GAMASUTRA
The Art & Business of Making Games

PRINT

Postmortem: Mommy's Best Games' Weapon of Choice

By Nathan Fouts

[Ex-Insomniac programmer Nathan Fouts created one of the standout Xbox Live Community Games for Xbox 360 in 2D side-scrolling shooter Weapon Of Choice, and explains just how in this in-depth Gamasutra postmortem.]

Working on games such as *Resistance: Fall of Man* and *Postal 2* was a dream come true. While I really enjoyed contributing to stunningly-complex 3D games, my secret passion was to create a smoking, 2D, side-scrolling action game that looked like it crawled off that high school, stoner kid's notebook, and then ate Rainbow Brite.

For me, directly mapping 2D stick controls to a 2D action game is like pizza and beer. And maybe some warmed pie on the side. And some ice cream on the pie.

Mmmm... anyway, I drugged my wife and convinced her that I should quit my great job at Insomniac Games, and use *all* of our savings to make my own game. (Okay, just kidding -- no one *actually* took drugs despite what *Weapon of Choice's* art direction may suggest.)

With most of my teenage years spent fighting Red Falcon and the Bydo Empire, *Weapon of Choice's* designs and drawings flowed freely. The game revealed itself to me over a period of months, and fortuitously, XNA became ready for primetime as well.

Figure 1. Final title and original pencil art. All the art in the game started as pencil and was then scanned and colored digitally.

My friend, AJ Johnson, wrote the dialogue and the story, and Hamdija Ajanovic composed the game's rocking, custom soundtrack. A programming friend helped with a few enemy prototypes, and another old colleague designed one of the levels.

Those poor saps worked remotely and agreed to get paid on the back end, based on sales profit. I also remotely contracted two texture artists for occasional environment texture work; they were paid with actual money.

My wife acted as the producer and business manager. Friends and family were the QA department, playtesting the game at milestones. That left everything else for me, which included the original concept, design, programming, art, animation, sound effects... you know, the game part.

While initially I wanted to release the game on Xbox Live Arcade, *Weapon of Choice* swaggered onto the Xbox 360's Community Games on November 19th and seems to be well received. The game is an "approachable hardcore game" meant for older gamers who don't always

have time on their hands for retail games.

Though I've worked in the professional game industry for over a decade, there were many sadly entertaining things I learned along the way. But let's leave the embarrassment for last.

What Went Right

1. Like PB&J That's Heaven-Sent, XNA & C# Taste Great Together.

I'm a gameplay programmer at heart. I'm most comfortable tweaking aiming algorithms, creating swarming behaviors, and calculating urine streams. Needing to know what port to read or when to flush memory is not my forte. Having XNA handle most of the low-level issues for *Weapon of Choice* was fantastic.

XNA handled most everything that's ever scared me about game development. With XACT, the suite's audio tool, I was able to incorporate initial sound effects and music within an afternoon and had minimal sound issues during development.

For weeks, I avoided creating a data file to save player progression. Accessing the 360's hard drive, displaying the Xbox Guide, knowing if the memory card is available -- this was pretty intimidating for a guy who specializes in how to best cleave an NPC's forearm.

Yet, I remember spending a morning reading XNA's help file and writing my save/load interface. I couldn't believe that within hours, those issues were resolved.

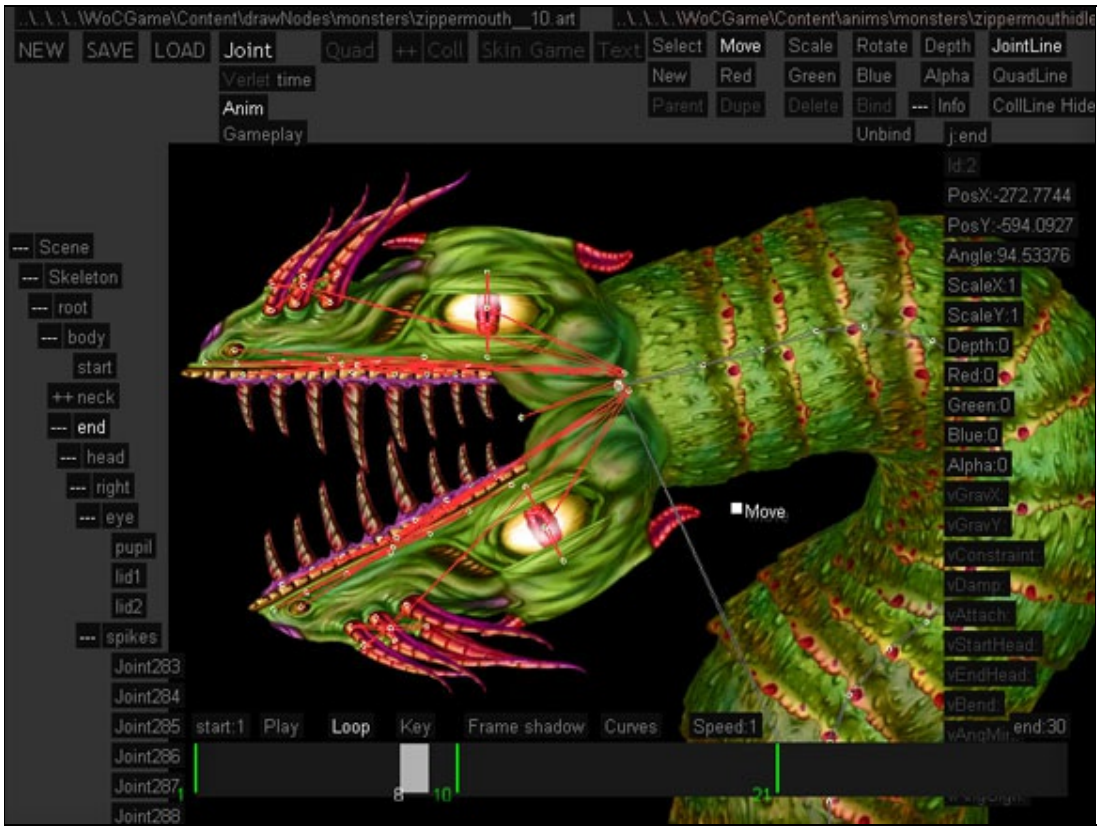

Figure 2. Zipper Lips in the Object Editor. Without the development speed of C# and XNA, I couldn't have written the editing software and game in a single year.

In addition to learning XNA, I was also using C# for the first time, which may seem like a recipe for confusion. C# was so streamlined and easy to use, I feel confident that using it with XNA made the length of the project significantly shorter than if I had used C/C++.

2. Barrel-aged Design and Gourmet Prototypes.

I designed and recorded ideas for *Weapon of Choice* for over a year before I started writing any code. Because of this early work, there was very little time wasted when I started programming gameplay. The key was to program each new element in a basic manner, try it out, and then continue forward if the gameplay looked promising.

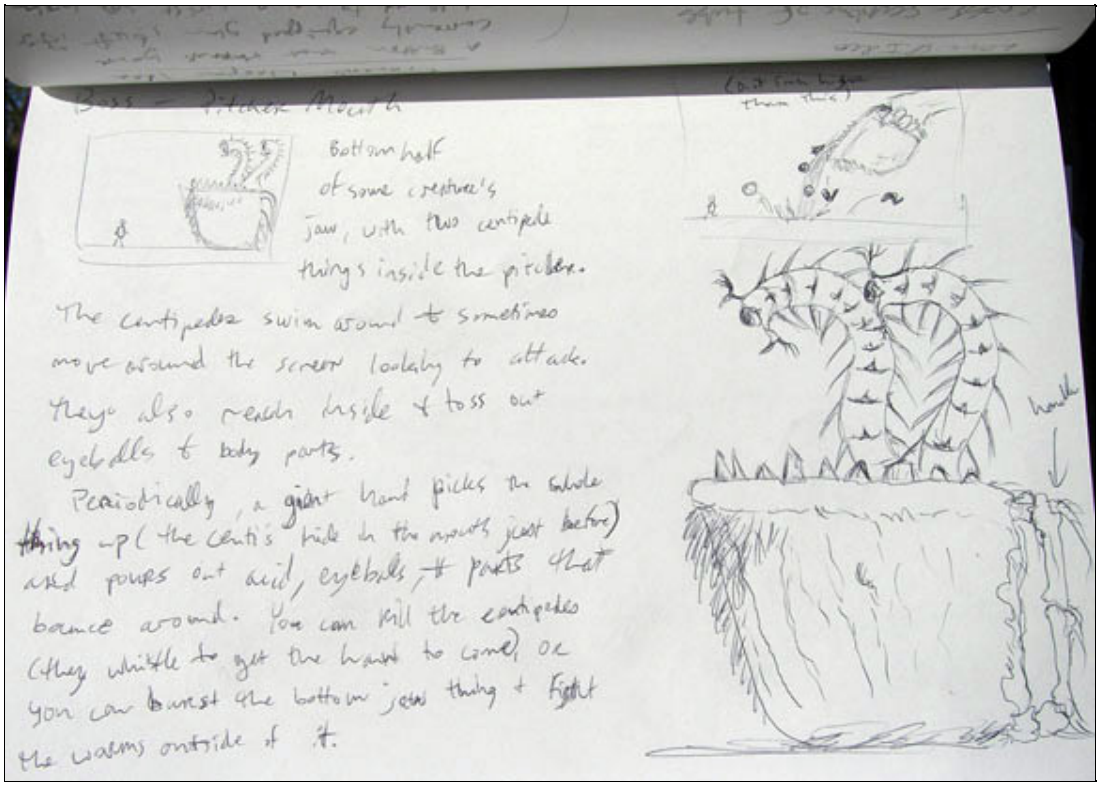

Figure 3. Pitcher Mouth boss sketch with notes.

The early prototype of the game had nearly every important game element incorporated except in-game decision making and the "Conflict Zone" map. I felt that Death-Brushing (the *Weapon of Choice* player ability to dramatically escape death) was crucial to get working early; if that couldn't be made fun, then I would have to rethink the whole game.

Prioritizing gameplay prototypes ensured it was okay to proceed, rather than potentially wasting time down the road.

Figure 4. Prototype effect for Jet Engine gun in test level.

I like blocking in all animations and character rigging with placeholder art. Using placeholder animations in the prototype brings it closer to the real thing, and allows for final animations to be dropped into place with theoretically little fuss.

Although I think this prototyping method helps, fairly often, I had new ideas which required re-rigging and new animation work. Combating feature creep required constant discipline.

3. Measure twice, cut twice.

I slowly build up my game ideas as such that when it's time to make a game, I have an attractive list from which to cherry-pick designs. For *Weapon of Choice,* I had initially designed over 40 levels, 50 operatives and weapons, and probably a hundred enemies.

Honestly, I wish God would just make *that* game for me, right now. I want to play it! Unfortunately, I was the one making the game, and the original design would have taken years.

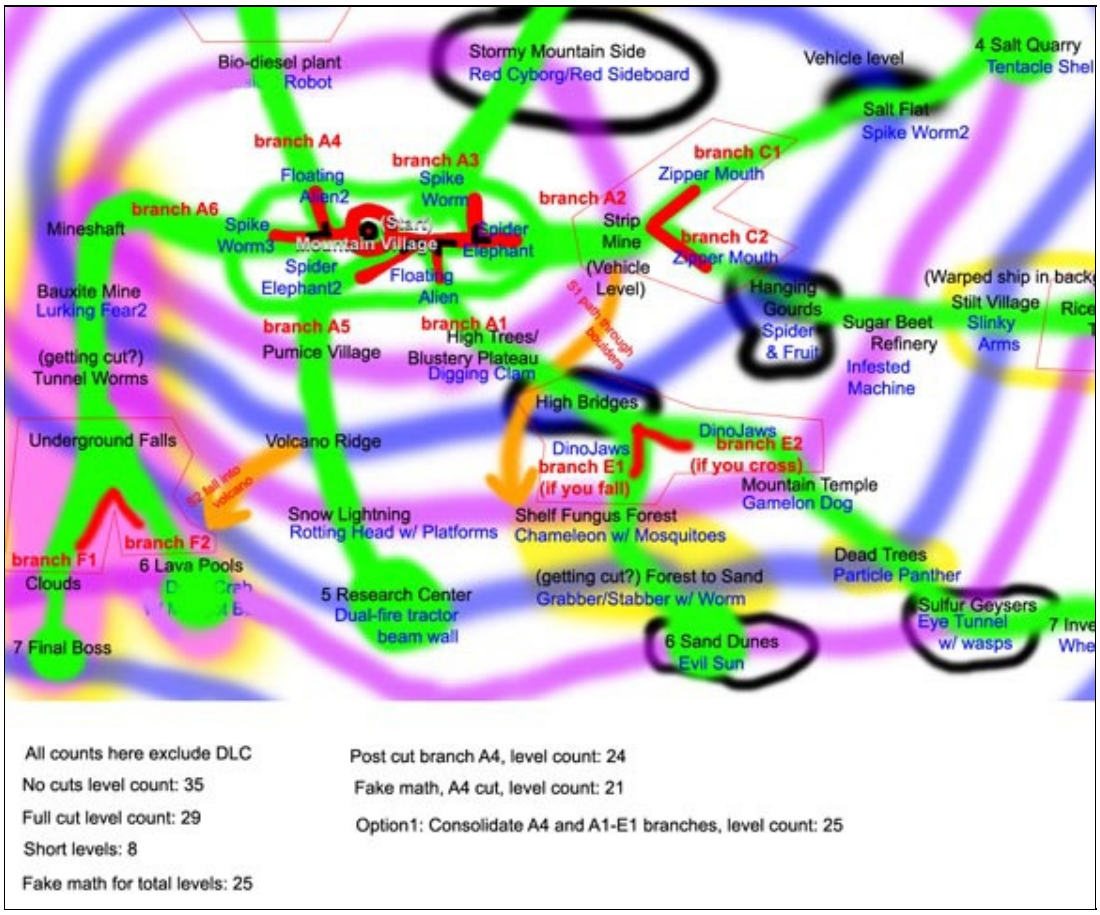

Figure 5. Early level layout before cuts were made.

Weapon of Choice's multiple story branches include unique levels, each concluding with a satisfying ending. The initial 40+ levels had 11 different endings. While not applicable to many games, the branching nature of *Weapon of Choice*'s game design made it easier to cut back.

The game was reduced in size three different times, with entire branches and endings culled. Even then, I still honestly thought I would be able to finish the game with over 20 levels. (Cuckoo!)

Figure 6. Before the action, the Conflict Zone map shows vignettes of each level.

After even more trimming, the final version has seven levels, seven Operatives, and four unique endings. The upside is that I was simply able to remove entire story branches as a method of cutting content.

This worked much better than the typical scenario of tearing apart a monolithic story, dropping levels, and attempting to mend the story.

---

4. Roll your own.

At the beginning of development, I spent a few weeks researching various modeling software, before realizing they either did not meet my needs or the cash in my pocket.

To create *Weapon of Choice*, I wrote Mommy's Best Level Editor and Object Editor (for modeling and animation) in about four months, which provided powerful control and fast gameplay creation.

I don't commonly see games with objects that animate color and translucency, and I made sure to incorporate this into the Object Editor from the start. The first creature encountered when playing, the Air Bladder, swells up before shooting, which changes its namesake, vein-covered sack translucent and slightly yellow.

This animation control was used many other places, and starting immediately with such a strong case provided a good example to follow.

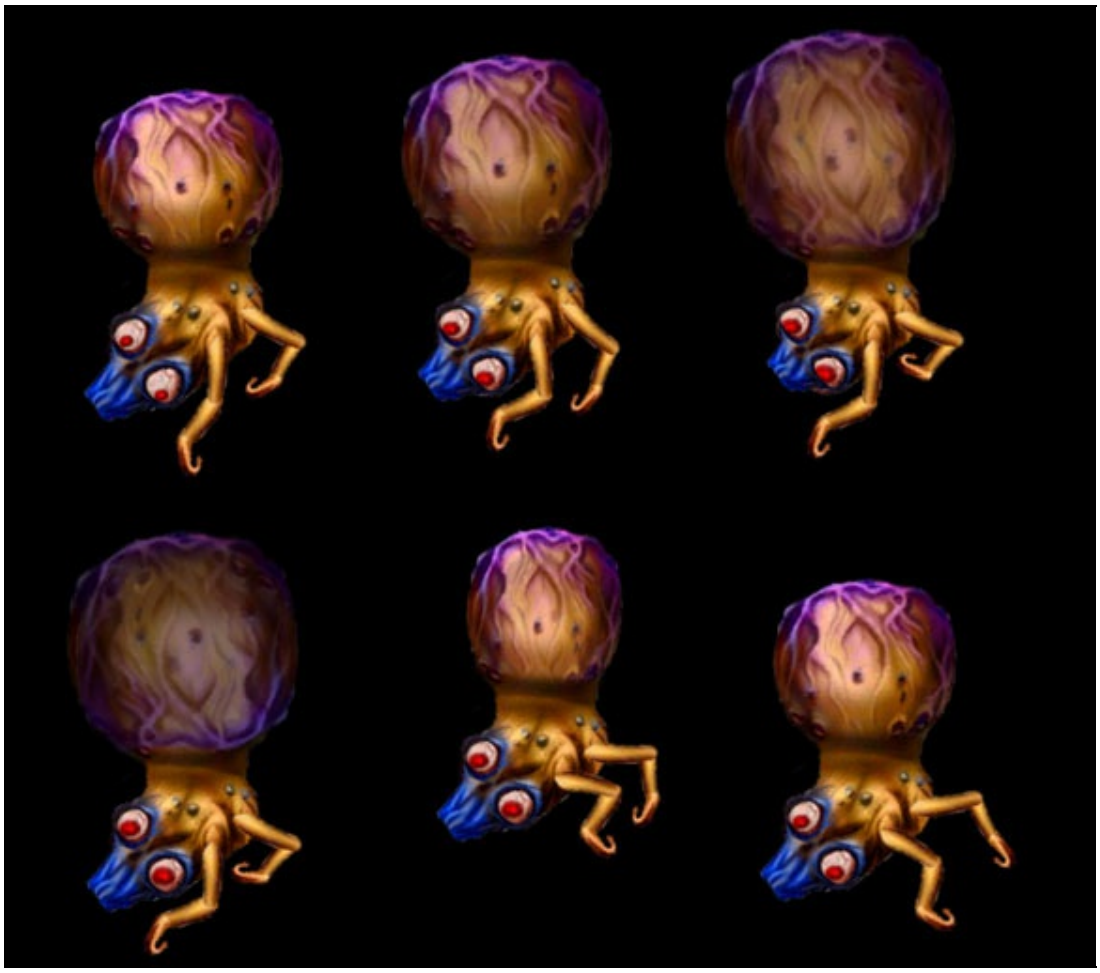

Figure 7. The Air Bladder 'shoot' animation shows the translucency in effect.

For me, the best use of technology is to incorporate the most unique aspects of the engine into the gameplay. Mommy's Best Object Editor allows for complex skeletal systems to be created and then rendered using sprites attached to the joints.

Limbs can use a verlet chain to control motion with a hand or foot driven by animation. Integrating the editor and game code allowed for subtle polish, such as accessing the joint chain for the legs of the operatives or aliens, and having their feet properly animate on uneven terrain.

This ability fed back into the level design; I felt challenged to create levels that looked opposite to often flat, tile-based design.

Figure 8. Screenshot of Xerxes fighting a Wrap Mouth. Xerxes' legs are reoriented in code after animation initially positions them.

5. Sometimes Quantity over Quality Works Too.

From the start, I had to decide if I should animate characters with a sprite for each frame of motion and only have a few, gorgeously animating objects, or should I have a ton of animation using a key-framed approach more akin to 3D games, which at times can bend limbs in a weird fashion.

While I love the look of the *Metal Slug* series, 16-bit *Aladdin, Flashback*, and other sprite-animated games, I decided a sprite system rigged to a key-framed skeleton would get me the most animation for my time.

This resulted in dozens of different trees, grasses, vines, leafy plants, spikes, tentacles, eyes, and appendages, all flowing, flapping, dangling, and swinging in perverse yet attractive fashion.

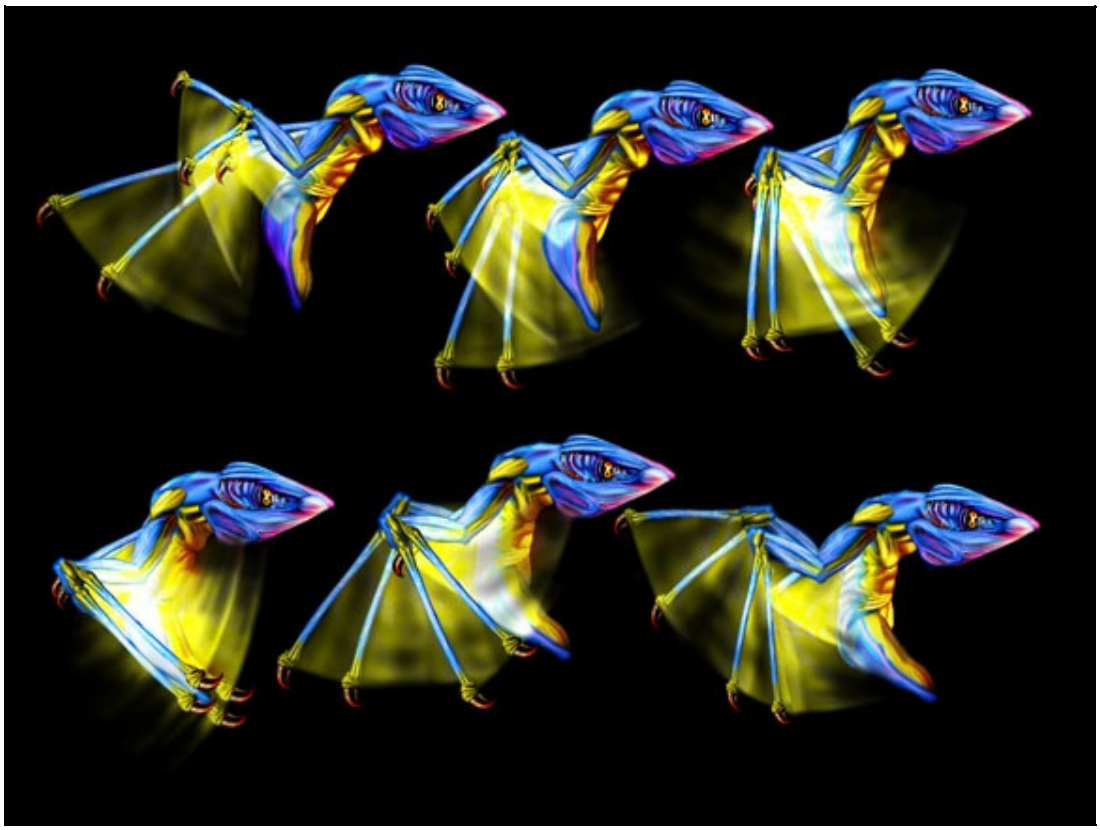

Figure 9. Wrap Mouth animation frames 18 through 24. Note the skewing of the limbs in some poses. While still noticeable, it's difficult to see during gameplay.

Here is a break-down of in-game art objects and their animations:

	Environment *(rocks, trees, clouds, etc)*	**Enemies** *(monsters, living bullets, etc)*
Art objects	250	125

Animations	85	600

Environment animations were shared across many different objects, but for the monsters, which are the most important, I created many unique animations. I believe it would have taken me years to fully animate that many creatures in the classic sprite fashion.

For MBG's next project, I'd like to improve the ability to change sprites on a skeleton in order to combine animation methods, resulting in the best of both worlds.

What Went Wrong

1. Garbage Strike.

In retrospect, I feel I waited too long to get *Weapon of Choice* to run on the Xbox 360 (mid-June for a November release). I went into it simply not knowing exactly what I was getting into and overestimated how much the Xbox would enhance the game's performance.

With XNA on the PC, a generational garbage collection system is used. This basically means it's smart about how and when to collect garbage.

As it turns out, the system on the Xbox 360 is non-generational, which basically means any little thing that needs garbage collection will slow down the game. When I first got *Weapon of Choice* running on the Xbox 360, the game's smooth PC framerate speed devolved into a slideshow.

It took months to deal with all the garbage collection issues and get the framerate up to a playable speed. I didn't do my homework on the target platform and it hurt badly, even though documentation existed.

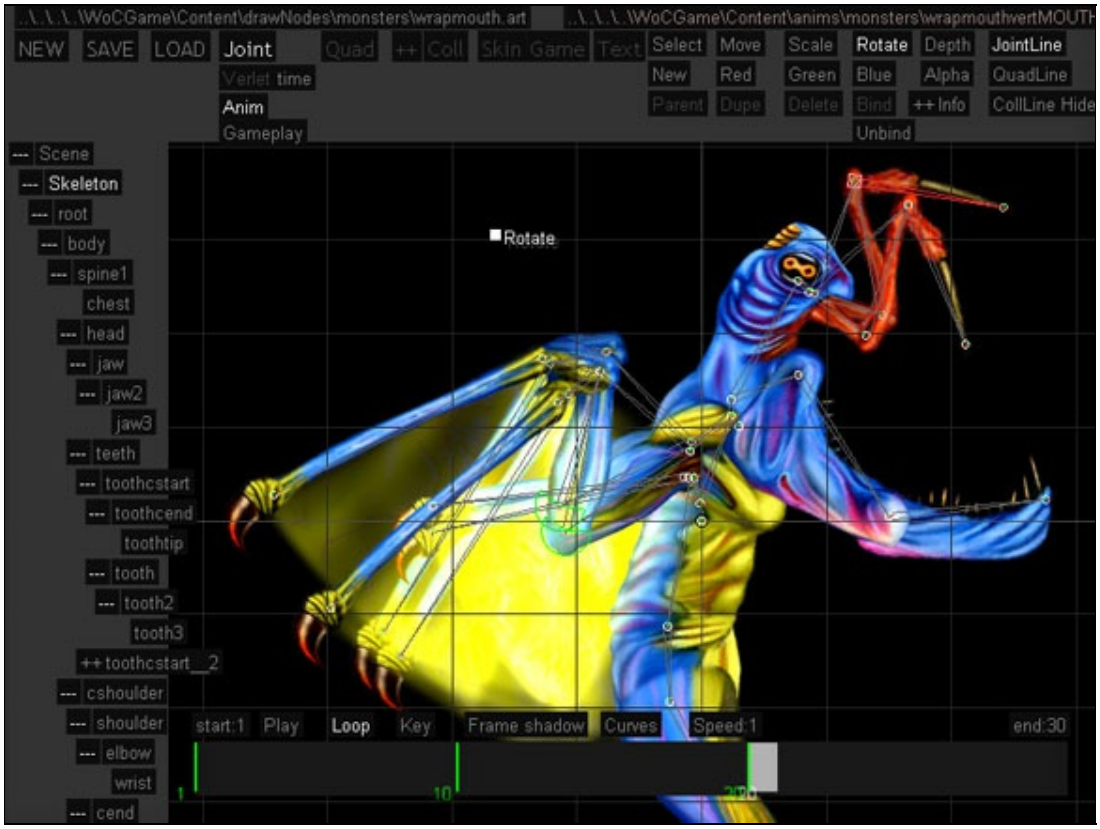

Figure 10. Wrap Mouth in Object Editor in "Stage 2" pose. Initially, loading animations like this took over two minutes of wasted time per level due to garbage collection.

If I were to do it all over again, I would have the game's prototype running on the 360 and use the XNA Framework Remote Performance Monitor from the start, to ensure garbage collection was not an issue.

2. My Pipeline Doesn't Look Like Your Pipeline.

As stated before, I developed custom animation and modeling software for creating the game art. I also wrote my own file managing system, even though the XNA Content Pipeline was created to manage assets. (Maybe I have some control issues?)

This worked fine on the PC, as I was able to subvert the Content Pipeline and load everything with my own system. However, my custom asset pipeline was rendered useless when I finally converted the game to the Xbox 360, as XNA only allows assets loaded through the

Content Pipeline.

I ended up keeping my custom loading code and simply wrapped it with special code to pass it through the Content Pipeline.

As it turned out, this special code incurred many garbage collection hits since, in .NET, strings are immutable, and with each new string, more memory was allocated. I wrote a special character parsing function to load the data (not using strings and incurring no new memory allocations).

This resulted in other issues that required a patch after launch; my parser failed to handle the idiosyncrasies of some languages which swap the comma for the decimal in floating point numbers.

3. On-the-Job Training.

Before this project, there were many things, such as rigging models and making sound effects, which I'd never tried before. At the top of that list was programming a multi-threaded game update loop.

I changed the game loop to utilize threading too late in the development, which created hundreds of crash bugs, stemming from code in which two threads would access the same static data.

I mainly operated from a "Use Threading Because It Will Make Things Faster" mindset, rather than confidently designing code to enhance the runtime performance. During profiling, I could toggle between single and multiple processor use on the Xbox.

This showed that threading provided around a 15% speed-up. This was good, but it definitely never reached the magical 60fps level.

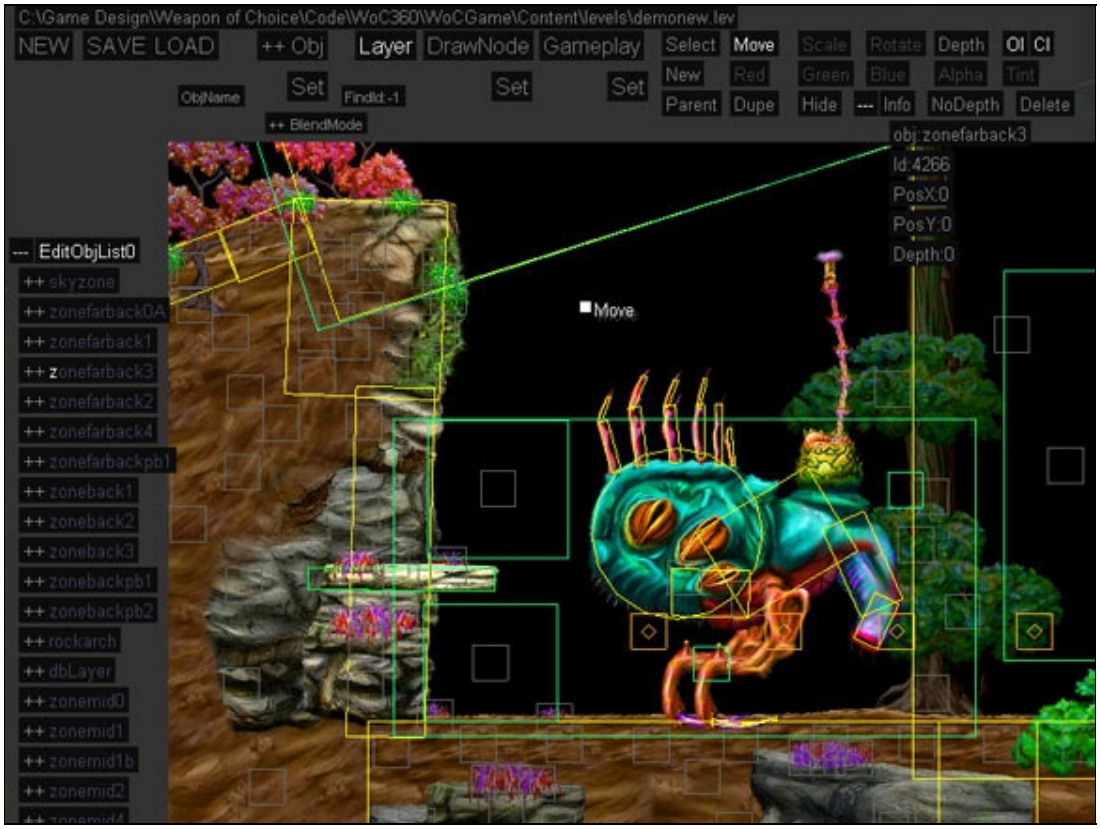

Figure 11. Level Editor view of a first level boss showing only the mid-ground layers. These layers contained the fighting and generated the most processor work.

The Xbox 360 has six hardware threads. In XNA, thread indices 0 and 2 are reserved leaving developers with four useable threads. I used thread indices 1, 3 and 4 for gameplay updates. Because I put loading on a separate thread so late in the project, I simply assigned loading to thread index 5, rather than try to dynamically switch thread functionality.

In my system, as the gameplay threads are initialized, level objects are assigned to different thread indices. For example, objects 1-1,000 go to thread index 1, objects 1,001-2,000 go to thread index 3, and the rest go to thread index 4.

The background-to-foreground, layered nature of the level design, however, created a situation in which one thread (index 4) would be given the layers with all the action and collision detection requests. I'm pretty sure clumping work onto one thread can retard the benefits of multi-threading.

I recognize that putting collision on one thread, general updates on another, and animation on a third makes sense, but after several attempts, I never got that method to work properly. My "Gosh, I Hope This Helps Some" approach to multi-threading probably explains the meager speed increases.

4. "Launch Title" Is a Four Letter Word.

Shipping a launch title is not something I enjoy. Technical and policy issues change constantly if they're yet known at all. Although the benefits of being one of the first games on a new, uncrowded platform outweighs these issues, there can still be plenty of problems.

Making a Community Game is different because generally, you have no special publisher connections and only get information as it becomes publicly available. For *Weapon of Choice,* we were not involved in the New Xbox Experience beta and therefore couldn't test our game with the new dashboard.

It was very frustrating designing features like trial mode and hoping they would work; we were only able to test trial mode definitively on the day the game went on sale.

All Community Games are required to use XNA Game Studio (GS) 3.0, which was made available shortly before the channel launch. Porting to GS 3.0 was quick and painless; however, *WoC* level load times increased by about 40%.

It's hard to say exactly what caused this, but it was probably related to the original, string-heavy method I was using to load data (*See Wrong 2.*).

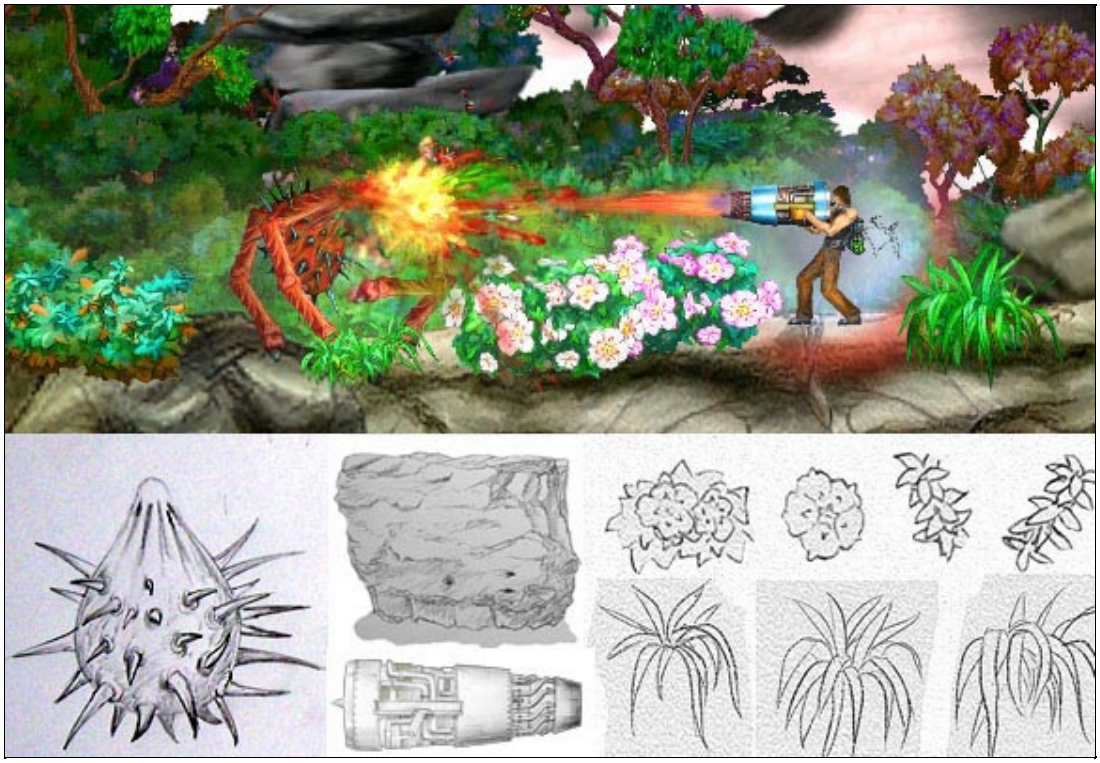

Figure 12. Screenshot of "MountainVillage" level with a selection of the drawings used to create the final art.

All Community Games developers had access to the betaof GS 3.0 for months. I didn't feel like I had the time or manpower to maintain GS 2.0 for development and GS 3.0 beta in a sandbox for testing, and thus only ported to the final GS 3.0 near our launch. Regardless, I had the ability to discover the loading problems and fix them earlier.

5. Turns out They Pay Managers for a Reason.

Though I did most of the work on the game I had some help in a few areas. Determining self-contained areas in which others could contribute was crucial. Story writing and music worked out perfectly. Programming and level design were a mixed bag.

Preparing prototype animations, designs, and code reference ate into some of the potential time savings of having someone else do the work. Making sure everyone was synched to the current build, and updating everyone to new Game Studio versions were hassles.

Many phone calls and emails were required to coordinate all the work, keep everyone up to date, motivated, and moving towards milestones. I learned the hard way that it's not as simple as "If something takes *me* three days, someone else doing it will save me*three whole days*!"

Mommy's Best Game

Well, Mommy's Best has shipped our first game, and we feel *Weapon of Choice* turned out pretty darn swell. Don't worry; I won't sprain anything patting myself on the back.

Microsoft made XNA to get more people into game development. I think that's a great thing. However, I saw it more as an opportunity to take my years in game development and turn it into a viable company.

I took the XNA/Community Games opportunity very seriously and dedicated more than a year, full-time, to making *Weapon of Choice*.

There are a lot of amazing things that had to happen to make *Weapon of Choice* possible. Microsoft created a free, full-featured game-making suite for anyone to use, then allowed anyone to run, publish and sell their home-brew games on the Xbox 360.

Additionally, my family and friends sacrificed a lot, and often, to help us see our game to completion. In the end, we're left with interactive art that makes me happy as a gamer every time I see or play it.

With everything we learned and created while making *Weapon of Choice*, I'm busting with energy and anticipation as we start Mommy's Second Best Game. Hmm... "second best"... that was supposed to come out better!

[For more on Fouts' history in the game business and the creation of Weapon Of Choice, interested readers can check out his recent interview with Gamasutra.]

Game Data

Project: *Weapon of Choice*

Developer and Publisher: Mommy's Best Games

Platform: Xbox 360 Community Games

Release Date: November 19, 2008

Full time developers: 1

Contractors: 6

Development time: 1 year (4 months for pre-production, 8 months for production)

Hardware: AMD 2.2GHz, 1GB RAM; retail Xbox 360; stylish, 19" flat screen monitor; ancient, Buick-sized SDTV

Software: XNA, Visual Studio Express Edition, Photoshop CS2, GoldWave, Mommy's Best Level Editor, and Mommy's Best Object Editor

Resources: www.soundsnap.com, creators.xna.com

Return to the full version of this article
Copyright © 2016 UBM Tech, All rights reserved