GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Stardock's *Galactic Civilizations 2: Dread Lords*

By Brad Wardell

In March 2003, Stardock released its space strategy game *Galactic Civilizations*. *Galactic Civilizations* is a turn-based strategy game set in the 23rd century in which the player takes on the role of leader of their civilization. Players colonize planets, research technologies, negotiate with alien civilizations, fight wars, and spread their culture across the galaxy.

The game was developed on a shoestring budget. The development budget was around $300,000 and the after-release budget reserve of another $300,000 to support updates, help with marketing, and provide product support depending on sales. While a total of 5 developers were involved on the project at different times, at the end of the day, only 3 developers, at most, were working on the game at any given time. Similarly, no more than 2 artists were working full time on the game at any given time. In total, 10 people were involved.

At the time of release, the expectation was that it would sell approximately 30,000 units in North America and around 10,000 units internationally. Our publisher was Strategy First and they told us we could expect about $10 per unit in royalties from North America and about half that internationally. We also had confidence that we could get around $350,000 in retail revenue. We also anticipated around 5,000 units in direct sales which would be worth $40 apiece for a total of $550,000 in revenue. So if we could do the game for $300,000 then we could make our investment back. The second $300,000 could be used in the event that the game did very well and needed more support.

Release Aftermath and Sequel Talk

Galactic Civilizations was released at roughly the same time as*Master of Orion III*. For a variety of reasons, *Master of Orion III* ended up helping the sales of *Galactic Civilizations* tremendously. Between that March and the end of that year, the game sold approximately 50,000 retail units in North America and about 15,000 electronically. It would eventually go on to sell approximately 150,000 copies worldwide.

The unexpected success did not translate to a windfall or income, however. Publisher Strategy First filed for bankruptcy without paying a significant portion of the royalties we were owed. So for a good chunk of retail sales, we never saw a penny. In addition, because our publisher's financial issues became critical in the middle of release, Stardock ended up taking on an increasing amount of the marketing and support burden. On units we were paid for, our royalty did not end up $10 per unit but closer to $7 per unit. Aspiring developers make note -- on a $39.95 game, if you're the developer, you can probably expect to make around $7 per unit at the high end if you're not the publisher.

The game's success guaranteed a sequel would be made. The question was purely how we would go about it. Stardock's business software division was doing incredibly well financially and put us in the position where we could afford to both develop and publish the sequel.

For the sequel, we put together a $900,000 development budget and a $400,000 marketing budget. Like the first time around, we also reserved $300,000 for after-release updates, support, etc. to cover the entire sequel's post-release life. Compared to other major retail games, our budget is tiny. But it was still over 3 times the original's budget.

It was decided that we would publish it ourselves. So instead of only making $7 per unit, we could expect to make between $15 to $21 per unit depending on various factors. With full control of marketing and with a bigger budget, we projected that we could expect sales of approximately 75,000 full-priced units in North America plus 25,000 discounted units along with a similar number internationally for a total of 200,000 units.

The much bigger budget and taking on the role of publisher meant an increased risk for us. But because Stardock's utility software such as Object Desktop was selling so strongly, it allowed us to develop the game while remaining in a strong financial position throughout.

What should the sequel do?

We had a laundry list of things from the first game that we wanted to address:

- Ship Design
- Resolution Independence
- More sophisticated combat
- Better UI
- Unique worlds

At first, our design was pretty modest:

Early concepts of *Galactic Civilizations II* assumed we would build onto the *GalCiv I* code-base with prettier sprites and new features.

Ship Design would be much like other games in the past - players would pick a ship model they wanted to use and then select what would be on the ship.

The new Colony Manager would be almost like a simulator rather than a game. The numbers would be further obfuscated to feel more organic.

But as we started looking at implementation, we realized that much of our design was half-assed. A cop out. What we knew we needed to do was simply something we hadn't done before -- we needed to go with a 3D engine. A 3D engine would provide us with a lot more flexibility.

The Political Machine

In 2004, Stardock released a political strategy game called *The Political Machine*. It was a half-way house between *Galactic Civilizations* and *Galactic Civilizations II*. It had a 3D engine but didn't have any 3D content! It was basically sprites on a 3D engine. It was also our first game to make use of DesktopX technology for creating user interfaces. It ended up doing well though we were pretty unhappy with reviews (I don't think we'll be making any more $20 games, reviewers don't take into consideration price when they nitpick what features should be in a game).

2004's *The Political Machine* had a 3D engine but used sprites.

But it put us in the position of having an engine to build *Galactic Civilizations II* on. All we had to do is...well...you know...make 3D stuff...

The freedom to make the game you want

By going to a 3D engine, especially one that we had made ourselves, we could start making the game we always wanted. For example, it made it possible to place planets on the main map rather than tucked inside of a "star" icon. Why? Because the 3D engine made it easy for us to let people zoom in and out of the map smoothly. In *GalCiv I*, we had a fixed camera at a fixed distance. So we had to be careful about what we displayed on the map. But with a 3D engine, we could put as much as we wanted on the map (within reason) and let the user adjust what they wanted to view.

Once we settled on a 3D engine things started to progress quickly.

Once we settled on a 3D engine things started to progress quickly.

The 3D engine combined with the DesktopX UI technology allowed us to create a game that visually looked much better than the original. The last piece to the puzzle was ship design. Once we'd gone to 3D, why not go all the way? Why not let people simply design up their own ships however they want? Not just what weapons and engines and life support but let them completely control how their ship looked. Let people go nuts.

Going to 3D allowed us to have ship design that was far more sophisticated than anything done before.

Players could control the colors, the shapes, the sizes, everything about their ships. And because we were so slow to the 3D party, we could have a game that would run faster for most people than the first one did.

With that final decision, we had 18 months to make the game.

What Went Right

1. Publishing Victories. One of the best decisions we made was teaming up with Take 2. We had worked with them before on *The Corporate Machine*. Back then, they were the publisher. But they were always honest, professional, and competent. One thing that many aspiring developers need to be aware of is just how many incompetent people you have to deal with. Without getting too specific, think carefully about the kinds of jobs some people take and imagine "Well, do the best and brightest people grow up and dream of being ?"

Usually, when you're working with large companies of any kind you essentially have isolated gems of super competence surrounded by utterly depressing levels of incompetence. To a small developer, that can be death because you simply can't control how things go. You're at their mercy.

Also by publishing the game on our own, we were free from outside interference. We could put our own ideas into action without having them vetoed. Whether that's a good thing or not remains to be seen.

Now for all I know, Take 2 has those issues too. But if they do, their super competent people have the dead weight hidden in closets or something because the people we worked with were very sharp. For *Galactic Civilizations II*, they were the distributor, not the publisher. Their job was to take our game and put it into stores. We would be responsible for doing all the marketing and taking care of all the market development funding for retail. They helped make the process much smoother by laying out in black and white what needed to be done.

Stardock was able to design game ads without third-party interference.

The business plan we developed to get *Galactic Civilizations II* into retail in significant quantity was put together almost like a cookbook

recipe -- do this, then this, then this and voila.

We also hired Brian Clair, who had been running Avault.com for many years to be in charge of our publishing efforts. Combining him with Take 2 resulted in having a first week sell-in to retail that was 3 times what the original had.

We also partnered with AEG who took over our media relations efforts. The results were immediate and astounding. Vast preview coverage in virtually every publication. This in turn helped our retail efforts to show that there was excitement behind the game and that we could effectively market our game.

2. Artwork. Anyone who's followed Gamasutra for a long time and seen Stardock's games mentioned in the past knows one simple fact: Stardock's games look like crap. Some games looked more crappy than others but art was our weak spot. In 2003, we got tired of being beaten up about how ugly our games were and started leveraging some of the advantages we have.

For example, Stardock runs the world's largest graphics design community -- WinCustomize.com. With over 20 million monthly visitors, it is one of the most popular sites on the Internet. And yet we never tapped into it for our game side. For *Galactic Civilizations II*, we started tapping. Popular artists like Paul Boyer, Mike Bryant, and others began working with us. We also built up our in-staff art department in other ways as well. For instance, I'd like to take this moment to thank Electronic Arts for securing monopoly rights on the NFL franchise of games. By doing so, they eliminated many other football games that were in development which in turn made a lot of very talented artists available.

The result was that we were able to make a game that had competitive graphics for any AAA strategy title. And they're getting better and better.

***Galactic Civilizations II* boasts the best graphics on any Stardock production to date.**

3. Original team intact. The small development team that had made the original *Galactic Civilizations* was intact. Even on the art side, we had no losses other than one person who went to Ensemble to work on *Age of Empires III*. And even there, he didn't go until he had finished modeling all the aliens for the game (i.e. he waited until he finished his major milestone).

This was a very important accomplishment because it meant that the team that knew the game would be able to continue forward with it already knowing the underlying technology and passing that onto new people we brought in.

4. Community Building. Like the first time around, we knew that the secret to the game's success was not in advertising or PR, it was in

word of mouth. This strategy hinged on a number of things:

- No CD copy protection. The best way to win goodwill we feel is to treat your customers with respect. In a single player, turn-based strategy, CD copy protection seemed a little excessive. Why inconvenience them? A lot of people, like myself, feel strongly on this issue. I know I make buying decisions based on whether I have to lug around a CD as if it's a dongle to play the game. And our experience from the first *Galactic Civilizations* confirmed that it does increase positive word of mouth.
- Similarly, we reserved budget so that we could continue adding new features and gameplay tweaks based on player feedback. If players could see their requests and suggestions made real, the hope is that they in turn would feel comfortable recommending the game to other like-minded people. Or put another way, being good to customers seems to make good business sense.
- A new website. The GalCiv2.com site is based on the latest/greatest AJAX/Web 2.0 "stuff". I don't know much on that but I know how interconnected it is and it allows users to gain access levels, win medals, get ranked, etc. for participating in various degrees. The goal was to create a community that in turn would encourage modding and other things that would extend the life of the game.

5. Future Protection. I can't really think of a better term for it. We wanted to make a game that even 2 years from now, would still be viable to purchase and play. Typically, games start to look and feel dated quickly. There are some notable exceptions, particularly in the first-person shooter market.

So we made an engine that would readily support 3 things that we believed would ensure that the game's lifespan was very long:

- Resolution Independence. Using DesktopX, we developed a SmartScaling technology. In essence, the game will run at any resolution at any aspect ratio. And it won't do it through standard scaling/stretching techniques. The DesktopX technology allows different screens to expand at different rates on a per control basis. The bottom line is that even when resolutions are 3000x2000 in a few years, the game will run at those resolutions natively.
- Unlimited Polygon engine. The 3D engine has no cap on polygon count. If your machine can handle it, you can have ships with a million polygons in it. It already supports bump mapping, specular lighting, and every other 3D trick we know of. Textures can be any resolution the designer cares to make them. So over time, more and more advanced models and textures can be put into the game. This really helps in the ship battle screen that one of our developers put together.
- Modding. Of course, the two things above would be weakened if users were at Stardock's mercy to add new models and interfaces. So it was decided that we'd use open formats where ever possible. The UIs are in DesktopX's .dxpack format. The graphics are .PNG, the models .X files and the data .XML. Everything is moddable.

What Went Wrong

For a game that has gotten critically favorable reviews and is selling like crazy, you might think that it was smooth sailing. But it wasn't. More things went wrong on this project than any other I've worked on. That the whole thing didn't fall apart was less a miracle than simply because the development team, individually, stepped up to make sure things went right.

1. Growing Pains. The entire *GalCiv I* development team stayed for *GalCiv II*. Moreover, they all are still here. That's good. But nearly every game developer we picked up between then and now we lost. So as I write this in March 2006, the *GalCiv II* team is a lot like the *GalCiv I* team except we have one additional developer.

So what happened? Each loss has its own story. In some cases, it's location. We're located in Plymouth Michigan. So some internal developers had a significant drive involved. For others, it was money. Contrary to what *Game Developer* magazine prints, the average game developer does not make big bucks. Game developers, on average, make less than developers in other industries. This is for the simple reason that it's more fun to make games than it is to write some database front end. *Some* game developers make a lot of money, but that is not the norm.

There was also a change in culture. Stardock had grown a lot and purchased a brand-new building. Our old place was a bit of a dump but it felt like a really unique small company feel. You might be working in a closet (literally) but it was your closet. The new building is very nice but it's much like any other office in some respect except with nice new furnishings. Some felt we lost a bit of that small company feel. As our staff increased, so did the difficulty of managing that many people. That meant the creation of policies and guidelines that we didn't need before. Some people resented that and eventually decided to leave.

And in some cases, it was personal conflict. With some of the senior level developers who came in, my style of development was in conflict with what they wanted to do. I tended to be more conservative with how I code things. I am less interested in using the latest/greatest STL techniques than I am in having readable code that is easy to look at in a debugger. This created some clashes during development as I would get frustrated when I'd find a bug in something that I found difficult to read. I personally get no joy from coding. I am only interested in making the game work. So I don't care about using some cool technique. I care about algorithms that are robust, reasonably fast, easy to modify, and will still make sense 3 years from now.

My way is not the best way to do things. It is simply one way. Because there's a counter argument to how I tend to do things: From a career point of view, unless you plan to spend the rest of your career making games, you need to learn and keep up with these new techniques so that you don't one day wake up as one of those 50-year-old COBOL programmers who has long since fallen behind the latest techniques.

Over time I did let up a little on this but too late for some of the people who left. But "latest coding techniques" can bring in other issues.

2. Underestimating compatibility with 3D. Our game is very multi-threaded. Unfortunately, some of the latest/greatest C++ goodies are not very thread safe. This is something that would cause us quite a bit of grief. Moreover, it made debugging difficult and time consuming. The project was months behind schedule by October of 2005, just months before shipping. My job at Stardock is running the company -- the whole company. But at that point, the project was so far behind that I ended up stepping in, moving out of my office and into one of the work stations in the main lab right in the middle of the team.

Features were altered and cut. And code thoroughly debugged. But we weren't ready. We thought we were. But we weren't. The game wasn't buggy. But we had no idea of how finicky different computers are with 3D games. We had no idea.

The last 4 weeks before release essentially involved us and our QA team sitting around playing the game all day. We took pride in our reputation of always shipping games that have virtually no support issues whatsoever. And we were probably a little too cocky about that reputation. We bought into the popular mythology that the reason why other games are "buggy" is because they're "rushed". Not us though. We were the developer and publisher. Nobody could pressure us to put out the game until we were ready. One gamma tester even accused us of having "separation anxiety". That we were holding the game back simply because we didn't want to commit to releasing it. It was ready to go.

And yet, within days of release it was clear that some people were having problems. We sold enough units to get a very specific numbers: 2.5%. A small number right? But on *GalCiv I*, that number was less than 0.25%. So that's a 10X difference. What was the problem? Had we missed something?

Well it turned out that we had. For one thing, competition in the 3D hardware arena between ATI and nVidia resulted in video cards that were running a lot closer to their max capacity than previous generations. And our game made heavy use of these new features. As a result, these cards could get hot. *Very* hot. So what would happen is that within a few minutes or a few hours, the game would just unexpectedly crash. We'd get these debug reports and they just didn't makes sense to us. Some simple drawing function would crash. What was going on?

When you combined that problem with the usual piddly bugs along with a not-so-piddly bug with a case where a user could click on a Goto button in a report screen which took them to another screen, which they could then go to another screen, which they could go to another screen, and if they then tried to create a ship and save, the game would crash (too many dialogs -- none of us had ever tried doing that), you ended up with a small but important percentage of people who were unhappy. Most of the reviewers didn't run into it. But if you have a problem that affects 2.5% of people and you have 100 reviews, then you are going to have at least 2 people who are going to have a problem.

We fixed it within the first couple of days of release with a GPU throttling option. Suddenly, by magic, the game became very stable for those people.

The other issue we totally underestimated was how many people don't ever update their video drivers. This continues to be one of the most frustrating problems to deal with. We got a harsh lesson in customer support and the limits of it. Even though the game would pop up a dialog saying "Your video drivers are too old to run this game, you will experience random crashes if you do not update them." (even the 1.0 retail version had that because we knew this from testing) we still had people who would ignore this, have the game crash, go to our forums and post a big "This game is bugggy!!111". It was very frustrating to work through these users only to find they had a 4-year-old video driver that didn't support the new DirectX 9 functionality we were using.

The usual response when we'd ask them why they ignored the warning dialog was "My other games work fine." Other games usually being 4-year-old sprite-based games.

What we learned from these two experiences were:

First, keep adding *new* users to your testing process. As people become experts at the game, they start to play the game differently from others. Yea, there's probably some QA guy reading this saying "Well duh" but it was not something we had done.

Secondly, users won't heed warnings or readmes. Next time, the game will have a dialog that will simply not allow them to play the game until they update their drivers.

3. Publishing: Babes in the woods. You go to some store and they have a big promotion for "Game X" or you go to some game website and they're promoting "Game Y". How do they go about promoting that? I don't know. It seems like I should know that. As a publisher, especially one with oodles of marketing dollars just looking for a place to spend them, I should know this. But I don't. I still don't. And what's worse, we have the money. But we don't know. That's something we're going to have to improve on.

Marketing was much the same way. We learned a lot about the media this time around that I wish I could forget. It's not corrupt. At least, it's not corrupt in terms of reviews -- though if you're a big name game, you'll get a heck of a lot more benefit of the doubt than an indie game.

Make no mistake on that. We couldn't even get preview coverage in all the game mags even though we had AEG as our PR firm. It got consistently better over the months and I think things will continue to get better as we learn more and the powers that be outside the editorial departments realize that Stardock isn't some start-up. We've made a lot of stuff over the years that was published by others. But we didn't do as good of a job at marketing as I would have liked.

We met our sell-in numbers but I would have liked to do more promotion to help push more people to the stores.

4. Are we a board game or a simulator?It's tough being a sell-out. It's less tough being a spineless sell-out which is the one thing that saved me from utter despair. In my strategy games, I like what is called "fuzzy math". I'm not alone in that. I don't like games whose game mechanics are so crisp that there's no intuition involved. But I also know that reviewers generally prefer strategy games to be almost like board games where every number in the game has a simple, straight-forward source.

In a board game, you'd ask: Why is your approval rating 40%? "Oh, because I have a population of 10 but only 4 entertainment centers. 4/10 = .4 = 40%."

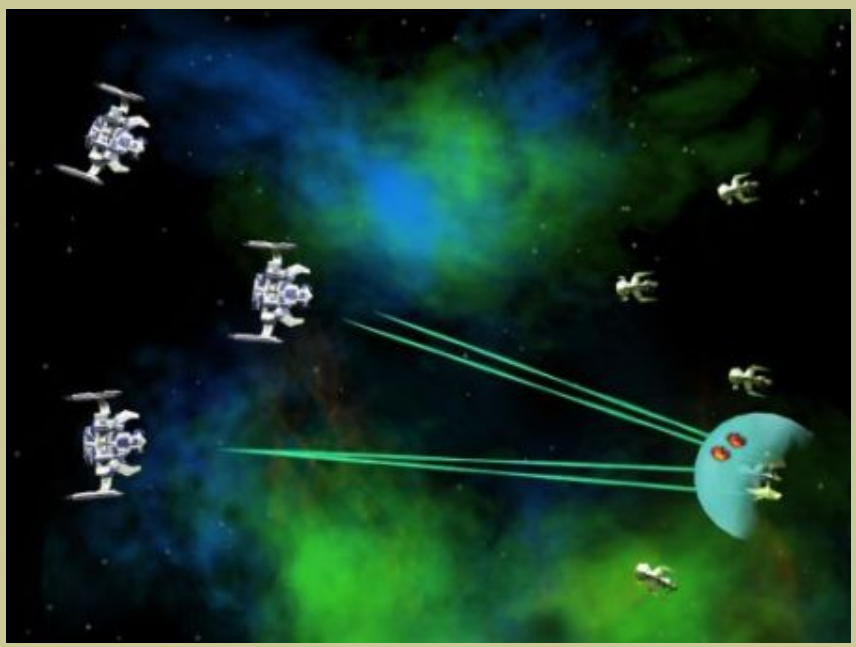
In a simulation, you'd ask "Why is your approval rating 40%?" "Oh, because your population is 10.4 billion, your tax rate is 46%, your morale ability is +20%, you are mining a Harmonic Crystal formation in sector 2,5 which is adding another 15% to your morale ability plus you've built a virtual reality center which is adding another 10%." Huh?

In a board game, you read the manual and you understand the mechanics. In a simulation you play the game and you start to get a feel for how things are interlinked. I prefer the latter but I know that most gamers prefer the former. Since I'm a spineless sell-out, we made *GalCiv II* like the former much more. Factories now produce X industrial units. Research labs produce Y industrial units.

But.. we still have that *Civilization* ability that can be increased in a variety of ways during the game. And that is where we end up having bits of intuitive play come in because you end up having to play-balance things and you can't do it all just by "nerfing" modules or starting values. Sometimes you have to tweak values and the result is fun for most people. But for some people, they really don't like that.

So we'll have to keep improving the game to try to make both camps happy. We want to keep the game from being a spreadsheet but we want to put areas in where people who want can really dig in and find out why every number is what it is if they really want to look it up.

5. IT backbone. Our websites all use technology we developed. Our forums, accounts, articles, libraries, etc. They all use the same stuff. Unfortunately, the popularity of *Galactic Civilizations II* put on a tremendous strain on this that we found ourselves unprepared for. You'd think that companies like us would learn from companies like Blizzard who end up having to beef up their infrastructure after their games ship. But we didn't and so we had to scramble to add more capacity.



Ships can become increasingly complex due to the engine's ability to take on increasingly advanced models.

Lessons Learned

Don't overspecialize developers. Because we had never had anyone leave during a project before, we were unprepared for the consequences of someone leaving mid-game. While no one left suddenly (everyone gave plenty of notice), it put us in the situation of having parts of the game engine that were not well understood by anyone.

Bring in fresh beta testers throughout. This was a key lesson we'll be applying into future games. New testers play the game differently than people who know a lot about the game. The most serious bugs in QA were found by new players.

Test on very high-end hardware. In the postmortem for *GalCiv I* I wrote as one of the key lessons:

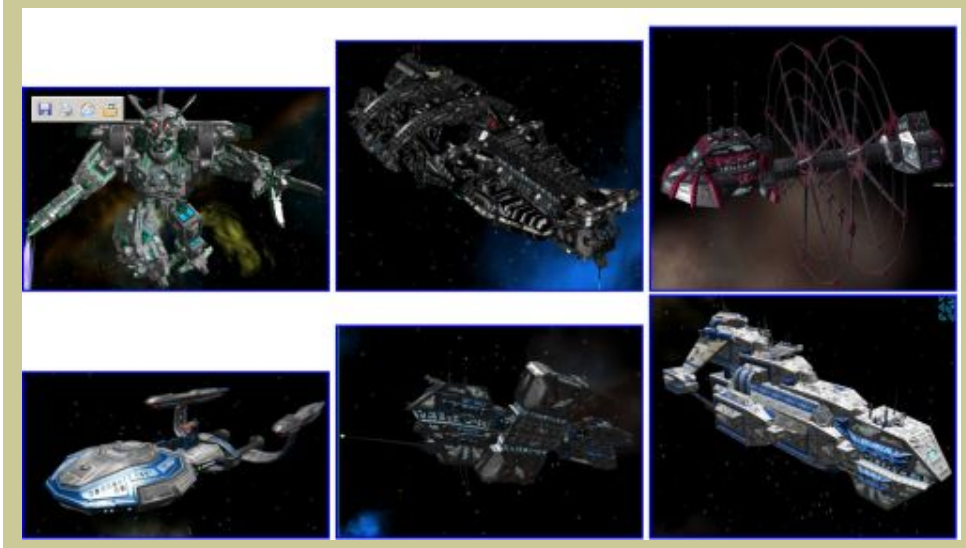
> **Play the game on low-end hardware as much as you can.** *Galactic Civilizations* would have been a pig if it weren't for some of us playing/developing the game at home on our lower-end hardware. Most of my coding was done on a ThinkPad T20 (600Mhz PIII) laptop. Probably about a third of my development time was spent rewriting the internals to improve the speed. For a 2D strategy game, it was very slow game, and I was astonished that the beta testers didn't complain about this more. On the Pentium 4's in our office, the game flew. But on the lower-end hardware, the beta version's frame rate was very slow.

And we took that to heart. Too much to heart. Because we focused so much on working on older, slower machines we missed the group of users who were buying the latest/greatest 3D cards and putting them into their existing computers which weren't well ventilated enough to deal with the GPU getting hot. If we had, the game would have had GPU throttling built in and a lot of problems fixed. For non-developers reading this, GPU throttling doesn't slow down the game, it just puts a sleep(10) call between frames if the frame rate is getting ridiculously high. GPU throttling has no negative on performance. It just keeps the card cooler.

Luckily, things worked out fine, but I would have liked to not had to put out an update to address something that could have been caught if more time was spent checking out the latest hardware.

License more technology. One of the things we learned is that we will likely license more of our technology in the future. For instance, *GalCiv I* had its own built-in sound system. This time, we used the Miles sound system that we licensed from Rad Game Tools. It was much much better and easier to work with.

Features that enable the users to be part of the content creation process are good. The ship designer in *Galactic Civilizations II* has become almost as popular as the actual game. In future games, we will be looking to see what kinds of features we can have that allow users to show off their creativity.



Sampling of ships designed by players using the included in-game content.

Conclusions

Unlike the first *Galactic Civilizations* for Windows which was the smoothest project I'd worked on, this one was the toughest. It was big and ambitious. The sales and reviews and customer feedback indicate that it was worth doing. But it was a real challenge. The first *GalCiv* went so easily because we didn't really have to push ourselves to make it. This one was a real challenge to make and tested our entire company's ability to make something that was truly for the big time. We got a taste of what the difference is between a "B" game and an "A" game. With any luck, *Galactic Civilizations III* will be a AAA game in every sense.

Final UI design

Game Data

Developer: Stardock
Publisher: Stardock
Number of Full-time developers: 12
Number of Contractors: 3
Length of Development: 18 months
Release Date: February 21, 2006
Platform: PC (Windows)
Budget: $1.3M
Development software used: 3D Studio Max, Visual Studio.net, Maya.
Development hardware used: Various PC Compatibles
Notable Technologies: BINK, Stardock 3D engine, Stardock DesktopX, Miles Sound System.
Project Size: ~5 Terrabyte, released game uses around 1.4 Gigabytes.