

Postmortem: *Startopia*

By Wayne Imlach

[Muckyfoot Productions](#) was formed back in '97 by four former employees of Bullfrog — Mike Diskett, Fin McGeachie and Guy Simmons, with the later addition of Gary Carr. It was during these early days that the concepts for their first two projects came together. Mike and Fin took the lead with *Urban Chaos*, a departure from the style of game they had worked on at Bullfrog. This would be the primary project for Muckyfoot, and it's debut game. Meanwhile, as a secondary project, Gary and Guy decided to follow their roots and develop a sim/God style game that would compete with the "theme series" they had helped create for Bullfrog. One of the ideas they had originally proposed to EA as a follow up to the successful "theme series" of games, revolved around running a space station, and while Bullfrog and EA showed little interest in the idea, their new independence allowed them to make the concept a reality. With publisher support for Muckyfoot acquired in the form of Eidos, *Startopia* (or *Space Station* as it was called then) was born.



The style for *Startopia* was quickly laid down in a broad concept document developed by Gary and Guy — the game would take place within a torus with multiple decks, and would revolve around the interactions of a number of distinct alien races. The player would have some kind of indirect control of these beings, and would have to provide facilities and entertainments to keep them happy. It would draw inspiration from classic Bullfrog games as the "theme series", *Populous* and *Dungeon Keeper* to provide gamers with a familiar hook, as well as introducing some innovative elements of it's own. Above all the game would be humorous and comic, borrowing heavily from popular science fiction, and satirising as many science fiction shows and books as possible.

The original concept-document conveyed a more mature aspect to the graphic style — perhaps with little thought to matching the game style to the target market, though it was at the time nothing more than a rough draft of how the game might eventually develop. With this very basic brief in mind, the basic technology framework was laid down, and creation of game models and objects were started.

For the next year, a small team of programmers and artists generated what was essentially a graphical demo of a space station game, with little regard to actual gameplay — an unfortunate necessity when trying to acquire continued funding from a publisher who will judge the progress and suitability of a project on look rather than feel. However, the demo successfully won the support of Eidos, and *Startopia* was officially signed as Muckyfoot's second project.

At this point more serious thought was given to how the game would actually play, and a game designer with experience in this type of project was hired to develop the gameplay systems, levels, and interface and produce a more technical design document that the team could refer to. Taking into account the demo work already done, *Startopia* the game began to take shape, detailing the interactions between characters and objects, the goals of the player, scripting language requirements, additional features required, and various other gameplay details. As the game progressed more serious thought was given to target audience and target platform, resulting in some modification of earlier artwork and concepts. We chose to go with as broad an audience as possible, toning down much of the overt sexuality in the game to the level of innuendo, so the humour would still appeal to the adult gamer while being sanitary enough for a younger audience. We also decided to aim for a lower base specification machine — fortunately a reasonable under-estimate of projected target hardware in the original concept document meant little needed to be removed or changed.

While work continued steadily on the game, it remained unplayable for quite a large proportion of the development cycle. This had to do with the complex nature of the interaction between all the final game elements — everything relying on something else to work in a balanced fashion. Until almost every game element was in a near complete state, it was impossible to construct or accurately balance playable levels. Everyone had to assume the disparate pieces of code and art they where working on would mesh together as planned, relying on faith and trust that the design would not only work in theory but in practice too. There would be no time to re-write or re-design the game if it didn't work as imagined.





Fortunately (particularly for me!) almost every gameplay concept and system, theorized and proposed in the game design document proved accurate, and the final product differed little from the game originally imagined.

Much of the project's success stems from the experience and professionalism of the *Startopia* team. Art, design, programming, and sound were each led by an industry veteran with a strong knowledge and realistic outlook of development. Supported by experienced staff, the team required little in the way of management — probably much to the delight of our Eidos based producer. Tasks could be handed over to individual members of staff with only the lightest of briefings, secure in the knowledge that it would be carried out in an efficient and professional manner. The dedication shown by the team was also impressive, and certainly helped ship the product in a reasonable timeframe, especially in the last few months of development.

Despite the team's practical approach to development, the atmosphere in the office during development always remained informal and light, which helped minimise tension and conflict among staff that usually arise during the more stressful crunch periods and deadlines. This was helped by the hands-on nature of the company directors — who not only run the studio, but also work full time as project leaders alongside the staff.

Startopia also benefited from experience gained while working with Eidos' external QA and localization during the development of *Urban Chaos*. Learning from the mistakes and problems encountered during that project, both Eidos and Muckyfoot revised their communication channels and protocols resulting in a much smoother flow of information between the two companies, an aspect of development most appreciated by our own small internal QA department.

Though by no means a perfect project, it's generally agreed that *Startopia* had a relatively smooth development cycle when compared to past games that the staff had worked on.

What Went Wrong

1. Early Emphasis on Look Rather than Feel. The early development stage of the game focused far too heavily on the graphical look of the game, and almost entirely avoided any thought as to how the technologies used would impact gameplay. While this proved influential in securing a strong publishing deal with Eidos, it had repercussions in the later stages of the project, resulting in unwanted game design restrictions to avoid re-writing code. And in other areas, a forced re-write of code that just didn't lend itself well to the gameplay systems we needed to make the game playable and enjoyable.

One particular area that would have benefited from some greater forethought, would have been user-expandability. As we later found out, the system we originally coded was quite narrow in scope, and required extensive code updates whenever a new object needed to be added to the game. It made third party expansion of the game practically impossible, as well as proving a pain for us to use.

Ideally, the technical aspects of the game design should have been started right at the beginning of the project, well before any models were built or code was written, and would have taken into account expandability as a primary factor.



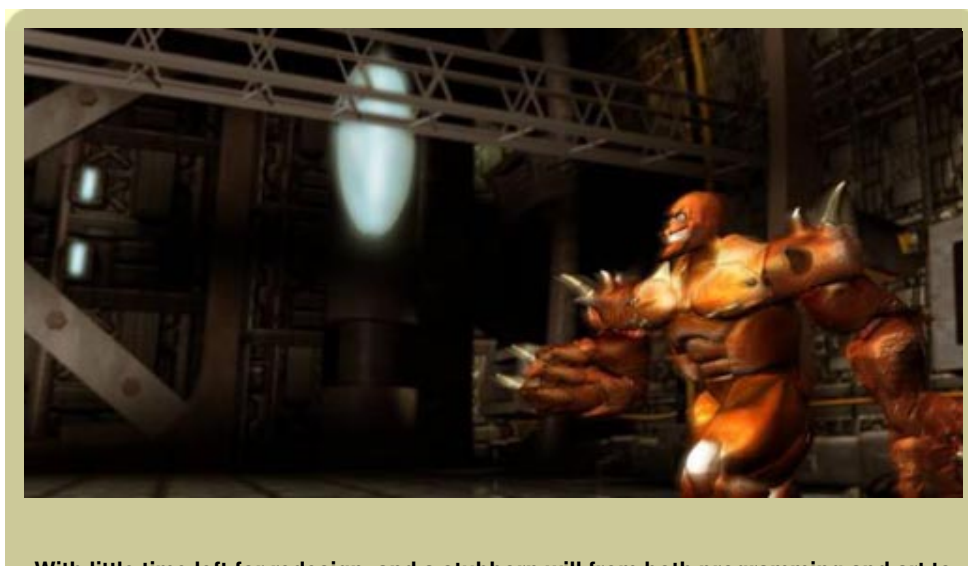
2. Lack of Custom Editing Tools. The amount of work required getting animations and special effects working in the game was severely underestimated. As such, no tools were developed for the artists or designer to easily insert and tweak models, animations, or objects. Almost everything had to be coded manually for it to work as we intended it to in the game. This led to an excessive amount of programming resources being spent tweaking animations and fixing models (often with code boggles), rather than leaving it to the art team, who found themselves under-tasked during the latter half of the project.

Again, this probably stems from a lack of design forethought on the part of the original team, and an underestimation of the sheer number of models and animations that would eventually be required to realize the game.

3. Code Ownership. One of the obstacles faced when developing this type of game was the way every system and element of the game interacted together in a vast and complex manner. Very few elements could be considered stand-alone, and as such the programmers began to find that they often needed to jump into someone else's code in order to get their own systems working correctly within the game. This originally began as an informal activity, as the code wasn't yet particularly complex, but as time moved on and each programmer found they had more and more individual code elements to maintain, the changes made began to creep in without everyone's knowledge.

This lack of communication and tracking led to several areas of code being rewritten or duplicated by different programmers, each unaware of the impact it would have on the others code. While the editing of code by someone other than the original author meant less time wasted on communicating the required changes to the original programmer and waiting for an update, in the long term it resulted in less streamlined systems, and unnecessary rewriting of some areas.

If there is a bug in person A's code that person B spots, A must fix it. Otherwise, if B fixes it, (a) a higher chance of bugs and (b) A now doesn't know what the code does, and will refuse to touch it again. But B doesn't completely understand the code; they just know the tiny bit they fixed. So now, no one knows what that module does. The result is chaos. Ownership of code can change, but it must be a well-defined handover, and B must be happy that they understand what the code does, and are happy to own it.



With little time left for redesign, and a stubborn will from both programming and art to finish the damn thing, it finally was completed — but at an excessive cost in resources, and without the polish or functionality we would have liked.

In the future, we will be looking at making the code as modular as possible (to minimize overlap of programming tasks), and having a stricter view on ownership and editing rights.

4. Front End. The design of the game front end was left till late in the development process, and on reflection was far too complex (read: arty and fancy) for it's own good. Where a simple set of menu items appearing on a static backdrop would have sufficed, it was decided to wrap the front-end system up in a clever 3D model, with a hinged body and extending screens. We seemed to have been suffering from some memory loss at this point in development, because the difficulties in trying to get other complex models working correctly in the game seemed to have been forgotten.

With little time left for redesign, and a stubborn will from both programming and art to finish the damn thing, it finally was completed — but at an excessive cost in resources, and without the polish or functionality we would have liked. We have since learned the advantage of economy of design, and will be implementing it in future projects.

5. Lack of Exposure. There seemed to be little exposure of *Startopia* among the gaming press, and no hype surrounding its release, even though it was developing some unique and cutting edge technology, particularly in the game engine and it's AI players. While in part it was the responsibility of our publisher to market and publicize the product to the consumer, the blame also rests upon us in naively believing that just developing a good, solid game would be enough to ensure sales. To our cost, we've found that the market just doesn't work that way any more. We should have been far more active in proclaiming that *Startopia* was the next great event in gaming, releasing snippets of information about the fantastic gaming opportunities it provided, and how it would redefine the God-sim genre.

With so many games being released across so many formats, and a much broader audience to sell to, it's no longer the case where the majority of gamers can seek out and decide for themselves, which titles are the best — you need to shout loudly to be noticed by the consumer. We were far too quiet, and as such were drowned out by other less accomplished games who where far more vocal about their release.


What Went Right

1. Design. The design of the game systems (object interactions, peep attributes, interface, level structure etc.) was done in a few months, and hardly changed at all during the development cycle. The only changes were the removal of a few game elements near the end of the project due to scheduling restrictions, which had no impact on the fundamental way the game played. This required a lot of faith from the team in the ability of the game designer to get it right the first time, as the game could not be realistically prototyped or tested for playability or balance until a vast majority of the elements had been coded and were in place. This trust in the designer's ideas and systems working as written wasn't blind. A past history of successful games did much to strengthen the belief that *Startopia*'s design was an accurate blueprint for a fun and enjoyable game. This highlights the importance of a strong credible design blueprint at the early stage of game development.

The benefit of this was one of economy — with a stable design that changed little during the development period, very little work needed to be scrapped or rewritten due to unexpected design changes. This was one of the main reasons why such a small team could put together a large game like *Startopia* within a reasonable development period (the other being sheer hard work from all involved!).

2. Scripting and Tweaks. A flexible yet simple scripting language was developed to build the missions that appear in *Startopia*. Stored in a simple text format, it allowed quick and easy editing of each mission without the need for any special tools. Missions could be updated on the fly, and required no extra compiling or processing before they could be tested. If a specific action was required in a mission, and it couldn't be adequately described with the current script commands, it was a straightforward task for a programmer to update the scripting language with a new command and provide support for the action in the game with no disruption to existing scripts.





A flexible yet simple scripting language was developed to build the missions that appear in *Star Wars*.

In addition, this simplicity made the game ideal for the modding community to build their own custom scenarios that could be easily slotted into the existing game, and distributed as just a set of small text files.

The same system was applied to the values that determine the balance, and in some cases, look and sound of the game. Referred to as "tweaks", as well as being adjusted from within the game, they were also stored in a text file that was loaded at the beginning of a session. Their simple text-based format meant that adjustments to game balance could be made, then saved out and distributed to game testers with a minimum of fuss. Then adjustments could be made on individual machines to see the overall effect on the game. The tweaks were also used to enable or disable a number of audio and visual settings. We proved invaluable in adjusting various parameters of 3D models, animation settings, and object behaviour without having to delve into code. Wherever possible, any values within the code that might need some adjustment or "tweaking" to get right were included in this text file.

3. Compatibility. Compatibility is incredibly important to reach the widest possible audience. It's hard enough making people want to buy your game without sneering at them because they have an obsolete computer. Our compatibility system was composed of three parts:

- Scalability: Make sure all game effects can be scaled down to work on even the simplest graphics card. The effect need not be quite as pretty on the lower-end cards, but it must actually exist and not look completely terrible. Alternatively, some special effects that are not vital to the game play can simply be turned off.

Each single game effect (e.g. "draw with a disease texture added on top") has multiple versions of the effect done with different methods. At the start of day, the game starts from the more fancy version, and tries to get that working on the current graphics card. If it can't, it tries the next one, which is a slightly simpler version, and will be either slower, more ugly, or both. And so on down the list. The last version in the list is the one that is guaranteed to work on every card. In this way the game automatically uses the prettiest and fastest effect that works on this card.

- Diagnostic dumps: Whenever the game is started, it dumps a whole load of diagnostics out to a text file. This includes any hardware information we could even dream was relevant — graphics card ID, capabilities, memory, driver versions, OS versions, which blends were used, which didn't work, etc. The idea was that if a user had a problem, instead of asking them questions about their system (most users do not know what is inside the box, or have faulty and incomplete knowledge), we just had to ask them to send us the file — we called it something friendly such as "your gfxcard.txt".

It could also present some of the more obvious problems. For example, all the user knows is that the game is slow and crashes a lot — an incredibly vague problem to track down. Looking at the diagnostics, if the card reports having 0MB of AGP memory, we can fairly reliably diagnose old motherboard drivers.

- Detecting cards: In theory, DirectX has a very powerful and elegant mechanism, called `ValidateDevice()`, to detect which of the blends and effects will work on a particular graphics card. However, this relies on the video card's driver being bug-free and well written. In practice, this is rare. So you get video cards claiming to do blend modes that either (a) the driver doesn't understand, and gives a random answer, (b) won't work on that hardware, though the driver claims it will or (c) would work on the hardware, except the driver is buggy. The only way to detect these cases is to sit a tester down and try playing the game with every video card in the building.

This is fairly easy to do, though it does take the testers time to go through all the different cards. The tricky bit is to detect what card is in the machine. You can't just ask the card for its name and look for the word "Voodoo" in it — that never works reliably. The only reliable way, is to retrieve a number identifying the card and the maker, called the `DeviceID`. Then we have a huge text file listing each `DeviceID` we know, followed by the list of fix-ups that need to be applied to get that card running correctly. The file was named `CardID.tom` — it is just a text file, but the bogus `.tom` extension stops inquisitive people fiddling with it and breaking stuff.

In some cases, we also found that although newer drivers did some effects correctly, older drivers did not (due to bugs or whatever). So as well as `DeviceID`, we also had subsections for a range of driver numbers. So if we had a problem with an earlier driver, but we had a later driver (e.g. version 1234) that did the effect correctly, we could say that driver versions 0 to 1233 disabled the effect, whereas drivers 1234 to infinity enabled the effect. This meant that we could do the fanciest blends on cards that we could without getting nasty bugs on systems where the drivers had not been updated for ages (in practice this is most systems!).

The system in action: OK, so we've shipped the game, and someone emails us with a problem. One example was "some of the game objects

keep flashing green and black blobs". This we diagnose as an effect known as "impostors" failing — a known problem with a wide variety of cards. Because the system is designed with scalability, we can disable them if they don't work, and use an alternative method. This alternative may not be quite as pretty, and may not be quite as fast, but it is guaranteed to work on all cards. So we ask the user for the "your gfxcard.txt" file, and from that find out what card and driver version they are using. We update the CardID.tom file with the data that "impostoring" should be disabled on this card with these drivers (and usually all previous drivers as well), post the new CardID.tom file to the user, and check that it works for them. Then we post the updated CardID.tom file on our website. In this way, the very first thing we recommend to people is to download this new text file — it's like a patch, but it's only about 50k in size! And it's easy to keep it incredibly up to date, since changes to the file are only enabling/disabling stuff that testers have already checked, so it doesn't need much re-testing.

4. VAL/ARONA Characterization. When developing the advisor character (VAL) for the game, we took the unusual step of basing the character around our ideal voice actor, rather than deciding on a character type and searching for a voice to match. We listened to a series of example voice actors using a variety of accents, listening for one that we thought would match our perceived image of the game.

The casting of William Franklin as both the voice of VAL the computer advisor and Arona the trader was a great step in enhancing the humour and tone of the game. Once we had chosen him and confirmed his participation, we began writing our scripts specifically with his voice in mind (which reminded us of the archetypal condescending British butler). In some respects VAL isn't played by William Franklin — VAL is in fact based on William himself!

A distinguished and very professional actor, Will took our scripts and brought them to life in a way even we didn't expect, making VAL, and eventually Arona (with the help of some digital manipulation) seem like flesh and blood beings rather than software-run scripts.



The casting of William Franklin as both the voice of VAL the computer advisor and Arona the trader was a great step in enhancing the humour and tone of the game.

The professionalism of Will allowed us to record all the material used in the game over two day-long sessions, which were set a few weeks apart to allow us to review the scripts and add or update them as required. Where the script seemed a bit vague or perhaps contained grammatical errors, Will would record alternatives for us to use, often producing a better flowing sentence or paragraph structure than the original material. He could also read the scripts cold yet produce a perfect recording on the first take, which is one of the main reasons we managed to record so much material in a relatively short period of time. This kept costs down, which was a bonus considering the premium you pay for quality voice actors (a premium definitely worth paying I may add!).

5. The Team. Probably one of the most important aspects of the game development, and the one most often overlooked by publishers and public, was the commitment and stability of the *Startopia* development team. While the team began with only four members, and finally grew to 15 full time staff, it never lost any of it's number due to the usual industry pressures. This resulted in a strong bonding and level of understanding among the team of how each member worked and thought, allowing a high degree of informal decision making and sharing of tasks with little or no reliance on management to approve actions. Indeed, there was none of the usual slew of managers when it came to *Startopia* — if there was a particularly thorny executive decision to be made, then the team Lead programmer or Artist could make the call, handily also being two of the four company directors. Even the concept of a 'producer' was somewhat alien to us, the position being provided

on a part-time basis by our publisher.

This was further strengthened by our relative small size and overall experience in the industry; with so few programmers and artists, communication among team members was maintained with little disruption to the schedule. People could literally turn from their desks and discuss a proposal with all the necessary people without having to leave their seats. At the worst it might require taking a few steps through a door if the art team needed to confer with programming over a subject. The experience and professionalism of the team also kept conflict to a minimum — trusting the judgement of the team member responsible for any particular section of the game, even if it might be contrary to personal feeling.

Conclusions

Overall, despite its flaws, *Startopia* is probably the game that members of the team are most proud of working on. It looks great, sounds fantastic, and is eminently playable, even among the developers who worked on the game for so long. It has garnered many great reviews in the press and on the 'net, and has been nominated for a BAFTA award in the UK.

Despite this, the game has unfortunately failed to be as financial a success as we had hoped. While we understood that the muted marketing that the game received would mean a slow uptake, we had hoped that its qualities, as a good game would help it achieve a respectable number of sales after its release. We won't be making the same mistake next time. Our next project will not only aim to be better than *Startopia*, we'll also ensure that every man and his dog knows it!

Game Data



Startopia

Game Data Publisher: Eidos Interactive

Number of Full-Time Developers: Core Team — 6 programmers, 4 artists, 3 testers, 1 designer, 1 musician (Support Resources — 2 programmers, 2 artists)

Number of Contractors: 1 scriptwriter, 1 voice actor

Estimated Budget: £2 million (\$3million)

Length of Development: 2 years

Release Date: June 2001

Platforms: Windows 98

Development Hardware (Average): PIII 550, 128mb, 20GB HD, Win 98/Win 2000/Win NT, Various 3D accelerator cards, Various sound cards

Development Software: WordPad, MS VisualC++, SourceSafe, Photoshop, Paintshop Pro, 3DS Max, paper, pencils and pens

Notable Technologies: Direct X, Bink, Peer Engine

Project Size: 335000 Lines of Code, 33000 Words of In-game Text, 125 MB of Graphics and Models, 151 MB, of Compressed Sound Effects, 48 Minutes of Game Music

[Return to the full version of this article](#)

Copyright © 2016 UBM Tech, All rights reserved