GOGS: USC GamePipe Online Game Server

Michael Zyda
USC GamePipe Laboratory
941 W. 37th Pl., SAL 300
Los Angeles, CA 90089-0781
+1-310-463-5774
zyda@usc.edu

Devin Rosen
USC GamePipe Laboratory
941 W. 37th Pl., SAL 300
Los Angeles, CA 90089-0781
+1-310-463-5774
devinpro@usc.edu

Bharathwaj Nandakumar
USC GamePipe Laboratory
941 W. 37th Pl., SAL 300
Los Angeles, CA 90089-0781
+1-310-463-5774
nandakum@usc.edu

ABSTRACT

Massively multiplayer online games (MMO) are at the height of interest and growing in popularity in the commercial video game market. New development studios are vying for the position of top MMO, both in the number of users and market value. Recently, interest in online games has entered the realm of educational practitioners and researchers. It was as researchers, as well as game developers and designers in mind, that the USC GamePipe Laboratory has developed its initial version of the academic, open source USC GamePipe Online Game Server (USC GOGS).

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems; real-time and embedded systems.

General Terms

Networked games

Keywords

Networked games; massively multiplayer online games.

1. INTRODUCTION

Online games provide new and exciting ways for educators to disseminate knowledge. Online education opens up new avenues of teaching by allowing customization of material tailored to the needs of students in a controlled fashion. By freeing the conventional, spatial boundaries of education, new forms of collaborative efforts are facilitated. Online digital communities allow new forms of interaction to emerge.

In addition to provoking new forms of interaction in education, online games and communities open up new opportunities for the study of social behavior. The goal, with respect to new research, involves studying the patterns of established and emergent groups in online games. With an open MMOG, researchers would be able to create and test social models in the digital world that could lead to insight in the actual world.

The USC GamePipe Laboratory, in assessing these growing needs, while striving to educate the engineers pursuing the games industry, has developed an initial version of an open source, academic online game server. In the effort to develop an MMOG, we found that there currently are not any freely available, open source solutions to help bridge the technological gap between single player and large-scale networked games. The goal was to provide a simple to use, intuitive software toolkit to allow researchers and small independent designers and developers to create multiplayer networked games. The priority was to eliminate technical challenges, to allow developers to take new or single player games and easily bring them to the online multiplayer community.

2. The Technology

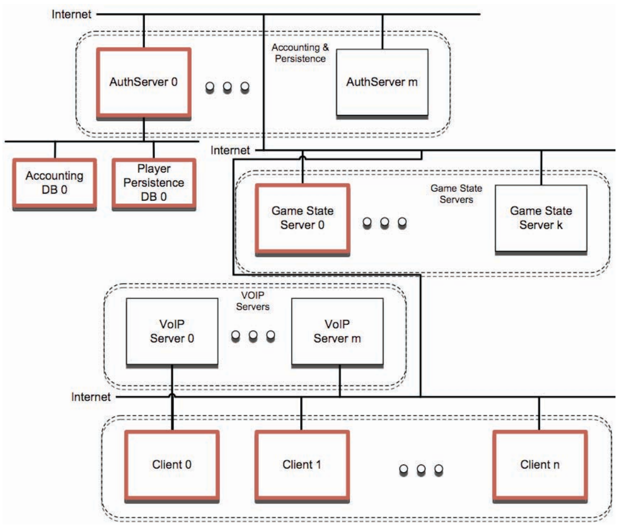
In the USC GamePipe Laboratory, the majority of games are developed in C++. The libraries most-often used are OGRE (Object-Oriented Graphics Rendering Engine) [1] and the FMOD sound engine [2]. Development is done on Windows based PCs. During the course of an average semester about twelve unique games are produced to the point of being a functionally complete and entertaining demo. In addition to these original concepts, a few games are continued from the previous semester for further development. The combination of C++ and OGRE has been so successful that we have created the GamePipe Game Engine (GGE) based around OGRE. The primary target of users for GOGS is the student developer in GamePipe.

By focusing on having the network package tightly integrate with the GGE, we chose to only support Windows based operating systems for both the network server and network databases. This decision may be expanded to support other platforms in future versions. All development of GOGS was in C++ using Microsoft Visual Studio and the database was Microsoft Access. All of the design choices for the engine were to support the Object Oriented design paradigm used in production in the lab.

3. The Design

The design of GOGS is a multilayered pattern to help support the rapidly scaling requirements of networked games. The architecture consists of the Authentication Layer, Game State Layer, VOIP Layer, Client Layer, and Accounting Database, and a Player Persistence Database. Each layer was conceptualized to allow for simple horizontal scaling.

The Authentication Layer is responsible for handling the initial connection of clients as well as verifying the clients. The Authentication Layer validates Clients. After correct validation, it sends both required player information from the Player Persistence Database and the current state of the game. The state sent takes into account the entry point of the player from the Game State Layer. This layer also allows a client to transfer from one Game State Server to another. This verification from the Authentication Layer to switch between servers was chosen to increase the security of the networked system.



The Game State Layer handles all game simulation calculations. The architecture of the system treats the clients as a proxy. This means the players make requests to the actual game characters being simulated on the server. The choice to use a proxy allows for stricter security policies to ensure the clients are limited from cheating the game. This method of control also allows for all game state inconsistencies between clients to be handled by the Game State Layer. By choosing to have a proxy system, we give up faster client side simulation speeds for the sake of having a consistent state of the game for all clients.

The addition of the VOIP Layer was added after the initial design of the system. The decision to include VOIP in the engine was based on the observation that if an MMOG did not support verbal communication in the game, players would turn to outside software packages to facilitate group communication and collaboration. The VOIP Layer only supports direct communication and group communication. This decision was based on two pieces of information. The first mitigating factor was the amount of bandwidth required to support the constant flow of audio data. We wanted to limit the amount of data sent to clients in similar geographic regions within a game, so we did not provide the capability to transmit voice to general players in an area. The second reason for limiting the communication to individuals or groups was for the security of the players if the engine was to be used for an academic game or as an academic tool. This decision was based on the safety of the player. We limited verbal communication to players in the client's friend list or groups that were chosen by the client.

The Client Layer was designed to be thin, lightweight, and easy to use by developers requiring network support. From the developers' standpoint, GOGS is an event-based system. This means that every object sent over the network is treated in the same fashion. All developers have to do to integrate networking is send and receive events. The decision to make all objects consistent was for ease of design. In order to use any object as an event, it just has to use the Event Interface and be serializable. In addition to simplifying the use of objects, we made the login process of clients easy by limiting the amount of code required connect to the engine.

The Player Persistence Database is responsible for keeping a cached copy of the client status on disk in the event of a system failure. This database maintains persistence between client sessions.

As a whole, GOGS features an event-based system with global time. Maintaining a global time eliminates any playback or packet flooding attacks. GOGS includes Lobby code that allows players to communicate by either text or verbal chat with other players in the lobby, as well as players in the current game.

4. Current State

The current version of the GOGS system was designed and implemented over the course of one semester. A team of eight graduate level computer science students in the USC GamePipe Laboratory created it. In order to facilitate the rapid design and testing of GOGS in a condensed timeframe, we limited the scope of the project to a single server for each of the layers. Taking an existing, session based network game and replacing all network code with GOGS compliant code initially tested this version of the engine. This allowed the game to maintain state between sessions and also enabled easy integration of VOIP.

5. The Games

The games developed using GOGS helped in determining the efficiencies and deficiencies of the initial design and implementation of the system.

Penguins is a multilayer snowball throwing game. The game revolves around groups of penguins engaging in a friendly snowball fight. Each player controls a group of matching penguins. The player controls one individual penguin in the group while the rest of the penguins on the team are there for supporting snowball attacks. The team penguins are controlled by AI to throw snowballs at the nearest threatening penguin or defend the player. During this snowball fight, no blood is shed, but rather penguins slowly turn into ice-cubes as opponents' snowballs hit them.

Originally the game was based on rounds in short individual sessions. It supported a maximum of four individual players during any given session. The particular challenge in designing the game was to minimize the number of packets sent while still allowing a large amount of data to be passed over the network. Every snowball thrown in the game generated data that needed to be sent to maintain a consistent state in the game. We had to primarily optimize the way snowball information was sent in order to allow players to throw snowballs as rapidly as they could click the mouse button. We intentionally did not want to place a limit on the number of snowballs created in any given game session. The reason for not limiting this was to see how far we could push the capability of the network software as well as maximize the level of fun in the game. This first version of the game was created using RakNet [3].

The second version of Penguins was converted to use the GOGS architecture. By switching over the network code, we were able to take advantage of several of the features that were built into the engine. The first noticeable benefit was the addition of player persistence and the Authentication Layer. After integration, we were able to create client accounts with passwords. In addition to these accounts, the player persistence database maintained the player preferences when re-entering the game. Players could maintain the costumes they had previously selected for their penguin squad. In the game, you could choose a hat and costume for your team to differentiate between the penguin groups. Statistics of the rounds were also collected on the player persistence database.

Another benefit of GOGS that related to gameplay was the easy addition of VOIP. With the addition of verbal communication, we saw changes in the way alliances were formed during game sessions. The second version of Penguins included a lobby that supported both text and verbal chat between players in the lobby as well as in the game. The most significant improvement to the Penguins game was the increase in the number of supported players.

Lockdown is currently in development in the USC GamePipe Laboratory. This game is a training simulation for first responders in hostile situations. The purpose of the simulation is to assess the players on their level of performance in handling threatening situations. The game is a squad-based game where multiple players are on a single team with the primary goal of securing a hostile environment. While the first objective of the game is to secure the environment, secondary goals include a triage mini-game in which players assess and classify any injured non-playing characters (NPC). Players are ranked based on their accuracy of classifying these NPCs in addition to the amount of time it takes them to perform the triage and to secure the environment. Lockdown was a prime candidate for GOGS due to the requirement of allowing the squad to have verbal communication during the simulation.

One of the bigger challenges in Lockdown, which we did not have to address in Penguins was the size and scope of the terrain. Penguins took place in a relatively small piece of terrain that was essentially a flat, confined, outside area. Lockdown in contrast occurs in a building with multiple floors. Each floor contains a variety of rooms, doors, hallways, and stairs. The game also supports both interior and exterior gameplay in relation to the building. The diverse terrain had no immediate impact on the performance of GOGS. Lockdown also features more simultaneous clients than Penguins supported. Lockdown will support eight to sixteen human players in addition to countless NPCs that are controlled in the Game State Layer.

6. Conclusion

As a team, we are thrilled about the successful use of GOGS in both of these games. The most important part is that the interest to continue to design and develop GOGS is spreading among graduate level students. These students realize the importance of having a successful open source solution to this challenging problem. There are a variety of technical improvements that will be made in the next iteration of GOGS. These improvements include a variety of optimizations. We plan to include increased threading support for buffering threads on both the client and server layers. Another step is to implement the horizontally scalable layers with a distributed server system that uses the publisher/subscriber model over TCP/IP channels. We would like to expand upon the level of packet compression and optimization. The next version will also include dynamic area-of-interest management (AOI).

We plan to extend GOGS beyond the USC GamePipe Lab to the general game development community in order to promote creativity and community in game design and development [4].

7. ACKNOWLEDGMENTS

We wish to thank all the students involved in the development of GOGS, including Bharathwaj Nandakumar, Devin Rosen, Praveen Kansara, Balakrishnan Ranganathan, Apar Suri, Ivan Chen, Sunil Shenoy, Pravin Babar and Nathan Greenfield. We additionally thank the National Science Foundation for its support of this project through the USC Integrated Media Systems Center.

8. REFERENCES

[1] Ogre3d http://ogre3d.org

[2] Fmod http://fmod.org

[3] RakNet – http://www.jenkinssoftware.com/

[4] USC GamePipe Laboratory web site http://gamepipe.usc.edu