**CGD- Personal Diary**

Alex Baden-Hinsley - 10025367

**25/09/2013**

The assignment was set. Since it’s team-based, we had to get into teams.

My team consists of people I was in a team with for a Play & Games assignment last year: Anthony Lee, Mathu Watts and Robert Cale.

We were given a short amount of time to think of a team name. We found it hard to agree, so I chose the name during the last thirty seconds: Gecko Faction. I don’t know if the lecturers heard me wrong or if they simply thought it’d sound good, but the name was officially listed as Gecko Faction Games.

After the teams were decided, we were then asked to brainstorm ideas for the assignment.

The assignment is to make a game named Retro Rocket Rampage, with somewhat strict guidelines. Controls are limited to three buttons per player, the game genre is racing, with randomly generated cars, steered by rockets.

We created a google document for our ideas, as I don’t own Microsoft Office, and alternatives can cause compatibility issues.

We listed different types of terrain for our game to feature, such as the surface of the sun, or a bouncy castle. I also suggested gameplay should revolve around staying within a certain area (king of the hill) rather than crossing the finish line first.

Our group started making a presentation for next week, where we’ll be pitching our ideas to our lecturers Simon and Andy. I created a simple mock-up screenshot of the gameplay in Maya, with four cars on-screen and a designated area to try to reach.



The two bars next to each player’s name represent the amount of booster fuel they have, which lets them unleash their main rocket to gain speed dramatically, and the amount of pressure built up in their engine, which causes them to fly further when they crash.

**02/10/2013**

All groups presented their initial ideas to Andy and Simon.

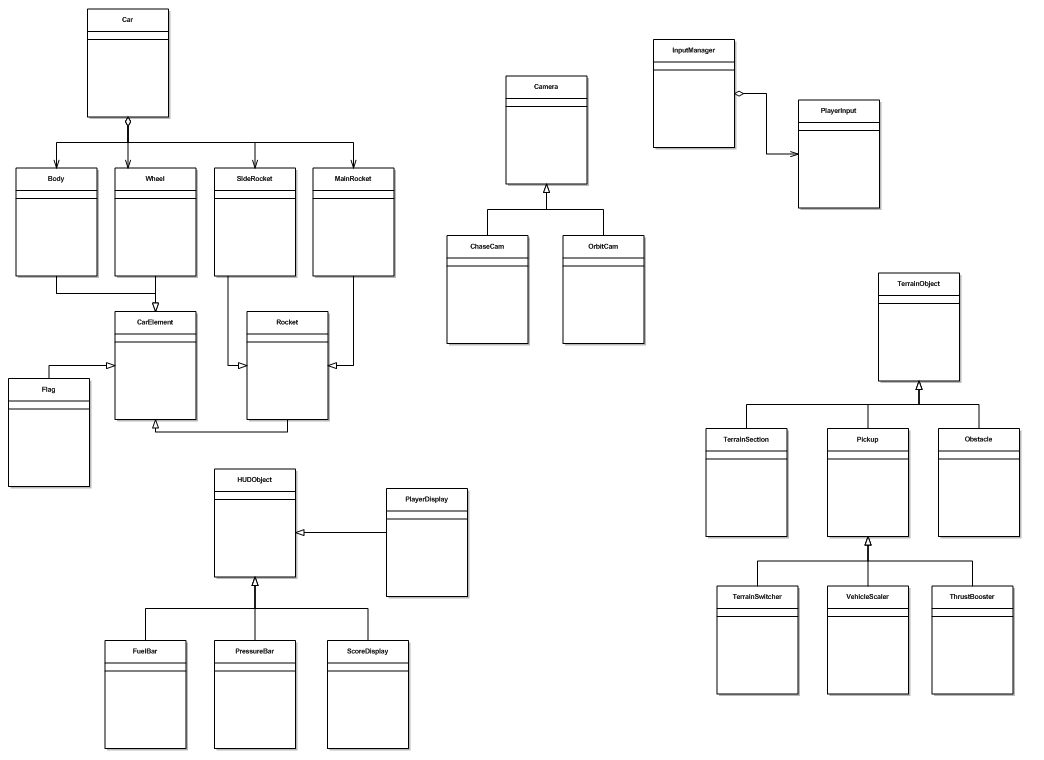
Andy told us the King of the Hill mechanic may be too difficult to implement without disrupting the flow of gameplay. He liked what I said about the different terrain types though, possibly because of the enthusiastic way I said it. We were encouraged to pursue a more “wacky” style of gameplay as our main focus.

The first thing we did after the presentation was try to identify a better way of scoring than King of the Hill that still encourages speed and chaos. As a group, we came up with the idea of a “Capture the Flag” mechanic, where one car holds a flag which grants them points over time. Other cars can steal the flag by crashing into them, encouraging players to crash, and also to try to speed ahead if they have the flag, increasing the risk of crashing into obstacles. The first player to cross the finish line would be granted enough points to win, but a player that had been holding the flag long enough could end up with more points, stealing the lead.

In order to maximize productivity, we decided on specific aspects for each member to develop. Anthony is to work on the car and gameplay physics, Mathu will work on procedural terrain generation, Robert will be focusing on the user interface design, and my job is to work on the random car generation.

**09/10/2013**

This week, we began work on the initial codebase. As a group, we decided on the class structure, and I created a simple class diagram using NClass to follow.



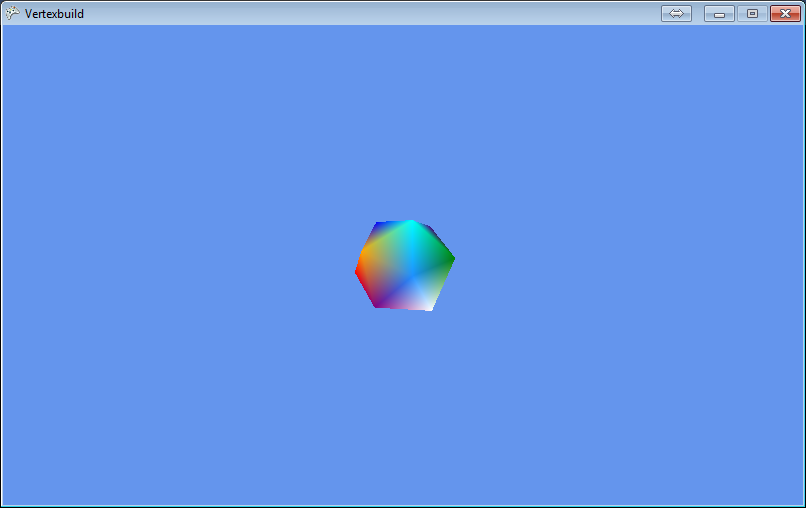
**16/10/2013**

In this week’s session, we familiarized ourselves with the group SVN repository, and with TortoiseSVN’s functionality. We made the first few commits to the repo, trying to get the hang of updating, committing and reverting.

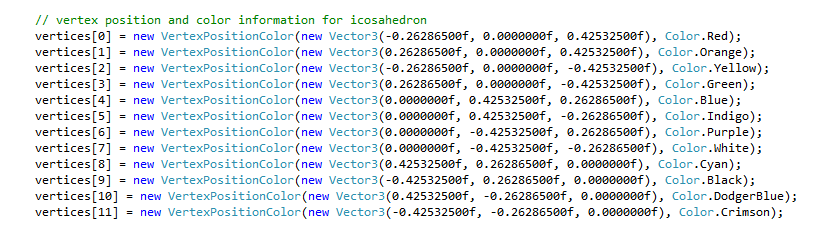
**23/10/2013**

Anthony and Mathu got a basic program working with a simple user-controlled camera and two 3D objects. Since we were told groups were expected to have models built from the vertex up, I decided I needed to learn how XNA handles building models from vertices.

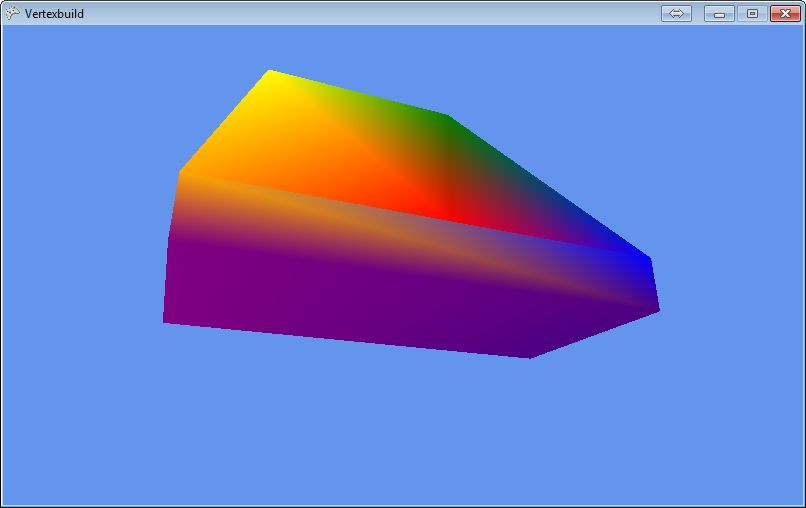
Firstly, I downloaded a simple application from MSDN, with a dodecahedron rotating in the center of the screen.



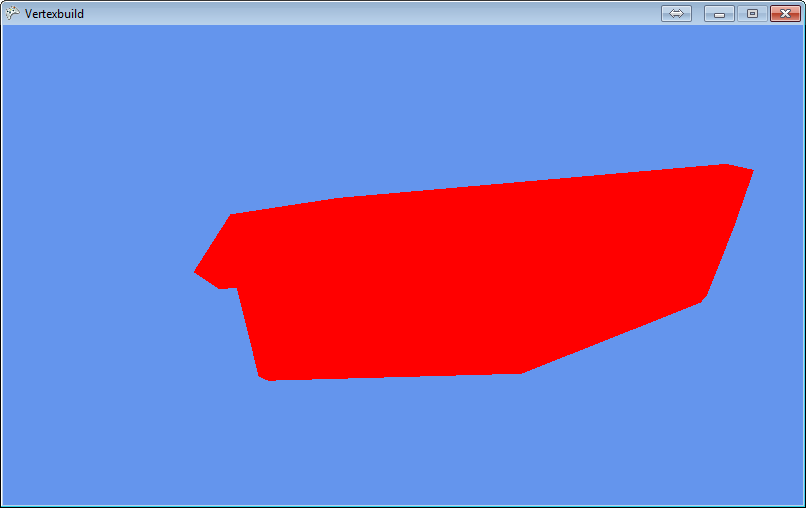
The dodecahedron is built within the application by specifying each vertex (corner) in the object, and how to connect them. The end product I wanted to achieve was a class in our game project which could load vertices and indices from a text file and create a model from them. To do that, I first identified the code which holds the vertices:



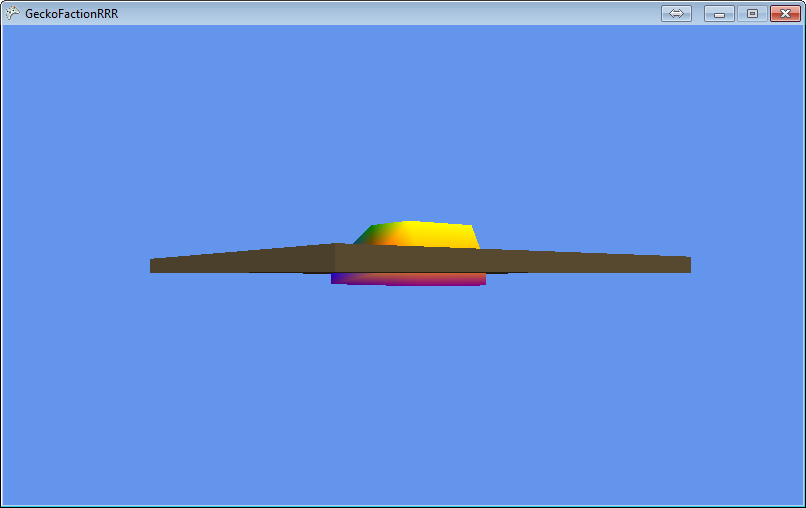
Then I experimented by changing a few values, making sure the changes were reflected when the program ran, eventually forming the approximate shape of a car.



I was then able to make a function which reads from a text file as vertex values indicated by the letter V, then as indices, indicated as I. The first model I created this way was a simple boat shape.



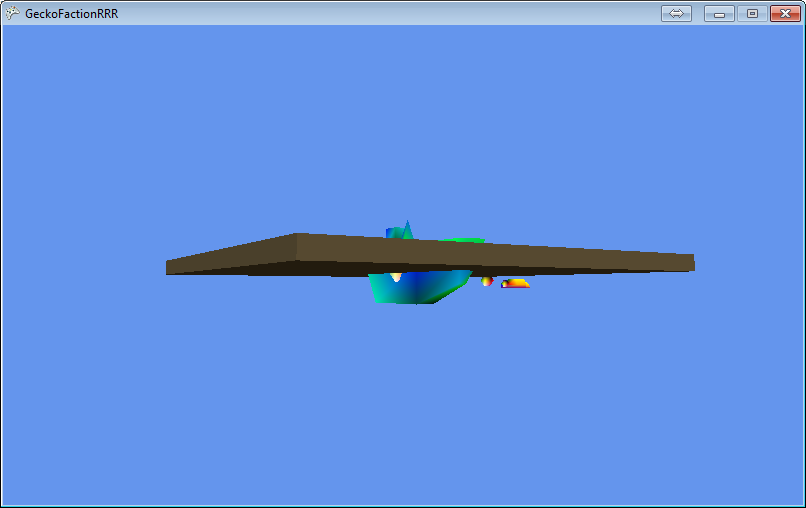
I was satisfied with this, so I incorporated this method of model loading into our game project, and made a model file of the car shape I made earlier. There were no errors, and the program worked as expected.



This shows the car model (read from a text file) intersecting with a cuboid loaded from an FBX file. The camera can be moved around to look at the car from different angles. The function that loads in models from text files assigns each vertex a random colour, leading to a strange rainbow-like colour scheme.

**30/10/2013**

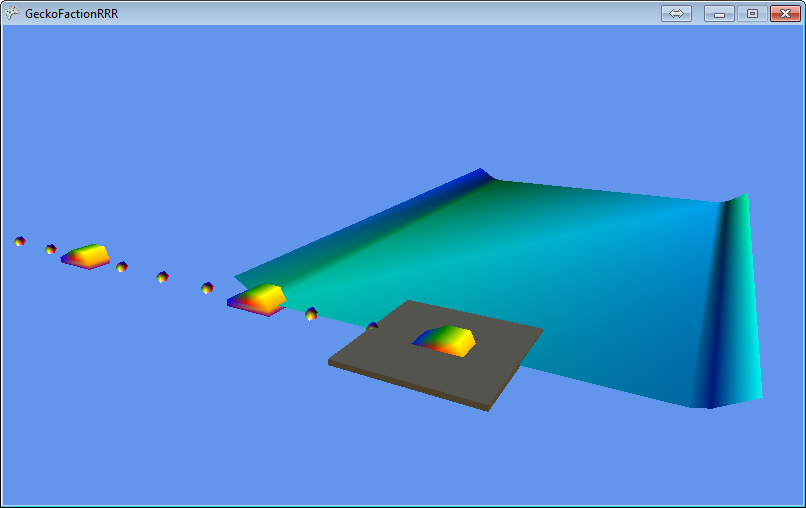
I worked a bit more on my boat model, which is based loosely on the Japanese 1942 Shimakaze Super Destroyer, adding a communication tower and something of a control room. I then helped Mathu with a function that spawns objects in sequence a certain distance apart, eventually for use in track generation.



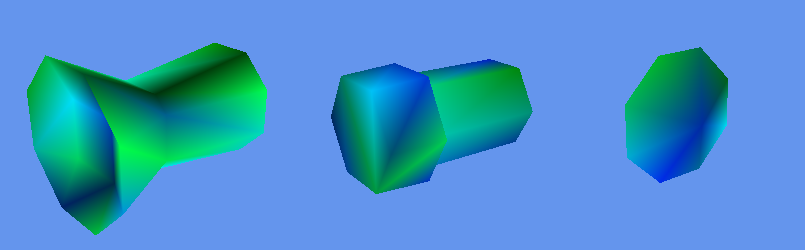
This picture shows the completed boat model intersecting with the FBX cuboid, with a line of car objects visible in the background.

**06/11/2013**

Initially, I worked with Matt to make a simple terrain surface model, which can be used for the track floor in the future.

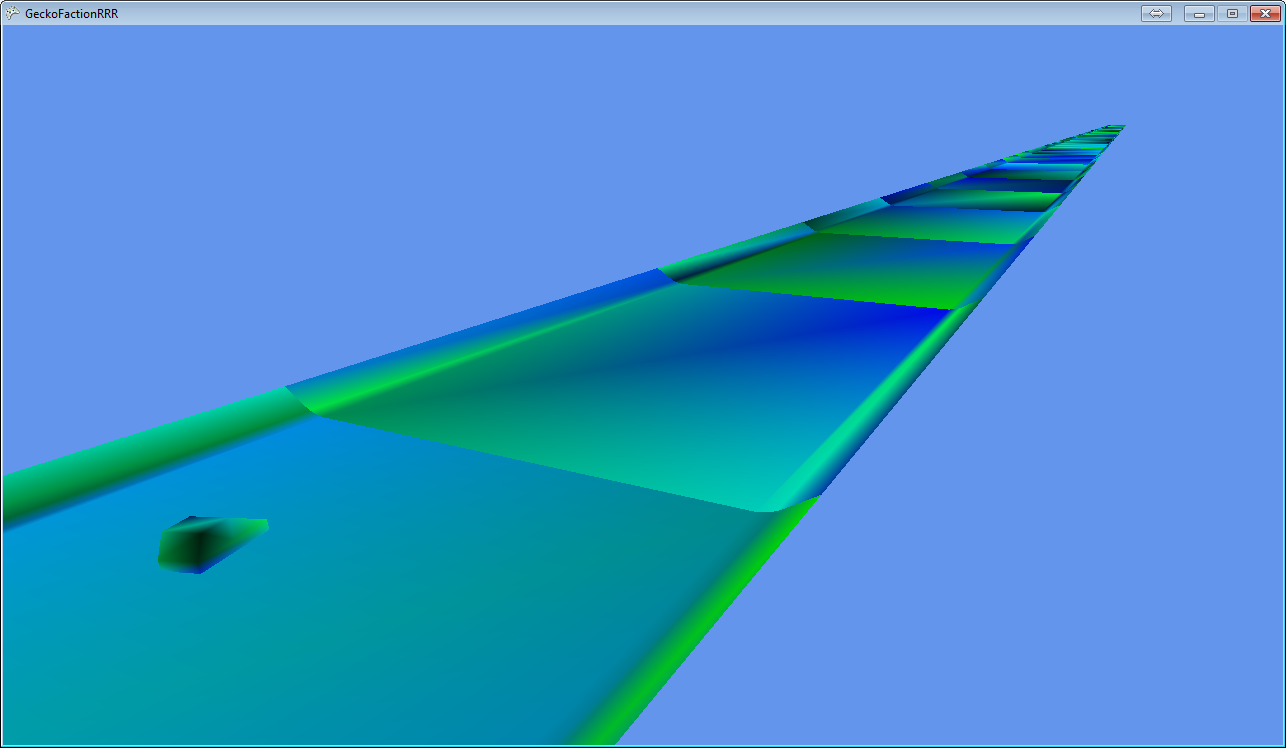


I also created three more models: a big rocket booster, a bolt-shaped booster and a thin wheel.



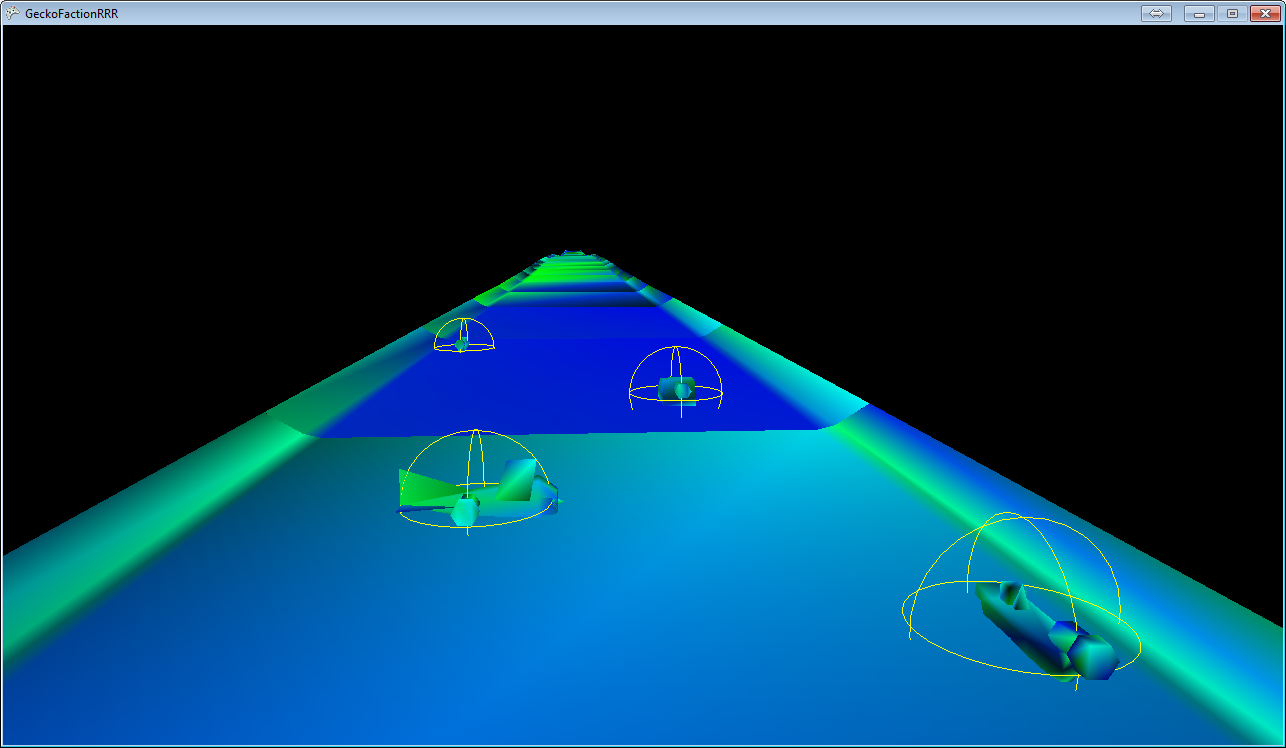
These are going to be among the available parts for car generation. Each car will consist of a main booster, a body, four wheels and two side boosters, each type having different stats which will affect how the car works.

**13/11/2013**



Mathu got terrain spawning properly, and Anthony made a controllable Car class, which rotates slightly as you bank sideways. I began to brainstorm possible designs for car parts, inviting the others to contribute as they see fit. Robert hasn’t contributed anything yet, so I asked him to try and make a main menu system.

**20/11/2013**



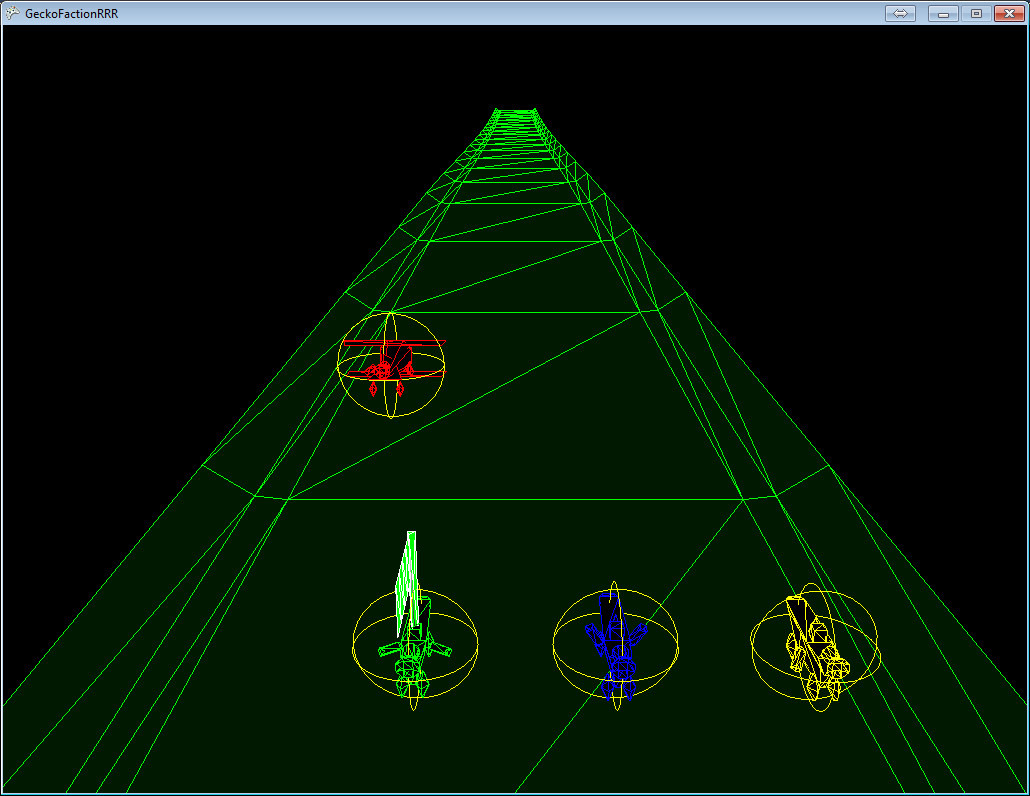
Anthony updated the game so all four players spawn vehicles, and each vehicle has rudimentary collision detection using a bounding sphere. I wrote a function to generate each car, randomly selecting from one of two boosters, and one of three bodies, including an oddly-orientated biplane model Mathu made. Anthony also updated the Chase Camera class to correctly follow the center point between all active vehicles. We later discovered that the X and Z coordinates were switched for most of our measurements, which was why Mathu’s plane appeared to aim to the right.

**27/11/2013**

We did a bit of work to correct the X and Z coordinates we’d been working with. I added a boolean to the model files, which indicates whether its X and Z are switched, and accommodates that when the vertices are built.

I updated the car generation function to include the side boosters and randomly selected wheels. The gradient visual style was never intended to be permanent, so I wrote a function to draw wireframes over a solid fill colour. This feels a lot more “retro” and is very aesthetically pleasing compared to the gradients we used before. Each player has their own wire and fill colour, so their car is always identifiable to them. Mathu created a flag object, which will transfer between cars when they crash.

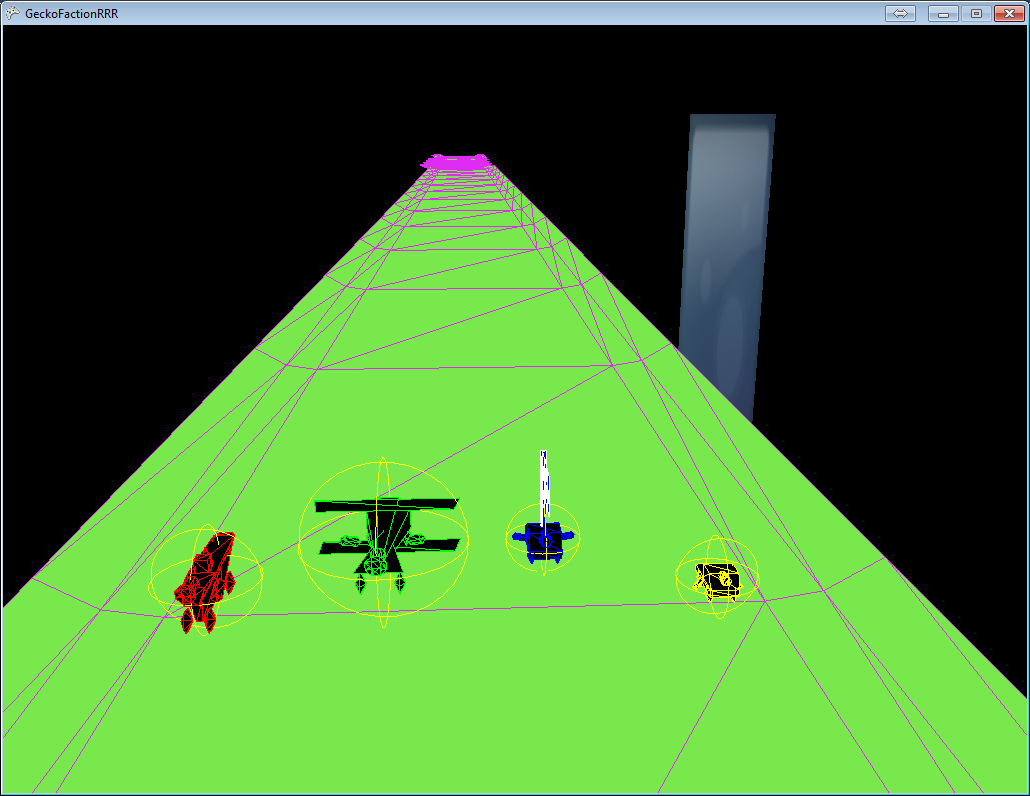
Anthony added collision detection to the track, so cars will now stay on top of it, bouncing slightly.



**04/12/2013**

We presented what we have so far. Reactions were as expected. Our track generation method was criticised, and we were told the game feels too slow.

I created a new class called Sprite3D to display 2D objects in 3D space. By default, the object will always face the camera, giving a similar impression to pseudo-3D games from the 32-bit era such as Galaxy Force and Outrun. This will be used for power-up objects as well as track scenery. I also added a function to change the colour of the track mid-game. This will allow us to change the “mood” for different terrain styles.



**11/12/2013**

We decided the next major goals are to re-do the track generation so that it doesn’t simply place models together, and to perfect vehicle physics to allow proper crashing and driving along a surface.

Over the Christmas holidays, I did some optimization on the Sprite3D class, adding a field to decide whether the object should face the camera or behave as a standard flat 3D object.

**22/01/2014**

I had a discussion with Anthony about how to handle the skybox. Since we’re going for a retro feel, I suggested we use a 2D image for the background and move it vertically to accommodate for the camera moving up and down, keeping the “horizon” where it would be expected to be. Anthony argued that a standard 3D sphere skybox would be more appropriate. We asked Simon for his opinion, and he said a 2D skybox would be nice, but it could be too hard to make sure the horizon moves properly, possibly breaking immersion.

**29/01/2014**

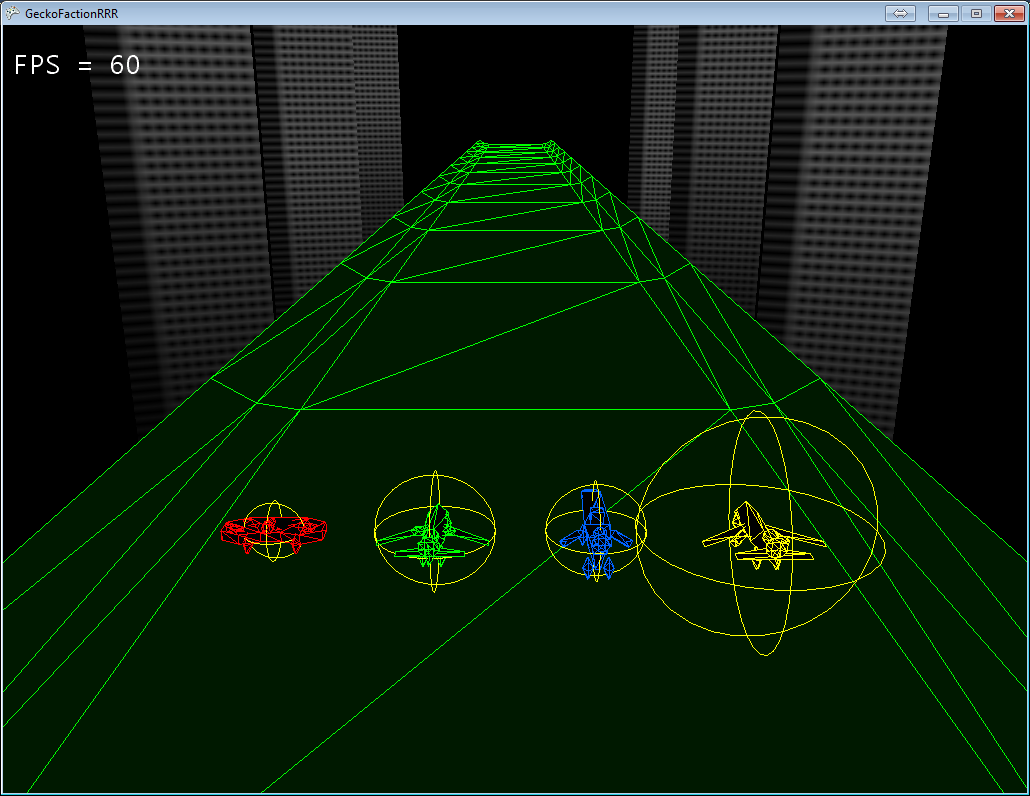
Robert committed some GUI code that couldn’t compile, so I went through that with Anthony to try and fix it.

**05/02/2014**

Anthony fixed the car physics, allowing them to drive smoothly along the track and bounce off each other when they collide. Robert’s code was completely unusable, to we tried to remove it.

I created two new car body models: A retro game joypad and a modern jet, which replaces Mathu’s biplane model. Each model file now contains its own size dimensions and centerpoint, as well as the stat for that model. The stat is stored as weight for bodies, boost strength for boosters, and stability for wheels, although they are not used in gameplay yet.

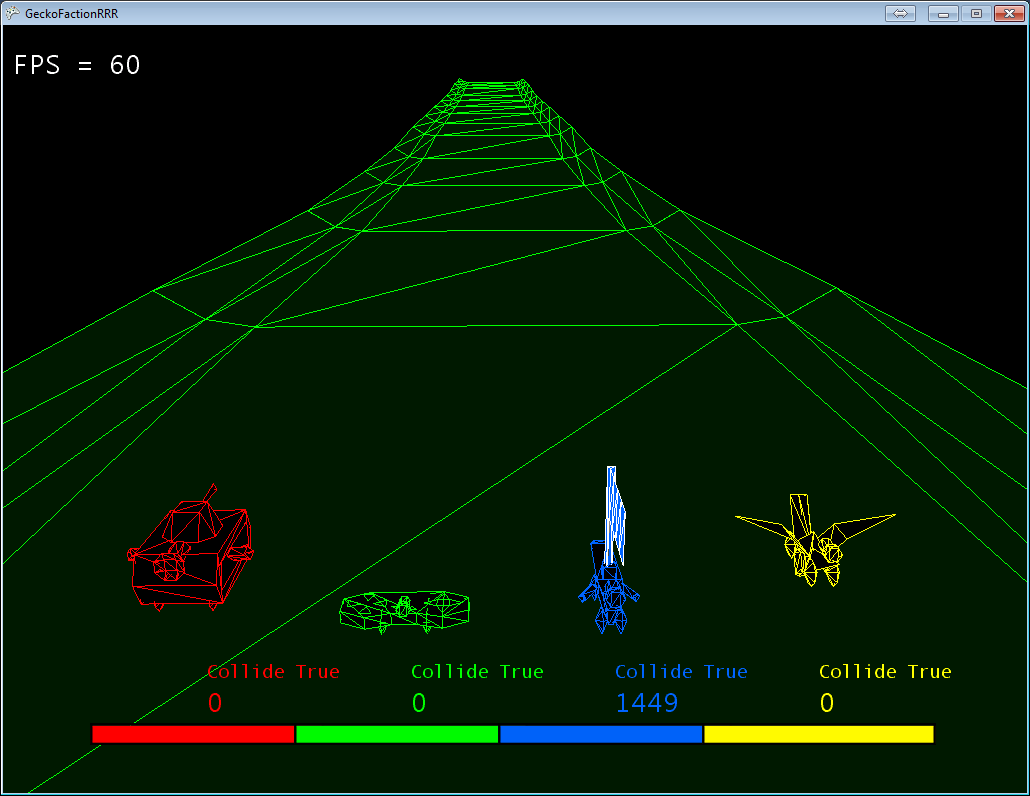
I also created a skyscraper sprite to use as scenery. A few test objects were placed either side of the track, giving a feel similar to Turbo Outrun and other arcade racers.



**12/02/2014**

I made another two body models: An origami crane and a panzer kampfwagen tank.

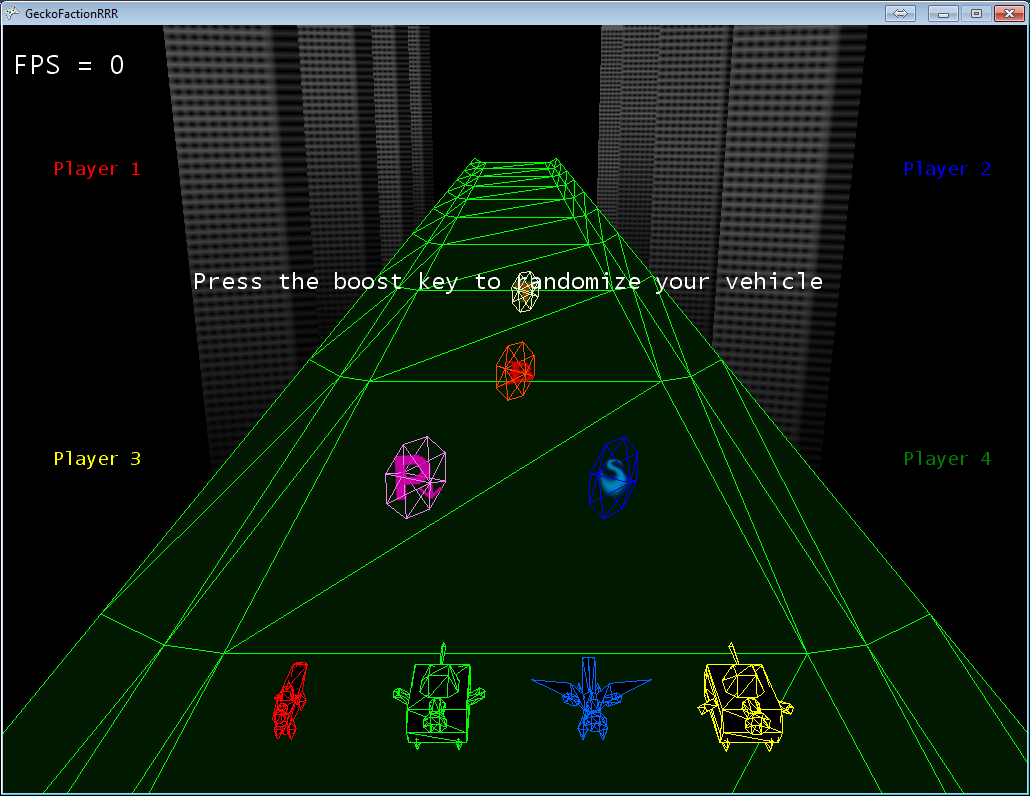
Mathu made HUD elements to handle players’ boost bars and their current amount of points, which are correctly increased when the player has the flag.



**19/02/2014**

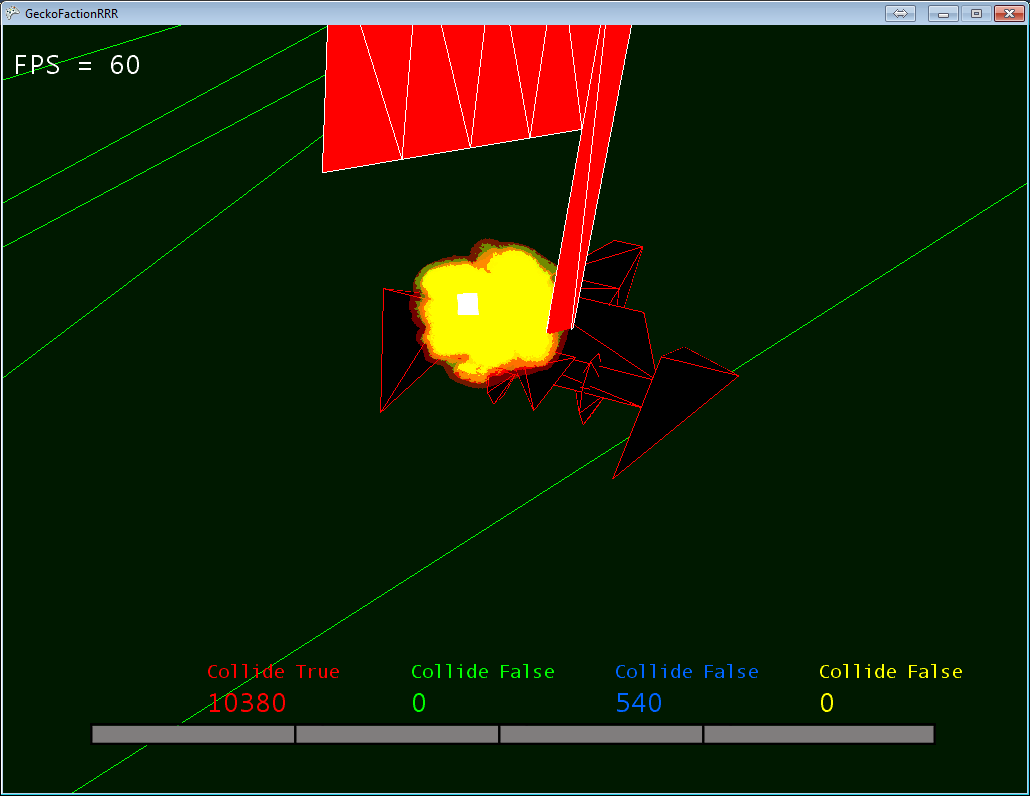
I made a body model in a shape similar to a sci-fi spaceship.

Matt added power-up objects using the Sprite3D class, so I helped him get the transparency working correctly.



**26/02/2014**

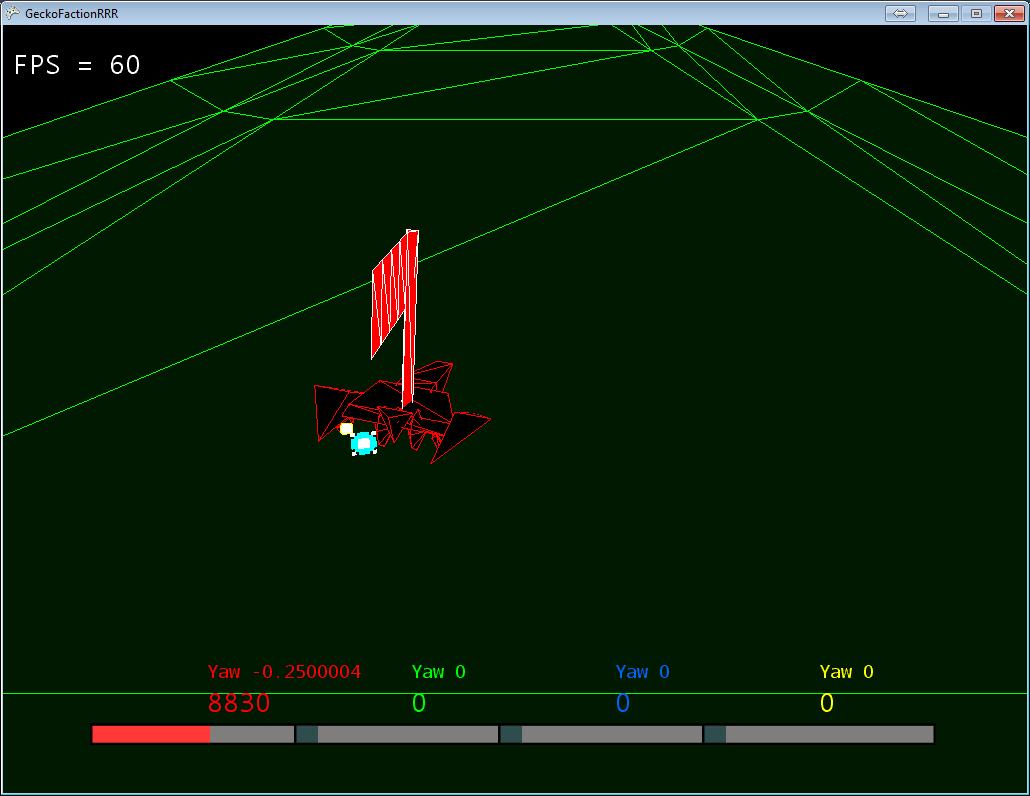
Anthony started work on an alternate terrain generation method which doesn’t involve loading models from files. Mathu added a simple particle system for side boosters. I adjusted the way boosters and wheels are placed on cars, but there are still issues with them being placed too far from the body.



**05/03/2014**

I changed the way boosters and wheels are placed. Body model files now contain offsets for the main booster, the left side booster and the front left wheel. I also created a particle sprite to use for the side boosters instead of the fire sprite Mathu included from a tutorial.

Steering was changed so applying the left or right side booster actually steers the car rather than pushing it along the X axis.



**12/03/2014**

As a group, we decided our boost mechanic should be reworked into a small but powerful shunt, so players can attempt to shunt others off the track.

The pressure mechanic was not scrapped, but instead it now occupies the same counter as boost fuel. Crashing increases your boost, but you risk flying further when hit if it isn’t used wisely.

We also decided to have a bomb hazard in the future, which causes the player holding it to lose points when it explodes, and changes hands by crashing, similar to the flag.

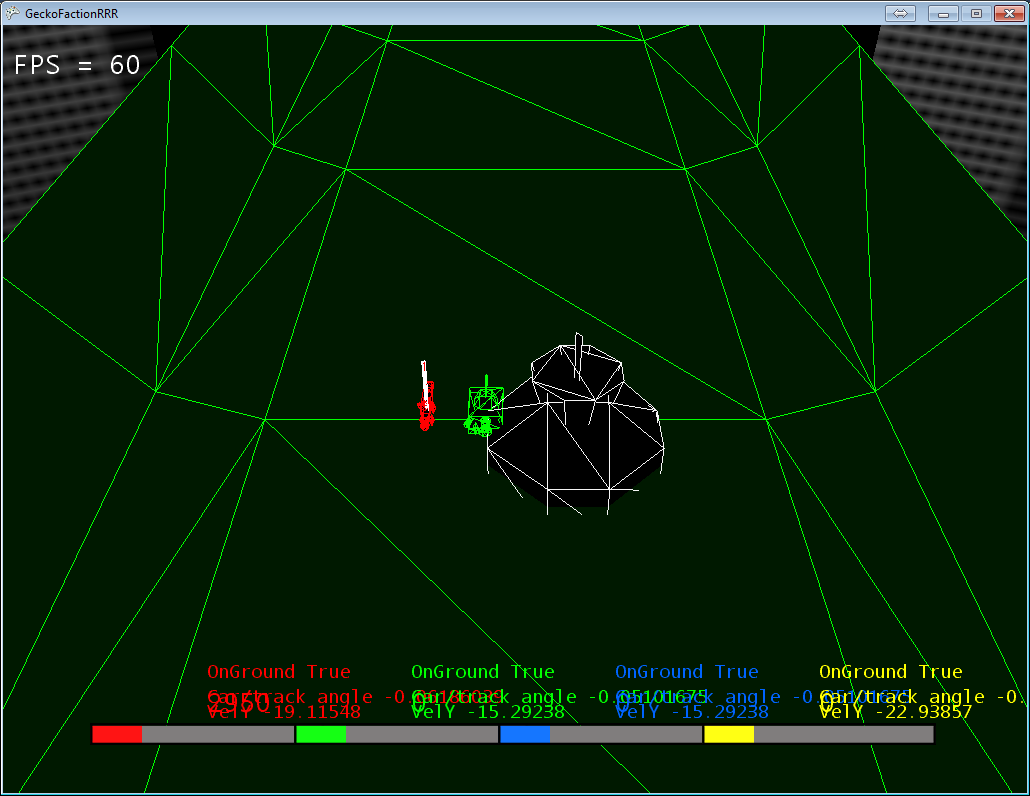
I made car stats more easy to access, so they can be used in physics calculations in the future.

**19/03/2014**

I added a model for the bomb, and Mathu made a class for it. It has basic functionality, including growing bigger over time before exploding (disappearing) and transferring to another player when you crash into them.

I also added a simple animation system to Sprite3D. Two texture files are defined along with a frame-rate, and the object will alternate between the two sequentially.

Anthony added his new terrain code, with working physics.



**26/03/2014**

I added a new main booster model in the shape of an elongated pyramid. I also added a feature to the model loading function that allows a model to define its default scale, so models that are made too large can be scaled down. There are no more weekly lab sessions after this.

**04/2014**

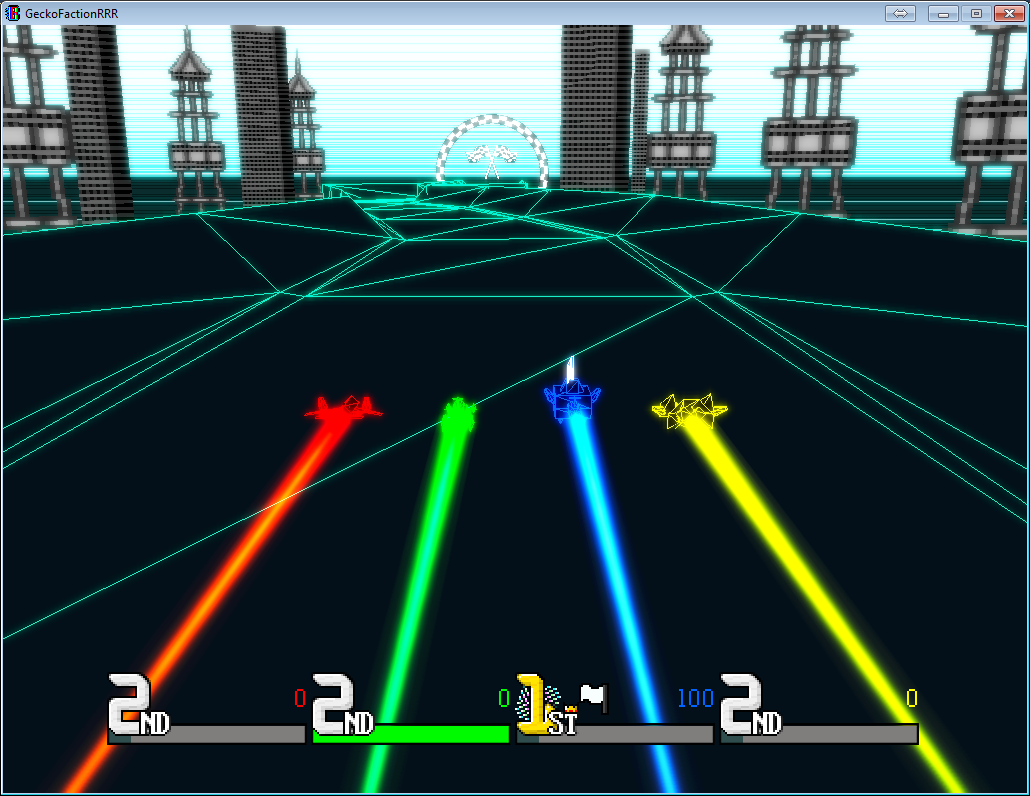
Mathu added a new booster model, and two new wheel models. Anthony continued to work on vehicle physics and I made a skybox texture which can be used for a 2D or 3D skybox.

**05/2014**

I created some more models: a booster, a wheel and a body resembling Bloodhound SSC.

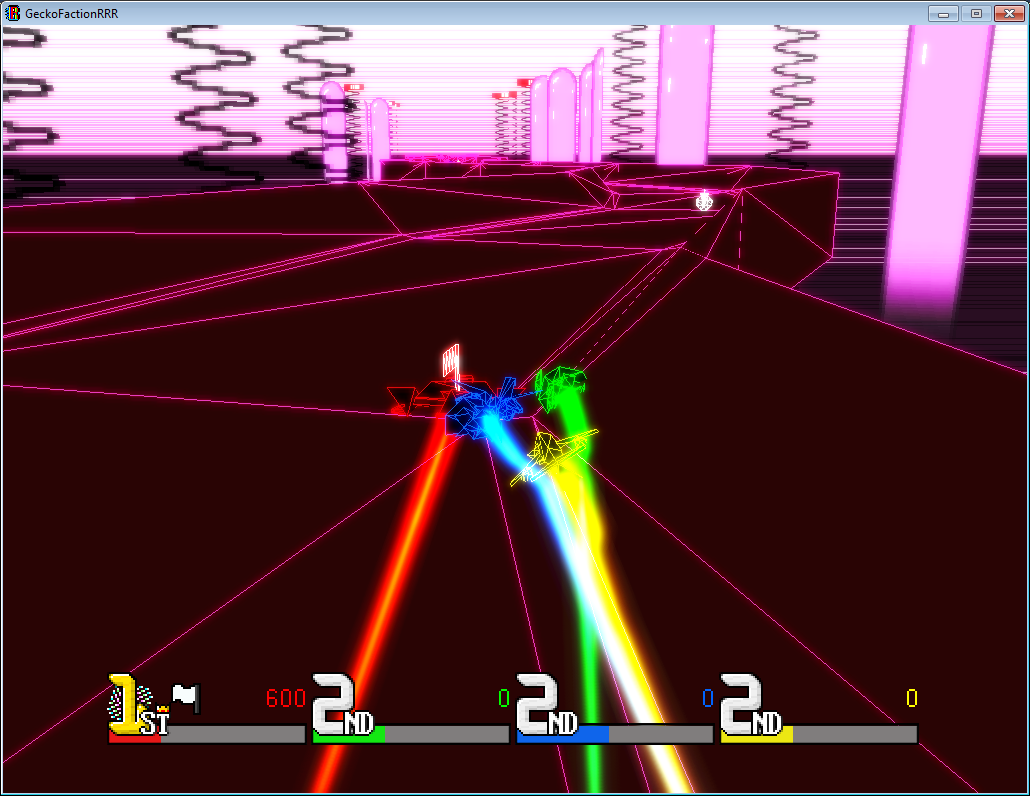
I created an icon for the game, as well as small sprites to indicate who has the flag and who has the bomb. I managed to get the 2D skybox texture working correctly, opting against converting it into a 3D sphere. The finished effect is quite impressive, and really captures the “fake 3D” retro feel.

Using an MSDN tutorial I imported a function to overlay a modified bloom shader onto the camera view, dramatically increasing the visual appeal of the game. I also added two scenery sprites for each of the four biome types, and a circular finish line sprite. Mathu added a new particle system for main boosters and side boosters, leaving trails not unlike those seen in the Tron movies.



I created a biome class to handle the changes in visuals when the terrain type is changed. They store track and sky colours, as well as a list of sprites to spawn on the outside of the track as scenery.

In the final hours, we implemented a working biome system, which affects the gameplay by filling everyone’s boost bar in the sun biome, and making the track surface bouncy in the bouncy biome. Anthony added a working track finish system, with a results screen. I attempted to fix a crashing issue when all four players are waiting to respawn at the same time, but I was unable to do so, as I am somewhat unfamiliar with the code that caused it.



That’s a wrap, for now.

**Personal Post-Mortem**

The problems I encountered in this project were due to my personal inexperience in collaborative programming. I found it hard to understand the code others had written, and I was unable to help out in certain areas as a result. On the other hand, graphical design for the project was enjoyable, and I’m proud of everything I contributed. In future projects I will try to pay more attention to the way other group members code, and attempt to contribute more to all areas, rather than a specific specialized subset.