Building a Massively Multiplayer Game for the Million: Disney's Toontown Online

MARK R. MINE, JOE SHOCHET, ROGER HUGHSTON
Walt Disney Internet Group, VR Studio

This paper presents an overview of the lessons learned building Disney's Toontown Online (www.toontown.com), a 3D massively multiplayer online game (MMP) for children ages seven and older. The paper is divided into three main parts. The first presents design highlights of Toontown Online and focuses on the challenge of building an MMP for kids. In particular, we discuss ways of incorporating kid-friendly socialization into an MMP. The second part of the paper presents an overview of Panda-3D, the VR Studio's open source 3D graphics engine. We focus on the aspects of Panda-3D that helped to facilitate the development of Toontown. In particular, Panda's expressive, platform-agnostic scene graph architecture, and flexible scripting language tools for distributed storytelling. We finish with an overview of Toontown's server architecture and present our *nothing-but-net* strategy for downloading a full-featured 3D online game.

Categories and Subject Descriptors: I.3.8 [**Computer Graphics**]: Applications

General Terms: Design, Experimentation, Human Factors

Additional Key Words and Phrases: 3D, computer graphics, online games, multiplayer games, Internet

1. INTRODUCTION

Disney's Toontown Online is one of the first 3D massively multiplayer online games designed for children ages seven and older. Filled with safe, friendly, and colorful environments, Toontown has a clean interface and simplified game mechanics that are ideal for younger children. Toontown, however, is not just for kids. According to survey feedback, a large percentage of Toontown subscribers are adults who enjoy Toontown's many humorous and nostalgic elements and addictive game play. Toontown Online was designed and created by Disney's VR Studio, formerly part of Walt Disney Imagineering Research and Development, now part of the Walt Disney Internet Group.

Toontown is built on top of Panda-3D, the VR Studio's open source 3D engine and storytelling system that consists of

- a portable and efficient 3D graphics engine written in C++;
- an interpreted, Python-based scripting layer, with numerous building blocks to assist in interactive storytelling; and
- a powerful suite of world-building tools called DIRECT (Disney's Interactive Real-time Environment Construction Tools).

The goal of this paper is to present some of the systems and lessons learned while

Author's address: Walt Disney Internet Group, VR Studio, Burbank, CA 91521-7716

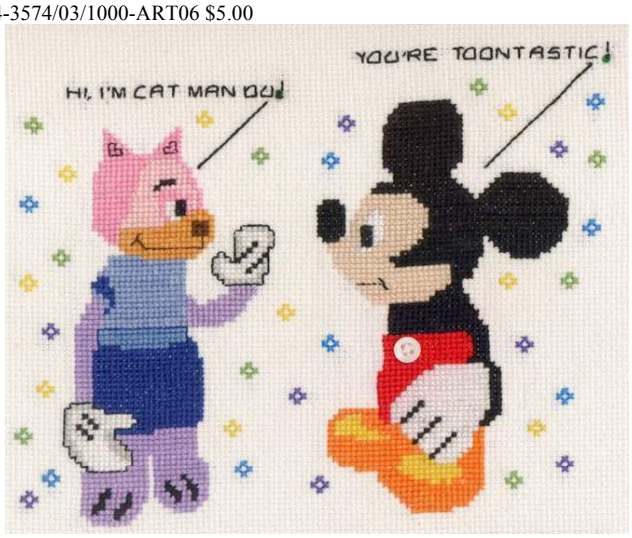

Fig. 1. Toontown cross-stitch fan art: Crossing the mass market barrier?

building a massively multiplayer online game (MMP) for the mass market (Figure 1). The paper is divided into three main sections. The first presents design highlights of Toontown Online and focuses on the challenge of building an MMP for kids. In particular, we discuss ways of incorporating kid- friendly socialization into an MMP, including

- *SpeedChat*: A hierarchical, menu-based chat system for game-related interaction with all players that prevents players from divulging personal information; and
- *Secret Friends*: A safe, text-based open chat system for interacting with real-world acquaintances.

The second part of the paper presents Panda-3D, the technological foundation on which Toontown is built. Here we focus on selected features of Panda-3D that facilitated the development of Toontown Online and enabled us to meet our design goal of creating a mass market MMP, including:

- a powerful and expressive *platform-agnostic* scene graph architecture; and
- a *distributed storytelling* system that facilitates the specification of complex animated sequences and effects.

Finally, we conclude with a description of Toontown's download and game server architecture, including

- a *background download system* combined with efficient encoding of characters and environments that enables near immediate entry into a full-featured, yet completely downloadable, 3D game; and

- a *transparent server architecture* and distributed object model that simplifies the development of a distributed persistent online world and enables players to join their friends at any time, regardless of which server they inhabit.

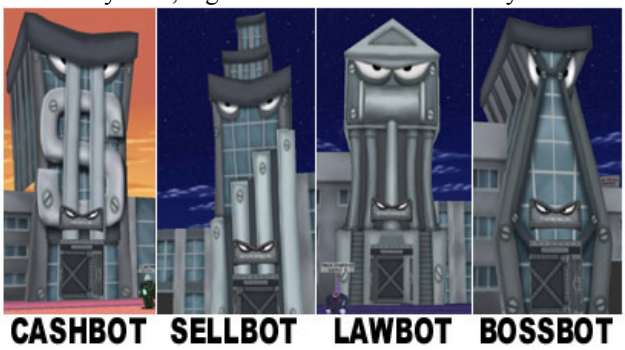

Fig. 2. Cog buildings.

2. TOONTOWN ONLINE DESIGN HIGHLIGHTS

Intrigued by the possibilities of online gaming, the VR Studio started the multiplayer game project in 1998. The goal was to explore the viability of creating an online game for the mass market. The result is Toontown Online, a massively multiplayer online PC game explicitly designed for children ages seven and older. MMPs are persistent online worlds inhabited simultaneously by thousands of players.

To appeal to a broader market, protect the Disney brand name, and be in line with COPPA policies, we tried to make Toontown a safe, social, friendly game that was simple to learn yet challenging to master [Bruckman 2002]. The following paragraphs highlight some of the choices we made and the lessons we learned; see also Goslin [2003] for additional discussion of designing multiplayer worlds for the mass market.

2.1 Using Positive Stories and Themes

Unlike many of the MMPs currently on the market, Toontown is not based on sword and sorcery or science fiction themes. Though highly successful, games in these genres primarily attract the hardcore gamer market (predominantly adolescent males of all ages), and are often too intense for more casual gamers. We opted instead to make Toontown a happy place where *Toons*, cartoon characters nostalgically based upon 1940's cartoon characters, live and play. Toontown is a colorful world that consists of six whimsical neighborhoods, each themed around a classic Disney character.

To provide a level of conflict within the game, we introduced the *Cogs*, an out of control mob of business robots that is trying to turn the colorful world of Toontown into a black-and-white metropolis of skyscrapers and office buildings (Figure 2). We chose robots as the primary antagonists, since destroying imaginary robots is less violent than destroying imaginary life forms.

Instead of the standard attacks that many games feature, we opted for a theme of traditional-cartoon practical jokes, such as letting Toons throw cream pies, squirt seltzer bottles, and drop flowerpots on their enemies. The Cogs fight back with office related humor, such as red tape, bouncing checks, and squirting fountain pens. This gives us a unique conflict system that has all the strategy of a traditional role-playing game, but

with none of the dark and hyper-realistic violence that can be a turn-off for the mass market.

Fig. 3. SpeedChat.

Fig. 4. Toon name generator.

2.2 Keeping Kids Safe

Most MMPs targeted at adults provide nearly unrestricted chat between players. To provide a safe mechanism for game-related communication, we developed *SpeedChat*, a hierarchical menu-based chat system that enables players to communicate using only preselected phrases (Figure 3). The menus are context-sensitive, adapting to the player's current status and in-game objectives. Initially we feared that menu-based chat would be overly restrictive for an MMP, but feedback from the majority of users has been quite positive for several reasons. Phrases are always appropriate, sticking to the theme of the

game. Canned phrases eliminate jargon that can be overwhelming and uninviting to new players. Novice typists don't need to be concerned about grammar and spelling. The

Fig. 5. Teaming up for a minigame.

majority of phrases are friendly, encouraging cooperative behavior between players. During focus testing we saw that these factors helped to overcome shy players' inhibitions. Most importantly, SpeedChat is safe. Players cannot divulge personal information to other players using SpeedChat, and chat strings are encrypted to prevent modification by hackers.

To provide more open exchange that still protects players from unwanted communication, we include a password-protected open chat system called *Secret Friends*. To use Secret Friends, a player gives a password obtained in game to a real-world acquaintance. Once the acquaintance enters the password on his computer, the two are marked as "Secret Friends" and will be able to chat freely with each other using the keyboard. There is no way to swap passwords in game; players must use other means such as phone, email, or ICQ. This ensures that a player will not be able to chat with someone who they do not already communicate with outside of the game. We also take extra precautions by allowing passwords to be used only once, invalidating them after 48 hours if not used, and capping the number of secret friends any single player can have. Our servers automatically filter all chat for profanity and other inappropriate content. We also log open chat for use in the event of a customer complaint.

Another feature that is sometimes abused by players is character naming. During our own Beta testing, we found it was impossible to create an automatic filter that adequately prevented players from creating inappropriate names.

To discourage players from using obscene or offensive names, we provide a humorous name generator (Figure 4). Selecting from several million possible combinations, the name generator creates appropriate character names such as "Good ol' Barney Fizzlehoffer." Generated names have the added benefit of being more consistent with the humorous theme of Toontown. Players may choose to enter their own names, but these must pass through client- and server-side filters *and* be accepted by a customer

service representative before they can be used by a Toon. Cartoon Network uses a similar system for safe random name generation at www.cartoonnetwork.com.

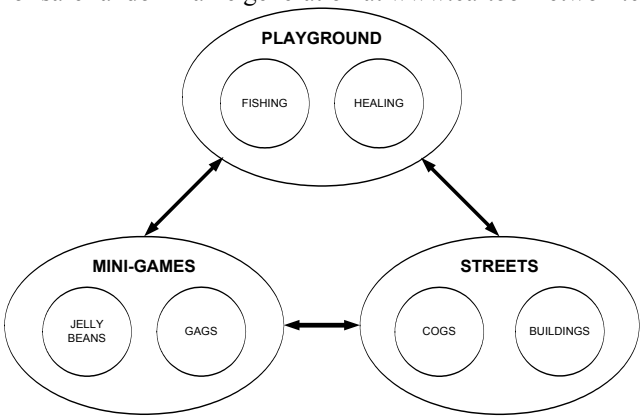

Fig. 6. Toontown trinity.

2.3 Encouraging Social Behavior

To help make Toontown a social place, we included many shared activities such as fishing, mini-games, and Cog battles that serve to break the ice and stimulate friendships. As a way to encourage this behavior, we made it very easy for Toons to team up and form groups, avoiding the more rigid grouping mechanisms found in many other games. To play a mini-game with a friend or a stranger, one simply hops on the same trolley (Figure 5). Similarly, one joins a battle just by walking up to it. No verbal communication or preplanning is required.

When designing minigames, we focused on cooperative rather than competitive play. We worked hard to put in game systems that allowed players to team up in useful ways, and we only put in a few ways that players could compete with each other directly.

We wanted to make it easy for *newbies* (newer players) and *elders* (more experienced players) to play together. This encourages socialization and helps build community within the game. Many experienced players enjoy helping out newer players, and expert guidance helps to build a newbie's confidence. We further encourage this by explicitly rewarding elder players when they help Newbies fight Cogs in the streets and buildings. We designed minigames so that players of any level can play together and still benefit from it. Having newbies join them in battle does not penalize elder players; all players receive rewards proportional to their contribution to the battle (avoiding the problem known as kill stealing, which plagues many other games). Newbies also receive some form of protection by battling with more experienced players; Cogs tend to attack the higher-level players.

Figure 6 shows the relationship between the various Toontown activities. Our goal was to divide a player's time into roughly three equal parts: battle, minigames, and social/playground activities. The activities were designed to be interdependent. To fight Cogs, Toons need gags. To buy gags, Toons need jellybeans. To earn jellybeans, Toons

play minigames. To play minigames, Toons need friends. To meet friends, Toons head to the playground. In the playground, Toons heal up for battle, and the cycle continues.

Fig.7. Using a portable hole to teleport to a friend.

2.4 Making Toontown Friendly

To address some of the social problems that plague existing MMPs, we expended considerable energy trying to identify ways in which players can abuse our various game systems. We were particularly mindful of *griefing*, a behavior in which players repeatedly harass other players, both verbally and through abuse of game features. Griefing is a kind of metagame; many players engage in griefing solely for the sake of ruining the experience of others, even when their behavior gives them no in-game benefits. Almost all of our systems in Toontown were designed to minimize the effects of griefing. This includes the positive, game-related nature of SpeedChat; the use of password protection and extensive filtering for open chat; the funny name generator to eliminate offensive and inappropriate names; and battle and mini-game designs that encourage collaboration and minimize a player's ability to monopolize resources at the expense of others.

We also tried to focus our efforts on creating fun experiences, choosing to reduce or eliminate game systems that introduce negative elements or unnecessary drudgery. For example, we chose not to include player vs. player battling, or to give Toons the ability to steal or hoard items. The very simplicity of the Toontown economy affords players few opportunities for disruptive behavior. We also didn't want people to waste time finding their friends. Each player has an integrated in-game friends list that lets him see which friends are online. Players can send messages to their friends, and even better, can instantly teleport to them using their "portable hole," even if they are on another server. Toons pull a portable hole out of their pockets, jump in, and pop out of another hole right next to their friend (Figure 7). This saves the Toon from having to run about the world searching for his friends.

Players also cannot do any permanent damage to their Toons. If a Toon is defeated in battle, for example, it does not die, it simply becomes sad and heads back to the playground until it is happy again. This "soft punishment" actually encourages the sharing of characters among family members; a parent does not have to worry that his child will irreparably damage his character, losing status or valuable items.

2.5 Keeping It Simple, But Not Stupid

As mentioned above, Toontown is, at its core, a role-playing game (RPG). In an RPG, a player creates and develops a character in a simulated world where events unfold over time. Many RPGs, however, can be quite complex, with advanced numerical statistics, game mechanics, and interfaces. Though desirable for more sophisticated gamers already familiar with these elements, the Toontown target audience is children and casual gamers. One of our chief design challenges, therefore, was to simplify Toontown's game mechanics and interfaces, while retaining many of the interesting qualities of an RPG.

We chose, whenever possible, to represent information in the game graphically, avoiding complex numerical statistics. All graphical representations are themed; for example, a player's health is represented using a graphical *Laff* meter, instead of a life meter.

The economy is simple. All gags cost just one jellybean, and players are limited in how many gags they can carry. We avoid complex economic systems such as trading; all items have fixed costs and all players have equal ability to earn jellybeans.

Toontown uses simple keyboard controls (arrow keys) for navigation. We avoided complicated modifier keys and hotkeys. Environments, though varied and visually interesting, are designed to be easy to navigate; we avoid loops and dead ends.

More complex concepts in the game are introduced gradually. Players begin the game in a simplified single-player tutorial. This allows us to introduce game concepts in a controlled setting before overwhelming the player in a bustling multiplayer space. During focus testing we found that spending a brief time learning the minimum skills to play saves time in the long run. New players experience less frustration and can get into the action quicker than they could without a tutorial.

Toon vs. Cog battles, the most complex aspect of the game, take place in plain view. This allows newbies to learn by watching more experienced players. During battle, players can see all choices made by other players. This also helps in the learning process, and provides opportunities for more advanced strategy for senior players (such as coordinated attacks).

Since casual gamers typically have less time available for playing games, it was important to design Toontown to support short-term play. We included activities such as fishing and minigames that can be accomplished in just a few minutes. We included a Toon Task system to give players explicit and measurable goals such as "Defeat 3 Cogs". This makes it easier for the player to always know what to do and gives them satisfaction when they accomplish short-term goals.

3. PANDA-3D DESIGN HIGHLIGHTS

Build or buy? One of the first decisions we faced during Toontown development was whether we should develop our own graphics engine from scratch or adopt a commercially available system. Prior to developing Toontown, we had created several generations of real-time engines to support our work in visualization and location-based

entertainment (Figure 8) [Pausch 1996; Schell 2001]. These engines were built on top of vendor- supplied graphics toolkits such as SGI's Performer [Rohlf 1994] and the Evans

Fig. 8. Location-based entertainment.

and Sutherland Interactive Environment Simulation Toolkit (EaSIEST). Using these commercial toolkits enabled us to hit the ground running and gave us rendering capabilities that were highly optimized for our target platforms.

This time around, however, we decided that it was best to build an engine from the ground up. Though it meant a great deal of additional work, we felt it was important for several reasons.

First, and most importantly, we realized that building and maintaining an MMP is a long-term commitment. A successful MMP will run for years, requiring constant upkeep and new features. Though in the short term, it might have been cheaper and/or easier to buy or license an engine, in the long term it was too risky to depend on some third-party middleware developer that could have been out of business in a year or two (as happened to several middleware companies during Toontown's development).

Second, developing our own engine gave us better control over the available feature set, which was important since we wanted a unique look and feel for Toontown. This would have been much more difficult with a "one size fits all" commercial system.

Third, by building an engine in house, we had full access to source code, and were able to avoid complex licensing issues.

Finally, since we continued to do theme park development work for Walt Disney Imagineering, we needed our own engine to support the many nonstandard rendering modes we use in our R&D projects.

The result is our 3D rendering and storytelling system Panda-3D (or Panda as we usually call it).

Panda, which stands for Platform Agnostic Networked Display Architecture, is a powerful rendering engine written in C++. Panda is a full-featured engine whose subsystems include

- particle systems
- physics/collisions
- soft and hard skin character animation
- 2D & 3D NURBS support
- 2D GUI support
- multipass rendering
- a real-time shader framework

Panda also includes a distributed rendering system called Pandamonium that is used for multidisplay rendering on PC graphics clusters. Similar in spirit to WireGL [Humphreys 2001], Chromium [Humphreys 2002], and Blue-c [Gross 2003], Pandamonium works through the distribution of low-level scene graph changes across a network. Pandamonium is primarily used in support of our visualization work and research into the development of immersive theaters.

Panda is open source, and freely available for download at panda3d.sourceforge.net. Toontown makes extensive use of Panda's particle systems for battle effects and atmospheric effects such as snow. Soft skin animation is essential for obtaining cartoon-like squash and stretch for Toontown's characters. A key feature of Panda's 2D GUI system is that it fully integrated with Panda's scene graph architecture. This enables us to create highly dynamic interfaces that can incorporate any arbitrary (including animated) 3D geometry. Due to the relatively low-end capabilities of our minimum platform (300 MHz or faster CPU, 128MB RAM memory, and an 8 MB graphics card), Toontown makes no use of multipass rendering or real-time shaders.

The following paragraphs discuss in more detail some of the key aspects of Panda that we feel were foremost in facilitating the development of Toontown.

3.1 Platform-Agnostic Graphics Engine

One of the most important features of Panda is that it is not tied to a particular operating system; currently, Panda runs under Windows, Linux, and IRIX.

Strategically, this was important, since, to be successful in the mass market, Toontown had to be built on top of a portable and flexible foundation. Though currently it is only being distributed for the Window's operating system (OS), our long-term plans for Toontown include Macintosh and console ports. Being platform- agnostic will make this easier when the time comes.

Operationally, being platform-agnostic means that our server architecture can run on whichever OS we choose (based on robustness, cost, security, ease of maintenance); we have already experimented with many (Windows, FreeBSD, IRIX, Solaris, and Linux) Panda does not depend on a particular low-level graphics API. Software constructs, called Graphics State Guardians (GSGs), decouple the high-level rendering API, seen by the content developer, from the specifics of the low-level software. The GSG plays multiple roles: abstraction of the rendering hardware, filter for unnecessary or wasteful state change, and workhorse to do the actual rendering of the scene graph (via the low-level API). Decoupling the high- and low-level APIs enables us to use many different back-end rendering libraries, even in the same application. New implementations of the GSG can be added without ever having to touch scene graph code. Furthermore, the GSG specifies exactly what functionality needs to be present (or emulated) on a given

platform. Currently, Graphics State Guardians have been written for DirectX, OpenGL, and Pixar's RenderMan [Upsil 1990].

Similar thin abstraction layers exist for other functions, such as sound, networking, process/threading, windowing, device I/O, and so on. Panda can easily incorporate third-party libraries such as VRPN (The University of North Carolina's Virtual Reality Peripheral Network) [Taylor 2001] or NSPR (Netscape Portable Runtime layer found at www.mozilla.org/projects/nspr/), since Panda only calls these functions via abstraction layers. In this way, we can leverage the work done in these areas without having changes in these packages affect the Panda code that uses it.

3.2 Expressive Scene Graph Semantics

With Panda's scene graph architecture, it was easy to create interesting and dynamic worlds for Toontown. Hierarchical object manipulation (such as putting a pie in a Toon's hand), instancing, transformation of objects relative to one another, and flexible and overridable control of an object's local properties (such as color, transparency, and lighting) are all easily handled using Panda's expressive scene graph semantics.

Object properties, such as color or transparency, can be specified on any node within the scene graph, not just on geometry in the leaf nodes. This makes it easy to temporarily and non-destructively modify an object's properties (turning a Toon black when he's covered with ink, for example) by setting that property on any node higher in the scene graph. Removing that change in property restores the object to its original state. If necessary, lower-level nodes can override state changes in higher-level nodes using a simple priority scheme. As a simple reward in game, for example, we make a Toon semitransparent by setting a transparency property on its top node. Using the priority override scheme, the Toon's clothes, which are subnodes within the Toon hierarchy, stay opaque.

The ability to specify state on nodes internal to the scene graph means that an object's state depends on an ordered accumulation of all the states in the object's ancestor nodes (up to the root of the scene graph). To avoid having to aggregate this state every rendering frame, Panda includes an efficient caching mechanism. An object's properties are initially determined by a full traversal of the scene graph, the results of which are cached, with the cumulative results stored at each node. After that, a node's properties are only recomputed when an intervening state has been changed.

3.3 Interactive Scripting and Interrogate

One of the strongest held beliefs in our studio is that interpreted scripting languages are essential when developing high-quality, interactive 3D environments such as Toontown (for an in-depth analysis of the benefits of scripting languages, see Osterhout [1998]). Interpreted, late-binding languages such as Scheme [Dybvig 1996], Squeak [Ingalls 1997], and Python [Beasley 1999] facilitate rapid prototyping, allowing programmers to make changes to their software while a simulation is running. This is particularly valuable, for example, when debugging game logic that depends on being in a particular state that is difficult or time- consuming to attain.

Late-binding languages, however, lack the efficiency of early-binding compiled languages such as C++. One of the key advantages of the Panda system is that it gives the content developer (or showcoder, as we call them) the best of both worlds: a convenient, interpreted scripting layer built on top of an efficient low-level C++ engine. (A similar

architecture can be found in Avango [Springer 2000; Tramberend 2001] and the Korean Institute of Science and Technology's NAVER system [Park 2002].)

Scripting languages let us mock-up behavior quickly, then optimize later. For example, the Cog AI, the Interval code (Section 3.5), and the Chat balloons system were all written first in Python, then ported to C++ for increased efficiency, after we were happy with the design.

To build an interpreted scripting layer on top of Panda, we used our Interrogate system and any interpreted language that has a foreign function interface (allowing calls to C libraries). Interrogate works like a compiler, scanning and parsing C++ code. Instead of creating object libraries, however, it creates an "interrogate database" of objects, methods, global variables, etc., that are contained in the corresponding C++ library. This database may be queried to discover these functional elements and their interfaces. To make use of this database, one creates an automatic code generator in the scripting language that scans the interrogate database and generates scripting language wrapper calls that execute the C library calls via the scripting language's foreign function interface. Interrogate imposes no restrictions on exactly how a scripting language interfaces with the C libraries, allowing them to interface in whatever way best "fits in" with the natural environment of a particular language. Another interface generator with functionality similar to Interrogate is SWIG, the Simplified Wrapper and Interface Generator [Beazley 1996], available at www.swig.org.

With an interactive scripting layer, instances of all exposed C++ objects can be dynamically created and destroyed using scripting language commands. The showcoder can call existing C++ class methods and C++ classes can be extended with methods defined only in the scripting language. If the scripting language is object oriented and supports class inheritance, such as Python, then new classes can be created in the scripting language that directly inherit from C++ classes. Many classes in Toontown, such as the classes controlling the 3D animated characters, have efficient C++ cores with showcoder-friendly extensions written in Python.

By inheriting from the Interrogate-generated glue layer, we are able to use scripting language-specific code to augment the C++ classes, without having Panda directly depend on a single scripting language. Panda, therefore, is scripting language-independent. We have in the past, for example, interfaced Panda with Squeak, an open, highly portable Smalltalk-80 implementation (www.squeak.org). Currently, however, our focus is primarily on Python (available at www.python.org), an interpreted, interactive, object-oriented programming language. Python has a large and active community, a plethora of useful tools and utilities, and is open source, which meshes well with the open source philosophy of Panda.

3.4 High-Level Tools for 3D Storytelling

Over the past ten years we have converged on a set of tools that we consider essential for interactive 3D storytelling. These tools, collectively know as DIRECT (Disney's Interactive Real-time Environment Construction Tools), are built on top of Panda in our Python scripting layer. Some of the most important include

- finite state machines
- events/hooks
- tasks

- intervals

DIRECT also has a suite of interactive world-building tools that includes widgets for direct 3D manipulation of objects (in the spirit of Connor [1992] and Strauss [1993]),

Fig. 9. Toon vs.Cog battle.

through-the-window camera controls similar to those found in Zeleznik [1999], and tools for building either integrated (rendered in the graphics window using Panda's scene graph) or decoupled 2D GUIs (in a separate window rendered using Tk [Ousterhout 1994]).

Finite State Machines (FSM) are used to organize and control transitions between states in objects that exhibit multiple states over time. Toons, for example, can be standing, walking, running, or jumping. FSMs, which are defined as a collection of states and the allowable transitions between those states, are particularly helpful on our game servers, which have to manage the state of the entire Toontown world (keeping track of the state of thousands of persistent distributed objects).

Event/hooks are used to loosely couple program modules, somewhat like a remote procedure call. Typically, we use the event/hook system for dealing with phenomena that occur at non-deterministic times, such as user input. When the player presses a mouse button, for example, a "mouse-button-event" is *thrown*. Alternately, events can be thrown via program execution; a program module might throw an event when an animation has finished playing so that necessary cleanup functions can be called. Objects hang *hooks* on events (i.e., register interest) with multiple objects being able to hang hooks on the same event. When an object hangs a hook, a pointer to that object is stored in a centralized dictionary along with an associated callback to execute. When an event occurs, the dictionary is searched for any interested objects, and if found, the associated callbacks are executed.

Tasks are used to control methods that are going to be called every frame for an arbitrary amount of time. Checking and responding to the current cursor position would be a good example of a task. To coordinate task execution we create a Task Manager. Tasks can be added and removed from the Task Manager's main execution list during program execution. Every rendering frame the Task Manager cycles through and

executes its list of active tasks. When a task finishes executing, it is removed from the task list. Tasks can have an associated priority to help determine order of task execution.

We used to use the Task system to specify complex sequences of action (timelines). This function has largely been taken over by the Interval system, which we discuss in the next section.

3.5 Intervals: Scripting Dynamic Behavior Over Time

Ensuring that all players have a consistent view of the world is one of the biggest challenges of distributed storytelling. We made this even harder in Toontown by allowing (in fact encouraging) friends to teleport to each other at almost any point in time, including in the middle of complex animated sequences such as Toon vs. Cog battles. To handle this situation, we developed *Intervals*.

Interval classes have been created for controlling position, orientation, scale, color, and transparency of any 3D object. We have also created Interval classes for controlling more specialized behaviors such as animation, sound, particle effects, and curve following. Intervals are based upon Panda's generalized linear interpolation functions.

When controlled by an Interval, an object's state is completely defined for the entire duration of the Interval. Intervals can be started, stopped, looped, and set to a specific time with the object's state updated accordingly. To build up complex sequences, we use grouping constructs called Sequences and Parallels. Sequences are used to group together Intervals that are to be executed one after another; Parallels are used to group together Intervals that are to be executed at the same time. Sequences and Parallels can themselves be nested together; multiple Sequences, for example, can be grouped together in a Parallel, which itself can be part of a larger Sequence. The Sequence and Parallel constructs are similar to the DoTogether, DoInOrder functions found in the Alice Interactive Graphics Programming Environment [Pausch 1995], (www.alice.org).

To orchestrate the execution of these arbitrarily complex constructs, we created an Interval Manager (IvalMgr). All Intervals are registered with the IvalMgr when created. The IvalMgr is then responsible for starting, stepping through, and stopping each Interval at the appropriate time. For efficiency, the bulk of the computation involved in processing Intervals has been pushed down to Panda's C++ layer. Only a thin Python layer has been left for the convenience of the showcoder.

Intervals are used throughout the Toontown code base. In the Toon vs. Cog battle sequence above, for example, a complex 3D battle movie is constructed on the fly using Intervals, with attack animations, sound, and effects based upon the Toon's selected attacks, and Cog reactions based upon the statistically determined outcome. If someone teleports to a friend who is in the middle of a battle, we simply compute the time elapsed since the start of the battle movie and begin playback "midstream" (Figure 9).

4. ONLINE DESIGN HIGHLIGHTS

In this final section we focus on some of the design choices we made that relate to the fact that Toontown is an *online* experience. We discuss our strategy of *nothing-but-net* delivery for Toontown, detailing how we made a full-featured 3D game completely downloadable. We also present our low-cost server architecture, with its emphasis on scalability, flexibility, and security.

4.1 Nothing but Net

One of the most significant and far-reaching decisions we made during Toontown preliminary design was that we wanted to make a full-featured 3D CDROM game that was completely downloadable and playable over a narrow-band connection. The reasons for this includes the nature of the mass market (limitations in broadband penetration), economic factors (cost of goods and limitations of retail delivery), and our realization that persistent online worlds need to be treated much more like an ongoing service (with constant updates) rather than a one-off product.

One of the biggest problems with nothing-but-net delivery is that it doesn't result in a very positive new-player experience. Most players, and especially children, have very little patience for an initial three-hour download. We wanted, therefore, to get the game up and running in about 10 minutes. We achieved this goal through two main actions: First, we analyzed all game systems and optimized them to minimize download:

- textures are downloaded as jpegs;
- animations are compressed with a lossy compression scheme. Some animations, however, are selectively left uncompressed due to unacceptable visible artifacts after compression;
- environments are downloaded as a kit of parts, one tree, one mailbox, each single window design, and so on, and then assembled at runtime using an efficient encoding of the street layout;
- characters are downloaded piecewise and mixed and matched by the user during avatar creation, and saved in the database as a small description of the parts;
- s2D GUIs ound effect files are small MP3 files, and our music is efficiently represented as MIDI.

The second step was to design and implement an innovative background download system. We started by downloading the core elements required to get a 3D graphics window up and running as quickly as possible. To keep the player occupied during this download, we played a 2D Flash movie that introduces the Toontown back-story. Once the 3D window was open, we began to download game-related code and art assets, which are sorted by game neighborhood and downloaded in a prioritized order *in the background*. This enabled us to quickly (in a matter of minutes) get the player occupied in some fun 3D activities meant to keep them busy while the rest of the game downloads. These include Create-a-Toon, the activity to design your cartoon avatar, and the Toontorial, where we explain how to play the game. Once we finish downloading the assets for a particular neighborhood, the player is free to enter that region.

One interesting aspect of designing these introductory activities is that we had to balance ease-of-use with speed-of-use. Since the goal of these activities was to kill time, we had to make them last as long as possible without getting too boring or frustrating. Though it would be faster to choose colors and clothes using palettes of choices, we presented it as an open-ended exploration where players cycle through choices one at a time.

4.2 Cheap, Scalable, and Robust: Pick Three

Toontown's game servers are designed to be low-cost, scalable, and robust. The engineering guidelines used when developing the Toontown servers are as follows:

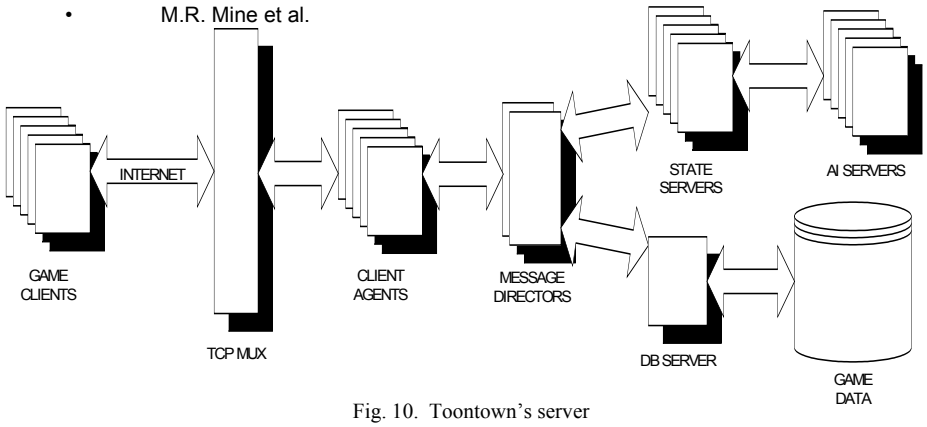

Fig. 10. Toontown's server

- *Internet bandwidth is a premium*. The game server should require no more than 20 kbps of bandwidth to or from a client. Currently, the client-to-server connection averages 2 kbps of inbound traffic and 3 kbps of outbound traffic per active user.
- *The client process and communication stream cannot be trusted*. All transactions from a client are treated as requests for action. The game server must approve them before they can affect anyone else's game play.
- *Critical game data should be protected from loss and corruption*. All data that is flagged as persistent is copied to nonvolatile storage as it is being altered.

Figure 10 presents an overview of the Toontown server architecture. A *TCP Multiplexer* is used to balance the client traffic to the game site, distributing connections to the available *Client Agents*. A client agent shields the site from open network access and is responsible for consolidating and validating incoming traffic. The *Message Directors* act as the central phone book for the system, routing messages to server processes that have registered interest, using a deterministic multicast protocol. There are three types of server processes. The *State Servers* are responsible for managing short-term game state and targeting messages to the appropriate clients as they register interest in new areas of the game. The *Artificial Intelligence (AI) Servers* contain the bulk of the game logic. They control the behavior of all objects in Toontown's distributed world. The *Database (DB) Server* is responsible for the long-term backup and storage of game information.

It may be helpful to think of the information in the system in a general flow model:

- A client requests an action by sending a message to the client agent; sending a request to open a door, for example, so that a Toon can enter a building.
- The client agent receives this message and determines if the client is allowed to send this request.
- If validated, the client agent forwards the request via the message director to all components that have registered interest in it. In this case the client agent would forward the request to a distributed door object, which is typically housed on a state server.
- The state server receives the information and stores and forwards it to other servers that have registered interest in the region the object is located in.

- If the request depends upon game logic, it is sent to the AI servers. The AI will receive the information and possibly trigger new messages based on the type of request and the corresponding game logic. In the case of the door, the AI would send a message indicating that the door was either locked or was opened successfully.
- If this message affects an object that has persistent fields defined, the changes will also be forwarded to the DB server for nonvolatile storage.
- If the message results in a change to the world that is visible to others in the game, the message is sent to all clients that have registered an interest in the local region. Thus, all other Toons within the vicinity of the door will see it open and the Toon walk through.

To keep the entire system flexible, every effort was made to avoid hard-coding game or show-driven constraints. All servers except the AI server knew almost nothing about the game. Information passes through the system using generic messages. Interpretation of the messages depends on an application-specific Distributed Class (DC) file, which the servers use to interpret the data. The only time game-specific knowledge is coded into a server is for security (such as account login and chat), or for optimizations and data integrity.

The servers currently support Linux, FreeBSD, Win32, IRIX, and Solaris operating systems. All servers are written in C/C++ except the AI Server, which is written in Python for all of the benefits of interpreted scripting discussed above.

4.3 Online Worlds Without Borders

One very important aspect of Toontown's server architecture is that it is not designed around a single district (a district is a single copy of the entire Toontown world that can hold up to a thousand players at once). We saw from our guest testing that when friends found each other in the game, they had a much better time. From a server perspective, we wanted to eliminate the technical limitations preventing this from happening. The player's current district, therefore, is nothing more than an attribute on the player's Toon. This is important because multiple districts are required to support more than a thousand simultaneous players. By decoupling districts from the server design, players are free to chat and teleport to friends, regardless of which district they inhabit. This overcomes the limitation found in many existing games that characters must remain in the geographic region or world in which they were created. Moving a character to a different server, in order to play with your friends, for example, is either impossible or in many cases requires paying an additional fee, and cannot be done in real-time.

4.4 Performance

The server components shown in Figure 10 represent logical processes not physical boxes or even executing binaries. A single site will consist of multiple copies of each server process or one big process containing all the functionality described above. A single physical PC can support multiple server processes (depending upon its processor speed). The system is highly robust, all components are hot swappable and can be brought up and shut down as needed.

It is believed that the current system can handle about 50,000 concurrent players without hitting physical barriers. The foreseen limiting factors are as follows:

- The Database Server can sustain around 10,000 distributed updates a second. This is a budget of 1 update every 5 seconds per active player.
- The Message Director is a star configuration. All tests indicate it will be Network Interface Controller (NIC) bound and not CPU bound at around ½ gigabits per second of sustained traffic.
- The Client Agents seem to be open TCP connection bound. Although we have tested them to 100k plus open connections, we currently feel comfortable with allowing around 5k concurrent connections per client agent process. Note that most of the work for TCP is handled by the OS and not the application. We believe that that server-side bandwidth will not be the problem. The ability of the TCP multiplexer and client agent to handle this many requests a second or even the physical limit of packets on the chosen transport media may be the limiting factor.

5. FUTURE DIRECTIONS

One of the biggest challenges of building a massively multiplayer online world is that it is more than just a game. It is, in fact, an ongoing service. As a result, one is never really done developing content. Environments must be constantly expanded and new game play added to keep the experience feeling fresh and interesting for long-term players. The following paragraphs describe some of the features we are currently developing for Toontown.

To expand on the existing battle system, we are currently creating new areas in Toontown known collectively as Cog HQ. Themed as being the headquarters that are producing the robots overrunning Toontown, Cog HQ will add extra depth to the game by introducing new environments, enemies, and game play challenges for the advanced player. Though much of Cog HQ will be centered around battling Cogs, it will breathe new life into the battle system by introducing new twists to the rules that the players have gotten used to.

To give players a sense of ownership in the world, we are adding Toon Estates. Each Toon will have its own home that can be customized as it pleases by choosing and arranging furniture, selecting wallpaper, and flooring. Housing adds extra breadth to the game. Not only does it add an extra dimension for casual gamers, who may not be as interested in battle, it also gives developers a flexible new area to add content where the rules are not so constraining.

To give players an alternative to the battle system, we are expanding upon our current fishing system. The goal is to create a skill-based system that can be both fun and captivating. Fishing adds both breadth and depth: it gives experienced players a different way to heal and earn jellybeans (taking pressure off the trolley games), and adds a collector's game that can be enjoyed separately from the battle system.

To give players a long-term activity, with new things to look forward to over the life of their subscription, we are adding a catalog system that will make a large amount of new content available on a weekly basis. This will include furniture for the player's estate, SpeedChat phrases, emotes, and other items to support activities such as fishing. The catalog system adds breadth to the game, introducing a new type of collector's game and nonbattle experience for casual players.

6. CONCLUSIONS

When we began development of Toontown Online, we did not fully appreciate the challenges associated with building a persistent online world for the mass market. Much is required to build a safe, social, simple yet nontrivial experience, especially for children. We owe a great deal of our success to the fact that we were building our game on top of such a powerful and flexible foundation. Panda's platform-agnostic, expressive, scene-graph architecture, combined with facile interpreted scripting layer tools, was the key to our success.

7. RELATED WORK

Related online projects designed for kids:

- Whyville (www.whyville.net)
- Neopets (www.neopets.com)
- CastleInfinity (www.castleinfinity.org)

Related MMPs:

- Ultima Online (www.uo.com)
- Everquest (www.everquest.com)
- The Sims Online (www.thesimsonline.com)
- Star Wars Galaxies (www.starwarsgalaxies.com)
- There (www.there.com)
- Second Life (secondlife.com)

Related authoring systems:

- Alice (www.alice.org)
- Kaydara (www.kaydara.com)
- WorldToolKit (www.sense8.com)
- NetImmerse (www.ndl.com)

Related streaming download technology:

- Flash (www.macromedia.com)
- Real Networks (www.real.com)

Other places to get Jellybeans:

- Jelly Belly (www.jellybelly.com)

ACKNOWLEDGEMENTS

Many thanks to all the members of the VR Studio, past and present, for their hard work and dedication. Thanks to Walt Disney Imagineering and the Buena Vista Internet Group for their support and for helping to make Toontown Online and Panda-3D a reality. Thanks as always to Sandra for her incredible support, and to Dylan and Coco for the never-ending joy they bring to my life.

BIBLIOGRAPHY

BEAZLEY, D. 1996. SWIG: An easy to use tool for integrating scripting languages with C and C++. In *Proceedings of the 4th Tck/Tk Workshop.*

BEASLEY, D. AND VAN ROSSUM, G. 1999. *Python Essential Reference.* New Riders Publishing.

BRUCKMAN, A. AND BANDLOW, A. 2002. HCI for kids. In *Handbook of Human-Computer Interaction*. J. Jacko and A. Sears, eds., Lawrence Erlbaum Associates.

CONNER, D., SNIBBE, S., HERNDON, K., ROBBINS, D., ZELEZNIK, R., AND VANDAM, A. 1992. Three-dimensional widgets. *Computer Graphics (1992 Symposium on Interactive 3D Graphics). 25*, 2, 183–188.

DYBVIG, R. 1996. *The Scheme Programming Language: ANSI Scheme*. 2nd ed. Prentice-Hall, Englewood Cliffs, NJ.

GOSLIN, M., SHOCHET, J., AND SCHELL, J. 2003. Toontown online building massively multiplayer games for the masses. In *Massively Multiplayer Game Development (Game Development Series),* Charles River Media, Hingham, MA.

GROSS, M., WUERMLIN, W., NAEF, M., LAMBORAY, E., SPAGNO, C., KUNZ, A., KOLLER-MEIER, E., SVOBODA, T., VAN GOOL, L., LANG, S., STREHLKE, K., VANDE MOERE, A., AND STAADT, O. 2003. Blue-c: A spatially immersive display and 3D video portal for telepresence. In *Proceeedings of the ACM SIGGRPAH 2003 Annual Conference*. ACM Press, New York.

HUMPHREYS, G., ELDRIDGE M., BUCK, I., STOLL, G., EVERETT, M., AND HANRAHAN P. 2001. WireGL: A scalable graphics system for clusters. In *Proceedings of the ACM SIGGRAPH 2001 Annual Conference*. ACM Press, New York.

HUMPHREYS, G., HOUSTON, M., NG, Y-R., FRANK, R., AHERN, S., KIRCHNER, P., AND KLOSOWSKI, J. 2002. Chromium: A stream processing framework for Interactive graphics on clusters. In *Proceedings of the ACM SIGGRAPH 2002 Annual Conference*. ACM Press, New York.

INGALLS, D., KAEHLER, T., MALONEY, J., WALLACE, S., AND KAY, A. 1997. Back to the future: The story of Squeak, a practical Smalltalk written in itself. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications 1997.* ACM, New York.

OUSTERHOUT, J. 1994. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, MA.

OUSTERHOUT, J. 1998. Scripting: Higher level programming for the 21st Century. *IEEE Computer 31,* 3 (1998), 23–30.

PARK, C., KO, H., AHN, H., AND KIM J. 2002. NAVER: Design and implementation of XML-based VR framework on a PC cluster. In *Proceedings of Virtual Systems and MultiMedia 2002 Conference.* 967–975.

PAUSCH, R., BURNETTE, T., CAPEHART, A., CONWAY, M., COSGROVE, D., DELINE, R., DURBIN, J., GOSSWEILER, R., KOGA, S., AND WHITE, J. 1995. Alice: A rapid prototyping system for 3D graphics. *IEEE Comput. Graph. Appl. 15,* 3 (1995), 8–11.

PAUSCH, R., SNODDY, J., TAYLOR, R., WATSON, S., AND HASELTINE, E. 1996. Disney's Aladdin: First steps toward storytelling in virtual reality. In *Proceedings of the ACM SIGGRAPH 1996 Annual Conference*. ACM Press, New York, 193–202.

ROHLF, J. AND HELMAN, J. 1994. IRIS performer: A high performance multiprocessing toolkit for real–time 3D graphics. In *Proceedings of ACM SIGGRAPH 1994 Annual Conference*. ACM Press, New York, 381–395.

SCHELL J., AND SHOCHET, J. 2001. Designing interactive theme park rides lessons learned creating Disney's pirates of the Caribbean – Battle for the buccaneer gold. In *Proceedings of the 2001 Game Developers Conference.* 723–731.

SPRINGER, J., FROEHLICH, B., AND TRAMBEREND, H. 2000. On scripting in dstributed virtual environments. In *Proceedings of the Symposium on Immersive Projective Technologies (IPT) 2000*.

STRAUSS, P. 1993. IRIS Inventor, a 3D graphics toolkit. In *Proceedings of the 8th Annual Conference on Object-Oriented Programming Systems, Languages and Applications.* A. Paepcke, ed. ACM Press, New York, 192–200.

TAYLOR R., HUDSON, T., SEEGER, A., WEBER, H., JULIANO, J., AND HELSER, A. 2001. VRPN: A device-independent, network-Transparent VR peripheral system. In *Proceedings of the ACM Symposium on Virtual Reality Software & Technology 2001, VRST 2001*. ACM, New York, 15–17.

TRAMBEREND, H. 2001. Avango: A distributed virtual reality framework. In *Proceedings of Afrigraph 2001*.

UPSILL, S. 1990. *The RenderMan Companion.* Addison-Wesley, Reading, MA.

ZELEZNIK, R., AND FORSBERG, A. 1999. UniCam - 2D gestural camera controls for 3D environments. In *Proceedings of 1999 Symposium on Interactive 3D Graphics*, ACM, New York.

Received ; Revised…