

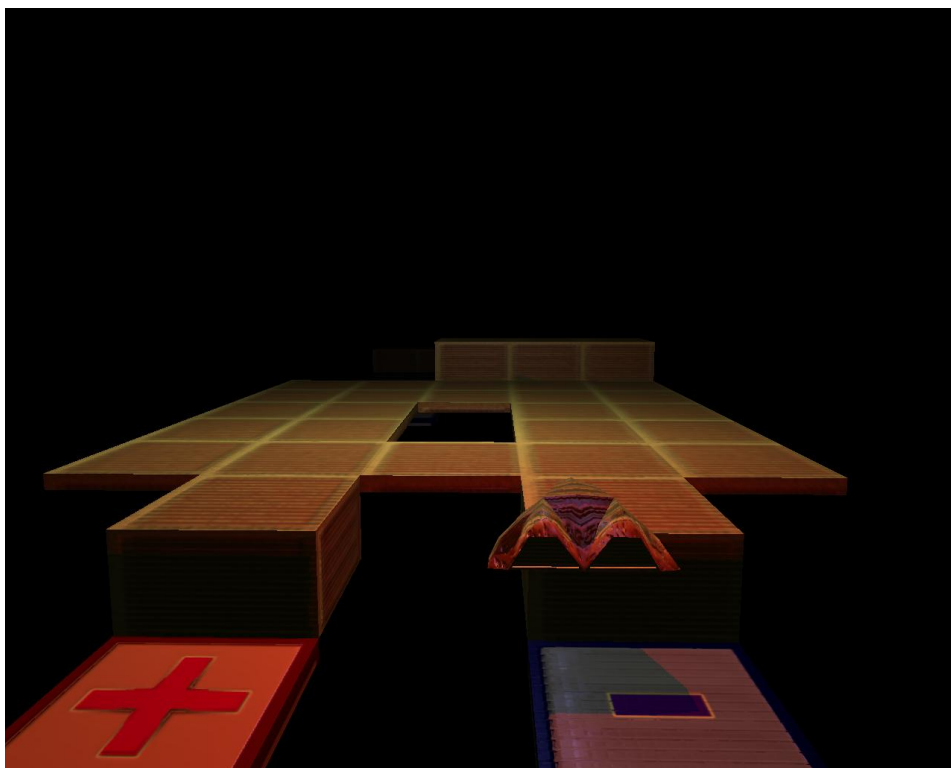
# Post-Mortem: Magna Karter

## Introduction

Magna Karter was produced by a group of three students from the University of Teesside for the 72 Hour Game Development Competition (<http://www.72hourgdc.com>). These three students were Kevin Williams, Luke Nockles, and Daniel Wright, working together under the name, "NFJ Software". The challenge set was to create an entire game from scratch within 72 hours, based on a theme which would only be released as the competition started. This included all code, art assets, music and sound effects, level design, as well as the game design itself.

The scoring was based on a peer-reviewed system, in which each competitor could play the other entries and score them in six categories before providing an overall score. The final product came seventh out of twenty-five entries. We are quite pleased with this result, and think that for the most part those entries which came higher than ours deserved to. Our scores broke down as follows:-

<b>Graphics</b>	<b>Sound</b>	<b>Gameplay</b>	<b>Innovation</b>	<b>Completeness</b>	<b>Theme</b>	<b>Overall</b>
7.14	5.93	5.79	5.36	5	6.64	6.14



*Magna Karter in its current form*

## What went right

### A solid game design early on

**Luke:** In the first hour we all got together and very quickly nailed down a solid and elegant game design. It was essential that we got off with a good sprinting start to set us up for the marathon of endurance that was to become the norm over the next 72 hours. Our design took a retro arcade design, remixed it with a modern technology marinade, and seasoned it with magnet focused gameplay mechanics.

**Dani:** Prior to release of the theme we would be working to, we had discussed the themes and decided that “magnets” was the one we would least like to work with, but that if it came up we would probably do some sort of contrived puzzle game. We were therefore all a little disappointed when the “magnets” theme was decided upon! Luke went out for a cigarette to console himself and while doing so had a flash of inspiration, which just goes to show that while smoking does lead slowly and inexorably toward your death, it might end up benefiting everyone else and is therefore a selfless habit. His design was influenced by a classic DOS game that I also have a lot of history with, so we were all moving in very much the same direction from the start. This meant we wasted very little of our 72 hours squabbling over petty design issues, or having to backtrack because of mistakes in the original game design.

### Software design provided ease-of-use and flexibility

**Kevin:** I spent some time developing a graphics engine for the game which was reasonably easy to use and provided decent flexibility to the rest of the team. We had set quite high standards for our graphics so I didn't want to settle with a graphics engine that had been knocked together quickly and then later find that it couldn't meet the demands we had set for it.

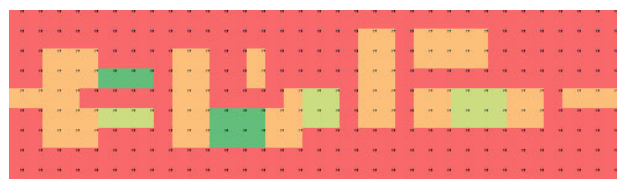
**Dani:** The first task I set for myself was implementation of the sound library, which I did in total isolation from the rest of the code. Luke was working on art assets at this point and Kevin's graphics library was in its very early stages, so there was nothing there to plug it into! This is quite unusual for an audio subsystem which is usually developed later into a project. As a result of this, I decided to wrap it up within one audio module, and to provide a relatively small and limited interface into this module, to avoid confusion when it came to integrating it into the game itself. This turned out to be the right decision, as we only actually ended up using the sound effects and applying them to Actors in literally the last hour of development and when we did so they slotted in with ease.

### “Chunk”-based space partitioning

**Luke:** Internally we broke down game worlds into smaller chunks that our rendering and update loop could far more easily digest than simply throwing the entire world at it and praying it didn't choke. It would have choked, and we saved ourselves a world of pain by only trying to ever render, or update sections of the world that actually benefited from it. We also noticed that our sorting for rendering meant that we rendered the world from a pure front to back ordering, minimizing overdraw as much as was ever going to be possible.

### Asset-creation software allowed for quick development of art and sound assets

**Dani:** I have some experience with a variety of synthesis and sequencing software, as well as some “home studio”-quality equipment lying around that I could have used. In the end, though, I settled with only two applications: FL Studio 6 for music and Audacity for sound effects. While tools like Reason and Cubase would have been much more powerful, FL Studio is perfect for quickly sketching down ideas and it's for this reason that I chose it. The soundtrack took about half an hour to compose, with the occasional five minutes later on to tweak it when I noticed a problem while playing the game. The sound effects (four in total) took around two hours – this is because I have no experience with creating sound effects. The Ogg Vorbis format was used throughout.



*Microsoft Excel made a great level editor*

**Luke:** Data was stored in suitable formats that were easy to work with, and performed optimally in our engine. We used dxt1 compression for all our textures to minimize bandwidth use, we used .x files which were easily loaded into the game, and then converted into our own better format, and we didn't for a moment bother with the nightmare of developing our own level editor in parallel. We instead used Microsoft Excel, which is quickly becoming my favourite editor for tiled based levels, and outputted to csv format.

## Interleaving our roles as programmers with our other roles within the project

**Luke:** The first part I played in this project was to create a first pass of all our art. In the first couple of hours, we had basic representations of all the visual assets our game would use. We knew what scope we had for implementation of our actors and our game code from an art point of view, because it was all there from the start. I had feared from experience doing both community work and professional development that art risked being a major bottleneck, especially considering the significant bar we had set for our rendering technology. Thankfully this was not an issue.

**Dani:** In some ways my mixed role as programmer and audio designer was a luxury, because I wrote the audio engine we'd be using and could design my effects knowing the strengths and weaknesses inherent in the library. It also meant that if I was having a moment of low productivity, I could move off code and write some music or sound, which really helped to keep me sane, and meant that little of my time was wasted. During such an intense development period it was nice to have another outlet to focus on.

## Use of the boost and FMOD libraries

**Dani:** I started writing the audio library using DirectMusic with DirectSound – both libraries I'd never used before – and getting my head around the intricacies of setting them up and controlling them was a real schlepp. Thankfully I only went down this road for an hour or so before deciding to look elsewhere, and FMOD was the first place I looked. FMOD is a wonderful audio library: It's free, it's portable, it's so easy to use, and it's quite powerful. It has since stood up to very different use in my dissertation.

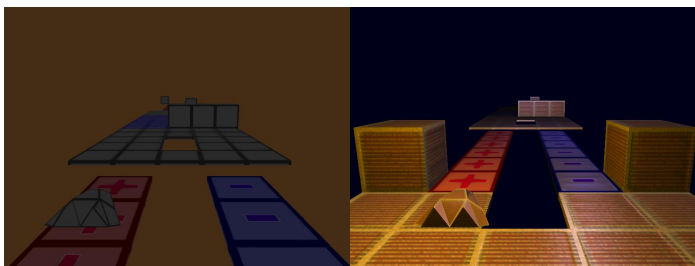
The other library we put to great use was the boost library. This is a collection of templates, largely developed with a view to being included in the next iteration of the STL. We only really used two boost concepts within our project: `shared_ptr<>`, and `bind<>`. However, `shared_ptr<>` alone is enough to have transformed me into a complete boost zealot, and Luke into a C++ programmer. It eased development a hundredfold. We have no memory leaks, and yet the keyword `delete` only appears once in our entire project. I would go as far as to say to any C++ programmer: use it. There is no excuse to use C-style pointers or `std::auto_ptr<>` unless you are interfacing with an external library which forces you to.

## Use of a simple messaging system to communicate between components

**Dani:** Though it had teething problems, I think the messaging system we incorporated was a wise choice. It didn't take that long to code, and it meant that as we added input, physics, and gameplay elements we didn't have to worry about how the components interfaced with each other; we just sent out a message and let their own message handlers deal with it however they liked. It wasn't perfect, mind you, and I'll talk about some of its flaws later.

## Powerful, feature-full graphics engine

**Kevin:** The graphics engine was an achievement, given the time constraints. The interface to the shaders allowed Luke to develop shaders which employed per-pixel blinn lighting, normal mapping, and parallax mapping, and later allowed me to add shadow mapping into the mix. The engine did actually contain support for cubemaps so that an environment-mapped reflective shader could have been developed, but we didn't get time to write the shader itself for that. We also decided to support Shader Model 1.1 in addition to 2.0, which was lucky because as it turned out many of the people judging had quite low-end cards which wouldn't have supported the SM2.0 shaders we'd written.



*It took only three hours to move from our first geometry on the screen to a nicely lit parallax-mapped scene*

## Mature approach to sleep and the occasional break

**Dani:** It was extremely tempting to start off the weekend with an all-nighter, and continue in the same fashion. With only 72 hours of development we didn't really want to waste any of it on sleep! However, I decided to give myself a minimum of four hours of sleep every night, and I think this led to my writing much higher-quality code during my waking hours. In addition to this, we stopped to cook and eat supper every night, and on the last night actually got out of the house and went to etsuko for a "final push" meal. I really think this was valuable time well spent. After so many hours coding solidly, quality and motivation would soon have fallen if we didn't get away from it for a short period of time.

## What went wrong

### “We’re making a game – not an engine”

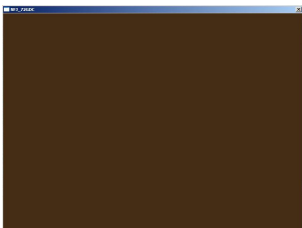
**Luke:** For the most part, we made an engine. This has its advantages, especially when you’re working in a team, and need to make code that’s going to work with everyone else’s, but it did mean that it was far too late into the project that we actually got gameplay happening.

**Kevin:** I spent too long working on the graphics engine, which didn’t leave time for a HUD or menu system. I maintain that it was worthwhile having a decent engine with a solid interface, however there were some things I spent too long trying to get “just right”.

### Too much time spent learning new things

**Luke:** A lot of the aspects we coded, we were doing for the first time, and I personally had never experienced coding as part of a team in real time. That aspect was just new to me, so time was spent learning the coding style of the team, and trying to match it, and write code that would be easily understood by anyone who would later work on it.

**Dani:** We definitely spent a lot of our time coding things we’d never done before, which was perhaps not wise for a competition in which time is the chief constraint. For my part, though I’d never coded such a fully-featured audio interface before, that wasn’t the hardest part. It was the creation of sound effects, in which I’ve had no training, that took far longer than expected and in the end didn’t actually sound that good. I started this three hours before the deadline, which meant that I didn’t have time to write the number of sound effects I’d originally drafted (I completed four out of eight), or to complete the soundtrack(I wrote one out of three planned pieces).



*For a long time, this is all that showed on the screen*

### Nothing on screen for the first 24 hours

**Dani:** This is the core problem, I think, which led to some of the faults with gameplay and physics in our game. The best gameplay comes out of a good starting point for design (which we had), and a lot of experimentation and tweaking. If we could have had something basic up on the screen early on we could have started experimenting much earlier, and would have realised earlier on – for example – that a physics model simply wasn’t appropriate for this style of game.

### Focus on pretty effects didn’t leave time for other more essential features

**Kevin:** We should have left some of the pretty graphical effects until later in the project and got the core system working first. I spent somewhere between four and six hours implementing shadow mapping which meant that I didn’t have time to complete the HUD before the final deadline. Some sort of shadowing technique was essential to give the player an accurate feeling for whether the Magnetorunner was hovering or resting on the ground – but a simpler solution could have been used until we decided we had time to implement full shadow mapping. In retrospect I think it’s sometimes better to take a break when faced with a bug and tired – especially in this case where realtime shadow mapping was something I’d not coded before.

### Attempt at an inappropriately realistic physics engine

**Dani:** This is by far my biggest personal regret with this project. I wasted about five hours of the final twenty-four implementing a Newtonian physics model which eventually had to be completely discarded. A realistic physics model really wasn’t appropriate for this style of game, and when I spent half an hour after throwing away all that work hacking together a quick simulation which simply adjusted speed and used a sine curve to simulate jumping (all good hacks use sine curves), the game played ten billion times better. The final implementation has a bug where the Magnetorunner floats up and down as it flies along; this is due to the use of a sine curve and would probably have been fixed had I got that five hours back!

### Difficult first level made the game slightly inaccessible

**Luke:** The first level of the game we shipped was far too hard. We should have had two that came before it, one that was easy as hell allowing the player to get used to controlling the Magnetorunners, and another introducing the magnet gameplay mechanics without too many hazard elements in the level too.

## General Conclusions

**Kevin:** Time constraints and the fact that this was our first time working together taken into account, the competition went reasonably well. We probably should have focused on the more important features earlier. For my own part, shadows could definitely have waited until a working HUD was in place.

**Luke:** We created a brilliant game demo, with solid gameplay mechanics presented in a unique way, and superlative technology. We kept up the momentum until the end of the project, and although we were all tired by the end, had enough energy left after finalizing our work to go to the pub and celebrate our success. I also think that our experience on industrial placement really helped us. All three of us were on placement last year – each at different companies – and I think this gave us a better idea of working to targets within deadlines, as well as working as part of a team. As a team we worked really well together.

**Dani:** I'm very pleased with the way the game has turned out. What surprises me most is how clean the code is. That's not to say that it's beautiful code – there are some parts of it that are absolutely disgusting, believe me – but the interfaces exposed by each system are relatively well-defined and provide a good deal of flexibility while remaining easy-to-use and not confusing to the client. I think this really helped us work together, and create a final product which is not totally ridden with bugs and design flaws.



*With us all being in the same room, communication was easy. We also used a wall-poster to keep track of tasks we had yet to complete*

## What we've learned

This was our first time entering a competition which imposed such restraints upon the development time we could spend. We've learned a lot from the experience and were we to enter a similar competition again there are some things we would do differently. It should be noted that these are points we would follow if entering another short competition – many of them would not be applicable if we had, say, a month of development.

**Luke:** I think we should have made a game and not an engine. Of course I'm going to contradict myself in the next section.

**Kevin:** We should definitely have got a basic renderer in place earlier on. This would allow for more experimentation with the actual gameplay mechanics, hopefully resulting in a more balanced game. In the same vein, we should have left a lot of the pretty effects until after the basic gameplay was in place and we had the luxury of creating something beautiful.

**Dani:** I agree with the points above, and would add that we should have paid more attention to the original game design spec. This wasn't too much of a problem in our case – for the most part we were all working in the same direction – but the disastrous physics engine is a good example of getting sidetracked by a feature without bearing in mind whether it has any relevance to the game design. Every time we start a new component within the game, we should consider how it relates to the game design overall, and base its functionality upon that.

## The future

One of the advantages of having spent too much time writing an engine rather than a game is that it is much more pleasurable to work on after the competition has finished than it would have been if we'd simply hacked it together. We have already begun, and since the game was completed just under a fortnight ago we have removed some of the bugs, added fullscreen support, fog, and support for XInput-compliant controllers with vibrating "rumble" feedback. We think the game has a lot of potential and would definitely like to develop it further. Here are some of our plans for it.

**Dani:** I would like to spend quite a lot of time honing my skills with sound effect creation to create some decent-quality sound effects for the game. In addition to this I'd like to compose at least nine new tracks for the game's soundtrack – to fit with the differently themed worlds Luke mentions, and provide some more variety. In terms of code, I'd really like to get a decent GUI and menu system put in place, and possibly implement some proper collision detection so we could have frantic multiplayer action!

I'd also like to develop the messaging system we're using. Currently, while very flexible, it has some serious flaws that need to be addressed. Primarily it is the way user data is sent along with the messages. Using the current system this requires dynamic memory allocation and two casts for every single message which has associated data, which is ugly and unacceptable. An alternative must be researched. Also, messages are passed on and interpreted in immediate mode at the moment, which is not the best way of doing things. A more formalised message queuing system would be much better and lead to more solid and structured code. It would also allow the sender of the message to give some idea as to which component should interpret it, as opposed to the current system where every message is sent to every message handler that has been registered. Finally, attaching some more information such as the source of the message and the time at which it was sent would really aid in debugging.

Escaping into dreamland for a moment, what I'd *really* like to do with the game is take some influence from F-Zero X / Extreme G. Currently our primary influence is Skyroads, which is a great game but is starting to feel a little dated. If we could take that Skyroads gameplay but apply it to a modern-feeling world and level design, I actually think we would have a marketable product.

**Luke:** We were able to make a solid and indeed extensible engine from which to build upon and develop our game further. This has opened up the gate for further development.

I am going to continue working on level design, and am going to design more levels and liaise with a talented artist to develop multiple visual themes to introduce the idea of 'worlds', that are unlocked as the player beats levels. I will work on the flow of the progression within levels, and develop a fun learning curve.

I'm also going to optimize the lighting system in the game, so we can maximize the number of lights we have visible on the world at any point whilst maintaining a solid framerate.

Finally, I'm going to code a solid particle system, so we can really dramatically ramp up the level of visual fidelity whilst staying cheap on the GPU.

**Kevin:** I'd like to get the HUD in place – a lot of the code is already there, we just didn't quite have time to finish it off before the deadline. After that my main target will be implementing drawlists, which will be necessary for Luke's particle system. When I'm finished with that I'll be free to implement more and more pretty effects!