Code Battle
Documentação e API

Fevereiro 2019

NUCC -FCUP

Núcleo de Ciência de Computadores da Faculdade de Ciências da Universidade do Porto

Documentação

1 Introdução

Nesta competição vamos utilizar uma versão modificada do TORCS (The Open Racing Car Simulator) que nos permite programar carros na linguagem de programação Java. Este manual descreve o software da competição e tudo o que precisas de saber para construir o teu carro.

1.1 O TORCS

O TORCS (The Open Racing Car Simulator) é um simulador de automobilismo open-source que nos permite programar os carros de forma a que estes corram autonomamente.
O TORCS é regularmente usado para competições e pesquisa no campo da Inteligência Computacional em Jogos.

1.2 A Competição

A competição consiste em duas etapas. Uma corrida de qualificação em que se decide, de acordo com as posições obtidas, a ordem pela qual os carros vão começar a corrida final e a corrida final, na qual, as três equipas a obterem uma posição no pódio ganham a competição.

AVISO

No desenvolvimento do bot, é necessário ter em mente que a corrida é em tempo real. Adequadamente, o servidor tem um tempo limite nas respostas do cliente: o bot deve executar uma ação (ou seja, retornar uma string de ação) em 10ms para manter a sincronização com o servidor. Se o bot for mais lento, provavelmente deixa de estar sincronizado com o servidor e por isso cabe a cada equipa descobrir como evitar que isso aconteça.

2 Personalização do carro

Para programarem o vosso próprio carro a interface **Controller** tem que ser implementada por uso dos seguintes métodos:

- **public float[] initAngles() –** este método é chamado antes do início da corrida e pode ser usado para definir uma configuração personalizada dos *sensores da pista*(ver secção **Sensores**): este método retorna um array com os 19 ângulos desejados para cada um dos 19 localizadores de alcance.
  Exemplo:

```
public float[] initAngles()     {

        float[] angles = new float[19];

        /* set angles as {-90,-75,-60,-45,-30,-20,-15,-10,-5,0,5,10,15,20,30,45,60,75,90} */
        for (int i=0; i<5; i++)
        {
                angles[i]=-90+i*15;
                angles[18-i]=90-i*15;
        }

        for (int i=5; i<9; i++)
        {
                        angles[i]=-20+(i-5)*5;
                        angles[18-i]=20-(i-5)*5;
        }
        angles[9]=0;
        return angles;
}
```

- **public Action control(SensorModel sensors) –** onde **sensors** representa o estado atual da corrida do ponto de vista do carro; este método retorna a medida tomada(ver secção 3).
  Exemplo:

```
public Action control(SensorModel sensors){
    if ( Math.abs(sensors.getAngleToTrackAxis()) > stuckAngle ) // check if car is currently stuck
    {
        stuck++; // update stuck counter
    }
    else
    {
        stuck = 0; // if not stuck reset stuck counter
    }

    if (stuck > stuckTime) // after car is stuck for a while apply recovering policy
    {
        /* set gear and sterring command assuming car is
         * pointing in a direction out of track */

        float steer = (float) (- sensors.getAngleToTrackAxis() / steerLock); // to bring car parallel to track axis
        int gear=-1; // gear R

        if (sensors.getAngleToTrackAxis()*sensors.getTrackPosition()>0) // if car is pointing in the correct direction revert gear and steer
        {
            gear = 1;
            steer = -steer;
        }

        clutch = clutching(sensors, clutch);

        Action action = new Action (); // build a CarControl variable and return it
        action.gear = gear;
        action.steering = steer;
        action.accelerate = 1.0;
        action.brake = 0;
        action.clutch = clutch;
        return action;
    }


    else // car is not stuck
    {
        float accel_and_brake = getAccel(sensors); // compute accel/brake command
        int gear = getGear(sensors); // compute gear
        float steer = getSteer(sensors); // compute steering

        if (steer < -1) // normalize steering
            steer = -1;
        if (steer > 1)
            steer = 1;

        float accel,brake; // set accel and brake from the joint accel/brake command

        if (accel_and_brake>0)
        {
            accel = accel_and_brake;
            brake = 0;
        }
        else
        {
            accel = 0;
            brake = filterABS(sensors,-accel_and_brake); // apply ABS to brake
        }

        clutch = clutching(sensors, clutch);

        Action action = new Action (); // build a CarControl variable and return it

        action.gear = gear;
        action.steering = steer;
        action.accelerate = accel;
        action.brake = brake;
        action.clutch = clutch;
        return action;
    }
}
```

- **public void shutdown() –** este método é chamado no fim da corrida, antes do módulo do carro ser desmontado.

- **public void reset() –** este método é chamado quando a corrida recomeça a pedido do carro.

3 Sensores e Atuadores

O software que usamos nesta competição cria uma separação física entre o motor do jogo e os carros. Assim, para programar um bot não é necessário nenhum conhecimento prévio do motor do TORCS ou da sua estrutura de dados interna. As percepções dos bots e as ações disponíveis são definidas por uma camada de sensores e atuadores. Nesta competição, o input dos bots é composto por dados sobre o estado do carro (a embraiagem atual, o nível do combustível, etc.), o estado da corrida (a volta atual, a distância percorrida, etc.) e o ambiente em redor do carro (as extremidades da pista, os obstáculos, etc.).

3.1 Sensores

O bot percebe o ambiente da corrida através de um número de leituras sensores que fornecem informações sobre o ambiente do carro (a pista, os adversários, a velocidade, etc.) e sobre o estado atual da corrida (o tempo da volta atual, a posição na corrida, etc.). Mais informações na API.

3.2 Atuadores

O bot controla o carro dentro do jogo através de um cujo de atuadores bastante comuns, por exemplo, o volante, o pedal acelerador, o pedal do travão e a embraiagem. Mais informações na API.

4 Testar o Bot

Para testar o bot tem que se entrar no diretório **src-client-java**, abrir um terminal e correr o comando:

$ javac -d classes src/scr/f.java

Noutro terminal (sem fechar o que já está aberto), é necessário correr o comando:

$ torcs

Depois do TORCS iniciar escolhe-se as seguintes opções:

$$Race \rightarrow Quick\ Race \rightarrow New\ Race$$

Agora, voltando ao primeiro terminal aberto, é necessário correr os comandos:

$ cd classes
$ java scr.Client scr.f

Após estes passos a corrida com o bot será iniciada.

f é o nome do ficheiro do bot.

API

O carro tem vários sensores, os quais retornam valores que serão utilizados para o controlo do carro. Estes valores podem servir por exemplo para obter a velocidade instantânea, a embraiagem atual, as rotações por minuto do motor, a posição na pista, etc.

Os sensores vão retornando os valores num intervalo de tempo de 20 milissegundos.

Estes valores são obtidos através de funções. Nas secções seguintes disponibilizamos uma breve descrição do seu funcionamento.

Funções sobre o carro e a pista

getSpeed()
Obtém a velocidade instantânea a que o carro circula. Estes valores podem estar entre 0 e 360 km/h.

getAngleToTrackAxis()
Obtém o ângulo entre a direção do carro e a direção do eixo da pista. Estes valores podem estar entre -π radianos e +π radianos, inclusive.

getTrackEdgeSensors()
Obtém um array de 19 sensores localizadores de alcance. Cada sensor c retorna a distância entre a extremidade da pista e o carro dentro de um alcance de 200 metros. Os 19 sensores cobrem o espaço à frente do carro a cada 20 graus, no sentido dos ponteiros do relógio, desde -90 graus até +90 graus em relação ao eixo do carro. Os valores no array estão entre 0 e 200 metros.

getFocusSensors()
Obtém um array de 5 sensores localizadores de alcance. Cada sensor retorna a distância entre o carro e a extremidade da pista dentro de um alcance de 200 metros. Os 5 sensores cobrem o espaço à frente do carro no sentido dos ponteiros do relógio, desde -90 graus até +90 graus em relação ao eixo do carro. Os valores no array estão entre 0 e 200 metros. Estes sensores só podem ser usados uma vez por segundo de tempo simulado.

getTrackPosition()
Obtém a distância entre o carro e o eixo da pista. O valor é 0 quando o carro está sobre o eixo, -1 quando o carro está na extremidade direita da pista e +1 quando o carro está na extremidade esquerda da pista. Se o valor for maior que 1 ou menor que -1 significa que o carro está fora dos limites da pista.

getGear()
Obtém a embraiagem atual do carro. O valor é -1 quando o carro está em marcha ré, 0 quando em neutro. Os restantes valores até 6, inclusive, são as respetivas mudanças.

Funções sobre os carros dos adversários

getOpponentSensors()
Obtém um array de 36 sensores de adversários. Cada sensor cobre um intervalo de 10 graus dentro de um alcance de 200 metros e retorna a distância do adversário mais perto dentro da área coberta. Os 36 sensores cobrem o espaço inteiro à volta do carro no sentido dos ponteiros do relógio, desde -180 graus até +180 graus em relação ao eixo do carro. Os valores no array estão entre 0 e 200 metros.

getRacePosition()
Obtém a posição na corrida em relação aos outros carros. O valor é 1 para a primeira posição na corrida, 2 para a segunda e assim sucessivamente.

Outras funções

(Usar caso seja necessário)

getCurrentLapTime()
Obtém o tempo decorrido, em segundos, desde o início da volta atual. Estes valores podem estar entre 0 e +∞.

getDamage()
Obtém os pontos de danos atuais. Quanto maior o valor mais danos tem o carro. Estes valores podem estar entre 0 e +∞.

getDistanceFromStartLine()
Obtém a distância, em metros, entre a posição atual do carro na pista e a linha de partida. Estes valores podem estar entre 0 e +∞.

getDistanceRaced()
Obtém a distância, em metros, percorrida pelo carro desde o início da corrida. Estes valores podem estar entre 0 e +∞.

getFuelLevel()
Obtém a quantidade de combustível, em litros. Estes valores podem estar entre 0 e +∞.

getLastLapTime()
Obtém o tempo, em segundos, em que foi completada a última volta. Estes valores podem estar entre 0 e +∞.

getRPM()
Obtém o número de rotações por minuto do motor. Estes valores podem estar entre 0 e +∞.

getWheelSpinVelocity()
Obtém, em radianos por segundo, um vetor de 4 sensores que representa a velocidade de rotação das rodas. Estes valores podem estar entre 0 e +∞.

getZSpeed()
Obtém a velocidade, em km/h, do carro em relação ao eixo Z do carro. Estes valores podem estar entre -∞ e +∞.

getZ()
Obtém a distância, em metros, entre o centro de massa do carro e a superfície da pista em relação ao eixo Z.

Atuadores

accel
Acelerador virtual. Estes valores podem estar entre 0 e 1, 0 é não acelerar e 1 acelerar ao máximo.

brake
Travão virtual. Estes valores podem estar entre 0 e 1, 0 não travar e 1 travar ao máximo.

clutch
Embraiagem virtual. Estes valores podem estar entre 0 e 1, 0 não embraiar e 1 embraiagem ao máximo.

gear
Valor da mudança da embraiagem.

steering
Valor de direção. Estes valores podem estar entre -1 e 1, -1 máximo para a direita e +1 máximo para a esquerda, que corresponde a um ângulo de 0.366519 radianos.

focus
Direção do focus em graus. Estes valores podem estar entre -90 e +90 graus. Ver **getFocusSensors().**