Final report for IMT3601 Game Programming.

Group: Single Rainbow Unicorns.
GitHub: https://github.com/Tezar/IMT3601
Members:

Morten Møller Andersen
Studnr: 080215
Github id: BoMann

Magnus Duun
Studnr: 100229
Github id: HerrDoink

Aleš Tomeček
Studnr: 120284
Github id: Tezar

Individual assessment:

Morten:

Report:

Things i worked on: Network, Engine, Particle engine, Menus, Gameplay and Documentation.

Network

So we had decided on making a server, that will handle all the multiplayer parts of our game, and itself only having a text ui. So one of the reasons we landed on using irrlicht was that we found it having a good and easy implementation of networking.
The one we used is called irrNetLite.
Following the tutorials and examples from that helped me make our network part, or at least what we have before we decided to focus on single player so we had a product to show at the end of the project.
What we have atm is; a server able to recognize new clients, and that can receive and send packets to the clients. Clients can create lobbies (not join them) and the server saves them and the clients displays all active lobbies.
The way it works atm, is keywords in packets of text, that is sent from/too the server and client. Each side have its own way of handling the packets and the information in them.
Rest of the multiplayer part planed for the game got put on hold (cut) because of the lack of time.

Engine

The parts of the engine i worked on was the reset function, and also the gameplay checks (will talk about those under gameplay).
So our reset functions job is to put all the vehicles on a common start line, and add spacing so it looks good. and an important part of it is to turn the cars so they face towards the next waypoint they need to move too.
Spent a lot of time getting the math here right, but as it normally is the simplest is often the best.
The problems was with rotating the cars right, so i ended up just making a base vector, then finding the vector from where i'm spawning to where i'm going, then using arc cos on

the dot product between the base vector and the new vector, this gives me the angle of rotation in radians and i simply rotated each car the same degree. Was proud because i found an simple way to do this.

Particle Engine

The Irrlicht have a really good particle engine built in so i did not have to make one, but i did get it to work after following some tutorials.
Our plan for using it was simple to add exhaust to the cars, add particles to the weapons (that got cut) and to show when the vehicles crash.
But what we instead ended up using it for was a visual indicator of the waypoints. So we are able to see the waypoints and then use them to test all the other parts of the code, that uses waypoints (A.I and gameplay parts).

Menus

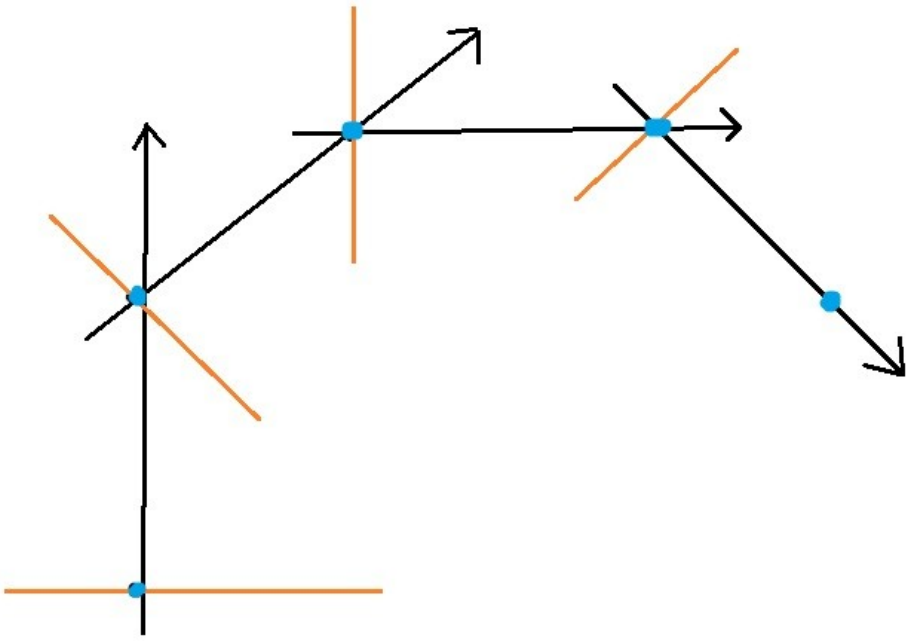
Now Irrlicht have a lot of base functions to make and use Menu Ui-elements (windows, buttons, list boxes, sliders etc). So what i made was our placeholders for all the menu's (config, settings hotkeys etc), also made the receiver that handles when / if we push buttons and the functions of all the buttons.
Under multiplayer i'm using a listbox to show all the active lobbies that the player is able to join.
Made and added a custom Font (Irrlicht uses images of a font to create the letters).

Documentation

I functioned as the secretary of the meetings and the keeper of the design document, so i had the job of keeping it up to date and note changes to it.

Gameplay

Now after some time we had our graphic engine, physics engine, and everything we needed of handlers of events, the camera working, we can drive the cars and everything of the base frame work of a game is ready, but we did not have a "game" (why are we driving? where are we going? what is the point? etc is what makes a game). So i started working on a way to make some kind of "track" so we had somewhere to race.
Now this was done by adding waypoints, that i formed into a makeshift track (simple loop made of 13 waypoints). So why drive to/past the points? Well we added a condition that the cars all had a point whey wanted to get to (the waypoint after the last waypoint they passed), that way we had an direction and a goal for the players.
Now why care? Because if you fall too far behind the leading car, you die! (turn of the driving of that car), and when all others have died the lead car gets a Point, and all the dead cars lose a point (everyone start at 5 points, first to 10 wins and you can't get lower then 0 points).
So we had our gameplay, but how did we figure out who was leading and when cars have passed waypoints? MATHS off course!



Blue Points = waypoints, black is the pathing of the track from waypoint to waypoint.
Now the interesting parts are the orange ones.
Simple i know the points, so a make an vector from the point you are going to (next waypoint) and the one after that. and then finding the vector that is perpendicular to that first vector (based in the next waypoint). I simply take the position of the car and see if it is in front or behind that vector (draw a vector from center to the car and see if the angle is positive or negative). If it is positive, the car is after the waypoint and it gets a new waypoint to get too.
Now then we have a way of finding how close you are to passing a waypoint, we can use this to find who is the closest to passing and so we get the leading car. And using the lead car we get a position that we can measure distance too from the other cars, to figure out how far behind the rest is. and if they are way too behind we simply stop rendering them and “kill” the car.

Summarize
Simply by fail and error, with some help from strangers on the net i was able to make more than i had dreamed of at the start of the project, and also we as a group had made more than i was hoping we were able to (realistically speaking , no matter what we wrote in the initial design document) at the start of the course.
Just looking through the code helped me learned a lot about how physic engines / graphic engines work, frame work and the rest gets together to form a game.
Personally i feel i know have a greater understanding for what really goes into making a functional game (even if we didn't really do it).
Feel that with abit more time (just by seeing what we got done the time we only had this project to work on) we might be able to make everything we hoped for at the start.
One important thing that the project also teached me a lot about was time handling (we/me are really bad at that atm). So hoping the next project will be a lot better (especially since it the bachelor oppgave).
All in all it was a great experience and helped a lot in showing what the future (hopefully) will bring.

Magnus:

Report:

Things I have worked on: AI, Audio, Controller, Vehicle model.

When I started making the AI I wanted to see if I could make it without reading any guides or tutorials. It worked, to a degree, but it ended up just driving off the map. I used a method not using look direction, but instead used the location on the last frame and location on the current frame to decide if the car was heading towards a waypoint, and if it was not it would(atleast, was supposed to) turn.
Since it did not work, with some help from Ales, we implemented the look direction and made it turn based on that. It still does not work the way we want it to, but it requires far less code.
For the Audio, I downloaded and implemented IrrKlang. I simply followed the tutorial on how to implement and all that, and then scoured the remaining tutorials for clues on how to do the rest, as needed in our project. It's still not fully incorporated in all the correct places, but it does play sound and if we had proper sound files to use it will be easy to add and play them. The tricky part would be to make them "blend" together without getting several instances of the same sound and just making a whole lot of noise. Still it should not be too much of a problem. I did look at a lot of site on the net for free to use sounds that we could have used, but I never found anything that sounded like what we needed. It didn't occur to me until much later that I could have simply grabbed a microphone and made the sounds myself.
The controller class was originally made for only 1 player on 1 machine. When we first started, we discovered that Irrlicht could only handle 1 event receiver, so an event manager had to be made. When I looked around on how that could be done, I discovered a post made by someone on the Irrlicht forums that had already made one. So I simply used that and added what we needed to our project. Credit has been given to that user in the EventManager.hpp file, along with the link to the post itself. Our project now can handle as many event receivers as needed. The 2nd player on the same machine was fairly simple to add, just needed some extra buttons.
The model for our cars isn't strictly programming related, more 3D modeling. I created the chassis for the car and the wheels as separate objects. I also made a UV-mapping that will apply a texture on the car in the game if someone wants the car to have different colors. The wheels have their own image file if they want to be altered as well.
To summarize, I've learned quite a bit while researching and writing these parts of the code. Some realizations have come too late to do anything about. I feel the 3D Modeling class has worked to our benefit in this project. I started this project with very little knowledge in these areas, but now I understand them a lot better(though some things might still need more work).

Aleš:

Report:

Aside from being only-on-paper lead, my work is primary in the physics engine implementation, track generation, graphic representation, overall design and VS solution configuration/maintenance. I also provided ad hoc support where needed.

The beginning of our project wasn't exactly excellent. One of our team members dropped after two weeks and told us only one day ahead. The second problem was chosen programming language - C++ - which I don't really fancy.
My work in early stages of the project consisted of research on several frameworks (to the final selection only Clanlib and Irrlicht make it thru) and designing basic flow of program. Later in the project I worked on track generation and physics implementation. To rapidly prototype various track algorithms I used Python, since it's one of the languages I am most comfortable with. In the end, we didn't use it due to several complications that are described in design document. In the mid-project we did quite important change and switched most of the resources to be in external files. Partly to speed up development and partly because data-driven design is one of the fields I wanted to explore more.
For the final part of the project my focus was on creating game editor that would allows us to create tracks more swiftly, because coding XML files by hand wasn't really appealing idea. Due to my consultation work on other parts of the project, it isn't finished.
As for the problems, I think the project could have went better. Primarily our resources distribution was little chaotic and the design plan had to be changed several times for reason we haven't foreseen - missing member, Irrlicht loading system, physics/network coupling. Fot that we could do hardly anything else than giving more margins to each tasks we created. But things like unnecessarily developing network code (since we haven't got time to properly implement it anyway) and problems with track generation could and should have been coped with. Hopefully next time.
However in the end, our project was quite ambitious and contrary to the evidence we did quite well. The most significant outcome for me is experience in team work (and that I really don't like being in leading position), Visual Studio voodoo with solutions and physics implementation. And last but not least, that CLI version of GIT is superior to any other.