Architectuur en Algoritmen van Computer Games

Hovercraft Universe

http://uhasseltaacgua.googlecode.com/

Olivier Berghmans, Nick De Frangh, Dirk Delahaye,
Kristof Overdulve, Pieter-Jan Pintens, Tobias Van Bladel

18 mei 2010

Inhoudsopgave

1 Introductie **2**

2 Gebruikte libraries **2**

3 Programmastructuur en -organisatie **2**
3.1 Algemeen 2
3.2 Besturing 3
3.3 Netwerkstructuur 3
3.4 Scritping en AI 3

4 Rolverdeling en verantwoordelijkheden **5**

1 Introductie

2 Gebruikte libraries

Rendering Ogre[1]

Physics Havok-Physics[2]

Netwerk ZoidCom[3]

Input Object Oriented Input System (OIS)[4]

Geluid FMOD Interactive Audio Middelware[5]

Scripting Lua[6] en LuaBind[7]

GUI Adobe Flash[8] en Hikari[9].

3 Programmastructuur en -organisatie

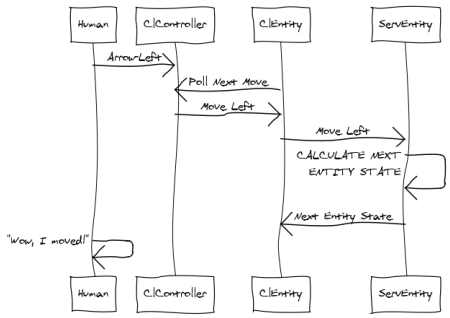
3.1 Algemeen

De basisstructuur van de engine achter Hovercraft Universe is gemaakt naar analogie met de architectuur van *Shellshock: Nam '67* [3]. Elk object in de spelwereld waarmee geïnterageerd kan worden, wordt voorgesteld door een `Entity`. Deze entiteiten worden gegroepeerd in de `EntityManager`, die de entiteiten tijdens het spel kan updaten. In onze implementatie is een `Entity` tevens een uitbreiding van `NetworkEntity`, die replicatie over het netwerk mogelijk maakt.

Entiteiten kunnen bestuurd worden door `Controller`s. Een controller genereert *events*, die door de entiteiten gepolld worden. Deze interface laat toe om entiteiten te besturen op verschillende manieren: toetsenbord/muis, AI, ...

Voor de visuele representatie op de clients heeft elke relevante entiteit een `EntityRepresentation`. Deze verbindt de conceptuele entiteit met een entiteit in Ogre. Om de wereld te kunnen tekenen vanuit het standpunt van één speler worden deze representatie-objecten, samen met de Ogre camera, bijgehouden in een `GameView`.

[1] http://www.ogre3d.org/
[2] http://www.havok.com/index.php?page=havok-physics
[3] http://www.zoidcom.com/
[4] http://sourceforge.net/projects/wgois/
[5] http://www.fmod.org/
[6] http://www.lua.org/
[7] http://www.rasterbar.com/products/luabind.html
[8] http://www.adobe.com/products/flash/
[9] http://code.google.com/p/hikari-library/



Figuur 1: The client entity polls its next move from the controller

3.2 Besturing

3.3 Netwerkstructuur

Over Zoid-Com en het repliceren van data over het netwerk.

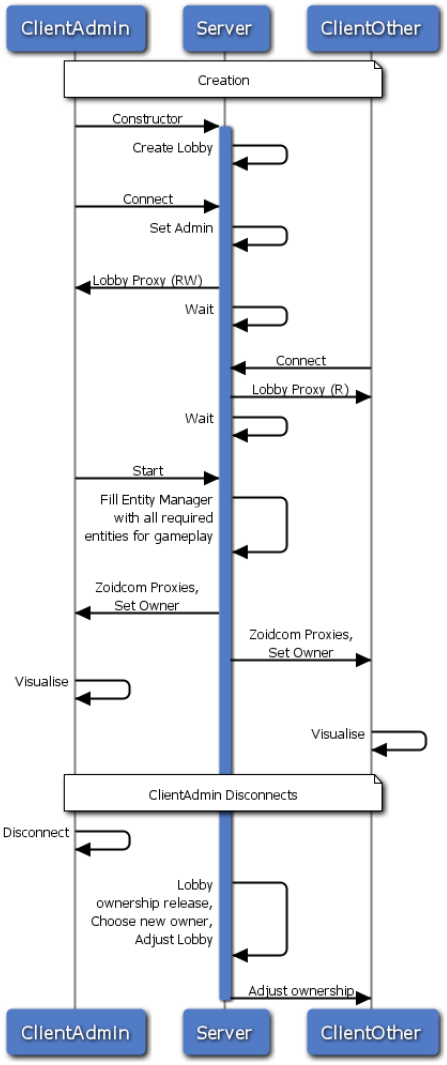
3.4 Scritping en AI

Voor de bots is een AI voorzien, die "menselijk" stuurgedrag nabootst. De algoritmen van deze AI zijn gebaseerd op de autonome stuursimulaties van Craig Reynolds [2]. De huidige AI is in staat om een voorgedefinieerd pad te volgen binnen een bepaalde marge ("Path Following"). Een pad wordt voorgesteld als een *multiline*: een lijst van punten in de wereld, telkens geassocieerd met een straal (`radius`). De straal op punt x_i stelt de *breedte* van het pad voor tussen punt x_i e, x_{i+1}. Deze paden worden opgeslagen in aparte bestanden. Dit betekent dat de level designers per map een ideaal pad (of meerdere paden) voor de AI kunnen aanmaken.

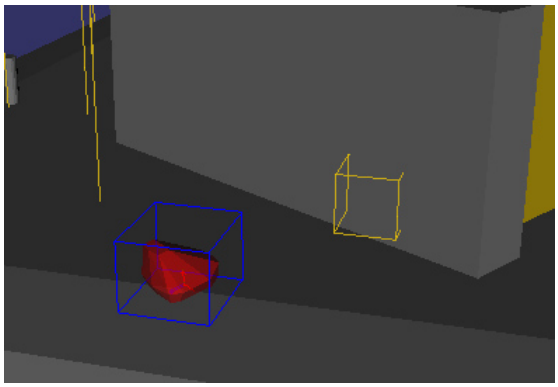
De AI is geprogrammeerd in Lua [1], om aanpassing van de AI-algoritmen mogelijk te maken zonder dat het spel hiervoor gehercompileerd moet worden. Voor de binding tussen Lua en C++ wordt Luabind gebruikt. De `ScriptWrapper` klasse abstraheert deze scripts en laat de programmeurs toe om met de scripts te werken, zonder Lua-specifieke constructies te moeten gebruiken.

Collision avoidance voor de AI is ook geïmplementeerd met behulp van Havok physics. Er zijn twee implementaties beschikbaar voor de `EntityCollision` interface. Het eenvoudige model (zie Figuur 3) plaatst een kubus op een bepaalde afstand van het voertuig om objecten te detecteren. Dit model gebruikt zeer weinig resources, maar is ook niet zo krachtig. Het geavanceerde model gebruikt een 3D-lichaam dat zich uitstrekt van het voertuig tot de gewenste afstand voor de figuur (zie Figuur 4). Deze manier van werken heeft een kleine meerkost in resources in vergelijking met het eenvoudige model.

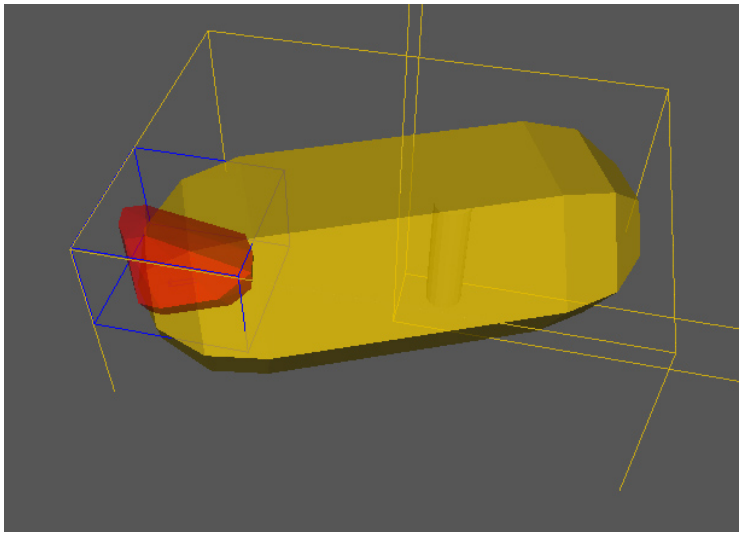
Aangezien deze methodes afhankelijk zijn van de physics, draaien ze op de server. Als een object wordt gedetecteerd, wordt dit door middel van een event doorgegeven aan een callback functie in de entiteit. Dit wordt vertaald naar



Figuur 2: ZoidCom werking



Figuur 3: Het eenvoudige collision avoidance model



Figuur 4: Het geavanceerde collision avoidance model

een boolean in de entiteit, die door het netwerk wordt gerepliceerd op de client. Client-side maakt de AI dan een beslissing om het obstakel te ontwijken.

4 Rolverdeling en verantwoordelijkheden

Referenties

[1] Roberto Lerusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes. *Lua 5.1 Reference Manual*. Lua.Org, 2006.

[2] Craig Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference 1999*, 1999.

[3] Jorrit Rouwé. The guerrilla guide to game code. Technical report, Gamasutra.com, 2005. http://www.gamasutra.com/view/feature/2280/the_guerrilla_guide_to_game_code.php.