Desenvolvimento de uma Biblioteca de Funções para Simulação no Torque Game Engine

Bruno Baère Pederassi Lomba de Araujo

Orientador: Bruno Feijó

Dept. de Informática, PUC-Rio

PIBIC
Programa Institucional de Bolsas de Iniciação Científica

PUC-Rio
CNPq

2007

Desenvolvimento de uma biblioteca de funções para simulação no Torque Game Engine

Aluno: Bruno Baère Pederassi Lomba de Araujo
Orientador: Bruno Feijó

Introdução

Foi realizado um estudo da implementação do motor de jogos (engine) Torque Game Engine, produzido pela GarageGames, para projetos de simulação física e de inteligência artificial. Este motor destaca-se dos outros pelo seu baixo preço em relação à qualidade (a versão básica para desenvolvedores independentes custando na época do estudo US$150,00) e pela grande comunidade de desenvolvedores reunida nos fóruns do site da GarageGames.

O Torque conta ainda com uma literatura em crescimento (Finney, 2004) (Maurina III, 2006), porém ainda focada no desenvolvimento pela sua linguagem de *script* própria, a Torque Script, e indisponível em versão traduzida para Português. Há o projeto de uma documentação online acessível apenas para os desenvolvedores licenciados, a Torque Developer Network, que mesmo em seu estágio inicial já oferece uma quantidade substancial de informações, aliada aos fóruns abertos supracitados.

Como um motor de jogos completo, o Torque fornece o motor gráfico (render em OpenGL ou DirectX na versão básica, apenas DirectX na versão Advanced), motor de física, interfaces para dispositivos de áudio (OpenAL) e de entrada, e suporte a inteligência artificial por máquina de estados, sendo possível a modificação desses componentes em C++ com a aquisição da licença.

Uma variante do TGE é o TGE Advanced, versão com suporte para os *shaders* mais recentes, que usa uma API baseada em DirectX 9 e proverá suporte a OpenGL no futuro. É possível ainda exportar as aplicações produzidas com a TGEA para o XBox 360 da Microsoft com a aquisição da licença Torque 360.

Objetivos

O objetivo do presente trabalho é desenvolver uma biblioteca de funções para o Torque Game Engine, focada especialmente em aplicações de simulação do tipo “Serious 3D Games”. Esta biblioteca deve possuir componentes de Física, Inteligência Artificial e Interface apropriados para aplicações sérias de simulação. Esta biblioteca deve ser projetada para diversos tipos de simulação 3D e deve garantir o desempenho das aplicações em tempo real.

Metodologia

É reconhecido pela comunidade que o motor Torque possui uma curva de aprendizado acentuada e não se encontrou nenhum material que se dispusesse a ensinar como criar uma aplicação que a usasse. A recomendação dos usuários do fórum é a utilização da aplicação demo que acompanha o motor como ponto de partida para o desenvolvimento de outras aplicações, supondo que assim o desenvolvedor adquira familiaridade com a organização do motor. Portanto, é natural que seja necessária a compreensão de como essa aplicação funciona para que outros projetos daí derivem.

A familiarização com a linguagem de *script* oferecida pelo motor foi realizada praticando os exemplos dos capítulos 3, 4 e 5 da referência (Finney, 2004), que consistem basicamente em fazer o *scripting* do carregamento do cenário, dos modelos e de como estes se relacionavam.

A linguagem de script Torque Script caracteriza-se por ser não-tipada e orientada a objetos. Esta linguagem possui uma sintaxe muito semelhante a linguagens padrão do mercado como C ou C++, o que permite uma rápida assimilação.

A referência, no entanto, não oferece noções de como se comportam ou de como gerenciar a criação de *bots*, entidades de inteligência artificial. Foi realizado um estudo aprofundado de como são instanciadas as entidades representativas do usuário da aplicação e das inteligências artificiais. A conclusão é que os *bots* derivam da mesma classe da qual o usuário é instanciado e o motor fornece algumas funções básicas de criação da inteligência artificial que deve ser inclusa no projeto do motor a ser compilado, visto que a inteligência artificial não é padrão no Torque. Uma outra conclusão é que o Torque fornece suporte à utilização de *waypoints* para navegação dos *bots* em cena. A criação dos *waypoints* é possível usando o editor interno World Editor.

Para a inclusão de modelos animados, o Torque Game Engine faz uso de um formato próprio, o dts. Há, no site da GarageGames, exportadores para as principais ferramentas de modelagem do mercado, como o 3DS Max e o Maya, além do suporte dos usuários para ferramentas gratuitas, como o Blender.

A parte de simulação física do motor permite uma alta customização, que se faz necessária a cada caso. Para simulações navais, por exemplo, é desejável uma representação a mais fiel possível de um corpo de água e de uma embarcação. A água é simulada por meio de uma estrutura denominada *waterblock*, que permite modificações na intensidade e direção de ondas. Para simular uma embarcação, pode-se fazer uso do *datablock* HoverVehicleData,

Fig.1 Ambiente simulado no motor

dando assim a sensação de que o veículo está flutuando na água. A Fig. 1 ilustra o ambiente e a Fig. 2 apresenta a IA dos testes básicos realizados.

Foi realizado um estudo da estrutura de diretórios e arquivos do exemplo do tutorial.base que acompanha a versão 1.3 do Torque Game Engine para servir como guia para melhor compreensão dos módulos de modificações que podem ser inclusos aos projetos que utilizem a *engine*. A descrição é dada a seguir:

tutorial.base/main.cs - Definição das funções de parsing de argumentos. Variável global $defaultGame armazena o nome do diretório padrão do usuário. Iniciar a engine com o argumento –mod modname faz com que modificações sejam carregadas. Primeiro, carrega o default game para em seguida carregar os módulos passados como argumento.

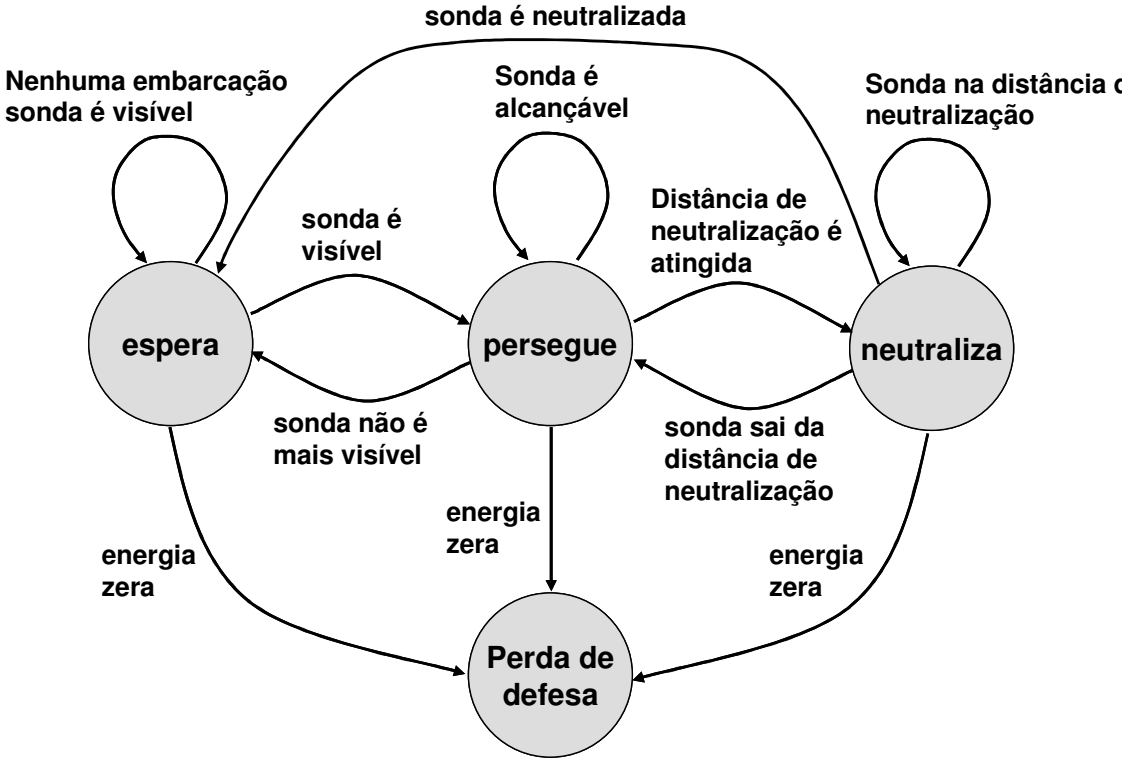

Fig. 2 Máquina de estado considerada

tutorial.base/tutorial.base/main.cs - Carrega scripts do diretório common e carrega valores *default* do cliente e servidor. Possui *override* das funções onStart e onExit. Outras funções importantes como initServer e initClient são definidas aqui. Nome da janela cliente é argumento para initCanvas (dentro de initClient). Função loadTestMission é definida aqui. Vemos também a passagem do *path* da missão a ser carregada.

tutorial.base/tutorial.base/pref.cs - Configurações.

tutorial.base/tutorial.base/client/audioProfiles.cs - Configurações de áudio.

tutorial.base/tutorial.base/client/default.bind.cs - Mapeamento de teclas. Define que função está associada a cada tecla, dividida pelas funções. O objeto ActionMap pode receber uma *binding* de comando (executa uma instrução definida na string de parâmetro) ou uma *binding* simples (o argumento da função associada é dado pelo tempo ou estado da tecla apertada). Os *binds* também podem receber como argumentos combinações de teclas.

tutorial.base/tutorial.base/client/default.cs - Configurações padrão.

tutorial.base/tutorial.base/client/loadingGUI.cs - Definição de funções associadas à interface com o usuário.

tutorial.base/tutorial.base/missionDownload.cs - Faz interface com a GUI do jogo. Quem carrega a missão de fato é o script common/client/missionDownload.cs.

tutorial.base/tutorial.base/client/playGUI.cs - Definição das funções de interface (ativação e desativação do objeto ActionMap (move map)) do default.bind.cs.

tutorial.base/tutorial.base/serverConnection.cs - Funções para lidar com conexão ao servidor.

tutorial.base/tutorial.base/data - Descrições dos arquivos de missão (cenários) onde são inseridos interiores, *spawn spheres*, *skyboxes*, *waterdatablocks*, etc. É onde são armazenadas as *meshes* dos personagens.

tutorial.base/tutorial.base/server/audioprofiles.cs - *Datablocks* de áudio simples e 3d.

tutorial.base/tutorial.base/server/camera.cs - *Datablock* da câmera do observador, também define a velocidade das outras câmeras disponíveis.

tutorial.base/tutorial.base/server/default.cs - Configurações padrão do servidor.

tutorial.base/tutorial.base/server/editor.cs - Declaração das *shapes* e funções usadas pelo Mission Editor.

tutorial.base/tutorial.base/server/game.cs - Declaração das funções de inicialização do servidor e da missão. Métodos de extensão da GameConnection.

tutorial.base/tutorial.base/server/player.cs - Carrega script do player (./data/...), definições de áudio, emissores de partículas, *datablock* PlayerData (PlayerShape) e métodos relacionados.

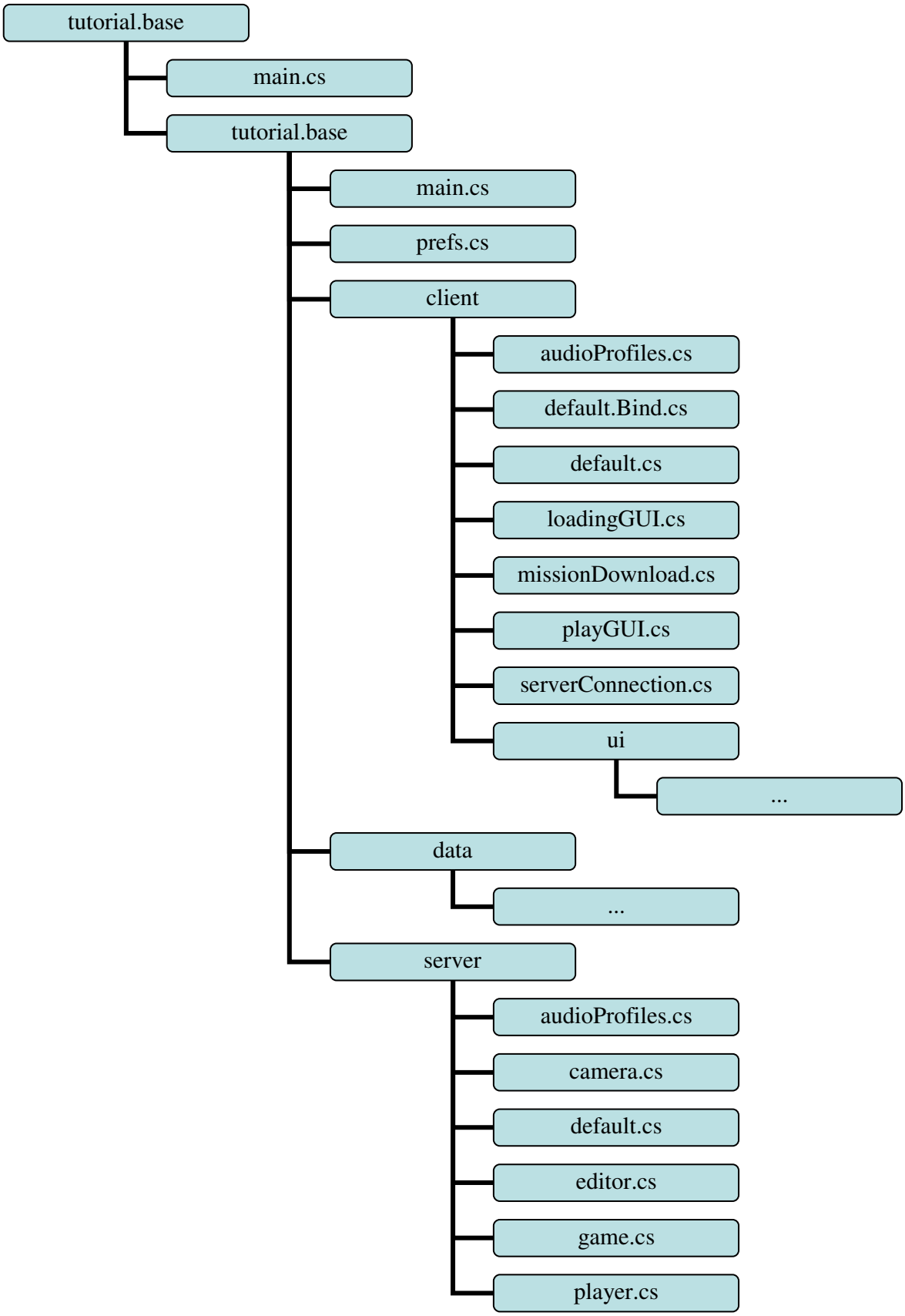

Fig 3 Estrutura de diretórios do módulo de exemplo tutorial.base

Conclusões

O uso do Torque para simulações envolvendo física e inteligência artificial mostrou-se mais complexo do que o inicialmente imaginado. Apenas os testes básicos foram realizados.

Em breve, pretende-se liberar à comunidade, sob forma de uma apostila, um guia do Torque Script com o detalhamento da organização da aplicação para finalidades de simulação. Além disto, deve ser compilado um manual de procedimentos para a inclusão de modelos e animações desenvolvidas pelo usuário e para a criação de uma inteligência artificial baseada em máquinas de estados para simulações navais.

Referências

Finney, K. C. 2004. **3D Game Programming All in One**. 1.ed. Canada: Premier Press.

Maurina III, E.F. 2006. **The Game Programmer's Guide to Torque**: Under the Hood of the Torque Game Engine (GarageGames), AK Peters, Ltd.