GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Blue Tongue Software's *Jurassic Park: Operation Genesis*

By Kevin Chan

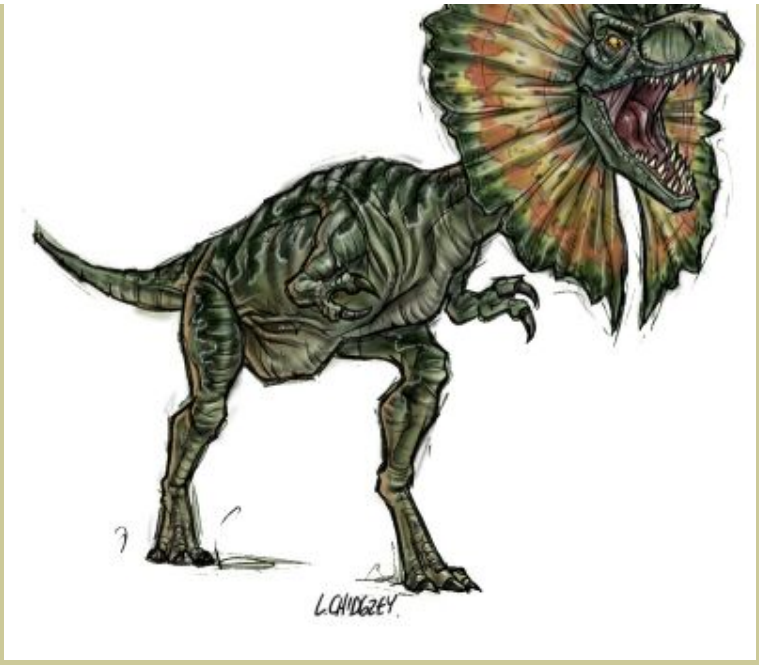
Since its release in 1993, the film *Jurassic Park* has spawned a large number of games. Many of them have been platform games, and almost all of them are about killing dinosaurs. Our publisher, Universal Interactive, wanted *Operation Genesis* to be different. Instead of asking the player to destroy and survive dinosaurs, they wanted to give players the opportunity to create and manage their own Jurassic Park.

All of us at Blue Tongue Software were excited to explore the *Jurassic Park* license as a simulation game. Blue Tongue Software has always picked unique and innovative projects to tackle. Previous projects include *AFL Finals Fever* (the first Australian Rules football game ever) and *Starship Troopers* (based on the book and movie of the same title).

The development of *Operation Genesis* was an exciting and challenging journey. While there is a wealth of console experience within the team, Blue Tongue Software has only developed PC games. So, as a company, we were new to console development. This was also the first time we've simultaneously developed for multiple platforms. The scope of the game also led us to creating a new engine in parallel with creating the game itself. And, to handle the increased scope, we increased the size of the team significantly. On our previous project, *Starship Troopers*, we had 20 in the team. At the end of *Operation Genesis*, the team had grown to about 40 members.

Build fences and attractions in your own Jurassic Park.

While there were a number of challenges, we are happy with what we were able to achieve. We created an original game, for three platforms, in multiple languages, and we did it in less than two years.



Dilophosaurus (concept art).

What Went Right

1. *Jurassic Park* as a simulation game. The *Jurassic Park* books and movies have always contained action. However, a core idea in *Jurassic Park* is the creation of a dinosaur theme park. It is surprising that only one other game has explored *Jurassic Park* as a park building simulation -- Konami's *Jurassic Park III: Park Builder*, which was released after *Operation Genesis* was already in development. We wanted *Operation Genesis* to be a cinematic simulation that would capture the feeling of the movies. Also, while simulation games are more common on PC, we wanted *Operation Genesis* to be as fun on a console as it was on a PC. We feel that *Operation Genesis* succeeded in achieving these goals.

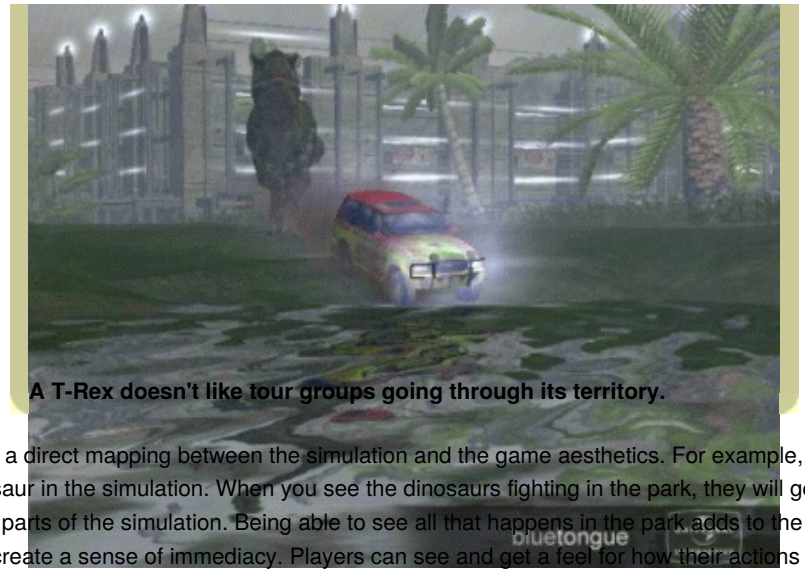
The process of conceiving *Jurassic Park* as a simulation game was fun, and generated an endless source of ideas. It was a vision that inspired the team, and every member of the team contributed to the design. *Operation Genesis* is not tied to any particular *Jurassic Park* movie or book. As a result, we were able to draw from everything in its universe. The game features dinosaurs from the first, second and third movies. To recreate the feel of the movies, characters such as Dr. John Hammond also make appearances in the game.

As *Operation Genesis* is a *Jurassic Park* game, we follow the facts set by the movie. For example, the Velociraptors in the game are human-sized, and the Dilophosauruses can spit venom at their prey. (Note that we later discovered that there is no evidence to suggest that real Dilophosaurus could really spit venom.) However, not everything in *Jurassic Park* fitted into the game. For example, the tiny Compsognathus ("Compys") are familiar to *Jurassic Park* fans, but were left out of the game due to their size. These small dinosaurs would have been very difficult to see, making game play difficult.

In the cases where the movie did not specify the dinosaurs' behavior, we drew information from the real world. Sources used include *The Dinosauria* by Weishampel, Dodson and Osmolska, and *Terrestrial Ecosystems Through Time: Evolutionary Paleocology of Terrestrial Plants and Animals* by Behrensmeyer and Sues. While we were not striving to be educational, the Dinopedia in the game provides some real dinosaur facts to the player. We wanted to give a rewarding experience to both fans of the movies, and fans of dinosaurs.

We looked at many issues when designing *Operation Genesis*. One issue was finding that balance between order and chaos. Jurassic Park as depicted in the film and books is about the loss of control and the chaos that ensues. However, players must be able to create order if their park is to be a success. Will the player enjoy Jurassic Park if everything is running well? This part of the design was one of the more difficult issues. If dinosaurs break out constantly, the player may feel that they have no control over their park, and feel frustrated. However, if the dinosaurs break out too rarely, the park will feel too safe and will not feel like the Jurassic Park people read about and saw in the films.

To address the issue of breakouts, we created various security measures that the player can use. The player can research and upgrade to stronger fences. This enables the player to increase their security as the game proceeds. However, to prevent the park from becoming too safe in the later stages, weather effects such as tornados and thunder storms can destroy sections of fence and stress the dinosaurs, causing them to rampage.



A T-Rex doesn't like tour groups going through its territory.

In *Operation Genesis*, there is a direct mapping between the simulation and the game aesthetics. For example, seeing one dinosaur in your park means there is one dinosaur in the simulation. When you see the dinosaurs fighting in the park, they will get hurt or injured in the simulation. This extends to all parts of the simulation. Being able to see all that happens in the park adds to the cinematic feel of the simulation, and also helps to create a sense of immediacy. Players can see and get a feel for how their actions affect the park.

Also, wherever possible, the game lets the players jump in and manipulate the park directly. For example, when a dinosaur breaks out, players can fly the helicopter out to sedate it. Or, if the dinosaurs are playing too far away from the viewing tower, players can use the helicopter to herd the dinosaurs closer to the viewing tower so that the visitors can see them. This level of direct manipulation caters to console players. However, for players who just want to manage the park, these kinds of operations can also be done automatically. For example, you can issue a sedate order, and an AI Ranger will fly out and sedate the dinosaur for you. This style of play is often found in PC simulation games. By supporting both styles, players can choose the level of interaction they want.

2. Toshi, the engine that could. Toshi is the new engine that we developed for*Operation Genesis*. It was designed from the ground up to be multi-platform, expandable, and customizable. We wanted the engine to be as reusable as possible.

Consequently, Toshi's design uses a component architecture. Components such as shaders (used for rendering), game creatures, and GUI objects are all just plug-ins to Toshi, making the entire system very modular and extendible. As it was built in parallel to *Operation Genesis*, the initial component "toolkit" was geared towards photo realistic outdoor environments, dense foliage and skinned creatures.

Toshi was also designed to work on multiple platforms from the beginning. This is a key feature that made it possible for us to develop for three platforms at the same time. Given the independent nature of components, the platform-dependent (primarily rendering) components could be implemented in parallel. This design also enabled new, enhanced versions of these components to be plugged in as they became ready. Like other cross-platform engines, Toshi enables the same application code to work across all platforms.

One of the most successful parts of Toshi is the animation system. The dinosaurs in*Operation Genesis* can blend ten or more animation sequences at a time. Often, these sequences also have sub-component animations driven by the AI system. This enables the dinosaurs to walk, look around, and blink all at the same time. This flexibility creates natural looking movements, and means that a dinosaur might do the same behavior slightly differently each time. The system went through a number of iterations during the course of the project. The final version was very fast, required a tiny memory footprint, and ran equally well on all three platforms.

As we were developing our engine at the same time as the game, we faced the risk of having major production bottlenecks. Our solution was to use prototyping and placeholders to keep the game development going. This was used in a number of situations and was perhaps the main reason we were able to develop the engine in parallel to the game without serious scheduling problems.

For example, before Toshi was rendering the structures, visitors and dinosaurs, a placeholder renderer was made to allow the development of AI systems to proceed in parallel. This renderer used text and icons to display the current state of each unit, enabling the game and AI architecture to be tested and tweaked. So a large part of the game architecture was fully functional before Toshi was rendering the objects on screen.

Another instance where we used prototyping and placeholders revolved around the path-finding system for dinosaurs. Early in development, we used a system we developed for *Starship Troopers* as a placeholder to allow the AI development to continue. But this solution was not adequate; it didn't cope with terrain modifications made by terraforming operations, and it made some assumptions about bipedal objects (which does not apply to the quadrupeds in *Operation Genesis*). However, using this old system as a place holder was enough to keep AI development going while a more fitting solution was prototyped. Since the prototype was developed using the same interfaces as the placeholder, it was a simple task to drop the final path-finding system into *Operation Genesis*.

Toshi supports powerful terrain rendering features. In the early stages of the project, procedural terrain texturing was prototyped using a software-rendering scheme. This allowed us to rapidly try out a number of different ideas. Care had to be taken as we needed to ensure that the design selected was achievable on the hardware of all three platforms. Having a prototype also allowed us to streamline asset creation, since artists could visualize their work months before the final system was implemented.

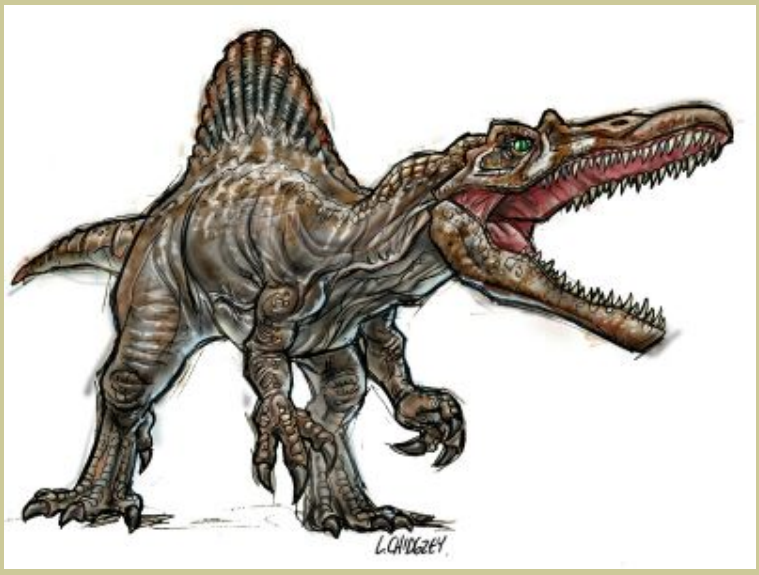
Prototyping was used extensively during the development of *Operation Genesis*; all high-risk components were prototyped. This was a valuable tool, enabling us to mitigate many risks very early in development.

T-Rex having a morning drink.

3. The Art of *Jurassic Park*. It was important that *Operation Genesis* capture the aesthetics created by the *Jurassic Park* films. Almost all of our players are familiar with the movies and we need to give them the *Jurassic Park* that they know. The artists on *Operation Genesis* did an outstanding job in accomplishing this goal. I believe the dinosaurs in *Operation Genesis* are some of the best dinosaurs seen in any game.

When people think of dinosaurs, they usually think of the colorful and lively dinosaur from the *Jurassic Park* movies and books. *Jurassic Park* did a lot to update the public's perception of dinosaurs; before *Jurassic Park*, most of the public saw dinosaurs as slow, gray creatures. Although we were creating *Jurassic Park* dinosaurs, the way each dinosaur should look was still not clearly defined. The appearance of the dinosaurs changed between the three different films -- the dinosaurs became more colorful in the later films. Because of this, the dinosaurs in *Operation Genesis* went through a number of iterations as we settled on a style. We initially used designs from each of the three films, but settled primarily on designs seen in *Jurassic Park 3*.

Our artists gave each dinosaur a number of special animations, imbuing each species with its own individual character. For example, the Triceratops will locks horns with each other in a playful show of strength, and the Velociraptors will climb the fences to escape. These kinds of animations breathed life into the dinosaurs of *Operation Genesis*, making them interesting to watch. This was important, as we decided early in development that the visitors in the simulation should only be entertained by things that will also entertain the player. I think these interactions worked well in the game. Quite a few members of the team were seen locking the camera onto their favorite dinosaur, and leaving the game running like a fish-tank. (This is the reason we created the "Site B" mode of the game. In this mode, the player does not need to worry about managing a park; the player can create herds of dinosaurs and just watch them.)



Spinosaurus (concept art).

Getting the sound and music right are also crucial for bringing the player to Jurassic Park. With the dinosaur sounds, we used the sounds

from the movie when possible. For example, when the Velociraptors call to each other, they use those evil sounding barks from *Jurassic Park 3*. And, the T-Rex has the familiar T-Rex roar from the film. For dinosaurs that did not appear in the movie (or did not get enough screen time to show us their calls), we constructed the calls by blending together real animal sounds. In some cases, unusual bird sounds worked well without too much manipulation or alteration. The sounds we created were then tailored and reworked to fit the character of each dinosaur. We used the sounds to highlight the character of each dinosaur, while keeping them consistent with the world of Jurassic Park.

Another aspect of art that worked well is the tools. The export process enabled the art team to drop new assets into the game without needing to create new builds. This enabled assets to be polished quickly, and also enabled the artists to work independently of the programmers. We also created tools for tweaking special effects such as particles and weather. Because these effects were data driven, it enabled the art team to fine tune them without needing to wait for new builds.

4. Data-Driven Game Play. Almost every aspect of the game could be tweaked via spreadsheets, from weather settings to the AI for all the different units. This enabled a fast turn around in tweaking the game balance as we did not need to wait for new builds to test the changes. It also enabled balancing responsibility to be delegated to the team members who are most qualified to do it. For example, tweaking the walking and turning speeds of the dinosaurs was best done by the animators who created those motions.

The data-driven aspects of the design also extend to unit definitions as well. All the units in the game (dinosaurs, visitors, buildings) are bound to the game in a dynamic way. They are like self-contained plug-ins to the game, and interact with the game world and other game objects via well-defined protocols. This allowed us to develop all of the game units concurrently. Also, as the game evolved, this system enabled new units to be created and plugged-in easily. In a way, all units in *Operation Genesis* are "add-ons" to our initially empty environment.

The data-driven approach required some initial investment. However, the time spent creating these systems was saved many times over, and allowed us to focus on game balancing and tailoring individuality into creatures. Having data-driven unit definitions also opens up the possibility of creating expansion packs with more dinosaurs, objects and buildings.

5. Artificial Intelligence. As mentioned earlier, *Operation Genesis* is a game where the simulation is tied to the aesthetics. The game is not purely driven by stats. We do not use rules such as, "One carnivore feeder will support two T-Rexes," implying a model where as long as there's one feeder within the enclosure the two T-Rexes will not get hungry. Instead, when a T-Rex gets hungry, it will search for suitable prey, chase it down, and eat it. Enabling the dinosaurs to do this makes the game play much more immediate, and also makes the AI system critical. As a result, we began work on AI very early in development. The systems worked out well.

Jurassic Park from a visitor's point of view.

The dinosaurs in *Operation Genesis* have drives and needs like real animals. When their needs are not being fulfilled, the dinosaurs will take action to satisfy their needs. They can perceive their environment, and know how to interact with other objects/creatures. Like many other aspects of the game, the dinosaur AI is data driven, enabling the dinosaur behavior to be tweaked rapidly.

The dinosaur AI is separated into three distinct systems: the neural net perception system, the behavioral drive system, and the behavioral execution system.

Each object in the game has a clearly defined set of traits. When a dinosaur perceives an object, it uses these traits as inputs into its neural net. The neural net will then classify the object into a category that is relevant to the dinosaur. For example, it may classify the object as a "threat", "friend", "food", and so on. The most exciting feature of this system is that it enables dinosaurs to respond appropriately to new objects/creatures that they have never encountered before. However, these neural net computations can be expensive. As a result, a caching system is used on the neural net results to reduce the computational load.

While the traits can be tweaked to create the behavior we desire, the neural net outputs are not 100% predictable. This means that the dinosaurs do not always behave as we predict. However, this works in the game because there is no right or wrong responses. Hypothetically, even if a small dinosaur decides to attack a larger one, it may be seen as brave or foolish, but not necessarily incorrect.

Once the dinosaur has classified the object it has perceived, the classification is sent to the behavioral drive system. The drive system will then determine which drive is the most dominant at this point in time. While only a subset of the dinosaur's drives is visible to the player, the set includes all the possible behavioral motivations that the dinosaur may have. These include hunger, thirst, the desire to hunt, the desire for territory, the desire to socialize, and the desire to flee from predators. These drives are grouped and prioritized. Important drives such as self-preservation are always given the highest priority.

Finally, each drive has a hierarchical finite state machine associated with it. These specify and carry out the actions needed to fulfill that drive. (For example, the actions that the dinosaur should take to find suitable food.) Different dinosaur species fulfill these drives in their own unique way, giving each of them their own personality.

The AI system also enables some interesting, emergent, group behaviors. For example, when a T-Rex approaches a flock of Dryosaurus, the flock will scatter and flee in all directions, as the Dryosaurus drive for self-preservation overrides the drive to flock and socialize. These emergent interactions add life and complexity to the dinosaurs.

Besides the dinosaurs, visitor AI is also an important part of the game. The visitor systems also have some interesting features. The visitors determine how well your park is doing. If a large number of visitors see attractions that they like, the park's star rating will go up. If a large number of visitors complain, the star rating will go down. For efficiency, each attraction (for example, a viewing tower) has a "performance analyzer" attached to it. The analyzer will look at the scene visible to the attraction and rate it. The analyzer scores are updated regularly as the dinosaurs move around and change their behaviors. Visitors can check the analyzer score and decide how much they enjoyed the attraction. This avoids the inefficiencies of having every visitor analyze the scene before them.

Although the performance analyzers started as a way to improve efficiency, they eventually turned into a part of the game play. The player is able to select an attraction and see its performance. The analyzer score depends on what thrilling, fun or historically accurate entertainment is visible to the analyzer. This gives direct feedback to the player on how they can improve each attraction.

Visitors generally stay on the road network. (The exception is when dinosaurs are loose. Sometimes, panicked visitors ignore the rules and run off roads to try to escape being eaten. In this case, the visitors use the dinosaur steering system.) Because of this, we store proximity information about buildings on the road network. This makes the visitor pathfinding extremely efficient. The roads themselves can trickle information across the network, keeping the CPU load down. When visitors come to intersections they can use this information to determine what attractions are close by without requiring each visitor to search the road network again.

There is also an interior steering system that visitors can latch onto for entering buildings. Some buildings have interior paths that were specified by the artists. This system allows the visitors to navigate perfectly around the complex building geometry. The paths also hook up to the logic of the building, making it trivial for buildings to have doors that open when visitors come up to them. In addition, the paths can specify turning direction and animations (such as sitting down at benches in the picnic areas), as well as calling routines that affect the visitor's state (such as buying food from a kiosk when the visitor gets to the front of the line).

Hervibores like to socialize with each other.

What Went Wrong

1. Three Platforms at once. *Operation Genesis* was developed simultaneously for PS2, Xbox and PC. This was the first time we developed a game for multiple platforms simultaneously. While this presented a number of challenges, the approach we took enabled us to deliver all three versions successfully.

As mentioned earlier, Toshi was built from the ground up to support multiple platforms. The game code was largely platform independent, and the platform-dependent engine components were designed as plug-ins that could be refined during development. This enabled the development of the game code and the platform-specific engine code for each platform to proceed in parallel.

The build environment was set up so that team members could have all three versions compiled on their machines without one version clobbering another. Also, the development kits for PS2 and Xbox were connected to the LAN, enabling any team member to use them. It meant that team members could run the game on every platform even if they did not have the hardware for every platform at their desk. This enabled programmers and artists to develop and test on all three platforms regularly. While this set up enabled us to work on multiple platforms simultaneously, there were a few bumps along the way.

Sound designer and music composer Stephan Schütze using his sound equipment. He insists that he needs every one of those buttons.

Because the game code was mostly platform-independent, we had one copy of the game code in version control, and used it for all platforms. (#ifdef's were used in the places that had platform-dependent code.) We did not keep separate copies of the code for each platform. This was helpful as it meant that we did not need to duplicate changes across multiple copies of the code. However, it also meant that changes to the game code would immediately affect all platforms.

Because of this, there were some occasions early in development where code was tested on one platform, checked in, and broke the game on another platform. The types of problems that manifested ranged from graphical glitches to occasional crashes. (These were usually fairly easy to fix once the cause was found.) After being stung once or twice, we implemented a number of processes to avoid this problem.

Although this was the procedure from the start, we became stricter with insisting that code be tested across platforms before it was checked in. While the team followed this rule for the most part, there were some hurdles that made this difficult. First, compile times were long. Just linking the PS2 or PC build would take several minutes! Second, there was an initial lack of development kits for the consoles. This lead to some contention for resources, especially access to the PS2 development kits. As the project progressed, the number of kits increased, but there never seemed to be enough. So, while we asked everyone to test on all platforms before checking in, it was sometimes difficult, especially in cases where a number of team members were waiting for a small change to be checked in before they could proceed. (The time to compile and test on the various platforms often far exceeded the time to make the change.)

To supplement the rule of testing on all platforms, we also asked all team members (programmers and artists) to keep a log of changes

they've made. The logs were written in a parse-friendly format, and checked into version control. A script was run daily to merge all the logs into a report. The report helped to keep the team informed of what other people were doing, and, in the late stages of development, it also helped in tracking the cause of new bugs. While these kinds of logs would be useful in any project, they were especially useful in finding bugs that were caused by not testing across all platforms. (For example, bug "fixes" that remove the bug on one platform, but introduce another on a different platform.)

Acrocanthosaurus catches a Dryosaurus for dinner.

Another challenge was dealing with the different capabilities and limitations of each platform. The differences that affected us the most were the different graphical capabilities and the different amounts of memory in each platform. While we wanted to keep the three versions as similar as possible, we also wanted to use each platform to its full potential. For example, the virtually limitless amounts of memory available to PCs enable them to support a highly detailed model set. This created some extra work as we needed to create and maintain multiple versions of the assets. However, we held off on multiple versions of assets for as long as we could to ensure that the assets created would be fairly final, avoiding the need to rework multiple versions of an asset.

Developing for multiple platforms simultaneously also made some milestone days a nightmare. There were occasions when we released seven different builds for a single milestone. There were versions for each platform and versions for different languages (US, Europe and Japan). As the project progressed, we organized staggered deliveries, delivering two or three builds on a milestone day. For example, we would deliver the US builds for all three platforms, or alternatively deliver the builds for a single platform in all languages. Staggering deliveries helped reduce the load on our internal QA.

2. Is it compiled yet? The size of *Operation Genesis* and the compilers we used meant some long waits for compilation. For PC, on some of the development machines, just linking the project could take several minutes. This led to some long debug-compile-test cycles. This was especially problematic as we were developing on three platforms simultaneously, and we wanted the team to test on all platforms before checking in.

There were a few days where a large number of system

changes were checked in. On these days, numerous full recompiles were required, and the team spent a lot of time waiting. Long compile times were also an issue in the days leading up to milestones as internal QA spent a lot of time waiting for new builds with the fixes needed.

Shane Stevens, the lead programmer, checks out some new artwork.

Close to the end of the project, we found that it was possible to speed up compilations for the PC build significantly using Microsoft's new .NET compiler. However, we deemed the risk of switching tools at that late stage too high, and stayed with the tools we were using.

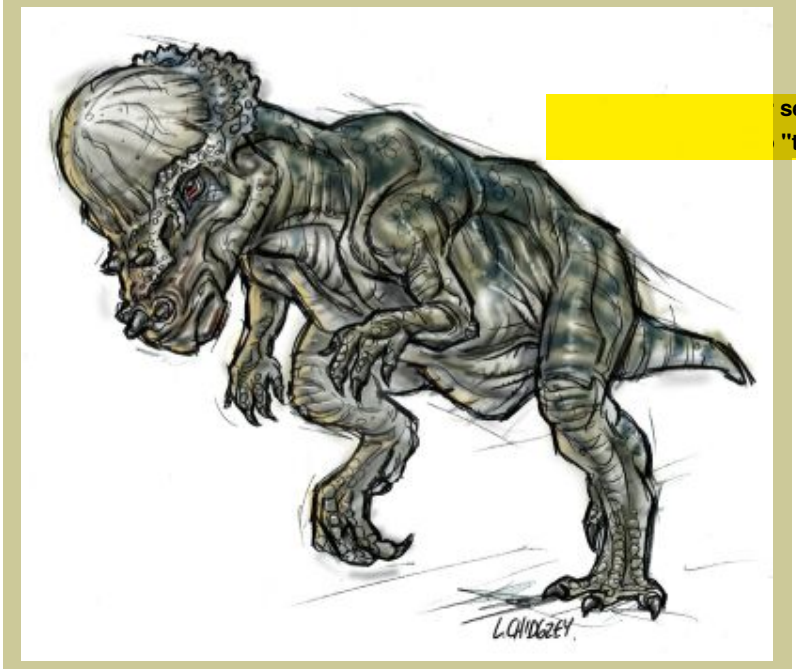
3. Over-generalization. From the beginning, the architecture and design of Toshi were kept very open. Some of the features were not ultimately used in *Operation Genesis*. And, in the later stages of the project, we re-engineered parts of the engine to increase its performance and improve memory usage.

For example, the animation system went through a number of iterations. With each iteration, we dropped some of the features that were not utilized and optimized the data structures further so that we could fit more animations for dinosaurs into the limited memory available on the consoles.


senior programmer, takes a "test".

Another example is the AI system. The systems that were built were very powerful, but there simply wasn't time to take full advantage of their capabilities. For instance, the data-driven neural net system used for the dinosaurs' perception lends itself perfectly to in-game learning. However, in-game learning did not fully fit the game design, and the situations where it could be applied were very contrived. As a result, we did not use in-game learning even though it is supported.

There is a conflict between designing general, open-end technologies that can be reused in future projects, and building specialized technology for the current project. As Toshi was a new engine, there was always a tendency to give it more features, and make it the best. While this did not create any serious problems for *Operation Genesis* (apart from needing to re-work some parts), it may be a good idea to use prototyping even more in future projects, starting with simple versions, and adding more functionality as the requirements become finalized.


Pachycephalosaurus (concept art).

4. Prototyping and the Media. While it's a good engineering tool, prototyping isn't really compatible with the notion of press releases. As mentioned earlier, prototypes and placeholders were used extensively early in development.

As we were developing Toshi concurrently with*Operation Genesis*, we organized the schedule to minimize bottlenecks and to keep development in parallel. However, this plan meant that enhancements to rendering and other aesthetic boosts were often placed at a lower

priority. (There were some bursts of aesthetic work for events such as the E3 Xbox demo.)

This meant that there were times when the game was not looking at good as it could, even though it may have only taken a day or two to make the aesthetic tweaks. As a result, we faced some criticism of rendering quality (and draw distance) based on some early incarnations of our rendering components.

5. Features lost. As with all games, there are always features that were left out due to schedule or other reasons. With *Operation Genesis*, we probably had enough ideas to fill multiple games. I'd like to use this section to lament a few features that were lost.

The Blue Tongue team.

Very early in the project, we dreamed of 40 dinosaur species. However, as development proceeded, the list was reduced to 25. The ones that we miss most are the sea-dwelling and flying dinosaurs. The main reason they were lost was schedule. In some sense, it was a choice between creating fewer, more interesting dinosaurs, or a larger number of less interesting dinosaurs. We decided to make the dinosaurs we had count, and focused on species that players would recognize from the films. However, models and art were created for some of the lost dinosaurs. We'll miss them.

We experimented with baby dinosaurs midway through the project. We tried schemes such as scaling the adult dinosaurs down to make them into babies. However, scaling adults down did not look right. Making baby dinosaurs that looked good required new baby dinosaur models and baby AI. This is similar to the work required for adding more dinosaur species. As a result, baby dinosaurs were dropped. Along with them, we also dropped any ideas for a "Dino Petting Zoo".

There were also some buildings that did not make it into the final game. These buildings include the hotel, hunting platform, and the dino-vet station. Some of these buildings existed in earlier builds of the game. Some were removed due to design decisions, and some were removed due to the memory constraints on the consoles. Having seen and played the game with some of these buildings, losing them was a difficult decision to make.

Operation Genesis evolved throughout the development period, and these features were cut for good reasons. However, if there's ever an expansion pack or sequel, I'm hoping that some of these features will make an appearance.

Conclusion

Operation Genesis was bigger than our previous projects, and was our first experience at developing for multiple platforms simultaneously. While there are still features that we would like to add and tweak, we finished with a product that we are happy with, and hope that players of all ages can enjoy.

Credit must go to the entire *Operation Genesis* team for all their talent, dedication, and hard work. Every team member made contributions to many and all areas of the game. I must also give credit to the many members of the team who contributed to the post-mortem. You input was invaluable, and gave me an opportunity to see the project from many points of view.

Game Data

Jurassic Park: Operation Genesis

Developer: Blue Tongue Interactive

Publisher: Universal Interactive
Number of Full-time developers: Varied throughout project, but 26 at ship date.
Length of Development: 22 months
Platform: PS2, Xbox, PC
Release Date: March 12, 2003 (PC), March 28, 2003 (PS2/Xbox)
Development Software: MS Visual Studio, 3DS Max, SN Systems (PS2), Visual SourceSafe, Test Track Pro, Photoshop, Excel.
Development Hardware: Ranged over course of development from 800MHz-2.2GHz CPUs, with 256-1024MB RAM
Notable Technologies: fmod, Bink, CRI.

Return to the full version of this article