Postmortem: Shiny Entertainment's *Wild 9*

By Didier Malenfant

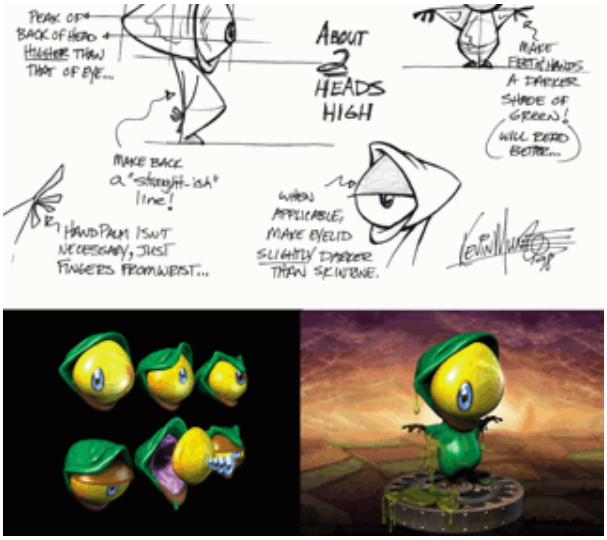
Shiny Entertainment is well known for its successful console platform games. And yet, huge hits such as *Earthworm Jim* and its sequel have resulted in huge expectations for future projects within this genre. Game players who might forgive minor imperfections in an innovative PC title, such as Shiny's *MDK*, are less understanding of faults in a platform game. The Golden Age of platform games, ushered in by the NES, the Super NES, and the Megadrives consoles, established some very high standards for this genre. In this environment, our team at Shiny launched the development of *Wild 9*.

Like None Before

We were fortunate in that Tom Tanaka, *Wild 9*'s lead designer, had been one of *Earthworm Jim*'s designers. His experience and love for the platform genre guided the project's efforts from the beginning. During some of the early design meetings, we decided that one way to differentiate our platform game would be to come up with creative ways of getting rid of the enemies — most platform games involve some form of the player jumping onto the enemies' heads. So from the very start, we designed *Wild 9* to be different in this respect. From this beginning point, we realized that the main character would have to possess some kind of weapon that could eliminate the baddies in a lot of different ways.

David Perry, Shiny's president, took an interest in our initial design and challenged us to create a weapon like none that had been attempted before in a videogame. Our first ideas centered on a female character wielding a glove that could remotely vaporize enemies or objects. This glove would deliver the same kind of actions you could perform with your own hands, only with a thousand times more strength. And then, somewhere in the early design stages, we realized that this wasn't a weapon at all. The glove evolved into a beam that came out of the back of the main character's suit. This system was to take center stage in the game's design and eventually was dubbed "The Rig." *Wild 9* was to become one of the most violent games around, but ironically the main character was to have no weapon.

Kevin Munroe, character and story designer and lead animator on the project, expanded Tom's first attempts and created a whole universe and storyline for the game. The main character became Wex Major, a male teenager lost in another galaxy. Wex's encounter with The Rig gives him the opportunity to fight the evil creature controlling the planet upon which he's just landed. As Kevin described his own design, "Imagine if George Lucas co-wrote Star Wars with Lewis Carroll. And imagine if George Lucas then codirected it with Tex Avery." Wex soon finds allies (eight of them), and together the *Wild 9* embark upon a "David vs. Goliath" battle against Karn, a 376-year-old being with the power of a god and the temperament of a toddler. Karn has set his sights on harnessing the ultimate power of The Glove and Rig, as well as the only being capable of using them: Wex Major. From his humble anti-hero beginnings as a pizza boy in earlier designs, Wex was now the planet's only hope, and the task ahead of him was nothing short of incredible.



Concept art for the Little Evil Green Men

Because Shiny had had luck with licensing its properties in the past, we created a game bible that contained the character profiles, as well as multiple sketches of all the characters in the game. This document was then used to show production studios and other interested parties the game universe from which TV shows or toy lines could eventually draw. As David had the chance to say in multiple speeches he made on the subject, a game bible is invaluable in the quest to license your game worlds and characters. In our case, it was also a useful reference to the huge database from which we were going to create a game. Kevin and Tom designed a lot of game content — to the point where we nearly had to lock them in Shiny's basement to stop them from adding anything else. In addition to the main character Wex, the designers came up with a sidekick named B'Angus; Nitro, who is allergic to everything and has a bad tendency to explode when he sneezes; Henry, who helps navigate the water levels (Wex doesn't know how to swim); Crystal and Boomer, the game's female characters; the multiple bad guys grunts Wex encounters along the way; Karn, the evil giant who faces Wex at the end of the game; the famous Little Evil Green Men (or LEGM as we later called the tiny, single-eyed green pests); Filbert the sniper; and many other characters, as well as the world they inhabit.

The initial game bible only served as *Wild 9*'s starting point, because we added a number of elements to the design as we went along (we even managed to finish some of them). Still, this document proved to be such a valuable resource that all of Shiny's future titles will start with the development of a game bible. Many times, titles are late because certain gaps in the original design were overlooked and the full design was never really laid down on paper before development started. Writing down design details forces the designers to make all the micro-decisions that the design encompasses and leaves less room for interpretation or hesitations from the rest of the team.

Wild 9's development began around the time that the first all-around 3D platform games had started to appear. Faced with this emerging trend, we had to make a choice whether to follow it or to stick to more traditional 2D-based game play, even if it meant evolving in a 3D environment. The second solution was chosen mainly because of The Rig. We wanted full freedom for the beam to move, and that meant using the full range of joypad inputs just to achieve that goal, never mind running around in 3D. So we decided that the main character would travel on a predefined path that would roam through a 3D universe, involving changes of direction and multiple camera angles. This decision also allowed the level designers to carefully place camera angles along the character's path, thus enabling dramatic scenes and varied views of the scenery. Later, we used the camera to give players more awareness of their surroundings and warn them of the dangers ahead by panning the camera one way or another.

The designers chose to follow a traditional 2D environment to allow full freedom of movement for The Rig, the beam-like main weapon

While *Wild 9*'s design was challenging from the start, we also wanted the game to push new limits on the technical front. We wanted to deliver not only great and innovative game play, but also amazing visuals. The results were a mixed bag, and our technological effort was probably the most tumultuous part of the game's development.

The Tools and the Talent

Wild 9 was originally targeted at two platforms; the Sony PlayStation and the Sega Saturn. We eventually dropped the Saturn version, much to the disappointment of many Sega fans worldwide. Still, our original plan influenced the technical side of the project throughout the course of the game's development.

Characters and levels in the game were modeled in 3D Studio Max and then exported using a proprietary plug-in that Malachy Duffin wrote. In order to account for both platforms, Malachy first wrote the export plug-in to handle both output formats, and the scripts that drove the export process contained lines referring to both the PlayStation and the Saturn. When we dropped the Saturn version, half of the data in these scripts became obsolete; from that point on, only the PlayStation side continued to evolve.

The export plug-in handles the character's geometry, which is stored as a certain number of limbs, and animation, which is stored as translation and rotations of those limbs. This system has a small memory footprint while remaining flexible enough for our purposes. (We didn't use single skin meshes in the game because the on-screen size of our characters would have rendered this refinement invisible.) One of the game engine's features was the ability to hide and unhide specific limbs, which was pretty handy in optimizing the game's rendering time. One of the tricks we used was to produce multiple level of detail (LOD) models of certain objects, and store those LODs as separate limbs within the same character file. We could then hide and unhide LOD limbs depending on an object's distance from the viewpoint; once again, the small size of our objects on the screen made the popping from one LOD to another minimal.

One of the tricks *Wild 9* used was to produce multiple level of detail models of certain objects, and store those LODs as separate limbs within the same character file.

The levels and characters also contain various invisible bits of information, such as trigger points, reference points, and collision boxes. A level's trigger points generate monsters and powers ups. A model's reference points provide anchors for multiple effects. If a character, such as the alien beast in the Beast Engine level, has smoke coming out of its nostrils, then these will be generated from a reference point placed on the character model. The reference point can be hidden and shown according to a visibility track placed in the model's animation in 3D Studio Max, which makes editing them simple. Of course, things are never too simple. We encountered some problems toward the end of the project when time came to get the PAL version of *Wild 9* going. The game's final level, which features an encounter between Wex and Karn, involves Karn trying to catch up to a running Wex. Because Karn is such an imposing fellow, his footsteps generate a puff of smoke and a loud bang; we generated these footsteps with reference points placed on Karn's feet. A problem arose with the PAL version because the animations played at a different speed to compensate for the 50Hz frame rate. Some of the reference points were skipped altogether, and Karn would appear to limp as only one leg would make a sound. We solved this problem by making the reference points active for multiple frames and preventing the game engine from processing those two frames in a row. If one reference point was skipped, we were sure to catch at least one another from the same group. It wasn't a very elegant solution, but it served its purpose late in the project.

All in all, 3D Studio Max was a decent editing environment for Stuart Roch, Lori Perkins, and Rich Neves to work with, although a dedicated editor would probably have been better in some cases. 3D Studio Max offered near-total layout freedom, but was lacking in the ease-of-use department, considering that we weren't even using a tenth of its features for our level-building and real-time character purposes. One feature that was very useful to us, however, was the user-defined properties it allows you to input for each object. By including a lot of keywords and properties within each game object, we could easily pass on the information that the export plug-in needed to and process its output accurately and efficiently.

My assignment for the project was to program the game's intermediate levels, which consisted of jetbike races chasing bad guys in various landscapes, falling down a huge tube trying to avoid projectiles, and the game's final encounter between Wex and Karn.

The tools and the game engine also allowed for animated textures and particles, which the team's artists used extensively. Jean-Michel Ringuet used animated textures in the Crystal Maze level, for example, to give crystals a extremely realistic glow. Erik Drageset provided levels such as Drench, which featured a series of multiple water falls and water effects — it had us all in awe the first time we saw it. In the meantime, Lloyd Murphy was working on Wreckage with the secret intention of making it the most graphically intensive level in the whole game. He kept that record to the end of the project with only a few kilobytes of memory and a few bytes of video memory left when this level was running. However, toward the end of the project, when we'd add a game element that would appear on all the levels (such as the in-game information panel), Wreckage would be the first level to break and run out of memory. We used the particle system extensively throughout the game for explosions and various pyrotechnics. Here too, unexpected usage of the particle system proved very effective in the Drench level, where raindrops fall and create very realistic ripples into the water — all done using different types of particles.

The tools and the game engine also allowed for animated textures and particles to be used extensively in *Wild 9.*

Two factors led to the literal flourishing of particles everywhere in the game. Gregg Tavares, who started as a programmer but left the project midway through development, added the ability to create and use particles from inside the game's scripting language. Furthermore, John Alvarado built his now famous Particle Studio module. This module allowed us to preview the effects of the particle system's many parameters while modifying them in real-time using the joypad.

These kinds of tools, together with our scripting language, offered our artists a creative freedom that paid off for us in a big way; it's likely to serve as a model for our future development efforts. Great and unexpected things resulted when the artists were able to try out new ideas out without involving the programmers. A typical scene involved a bunch of programmers looking at the artist's screen and saying something along the lines of, "I didn't know it could even do that."

On the other side, every time a task required the programmers' help, synchronization became a major problem, with the programmers trying to fit these extra tasks into an already busy schedule

Concept art for *B'Angus*

and artists waiting around for a very long time before they could wrap up their levels. One perfect example of this was the game's enemy AI. Even though the artists could place enemy actors anywhere in their levels, they always needed a programmer to code at least a place holder AI routine so that the artists could preview their work. As a solution, we built a test level that we could use as an object viewer. Artists could drop their models into this test level to get a preview of what they would look like in their final, real-time, PlayStation form. While the test level helped, it didn't solve completely solve our work synchronization problem. Enemy behavior still needed to be coded in for the level to be fully tested and polished.

Still, a really effective solution to many of the problems that we experienced involved not only extra organization and proper scheduling, but also better communication. If a programmer is unaware that another team member is waiting on a certain feature, then the process would end up in a gridlock situation. So, while the organization we chose worked, it was never perfect. We used Microsoft Sourcesafe to share code between the programmers, but the only trusted version of the game was always living on Gavin James's computer. Gavin personally made sure that he'd incorporated every change we made to the game by keeping track of everybody's progress and updates. For future projects, we hope to implement some kind of registry or database where all the game elements can be recorded. Our hope is that such a system would allow us to better see how changing some part affects all of the other parts.

Our programmers developed a scripting language to enable easy implementation of the characters' AI. The language is based on a byte code, which is interpreted in real-time during the game and specifies three main threads for each character. The intelligence, animation, and movement threads control the actor's behavior and also have the ability to spawn additional threads, if necessary. The scripting language also handles collision events using a table that contains all the different classes of characters in the game and the corresponding collision types allowed to occur between them. The system can thus trivially reject collision tests between objects that should never collide with each other, such as two static objects.

As Gavin continued to improve the game engine, we kept extending the scripting, adding new features and allowing access to just about any low-level function through C function calls. Though imperfect, the scripting language was fairly easy to use once all its loopholes were known, and it enabled faster prototyping than straight C. A testament to the versatility of the scripting language is the fact that even the menu system that Malachy wrote is, in fact, treated as a normal game level by the engine. Every menu item that the players see on the screen is a game actor and the background is a huge game level.

Progress

Wild 9's development was divided into two relatively distinct periods (more on the reasons for this later). The first team that worked on *Wild 9* had most of the game engine written, a few levels started, and some game play mechanics implemented, but the game itself was not what we could call playable. When the second team took over, Gavin spend most of his time cleaning up the engine, while the artists and level designers focused on finishing one level. This was the long birthing of Gulag, which took a full two months to finish; more than any other level in the game.

To the team's credit, a second level, called Bombopolis, was started during the last few weeks of Gulag. Then things started to really snowball. People were getting used to the tools and the limitations of the engine, and we spent the remainder of the project making progress at an extremely scary pace. At this point, the engine was getting pretty stable, and we were able to start adding new features. Levels started popping out of nowhere, and before we knew it, we were playing a full-blown game.

What Went Right

1. The second team

Shiny had originally wanted to release *Wild 9* last year. Back then, another team was working on the project. But following a major staff upheaval, the only people left from the original staff were Tom and Kevin; the rest of the first *Wild 9* team, in one way r another, left Shiny. I'll elaborate on this in the "What Went Wrong" section, but needless to say, inheriting a project, engine, graphical work, and early levels from another team is indeed not an easy task. A constant tension exists between trying to work with old stuff and wanting to redo it all over again.

Most credits must go to Gavin, who had to abide by technical decisions made by other programmers before him; he spent a lot of time trying to make sense of the game engine's features and inner workings. Word is that even though the game is now finished, he's still somewhere back there trying to make sense of part of it.

In truth, he spent a lot of time trying to anticipate what people would need in order to be productive and gave, where possible, top priority to other people's needs to avoid gridlock and keep things moving. He also helped a lot by sorting out small and easily definable tasks and handing them to people who were learning parts of the system, thus making their learning curve easier.

If the first team's working atmosphere was not perfect, that of the second team was a completely different story. People were gradually added to the team as development progressed, and the last few months were spent under incredible pressure and extremely tight deadlines. And yet through all this, we get along well with one another, and no homicides took place. Joking aside, the chemistry among the second

team was incredible, and I think I can safely say that everyone had a great time working on this title. A great atmosphere meant suggestions on each other's work were well-taken, spirits were up, and productivity was positively influenced.

2. The scripting language

As I mentioned earlier, we used the scripting language for just about everything in the game. In fact, the language turned out to be rather easy to learn, and it was very powerful once some key features were added to it. It was a great way to prototype certain actors very quickly and work with the animator and the level designer to polish up behaviors and attack patterns. The interpretation of the byte code wasn't the fastest thing around, but it turned out to be a positive trade-off for flexibility and ease of use. Also, most of the low-level functions were performed by straight C code that was triggered by scripting commands from within the actors' AI code.

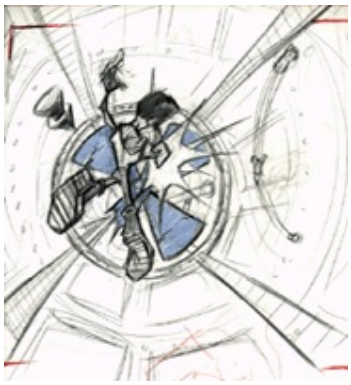
***Wild 9'* s scripting language was effective for prototyping actors and polishing behaviors and attack patterns**

3. Stuart Roch stepping up as a producer

During a rough patch in the second team's initiation, when we thought things couldn't get much worse, *Wild 9* lost Scott Herrington (it's producer), who had to replace the departing Simon Cox as Shiny's PR manager. Stuart Roch had been hired initially as an assistant designer, but he quickly stepped up to full-blown producer, even though he only officially earned the title later on in the project. Stuart's attention truly helped keep the project on track, as he did an awesome job at keeping everything together, organizing schedules, and making sure that we as the team had everything we needed. It's no surprise that now, after *Wild 9*, Stuart will go on to produce other Shiny titles.

4. The addition of Big Grub

Toward the end of the project, it became clear that we weren't going to finish all the ideas that we had had for this game if we didn't get serious help in the form of extra manpower. We needed art for the menu system, a closing movie for the end of the game, and a dozen programmers to implement the behavior of all the actors that the level designers had in mind for their levels. We didn't get a dozen programmers, but we did decide to flout Brook's Law and add more people to the project. Learning from past mistakes, we looked for people who were not only talented, but work well with the rest of the team. Nobody fit this bill better than the Big Grub guys.



Big Grub, a small company from Irvine, CA, was contracted to help develop *Wild 9*'s falling levels from initial concept (left) to completion (right).

Big Grub is a small company from Irvine, Calif., whose talents we contracted to help us out with all the content ideas we had for the game. John Alvarado, Ron Nakada, and Mike Winfield helped program the AI for many of the game's characters, Brandon Humphreys worked on the interface art and the end movie, while Neil Hong designed the second falling level and helped with the end movie. We couldn't have finished the game in time without them, but we also took a big risk bringing on new people toward the end of the project.

5. Keeping everybody in close quarters

Shiny is housed in a three-floor building. The first floor is mainly administrative, the third floor houses the *Messiah* and *Sacrifice* teams, which leaves the second floor for *Wild 9* and *Stunt RC Copter*. Shiny has no private offices, so to speak; both development floors are organized in an open space layout, with separating half-walls in some cases. *Stunt RC Copter* is only a three-person team, so the rest of the space was occupied by the *Wild 9* team. While we weren't cramped, it's safe to say that some people got to know their neighboring coworkers rather

well.

This proximity became invaluable because it enhanced communications among us ten-fold. One early problem with the project involved assigning tasks to the individual team members and keeping abreast of who was up to what. Weekly meetings going over everybody's schedules and a lot of direct and constant communication among us (most of the time without even leaving our desks) directly addressed this problem.

What Went Wrong

1. The first team

I once heard David Perry say that the first *Wild 9* team was a great collection of talented individuals, while the second was a talented team. This statement alone sums up the main problem encountered by the first team. As we found out later in the project, communication and good atmosphere were two factors that made this project move forward; it's now obvious that the best thing to do was to start over rather than try to salvage a situation that was going nowhere.

2. A lack of existing documentation

I've already sung the praises of our scripting language, but not everything was totally perfect in this picture. Although it ended up being very useful, it became clear early on that the language and its op-codes had massive loopholes and "gotchas" waiting to jump on us. To help other programmers with their first steps, Gregg started maintaining a document as he was learning the AI system. This document described all the major problems that he'd encountered and ways around them that he'd discovered. When Gregg left the project , Gavin and the rest of the programmers took over the responsibility of updating this document as new problems appeared or new solutions were found.

This process continued until the very end of the project. The document was invaluable in getting somebody up to speed on the system, such as when the new programmers from Big Grub came to the rescue. Overall, the language's design was, in some places, questionable; for instance, the actor registers were called params and the virtual processor's variables were called regs, which was greatly confusing.

3. The physics engine

The physics engine turned out to be one of Gavin's biggest nightmares during the *Wild 9*'s development. He inherited this code from the previous programmers, and he spent more time trying to unravel, optimize, and just get that part of the code functioning than on any other identifiable part of the game. His advice to engine inheritors: unless the physics are perfect and written exactly as you want them, grow your own.

The code for *Wild 9'* s physics engine was inherited from previous programmers and was difficult to integrate.

4. Deal me in

Our tools programmer and resident Irish person Malachy once decided to bring us a little multiplayer game he was working on in his spare time. The game was called Deal Me In, and he wanted us to help him beta test it. Deal Me In is best described as a very addictive cross between Scrabble and Poker. Actually, "addictive" doesn't even begin to describe the darn thing.

Everyone at Shiny plays games. So when it comes to testing a very addictive game, we are very thorough. The game became an obsession, to the point where it was affecting our productivity. It turns out, after extensive research, that there is no such thing as a "quick game of Deal Me In."

5. Things that did not make in

Tom and Kevin are two very creative fellows. The material they came up with is enough to fill up three or four games like this one. In addition, the whole team took this game to heart and ended up coming up with suggestions of their own. We were thus faced with a situation toward the end of the project: we could not fit everything into the game.

Wex's weapon, The Rig, is normally blue. But it possesses a red mode, which is greatly under-used in the game. The red mode was initially meant to be a more powerful blue beam, but this created game play issues, as a player will sometimes want to transport enemies in the beam from one point of the level to another. The red beam would vaporize just about everything on contact, which was obviously a problem. This is one of the only design issues that was never truly fixed and the red beam, even though it made it into the game, isn't used to its full potential.

Many other great actor ideas never made it into *Wild 9*: for instance, the 3D cow that was intended as a reference to *Earthworm Jim*, or the tank that Lloyd Murphy modeled that was supposed to populate the Wreckage level. In fact, Lloyd's tank became sort of a myth, which we would bring up at various meetings: "So we don't have time to fit even a tiny little tank then? Are you sure? What if I pretend it's a background?" The need to stop development at some point made it impossible to use it all the ideas and content that we had, although I hear from reliable sources that the cow did eventually make it in there somewhere.

We Did Our Best

It's an amazing feeling after working so hard on a game to sit back and start playing it. In our case, the general feeling was that *Wild 9* turned out amazingly well, especially in light of its rocky development history. One of the best feelings was to see the press reaction at E3 this year, and to hear the comments that *Wild 9* had come a long way. If you ever saw anything from *Wild 9* a year ago, go and check out the final version. We did our best to make sure you will have a pleasant surprise.

Didier Malenfant is a programmer at Shiny Entertainment. Even being French he is only mildly rude and enjoys snowboarding, driving his yellow car, researching in all aspects of 3D programming and running Linux boxes. He can be reached at dids@shiny.com.

Wild 9

Shiny Entertainment
Laguna Beach, Calif.
(949) 494-0772

http://www.shiny.com or http://www.wild9.com

Team Size: Thirteen full-time developers at Shiny. Five contractors from Big Grub.

Release date: October 1998.

Intended platform: Sony PlayStation

Hardware used: Loads of very powerful PCs and PSY-Q devkits.

Software used: 3D Studio Max, Photoshop, Painter, PSY-Q development system, Slick Edit, Codewright, Visual C++, Sourcesafe, and custom tools.

The *Wild 9* team

Standing up from left to right: Erik Drageset, Jean-Michel Ringuet, Scott Herrington, Malachy Duffin, Tom Tanaka, Gavin James.

Sitting down from left to right: Lori Perkins, Rich Neves, Kevin Munroe. Not pictured: Klaus Lyngeled, Stuart Roch, Lloyd Murphy, Didier Malenfant.

And the Big Grub guys: John Alvarado, Brandon Humphreys, Mike Winfield, Ron Nakada, Neil Hong.

Return to the full version of this article