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

31 mei 2010

Inhoudsopgave

1 Introductie **2**

2 Gebruikte libraries **2**

3 Programmastructuur en -organisatie **2**
3.1 Algemeen . . . 2
3.2 Netwerkstructuur . . . 3
3.3 Modellen en user-data . . . 3
3.4 Visualisatie en Ogre . . . 4
3.5 Physics . . . 4
3.6 Grafische userinterface . . . 5
3.7 Geluid . . . 5
3.8 Besturing . . . 5
3.9 Scripting en AI . . . 5
3.10 Game Logic . . . 6
3.11 Level of Detail . . . 7
3.11.1 Netwerk . . . 7
3.11.2 Rendering . . . 7
3.12 Misc. . . . 7
3.12.1 Configuratiebestanden . . . 7

4 Rolverdeling en verantwoordelijkheden **8**

1 Introductie

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

meer blabla: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 `Controllers`. 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`, waarvan de `HovercraftAIController` en `HovercraftPlayerController` afgeleid worden.

Iets over de application / server application structuur

3.2 Netwerkstructuur

Server, client, netwerk, zoidcom, chatfunctionaliteit, replicatie, interpolatie, ...

3.3 Modellen en user-data

Olivier!

De **modellen** zijn gedefinieerd in het (niet open) bestandsformaat van 3DS Max. Deze worden geëxporteerd naar het DotScene XML-formaat met behulp van de OgreMax Scene Exporter[10]. DotScene[11] is een gestandaardiseerd XML-formaat dat informatie bevat over het soort entiteit, de grootte van het model, de gebruikte materialen en meshfiles, etc. Tevens bevat dit DotScene bestand informatie specifiek voor gebruik in HovercraftUniverse (snelheid, gewicht, ...). Een voorbeeld van deze user-data is gegeven in Listing 1.

Listing 1: UserData in DotScene

```
<Hovercraft>
    <Name>hover1</Name>
    <Description>Nonw</Description>
    <OgreEntity>hover1</OgreEntity>
    <ProcessInterval>0.016</ProcessInterval>
    <Speed>100</Speed>
    <Mass>50</Mass>
    <Acceleration>10</Acceleration>
    <Steering>100</Steering>
</Hovercraft>
```

Voor de **mappen** (tracks) wordt er één groot DotScene wereld gegenereerd waarin alle relevante nodes zijn gedefinieerd: planeten, checkpoints, statische objecten, startposities, etc. Een voorbeeld van een dergelijke node is te zien in Listing 2

[10] http://www.ogremax.com/
[11] http://www.ogre3d.org/wiki/index.php/DotScene

Listing 2: Eén van de nodes in de DotScene van een wereld.

```
<node name="Jump01">
    <position x="-148.891" y="21.0134" z="-20.245" />
    <scale x="1" y="1" z="1" />
    <rotation qx="0.13454" qy="0.990908"
        qz="-5.34715e-008" qw="-7.48115e-008" />
    <entity name="Jump01" castShadows="true" receiveShadows="true" meshFile="Ju
        <userData>
            <![CDATA[<OgreEntity><GameEntity>Jump01_ent</GameEntity></OgreEntit
        </userData>
        <subentities>
            <subentity index="0" materialName="Jump02" />
        </subentities>
    </entity>
</node>
```

3.4 Visualisatie en Ogre

De `EntityRepresentation` klasse zorgt ervoor dat Ogre de verschillende entiteiten kan visualiseren. meer hierover

Ook worden hier een aantal visuele effecten aan toegevoegd. Ogre Particle Systems worden gebruikt om rook, vonken, etc. weer te geven. Er is tevens een tekst-overlay voor de nicknames voorzien zodat de spelers elkaar kunnen herkennen tijdens het spel. Aangezien deze effecten enkel client-side gebeuren bij het tekenen van de entiteiten, heeft dit geen invloed op de performantie van de netwerkcomponenten.

3.5 Physics

Alle acties op de hovercrafts en interacties met de omgeving gebeuren door middel van fysische krachten. Dit gebeurt volledig op de server, die een Havok-thread draait. De input van de clients, zowel menselijke spelers als AI, wordt vertaald in een kracht op de hovercraft. Havok berekent vervolgens alle totale krachten en daaruit volgende versnellingen en snelheden. Hieruit zal de nieuwe positie van alle entiteiten bepaald worden. Deze posities worden dan doorgestuurd naar alle clients zodat zij de wijzigingen kunnen zien. Bovendien zorgt Havok ook voor de collision detection zodat verschillende entiteiten niet tegelijk op dezelfde plaats kunnen bestaan.

De zwaartekracht op de verschillende planeten is afgeleid van het `PlanetGravity` voorbeeld van Havok. Concreet wordt er rond elke planeet een *phantom* geplaatst. Wanneer een object binnen deze phantom komt, wordt er een kracht uitgeoefend waardoor de objecten aangetrokken worden door de planeet. Deze kracht kan per planeet verschillen zodat elke planeet een eigen zwaartekrachtsterkte heeft.

Op elke hovercraft-entiteit wordt nog een extra actie gedefiniëerd om het object te laten zweven. Dit is een kracht die in tegengestelde richting ten opzichte van de zwaartekracht werkt. Afhankelijk van de hoogte verandert de grootte van

de kracht. Dicht bij het planeetoppervlak zal er een sterke tegengestelde kracht gebruikt worden. Hoe verder van de planeet, hoe zwakker de kracht wordt. Vanaf een bepaalde hoogte valt de kracht weg, zodat de hovercraft terug naar beneden begint te vallen. Op deze manier krijgen we een lichte oscillatie van de hoveringhoogte.

3.6 Grafische userinterface

De onderdelen van de GUI zijn geïmplementeerd in ActionScript[12], 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.

Nick

3.7 Geluid

iets over FMod bestanden en hoe het werkt

Iets over de 3D emitter...

3.8 Besturing

...Nick?

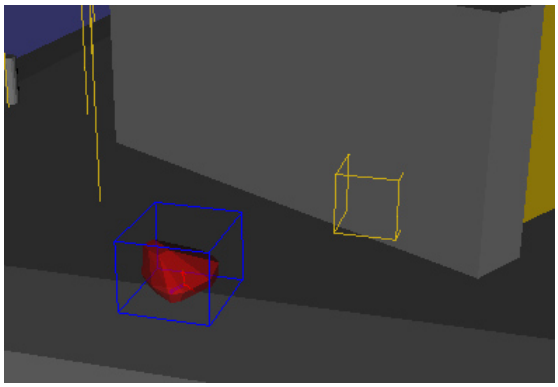
De centrale klasse voor de besturing is de `KeyManager`. In deze klasse kunnen alle nodige acties geregistreerd worden door middel van een uniek ID, bijvoorbeeld in een enumeratie. Aan elk van deze acties kunnen één of meerdere knoppen gebonden worden. Telkens wanneer de `HovercraftPlayerController` 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 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.

3.9 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

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



Figuur 1: Het eenvoudige collision avoidance model

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.

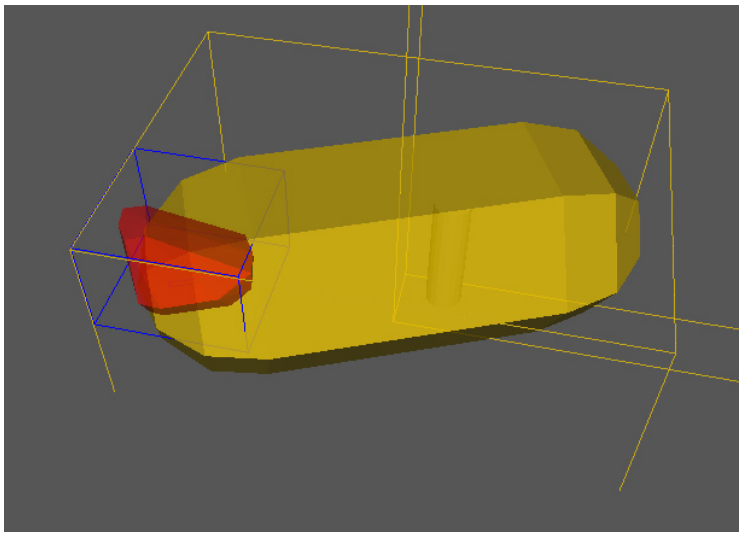
Collision avoidance voor de AI is ook geïmplementeerd met behulp van Havok physics. Er zijn twee implementaties beschikbaar voor de `EntityCollision` interface. Het eenvoudige model (zie Figuur 1) plaatst een kubus op een bepaalde afstand van het voertuig om objecten te detecteren. Dit model gebruikt 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 2). 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.

3.10 Game Logic

Lobby, admins? of bij netwerken

Raceplayer, Racestate, positiebepaling



Figuur 2: Het geavanceerde collision avoidance model

3.11 Level of Detail

update rate afhankelijk van de afstand tussen A en B, ...

3.11.1 Netwerk

3.11.2 Rendering

Polycount verminderen afhankelijk van de afstand van het te tekenen object met ogre...

3.12 Misc.

3.12.1 Configuratiebestanden

Configuratiebestanden zijn beschikbaar voor de server en client, in INI formaat (zie Listing 3). 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.

Listing 3: Het INI bestandsformaat

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

4 Rolverdeling en verantwoordelijkheden

dit erin houden?

Kristof Overdulve Algemene architectuur, 3D modellering, Ogre visualisatie en animatie

Pieter-Jan Pintens Havok Physics, 3D modellering, Ogre visualisatie en animatie

Dirk Delahaye Scripting, AI, visuele effecten, configuratiebestanden

Tobias Van Bladel Input, controls, physics

Nick De Frangh Grafische user interface, Geluid

Olivier Berghmans Algemene architectuur, netwerk

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.