## Week 1 (25/09/2013)

Introduced to assignment. Formed the team and picked a team name.

After we were given the assignment specifications we held a brainstorming session, dumping our ideas into an online shared document. We used Google Docs for this as not everybody in the group owned the Microsoft Office Suite at home.

Ideas discussed ranged from what kind of different terrain types we could include to specific numbers such as how long each race should last.

I looked at some games that featured game mechanics that might be of use to our assignment. Renegade Ops by Avalanche Studios, for example, applies a subtle screen shake every time a vehicle explodes to feedback to the player the impact of the explosion and carnage happening around them.

Another game I looked at was the Super Smash Bros series of fighting games by HAL Laboratory. In particular I looked at how the damage system is used in replacement of a more traditional health bar, a system where as the player takes more damage, the easier it is to knock them off the stage. This was a mechanic we decided to include in the final presentation to help our game stand out.

The other key game mechanic we decided to include to help us stand out was a King of the Hill system. The player, who spent the longest amount of cumulative time in the lead during the race by the end of the track, is the winner of the race. This mechanic was inspired by similar game modes in many competitive multiplayer first-person shooters such as Team Fortress 2 by Valve.

I summarised and typed up some of our consolidated ideas into a presentation format along with Alex Baden-Hinsley.

## Week 2 (02/10/2013)

Week of the presentation.

The Publisher was not too keen on the King of the Hill idea used for scoring as they felt it would have been too tricky to implement well. The more 'wacky' terrain types that we mentioned were well received though and we were told to focus on that aspect.

We brain stormed some alternative scoring systems. The idea with the most potential was to adapt the Capture the Flag game mode from many competitive multiplayer first-person shooters. A player can hold onto a flag object, this flag object can be stolen from them by other players if they are sent off the track or fall too far behind. The player, who holds the flag when the finish line is crossed, wins the race.

The Publisher suggested we include this along with the traditional "first person across the finish line wins" winning condition.

We split up the various development roles among our group.

I would be focusing on programming the physics and other general gameplay mechanics.  
Mathu Watts would be working on the terrain and the procedural generation of the terrain.  
Alex would look at generating the vehicles randomly and modelling the different car parts.  
Robert Cale would implement the user interface and menus.

## Week 3 (09/10/2013)

We discussed what classes were needed and what their properties are likely to be. This was noted down in a class diagram.

I started working on porting over code from my previous assignments to be used as the base of the game. Allowing us to start adding in the features of the game in a familiar environment.

This base code included game objects templates used to manipulate an object within the world, using some simple physics as well as a static camera so we can observe our changes.

Due to the requirement that our assignment be built from the vertex up, rather than using pre-existing assets, I started looking at how vertexes and indices are used to build shapes and how these techniques can be integrated into our current project.

While I found understanding how primitives are built to be intuitive enough, it was the latter problem that I struggled with. As nothing was showing up on the screen, it was hard to tell whether it was the shape drawing code that did not work correctly or whether it was the camera I added that was facing the wrong direction or was non-functional. Some rudimentary controls needed to be added to the camera.

For now I am drawing all the shapes with the BasicEffect shader included with the XNA package to save time. It would have been difficult for the rest of the team to start their work if we could not add and display objects at the very least.

## Week 5 (23/10/2013)

Started looking at implementing a more advanced physics system than what we currently have. I was planning to let any physics object to move with momentum. This momentum could then be used to calculate what would happen when two objects collide with each other.

I also started looking at the mathematics behind rigid physics and how objects will rotate when a force is applied to one side. Most of the stuff I found was not really suited for video games, as they were focused more on the accuracy of the simulation of physics as opposed to program performance.

I noticed I wasn't making that much progress so I decided to put off the physics for another day and start working on simpler things.

## Week 7 (06/11/2013)

I had started work on some basic game logic for the car handling.

Using methods to simulate the different forces the 3 rockets would impose on the car. These methods were set up so they can be easily used by the future control handling methods.

Adding a tilt to the car as it moves left and right looked surprisingly convincing. Really adds to the feel of handling the car.

I went through my team mates' code and refactored it, as there were many features of the game that were coded in large chunks of code. By separating their code into different modules, not only is it easier to read what a specific method does but there is now less duplicate code so the project is easier to maintain as a whole.

For example, I had moved Alex's code for loading in model files into the base game object classes so it was available to use for every object that had a model. I also went through and made sure there were fewer methods using hard coded numbers.

## Week 8 (13/11/2013)

I am starting to feel more comfortable now that we have a base code and some objects to work on top of.

Re-worked Mathu's terrain generator so that it wouldn't use redundant variables when the same information was already available elsewhere.

Started consolidating the experimental code we had written so far into proper classes now.

I put together a system that loads the game's settings in from an external settings file. I decided it was appropriate to add one in at this stage as it would help speed up development time in the future as we would not need to hunt down where a particular variable constant was stored. Example settings include the game's window resolution, the control scheme being used (whether keyboard or arcade) as well as the key bindings for the controls.

While C# does not natively support using .INI files for game settings, I went with a similar format as it is the format most players are familiar with. This is stored in a plain text file, labelled config.txt to avoid confusion.

Gamepads are now supported and can be used alongside the keyboard/arcade controls, to help avoid overcrowding during 4 player sessions.

## Week 9 (20/11/2013)

I added a basic chase camera so that we can now follow the cars further than 20 metres. While crude, it does the job well enough so that we can start implementing more features of the game without the restrictive static camera getting in the way.

We ran into a problem at this point with merging our code together. The non-functioning and defunct code chunks that I deleted had merged back in when Alex committed his version without paying attention when checking the conflicts, causing complier errors as they were not all merged back in intact. The lesson we learnt here was not to assume conflict errors can just be resolved by accepting any chunk of text that looks unfamiliar in the differences window.

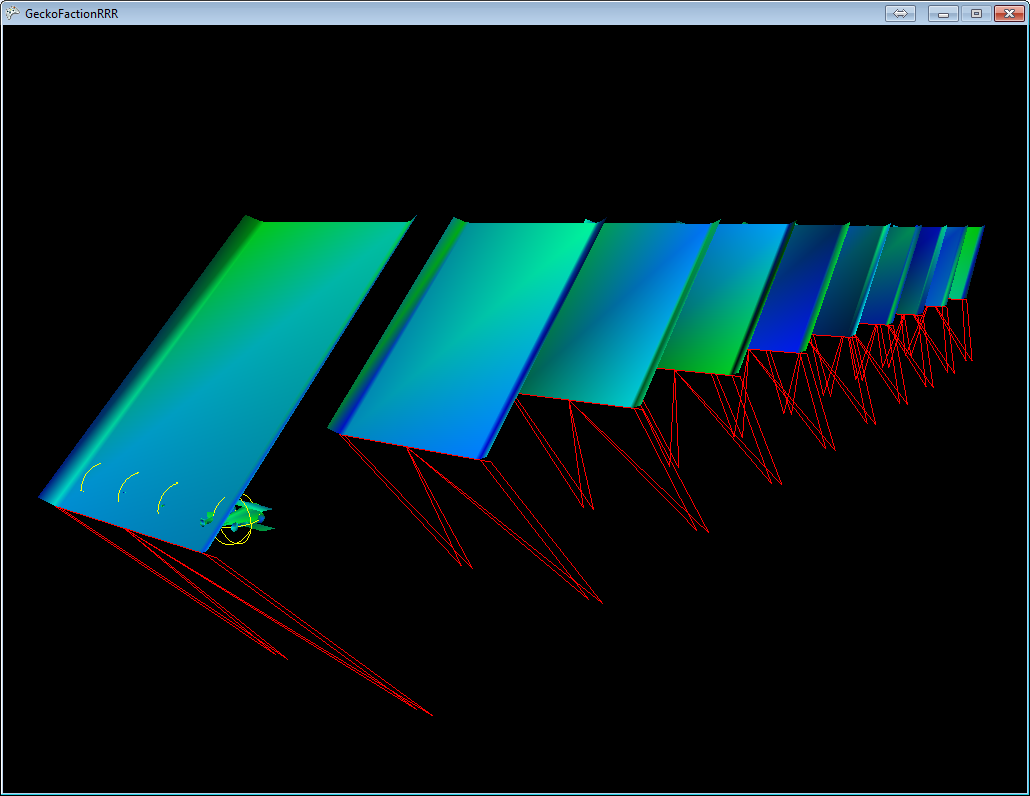
Started work on a collision detection system between cars. Starting with a bounding sphere around the entire car for now. Thinking of eventually adding addition bounding spheres to each car part, although depending on the final camera angles, it might not be necessary.

Found a class in the XNA documentation section of MSDN called DebugShapeRenderer. I'm using this to draw out the collision models for easier debugging.

Thinking against using bounding boxes for now, as they would need to be re-created every time the car turns around.

To test the collision detection works, I just move the car up for an obvious visual effect.

## Week 10 (27/11/2013)



Debugging terrain collision mesh

- Changed the pitch of the terrain pieces around to observe any effects as there was a 50% chance that I got the angle calculations the wrong way round.

Simon was displeased with our method of creating the track. While I agreed, we decided to work on changing that after the alpha presentation due to lack of time and the other things we needed to finish first.

Continued on implementing car collision models.

Initially, to move the bounding sphere, I just re-created the sphere every frame as just updating the sphere's velocity property wasn't doing anything. Figured this is probably inefficient.

Then I moved on to trying to add the car's velocity (after it had been converted to a matrix) to the bounding sphere's world matrix. The bounding sphere would still need to be re-created.

I will be using a loop in the game1's update loop to check whether each car has collided with another car/obstacle/the floor.

Started to figure out how to I would go about creating the collision mesh for the terrain.

One way would be to specify any bounding sphere/boxes in each terrain model file. This sounded tedious as at this point we did not know how many different track pieces we would have, and that the collision mesh would need to be manually edited whenever we change the models.

[**http://blog.goltergaul.de/2009/10/xna-directx-c-game-project-basic-collsion-detection/**](http://blog.goltergaul.de/2009/10/xna-directx-c-game-project-basic-collsion-detection/)

Looking at the above blog for inspiration, I planned on creating a plane for each triangle of the model.

Since I already have all the vertices from the model file, it was simple enough to create all the triangles by taking each set of 3 vertices and then updating them once the track piece has been put into position.

I didn't immediately realise but as seen in the above screenshot, the collision mesh doesn't actually draw out the correct shape. This was due to me creating the plane faces from the list of vertices, rather than the list of indices.

This collision mesh didn't actually allow for accurate detections as it turns out the planes weren't the exact triangles I created them out of. I was misled due to picture in the above blog link as well as the fact that plane's had a constructor that uses 3 points.

While I needed to keep the planes to hold the normal vector of the terrain's faces, I needed to use something else for the initial collision detection. For now, I decided to go with bounding boxes as they were fairly simple.

To create the boxes, I find the minimum and maximum point of a triangle and use those to create the box. I find these points using the Min and Max functions of the Vector3 class.

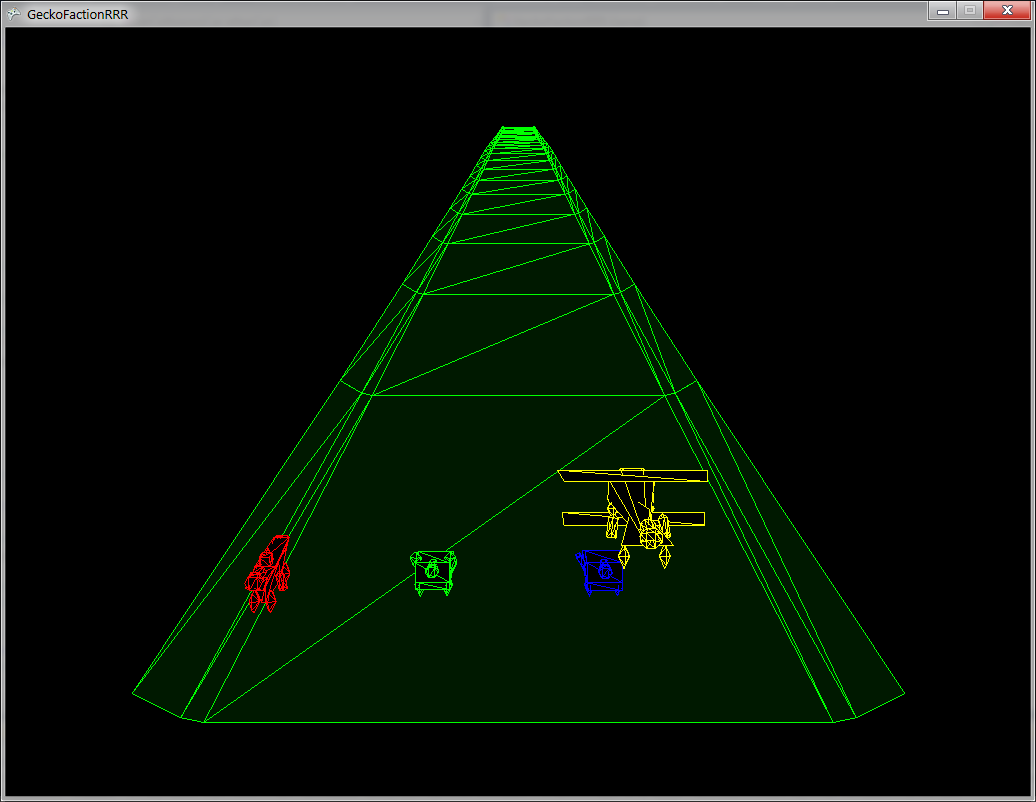
As I imagine most of our terrain models will use extensive use of rectangular faces, I wanted to reduce the creation of redundant bounding boxes as most faces would be made up of 2 triangles. As the collision checking time would increase exponentially (O(n^2)), reducing the amount of bounding boxes should help performance quite a bit.

I started off just checking whether the minimum and maximum points of the new to-be bounding box was the same as all the previous boxes of the model but this didn't reduce the number of created bounding boxes as I expected. I changed it so it checked whether the points were approximately the same as the previously created bounding boxes due to floating point imprecision. This dropped the number of bounding boxes created by over a half for the current track piece model.

To debug whether collision detection was working correctly, I reflected the car's velocity off the normal vector of the plane of the bounding box that was touched.

Bouncing off the side ramps is actually kind of fun. Increased the size of these side ramps in the model file to make it easier to get a long airtime.

All the collision checking so far doesn't seem to impact the performance significantly, thankfully. Although we still haven't solved the occasional stuttering bug. Group discussion speculated that it has something to do with our model file loading function. It's not something we can rework before the alpha presentation though.

Added a check to see whether an object's bounding sphere intersects with the camera frustum, to decide whether it should be drawn or not. "Eh, why not, easy enough to implement" was the reasoning at the time.

Each track piece's overall bounding sphere is also checked first, as a first pass, before its bounding boxes to reduce the number of collisions that needed to be checked.

Added in a sort of reset function as I felt it would help smooth out our alpha presentation. Adding this in highlighted some areas of the code that are "all over the place".

Tried playing around with the draw distance. Looks pretty cool when the race track looks like it is being drawn in as you go.

## Week 11 (04/12/2013)

Week of alpha presentations.

### Our predictions of our feedback was:

- Dislike of our "scalextric" style track building

- Dislike of our current placeholder models, as we're meant to be the wacky game of the year and all the placeholder models are cliche. Something that we've talked a lot about during our lunch breaks but I haven't mentioned it yet in the diary.

- Like the sense of speed we have going on (compared to what I saw of our fellow classmates during the sessions).

- Like our retro wireframe art style

### Actual feedback:

- As predicted but they made no comment on the speed. Although they did mention that our rocket handling was wrong, as we had assumed they wanted the cars to strafe around rather than turning.

- Suggested we go for a sprite look rather than a wireframe look. Which was surprisingly similar to a conversation I had with Alex a few nights before about using sprites for the track decoration. Such as how the palm trees are added here: <http://youtu.be/EZW8yglpvRY?t=4m27s>

- Said we should add a fluttering animation to our flag. The liked how comically big the flag was.

- They wanted the bouncing to be more exaggerating for wacky gameplay. Even though I did mention I only added the bouncing for debugging collision detection and verifying the normal vectors of the track's faces.

### Comments on the other group's presentations

Almost everybody went for a large landscape filled with hills look for their terrain generation. While it's too early to make any fair judgement, I honestly think it looks really bad.

We agreed as a group that when we rework our track procedural generation, that we would keep our track floating in the sky, F-Zero, look.

Only two other groups had fast moving vehicles.

One of them did not implement a skip button for the initial vehicle picker screen. I don't know how they stay sane during development. They were also using pre-made fbx models for their vehicles; they seemed pretty detailed for a placeholder.

The other of the two groups, the one trying to go for a motherboard race track look, had a nice vehicle handling feel to it. The way their lighting was highlighting the slopes and peaks of their motherboard made it look like Minecraft.

The group working in DirectX instead of XNA had their game menu in the actual world itself. I think it would've been better if they drew the menu using a separate camera so they don't have to worry about positioning of the menu and camera in the world when they start working on a proper race track and starting line.

The one group that had a day and night cycle was pretty neat.

## Week 12 (11/12/2013)

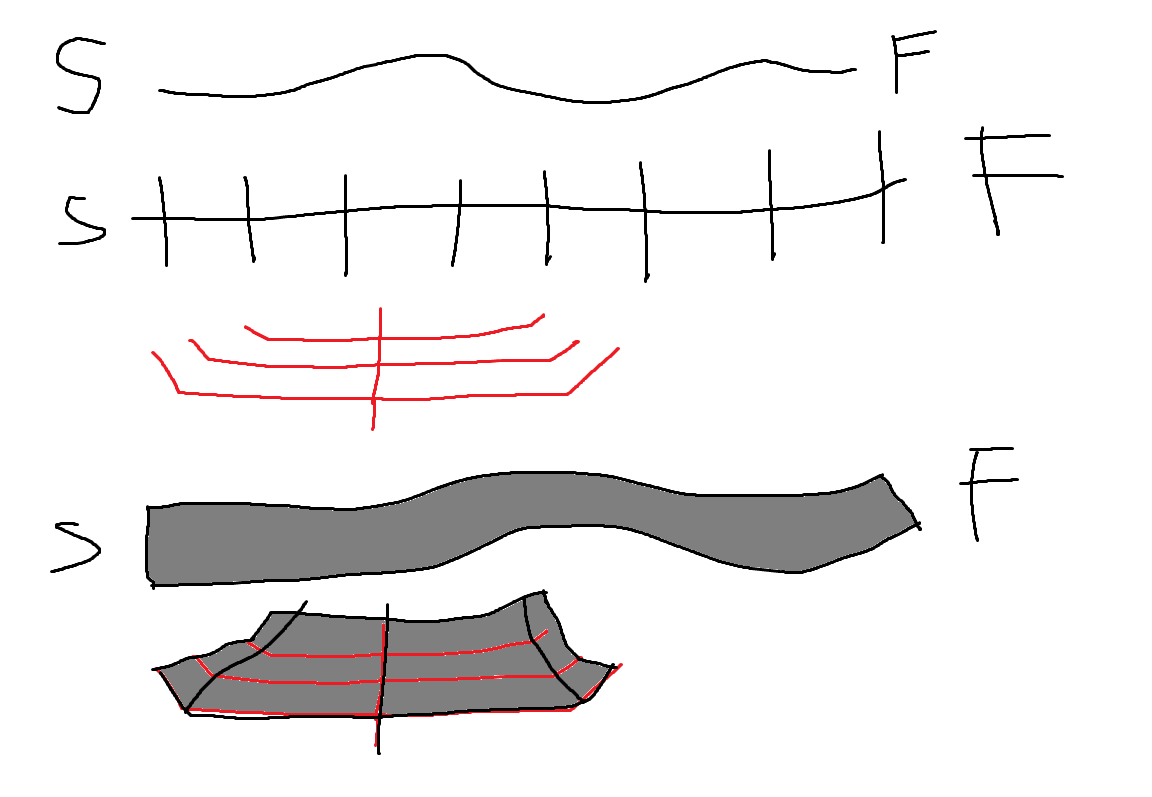
Andy announced that some groups might get an Oculus Rift or a Novint Falcon controller. As having personally tried both previously, I don't think either is particularly desirable to have for our game.

A VR headset would not make much sense as the game is multiplayer and we're not striving for full immersion.

The Novint Falcon could be used to provide physical feedback when a player's car collides with something but the whole thing is still pretty gimmicky and doesn't necessarily lend to a more enjoyable experience.

Planned out as a group what we should focus on next.

Terrain generation  
- Redone  
- Perhaps, procedurally draw out a line, that represents the track flow. Then widen that line to create the road. Similar to how I created my roads for my Scripting 3D track generator last year.

- Or perhaps, to pre-define a line that represents the road's width. Duplicate that along the track flow line and interpolate between them to fill them out. Carina suggested taking into account the banking of the road as it curves around.

Front view

Front view

### Model Loading

- The way we load the model files in needs to be reworked. For one, we should not have to read in the files for every object, we should look into some kind of instancing, or at least hold the vertices/indices information for each model object so we're not reading from the file every time.

- Hopefully reworking it will fix the bug of the models completely breaking when the game has been reset a few times.

### Car Assets

- Less cliche

-Baby pram (with rockets attached to it)  
-A blimp (with rockets attached to it)  
-An office chair (with rockets attached to it)  
-Ice cream Van, with a giant plastic ice cream cone on top, with a rockets attached to it.

### HUD

- Add some of the game UI elements in, to at least help debug the game when we start implementing the gameplay mechanics in.

### Vehicle Generation

- Right now, each car part has its stats associated with it in the model file but the code does not take it into account right now. Such as car weight, rocket booster strength etc.

## Christmas holidays

*"Ugh, who wrote this stupid piece of code? Oh wait, it was me"* - Anthony Lee, 2014.

## Week 13 (22/01/2014)

A member was missing this week.

Alex suggested that we use a sky-box technique similar to how the old 2D games used to do it. Where the sky-box was just a 2D image that scrolled left or right infinitely, perhaps layered using parallax for more detail.   
As seen here in Mario Kart: <https://www.youtube.com/watch?v=AlAmXXNz5ac>

While the idea seemed like it would be easy to implement, I felt that the modern way might still be more appropriate. We checked with Simon and he explained that with the old way, we would have trouble consistently aligning the sky-box with the horizon as we had a 3D camera.

Looked into re-working our track generation as it did not satisfy the procedural requirement.

Decided to go with the first idea that I had previously.

To generate a curve that defines direction of the track, duplicate that curve and offset it to create the outline of the track and then create the track model from those outline curves.

To create the model I would probably use a triangle list as all the curves should have the same number of points, meaning it would be simple to imagine how the triangle list would be ordered and it would be more efficient than using triangles.

Found this article that exposed some more details I might want to take into account such as adding splines to the direction curve. <http://gamedev.stackexchange.com/questions/25280/in-very-simple-3d-racing-games-how-are-collisions-handled>

Simon was nodding along as I explained this to him.

## Week 14 (29/01/2014)

We had some trouble with Robert committing unfinished, never actually compiled, code to the repository. I had to do a quick fix before any of the other members downloaded this version. Some asset files were also missing so I just used some placeholders to keep the complier happy.

<http://www.peroxide.dk/papers/collision/collision.pdf>

This week I started working towards implementing proper track collision, primarily using the techniques mentioned in the above paper.

I started off by just checking intersections between the track's planes and the bounding sphere of the car. As opposed to the previous implementation where I was checking against the track's bounding boxes.

From there I calculate the distance between the car and the plane, and use that distance to push the car back so it is just on top of the track.

Then I projected the car's current velocity onto the plane, to find a new velocity that is parallel to the plane.

After having tried to implement proper track collision on and off for a few months, I had finally figured out why all my previous attempts failed. The first pass for collision detection had a (several months old) logic bug that meant the function would not always return a collision detected when it should. This was because I had used bounding spheres for the track, which of course would not cover the corners of the track.

I settled with just checking the planes of the track for the track collision detection as I could not finish proper collision detection in time for the presentation. Although I started looking into the maths behind sphere intersections with triangles as possibly the next collision pass after plan intersection detections.

This meant that the car could not leave the road surface as the planes stretch on indefinitely. If the car would try to move off, collision detection would be detected anyway and the car would be set back down into the plane.

Also, as I combine the velocity with the gravity force before projecting the velocity onto the plane, it meant that the cars had a particularly tough time moving up the slopes on the side of the track.

However, the current solution works quite well, the car moves along the track smoothly. It is so much nicer than the previous debug implementations with the car juddering, getting stuck in the track and shooting off at a really high speed etc.

Even though it ended up as just a couple dozen lines of code, it required so much time spent into reading around the subject, reading different interpretations of the same topics to help me understand how everything actually works. I could actually add this paragraph into every week entry of this diary.

## Week 15 (05/02/2014)

Quick Presentation week.

There was an even wider range in progress between groups than before.

Surprisingly we came 4th in the year group in terms of total number of repository commits. I had initially guessed we were in the middle-lower half. Although it was no surprise who the top 3 groups were, just judging by the quality of their work so far.

Felt that the presentation went smoother this time round as we did not leave anything important out. The frame rate drops also seemed to have magically disappeared recently though we're not too sure why.

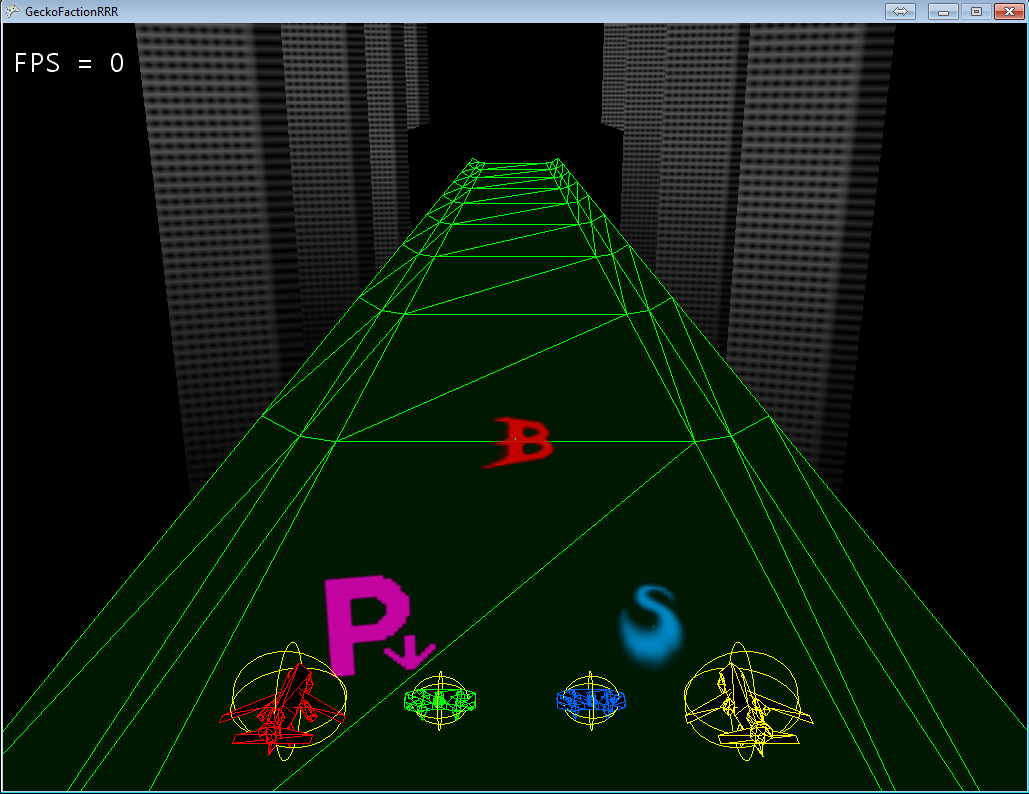
Not much feedback, basically just an "Ok, good" from Simon.

The new models that Alex had added looks slick. Although we both agreed that it was actually quite hard to find interesting ideas that would still be identifiable in the wireframe art style.

I had a quick chat with the group that is going for the motherboard art style about their method for track collisions. It looked like they are doing checks for each wheel, then applying their distances through the floor into a quaternion.

Did some thinking about how we might change our car handling to be more accurate. Perhaps use a forward vector and just change the angle of the car when the side rockets are used.

I looked into creating rotation matrixes from an axis and angle to accomplish this. I had some trouble getting this matrix to actually rotate the car model. While the solutions Simon suggested weren't what I was looking for, watching his process of debugging actually led me to consider parts of the code or other classes that I would have otherwise thought were irrelevant.

The actual bug was that I was applying the rotation matrix after I had already rotated all the car parts down the hierarchy.

Current game.

Power ups have started to be added.

Sprites added for the background.

## Week 16 (12/02/2014)

I mainly spent this week cleaning up bugs and working on the power ups with another team member.

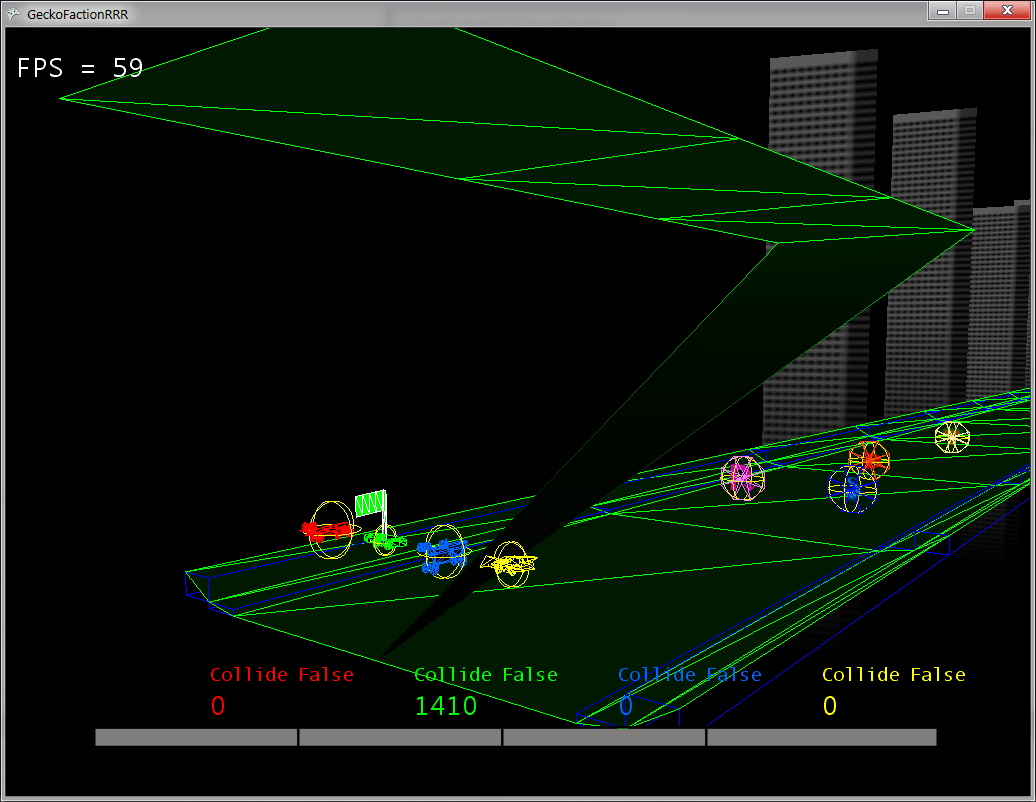
I had a chat with one of the guys from team Content Moose. Turns out they were using a third party physics engine, which explains why the physics and vehicle handling in their game looked so polished.

I had a look around a few available physics libraries including BEPU Physics, the engine Content Moose are using. After playing around with the starting tutorials, I've found BEPU Physics makes it very simple to get a world with collisions and vehicle physics running. However, using any of these engines would require rewriting most of our current code base, so we decided as a group to pursue a more arcade-y feel. As BEPU Physics is open source I'll still be studying its code though.

## Week 17 (19/02/2014)

For a change of pace, I started coding the new procedural terrain.

As I thought about how I would go about building the model from a list of points, I studied the existing code for model building. I ended up rewriting most of it, splitting off the long function into several smaller ones or simplifying the algorithms, as I found a few cases of unnecessary loops, duplicate code etc.



Screenshot shows the new procedural track being built above the old pre-made model track.

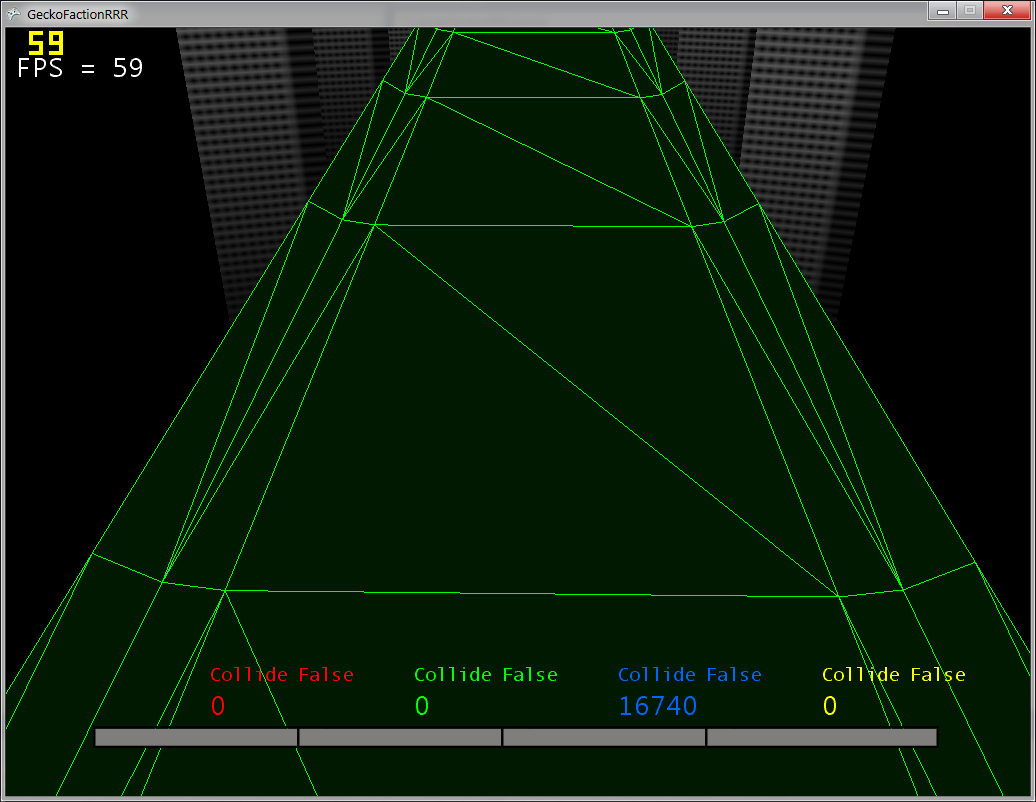
The screenshot also shows the bug that led me to re-work the code further with the draw methods.  
The bug was that the draw code was trying to draw primitives (in this case, the triangles) that didn't exist. I.e. The "number of primitives" parameter in the GraphicsDevice.DrawIndexedPrimitives method was set to the number of indices. I only noticed it now as it didn't have a visible effect on our previous models which were built with triangle lists, whereas I'm using a triangle strip for the new track.  
It also turns out that all the car models were being drawn twice every frame.

## Week 18 (26/02/2014)

With the re-work of old code out of the way, I continued building up the track model to at least match the pre-made model for now.

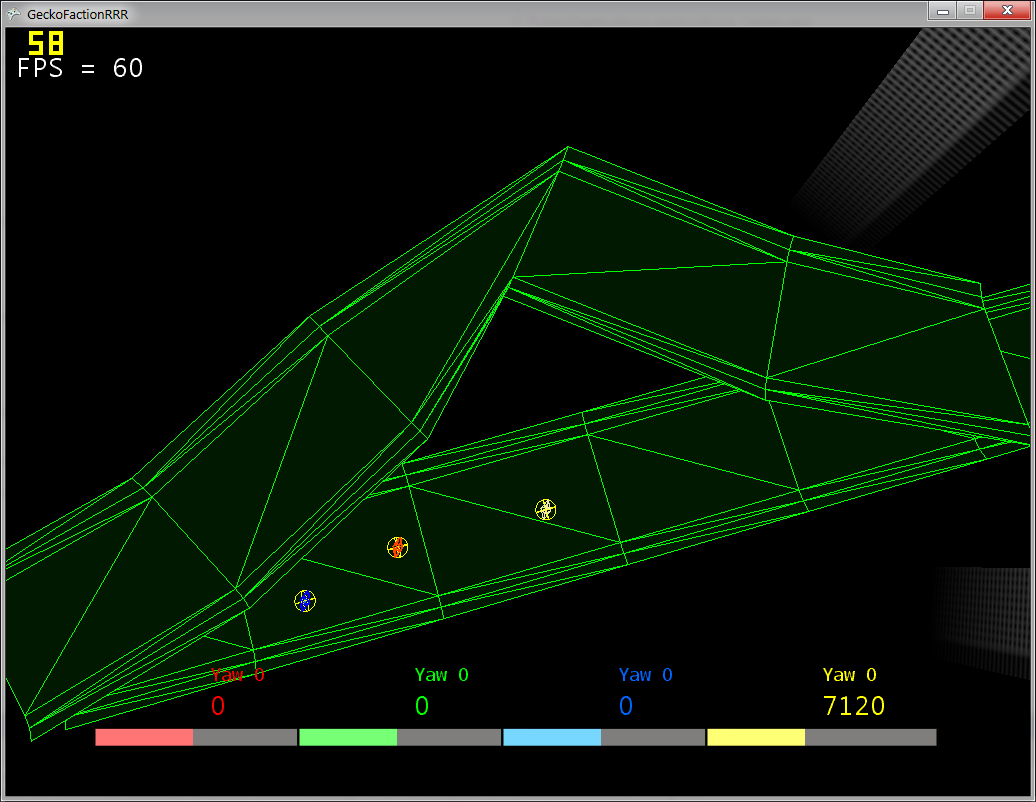
I started off adding only one piece to the track at a time to make sure all the steps of the code worked as intended. Here I noticed something weird, when I added the leftRamp piece, it appeared on the right side.

The Z axis is backwards. I poked my team mates to show them, we all then proceeded to list out all the instances where we had to make the X value negative because whatever we were working on came out backwards but we had never shared with the group until now. We now realise that these instances were all linked to the same bug, that "forward" in the Z axis is -1, not 1, which makes sense when you imagine a graph.



Since the triangle strips alternates drawing the vertices up the line then back down the next line, and the number of lines was even, I made it draw back over a line twice in order to keep the wireframe pattern symmetrical.

I had a look at some performance profilers out of curiosity.   
A few interesting things I found, were that the draw calls take up most of the processing time while all the update logic barely show up on the chart.   
Another interesting find was that the code to "check whether an object is on the screen before trying to draw it" was taking up more time than the actual draw calls.



I added the ability for the new track to curve by giving each point along the main curve a "forward direction vector" and using that vector to expand the sides out before creating the model. Although the track pinches really tightly if the turn is too steep, something to keep in mind when I randomly generate the curve shape in the future.

## Week 19 (05/03/2014)

Another 9am session means more bug fixing. There was a problem with the car collisions where they would occasionally get stuck inside other. This was because their velocities were just being reflected upon detection.  
The solution was to push the cars back so they no longer overlap before reflecting their velocities.

Fixed a bunch of bugs related to car parts being parented twice or being redundantly rotated even when they're already inheriting their parent's rotation.

The car's bounding spheres are now being moved around with (oldPosition - Position) rather than using the velocity to prevent them de-syncing with the models. This seems like the more reliable method as it looks like there is code somewhere that is moving the cars around outside their physics system.

I changed the car handling system, so they now turn like an actual car rather than just strafing side to side, we've put it off long enough. Although it does look like the cars are sliding around when one of them moves at an angle to the rest of the racers.

## 

## Week 20 (19/03/2014)

Robert’s commit in a while, he added a single comment to an already clear statement.

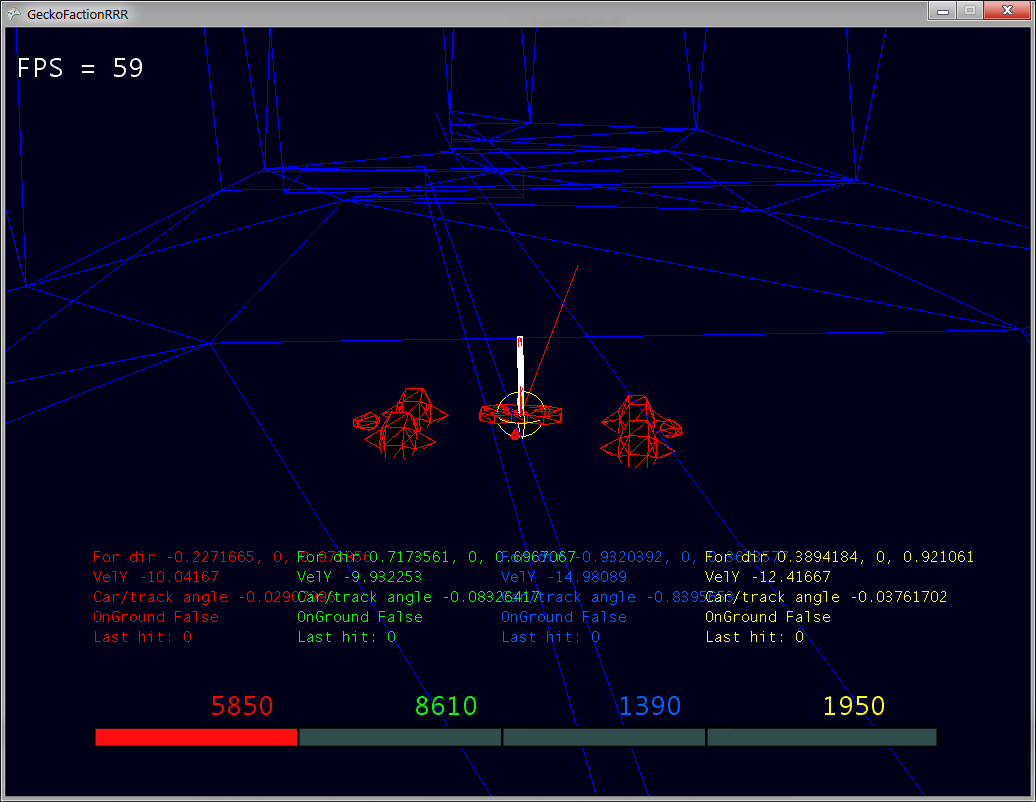
Continued work on track collision (how many months has it been now?) and it works well enough to stop using the old track entirely.

The collision response is split into bounce car and slide car, using an arbitrary value of Velocity Y and an arbitrary value of the angle between the car velocity and track to decide which one to use. It’s fake but it looks natural enough.

## Week 21 (30/04/2014)

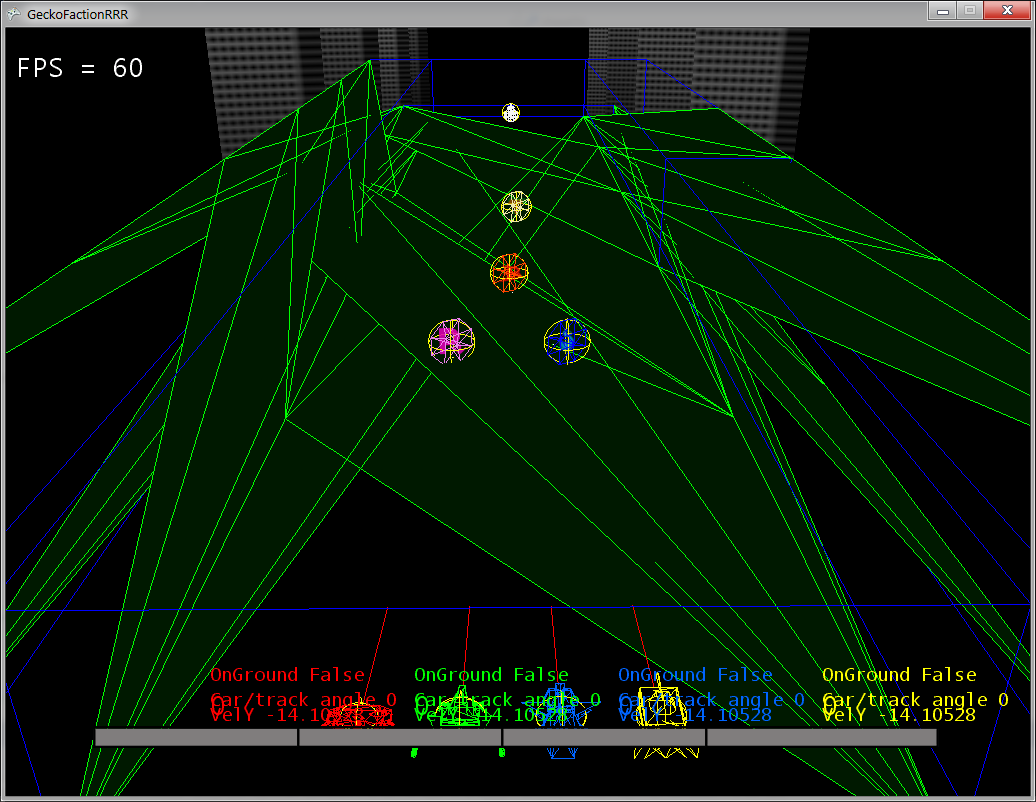
Doing some track collision optimization as the collision methods are using up a significant amount of time per frame for long tracks.

I went with a quad-tree like method, splitting the model mesh up into big bounding boxes, with a few triangles associated with each one. The results were unexpected, the track wasn’t in a neat grid, turns out I had forgotten the indices are ordered up the track and back down. There were also extra invisible triangles at the end of the track, where the indices turn around to move back down the next column.



The resultant big mess of bounding boxes; wasn’t a good idea to make them the same colour as the track’s wireframe. Works really well in reducing processing time though.

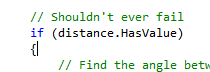
This screenshot also shows some formatting I did with the debug text as it was starting to get out of hand. I had forgotten you could concatenate strings and new lines.



I decided to re-structure the main centre points into their own class as they started to hold more and more variables and required quite a few helper functions to use effectively.

The above screenshot shows what happened when I forgot to convert a track generation method to the new track points.

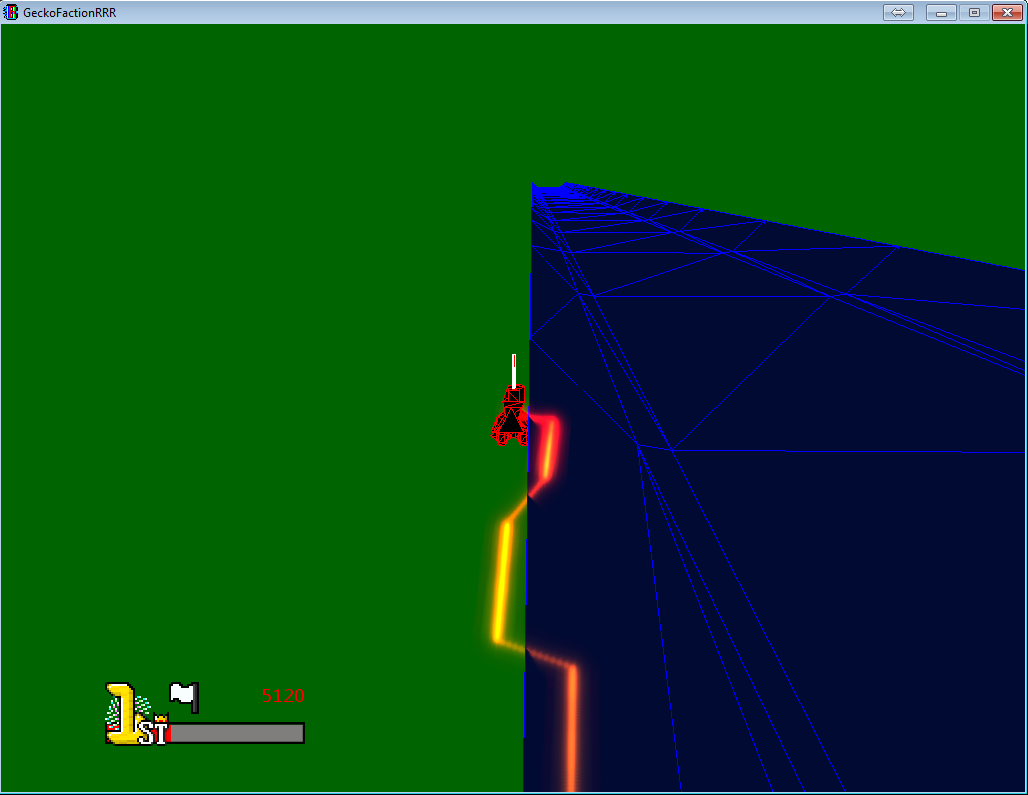
I had a look at using perlin or simplex noise to procedurally generate the track, rather than just using a random number generator but I never got round to it as there were other pressing things to do.



The source of a rare “fall through floor” bug. Turns out this does actually fail occasionally.

Not too sure why, the ray cast is sent from the car directly against the plane’s normal, so should always guarantee an intersection.

I attempted to fix this by sending a second ray to the triangle instead, should the above case happen. Weird fix as I use the plane for ray intersection as they are meant to be more reliable than a small face.



Although this “fix” has the side effect. The player can get stuck if they move up to the edge of the track, it looks like the player is grinding. If you add a railing model to the edge it might even look like this was an intended feature.

To try and make the car collisions look more natural, I added some fake physics to rotate them away from each other upon collision. This was done with a couple of dot products calculations to figure out which quadrant the other car is and from this, whether to turn the car clockwise or anti-clockwise. The cars initially turn a set angle every time, intending to eventually change the amount of rotation based on the strength of the collision but it looked convincing enough so I just left it like that.

This is the point where I notice that a lot of hacky/dirty methods were starting to get used more frequently to fix or implement features.

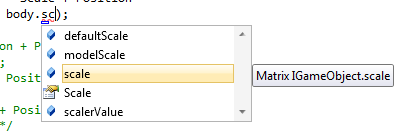
For example: the track collision only works one way so players could actually clip through the outer ramps from the outside, back into the track. Fix was to add a wall that extends below the track to stop any cars.

Next up was to finally fix the arcade build.

I had struggled with adding custom complier builds and all the results from google were about either using the already built in DEBUG option or creating your own from scratch, which wasn’t an option as the WINDOWS option needed to be present for XNA.

Simon hinted to building a branch of WINDOWS in his presentation so in the end, I just tried “WINDOWS, ARCADE”, complete guesswork. It just worked.

There were lots of instances of the Random class, littered all over the project, many of them being instantiated at effectively the same time. This caused lots of bugs where the same random number was used. I had read about the concept of a Singleton on StackOverflow at one point and thought this was a good time to try it out.



Working with teams.png

Simon had commented that the cars looked like they were floating, which was caused by using a very large bounding sphere to handle the track collision responses.

I added a second bounding sphere, for the sole purpose of moving the car on top of the track. I needed the height of the wheels to do this but as the above screenshot shows, I needed to consult my team mates on what their variables mean. They didn’t know either.

It didn’t look like the dimensions stored in the model files were used for wheels so I just edited them to suit my own purpose.

To implement sound I had based my sound manager on the sound class of EvilGod curvature, a game jam game I had stumbled upon while researching rotation matrixes. The source code was included with their release and their sound class looked really simple.

XNA has a bug with their sound framework where the game would occasionally crash on exit. Shouldn’t be a problem to the players but it is annoying while developing as debugging mode needed to be terminated manually in Visual Studio.

Some of the sound files Andy had provided weren’t in the right format as XNA does not support 24-bit audio. I used Audacity for converting and cutting out individual takes.

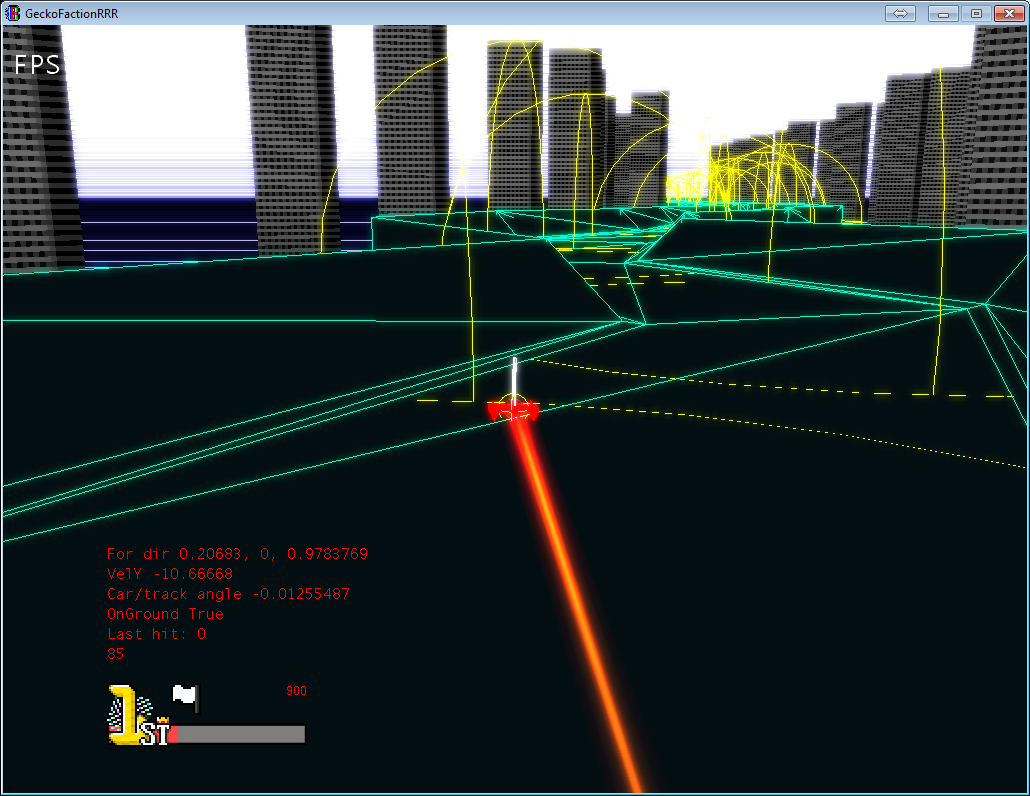
I looked around for information on how to rotate the car based on the angle of the floor underneath it. I was struggling with breaking everything down in Euler angles so I asked Simon for advice. He dug out a really simple solution which involved rotating the car around an arbitrary axis.

While it seemed to work as intended, it had a weird side effect of causing the car to shrink in size. I tried several combinations of matrix transforms but nothing really worked. I asked Simon for help again and he informed me that a cross product between two equal axis’s would result in (0, 0, 0), which is non- nonsensical for rotations.

Funny story, just before I left for dinner I noticed a controller-related bug. I patched this up in a few minutes and left. When I had come back, it turns out Mathu had noticed the same bug and committed a different fix while I was away, causing conflicts everywhere. This is probably why bug trackers exist.

Time was running out and no one wanted to implement the races so I picked up the task. This involved leaving my lovely little encapsulated classes and venturing into the spaghetti that is Game1.cs. Spent the next few hours just trying to work out what was going on in between all the test code that are still around near release and hacked together game states. This is not fun.

So many things needed to be implemented to get a race going that I didn’t really know where to begin at first.



Current look.

I tried fixing the camera chasing off players blown off the track by including the track’s checkpoints into the average position. This ended up in hilarious effects with the camera bouncing around all over the place. I gave up.

I used a second camera to render the cars during the countdown stage, so players can get a better look at their generated car while the first camera flies around the track. Although I never got round to implementing the track fly-by camera.  
I also envisioned a 2D mini-map during countdown to show an overview of the track but it was cut too due to time constraints.

The races took so many hours to implement due to all the bugs with restarting the game. That was not fun.

Pondered over how to handle the finish line. If the finish line was just along the track then players might miss it if they flew off into the side but maybe we want it to act like this.

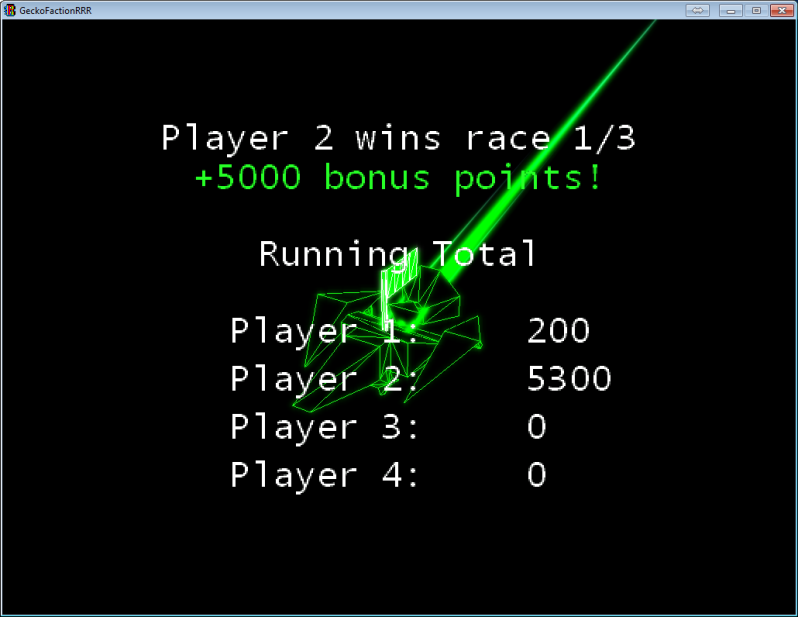
I discussed this issue with Alex and along the way we thought up the idea of a circular warp gate.



Source: <http://forums.penny-arcade.com/discussion/120894/screenshots-9-needs-more-spiky-bits/p78>

Similar idea to the above screenshot from Tera.

I used a bounding sphere to represent the finish line to match this circular shape and becaues of how much easier it would be to create one as opposed to calculating the minimum and maximum points of a box or the direction and normal of a plane.  
I run a second check in case two players hit the finish “line” in the same frame.  
Although one problem that I did not think of until after the final build is that with a sphere, the finish line would technically be closer in the middle of the track.



The way car falls into oblivion with the trail behind it looks great. This camera angle just ended up being almost perfect as you may see the loser cars trailing off in the background.  
I added an animation to the final scorecard that shows the overall winner of all three races. The players are ordered by ranking in a list, with the top players in a bigger font than the bottom players. The scorecard would then slowly scroll up from the bottom. Pretty arcade-y.  
One problem is that the white text is hard to read depending on which player colour wins the race.

We went from 250 commits to 290 over the last week.

Personal post mortem.

Working in teams is great as you can off-load the less interesting tasks to others however it can get very messy very quickly when all the team members are working in the same file, such as the Game1 and Car classes.

Creating the settings file proved to be such a time saver, allowing us to quickly adjust settings without requiring re-compiling or searching through the code to find that one variable. In the future I will aim to store more game data into scripting files.

I personally feel responsible for being stubborn and not using a pre-existing engine/framework to handle the physics and collisions considering how long I spent working on those and they‘re still not all that great and error-prone.

In future projects I will aim to not “re-invent the wheel” and look towards existing solutions