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

27 mei 2010

Inhoudsopgave

1 Introductie

Hovercraft Universe is een race game waarin spelers tussen planeten in de ruimte kunnen rondvliegen met hovercrafts.

meer bla-bla:p

2 Gebruikte libraries

Physics De fysische gamewereld wordt gesimuleerd door Havok Physics[1]. Havok draait enkel op de server en zorgt voor alle berekeningen die nodig zijn voor fysische interactie.

Rendering Voor het renderen gebruiken we de Ogre Open Source Graphics Engine[2]. Ogre draait enkel op de clients en zorgt voor de visuele representatie van de game-entiteiten. Andere, uitsluitend visuele, effecten en overlays worden ook door Ogre afgehandeld: particle effects, tekst overlays boven de entiteiten, ...

Netwerk ZoidCom[3]

Input Object Oriented Input System (OIS)[4]

Geluid FMOD Interactive Audio Middelware[5]

Scripting Lua[6] en LuaBind[7] bieden de mogelijkheid om op een dynamische manier eigenschappen van het spel te veranderen.

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, ...

[1] http://www.havok.com/index.php?page=havok-physics
[2] http://www.ogre3d.org/
[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/

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`.

De bovenstaande klassen zitten vervat binnen het `CoreEngine`-project, dat in principe voor eender welk soort spel gebruikt kan worden. Voor Hovercraft Universe is een specifieke implementatie voorzien: de `HovercraftController`,

3.2 Netwerkstructuur

Olivier?

3.3 Physics

Pieter-Jan?

3.4 Grafische userinterface

De onderdelen van de GUI zijn geïmplementeerd in ActionScript[10], en worden met behulp van Hikari gevisualiseerd in Ogre. Hikari is een library die Flash (SWF) bestanden kan inlezen, en voor de binding tussen Flash en C++ zorgt.

3.5 Geluid

iets over FMod bestanden en hoe het werkt

Iets over de 3D emitter...

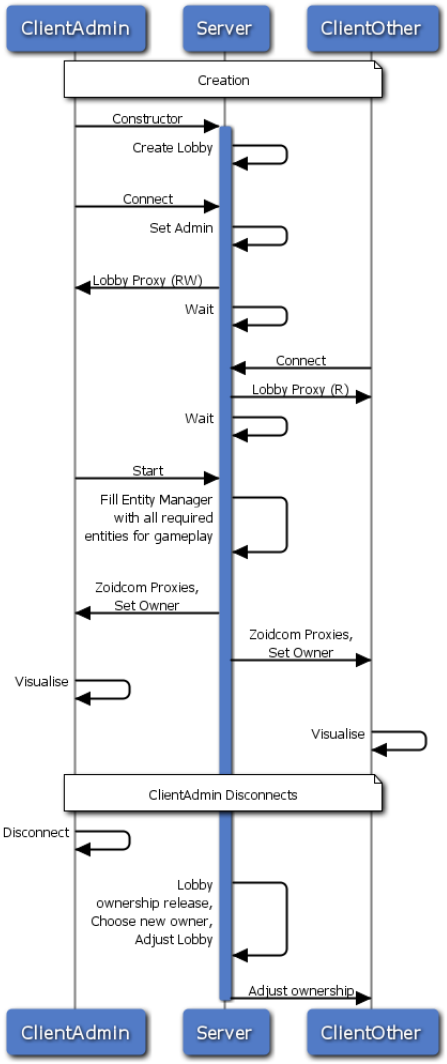
...Nick?

3.6 Besturing

De centrale klasse voor de besturing is de `KeyManager`. In deze klasse kunnen alle nodige acties geregistreerd worden door middel van een uniek ID, bijvoorbeeld in een enumeratie. Aan elk van deze acties kunnen één of meerdere knoppen gebonden worden. Telkens wanneer de `HovercraftPlayerController` een key event binnenkrijgt, zal er bij de key manager nagegaan worden welke actie er overeenkomt met deze toets. Op deze manier is het mogelijk om de spelbesturing gemakkelijk te wijzigen zonder dat de controller aangepast moet worden. Bovendien kunnen er meerdere toetsen voor dezelfde actie gebruikt worden.

In de `KeyManager` staat ook nog een mapping tussen alle toetsen die OIS ondersteunt en hun bijhorende string-representatie. Zo kunnen de controls ingelezen

[10] http://www.adobe.com/devnet/actionscript/



Figuur 1: ZoidCom werking

of weggeschreven worden naar een configuratiebestand. Standaard is dit bestand te vinden in de data\controls map, maar wanneer dit bestand ontbreekt zal er automatisch een nieuw aangemaakt worden met de default controls. In dit configuratiebestand staan alle geregistreerde acties per categorie. Voor elke actie kunnen één of meerdere toetsen opgegeven worden, gescheiden door een komma. Wanneer een actie ontbreekt, zal de standaardtoets voor deze actie gebruikt worden.

3.7 Scripting 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.

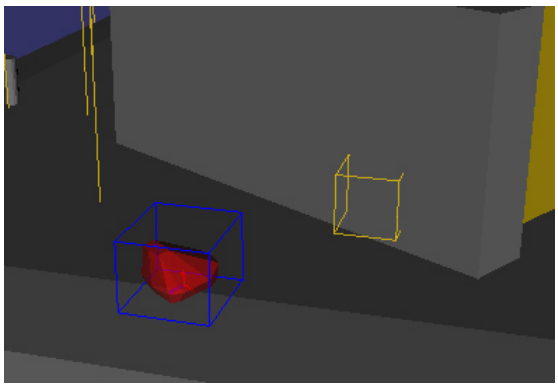
De AI is geprogrammeerd in Lua [1], om aanpassing van de AI-algoritmen mogelijk te maken zonder dat het spel hiervoor gehercompileerd moet worden. Voor de binding tussen Lua en C++ wordt Luabind gebruikt. De `ScriptWrapper` klasse abstraheert deze scripts en laat de programmeurs toe om met de scripts 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 2) 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 3). Deze manier van werken heeft een kleine meerkost in resources in vergelijking met het eenvoudige model.

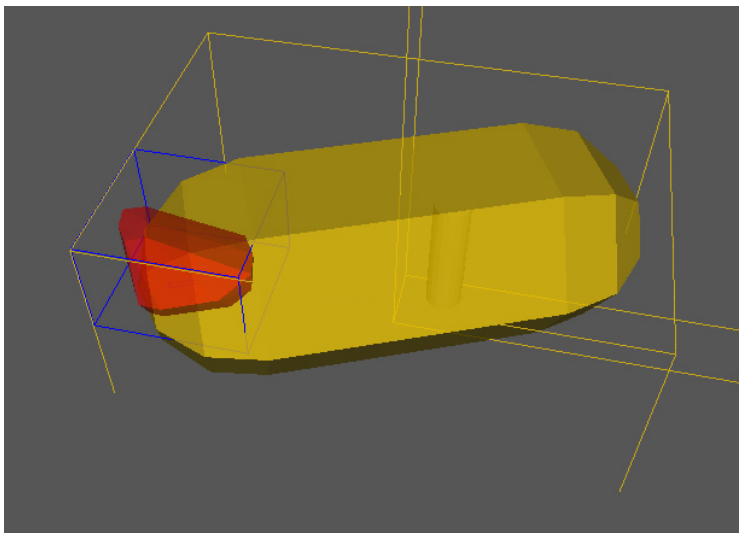
Aangezien deze methodes afhankelijk zijn van de physics, draaien ze op de server. Als een object wordt gedetecteerd, wordt dit door middel van een event doorgegeven aan een callback functie in de entiteit. Dit wordt vertaald naar een boolean in de entiteit, die door het netwerk wordt gerepliceerd op de client. Client-side maakt de AI dan een beslissing om het obstakel te ontwijken.

3.8 Configuratiebestanden

Configuratiebestanden zijn beschikbaar voor de server en client, in INI formaat (zie Listing 1). Dit laat gebruikers toe om hun nickname, gekozen voertuig, enz. te kiezen en op te slaan. Een aantal server-specifieke instellingen worden ook ondersteund (aantal spelers, AI-script, sterkte van de zwaartekracht, ...). Het spel kan deze bestanden ook opslaan met de huidige instellingen.



Figuur 2: Het eenvoudige collision avoidance model



Figuur 3: Het geavanceerde collision avoidance model

Listing 1: Het INI bestandsformaat

```
[section name]
keyname=value
#comment
keyname = value, value , value #comment
```

4 Rolverdeling en verantwoordelijkheden

dit erin houden?

Algemene architectuur Kristof

Networking Olivier

Scripting en AI Dirk

Physics Pieter-Jan, Tobias

Controls Tobias

Geluid Nick

GUI Nick

3D modellen, Ogre Kristof, Pieter-Jan

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.