NitroGEAR: Overdrive

Core Project Document

Minor *"Software Ontwerpen en Toepassen"*

Computer Science

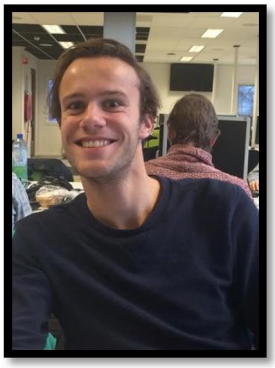
TU Delft

21-1-2015

Made by: Jan-Cees van Senden, Jasper Binda, Jorien Knipping, Joseph Verburg, Koen Ziere en Rick Koster

Table of contents

Team



Name: Jan-Cees van Senden
Study: Applied Physics
Faculty: TNW
E-mail: jancees.van.senden@gmail.com
Student number: 4178017

Name: Joseph Verburg
Study: Electrical Engineering
Faculty: EWI
E-mail: josephverburg@gmail.com
Student number: 4018575

Name: Jasper Binda
Study: Mechanical Engineering
Faculty: 3ME
E-mail: jasperbinda@hotmail.com
Student number: 4226313

Name: Koen Ziere
Study: Civil Engineering
Faculty: CiTG
E-mail: koen@ziere.nl
Student number: 4195760

Name: Jorien Knipping
Study: Applied Mathematics
Faculty: EWI
E-mail: jorien.knipping@gmail.com
Student number: 4212568

Name: Rick Koster
Study: Applied Physics
Faculty: TNW
E-mail: rick_koster12@hotmail.com
Student number: 4110749

Introduction

This report gives some insight on the game NitroGear: Overdrive. The game was an assignment for the minor project EWI3620TU. A course we follow for our minor “Software Ontwerpen en Toepassen”.

Our game is a fast, unpredictable and highly competitive 4 player 3D racing game. Players will race in 4 different mini games, randomly chosen. In the mini games there is a variety of ways to get points. The player with most points at the end of the mini game is the winner. If two or more players have equal amount of points at the end of the mini game, than they are both winners. The winner or winners of the mini game get a point, first player to get 3 points wins the game.

Target Audience

The intended audience for this game is primarily casual gamers who like playing racing games with friends since the game can only be played using LAN multiplayer. But we think other gamers will also find this game a lot of fun thanks to the mini games.

Platform & Controls

This game will be made for the PC. The controls will be on the keyboard and it is also possible to use the mouse for shooting.

Control	Purpose
Horizontal arrow keys	Turning
Keys A and D	Turning
Vertical arrow keys	Acceleration
Keys W and S	Acceleration
Space bar	Brake
Ctrl	Shooting
Left mouse button	Shooting
Backspace	Skip level tour
Esc	Pause
Key R	Reset car position

Story and setting of the game

At the start of the game the 4 players and their cars are dropped inside an arena somewhere on earth. They will battle against each other in mini games. There are 4 types of mini games. If a player wins 3 mini games the game is ended and he is the winner. If a player wins a mini game he receives an overall point. All mini games have a time limit of 2 minutes. Next, the 4 mini games will be explained.

Race mode

The purpose of this mini game is to race as fast as you can to a checkpoint. The checkpoints will be generated with a genetic algorithm. The checkpoints disappear when all players minus one have reached it. If this does not happen fast enough the checkpoint will also disappear after 30 seconds. The amount of points received for reaching a checkpoint depends on the quantity of players. The last player will always get zero points. Beneath there is a table with points received. The mini game ends when a player has 10 points. This player will get 1 overall point.

Finished	Points received
First	Quantity of players - 1
Second	Quantity of players - 2
Third	Quantity of players - 3
Fourth	0
Last	0

Race to the top mode

When this mini game starts a tower of planes connected by ramps is randomly generated. On top of this tower a checkpoint is initialized. The first player to reach this checkpoint wins the mini game and receives 1 overall point. This mini game has a time limit, when this is reached the mini game ends and the player that has been on the highest flour wins.

Zombie mode

During this mini game more and more zombies will be generated. The goal is to kill as many zombies as possible. There is a time limit to this mini game. When the time limit is reached the player with most kills wins the mini game, and receives 1 overall point.

Tron mode

In the tron mode all players have a line following them, this line will work as a barrier for all the players of the game. When a player touches this line, he is dead. If all players except one are dead, the mini game ends, and the remaining player gets 1 overall point. If the time limit is reached all remaining players receive 1 overall point.

Technical components

Computer Graphics

Subject	Difficulty	Who is responsible?
Level Tour	**	Jorien Knipping
High Scores	*	Joseph Verburg
Options	*	Koen Ziere
Credits	*	Koen Ziere
UI Animations	*	Koen Ziere
Start-, Pause-, End-Screen	*	All
Procedural meshes	***	Rick Koster and Joseph Verburg
3D Animated Models	**	Rick Koster
3D Models	**	Rick Koster
Sounds	*	Koen Ziere
Textures as input	**	Rick Koster
Unsteady camera	**	Jorien Knipping ad Joseph Verburg
Particle Systems	*	Jan-Cees van Senden
Total	**20**	

Level Tour

4 points in the scene, camera travels this 4 points while looking to the middle of the arena.

High Scores, Options, Credits, UI Animations

All fixed with new UI system. The UI animations in the editor where prominently used, also a little scripting.

Start-, Pause-, End-Screen

Start -Screen is the menu. Pause-Screen can be called by the key esc. For the End-Screen Text displays and script were used to rank the scores. The End-Screen shows the scores.

Procedural meshes
The tower meshes where entirely made in Unity. The fences are constructed using models made in Blender and scripting within Unity.

3D Animated Models
The car model was made using Blender. Initially the model was animated within Blender.

Encountered problems: Importing the animations into Unity was unsuccessful. Animating the suspension within Unity was unsatisfactory. The wheels of the car model are animated within Unity.

3D Models
All models were made using Blender.

Sounds
Music sound and sound effects where implemented in the editor of the scene.

Textures as input
The arena is textured using a script.

Unsteady camera
The camera follows the player with a delay. All scripted.

Particle System
Particle systems are used for the checkpoints and for the placement of the arena assets.

Artificial Intellegence

Subject	Difficulty	Who is responsible?
Genetic algorithm	****	Jan-Cees van Senden
Enemies	**	Jasper Binda
Total	**6**	

Genetic algorithm
This algorithm is used to generate the checkpoints. It accounts the position of all players so it will be a fair race. All scripted.

Enemies
If player is near enemy, enemy attacks and follows player.

Encountered problems: Implementing this in multiplayer was difficult. Choosing the player nearest to the enemy gives some difficulties in multiplayer. First the enemies choose their victim locally. This gave problems due to a lag. As solution the server determined which player was closest to the enemy, this data was sent from server to the other players.

Web & Database

Subject	Difficulty	Who is responsible?
Collect play through data	**	Joseph Verburg
Store player data on webserver	**	Joseph Verburg
Online accounts with avatars	***	Joseph Verburg
Collect and show high scores from webserver	**	Joseph Verburg
Visualize data on web server	**	Joseph Verburg
Total	**11**	

Collect play through data

The scores are gathered and sent to the server.

Store player data on webserver

JSON is used to send data and MyCyle is used to store the data**.**

Encountered problems: Co-routines gave some difficulties.

Online accounts with avatars

Unity monodelopcompatibility.dll was used to make file open dialog work, then with a web request send to the server. Requesting has been done by putting www.texture property in web request and a RawImage UI element.

Collect and show high scores from webserver

To make statistics from database Grouping Queries was used.

Visualize data on web server

The data is visualised on this url: http://sot.meaglin.com/dashboard.html .

Programming

Subject	Difficulty	Who is responsible?
Arena texturing	**	Jasper Binda
Procedural asset placement	**	Rick Koster
Arena rendering	**	Jan-Cees van Senden
Fading platforms	*	Jan-Cees van Senden
Online multiplayer	****	Jasper Binda
Rear view option	*	Jorien Knipping
Physics collisions	*	All
Vehicle movements/ controlling	****	Jorien Knipping
Mini-map	***	Jorien Knipping
Countdown	*	Jorien Knipping
Race against the clock	*	All
FPS independent	**	Jasper Binda
Total	**24**	

Arena texturing
Both the ground and the walls have 2 different textures, these are randomly chosen. It was implemented using script and splat prototypes.

Encountered problems: Splat prototypes gave some difficulties.

Procedural asset placement
The arena assets are placed in the game randomly with a script. It has some restrictions in placement, it cannot be placed on another arena asset or a checkpoint.

Encountered problems: The restriction not to place an asset on a checkpoint gave problems. This because the checkpoints were placed with another script. It was solved by restricting both the checkpoint script and asset placement script.

Arena rendering
A terrain was manipulated with script.

Encountered problems: On the fly modelling gave to many problems. So this was skipped.

Fading platforms
An algorithm written in script controls fading behaviour.

Encountered problems: Fading gave huge problems with textures. This was solved trying several methods until a working method was found.

Online multiplayer
The multiplayer was mainly made by using existing functions of Unity. The data is stored in the master server of Unity. The data is send through the function OnSerializeNetworkView(). The script uses Network.Instanciate() instead of Instantiate(). All objects that use the network have their own NetwerkView.

Encountered problems: There were loads of problems translating single player methods to multiplayer methods. And a big amount of methods were called in wrong order.

Rear view option
Implemented in script for unsteady camera.

Physics collisions
Implemented in multiple scripts.

Vehicle movements/controlling
Car is controlled by wheel colliders.

Encountered problems: To make a real and stable car controller gave many difficulties. First the controller script was based on a simple player controller, this was not real enough. Second it was tried to make a script using the physics addTorque() and addForce(), this was also not real enough and difficult. At the end a script using wheel colliders was made. This was very difficult and the car tends to roll over when it had a high speed and turned. This was partially solved by making a function called downForce().

Mini-map
Mini-map is made with new UI system, using one prefab and scripting.

Encountered problems: First the Mini-map was made with the old UI system, however it had to be made with the new UI system. Scripting with the new UI system gave many problems and took a huge amount of time to fix. Also implementing sprites made by Koen Ziere was almost impossible so he made transparent textures, this worked.

Countdown
Countdown is made with the new UI system. Using Unity editor and scripting.

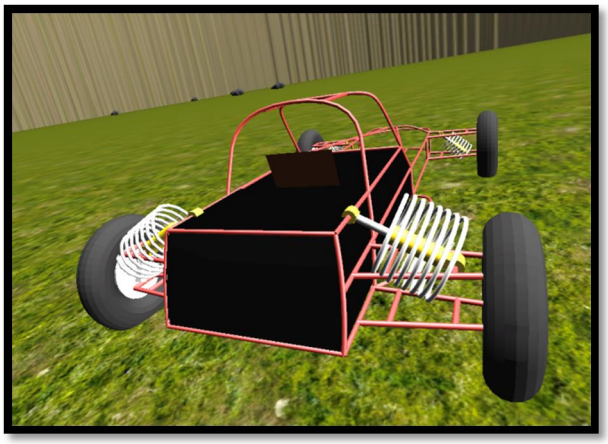
Race against the clock
All mini games have a time limit implemented in their script.

FPS independent
This was used in synchronising car position.

Code quality
Editor formatting plugins was used to maintain one style. The variable and function names were carefully chosen to keep a good overview.

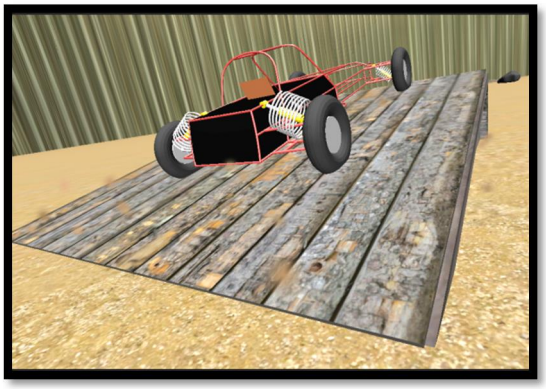
We frequently worked on the same code. This did not give us any problems, because we used GitHub persistently, loads of commits.

Art
Only the sounds were downloaded externally.



What: Car, player
Made by: Rick Koster

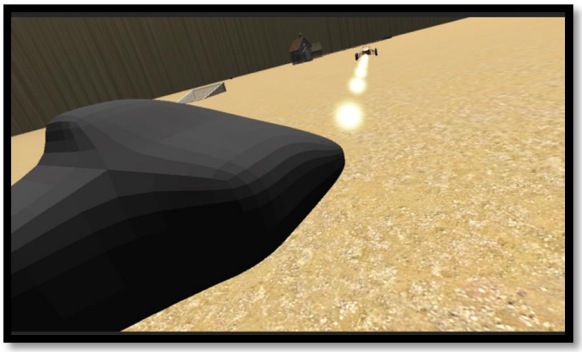
What: Church, arena asset
Made by: Rick Koster



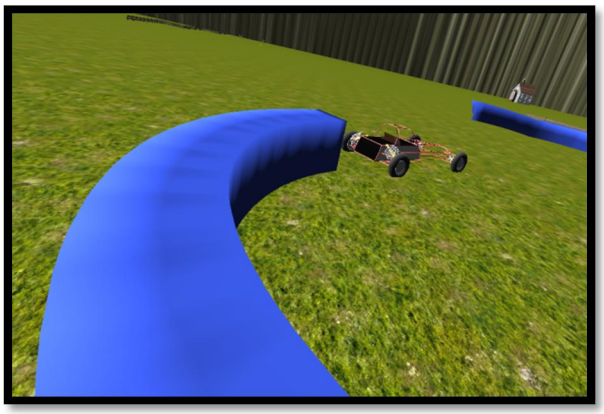
What: Barn, arena asset
Made by: Rick Koster

What: Ramp, arena asset
Made by: Rick Koster

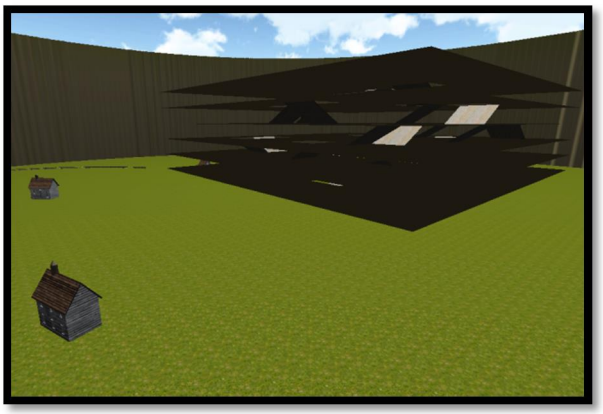
What: CheckPoint
Model made by: Rick Koster
Particle system made by: Jan-Cees van Senden



What: Enemy, zombie mini game
Made by: Jasper Binda



What: Line, tron mini game
Made by: Joseph Verburg



What: Tower, race to top mini game
Model made by: Jan-Cees van Senden
Textures made by: Rick Koster

Process

We used Scrum to plan our issues every week but because we could not find a proper velocity we stopped using time estimates after 3 weeks because a lot of unforeseen issues would disrupt our time planning.

Conclusion

Some of the goals we set at the start of the development of the game turned out to be unreachable. For example, the team based multiplayer was a core idea for the mini-games, but we simply didn't have the time to implement this feature.

Working as a group went well. Using issues within GitHub allowed us to set goals each week. Dividing them amongst the team members made sure everyone did their part. Also, keeping track of all the changes that were made to scripts, prefabs and so on was relatively easy using Git.

It was a set-back that we could not use a lot of external tools. This meant that we wasted a lot of time on trivial things or simply could not implement features. For example, this resulted in a lot of problems with controlling and animating the car.

We set out to create a fast paced multi-player game. In the end we think we have succeeded.