

3D Game Engine with the .NET Framework

By

Quentin Maxwell Randolph

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

University of Cincinnati
College of Applied Science

June 2005

3D Game Engine in the .NET Framework

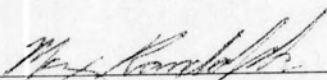
by

Quentin Maxwell Randolph

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements
for
the Degree of Bachelor of Science
in Information Engineering Technology

© Copyright 2005 Quentin Maxwell Randolph

The author grants to the Information Engineering Technology Program permission to reproduce and distribute copies of this document in whole or in part.



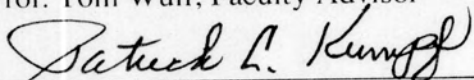
Quentin Maxwell Randolph

06/02/05
Date



Prof. Tom Wulf, Faculty Advisor

06/02/05
Date



Dr. Patrick Kumpf, Interim Department Head

6/2/05
Date

Acknowledgements

I would like to extend special thanks to my parents for giving me the support and attention I needed during this project. And I would like extend special thanks to my teachers who allowed me to turn what ever they were teaching into part of my project. I would like give special thanks to all of the people that believed that I could do this. Without them the doubters might have been able to sway me from my course.

Table of Contents

<u>ACKNOWLEDGEMENTS</u>	1
<u>TABLE OF CONTENTS</u>	2
<u>LIST OF FIGURES</u>	4
<u>ABSTRACT</u>	5
<u>ABSTRACT</u>	5
<u>INTRODUCTION</u>	6
<u>PROBLEM STATEMENT</u>	10
<u>THE SOLUTION</u>	11
USER PROFILES	15
DESIGN PROTOCOLS	18
<u>DELIVERABLES</u>	28
<u>DEVELOPMENT</u>	30
TIMELINE	30
BUDGET	32
<u>PROOF OF DESIGN</u>	33
THE CHAOS ENGINE	33
DIRECT X RENDERER	33
X FILE IMPORTING	34
3DS FILE CONVERSION TO .X FILES	35
HEIGHT MAP RENDERING	36
DIRECT INPUT MODULE	36
INPUT MAPS	38
DYNAMIC INPUT BINDING	38
DIRECT SOUND RENDERER	39
RENDER WAV FILES	39
XML CONFIGURATION FILES	41
COMMAND LINE INTERFACE	43
VSA SCRIPTING ENGINE	43

PLUG-IN PORTS	44
END MODULE	44
GAME FORGE	45
PROJECT WIZARD	45
ENGINE CONSOLE	46
ENGINE CONFIGURATION	47
GAME OBJECT EDITOR	48
INTERACTION EDITOR	49
HTML API REFERENCE	50
EASY TO USE TUTORIALS	51
 TESTING PROCEDURE	 52
 CONCLUSION	 52
 RECOMMENDATIONS	 53
 REFERENCES	 54

List of Figures

Figure 1. Shows how a game engine works	6
Figure 2. List of three well known off the shelf engines that I have experience with	7
Figure 3. Screenshot of a map rendered using the Genesis3D engine.	7
Figure 4. This is a screenshot of the ship demo using the Fly3D engine.	8
Figure 5. This is a screenshot of Realm Wars Community Project which has been created using the torque engine. This screenshot displays the animation capabilities of the engine.	9
Figure 6 Shows the target users and goals of the these users.	17
Figure 7 Layered organization of the Chaos Engine	20
Figure 8. Data flow down into the engine.	21
Figure 9 Design of the Game Project Wizard	24
Figure 10 User Interface for game forge	25
Figure 11 Screenshot of engine API reference	26
Figure 12 Timeline of the development process	30
Figure 13 Software I needed to do this project	32
Figure 14 Software I needed to organize this project	32
Figure 15 Additional resources	32
Figure 16 Engine rendering the tiny.x file after importing it.	34
Figure 17 Shows the menu item that triggers a conversion	35
Figure 18 Bitmap that is used to generate the height map	35
Figure 19 Ground texture for the terrain	36
Figure 20. XML Representation of a simple key binding	37
Figure 21 Screenshot of a dynamic binding	38
Figure 22 How a game engine works	39
Figure 23 Code Snippet showing how to transform an xml node into a live object	41
Figure 24 Code snippet of the xml server retrieving an xml node to be converted into an object	41
Figure 25 Command pattern used in the Chaos Engine	42
Figure 26 Code snippet that adds a free fall controller to the airplane that cause the crash sequence	43
Figure 27 New Project Wizard	45
Figure 28 Result folder of creating a new project	46
Figure 29 At the top shows a command issued through the console	46
Figure 30 Engine configuration form that allows you to alter sub system settings	47
Figure 31 Game Object Properties box	48
Figure 32 Interaction editor for game object interactions	49

Abstract

For my Senior Design project, I designed and developed a fully featured 3D game engine using the .NET Framework, Managed Direct X and various other technologies to address the common problems I experienced in game development. The final product is not be a game; instead the final product is a flexible object-oriented framework and toolset from which to build video games rapidly. This project consists of:

- An Engine with Simple Architecture
- Easy to Use Integrated Toolset
- Thorough Documentation

The Chaos engine is built for people that do not have the resources of a professional game developer. It is intended to make simple games cheaply and rapidly. This will open the world of game development to a broader base of people.

Introduction

The developers of Doom, id software did more than create a ground breaking video game, they also popularized a new game programming model called the game “engine”(8). A game engine is a modularized, data-driven architecture that facilitates the ability of game developers and the gaming community to tweak an existing game, or make an all new game without the need to alter the core architecture of the engine(5)(7).

A game engine, while unique, works the same ways as all other game engines. A game developer feeds an engine game content, game rules and configuration data. The engine operates on those elements to produce a game.

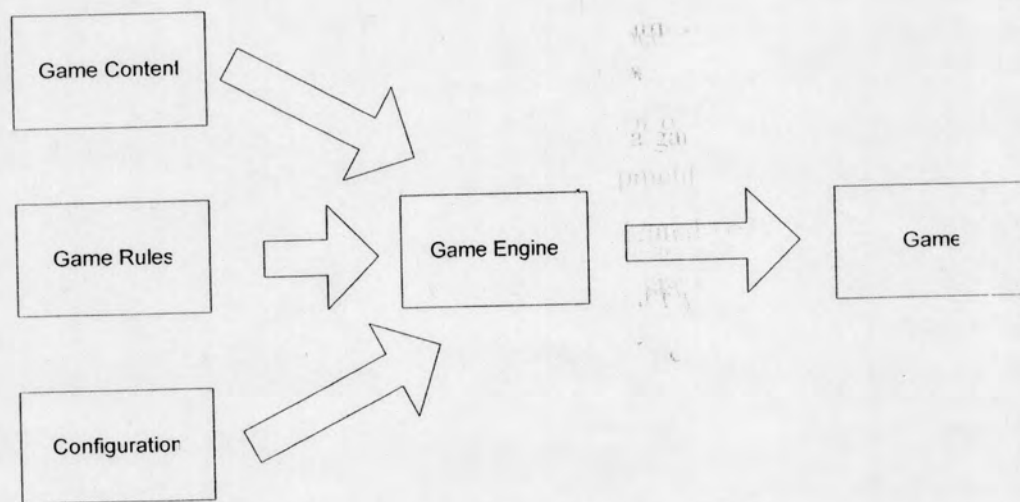


Figure 1. Shows how a game engine works

The idea is that if someone wants to make different games, he/she would only have to change the elements that went into the game engine. Modification of the core game engine should not be necessary. The over all goal of a game engine is to make game development easy.

Examples of Other Game Engines

There are a number of off the shelf game engines that try to make game development easy. Below in Figure 2 are a few of the engines that I personally used. Each of these

engines have common problems that make game development unnecessarily difficult.

Game Engine	Developers	Cost
Genesis3D	Digital Delirium Design	Open Source (free)
Fly3D Engine	Paralelo	\$70
Torque Engine	Garage Games	\$100.00

Figure 2. List of three well known off the shelf engines that I have experience with

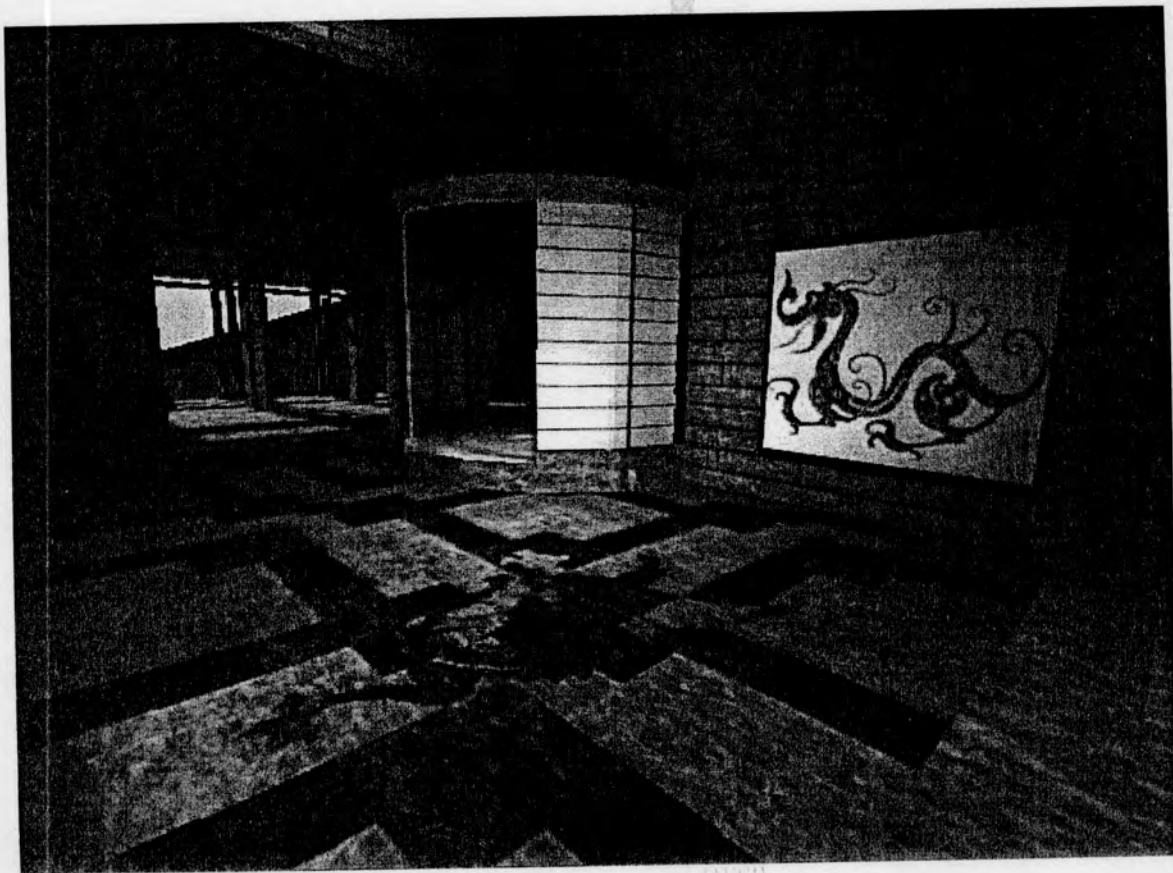


Figure 3. Screenshot of a map rendered using the Genesis3D engine.

Genesis3D is an open source project from Digital Delirium Design. The Genesis3D engine is a First Person Shooter centric game engine based on older technologies, but it is still capable of supporting compelling game content.

The documentation covers the API, but it lacking tutorials on how to actually use the engine. Using Genesis3D engine would mean that a user would have to reverse engineer the engine to understand how it works and how to extend it.

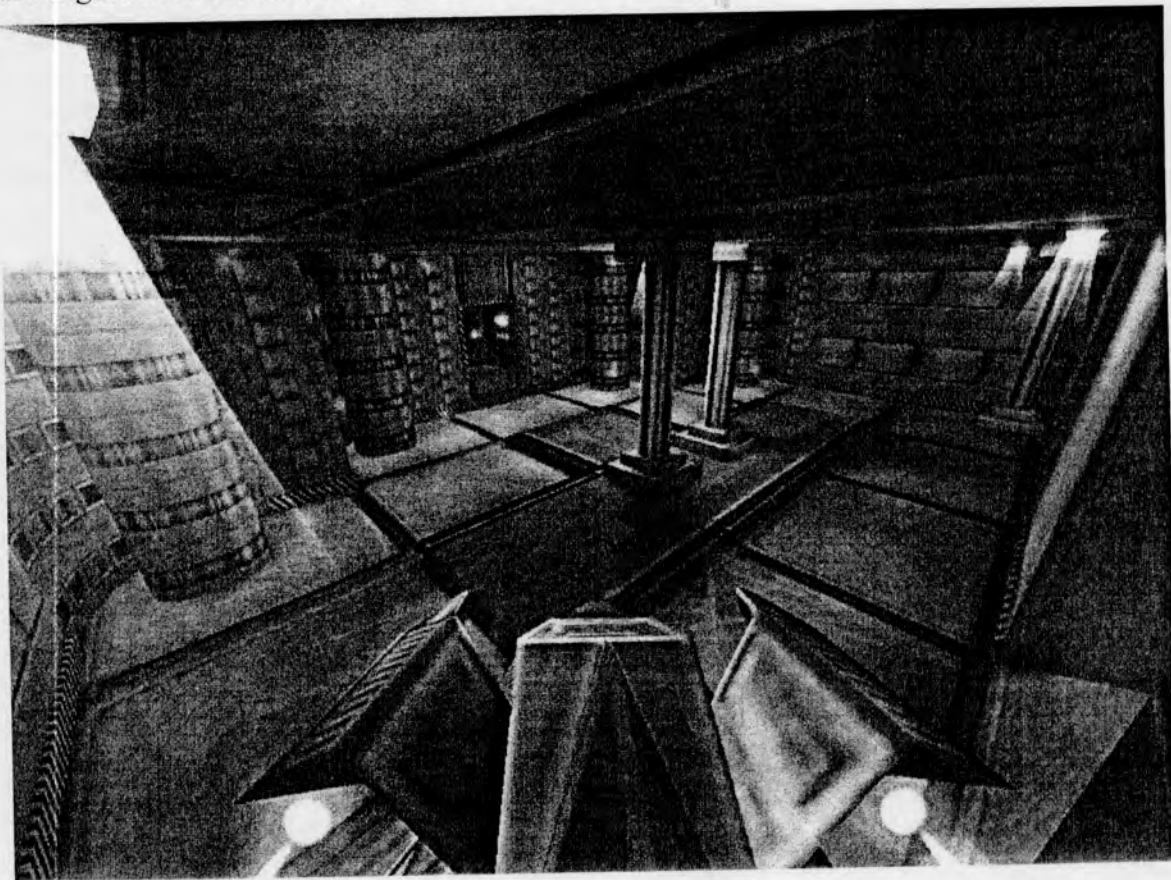


Figure 4. This is a screenshot of the ship demo using the Fly3D engine.

The Fly3D Engine, made by Paralelo, is a more general 3D engine. Fly3D is packaged with Alan Watt's 3D Games: Real Time Rendering and software technology book on game engine design and the implementation of Fly3D. Also included are a small set of tools that aid in creating game content. This engine is also a few years old. However, there are updates every couple of months and there is a large amount of documentation

and tutorials on how to use the engine. Another attractive feature is that the implementation of the tools is exposed so that a user is free to modify existing tools and make new ones.(9)

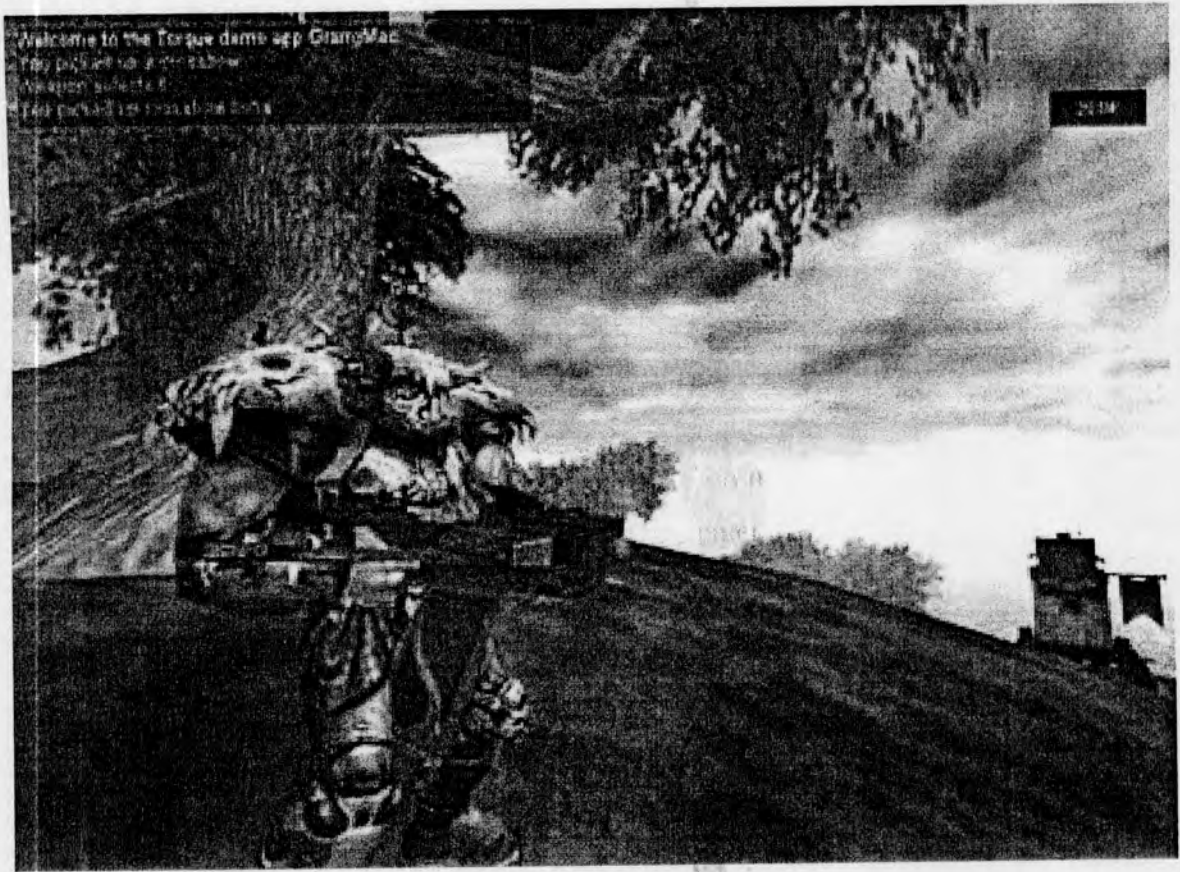


Figure 5. This is a screenshot of *Realm Wars Community Project* which has been created using the torque engine. This screenshot displays the animation capabilities of the engine.

The Torque Engine is the most popular of the three. Torque was developed by Garage Games which is the largest independent game development community. Torque comes with an extensive set of tools to aid in creating a game. The documentation is the most extensive of the three. And there is a large community from which to draw resources. The tutorials provided are detailed on how to use the engine and the tool set. And since there is such a large community that uses this engine, finding content or help is not a problem.

Problem Statement

I have used each of the engines mentioned to try to make a video game. Each time that I tried to do this I ran into common problems that mired my progress. These problems were discussed Dr. Michael Young of NC State University. When asked about the common problems with the game engines, Dr. Young pointed out three problem areas. These problems are things I have seen in game engine design books and are same problems I have encountered when trying use the game engines mentioned above.

Complex Engine Architecture

The complexity of most engines tends to be their major stumbling block. Too often this underlying complexity is exposed to a game developer. If there needs to be a modification to an overly complex engine a developer has to deal with that engines complexity every time he wants to change something. (7)

Lack of Supporting Tools

A good game engine needs to have a set of good tools. For most engines, there are third party tools that help create content but those engines do not have good tools for editing the data configuration files. Again this can be compared to Linux. Altering system configuration files by hand in a simple text editor can be a tedious operation. Using GUI tools to alter these files made operating on configuration files much easier. It is the same with game engines. (7)

Poor Documentation

It is extremely important to have good documentation for a game engine because in a lot of cases game engines can rival or surpass the complexity of some computer operating systems.(7) Without adequate documentation to guide a developer the experience would resemble trying to figure out something like the Slackware distribution of Linux without instructions. Good documentation is essential to making a game engine usable.

The Solution

For my Senior Design project, I propose to develop a fully featured 3D game engine using the .NET Framework, Managed Direct X and various other technologies to address the common problems I experienced in game development. The final product will not be a game; instead the final product will be a flexible object-oriented framework and toolset from which to build video games rapidly. At the conclusion of the project, I will have three technical demonstrations to show the technical aspects of the engine and to show how effective the architecture is at rapid game development.

IT Emphasis

Building a game engine is a very complicated project. It involves many, if not all, aspects Information Technology. But there are two aspects of IT that this project focuses on. The first and most prevalent IT area being represented by this project is the Programming aspect. This project is 95% programming. I have to program the engine and the tools. The second most prevalent IT area being represented is Multimedia. A game is a structured and interactive collection of multimedia elements. The engine resembles many aspects Macromedia's Flash Player. They both take sounds, images, movies and other multimedia elements and combine them into a multimedia production.

This project will address the three main problems areas discussed above. Well designed engine architecture is the primary focus of the solution. Development of useful tools to modify engine is the next priority. Good documentation is also an important part of what makes a game engine successful.

The Chaos Engine

There will be a number of features built into the engine that will facilitate the easy use of the engine. The first is that the engine will be designed with good object oriented practices. This means that the functionality of the engine will be encapsulated into well defined and modular classes. The engine will also be designed to be modified, extended and reused. The major features of the engine include:

- **Modular Design**

The engine will be highly modularized. Most of the major subsystems of the engine are stand-alone with only a few tenuous cohesive elements between them. This will facilitate three things. The first is that the engine will be extremely stable. If one sub-system fails it will not break the entire engine. The second is that it will make it easy to extend the engine's architecture. Since the engine is modular a developer will not have to delve into multiple sub-systems to make a new component. And the third is that troubleshooting is easy because the engines functionality is encapsulated.

- **Data Driven Architecture**

The engine will have a data driven architecture. This means that elements that are normally hard coded, namely constants, will be soft coded. This means that the engine will use a simple XML based database to hold all of the constants for the engine. The reason for using XML as apposed to using an SQL based database is that the focus of this engine is to be easy to use. Using an SQL based database means that someone who wants to modify engine constants would have to know how to use an SQL based database. XML is a plain text file format. This means that someone with no experience with XML can still enter the file and alter engine constants.

- **Interoperability**

The interesting thing about the .NET framework is that code written in one language in .NET can be used in another .NET seamlessly. For example, the core of the engine will be written C#. If a developer only knows VB.NET or J# that is no problem. He/she will be able to write engine extensions in their native language. This means that development teams don't have to take time to learn the language of the engine he/she simple can use the language he/she know.

- **Integrated Scripting**

The engine will have an integrated scripting engine. Most scripting engines have to have a method of mapping engine functionality to the scripting engine. Another way is to have a developer to write their own script interpreters. Both of these methods can be time intensive, complicated and very tedious. Using Microsoft's VSA Engine frees a developer from having to deal with the complexity of getting a script working. The scripting engine uses native .NET code and though reflection, invokes methods, classes and structures seamlessly. The VSA Engine that I will be using current supports VB script and Jscript which both are widely known and very familiar languages. This will make modifying engine behavior easy.

- **Plug-in System**

The plug in system is a way to further modify the architecture of the engine. This will allow the developer to add features to the engine without having to go into the finished core of the engine.

- **Standard File Formats**

Too often game developers create their own proprietary file formats and pack files to intentionally make it difficult to use their content. This is understandable because they would like to protect their investment. However, when a developer is using their engine it makes it difficult for someone to get game content into the game. The developer ends up needing special tools and plug-ins to convert their work into a format that the engine will use. The engine I will be build will us standard file formats for things like models, sounds and music.

The Tools

Tools are an important part of a game development package. Tools can make interfacing very easy and thus speeding up the development cycle. Some of the features the tool will have are as follows:

- **Project Templates and Wizards**

There are a finite number of video game types. Instead of having to re-invent the wheel every time a developer wants to make an FPS (First Person Shooter), there will be an FPS template built into the engine. It will be as simple as selecting the template and the tool will take care of creating the project and setting up basic scripts and configuration files for you. Then the developer can spend his/her time working game content and game play.

- **WYSIWIG User Interface**

"What you see is what you get" is a concept that is important in trying to put together multiple multimedia elements. Since the engine is embedded into the tool, the way something looks in the tool will be the way it looks in the engine.

- **Visual Scripting Interface**

A lot of programmers are accustomed to the visual programming. And most know that the visual programming paradigm has drastically sped up application development time.

The Documentation

The documentation of a game engine is very important part in educating a developer in the use of the engine. Most engines lack sufficient documentation but this one will not.

The following material will be included in the documentation of the engine:

- **API Reference**

This is important to any developer that needs to modify the core of the engine to achieve something new. The API reference will outline the intention and functionally of every class and function with in the engine architecture. The goal would be to have a documentation structure similar to MSDN library.

- **Tutorials**

The tutorials will show how to use the tools to make a game project happen. The focus of these will not be reference it will be to educate the developer on the use of the engine. The tutorials will include well defined steps with many pictures so a developer can not get lost.

User Profiles

The Chaos Engine will have to accommodate a number of users. During the game development life cycle dozens of people might have to interact with the engine to make a game. Those people can be broken down into four general groups. These groups are:

The Programmer

The programmer's role is to interact and develop programming code for a game. This person needs to know basic procedural, object-oriented and event based programming. It would be helpful for this user to have some experience with Visual Studio .NET but is not required.

The programmer's interaction with the engine will be programming related. They will have to write and modify Game DLL's in a .NET language. They will also need to write scripts using VBScript or Jscript. The goals of the programmer role are as follows:

- Write Engine Plug-ins
Engine plug-ins are DLL extensions of the engine core. The end goal is to make it so that the average programmer will not have to write a plug-in but there are situations that will warrant it. The number one reason is performance. A hard coded element of the engine will run much faster than a scripted element.
- Write Game Play Code
The game play code of a game can be described as the rules of the game. The goal of this project is to make tools with a library of common rules that a game designer will be able to simply plug into the game with out having to code any thing. But there will be new game play elements that I did not preconceive. This is the situation that a programmer will be needed to code new game play elements.
- Alter Engine Source
There might be a need to alter the way the core engine functions. This will be a rare job for the programmer but the need may arise.

The Content Creator

This user represents the artists of the game development group. These users will need to know how to use 3D modeling software, skinning tools, Photoshop and various other

multimedia applications. This user will have to have the most basic of programming ability.

The content creator's interaction with the engine is limited to content management. To them the Chaos Engine will resemble a production application. They will create content in applications like 3D Studio Max or Photoshop and import the media content into the engine for use. The content creator's goals are as follows:

- Test Game Content in Engine
Content created in other applications have to potential to look very different when run inside of the engine environment. That is why it is important to be able to test and configure game content.
- Design UI for Game
An important part of any game is the User Interface. There are limited choices for designing and implementing user interfaces for a game. There is a need to have a tool that allows the designer to do this with out having to hard code the user interface.

Game Designer

This user designs the overall game play of the game. They would need to know a little about all of the other roles. The game designer will look at the game development process from an abstracted prospective.

The game designer's interaction with the engine is from a supervisory position. They will be using all of the tools that the engine provides however the visual tools will be most used.

- Layout game flow
The game designer will have to layout the game play of the game. They will not want to have to do programming to accomplish this.
- Define game object interactions
Central to game play is how the game objects interact with each. This will be a very important part of what the game designer does. It will be useful to have tools that allow the designer to manipulate these interactions easily.

Game Player

The game player is the person the buys the end product of the game development team's labor. They need to have played other video games and have some basic knowledge of the hardware that the game is supposed to run on.

This user might want to customize their game. The game player will use the engine's console and configuration files to tweak their game.

Figure 1. High level use case diagram showing user goals and relations.

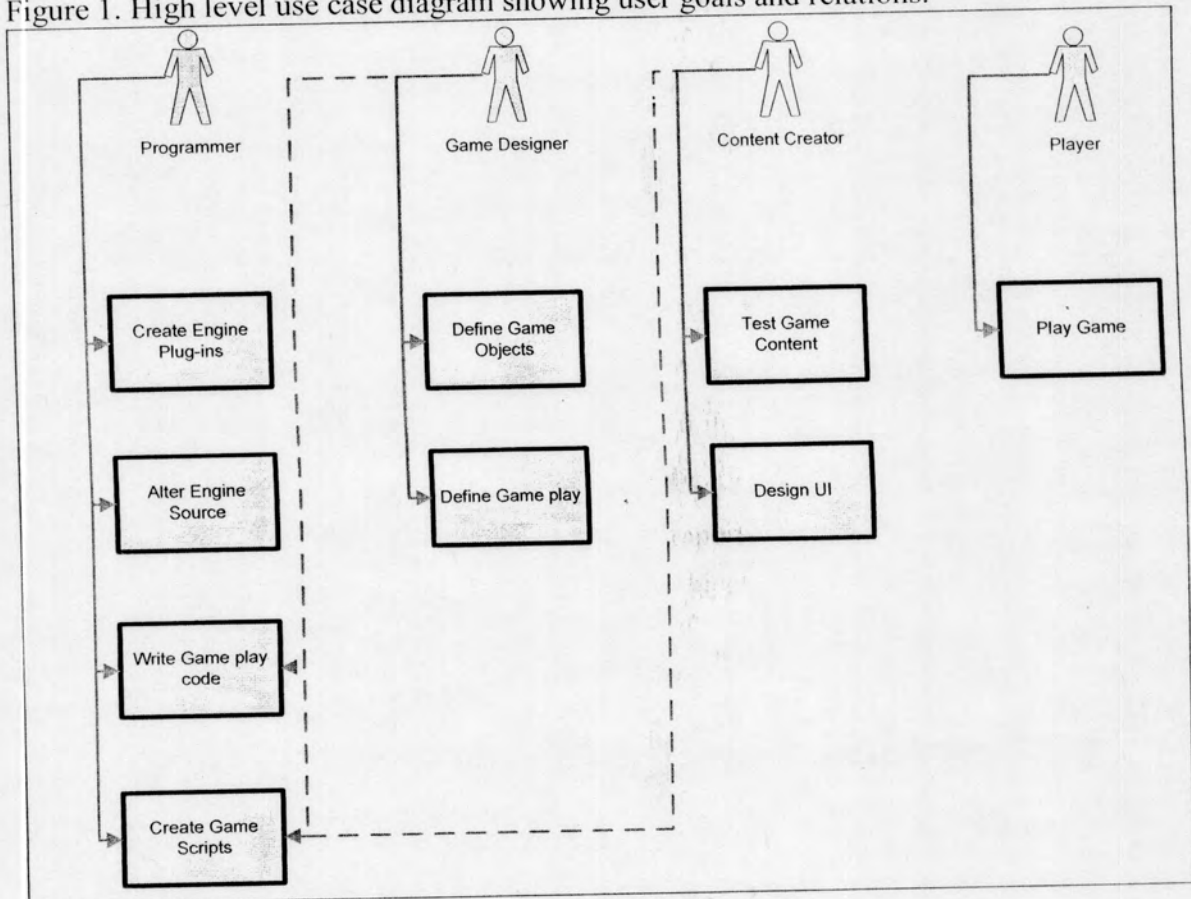


Figure 6 Shows the target users and goals of the these users.

The figure above shows the goals of the users and how they relate. The solid lines leading from users to goals are represent a primary goal. The dotted lines represent secondary goals.

Design Protocols

Due to the risky nature of this project and the fact that a number of elements that make up the project are largely unknown, I have decided to implement this project with an iterative design process. Every iteration will be stand alone and capable of completing the requirements of the project in some limited manor. They will also be stronger and more featured than the last iteration. This way if the development process becomes mired for some reason, I will always have a project to present.

Each iteration will have an overall goal. In most cases, this goal will take the form of a simple game. In subsequent iterations the goals will get more elaborate and thus adding to the features of the engine. At the end of the development process, not only will the project be complete but there will also be excellent set of examples for the end user.

There are 5 iterations that are currently planned.

- **Iteration I: Pong & Core Architecture**
This iteration will be focus on making a port of the old pong game. To do this the core infrastructure will be implemented and the rudiments of the toolset will be started. This iteration will be the longest because the core infrastructure will have to be built.
- **Iteration II: Pong 3D & Collision Detection**
Pong 3D will be the 3D take on the old Pong game. To accomplish this, a basic 3D collision detection system has to be constructed. This will also prompt the usage of some scripting features.
- **Iteration III: Flight Simulation & Basic Physics**
The flight simulation will be a simple game where there is an airplane flying through a landscape. This will involve using models, implementing terrain and sky boxes. This is the point were a tool for these elements will be very useful.
- **Iteration IV: Vehicle Simulation & More Physics**
The vehicle simulation will be some vehicle(hover craft, car, tank) driving on a landscape. This will involve a more advanced physics model than the flight simulation. Elements such as gravity and friction will be incorporated.
- **Iteration V: First-Person Shooter**
The first-person shooter will be the most difficult because it will involve animated models, detailed collision detection, terrain and a few other elements. This will conclude the process and will be the iteration that more care will be given to the tool set.

This approach is in no way fixed. As requirements change the design process can change with little impact to success of the project.

The Engine

There will be a number of features built into the engine that will facilitate the easy use of the engine. The first is that the engine will be designed with good object oriented practices. This means that the functionality of the engine will be encapsulated into well defined and modular classes. The engine will also be designed to be modified, extended and reused. The major features of the engine include:

- **Modular Design**

The engine will be highly modularized. Most of the major subsystems of the engine are stand-alone with only a few tenuous cohesive elements between them. This will facilitate three things. The first is that the engine will be extremely stable. If one sub-system fails it will not break the entire engine. The second is that it will make it easy to extend the engine's architecture. Since the engine is modular a developer will not have to delve into multiple sub-systems to make a new component. And the third is that troubleshooting is easy because the engines functionality is encapsulated.

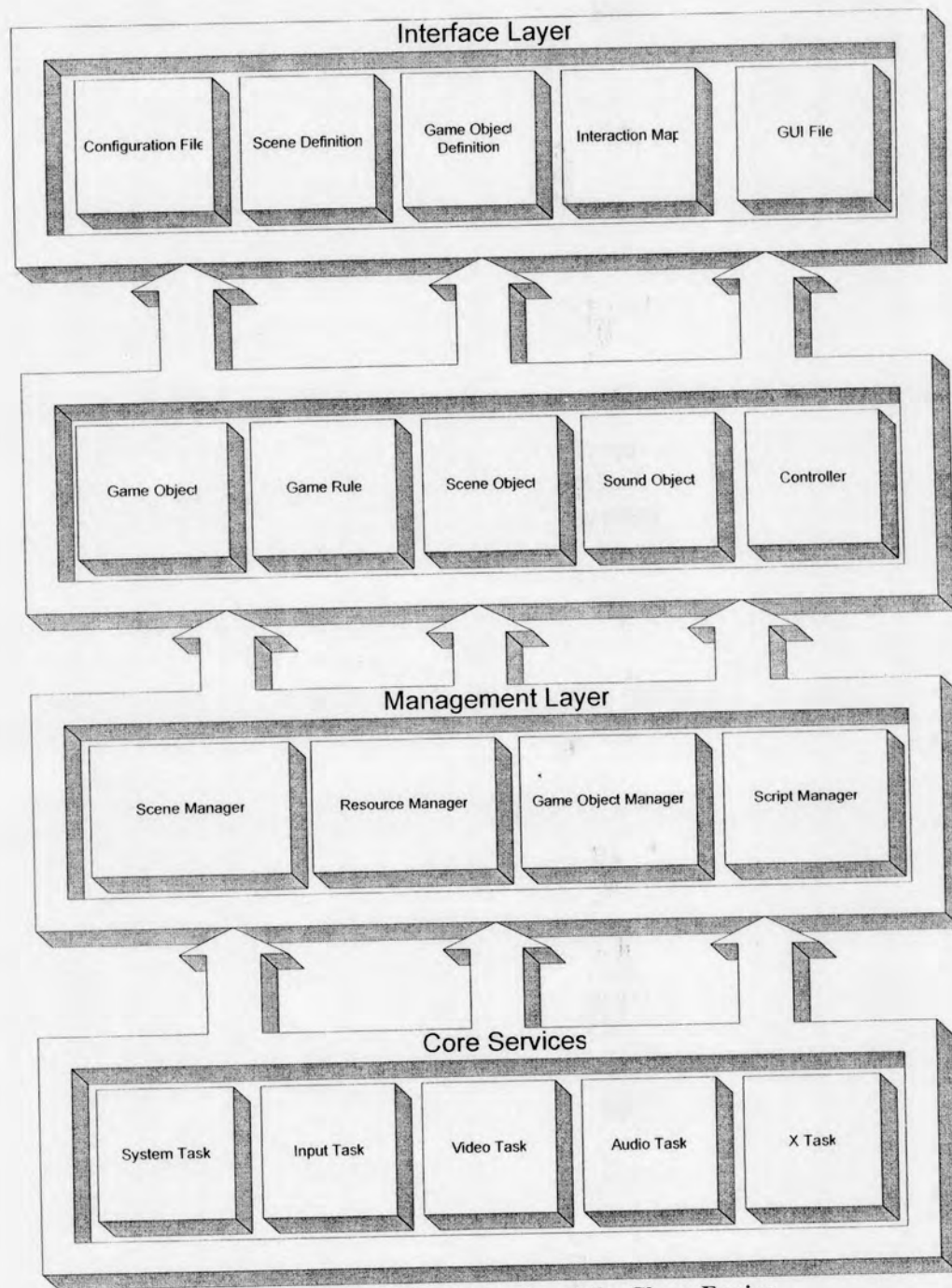


Figure 7 Layered organization of the Chaos Engine

In Figure 2 the engine has been broken into 4 layers of abstraction. The top most layer, the interface layer, serves as an interface between the user and the engine. All of the users interact with this layer and is designed to be as easy as possible to do so. The next layer is the object layer. Most of the data in the XML files will be used to

populate and create objects from this layer. The next layer below that is the management layer. This layer interacts with all layers but is most responsible for abstracting the system API's from the rest of the engine. The core services is the direct interface between the operating system and the game engine.

- **Data Driven Architecture**

The engine will have a data driven architecture. This means that elements that are normally hard coded, namely constants, will be soft coded. This means that the engine will use a simple XML based database to hold all of the constants for the engine. The reason for using XML as apposed to using an SQL based database is that the focus of this engine is to be easy to use. Using an SQL based database means that someone who wants to modify engine constants would have to know how to use an SQL based database. XML is a plain text file format. This means that someone with no experience with XML can still enter the file and alter engine constants.

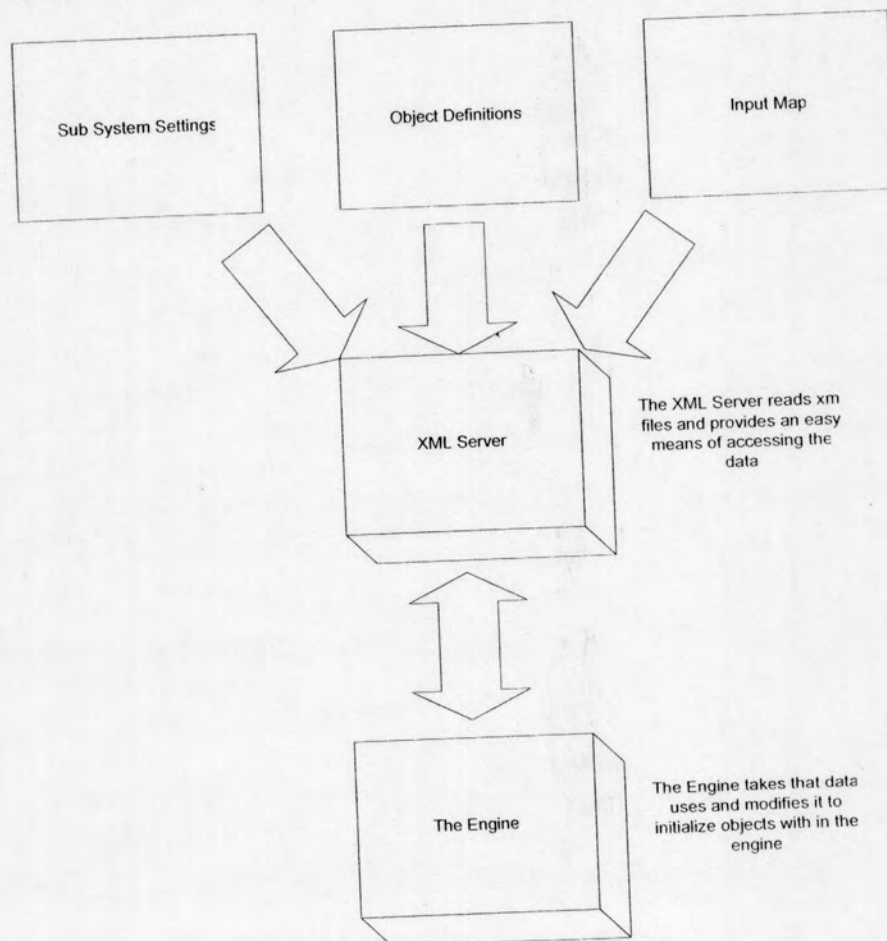


Figure 8. Data flow down into the engine.

- **Interoperability**

The interesting thing about the .NET framework is that code written in one language in .NET can be used in another .NET application seamlessly. For example, the core of the engine will be written C#. If a developer only knows VB.NET or J#, that is no problem. He/she will be able to write engine extensions in their native language. This means that development teams don't have to take time to learn the language of the engine he/she simple can use the language he/she is most comfortable.

- **Integrated Scripting**

The engine will have an integrated scripting engine. Most scripting engines have to have a method of mapping engine functionality to the scripting engine. Another way is to have a developer to write their own script interpreters. Both of these methods can be time intensive, complicated and very tedious. Using Microsoft's VSA Engine frees a developer from having to deal with the complexity of getting a script working. The scripting engine uses native .NET code and though reflection, invokes methods, classes and structures seamlessly. The VSA Engine that I will be using current supports VB script and Jscript which both are widely known and very familiar languages. This will make modifying engine behavior easy.

- **Plug-in System**

The plug in system is a way to further modify the architecture of the engine. This will allow the developer to add features to the engine without having to go into the finished core of the engine. The planned pluggable components are:

- **Custom Game DLL**

This is an extension of the engine that holds custom game logic. If there is not a game type in the templates list you can write your own.

- **Custom GUI Components**

Used to add custom GUI components to the game.

- **Custom Sub- System Task**

There could be a need for a new Sub system task like networking or a new input task that can handle joysticks

- **Custom Command / Script Host**

Hosts are façades for an object with in the engine to be commanded.

- **Custom Game Objects**

There will be a need to have custom game object depending on the game being made. An example would be a particle system object.

- **Standard File Formats**

Too often game developers create their own proprietary file formats and pack files to intentionally make it difficult to use their content. This is understandable because they would like to protect their investment. However, when a developer is using their engine it makes it difficult for someone to get game content into the game. The developer ends up needing special tools and plug-ins to convert their work into a format that the engine will use. The engine will use standard file formats for things like models, sounds and music.

The Tools

Tools are an important part of a game development package. Tools can make interfacing with the engine very easy and thus speeding up the development cycle. The toolset is the preferred method of making a game. The other options are to hard code the game or to use the command line console within the engine to develop a game. Some of the features the tool will have are as follows:

- **Project Templates and Wizards**

There are a finite number of video game types. Instead of having to re-invent the wheel every time a developer wants to make an FPS (First Person Shooter), there will be an FPS template built into the engine. It will be as simple as selecting the template and the tool will take care of creating the project and setting up basic scripts and configuration files for you. Then the developer can spend his/her time working game content and game play.

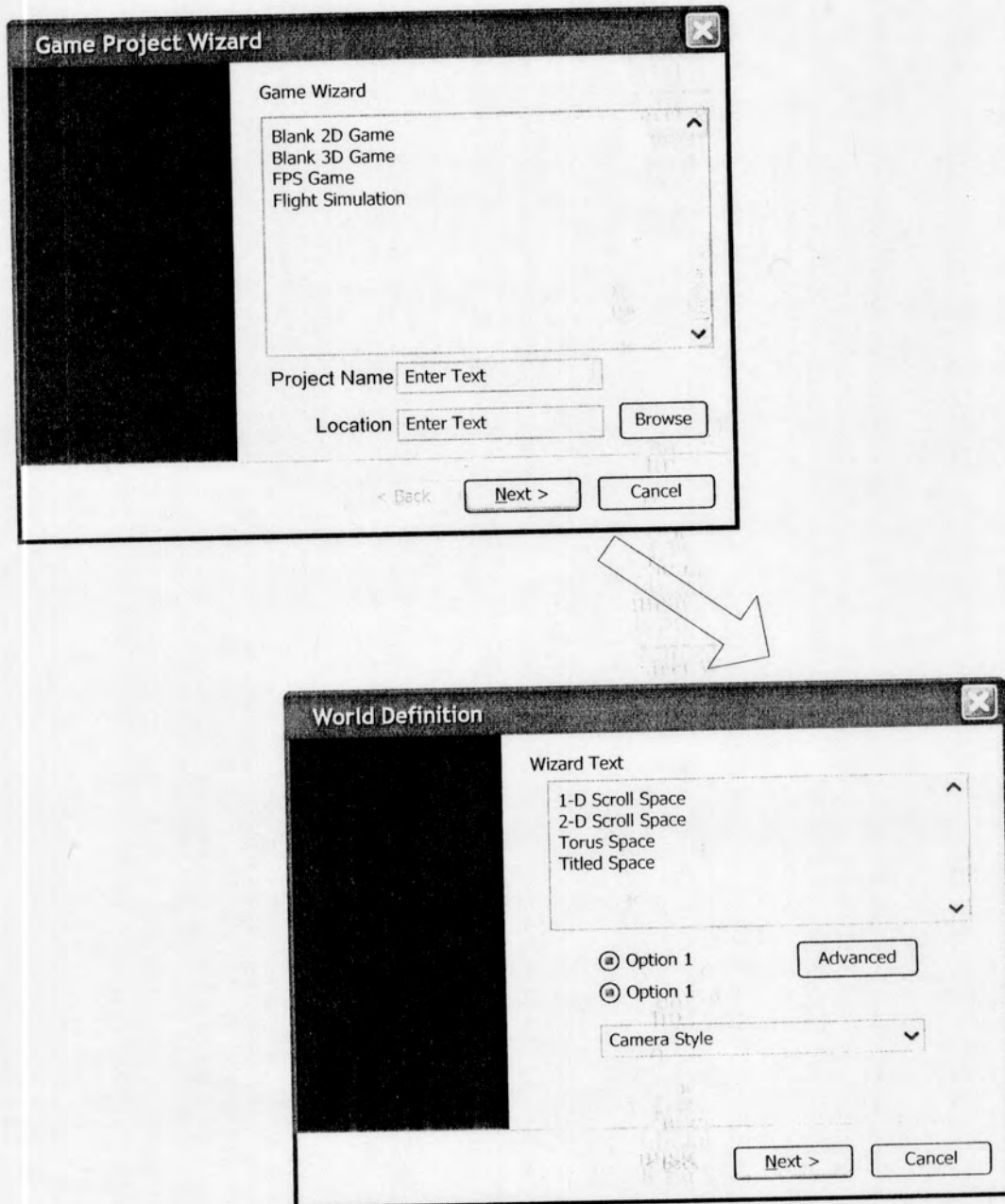


Figure 9 Design of the Game Project Wizard

- **WYSIWIG User Interface**

"What you see is what you get" is a concept that is important in trying to put together multiple multimedia elements. Since the engine is embedded into the tool, the way something looks in the tool will be the way it looks in the engine.

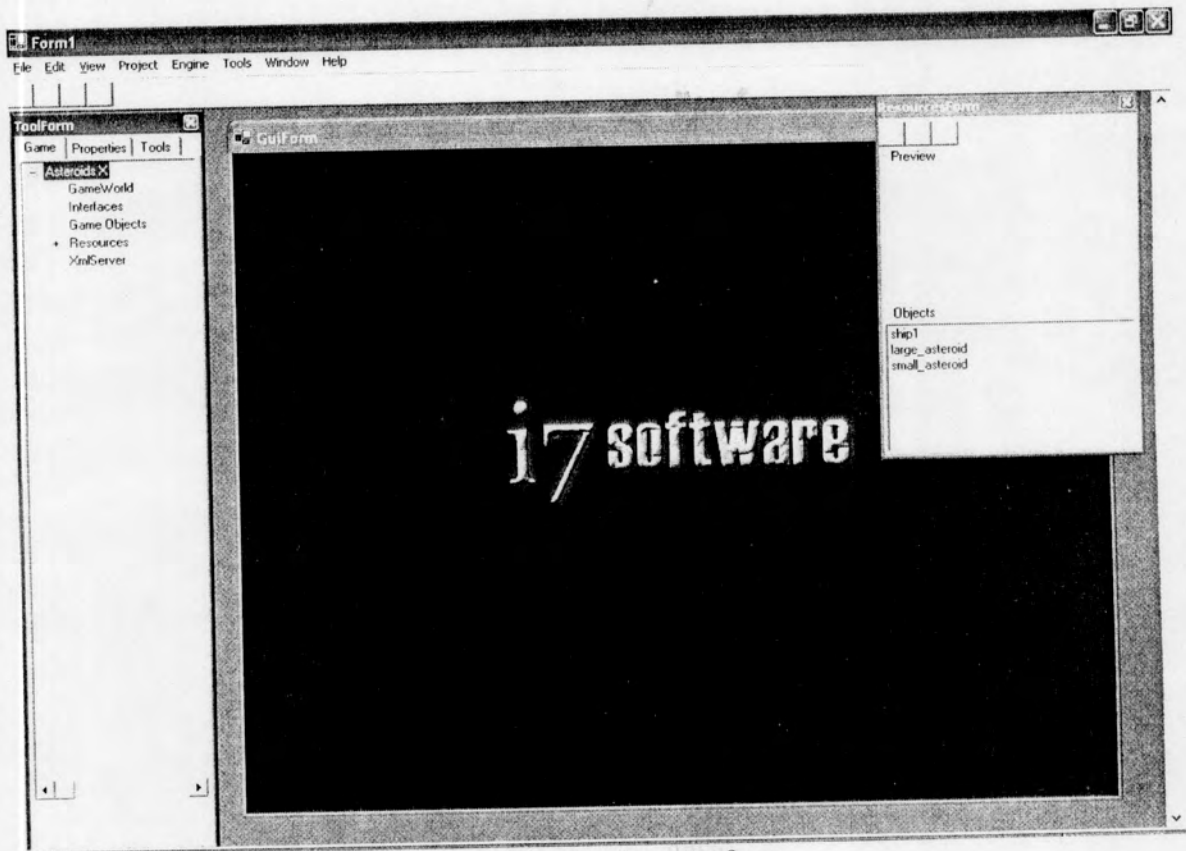


Figure 10 User Interface for game forge

- **Visual Scripting Interface**

Programmers are accustomed to the visual programming. Most programmers know that the visual programming paradigm has drastically sped up application development time.

The Documentation

The documentation of a game engine is very important part in educating a developer in the use of the engine. Most engines lack sufficient documentation but this one will not.

The following material will be included in the documentation of the engine:

- **API Reference**

This is important to any developer that needs to modify the core of the engine to achieve something new. The API reference will outline the intention and functionality of every class. The goal would be to have a documentation structure similar to MSDN library.

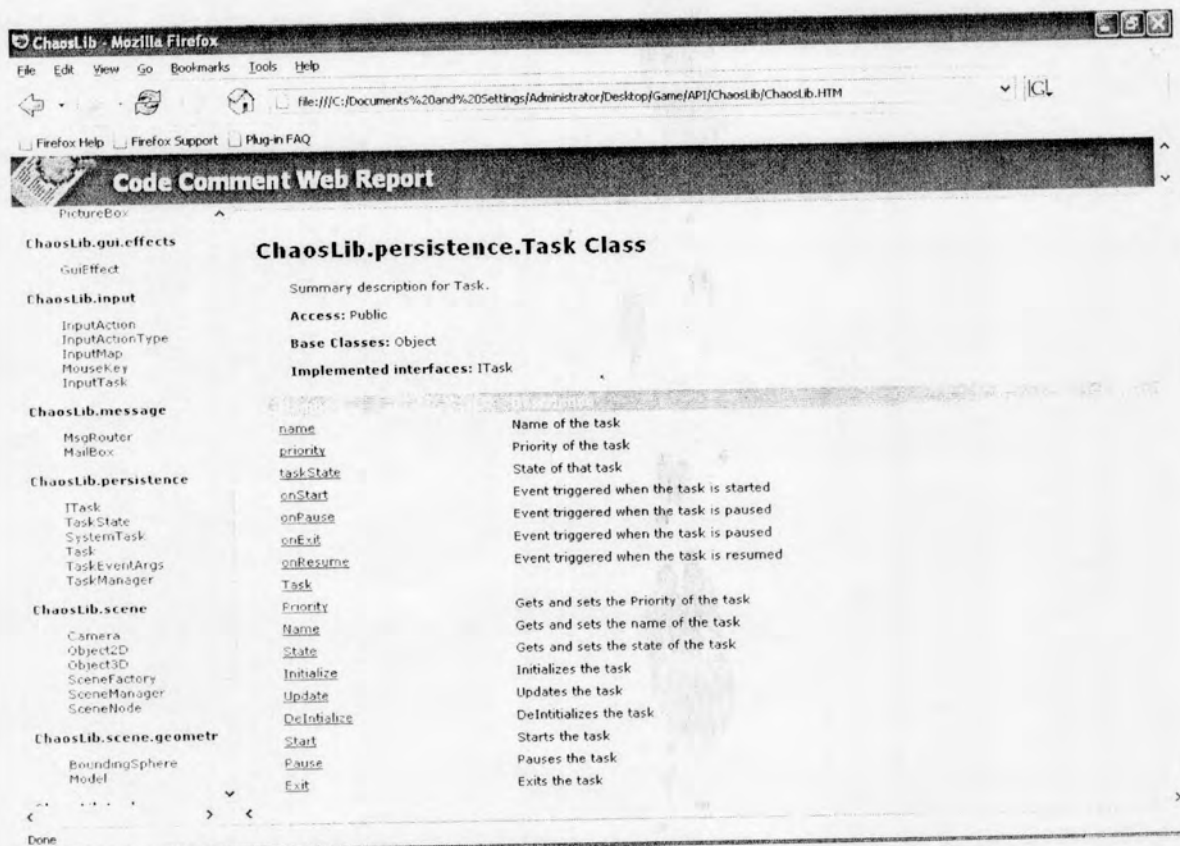


Figure 11 Screenshot of engine API reference

- **Tutorials**

The tutorials will show how to use the tools to make a game project happen. The focus of these will not be reference materials. It will be to educate the developer on the use of the engine. The tutorials will include well defined steps with many

pictures so a developer can not get lost.

Coding Protocols

For this project I will be using the .NET framework and C# to implement the engine.

Concurrently, I will also be using the .NET standard coding conventions and practices.

- **Classes**
The first letters of each word in a class name are capitalized. Names are not to be abbreviated.
- **Properties**
The first letters of each word in a property are capitalized. Names are not to be abbreviated.
- **Member Variables**
Variables names need to all be lower case. The variables must have full un-abbreviated variable names.
- **Input Arguments**
Variables names need to all be lower case. The variables must have full un-abbreviated variable names.
- **Logical Organization**
All related code must be placed into namespaces. Code that is specific to a game must be put into a script or a plug and not in the engine core.
- **Error Handling**
Exception throwing code must be put into a try/catch block. The error code must write debugging information to a log file.

Deliverables

The deliverables for the engine portion of the project have been selected to ease the production of a simple 3D game. These deliverables are mostly the core of the technology.

I. Graphics

- a. **3D Direct X Renderer**
Ability to render objects in 3D and 2D with Direct X
- b. **X File Importing**
Ability to import and render 3D objects in the X file format
- c. **3DS File Conversion**
Ability to convert 3DS Max files to X files
- d. **Height Map Importing**
Ability to import and render height maps

II. Input

- a. **Direct Input Module**
Ability to use direct input to handle input
- b. **Input Maps**
Ability to import input bindings from an xml file
- c. **Dynamic Input Binding**
Ability to bind input to actions during runtime

III. Audio

- a. **Direct Sound Renderer**
Ability to render sound using Direct Sound
- b. **Render wav files**
Ability to play wav files

IV. Control

- a. **XML Configuration Files**
Ability to use constants and state information in XML files
- b. **Z Command Engine**
Ability to execute simple commands against the engine
- c. **VSA Script Engine**
Ability to extend engine logic through VSA scripts
- d. **Plug-in Ports**
Ability to extend engine logic through DLLs

The tools are a very important part of the Chaos Engine. The deliverables listed below have been selected to make using the engine much easier for the non-technical users. Also the documentation of the Chaos engine is very important to the ease of use of the system.

Integrated Tool Set

- I. **Project Wizard**
Ability to setup game project through engine tool
- II. **Engine Console**
Command line prompt tool for manipulating the engine
- III. **Engine Configuration Editor**
Configuration tool to edit major sub system settings
- IV. **Game Object Editor**
Editing tool to create and edit game objects
- V. **Interaction Editor**
Editing tool to create and edit game object interactions

Documentation

- VI. **HTML API Reference**
Engine code reference in html format for easy viewing
- VII. **Easy Tutorials**
Instructions to perform specific task with the engine.

Development

Timeline

There will be three Phases in this project. These phases are dictated by the three Senior Design classes. The First phase will be the inception phase of the project. The second will be the design phase. And the third will be an implementation phase. In figure 1 there is a Gantt Chart of the projected schedule of this project.

I am also planning to use the Unified Process to develop this project. The Unified Process is an iterative software development process that is risk driven. During every iteration I will make a game. The games will start out being simple and become increasingly complex. At the end of the development cycle the engine should have all of the required features. Also this approach to development is designed to route out flaws in the software.

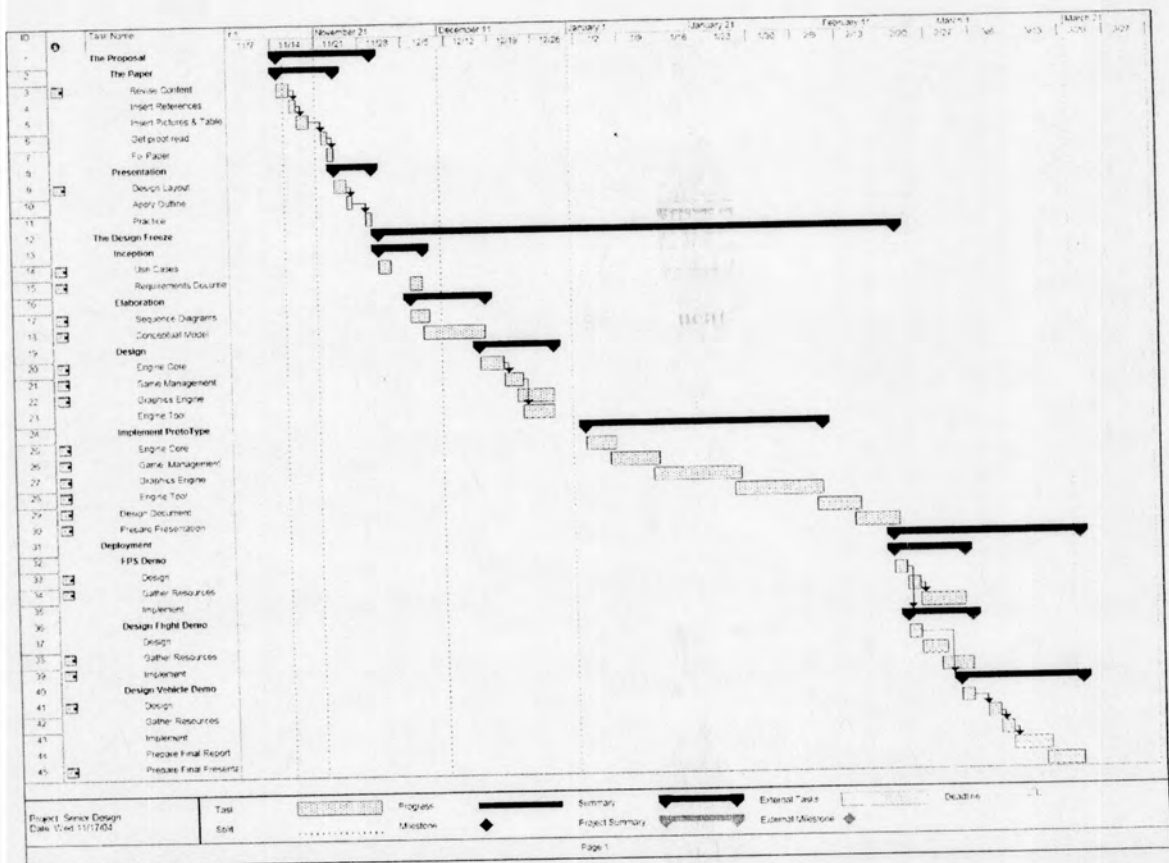


Figure 12 Timeline of the development process

There will be a few major items that will take up most of the time. One is the time to learn the technology that I have decided to use in the engine. I have accumulated experience in most of the technologies that will be using in the engine. The next item in time is designing the engine. This will take some time because I am concerned about the integrity of the project. The next will be the actual implementation time. This will take the most time. This will include the code portion of the development. And the final time item will be rapping up the project which will include testing and documentation.

Budget

The cost of completing this project is large. However, through the schools academic agreement with Microsoft, costs are lower than the retail prices depicted below. The other costs that are associated with the project are resources like game content and books. This constitutes the bulk of the actual spending that will be done of this project.

Functional Software

Software	Developer	Retailer	Retail Price	Actual Price
Visual Studio .NET 2003	Microsoft	Software Media.com	\$519.95	\$13.00
Photoshop 7	Adobe	Bargin Software	597.00	0.00
3D Studio Max 7	Discreet	Discreet	593.95	0.00
Managed Direct X 9	Microsoft	Microsoft	0.00	0.00
Goldwave	Goldwave	Goldwave Inc.	45.00	45.00
		Total	\$1755.90	\$58.00

Figure 13 Software I needed to do this project

Logistical Software

Software	Developer	Retailer	Retail Price	Actual Price
Microsoft Office	Microsoft	Microsoft	\$179.00	\$6.50
Microsoft Project	Microsoft	Microcenter	549.00	6.50
One Note	Microsoft	Microcenter	199.00	0.00
		Total	\$927.00	\$13.00

Figure 14 Software I needed to organize this project

Additional Needs

Item	Producer	Retailer	Retail Price	Actual Price
Dark Matter	Game Creators	Game Creators	\$88.99	\$88.99
Sound Matter	Game Creators	Game Creators	14.95	14.95
		Total	\$103.95	\$103.95

Figure 15 Additional resources

Proof of Design

The subsequent sections discuss the deliverables and the design choices made during implementation. There will also be explanations, screenshots and/or code snippets to support the existence and effectiveness for of the delivered feature.

The Chaos Engine

Direct X Renderer

The rendering for the Chaos Engine is actually simple. But I saw the possibility for more sophisticated rendering techniques. For this I designed the rendering system so that you can actually switch out the renderer to accommodate rendering styles. For instance, the default renderer is very basic. It handles basic things like hardware lighting, hardware fog and the usual transforms associated with the rendering geometry. But there are certain effects like dynamic lighting, cell shading, per pixel lighting or bump mapping. Better graphics programmers than I will be able to add these features in relative ease.

X File Importing

The X file format is the standard 3D model format of the Direct X API and actually is starting to become an industry standard. Because of this I chose to support this format. The other reason for using this file format is that it is easy to incorporate it into any game project.

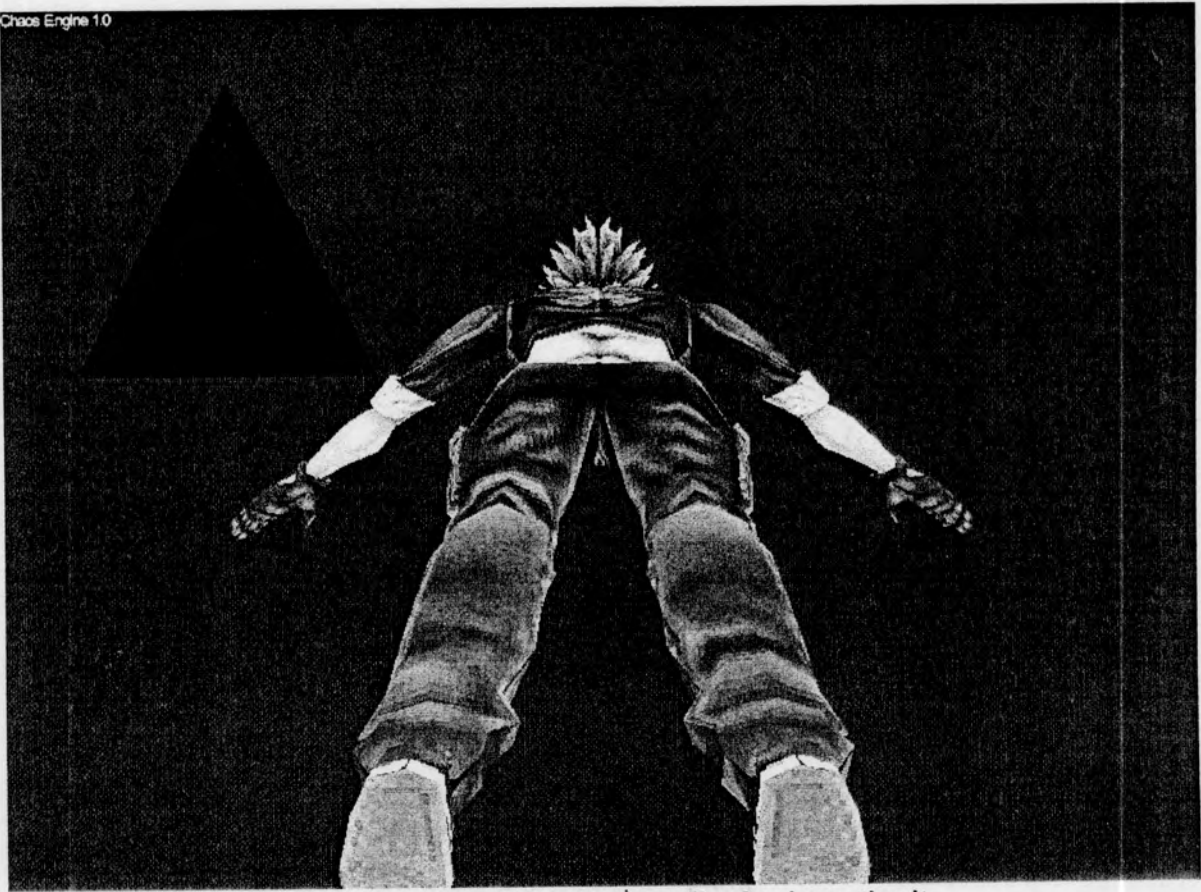


Figure 16 Engine rendering the `tiny.x` file after importing it.

3DS File Conversion to .X files

There are very few tools out there that natively export to the X file format. In fact, I know of none, so it is necessary to have a program to convert the 3D modeling software's native file format to an X file. There are two ways that I have tackled this issue.

Built in X file converter

This is a standalone console application that comes packaged with the Direct X SDK. I built in an option in Game Forge to invoke the converter for you that makes the process much easier.

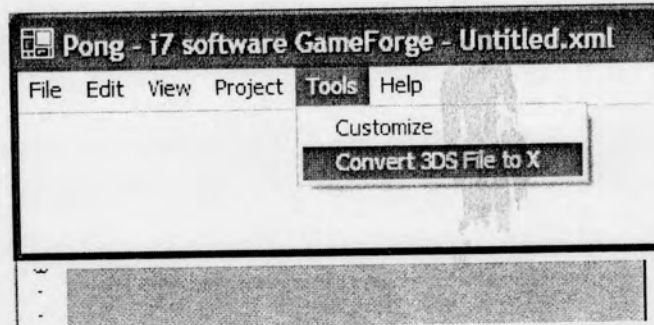


Figure 17 Shows the menu item that triggers a conversion

Height Map Rendering

Height maps are generated from two different textures. The first is a grey scale bitmap. That use the pixel values between white and black to produce the height of the terrain. The second image is the texture use for the ground texture of the terrain.

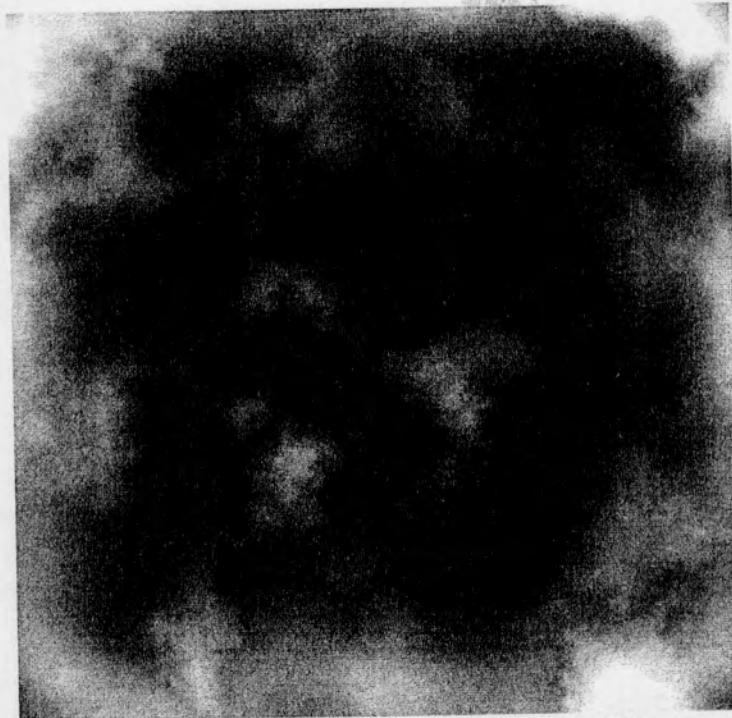


Figure 18 Bitmap that is used to generate the height map

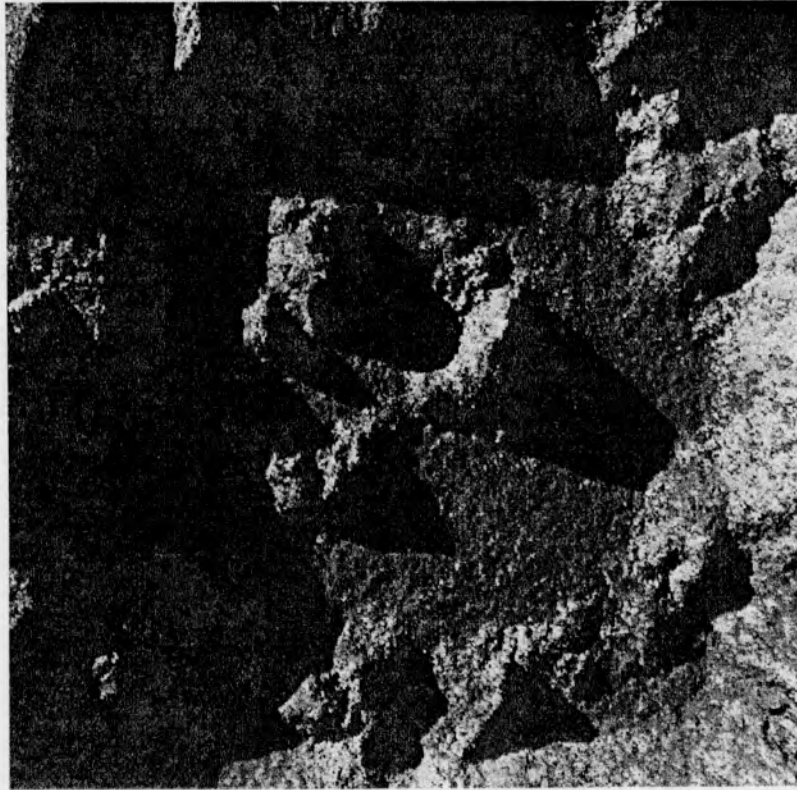


Figure 19 Ground texture for the terrain

Direct Input Module

The input module is a task the runs continuously within the task manager for the game engine. At the initialization phase of the runtime, it must gain control over all of the input devices like the keyboard, mouse or joystick. At the current time the keyboard and mouse are supported input devices, however the Chaos Engine is built to extended through plug ins. This allows for someone to write a new input module plug-in that can handle addition input devices like joystick, web cams or micro phones.

Input Maps

The input map object is a collection of key-action bindings. The engine is designed to have a great number of input maps to accommodate for the different game objects that the user might control, the different GUI Components or any other undiscovered use for it. I designed this to be as flexible as possible because when putting together a video game there can be a great deal of small adjustments to user input that can happen. In the past this was done inside of the code for various reasons, execution speed being one of them. The ease of storing data in xml and .NET's reflective capability allows me to store

input mapping in xml and have execution of the action be extremely fast. Also the format for the input maps is simple. In the code snippet below you can see an example key binding.

```
<binding>
  <Name>Exit</Name>
  <ActionString>Engine.Exit</ActionString>
  <Key>Escape</Key>
  <InputActionFlag>Released</InputActionFlag>
  <MouseKey>Left</MouseKey>
</binding>
```

Figure 20. XML Representation of a simple key binding

The “name” element is used to identify the binding during run time if you want to change it. The “action string” is a simple z command that executes and action. Here script calls can also be made. The “Key” designates the key on the key board. The “InputActionFlag” is used to denote the type of event to respond to. And the “MouseKey” element is an alternate key to use.

Dynamic Input Binding

Dynamic input binding is a useful feature. Within most games the user is given the ability to customizing the game controls to suite their own preferences. I thought this would be a great feature to increase the flexibility of the Chaos Engine.



Figure 21 Screenshot of a dynamic binding

Direct Sound Renderer

The Direct Sound Renderer is a very simple mechanism that updates the attributes of the sounds according to what is happening in the game world. If there is a 3D sound the position of that sound needs to be changed and rendered with different attributes.

Render WAV Files

Using Direct Sound allows for great flexibility and ease when loading, manipulating and rendering wav formatted sounds. Microsoft's Direct Sound is capable of loading a number of popular sound formats but I felt that using the wav file is more commonly used in the game development industry.

XML Configuration Files

A game is nothing more than an extremely interactive and dynamic database. Each element in side of a game is represented by some bit of data that is transformed into the sights, sounds and interactions of a game. With this in mind, data management is the center piece from which the rest of the Chaos Engine rotates.

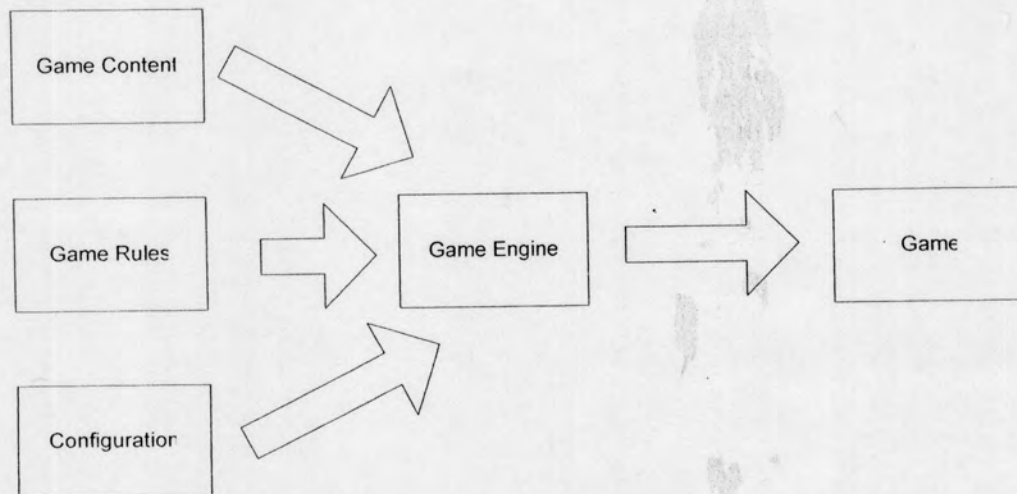


Figure 22 How a game engine works

The Problem

The Chaos Engine is built on a data driven architecture that takes xml files that contain game content, game rules and engine configuration and turns it into a game. The whole point behind having the engine being data driven is that if you change the input files the resulting game will be different with out have to change the underlying architecture of the engine.

Since every function of the engine hinges upon how the game data is handled, it is very important that this game data is:

- Easy to Understand
- Easy to Modify
- Easy to Use

The Solution

The Chaos Engine's data management systems were built with the above goals in mind. The means to the above ends lied with the usage of XML files to store information. XML provides a perfect vehicle for data that is easy to understand, easy to modify and easy to use.

XML is a structured data format that is easy to look at and understand. Since it is in a plain text file format, anyone will be able to open XML files and view the data inside. Also the object oriented structure of XML is better at representing objects within a game than some other structures.

Discussion

As mentioned above XML files are a plain text formatted file. This is important in having the ability to modify these files. Some game engines use files that are in binary formats that make it difficult to open and modify the data within the file. Special tools are needed to read and modify these files.

One of the most important attributes of XML is that it is very easy to use with the .NET Framework. Using the System.Xml and System.Reflection libraries make it easy to manage, use and store data in XML format.

The Object Serializer

When making a query to the XML server there are only two results, which come in the form of a string or an xml nodes. The strings results are the old method of getting the data from the xml server. The old method of transforming the information in the xml server involves a programmer manually converting the string into some data type. The other

method of getting data returns an xml node. The object serializer uses this xml file and an input object to convert the xml node that came from the server directly into the data needed for the host object.

```
ObjectSerializer serializer = new ObjectSerializer(ref doc);  
XmlNode node = serializer.Serialize("Mario", gameobject1);
```

Figure 23 Code Snippet showing how to transform an xml node into a live object

The above code demonstrates how to serialize an object in the game. A programmer would simply instantiate a new ObjectSerializer object and pass in the XmlDocument used to store the data. Then the programmer would simply pass in the name of the object and the object itself. The result is an xml node that can be written out later. The reason that I chose to write my own serialization objects is that the Serialization facilities in the .NET Framework did not allow you to search the xml for the exact object you wanted. You would have to de-serialize the all of the xml to get to some objects.

The XML Server

The XML Server's purpose is to provide a central point of access to game data. It reads in the data from an xml file and provides access to data elements via xpath queries.

```
XmlNode gameobjectInfo = Engine.Server.GetNode("//game/mario");  
serializer.Deserialize(gameobject2, gameobjectInfo);
```

Figure 24 Code snippet of the xml server retrieving an xml node to be converted into an object

The above code is an example how to use the xml server to extract the xml node that contains information about the Mario game object. Then using the serializer the programmer can pass in the object that he wants to initialize and the xml node.

Command Line Interface

The Z in the title of this system stands for Zen. During my first game project I wrote my own scripting language and my own interpreter. It was very impressive considering I had never had any training in this subject. The intent was to allow a user to have a simple

command line interface to the functionality of the engine. But back then I was using plain old c++ that made the command engine difficult to maintain and extend. But now with the .NET Framework and couple years of hard experience the Z Command engine has become as powerful as I need it to be. The current needs to for the system to perform are:

- Needs to have Flexible Syntax
- Needs to be able to make Direct calls to Engine functionality
- Needs to be easy to add new commands
- Needs to be able to store the commands coming from the user
- Needs to be as fast as possible

Using the .NET Framework's string processing and reflection capabilities having the Z Command Engine have all of the above requirements is fairly trivial.

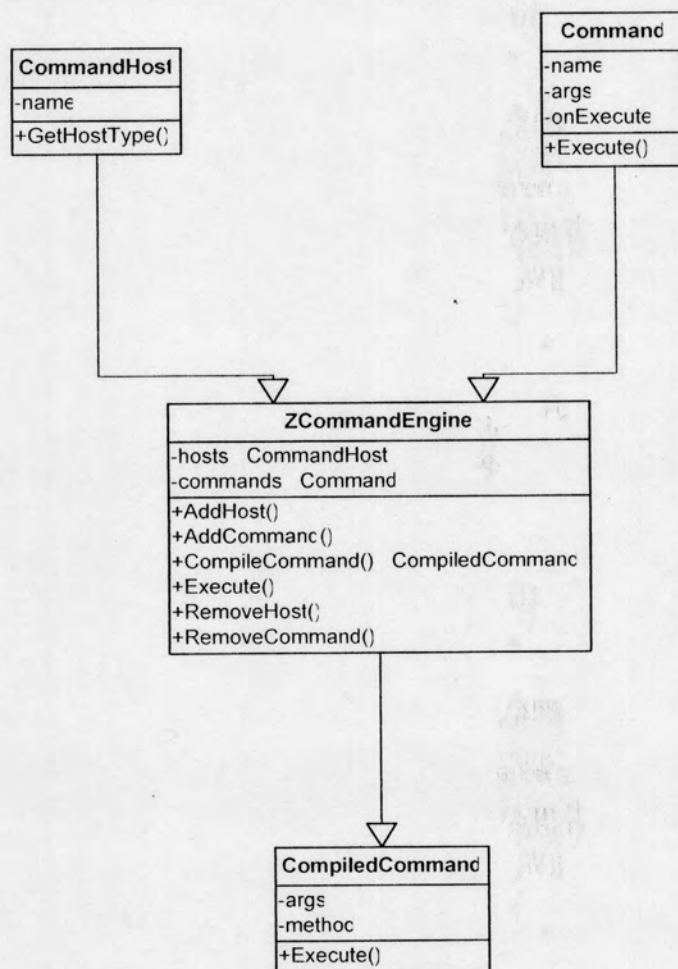


Figure 25 Command pattern used in the Chaos Engine

VSA Scripting Engine

The VSA Engine is Microsoft's method of giving application users the ability to customize the logic of an application. In the Chaos Engine the VSA Script engine is used to script any object that a user wants. Even though any object can be scripted, I only used scriptable for the engine core and every game object.

```
imports System
imports ChaosLib
imports ChaosLib.game.controllers
'imports Microsoft.DirectX.Direct3D;
Module GO
    'Required Method

    public sub onCollision
        dim controller as new FreeFallController()
        this.Controllers.Add("freefall", controller)
        Engine.Audio.Play("explode")
    end sub

End Module
```

Figure 26 Code snippet that adds a free fall controller to the airplane that cause the crash sequence

Plug-in Ports

The code below actually instructs the engine to load the TestBed Plug-in and run the GeometryTest class as an Engine Task using VBScript. The interesting part about this is that it uses the live objects within the application and directly manipulates these objects.

```
imports System

Module myModule
    'Required Method
    public sub Initialize
        'Engine.WriteLine("Initializing Game Dll....")
        dim path as String = "C:\\Documents and Settings\\Owner\\My Documents\\Game2\\TestBed\\bin\\Debug\\TestBed.dll"
        dim className as String = "TestBed.GeometryTest"
        Engine.TaskManager.AddTaskFromDll(path, classname)
        Engine.TaskManager.RunTask("GeometryTest")

    end sub

    public sub onSpawn
        Engine.WriteLine("Spawning Game Object")
    end sub

End Module
```

Game Forge

The Chaos Engine is built to use xml data files to generate a game. Although the XML format makes it easy to store information needed to make a game, it can still be complicated to create a game purely with XML alone. To solve this problem, the engine tool set, Game Forge, simplifies generating the xml files needed to make a game. Game Forge is really a thin graphical shell over the tools that are available in the Chaos Engine. Most things that are done in Game Forge can be performed through the command line interface. This allows users to easily create their own tools to fit the game being developed.

Project Wizard

Often when trying to use some of the game engines I have sampled I have run into a common problem; how to start. Most of the time it is very confusing and difficult to start a new game, so as part of Game Forge I have included an easy to use Project Wizard to assist in setting the game project, leaving the developers to the task of creating the game.

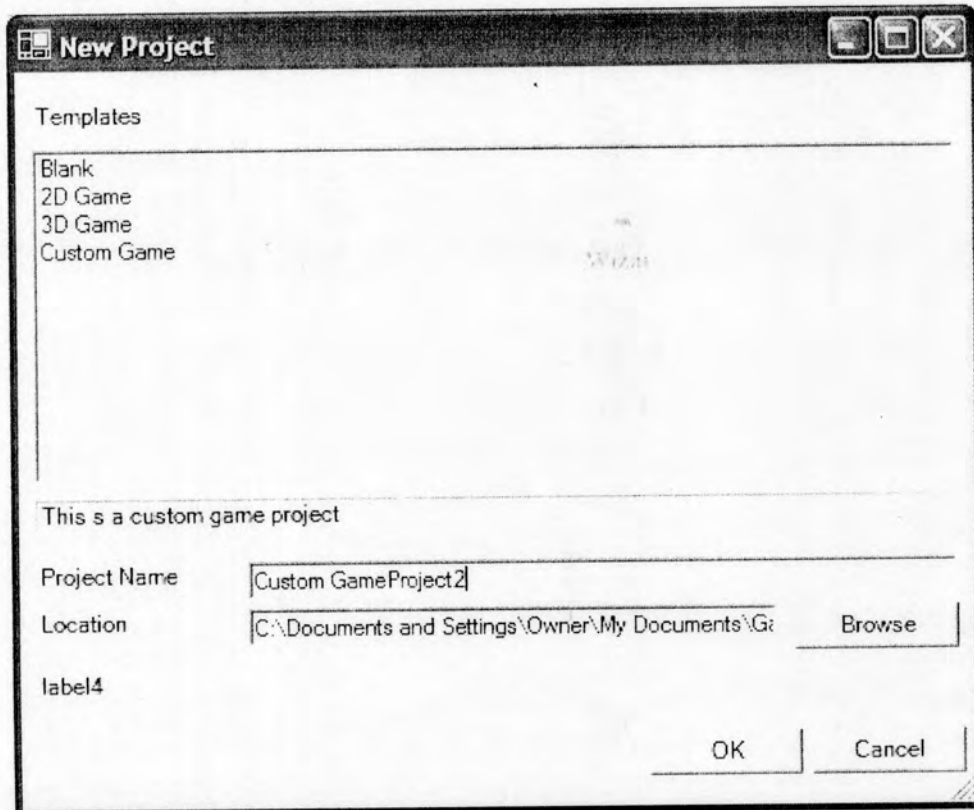


Figure 27 New Project Wizard

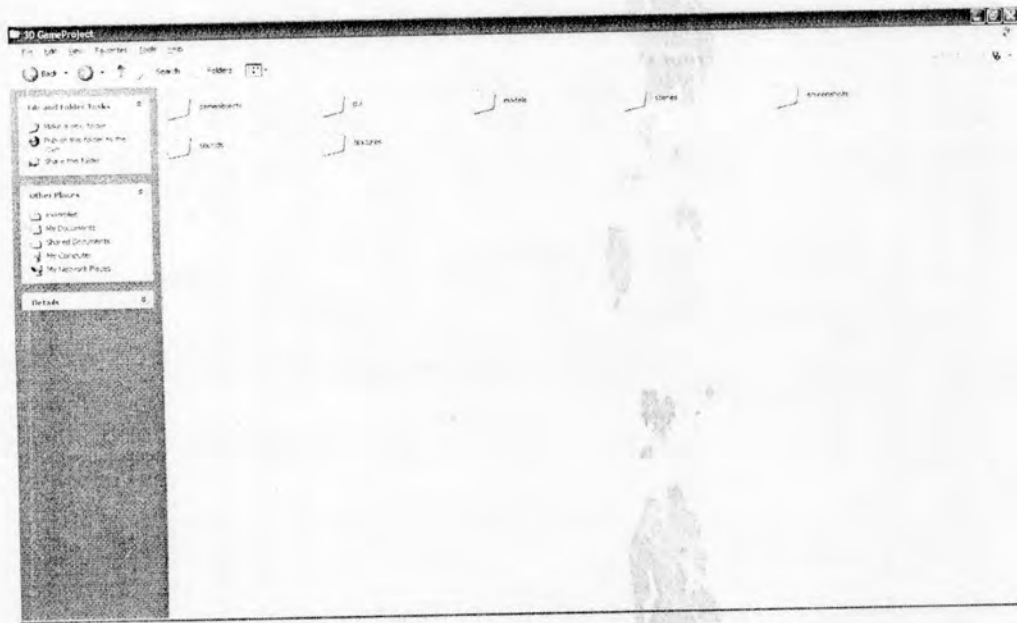


Figure 28 Result folder of creating a new project

Engine Console

The engine console is a tool to issue commands to the engine and to view debugging information during runtime. Without this feature the only other way to determine the cause of the problem is to use the log files.

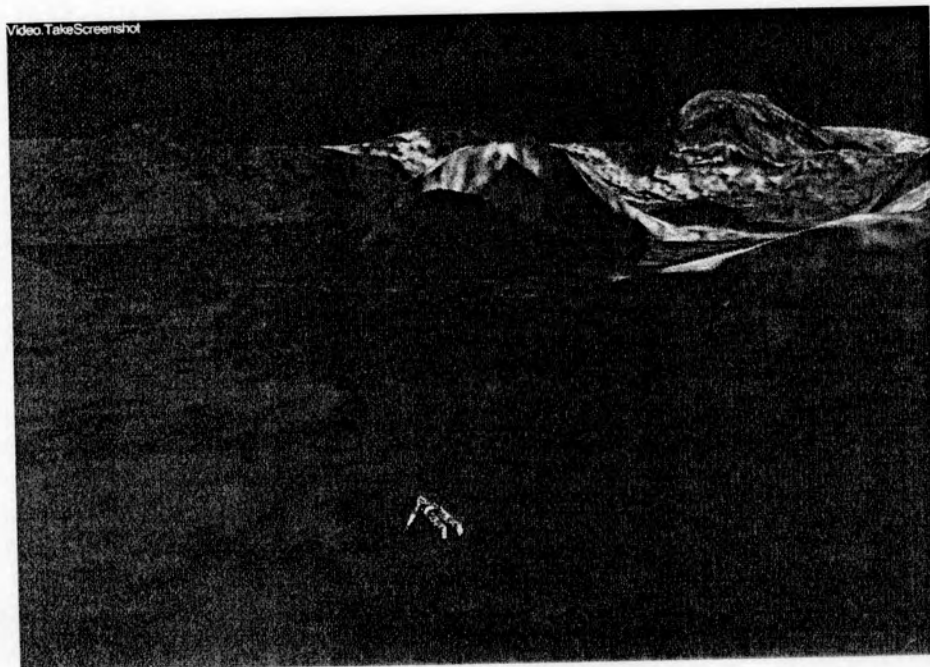


Figure 29 At the top shows a command issued through the console

Engine Configuration

The engine configuration tool is built to easily configure the settings for the engine using a simple to use graphical interface. For the interface design, I took a lot of queues from Visual Studio and the Fly3D engine.

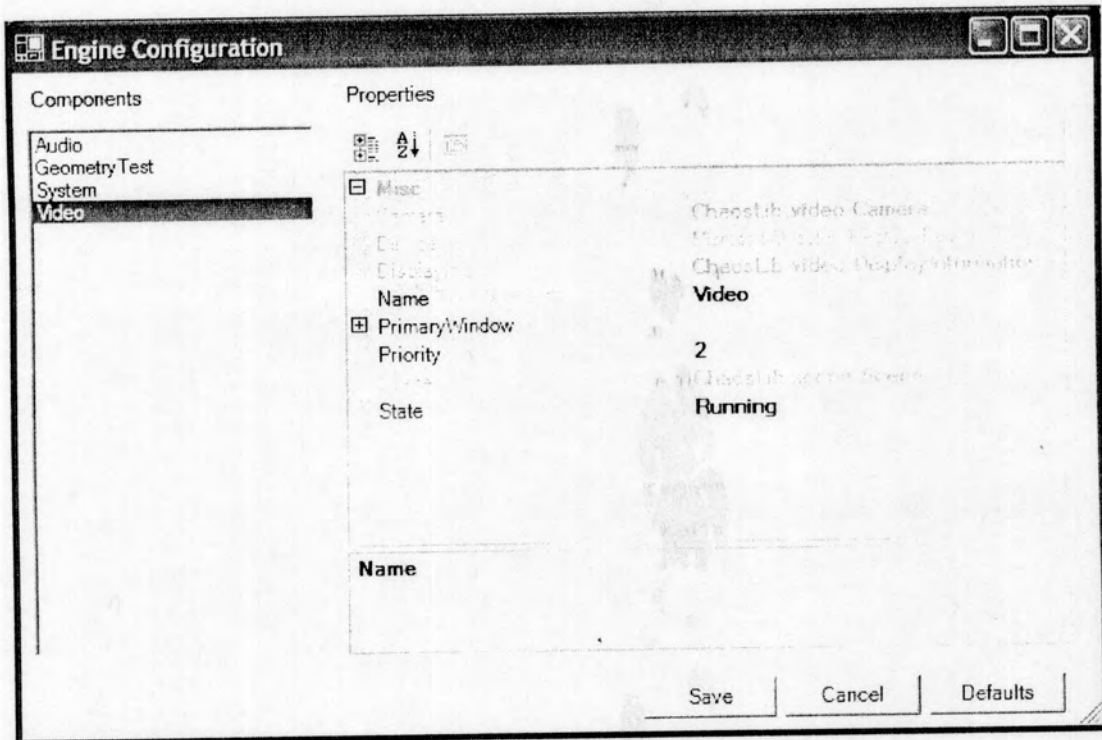


Figure 30 Engine configuration form that allows you to alter sub system settings

This method of manipulating the engine settings turned out to be a lot more useful and user friendly than originally conceived. The original way of doing it was to go into the config.xml file and manipulate the individual data elements. Now it is much easier.

Game Object Editor

The game object editor is a general tool that makes it easier to edit the internal values of the game objects. The top drop down box allows you to select the game object you want to manipulate. Then the properties grid is where the game object properties are display.

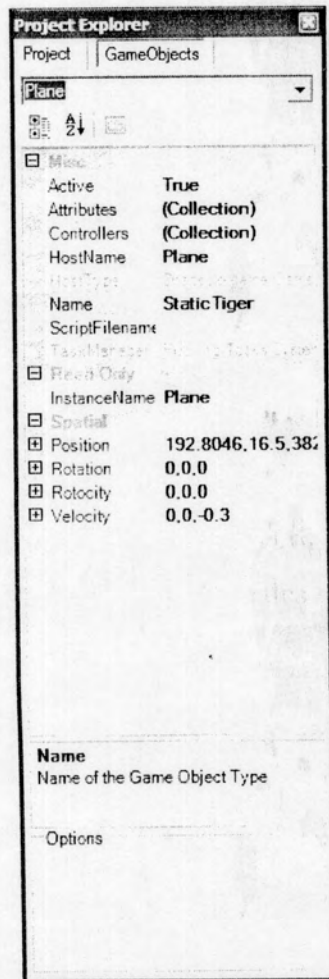


Figure 31 Game Object Properties box

Interaction Editor

The game object interaction editor is a simple tool to map the interactions of the game objects to a response. These interactions are listed in a list box with the collision participants and the function call. The collision response code can be triggered by using a z command or by actually writing script code.

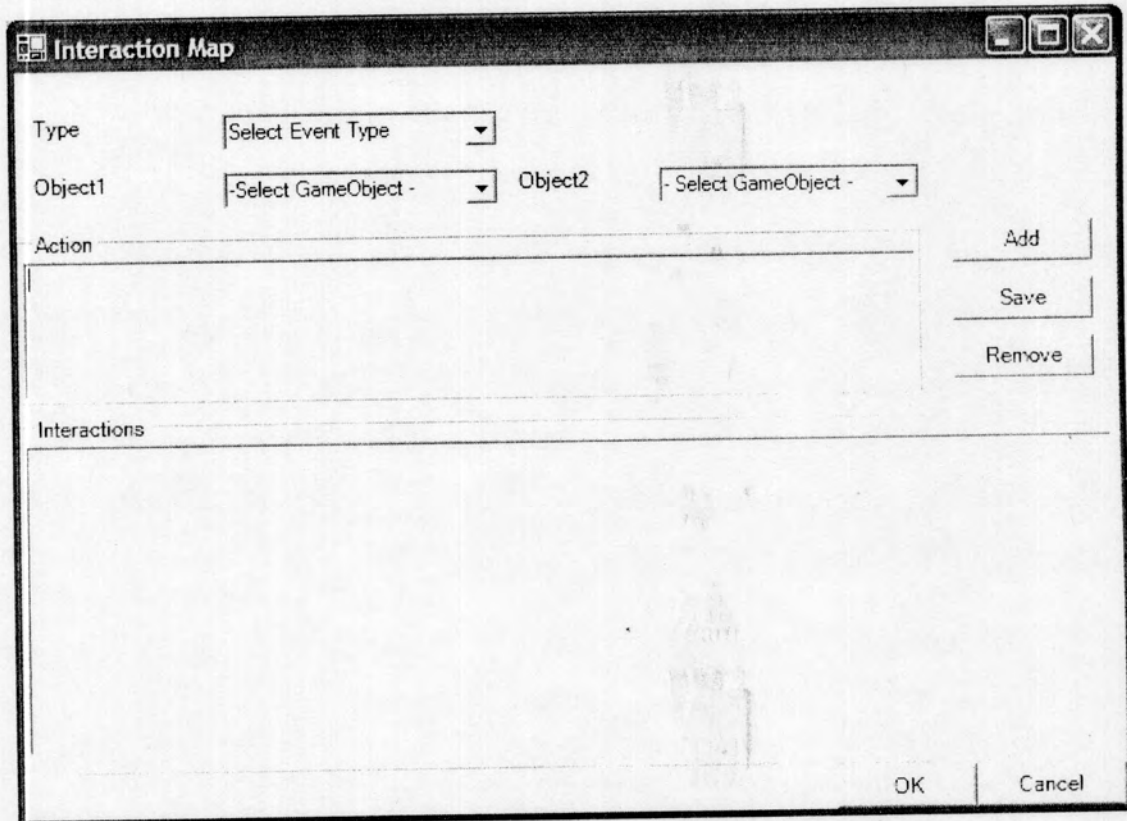


Figure 32 Interaction editor for game object interactions

HTML API Reference

The API Reference is generated from the comments made within the source code. Here a user will be able to see information on any function with in the Chaos Engine. There are only explanations for public functions because the private functions will be unusable to normal users.

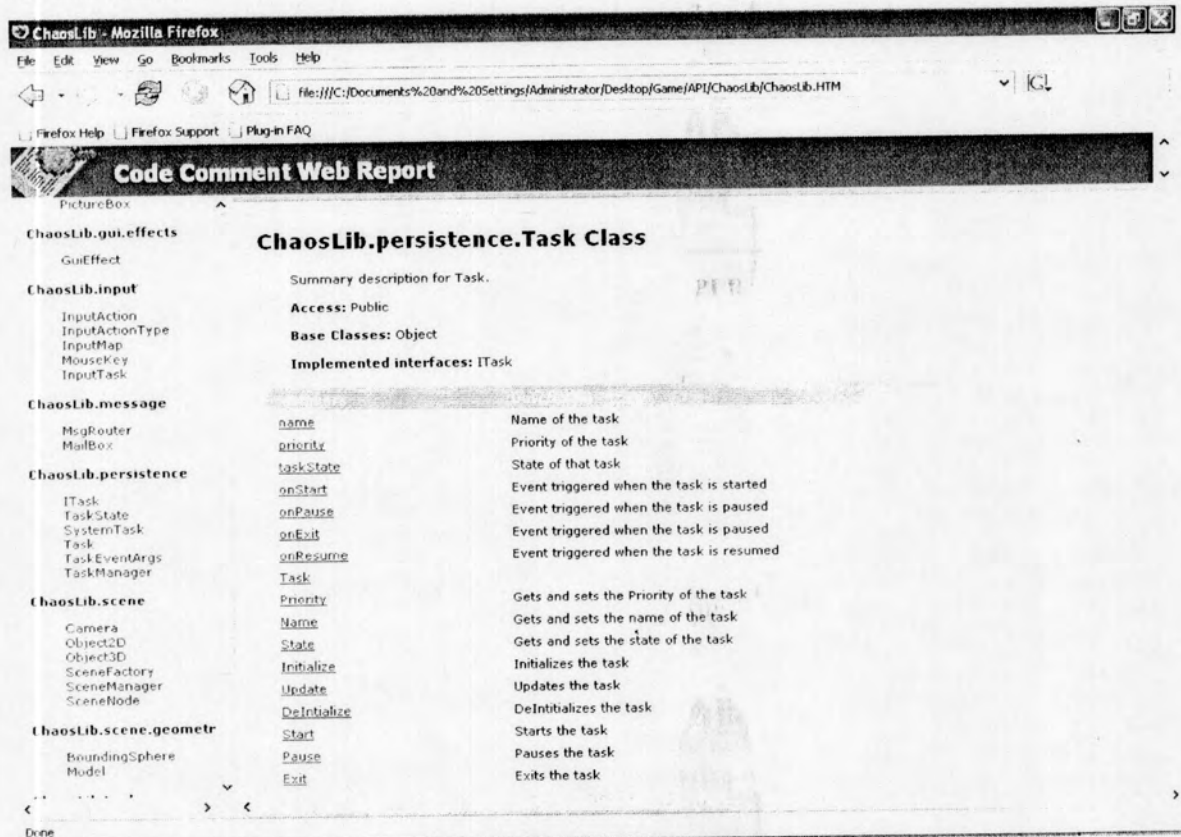


Figure 33 Screenshot of the API Reference

The API Reference will be on the CD in the back of the paper. There is some debate on whether or not I should put it up on the web.

Easy To Use Tutorials

Another important part of the Chaos Engine is the showing on how to use the engine.

There are a few tutorials on how to use the engine:

- **How to Build a Front End**
Explains how to get the engine running.
- **How to display an object**
Explains how to show a game object in the game world.
- **How to make an object move**
Explains how to make a game object move.
- **How to control an object through the keyboard**
Explains how to control an object with the keyboard

These tutorials can be found on the CD attached to the back of the report. Later there will be tutorials on my website. www.i7software.com

Testing Procedure

The testing phase of the project will consist of the types of testing. The first will be developer testing. This phase of testing will be carried out by me, the developer. The second phase will be user testing. This will be carried out by each of the intended users. And the third will be a practical test. This will involve someone building a game using the Chaos Engine.

- **Developer Testing**

This test will be carried out by myself. For this phase of testing, I will run through the list of deliverables to see if every feature works correctly. From this I will compile a list of other engine functions for the users to test as well as the ones that I tested. I will also make up a survey to gather their opinion.

- **User Testing**

Users will be conducting this round of testing. There will be four different types of user testing to match each of the users. I will setup a list of tasks for the users to complete. The survey will be on my website.

- **Practical Testing**

The practical test will be given to a few of the old i7 game development group. I will ask them to make a small game like pong or asteroids. These games should be easy to create using the tools and the documentation. After they finish the task I will ask them questions about how they did it and how difficult it was to do.

- **Fault Testing**

The last phase will be to test for faults in the engine. In this phase of testing other users and I will try to make the engine crash.

There was a number of things that were caught by the testing. The most notable thing discovered was that there were some interface design issues generated by my bottom-top approach

Conclusion

I designed and developed a fully featured 3D game engine using the .NET Framework, Managed Direct X and various other technologies to address the common problems I experienced in game development. The final product is a flexible object-oriented framework and toolset from which to build video games rapidly.

The Chaos engine is built for people that do not have the resources of a professional game developer. It is intended to make simple games cheaply and rapidly. This will open the world of game development to a broader base of people.

Recommendations

While working on this project, I discovered a number of things that will aid anyone trying to tackle such a project. The most important recommendations that I can make is that anyone attempting to do a project like this is one:

Advanced Knowledge of the .NET Framework

This project would have benefited greatly from a greater understanding of the .NET Framework and the constructs built in. In particular, the Reflection library would have changed the entire face of this project if I utilized it earlier.

A Good Command Over Software Design Patterns

One thing that aided me in my project was the knowledge of software design patterns. When designing this project it was very useful to know that I did not have to re-invent the wheel. Everything under the sun has already been done. With knowledge of design patterns the only challenge is identifying when and what pattern to use.

Experience in Game Development

I would recommend that any one doing this have experience writing games instead of game engines. There are number subtle design solutions that seem good, but when writing a game they are actually very cumbersome.

REFERENCES

1. De Goes, John. 3D Programming with C++. The Coriolis Group, Scottsdale, 2000.
2. DeGoes, John. "Getting into Game Development", GameDev.net. 14 Sept 1999. <http://www.gamedev.net/reference/articles/article714.asp>.
3. Deloura, Mark. Game Programming Gems. Charles River Media, Academic Press, San Diego, 2000.
4. Eberly, David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics, San Francisco, 2000.
5. Fine, Richard. "Enginuity", GameDev.net. Oct 2003. <http://www.gamedev.net/reference/programming/features/enginuity1/>.
6. Roche, Don. Tricks of the Windows Game Programming Gurus, Sams, 1999.
7. Rollings, Andrew & Morris, Dave. "Game Architecture and Design" The Coriolis Group, Scottsdale, 2000.
8. Simpson, Jake. "Game Engine Anatomy 101", ExtremeTech. April 2002. <http://www.extremetech.com/article2/0,1558,594,00.asp>
9. Watt, Alan. 3D Games: Real Time Rendering and software technology. ACM Press, New York, 2001.