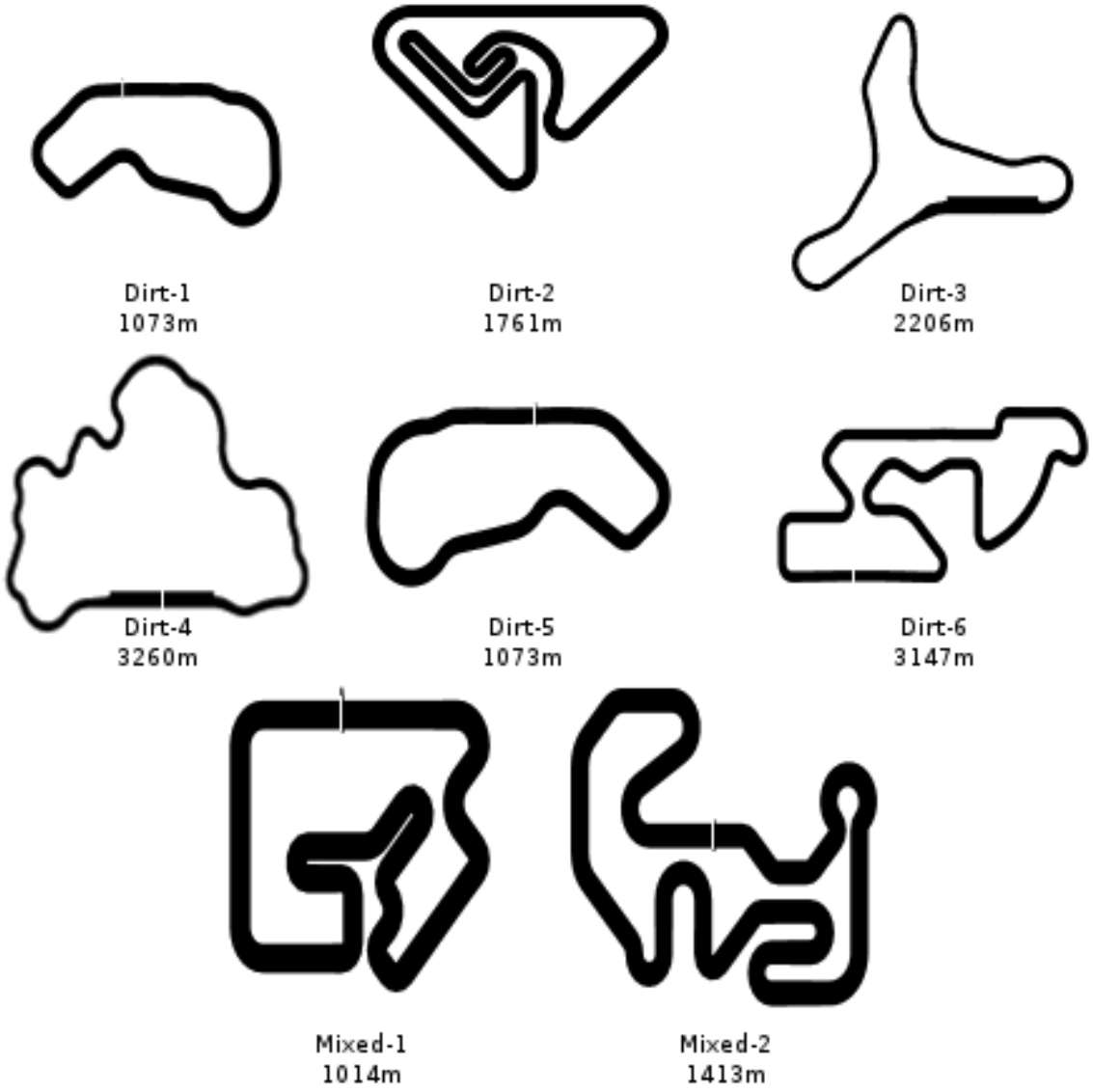

Figura A.2: Categoria *dirt*, todas têm 10m de largura

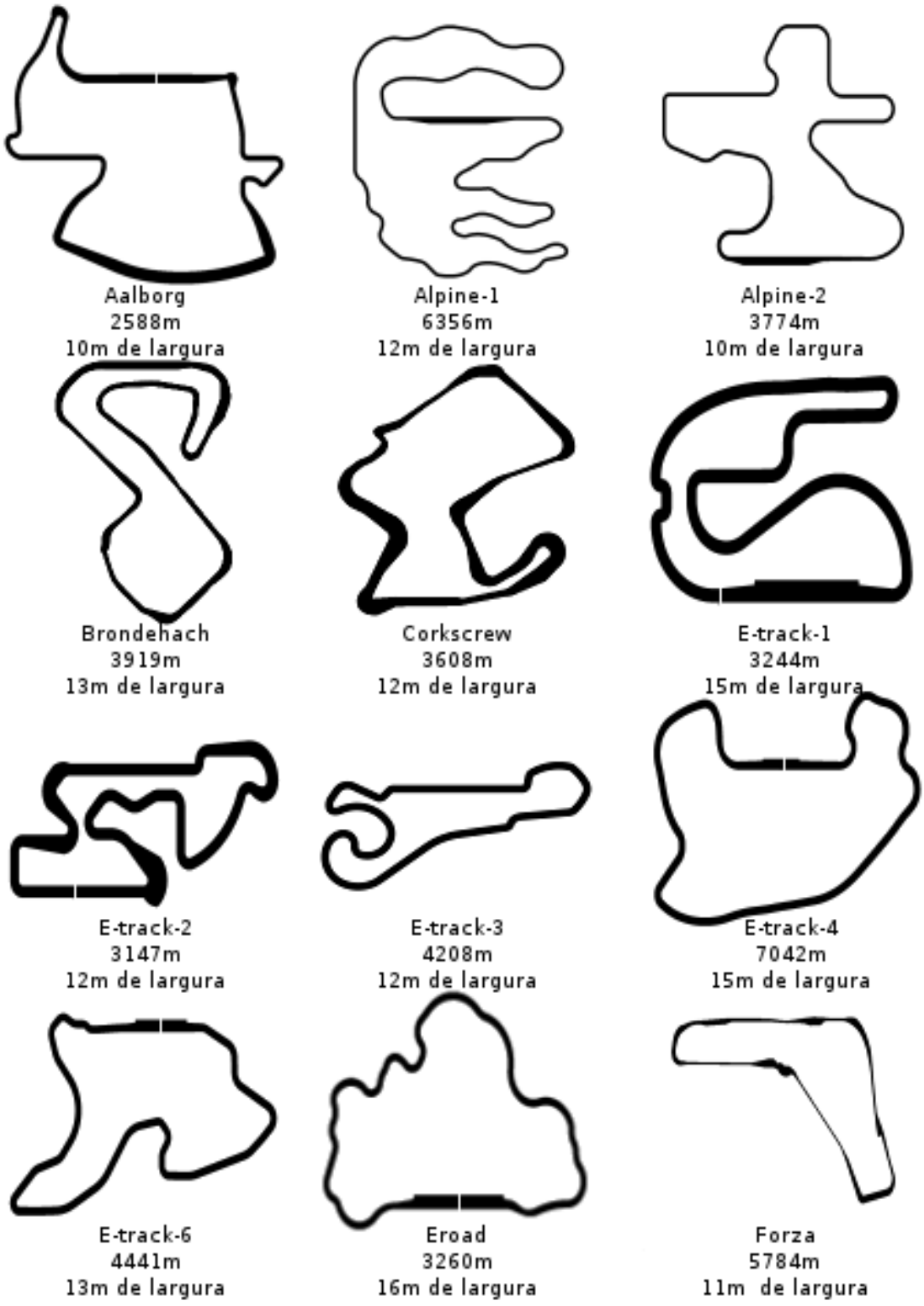

Figura A.3: Categoria *road* parte 1

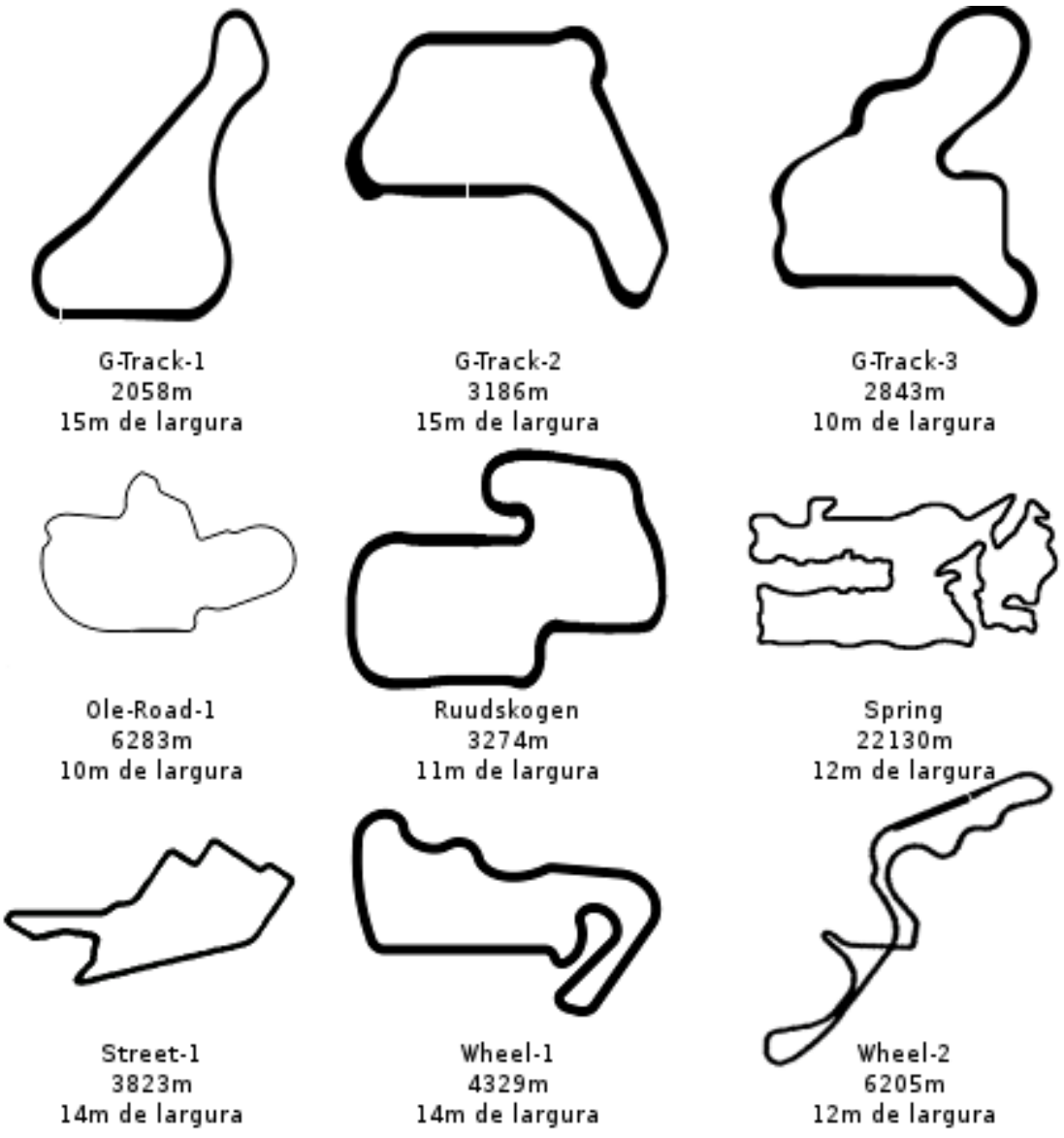

Figura A.4: Categoria *road* parte 2

Anexo B

Exemplo de arquivo de configuração

```
<params name="config">
  <section name="Header">
    <attstr name="id" val="teste"/>
    <attnum name="mindist" val="40"/>
    <attnum name="maxticks" val="3000"/>
  </section>

  <section name="Races">
    <section name="1">
      <attstr name="track" val="g-track-3"/>
      <attnum name="laps" val="2"/>
    </section>
    <section name="2">
      <attstr name="track" val="michigan"/>
      <attstr name="category" val="oval"/>
      <attnum name="laps" val="2"/>
    </section>
  </section>
</params>
```

Anexo C

E-mails trocados com Bernhard Wymann

Clara Caldeira wrote:

Dear Mr. Bernhard,

I, with the help of two professors, have been working on a tool for testing TORCS drivers in the client-server architecture for the last few months. It's called TTI, or TORCS Training Interface, and it is available for download here <https://github.com/clarac/tti>.

To put it shortly, it offers configurable testing and returns lap times, damage and game ticks spent outside of the track for a group of drivers. It's easy to use. I would like to ask if you could take a look at it and offer some feedback, if possible. Also, can I talk about it in the torcs-devel mailing list? TTI is still under development, we are looking into adding a few options which we think would be useful, but I believe that what is currently available could already be pretty helpful for people working on their own drivers. There are more details about it here <http://www.sbgames.org/sbgames2013/proceedings/comp/04-short-paper-comptrack.pdf>,

along with where we see it going in the future.

Sincerely,

Clara Caldeira

Bernhard Wymann wrote:

Hi Clara

Bernhard is just fine, the Mr. makes me feel older ;-)

I read the linked paper, it looks nice, if I got it right the SCR patch not required for TTI? I am in the process of finishing 1.3.5, after it is out (I hope sometime next week) I will have a look at your work (code), so please be patient.

Of course you can announce/discuss TTI on the TORCS mailing list(s) of your choice, that's what they are for (because of spam you have to subscribe first, the first post will go through moderation).

You should as well announce it here, because this is the place of the SCR related discussions usually: http://groups.google.com/__forum/#!forum/__racingcompetition

Thank you for contributing to TORCS.

Best regards

Bernhard

Bernhard Wymann wrote:

Hi Clara

I am looking at it right now, here some thoughts (without building/running it so far): - GPL v3, is legally ok, but because TORCS uses GPL v2 not backwards compatible (but maybe SCR is v3 and you had no other choice?). - The structure and technique is very similar to C++ unit testing suites (this is not bad nor good, just came into my mind...) - I like the architecture (pretty lose coupling with TORCS), coding style and documentation - I think it is a requirement to have the SCR server patch applied to TORCS, this should be clearly documented (where to get it, version, etc.)

Questions: 1. TTI.cpp, callRace, 52, what is the "std::thread torcs;" for (unused)? 2. TTI.cpp, callRace, 62, "std::thread *cl = new std::thread ..." -> memory leak, you join the thread below and the collection of pointers goes out of scope and is released, but not the objects pointed to (the threads), please check. Maybe using a thread pool would be handy. 3. Architecture, did you explore other integration possibilities, why did you choose this approach (e.g. you collect the data in the driver, you could have collected it as well in torcs or in a new torcs interface for that purpose)? Did you explore the possibility of creating TTI without the requirement of having SCR (e.g. what hooks would need to be in TORCS to make this possible, abstraction of communication channel (not UDP only, maybe shared memory/pipe when on same machine), ... lots of stuff to think about?

Very nice work:-)

Kind regards

Bernhard