Origin, Systems Inc.

5918 West Courtyard Drive
Austin, TX 79730-5036
512/434.4263
fax: 512/794.8959
http://www.ea.com/origin.html

Technical Design Document

Loose Cannon team presents:

Privateer 3

November 21, 1997

Third Draft - version 1.30

- **Hugh David**
- **Ala Diaz**
- **Hideki Ikeda**
- **Allen Jackson**
- **Andrew Tjew**

Confidential

Discussion Agenda

Table of Contents

CHAPTER 1 8

Introductions 8
Executive summary 8
3D 8
The Game 8
Networking 8
Game engine 8
Introduction 9
Framework 9
Internal and External Enablers 9
External Enablers - Target Platform and Scaling 9
External Enablers - API's and Libraries 10
Internal Enablers 10
Interface 11

CHAPTER 2 12

Concepts: 3D Engine 12
Internal Enablers - The 3D Engine 12
Overview 12
Major Features and Concerns 12
Hierarchy Object Animation - What It Gives, How Much It Costs 13
Scaling to Different CPU's 13
3D Enablers 13
H3D Support 16
Selecting a Display Driver 16

CHAPTER 3 17

Confidential

Concepts: Networking 17
Game Server: Introduction 17
Obstacles 18
Network Design 18
The communications model 18
Winsock API 19
Asynchronous Server/Client Model 19
Channels 20
Handling the Channels 20
The Game/Network decoupled model 21
Server-Client Relationship 21
Grouping the Clients 22
Packets 22
Packet Optimizations 22
Packet's Content 23
Tokenization 23
Packet Size 24
Synchronization 25
The Server's Game is Gospel 25
Spline Prediction Model 26
Reconciliation Methods 26
Determining Results in Advance 27
Determining a Hit 27
Starting a Network Game 28
Network Game Types 28
How to Find a Game Server 29
Starting a Game Server 30
Account Maintenance 30
Running Privateer 3 on the Internet 31
Overcoming the Internet's Limitations 32
Optimizing the Space Combat Sequences 32
Grouping the Players to Decrease Bandwidth 33
Using AI to Decrease Bandwidth 33
Using a History of Coordinates 33
Dealing with Lost Client Connections 34
Dealing with Cheating 34

CHAPTER 4 35

Concepts: The Game 35
Game Interfaces - On Port 35
Main Port Screen 35
Commodity Exchange 35
Information Exchange - Single Player 35
Information Exchange - Multiplayer 36

Confidential

Hangar 36
Arena - Single Player 36
Arena - Multiplayer 36
Load/Save ATM Machine 37
Personal Access Directory 37
Ship's Log / Player Diary 39
Game Interface - Spaceflight 39
Nav Maps 40
Full Nav Map 40
Game Interface - Alliances 41
Initiate Menu 41
Member Menu 41
Commander Menu 41
Co-President Menu 41
President Menu 42
Planetary Economics 42
Interstellar Trade Routes 42
Planetary Events 43
Stock Market 44
Port Government Influences 44
Combat AI 44
Arena 45
Environment 45
Conditions 45
Multiplayer Arena Only Games 45
Sysop Options 46
Power Ups 46
Mission System 46
Random Missions 46
Player Alliances 47
NPC Alliances 48
NPCs 48
The Ship 48
The Power Plant & Powered Systems 49
The Cockpit & Gauges 49

CHAPTER 5 50

Concepts: Miscellaneous Game engines 50
Game Shell & Engine 50
Threader 50
IOWin 50
IOWinManager and Event Processing 51
Item/Script scheduler 51
Event scheduler 51

Confidential

Script manager 52
Scene (level) manager 52
Memory manager 52
Graphics 52
Music and sound 52
Misc. 53
Language manager 53
Palette manager 53
Collision scheduler 53
Physics 54
Control of the ship: 54
Collision handler 55
Movie Player 55
Object layered above movies 55
Dynamic cinematic 55
Sound System 55
3DSound 56
Input Devices 56
Input Macro Recorder and player 56
User Input Manager 56
DirectInput 56
Input Device Consolidation 56
Key Mapping 57

CHAPTER 6 58

Task list and schedules 58
Schedules 58
Projections for Milestones: 59
Projections for Alpha: 61

CHAPTER 7 63

Appendix 63
Appendix A: Tools 63
Alias Model Converter 63
Alias Model Viewer 63
Texture Bitmap and Palette Viewer 63
Object Hierarchy Viewer 63
Mission / Scenario Builder 63
XMIFF 64
Appendix B: Alias Model conversion process 65
Appendix C: Data handling – data driven game 66
Audio (Music and sound) 66
Fonts 66

Confidential

Mesh....66
Space/Mission....66
Appendix D: CD and Media storage....67
Music and Sound effects....67
Movie....67
DVD....68
Appendix E - XMIFF documentation....69
XMIFF - Extended MIFF compiler....69
Appendix F - Sound document....77
Sound document....77
Overview....77
Music....77
Programming....77
Programming....82
Menus / Interface....82
Gameplay....82
Movies....83
Programming....84
In-flight computer (male and female versions)....84
Mission Specific....84
Space Port....84
Movies....84
Translations....85
Scenes....86
Production formats....86
Production flow....86
Appendix G: 2D and 3D Interface....87
Using Objects....87
Creating an Object....87
Destroying an Object....87
Concrete Objects....87
Template CS_ARRAY<DATATYPE>....89
class CG_DISPLAY....91
class CG_SCREEN....93
class CG_CONTROL....97
class CG_CAMERA:public CG_CONTROL....100
class CG_2DSPRITE....102
class CG_2DSPRITEVERTEX....104
class CG_TEXTUREPALETTE:public CS_ARRAY<CG_TEXTUREPALETTECOLOR>....105

INDEX....106

Confidential

Chapter 1

1

Origin, Systems Inc.

Technical Design Document

Introductions

Executive summary

This document is the road map for the implementation of *Privateer 3*. A framework is chosen and details are attached to this framework. This framework is used as the basis for organization.

The framework is based on the well-established 'divide and conquer' concept with a "top down" and "button up" part. This framework organizes by "concepts" and "enablers". These entities allows task lists and schedules to be determined.

- Concepts are put in the top down part of the framework
- Enablers in the bottom up part.

This framework divides the game into four major parts and identifies each part with a technology. These parts are the 3D graphics engine, AI, network technology and game engine.

3D

The 3D part discusses the implementation details of concepts such as hierarchy animation, dynamic lighting, special effects and collisions. The ability for hierarchy animation is a first in this industry.

The Game

The Game section of the document discusses the tasks that are actually needed to create the game, separate from the engine or network. It includes the game interface, internal game systems, and AI.

Networking

The Network part discusses the implementation details of concepts such as design issues, synchronization, packet management, prediction, and dealing with Internet limitations.

Game engine

The game engine discusses the issues and technologies based on scheduling and handling events, items, threads, sound, external input, media streaming, and many other miscellaneous that would not fit in specific categories of the 3 parts mentioned above.

Confidential

Introduction

The document is about the implementation strategy of *Privateer 3*. It is meant as a working document. The following is a list of what the team wants get out of this document:

- *Framework*. A good framework will allow the team to structure concepts and implementation methods.
- *Task lists and milestone schedules*. These require coordination and planning in order to meet the timelines.
- *Feature discussion*. Difficult and uncertain technologies and implementation are singled out and thoroughly discussed to focus the team on these problems.

Framework

Internal and External Enablers

Enablers can be subdivided into internal and external enablers. External enablers are enablers that are given to the team. Internal enablers must be developed by the team. Most of the work and this document concentrate on developing these internal enablers. For example, TCP/IP is the external enabler and the additional network programming is the internal enabler for multiplayer support.

External Enablers - Target Platform and Scaling

The target platform is an external enabler. Obviously, more powerful hardware realize more sophisticated concepts. For example, without a proper 3D graphics card, most of the graphical elements of *Privateer 3* cannot be realized. External enablers are determined based on market research.

The current requirements are:

- 32 MEG of RAM.
- Pentium CPU.
- 3Dfx card.
- CD ROM drive.
- Windows 95.

The *optional* requirements are:

- Sound card.
- Modem.

In addition, the recommended platform includes:

- 200 MHz. Pentium CPU.
- 28.8 modem or network card for network play.
- 8x speed SCSI CDROM

As an enabler, the CPU comes in various capacity and determines the crunching power available to the game. Accommodation of this variable is mostly done through scaling the graphical details. The next chapters will discussed this problem in details.

Confidential

External Enablers - API's and Libraries

Some enablers are already available in the form of API's and libraries developed by Origin itself or others. The current list of external API's and libraries includes:

- Glide. This is the API for the 3Dfx card.
- The Microsoft DirectX 5 API and Win32 for integration into Windows95.
- Winsock 1.1 for networking.
- Sound libraries, movie players, etc. developed by Origin.

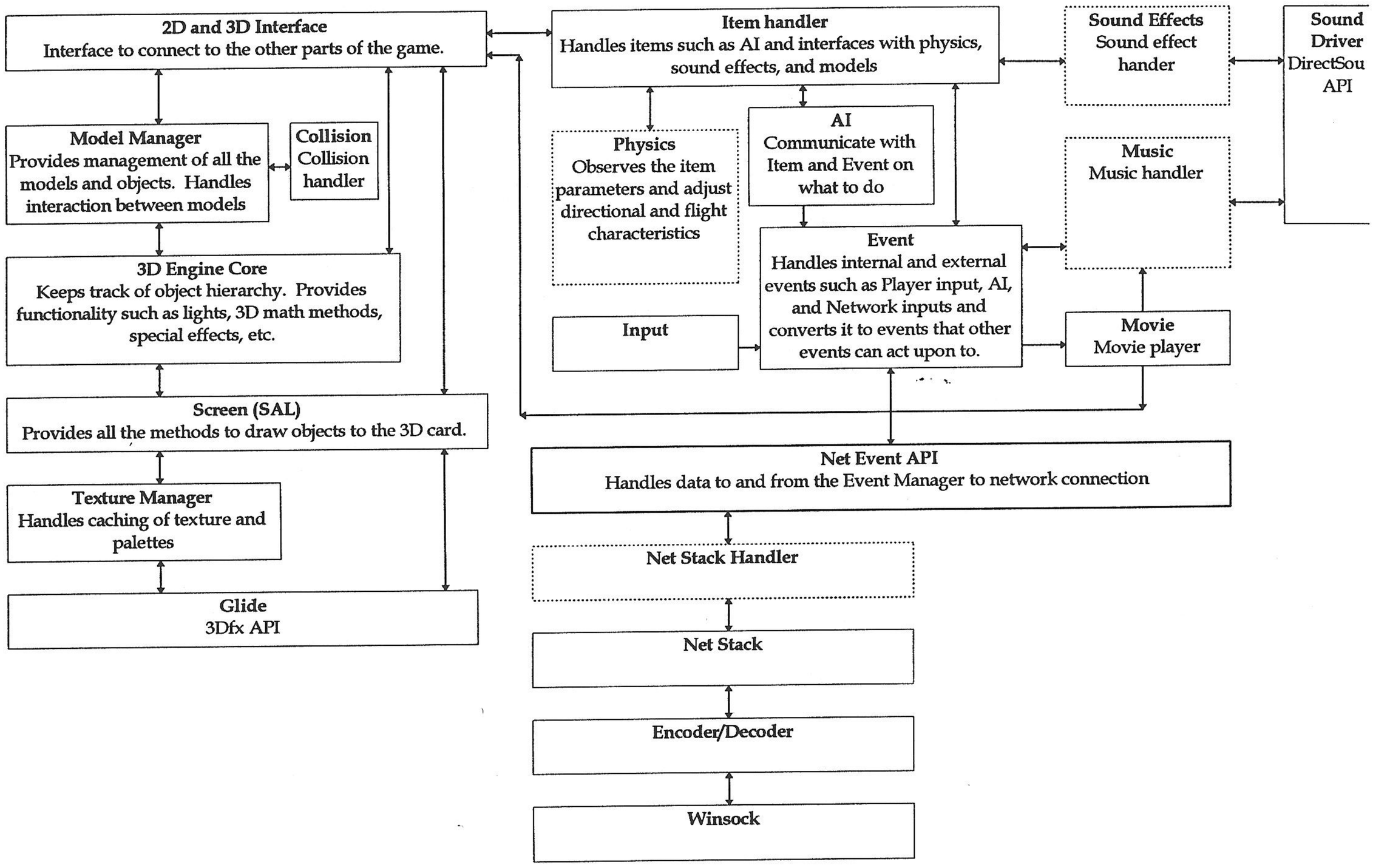
Internal Enablers

Most of the programmers' job involves implementation of internal enablers. The rest of this document is mostly devoted to determining and implementation of the enablers from the concepts in the *Design Document*. Implementation models of major concepts will be discussed in details. For each enabler, a description of what it should do will be stated (i.e. the objectives). Obstacles will be discussed if there are any. Implementation details, time and other resources necessary for each enabler are listed in Appendix in the back. This Appendix allows task lists and time schedules to be generated.

Confidential

Interface

Figure 1 shows the interface of all the basic modules for *Privateer 3* engine.



• Figure 1: Simple Interface API

Note: the dotted box indicates that they are separate threads.

Confidential

Chapter 2

2

Origin, Systems Inc.

Technical Design Document

Concepts: 3D Engine

Internal Enablers - The 3D Engine

Overview

The 3D engine allows the 3D concepts of the game to be realized. The Loose Cannon team is determined to 'push the envelope' in terms of 3D animation, since *Privateer 3* has the go ahead to be a 3D hardware accelerated product only. This chapter is to discuss the enablers of these 3D concepts in details and the associated tools. Time estimates are giving as well. The process to convert Alias models is also discussed.

Major Features and Concerns

The major features and concerns are:

- *Hierarchy Object Animation*. This is a big feature and is discussed in its own section below.
- *Dynamic lighting*. Good lighting brings life to the scene and dynamic lighting enhances it more. The major concern is again the thirst for CPU power. All lights are colored.
- *Hardware independence*. Even though *Privateer 3* is currently 3Dfx only, the engine is designed with hardware independence in mind. All rendering goes through two classes and only these class knows about the 3D hardware and API. Nothing else knows anything about the underlying 3D hardware. Porting to a different 3D card with the similar capability as the 3Dfx should not be more than modifying these two hardware aware classes.
- *Limited Z buffer depth*. Currently, the 3Dfx only supports 16 bit Z buffers. For a game like *Privateer 3* the viewing distance must be big in order to translate the vastness of space to the player. Tricks like object popping or distance fog cannot be used here. A limited Z buffer depth introduces depth aliasing artifacts. To overcome this limitation, the space is divided into three different sectors. This division is based on distance and each sector has it own its own Z buffer that covers this distance. In practice, objects has to be sorted by Z distance and drawing is from the farthest sector to the closest. The objects in the farthest sector are rendered first, the Z buffer is then reset, the objects the middle sector are rendered, the Z buffer is reset again, etc. An additional restriction is that sectors must overlap. The amount of overlap determines the largest possible object. The larger the overlap the larger objects can be but the least effective this approach becomes. The price to pay for better depth is thus object sorting, having to reset the Z buffer multiple times and a maximum object size. Object sorting is quite fast since there are not many objects in the game and they only need to be sorted into sectors. Clearing the Z buffer is quite fast also. Most objects are smaller than the size limitation imposed by this approach. Objects bigger than the maximum can be separated into two smaller ones.

Confidential

Hierarchy Object Animation - What It Gives, How Much It Costs

Hierarchy Object Animation (HOA) is a feature that will set *Privateer 3* apart from other 3D games. HOA is implemented using a standard Directed-Acyclic-Graph (DAG) data structure. Objects are attached as nodes to the DAG. All models in the 3D engine uses this data structure. It is flexible enough to provide the following features:

- *Model Sectioning*. This allows big models to be built out of sections. A section is called an object. Building models out of objects allows independent actions on each object. For example, damage can be applied locally to an object. This gives a more realistic damage model, especially for huge models. It also allows each object to have its own behavior. For example, an object can be animated.
- *Hollow sections*. This allows the player to fly inside big space stations.
- *Hierarchy Animation*. Since objects are organized in a DAG structure. Hierarchy animation is possible. This allows a object to sit on another object. Moving the second object will also affect the first. For example, a gun turret usually have two degrees of freedom, i.e. the gun sits on a chassis and can be elevated. The chassis itself is attached to the ship and can be rotated.
- *Level of Details*. Level of details is necessary to keep polygon counts to a manageable number. Objects with different level of details can be attached to the DAG. The object selected for rendering will be based on the viewing distance.
- *Object Composition and Decomposition*. These can be realized by combining multiple DAG's into one and splitting a DAG into sub DAG's respectively. An example would be the cargo hull of a cargo ship breaking off.
- *Object Culling*. Big space stations are built out of multiple objects. Culling is done on a per object basis, so at close-ups one does not need to render all objects belonging the space station. This speeds up the graphics pipeline.

Models are built using Alias. Alias already uses DAG for data representation. A tool to extract Alias models along with their DAG information is already developed. Building these models do demand extract care and skills from artists. *Appendix B* lists the lessons learned from the Alias model conversion process. This Appendix will be updated regularly and distributed to artists when new lessons are learned.

Scaling to Different CPU's

The most expensive portions in terms of CPU cycles are in the 3D rendering coordinate transformations and light calculations. Therefore scaling for slower CPU's is done by reducing the number of transformations and light calculations. Coordinate transformations can be reduced by using the lesser detailed models more often. The DAG structure can easily accommodate this. The number of lights can be reduced to an absolute minimum on lesser CPU's. This means that the light effects are less dramatic. Using these two techniques appropriately, one should be able to reduce the number of CPU cycles per scene calculation by half.

3D Enablers

The following is a list of enablers to be implemented for the game:

Confidential

- *Texture Caching* - Currently, textures in *Privateer 3* require more memory than available on the 3Dfx card. Caching will work around this limitation. The class for this enabler is 3D card dependent.
- *Screen 3D* - This enabler implements all the 3D rendering methods, the Z buffer work around and requires Glide. The class is 3D card dependent.
- *Screen 2D* - This enabler implements all the 3D rendering methods and requires Glide. The class is 3D card dependent. The class for Screen 3D and Screen 2D are the same. 2D images are actually polygons layers. To be able to manage this, the 2D handler must be able to handle dynamic texture mapping for animation purposes. It will have the capability to handle large polygons in a 2D manner such as menu screens. One advantage of approaching everything in a polygon manner is that we can pull off attractive special effects such as parallaxing layers of 2D map one on top of the others, resulting in a static screen to look more attractive and interesting. Playing a movie is basically, same idea as dynamic texture mapping. Since movies are played in a scene of its own, their is no worries about limitations of texture map space that we have to consider during space flight. There should be enough texture map space available after laying the movie onto the screen, that we can overlay subtitles and 3D objects on top of the movies with ease as well.
- *Lights* - To provide points lights, directional lights and ambient lights in the game.
  - *Point lights* - Point lights are used to simulate light from stars. They are characterized by two parameters, the distance at which the intensity reduces to half and the maximum distance. Point lights are also used to light up the environment when explosions occur.
  - *Directional lights* - Directional lights are used to simulate lights from a far away distance. They are used mainly for indoor objects.
  - *Ambient lights* - Ambient lights are used simulate light that has been scattered many times. They are used to light up indoor objects.
  - *Beam lights* - Beam lights are similar to point lights except they are effective only over some solid angle. They are used to simulate lights from directional beacons.

  Light calculation occurs per vertex and is therefore quite CPU intensive. To deal with the different CPU's, the world has a limited number of point lights that are always there. Additionally, they is a pool of point lights to simulate light from explosions, missiles etc. These lights are allocated using a priority based method. When all lights are allocated and an additional light is required, the oldest light gets reused. The size of this pool is adjustable for different CPU's. This way, the faster the CPU the more lights there can be in a scene before a light suddenly starts to disappear.
- *Object Hierarchy* - Implementation of a Directed-Acyclic-Graph (DAG) data structure and node traversal methods. The DAG is necessary for hierarchy animation.
- *Math* - To provide some basic 3D math functionality such as vector math, trigonometric functions, etc.
- *Camera* - To define the camera in the game. This allows one to define the camera position and orientation.
- *Control* - Control provides objects the ability to move in the three dimensional world.
- *Collision* - Collision gives objects the ability to collide with other objects. The type of collisions depends on the size of the objects. Collisions between small objects are checked using bounding spheres. Collisions with large objects are checked per polygon basis. This gives the ability for a small object (like a fighter) to 'scratch' a big space station.

Confidential

- *Polygonal Explosions* - Small explosions are realized using standard red-yellow explosion bitmaps. Big explosions are realized by red-yellow bitmaps followed by a flashing shockwave explosion.
  Resources: Alias Models
- *Jump Points* - To initiate jump sequence in game.
  Resources: Alias Models
- *Heat and Smoke Damage* - When a ship is hit by bullets, the skin heats up. This simulated by a sequence of bitmaps. When a ship is damaged by missiles, smoke will come up from the impact side after an explosion. Smoke is simulated by transparent bitmaps.
  Resources: Alias Models
- *Space Dust* - Space is empty. Space dust provides the player with a feeling that he is traveling in space. Higher velocity translates into longer 'space dust lines'. Additionally, these 'lines' are also determined by the angular velocity.
- *Particles* - Particles are used in explosions and is a must these days. Examples abound in Star Trek movies.
- *Degree star map* - To provide a background.
  Resources: Alias Models
- *Lens Flares* - Required special effect.
  Resources: Alias Models
- *Black Outs and Red Outs* - These provide visual clues of what the space pilot would see when the G-forces becomes too strong. These effects are visual effects and are implemented as a color overlay.
- *Head Turning and Shaking* - This provides visual clues of what it is to be thrown around in a cockpit.
- *Shock wave* - Supplement to explosions. It adds realism to explosions. Additionally, shock waves do damage to ships.
  Resources: Alias models.
- *Black Holes* - Players are pulled to these black holes.
  Resources: Alias models.
- *Neutron Stars* - These stars can sent out pulses that can destroy player ships.
  Resources: Alias models.
- Missiles and Bullets - Weapons and weapons effects.
- *Nebula* - Players can fly into these giant 'clouds'.
  Resources: Alias models.
- *Landing Sequences* - To display the player's ship on an AVI.
  Resources: AVI
- *Planets* - 3D polygonal planets. Some of them might have rings and atmosphere.
  Resources: Alias models.
- *Model Detail Level* - This provides each model with different detail of level functionality.
  Resources: Alias Models
- *Model Exhausts* - Translucent colored exhausts for models with engines.
  Resources: Alias Models
- *Base Model* - To provide a base class and data structure for all other models. Models are classified as small, medium and big depending on the complexity and work involved.
- *Small Model - (Fighters, Freighters)* - Fighter model
  Resources: Alias Models
- *Medium Model - (Capital ships)* - Fighter model
  Resources: Alias Models

Confidential

- *Big Model - (Space stations)* - Fighter model
  Resources: Alias Models

H3D Support

H3D will be supported by *Privateer 3* to give the player an enhanced 3D experience. Compared to a indoor world game, Privateer 3 lacks the usual visual clues (walls, floor, ceilings) to provide the player with depth information because it's environment is in outer space. H3D provides a direct mean to restore the depth information loss.

Supporting H3D should be smooth from experiences from other developers, the fact that the porting manual is less than five pages and the fact H3D provides excellent support.

Selecting a Display Driver

Besides Glide, *Privateer 3* wants to support as many as 3D card as possible through Direct3D including future 3D cards. Also, Windows 98 has the ability to support multiple monitors and graphics cards, so it is not hard to image that a gamer would have a system with multiple 3D cards from different vendors installed on his machine. The question is then which 3D card to use.

In order to address the above problem, the player is required to select the display driver when *Privateer 3* is installed. The install program will try to pick the fastest display driver as the default. After install, the player still has the option to choose a different display driver in case he replaces or adds another 3D card or just wants to try out the other display drivers.

This scheme is used in a lot of 3D games that supports Direct3D, so the process of selecting a display driver should not be unfamiliar with the player.

Confidential

Chapter 3

3

Origin, Systems Inc.

Technical Design Document

Concepts: Networking

Game Server: Introduction

The networking objective is to make a "death match" or "semi-persistent universe trading" game where the server resides on a customer's machine and players, up to 64, connect over the Internet to these servers via some intermediate dispatching server on MPath. Theses games would play much like the single player version where the player flies from planet to planet trading his goods, upgrading his ship, joining alliances and going on missions. In fact, the engine will not immediately notice a difference between an AI player and a human player.

It is important to note that there are two types of network games that will be supported: "death match" and "semi-persistent". Death match is loosely defined as a game where players meet in an area to fight. There will be no money (other than to make a customizable ship) and there will be no trading. The semi-persistent universe trading will be a trading and dueling game where the player plays like he would in the single player, but some of the other traders will be human. We plan to support up to 64 players for both of these types of games, but they will not impact the other kind of game. So if you died in the death match, your character in the semi-persistent game would still be alive.

In both of these network game types the game will support players dropping into and out of the game at arbitrary times. This will be pretty easy to handle in the death match games, but the semi-persistent universe we want the game to handle these drop-ins and drop-out gracefully for things like clients suddenly losing a connection for whatever reason.

Another way to lessen the chance for clients losing their connections, the minimum target machine for the game server will be a Pentium 200 with 32 megabytes of RAM running Windows 95. This will provide enough horse power to handle 64 people at once in the game, provided it is a dedicated server not rendering the game graphics. Thus, the server program would not be able to run as a client, rather the person running the server would have to run the client and connect to it like any other client.

There is going to be no support for modem head-to-head connections. The game really does not lend itself to this kind of model. The Internet game allows for just a two player game if the players really want this option.

Confidential

Obstacles

There are many issues that Loose Cannon needs to address in order to make this a great network game:

1. Network Design - This section will describe the communication model that we have chosen and how we plan to implement the foundation of this model.
2. Synchronization - This section describes the techniques that will be used to synchronize the clients in the game.
3. Starting a Network Game - This section explains the different types of network games, how a player starts his own game server, how these separate game servers will be found, and the master server's responsibilities.
4. Overcoming the Internet's Limitations - This section will describe the limitations from the Internet and how these limitations will be solved.

Network Design

Objective: To put together a network model that has a server that can handle up to 64 players at once on a P-200 with 32 megabytes of RAM running either Win95 or WinNT. The game server will not display any graphics, thus the CPU time will be placed exclusively on network event management.

Obstacles
We have to choose a networking model that will accommodate the objectives for the project including the ability to scale, speed, and robustness.

The communications model

Objective
The network model for *Privateer 3* will be a client-server model where all of the clients will report to the server and the server will combine, resolve, and return the information to the clients.

Implementation
The client-server communications model is a natural fit for *Privateer 3*. All of the clients will report to the server and the server will combine, resolve , and return the information to the clients. Clients can also be grouped together and partitioned in order to minimized inbound traffic to the clients.

The hybrid model the can also be utilized if the clients can talk with themselves with out any impact on the actual game. For example, voice communication with a certain client could be handled that way. Anything with a game effect must go through the server so that all the other clients can be notified of it.

The model can be extended to a "master server"-"game server"-client model. The master server's responsibilities would be to collect the IP addresses for all of the game servers. The game server would run the game and coordinate game information between clients, as well as report the player statistics to the master server. The client would first connect with the master server in order to get a list of game servers that are available. After selecting the game server they want to play

Confidential

on, the client would only talk with that game server, instead of being routed through the master server.

Winsock API

Objective
The network engine for *Privateer 3* will be written in Winsock 1.1 API code utilizing its Windows specific paradigm. It will open up two sockets initially, one reliable socket and a connectionless socket. All data will be pipelined through these two sockets.

Implementation
The book *Windows Socket Network Programming* describes Window sockets as "… an open interface for network programming under Microsoft Windows" is a specification that describes the Winsock network API. Most of the code is directly portable to Berkley Sockets UNIX machines, while other API calls are not. Winsock code is very portable, fast, and basic to use.

Having full control of the socket communication requires extra coding like queue management, socket connections, and asynchronous communications. Queue management involves handling data packets by putting them into their own stacks until they are read by the application and handle streams by parsing out the messages and putting them into a queue stack.

Socket connections are made either through datagram UDP connectionless sockets, or through a listening connection TCP socket.

Privateer 3 will be using Windows specific asynchronous socket management via the Winsock DLL. Using the WSAAsyncSelect() function an application can set up a socket to send its non-blocking commands to a window handler callback function. In this way *Privateer 3* will be using Win95's built in multitasking engine to handle sockets.

Asynchronous Server/Client Model

Objective
Inside of the Winsock API is a unique operational modes for sockets. A programmer can set up the socket to send a message to a window when a socket is ready to write, read, connect, or close. *Privateer 3* will be using this technology inside of its network design.

Implementation
There are three operational modes for Winsock API programming: blocking, non-blocking, and asynchronous. In blocking mode, a Windows Sockets function returns only when the operation completes. For the most part this is a clean and useful way to open a socket, but there are better ways of using the CPU than waiting for a Winsock function to come back.

Opening a socket in non-blocking mode allows the program to continue after any call to Winsock. It returns a specific error if it did not complete and allows the application to continue. The programmer needs to record the error and react to it later via a polling method usually. This can also take up a lot of CPU time polling a Winsock function until it works, or at least gives it a error that indicates a fatal error.

Confidential

Winsock asynchronous operational mode offers the non-blocking mode's advantage of continuing after a Winsock function call, but also sends further socket messages via Winsock DLL to the application. Basically you set up a window handle to take all the Winsock application notification messages and react to them like another Window's WM_ command. This fits in real well inside the Windows paradigm, since the network code will operate just like another window. Using the Winsock asynchronous operational mode in will make a top performing Windows game that can handle up to 64 players at once on a Win95/NT platform.

Channels

Objective
There will be a line of communication between the client and server for each type of data that needs to be transferred. Further, this Channel class will either be TCP or UDP, according to the type of data.

Implementation
To separate different priorities and data types we will open a channel for each socket and name it. Each channel can be either use a TCP or UDP socket, and for each socket an invisible window will be created in order to handle it's own Winsock notification messages. When we send or receive information from a socket, we will already know what "kind" of information is being sent of received. For example, if I start up a channel called TRADE then when I send a trade packet it will be funneled through this channel and will come out the same channel when it hits the game on the other side.

We can have a maximum number of 256 channels opened at once. This is in no way connected to the number of virtual areas the players can go or the number of player this scheme can support, it only limits the different type of data that can be sent along this networking scheme.

Handling the Channels

Objective
There are two basic operations in handling these channels. One is to react to the Window messages sent by the Winsock DLL, and the other is polling each channel to check if there is data packets waiting.

Implementation
Reacting to the Windows messages is handled inside the channel's WndProc without the game knowing it. As the sockets send packets to one another, the channel's FIFO queue grows. These packets stay in the queue until the channel handler "pops" the packets from the queue. In much the same way, when a packet is pushed onto a channel, it tries to send it immediately or gets to it when the socket is ready.

When polling the network, each channel will be queried to see if it has any packets ready, then each message will be popped from the queue and dealt with in the game. This polling will be done with inside of the Main Loop as part of the input phase along with mouse, joystick, and other devices. In fact, the network will be seen as just another IO device attached to the game.

Confidential

The reason it is not threaded is because the game may not be ready for its input. For example, there is no reason for the game to resolve input while it is trying to play a sound or rending a 3D scene. In fact, Windows is already doing its multitasking by using the asynchronous operation method for the sockets.

The Game/Network decoupled model

Objective
The network will be seen as another IO device. When the game goes through all of its input devices, the network will be interpreted as a virtual device. This abstracts us from the network and allows us to build the engine in such a way that it can receive input from AI, the joystick, the mouse, or the network in the same way.

Implementation
To implement this kind of framework, there will be an intermediate class between the game and the network call NetGame. It will be in charge of the interface between the network packets and the game. For incoming packets from the network, NetGame will reinterpret these into game inputs. When the game "pushes" results for its "output" device, NetGame will turn those into packets and send them to the appropriate IP addresses.

NetGame would act as an input device, allowing an AI to jump in the player's seat for a moment and "drive" around while the player is not around. Further, it allows us to use AI for all the other player slots that are not filled at the moment, and later we can replace the AI ship with a player's ship to keep the game always going.

The advantages of this model are synchronization, being able to replace the network engine, and dropping in the single player game. This model lends itself well in designing a single and multiplayer game at the same time, since all we have to do in the single player action is drop in AI for the input and a story.

It is important to recognize that the network code knows nothing of the game. Being completely independent means that a tad bit of overhead must be taken, namely C++ abstraction class layers, in order to efficiently design a game with such a wide scope.

Server-Client Relationship

Objective
The client is responsible for connecting to the server, sending input data to the server, and interpreting the resultant absolute data from the server. The server is responsible for communicating between clients, processing the AI and universe, keeping an absolute record of the universe, and reporting the universe resultant data to the clients. Thus, the server will synchronize the universe and the client will be transferred that universe.

Implementation
When the server is started, it will load all game state saved data and wait for clients to connect. After a client connects, the server registers the player into the game and send out the clients statistics to all clients so that they can register the new client.

Confidential

Since all the clients communicate through the server, the server needs to let the client interact with each other by accepting the input data from each client and applying that input to the server's version of the game. The input is processed like any other input on the server side. Thus, AI and clients are treated the same.

The server is also the central location for all the clients. For all the clients to see the same thing, the server will be the only node accepting any delta information. After the server has processed all the inputs of all the clients and AI, it sends out relevant information to the clients in absolutes. For example, if a client buy 33 Grain from Maxxor, the client notifies the server of this request. The client does not assume it has purchased this grain, rather it waits for the response from the server. The server processes all of the input, in the order it was received, and finds out if there is enough grain and if the client can pay for it. If everything is okay, the server makes the purchase, modifies the client's credits, and sends out the player's total grain and total credits. Thus, the client will never get a packet saying to "subtract 33 grain", rather it will receive the message of what to change it's grain value to.

Grouping the Clients

Objective
Privateer 3 will cluster the clients into hierarchical groups in order to decrease the amount of inbound traffic to clients, and outbound traffic from the server. The NetGame class will send events to only those clients that need to know it, rather than sending every event to every client which would slow down the game.

Implementation
For the trading game NetGame will divide up the universe into Quads, Systems, and Navigation Points. A client will be a member of each of these groups. When an event occurs, depending on the type of event, NetGame will route this to only the appropriate client groups.

Since the network code knows nothing of the game, network classification code will be data driven to make these groups. Thus we could have the clients grouped into a red team and a blue team, or into location groups like in the trading game.

When directing packets, the NetGame can choose to send to a specific group that will know its members, which will then only send the packet to those members. Further, if the server needs to send a packet to all clients in the game, it could send it to a specific part of the hierarchy which will then send it to every client in its lower branches. For example, in the trading game if an event needs to be transmitted to all of the clients the message could be transmitted to the "Universe" group which all players are apart of.

Packets

Packet Optimizations

Objective
One way of keeping network data communications quick and clean is optimizing the packet of data that will be sent. In *Privateer 3* we plan to scrutinize the packet size, contents, and categorization in order to optimize this part of the application's transportation layer.

Confidential

Implementation
Packet sizes are very important with a TCP socket because sending one of these packets requires a confirmation each time. Thus, sending many of these packets instead of combining these packets into more modest sized packets could cause the game to bog down.

On the other hand, an UDP socket can become ineffective if its packets are too large since they can be fragmented or even dropped if they are too large. Here the size does not matter as much as the contents. *Privateer 3* will trim all packets down to "what it needs".

We plan to tokenize the packets in such a way to let the game already know something about the packet before it tries to process it. This allows the game to process the packet in an efficient manner because it does not have to figure out the packet's purpose by its data.

Lastly, each packet has its own structure that describes the type of information inside of it. One of the benefits of this will be knowing how large the packet should be by using sizeof() on the structure, and obtaining information from the packet will be as quick as referring to its data field pointer.

Packet's Content

Objective
We have created a generic packet structure that can send any kind of information, and lends itself to quick interpretation from within either the client or server.

Implementation
The packet consists of a header and then the raw data. The first two bytes of the header describe a short integer which is a length of the packet in bytes. The next two bytes describe a short integer which is the token, which is the "way" the game will handle the input. The next four bytes describe the player's id (which happens to be his IP address). The last byte in the header is the channel that the packet should be pushed on. The last part of the packet is data which is equal to the number of bytes in "length - headerSize". Each packet will have a structure made for it inside of the Packet class. When the game changes the data from raw byte data to structured data it will be able to use the sizeof() method to determine if the data is out of sync, or if it has been corrupted. In fact the data pointer could be simply casted to a structure pointer of its type.

Data structure(s)

```
class Packet
{
        Packet() { length = 0; token = 0; id = 0; channel = 0; data = 0; }
        unsigned short length;
        unsigned short token;
        unsigned long  id;
        unsigned char  channel;
        char* data;                     // raw buffer for the packet information
};
```

Tokenization

Objective

Confidential

Each packet will have a game specific token use built into it. This token will be used inside of the game packet's interpretation switch statement. Tokenization allows the application to already have some information about the data that it receives.

Implementation
The token will be second short integer in the packet, enabling the game to use 32,000 tokens per channel. Each token will hold some kind of game specific information to describe how the data is to be used. For example, the token for "buy from planet" could be 128 and the data might be "33 grain", then the packet would be sent through the TRADE channel. The packet's data would be handled in the switch statement in the NetGame's trade packet parsing routine.

When the server is done resolving the input, it will then construct a response packet using the appropriate client token to send the resultant data back to the client.

This method is easy to maintain, upgrade, and can help debug the code. For example it can catch "trash" packets that are either sent or received.

Packet Size

Objective
The packet's size and contents are important in that we want to maximize the network's capability of transferring data. In doing so, we need to be aware of the type of data we want to send, how often it needs to be sent, and pairing down the data to a size where it is useful but not wasteful.

Implementation
Below is a chart of the information that the game server needs to manage. Type is the packet type, priority is ranked from Low to Medium to High, size is in bytes. This is not a complete list of packet types that can be sent in the game, but it serves as a starting block.

Type	Priority	Who	Size	Description
Positional	High	Server	48	This holds the present and two past x, y, z coordinates for the ship plus its orientation.
System Operations	High	Both	Varies	These are non-interactive game actions like logging in or out of a game, receiving a timeout message, and so on.
Communications	Medium	Both	Varies	Usually text information, but there should be a provision for stored text data for NPC's and the like
Space Events	Medium	Server	4	Like explosions and jump gates
Speech	Medium	Client	Varies	Compressed audio that a player spoke can be sent
Navigation Data	Low	Client	2	Tells where the player is going to jump to next
Player Information	Low	Server	4000	Tells the server what assets the player has, this is mostly to be updated between the master and game servers
Trading	Low	Client	8	Client's request to purchase something
Commodities	Low	Server	12	The server sends this information out when the player trades. It will show the result.

For the most part, the packet size is not going to be a problem. Assuming each packet is less than 100 bytes, a great deal of small packets can be sent. The only problem with a large amount of small packets would be on a TCP channel. If it had to wait for the reply from a TCP for every small packet, it could slow down the game. In these cases, an UDP channel will be used to send large amounts of small packets. For example, positional information can make up for lost data packets by using a series of packets to build a trend or a prediction model.

Confidential

Synchronization

Objective
In *Privateer 3* there are two areas of the game that need to be synchronized: combat and non-combat. All non-combat data will be sent via TCP channels in order to keep a reliable information base on both the server and clients. For speed reasons, combat orientated information will be sent via UDP channels and corrected from with in the server and/or clients by prediction, scheduling, or dead reckoning.

Implementation
One of the largest concerns in any network game is coordinating the clients' packets in such a way to sync them with themselves and the sever. If one client sees a ship blow up, it is imperative that all the clients in that area see that same ship blow up the same way (or nearest to the same way as possible). To synchronize the clients we plan to run the game on the server, let the clients send input to the server's game (or react to the client's lack of input), and send out the results to all relevant clients. The client in turn will need to interpret some of the server's packets to further "show" the data.

The Server's Game is Gospel

Objective
The game server will run the conclusive version of the game. All collisions, trades, and modifications to the game will happen first in the server, then it will be broadcasted to each client so that they can update their own games.

Implementation
There are many ways to keeping a game synched. The clients could send deltas to the server, which would simply update its game to then immediately send out that delta to all the other clients to update their game. This would be fine for awhile, but if you are using a UDP channel for its quickness, you could loose a couple of packets and then the games would be out of sync after awhile. Furthermore if the delay from the client to other clients is significant enough, the deltas may come too late to update all the clients properly. On the other hand, if the clients sent all of its deltas to the server, where the server updated the game and then sent out only the absolute resultant data to all the clients (including the originator), all the clients would know what the server "sees" and they could adjust their games to smooth it out. The latter model is the one *Privateer 3* is going to use.

Once the game on the server has started, either from a saved universe or a brand new universe, the only version of the game that has any merit will reside on the server. The clients will be fully dependent on the server for game information in order to change their own version of the game.

The advantage of this model is no client can mess up anyone's version of the game. Being entirely dependent on the server is also beneficial when two clients are trying to do the same thing like buying the same commodity from the same planet. In this model, the first client's packet to make it to the game buys the commodity. Anyone on the planet and is in the commodity screen would see the results of the purchase. Thus, the client requests to do things but it does not get to do those things until the server gives the okay.

Confidential

Spline Prediction Model

Objective
In order to keep the space combat portion of the network game fast, *Privateer 3* will be using an UDP channel for sending positional data packets. The one draw back to this is that UDP does not guarantee the packet's contents or its sequence. There needs to be some way to handle these packets with out severely altering the client or the server. To do this the client will send its 3d coordinate plus its last two coordinates for every positional packet. This will enable the server to plot a course for the client by using a history of plotted points in 3d space.

Implementation
Sending the client's three coordinates enables the server to convert this information into the clients position, velocity, acceleration, and heading. Inside of this coordinate information is the x position, y position, z position, yaw, pitch, and roll.

In response, the server will send the other clients' positional information in much the same way. A client's engine can predict where the other player is going by creating a 3d spline prediction. Further, since the player's will be in 3d space, the clients, AI, and space objects can be described in relative terms. That is, when the server sends an update to a client, the server can explain that another client's position by saying, "It is over -34 units on the X axis, +50 units on the Y axis, and +5000 units on the Z axis." This way we do not need to jump the client around to adjust itself to the server's exact game.

Data structure(s)

```
struct 3DCoord
{
        float x;
        float y;
        float z;
        float yaw;
        float pitch;
        float roll;
};

struct CoordPacket
{
        // for the present coord, present coord - 1, present coord - 2
        3DCoord[3];
};
```

Reconciliation Methods

Objective
At some point in time the space combat will become somewhat out of sync since the nature of it is fast and furious. There needs to be a way in which the game can sync the players with out affecting the gameplay. Most of the combat will be handled by soft-reconciliation, but when there is an event that has to happen the game will switch to a dead reckoning style of delivering data.

Implementation
There are two kinds of reconciliation methods that will be used inside of *Privateer 3*: soft-reconciliation & dead reckoning. Soft-reconciliation allows the clients to become somewhat out of sync as they might not see each other, they are too far apart, or maybe dead reckoning is too costly

Confidential

at the moment. If the two clients were simply flying to the same port the engine would simply fly them near each other in the same direction.

Soft-reconciliation is the default method for space travel. The server knows where they really are, but the clients may have an approximation. The server will be sending the real coordinates and the client will need to gracefully coordinate the information to make the scene smooth.

During a dog fight the server will try to keep the clients as close to their true paths as possible, but when their is a "collision" the game server will make sure to send that data on a reliable channel. The "collision" is not only when client hit other 3d space objects, but when a certain event must happen or must be seen on all client machines. In these cases the dead reckoning method will be used to sync the players. The clients will know which data is dead reckoning data by the token type and by which channel it was sent on.

Determining Results in Advance

Objective
One way of synching the players is for the server to calculate an event before it really happens, then schedule the event when all of the pieces are in position. This can be done to alleviate sending every update to the clients on a per frame basis, rather it can predetermine events for AI or other 3D objects.

Implementation
To sync the players by predetermining events the game server will need to send the current game tick count in order to get all the events to happen at the same time on each client machine. If a player was to shoot a missile at an AI ship, the server could do all the necessary calculations to see if the missile hits before the missile does hit.
This information is sent to all of the clients. On the game tick that the missile has "made it" to the ship the resolution will occur on those clients at the same scheduled time. In this way the missile becomes a scripted event, guaranteed to happen. When the AI ship is predetermined to be hit, the client would "move" the ship to where the event is going to happen and let the ship get hit on the exact "game tick" that it needs to happen.

This model really helps when a player fires a weapon that should kill in one hit. On the other hand, it would be better to equip the player with a type of weapon that would take several hits from before the enemy ship would be destroyed. Again, if a player shot a laser at another ship, the server could predict if the laser shot will hit then aim the bolt in the right direction and resolve things normally. Since the damage on the laser bolts will not kill the first hit, the discrepancies would not become immediately obvious.

There are situations where a predetermined scheduled event might cause some problems. If a group of clients receive a scheduled event, then another client joins that group, the server will need to update that client of the event to happen. The game server needs a list of active scheduled events per area to set this up. This way all the clients can remain synchronized.

Determining a Hit

Objective

Confidential

All collisions will be determined on the server, but when a client is shooting there is a weight on the projectile. Another 3d object that predicts that it will be hit by that projectile sends back a messages with a "target weight". The shooter has a slightly higher weight than the target in order to give the shooter a more chance to see things his way.

Implementation
When a player shoots a projectile that client machine does an initial calculation to see if it "will hit" and "what it will hit". This is simply a Boolean YES or NO flag sent along with the "I have fired" message to the game server.

When the game server receives this packet, it "fires" a shot from the attacker's ship and tracks the bolt to see if it hits anything. This is not predicted, the projectile flies its path until it has either collided with another 3d object, or the projectile times out (or "dies").

All the clients in the area get the message that the attacking player has fired a projectile. If a client thinks it was hit by this shot, it sends a message to the server saying that it was hit, otherwise it does not send any message.

Each of these flags holds a weight with it. For the shooter it will be 3, for the game server it will be 5, and for the target is will be 2. If they record a hit, the weight is positive, and if they record no hit at all it will be negative.

The results are listed below. The Attacker and the Target predict the path of the laser bolt only once, while the server actually resolves the event. The target can be a any 3d object like another client, AI ship, or an asteroid.

Attacker	Server	Target	Result	Description
+3	+5	-2	6	Hit @ 60% strength
-3	+5	+2	4	Hit @ 40% strength
+3	+5	+2	10	Hit @ 100% strength
-3	-5	+2	-6	Miss
+3	-5	-2	-4	Miss
-3	-5	-2	-10	Miss
-3	+5	-2	0	Hit @ 25% strength

Starting a Network Game

This section explains the different types of network games, how these separate game servers will be found on the Internet, and how a player starts his own game server.

Network Game Types

Objective
There will be five different types of game servers for the *Privateer 3* game, the semi-persistent universe trading game and four death match games: Arena Solo, Arena Team, Slaughter, and

Confidential

Capture the Flag Team. The trading game will be built first which will include most of the features for the "death match" games. Network space combat will be built in the following phases:

1. the space combat system with multiple ships running on a server
2. placing 3D objects inside of the space combat with basic AI
3. the arena combat system featuring a true 3D area map
4. the interactive 3D objects like "power ups" and flags will be placed into the network space combat system

Implementation
All of the "death match" games will take place in either space or inside of a 3D object like an arena. Both of these types of combat areas will be inside of the trading game, so to make the "death match" games we need to include an interface and setting for the players to be played.

The first phase, namely the space combat system with no other 3D objects other than ships, will be completed when multiple ships can log onto the same game server and be able to shoot at one another.

The second phase, placing 3D objects into the combat, will be completed when the clients can interact with the 3D object in space that are controlled by the game server. For example, having multiple ships flying through an asteroid field. Basic AI will be put in at this point to prototype its interface inside of the network game.

The third phase, arena combat, will be completed when multiple ships can fly through a 3D mesh constructed of single sided polygon facing inwards.

The forth phase, dropping in interactive 3D objects, will be completed when the ships can interact with other 3D non-player objects that have effects in the game like "power ups" and flags.

The Arena Solo game features either 3d terrain where all the clients will fight for the most deaths after an amount of time. The game server starts and waits for players. When the systems operator has all the players that he wants to start the game, he presses the "go" button. The Arena Team is the same type of game as Arena Solo, except the players are grouped together into teams. We will create a channel group that would only broadcast to the team members, as well as all the other positional channels. The Slaughter and the Team Capture the Flag game are played like the Arena version but has a "flag" to capture and use in the game for scoring.

How to Find a Game Server

Objective
There will be a central location where the game server registers its IP address and a description of that game. Then the clients can find game servers by going to this central location and start playing a *Privateer 3* game.

Implementation
The game server must first register its game on a central location for player to find. When the game registers its game it will supply its IP address, a description for the game, a name for it, and a password.

Confidential

When a client starts, he will query this central location for the active game servers. When the client is given the IP, the client then asks for a "Server Information" packet. This packet will contain the current players, their status, the server status and options, and latency between client and server. A client would also be able to update specific sources, and manage between favorite servers. When a client wants to join a game, he should be able to find the game in the compiled server information packets.

Starting a Game Server

Objective
The systems operator for that game server will start the game and register its IP address in a central location so that other clients can find universes to play. The clients are not restricted to just those IP address as one central location, rather they could simply plug in the IP address of their choice. This central location is intended to help any user to find a game server to play on.

Obstacles
- Finding a central location.

Implementation
The game server is a central processing machine. The responsibilities include gathering client packets, processing the world changes, calculating AI, and returning the packets to the clients. The server will not be displaying any graphics, just keeping track of all events.

When a server is started, it will send its IP address to the master server. From there, the server waits for clients to join, and process the game. When a client attaches, the server checks its "ban" list, collects the clients information, and after everything is valid and transferred, gameplay begins.

There will be many options so that not all servers are not the same. Some of those options will be team play style (friendly fire hurts others or modifying parts of the trading game universe), maximum number of players, maximum number of AI players, allowing alliances or not, allow ship transfer, making a client vulnerable if the player is in combat and he drops out (either timing out of pulling the plug), password protection.

Account Maintenance

The game server will keep track of all accounts on it's server. Each account will have a name, password, last IP, and last date accessed. In addition, each account will have various sysop definable flags such as purge on/off. The sysop will have the ability to delete accounts, ban accounts (which will ban the last IP), close game (refuse new accounts), and open game (allow new accounts - default). The sysop can also set the maximum number of accounts. Accounts will be purged after an interval of inactivity set by the sysop. When an account is purged, it will check through alliance data and remove the player from there as well if necessary.

Confidential

Running Privateer 3 on the Internet

Objective
To run a *Privateer 3* game server on the Internet we will need to provide some services for servers and clients to play the game: IP matchmaking, a central location for downloading patches, possibly saving player statistics on server side, and allowing players to transfer their characters between game servers.

Implementation
There are a couple of ways that we can have players find other game servers:

- One way is to have all the game servers register their IP address on an online gaming service. When they start a game server the central location could do some checking, they place the game server on a list which the client can choose from. Further, this central location could start its own game servers and place its own games on the IP address list of available game servers.
- Another way to provide the same service is to have the game server register its IP address on a certain web page. To connect on a game server, the clients would pull up the web page, find an active game server IP address, read all the game server options, and then choose one of the game servers to connect to.

The game will allow the client to attach itself to any IP address that is running the game server, but at first the client would not know the IP addresses of any game servers. Thus, having a central location is essential for the game networking to start.

Another feature of this central location is autopatching. Whether or not the patching procedure is automatic or not, having a central location for all the *Privateer 3* customers will be very clean for the customer. In any case, there should be a way for the customer to apply a patch from within the game. Note, the game servers will not have the latest patch, the central location would. This ensures that a game server systems operator can not trick a client into downloading a virus infested patch. Further, the patch would be the same for all clients.

One function that could be done in many ways is saving the client's data. The first question is where is it going to go? It could be put in three areas: on the client, on the game server, on a central area on the Internet. If the data was saved on the client's side the player could take his "character" from game server to game server, but since the data resides on the client side it is very susceptible to hacking. There are many procedures for encoding this information and preventing hacks which are discussed at the end of this chapter.

If the data were saved only on the game server hacking would be much hard (especially if the data was encrypted), but the major draw back is the client would not be able to take his "character" over to other game servers (there are ways of transferring the client's data with out it ever going through the client, but the require some sort of FTPing which could hit a firewall).

If the data is saved on a central location we would get the best of both worlds, we would be able to save the client's data in one place and the client would be able to jump from game server to game server. The chance for hacking goes down quite a bit since the data does not reside on the client's machine. There is one major draw back to this, OSI would have to maintain this "master server" in

Confidential

one form or another. Responsibilities like deleting old player information, backing up the data, and validating the client's request for the information to send it to a game server would all be on OSI on a continuing basis.

Overcoming the Internet's Limitations

Summary
There are two major limitations that *Privateer 3* must address in its network design: latency and bandwidth. As if the challenges of living in a relatively high-latency environment were not sufficiently daunting, the variable nature of Internet latency introduces an additional challenge for the multi-player game developer. Regardless of the typical latency values, significant variation in latency can degrade game play and complicate game synchronization models. We expect to handle any delay from 250ms to 300ms. For an Internet game this is a reasonable expectation. Most of the packets the clients can just wait for the absolute results to come back to it from the server, but for a quick space combat system we can not afford this kind of interface. Further, most other packets like trade, text messages, system, and auto jump messages will be sent via TCP channel.

In terms of bandwidth, examining the size and contents of each tokenized packet structure will be the starting place to alleviate most bandwidth challenges. Another way of handling bandwidth is aggregating and combining packets before sending them, this will prevent sending a load of tiny TCP packets all waiting for their receipts. Finally, we plan to group the clients together so that we can choose to send a packet to only those clients in a certain group.

Optimizing the Space Combat Sequences

Objective
The most unique relationship between client and server in the *Privateer 3* network model is going to be in the space combat. This is the number one place that a player will see latency. The player is reacting quickly to an enemies actions, thus the input becomes quick and furious. We will deal with this outbound data for this particular instance in two ways: the data will be send via a UDP socket, and the data will be aggregated into a larger packet before it is sent (it will delay sending it for something like 50 milliseconds).

Obstacles

- Why not use compression?

Implementation
Sending a lot of UDP packets inside of space combat is easy enough. The place where this kind of code gets nasty is when you start compensating for lost UDP packets. The first thing to do in this code is to generate a list of packets received. The next thing will be determining the breaks in the packets. The third part is compensating for the lost packets. In space combat and movement the client will be continuously sending its coordinates in sets of threes. These points will be overlapped by one coordinate so that it can create a chain of points. When it receives a packets that does not have a daughter coordinate, it will place in points into the coordinate chain by a prediction method discussed in a further section. When or if the engine receives the out of sequence packet, it will either place it in the chain or simply discard it if it is too old.

Confidential

Grouping the Players to Decrease Bandwidth

Objective
Another way to help the bandwidth limitation will be the grouping of players by their locations. For example, in the trading game each player will be a member of the universe, one quadrant of the universe, one system of that quadrant, and a navigation point of that system.

Implementation
For space combat, the client will only be receiving information about that navigation point. Thus if there are sixteen players in the game and three of those players are in a navigation point with two AI ships, one client would be getting updates about the other two human players plus the two AI ships but no information about the other thirteen players also in the game. This kind of grouping will help the inbound traffic immensely.

Using AI to Decrease Bandwidth

Objective
Privateer 3 will also be able to skip AI movement packets for awhile if the server outbound traffic is becoming too heavy to a certain client.

Implementation
Since the server and client hold the same AI engine, it is possible for the server to tell the client to move the AI every other time. This way the client would not need to have all the AI ships updated every network update, and lower the number of packets needed to update the client's view screen. The AI on the client's machine will be required to determine its next event every other update.

Using a History of Coordinates

Objective
There will be a history of plotted points generated on the server and clients during this phase of the game in order to better predict and to tell when there has been a dropped packet. The client will send three points of travel every positional packet: one for the current coordinate, one for the coordinate the client was one move ago, another one for the coordinate where the player was two moves ago, and the current orientation. With these three 3d coordinates we can determine the following about the player: facing, velocity, acceleration, and the absolute position of the player. This makes predictions, collisions, and synchronization on the server easier.

Implementation
The server will produce this same history of three coordinates for each client in order to describe the clients' moves. This means both the clients and the server will be fed the same kind of information which will make coding the results and scripting events easier.

In order for this history of coordinates method to work, a "follow the dots" engine will need to be made for the game engine. Since the server and the clients will inherent this engine, the results will look very smooth and synchronized.

Another place where this history of coordinates will help is when the server loses connection with a client for a short time. Since the network is seen as a virtual input device, the game server could

Confidential

generate some AI input using the history of coordinates to help predict the player's moves. Note, this method would only be used for a maximum of a couple of seconds, but it would really help hide the Internet's latency. As soon as the server starts receiving packets from the lost client, it would course correct the ship's position to compensate.

Dealing with Lost Client Connections

Objective
If the server could not reconnect with a client, either because he disconnected or because of massive latency, the player's ship will decelerate to a halt. After a certain amount of time it will simply let the ship disappear. This type of delay allows the opposing player to try to destroy the ship, but does not make the ship a permanent sitting duck.

Implementation
The network will start slowing down the client after a certain amount of time passed without any packets from the client. After awhile the ship will finally stop, letting the ship sit for a minute or two. If the client's packets start coming back, the ship will accelerate appropriately. Although, if the client never comes back the client's ship simply disappears.

When the player wants to logout, the game server will save the location that the client was at plus all of his ship and equipment. The pause before saving off the client is to prevent players from pulling the plug when they are in trouble. When the player logs back on, the game server will place the player in the same location, whether it is one a planet or in space, with the same space ship including any damage that was sustained. If the player was destroyed while he was gone, the game server will place the player on the last planet or port that he visited.

Dealing with Cheating

Objective
The way players cheat the games that are played over the Internet is by modifying their character's statistics and send them to the server to be updated. There are a number of ways to prevent this kind of tampering like randomly doing a checksum on the data, keeping track of the inbound/outbound ratio to trace inconsistencies, encrypting the data, detecting the software that allows this kind of tampering, and identifying the client's hardware with a unique number.

Implementation
Game server sends random key to client, client has 3 seconds to return checksum, otherwise client is bumped off the server.

Game server keeps track of inbound and outbound traffic to determine if any cheating is going on client turns modem off/on real quickly

Data is sent encrypted and saved on the client side encrypted.

Client & Server detects for Soft ICE ™.

Confidential

Chapter 4

4

Origin, Systems Inc.

Technical Design Document

Concepts: The Game

Game Interfaces - On Port

This section details the On Port interface screens and their relationship with game data.

Main Port Screen

This screen allows access to the various planet screens allowed by each port.

Commodity Exchange

The commodity exchange allows players to buy/sell cargo at a port. If more than one player makes a request, the first request recognized by the server is honored (first come, first served). Aside from modifying commodity information on a planet, which could result in price fluctuations, and player cargo and credits information, the data accessed by this exchange is static. After a transaction, the planet will check to see if the player has caused a planetary event (such as a Xenomorph Infestation) and react accordingly.

Information Exchange - Single Player

The information exchange section allows the player to access information about the game. It contains the following screens:

- *News Wire* - The news wire displays the last 100 major events that have happened. Each system containing ports will keep its own list of the last 100 major events, because events can be scoped to a certain broadcasting range, therefore, not every system will hear of every event. This will "personalize" each planet's news and events.
- *Ships* - Allows the player to access information on publicly known ships.
- *Want Ads* - The player can hire an AI wingman and turret gunners from this screen. The more skilled the AI, the more it costs. The effect of hiring one of these is seen only in spaceflight.
- *Buy/Sell Stock* - Allow the player to purchase and sell stocks. The stock market is explained under the Economic System section.
- *Sports Wagering* - Allow player to bet on AI arena battles. After bets have been placed, the AI's will duke it out in an arena. The host planets will keep a list of what bets have been laid and for how much, then will pay off after the battle concludes.

Confidential

Information Exchange - Multiplayer

The information exchange under multiplayer conditions is the same as the single player with the following additions:

- *News Wire* - Includes player deaths and kills. The scope of the broadcasting of these events is based upon the overall reputation of the killer and the victim.
- *Personnel's* - This section functions much like the News Wire, except players can input their own messages for a fee, and there is only one personals database for the universe. The last 50 messages are viewable.
- *Player Information* - Allows players to access information about the players on the server. It lists all players who have played on the server recently (and haven't been purged for inactivity). For each, it will display the players' names, alliances, personal bio, and whether or not they are logged on at the time.
- *Alliance Information* - Displays information about player created alliances. Information includes alliance name, icon, bio, president's name, net worth, and stock symbol.
- *Alliance Access* - This option is only available to players that are members of an alliance. The options available under this option are detailed in Game Interfaces - Alliances.
- *Sports Wagering* - Allows betting on player arena matches.
- *View Arena Combat* - Gives the player a view into a current arena battle so he can watch the fight in progress.

Hangar

The hangar is where ships are bought and sold, repaired, dry-docked, and outfitted. The following screens are available:

- *Ship Sales* - Allows the player to access static information about various ship types, and allows purchasing of ships.
- *Sell* - Lists the resale values of the components of your ship and allows you to sell what you want from them.
- *Repair Bay* - Lists damaged items on your ship and their repair cost. Allows you to repair damaged items.
- *Equipment Sales* - Displays a graphic of your ship and open spaces on your ship, obviously dependent on the type of your ship. Allows the player to add/remove equipment.
- *Dry Dock* - Allows the player to store a ship at a port. Where a ship is stored will be in the player's data, not the planets.

Arena - Single Player

The arena interface only appears on the pleasure planets in each quadrant.

- *Choose Opponent* - In single player mode, the player is given a list of AI challengers whom he can select from to battle in the arena. The more difficult the fight, the more reward that is offered for victory.
- *Choose Arena Type* - Player can choose what type of arena environment he wants to fight in.

Arena - Multiplayer

In multiplayer mode, players can set up their own private arena scenarios.

- *Join Game* - During the waiting period after the arena has been initialized, any player who is authorized to join the game may do so.

Confidential

- *Set Up Arena* - Player chooses the environment, and then either One on One, Last Man Standing, Team, or Alliance Battle, after which the player is taken to the appropriate sub-screen.
- *One on One* - The player setting up the arena has the options to either specify an opponent or to take on anyone who wants to fight. If no challengers enter after 5 minutes, the arena battle is canceled.
- *Last Man Standing* - The player setting up the arena can set up number of combatants allowed into the arena. If the roster is not filled after 5 minutes, the game begins with the players who have registered.
- *Team Battle* - The player setting up the arena sets up the teams, which includes the number of combatants on each side and who they are. If after 5 minutes the team members have not joined in, the battle is canceled.
- *Alliance Battle* - The player setting up the arena chooses which alliances are fighting and how many men each side can bring into the arena. If after 5 minutes the rosters are not filled, the battle is canceled.
- *Tally Board* - After all arena battles, the tally board for the planet is updated with the results of the latest battle. The tally board remains until the next battle starts, when it is wiped clean.

Load/Save ATM Machine

Allows the player to load saved games or save games.

Personal Access Directory

The Personal Access Directory (PAD) appears as a bar at the top of the screen that unfolds like a Windows bar with auto-hide on. It allows access to various static data and also allows the player to alter certain data. It contains the following menus in addition to a close button (which removes the bar. There will be a key to bring the pad up as well.

- *Personal Information* - Brings up a display with the following static information about the player.
  - Name
  - Credits
  - Total Cargo Cost
  - Net Worth
  - Ships Owned - For each ship, it displays the hull type and where it is dry-docked.
  - Alliance Information - For each Alliance the player status with that Alliance is displayed, and how many of that Alliance the player has killed.
  - Total Kills / Kill Ratio
  - Ship's Log - This area will contain a scroll bar so that the player can scroll through the journal entries. For information on how the log works, see Ship's *Log/Player Diary*.

- *Ship Manifest* - For your current ship, the Ship Manifest contains the following information:
  - Type of ship (base hull type).
  - Number of storage compartments.
  - Ship tonnage / tons used.
  - Current inventory. For each compartment it will tell the type of cargo, the amount of cargo, and the value of the cargo. In spaceflight, the player will have the ability to eject compartments from

Confidential

the current inventory. Portside, the player will have the ability to trade compartments from the current inventory.
- Secret inventory. Certain ships have compartments that are shielded from scanners. The inventory of those is displayed here the same way as the Current Inventory.
- Special Items / Passengers - a list of any unique items or passengers that you are carrying.
- Hired Wing man - information on your wing man.
- Trade With Player (Port Only) - Selected goods or credits are moved to a trading area. The player you select to trade with offers his goods / credits in exchange, then both sides either accept the trade, in which case it goes through, or one declines, in which case the trade is canceled.

- *Nav Map* - Brings up the full Nav Map. See *Nav Map* section for more detail.

- *Ship Status* - In spaceflight, this gives you information about your current ship. On port, you can cycle through all the ships you own. It gives the following information:
  - Type of Ship (hull type)
  - Armaments
    - List of weapons and their status
    - List of types of missiles, and quantity held. Also gives status of launchers for each missile type.
  - Turret Status - for each turret, what is there and it's status.
  - Power Plant - type of power plant and output.
  - Shielding
  - ECMs/ECCMs
  - Inertial dampeners
  - Decoys
  - Software
  - Additions/Upgrades

- *Communications (Port Screen only)* - Allows the following communications methods.
  - *Chat (Multiplayer Game Only)* - Brings up your chat window and allows you to enter a message to either your Alliance members, Friends (a list of players you have designated), Everyone, or a single user
  - *Game Mail* - Allows you to read your game-mail or (in multiplayer mode) to send a game-mail message to your Alliance, Friends, or a single player. There is no spamming (although a player could add everyone on the server to his Friends list)
  - *Wire Service* - Allows a player to send money to another player or to his Alliance. Alliance coffers will automatically take the money and log the payment, while a player will receive it only if he is on-line. If he is off-line, it will be sent in a game-mail message and he will receive the money when he reads the mail.

- *Game Options* - Allows the player to modify game options in play including:
  - Sound effects / music volume
  - Voice options
  - Sound effects Options
  - Reality Level (toggle Lens Flares / Blackout)
  - Detail Level
  - Skill Level (Single Player Game Only) - 1 - 10 / invulnerability
  - Landing Sequences - watch once / watch always

Confidential

Ship's Log / Player Diary

To insure that the player does not get lost in the plots, every time the player completes part of a plot a note of it is made in the ship's log. A note of anything the player needs to do next is also logged.

Example:

```
12/1 Delivered Powered Armor to the Rebels at New Damascus.
12/2 Acquired retrieval mission for derelict ship at Nav 124.
12/4 Destroyed Alicia's ship at Nav 126. Going to Antwerp to collect
10,000cr bounty.
```

This data will be stored on the player's machine in text file format, and will be updated as needed by the game. Alteration of this data for cheating purposes is not an issue, since this is read-only for the player, and no data from this file is sent to or relied upon by the server in any way. The only person who would see any effect from altering this data is the person who altered it.

Game Interface - Spaceflight

In spaceflight, the game interface takes place on your HUD, where you can call up and hide various gauges that give you information about your immediate environment.

- *Player Damage Display* - As the name implies, this is a readout of the damage the player has taken and the current status of the ship shields. It consists of the player's ship within 2 rings - an inner one for armor and an outer one for shields.
- *Target Damage Display and Read Out* - The Target Damage Display is the same as the Player Damage Display, except that the ship in the middle of the rings is the target ship. Depending on the player's scanners, he may have a readout for the target as well, which appears next to the Target Damage Display and shows various information about the targeted ship.
- *Target Acquisition Scanner* - There are 2 types of Target Acquisition Scanners (TAS), a 2D and a 3D version. The 2D version will be the same as the *Prophecy* TAS, while the 3D version will be a sphere with the player in the middle and the blips scaled to represent depth in the scanner.
- *Gun and Missile Display* - The Gun and Missile Display are separate gauges, but function exactly the same. Each gun and missile have their own icons which will be represented in the appropriate display. Each will also be color-coded to show damage levels. The missile gauge will also show the quantity of remaining missiles of the selected type.
- *Communications and Messages* - The communications and messages area is where the player can read and send game messages. The incoming text message area is 4 lines (expandable by player) and allows the player to cycle through the past 20 messages. The incoming buttons allow the player to filter out messages that he doesn't want to receive. The mail and SOS indicators flash when the player has a game mail message or an SOS game mail message that hasn't been read yet.
- *Power Distribution Display* - Displays and allows the player to manipulate the amount of power going to the Weapons, Shields, and Engine systems.
- *Camera* - This gauge is essentially another camera view that the player can watch. It will either show a close-up of the target, a rear view of the player, a missile cam, or a drone cam.

Confidential

- *Meters* - These are various minor gauges that give information about the ship. They include the Speedometer, the Set Speed, Jump Indicator, and Fuel Gauge.
- *Defensive Measures* - This is a panel which gives the player indication about both missiles being fired at him and what defenses you have to counterattack with. The DF, IR, HS, and FF/SK are merely indicator lights which tell you if there is a missile of that type targeted on you. The ECM indicator shows if you are being jammed. The ECCM indicator shows if the ECCM is active and can defend against FF/SK missiles. To the right of IR and HS are indicators showing the quantity of countermeasures you have against those types of missiles.
- *Target Reticule* - The target reticule has various small components that make it up, including the center reticule, the Target Direction Arrow, the Port Direction Arrow, the Analog Gun Energy Gauge, the Analog Speedometer, Missile Hash Marks, Match Speed Indicator, and Smart Targeting indicator.
- ITTS - There are two independent ITTS cursors, the gun and missile ITTS. The gun shows how much the player should lead the target to hit it at it's current speed and vector. The missile lock follows the targeted ship and flashes when the target is locked. There is also the Digital ITTS Activation Gauge which gives the range to the target if it is out of ITTS range, and the Blinking gun ITTS cursor which blinks when the player has the target in his sights.
- *In-Flight Nav Map* - This is a much simpler version of the full nav map that pops up onto the player's screen and allows the player to make quick changes to in-system destinations. For elaborate navigational plans, they will have to use the full nav map. Also, if a destination is changed in the middle of an elaborate route, the route will have to be re-plotted from the player's new location.

Nav Maps

Full Nav Map

The full nav map is the standard map that the player will pull up most often. It consists of an information window, a couple of information readouts giving the player's current location and destination and some buttons. Features include:

- *Pathfinding* - The player can click on a navpoint or port and the computer will pathfind the shortest route to their destination.
- *Change Views* - Allows the player to view the universe on a system or quadrant level.
- *Port Information* - Player can click on a port and having information about the port pop up in the info window.
- *Mission Specs button* - shows current mission specs in the information window.
- *Missions Available button* - allows player to look through current missions offered in his quadrant without landing.
- *Player Diary* - brings up the player diary in the information window. See Ship's Log/Player's Diary for more information on this feature.
- The joystick will rotate the current map (either system or quadrant) in 3D.

Confidential

Game Interface - Alliances

This section explains the game interface for player created alliance stations in the multiplayer game. The higher rank a player has in an alliance, the more options that are open to him/her. Each rank below has all the options in its section and all prior sections.

In the single player game, there are no player created stations, so this is not applicable.

Initiate Menu

- *Bulletin Board* - Just like the personals except it is restricted to alliance members only. A separate bulletin board is created for each alliance.
- *Disassociation* - When a player selects this option, he is removed from his current alliance.

Member Menu

- *View Members Stats* - Lets you view the members and their stats.
- *Locate Members* - This option displays a list of alliance members, their current locations, and whether they are online or not.
- *Location of Alliance Station* - This displays the location of the alliance base.
- *Pay Dues to Alliance* - This allows the player to pay his dues to the alliance treasury.

Commander Menu

- *Warehouse Ledger* - For each alliance, a warehouse ledger is kept, which records the cargo transactions at the alliance station. Only commanders and above can see this ledger.
- *Buy/Sell Station Equipment* - This allows the member to buy or sell weapons, armor, shields and equipment for the space station.
- *Hire AI Mercenaries* - This is the similar to the want ads section, except this is for mercenaries who will guard the alliance station and defend it.
- *Add/Remove Member* - Add to or remove a member from the alliance. A member cannot remove a member of higher rank (i.e. Commanders cannot remove Co-Presidents). New members default to Initiates.
- *Promote/Demote Member* - Change the rank of another member. A member cannot affect a member of the same or higher rank (i.e. Commanders cannot demote Commanders and above) and cannot elevate another member above his current rank.
- *Invite Player to Join* - Sends a game mail message to a player inviting him to join the alliance. If the player accepts, he is automatically joined into the alliance as an Initiate.

Co-President Menu

- *Change Alliance Information* - Lets player change alliance bio information and alliance icon.
- *Transfer Alliance Stock* - Lets player give alliance stock to another player, whether in the alliance or not.
- *Retire Alliance Station* - Sells back the station.
- *Dissolve Alliance* - Dissolve the alliance permanently.

Confidential

- *Create self-destruct Password* - Set the self-destruct password. Requires knowledge of old password.
- *Initiate self-destruct* - Self-destruct the station with a time delay.

President Menu

- *Promote/Demote Member* - For the President only, this option gives him the ability to abdicate to another alliance member.

Planetary Economics

The planetary economics are fairly straightforward as detailed in the *Design Document, Appendix E.* Each planet has a price range on the commodities it will buy or sell, and an amount that it will produce or consume of these products each universe turn.

In addition, planets will not buy or sell all commodities. They will sell the commodities that they produce, and buy the commodities that they need. This will be set on a planet by planet basis. The port government also influences this list.

The prices that a planet will pay for a commodity are based upon its usage of the commodity. When more than the planet can use has been sold to it, the price it will buy the commodity for will begin to drop. At a certain level, the planet will no longer buy any of the commodity. Prices rise again as the planet uses up the commodity.

The prices that a planet will sell a commodity for are based upon its production of the commodity. When stores of commodities fall, prices begin to climb as the commodity gets scarcer. Prices fall as the stores increase due to planetary production. When the price hits the minimum that the planet will sell for, it ceases to produce this commodity until the amount on planet is reduced through trading.

Interstellar Trade Routes

Interstellar Trade Routes are trade routes routinely run between various planets. Each planet will have a trade scheduler that tells it when to send another shipment and to where. This *trade scheduler* is dynamic and can be altered as the result of external events such as the planetary events or as a generated mission.

Route data will specify whether it is a one-time or recurring trade route. One time routes will be added by the *mission generator* and can be added as well by the effects of planetary events. Recurring trade routes will automatically add another instance of themselves into the scheduler to take place after the appropriate time interval.

As recurring trade routes are run, they will keep track of whether they encounter any opposition in their route. If they do, the originating planet will begin increasing the strength of the ships sent over that route, as well as their escorts. If no opposition is encountered, the convoy strength will gradually decrease over time back to its original strength, so the trade routes will dynamically adapt to a changing universe.

Confidential

If a trade route is intercepted and destroyed, the price of the commodities it was transporting rise on the destination planet, as the destination planet now has a shortage of the commodities and is willing to pay more for them.

Interstellar trade routes can also be set up for AI alliance maneuvers in the single player game, and for AI Pirate, Mercenary, and Confed patrols. They would act the same as normal trade routes, except they would carry no cargo. When the routes of opposing groups intersect, there is a chance that combat could be started (spawning a mission for the player).

Planetary Events

From time to time, various events occur in the universe. Most of these events have a purely economic effect on the game, although some can have a more dramatic effect than others. These events can happen randomly or can be influenced by the actions of players. These events are detailed in the *Design Document, Appendix F.*

Each event type will have a data table containing which goods it affects and by how much. In addition, each will also have a severity rating and a duration based on the severity and event type to govern the length of time the planet is affected by the event. The severity ratings will determine the range of knowledge about the event. A planetary event will be broadcast based on this criteria, so you won't hear about a minor event halfway across the universe.

Also, events will be able to spawn other processes, such as Interstellar War which, for example, will begin a war between two different planets, and the duration will be for as long as the war continues. If there is a winner, the defeated planet's governmental type would most likely change, as would the planet's allegiances, affecting the way business is done at that planet. For instance, a pirate planet taken over by the Confed would not be such an easy place to get rid of contraband goods anymore. When the war is resolved, another message announcing the end of the war and the result will be broadcast. So not only would this event affect the economics of both planets involved, it would make it more difficult to reach either of them since you would have to fly through an interstellar warzone, and it could affect the dynamics of the universe for the rest of the game.

While many of these events will happen randomly in the universe, each planet will also have an event scheduler, whereby the planets can have prearranged times for various events to happen (such as holidays), and new events that are caused by external stimuli can be inserted. They can also be either one time events, or be continuously reinserted into the scheduler at decided intervals.

Events can also be caused by players as well. Each event will have certain criteria that need to be met before an event occurs, and each planet will keep track of these criteria to see if an event has been forced into occurring by the players' actions.

Obstacles:
The main obstacle of the Planetary Events is that while they do share much in common, they also have their own disparate effects upon the universe as seen above, which will need to be individually coded for each event that needs a unique effect. Also, the criteria for which players can create these events needs to be specified and implemented.

Confidential

Stock Market

The stock market is an additional economic feature. Stock market symbols and starting values are listed in the *Design Document, Appendix D*. Each company will be linked to various commodities that the company depends on and produces. Planetary events that affect commodities will also affect related stock indexes. For example, if the price of ore drops due to an event, mining companies will see a dip in their stock valuations while companies that manufacture using ore will see their stock rise as higher profits are anticipated as a result of lower materials costs.

Aside from the planetary events, the stocks will fluctuate based on various randomly determined factors. The first factor is the production factor. For each type of produced commodity, there will be an associated group of companies. Note that a company can be in multiple production groups.

The second factor will be the materials factor. This represents the changing value of companies based on the changing cost of materials. For each material (Di-Cor, grain, etc.) that is a base material, there will be an associated group of companies. Note again that a company can be in multiple materials groups.

The third factor is the individual company factor. Each company has its own individual factor, which will keep all related companies from acting exactly alike, although general trends can be influenced by the group factors.

These factors will be randomly generated in a predetermined range. After they are determined, they are factored into company stock prices, which are then updated.

Port Government Influences

Each planet has a certain type of government running it (see *Design Document, Appendix G*). These government types mainly affect the economies of the planets.

Each government type will affect consumption, production, and cost of goods available. They will also be a factor in determining what is for sale at each port and what each port will buy. In addition, certain events are more likely to happen under certain government types (Ex. A rebellion is more likely to occur on a Totalitarian planet than a Democratic planet, while holidays are vice-versa).

Combat AI

For combat AI, we will be using the *Wing Commander : Prophecy* AI system as our AI engine. This AI engine supports many features including formation handling, AI to AI comms, maneuvers affected by pilot skill and ship characteristics, specific maneuvers for each ship type, multi-ship maneuvers, and environment tracking and reaction.

One feature not supported for *Prophecy* is the recognition of enemy status (i.e. 2 AI ships will not attack each other without provocation or having the attack scripted in the AI). However, this feature should be fairly simple to implement, as each ship will have an alignment and a reputation ranking. With that data, it would be simple for a ship to look at the other ships in it's area, and choose one to attack, if any.

Confidential

The AI's also contain event handlers, so the AI can interrupt it's script upon the occurrence of various events such as being shot, reaching a predetermined level of damage, or any condition upon which a special routine needs to be called.

AI attacking will depend upon the pilot skill level. Different skill levels have access to different maneuvers and handle the ship at different levels of competence.

AI scripts and event handlers are written in MED, which compiles them into FLO code which is interpreted by the game and calls AI commands written by programmers. The AI command list is easily expandable. In addition, the scripting is all data driven and easily modifiable by the designers.

Possible Obstacles : Speed. The AI in *Wing Commander: Prophecy* needs only worry about one area at a time, the other areas in the mission are inactive. We have a dynamic universe with activity going on in multiple places, not just where the player is, so many more AI's can be active at one time, and can interact with many more objects. Due to this, there is the potential possibility of the AI slowing down the game, especially in the single player game where the machine has to keep track of the universe and render the game. If so, optimization and possible restructuring of the internals of the AI system will have to be considered. Also, since our game is structured differently than *Prophecy*, we will probably need additional modifications to the AI system that we have not predicted. However, I believe that the advantages gained by using proven technology as a solid foundation outweigh these obstacles.

Arena

The arena is a place for organized combat. In the single player game, it's use is as a source of money. In the multiplayer games (both arena only and trading game), there are different conditions and settings for the arena to vary the game.

Environment

The arena will have ten different environments which the player can choose from, which will be constructed by the designers. The objects in the environment would be the same as objects from other areas of the game, so no unique coding should have to be done to provide these environments besides allowing the choice.

Conditions

The arena will keep track of the players inside it and the winning conditions, and as players drop out it will test to see if the winning condition has been met. When the condition is met, the battle ends and is resolved. The arena will also keep track of what money has been earned in the arena, which is paid out to the players at the end of the battle.

Multiplayer Arena Only Games

The server setup allows for the setting up of an arena only game, without the trading game.

Confidential

Sysop Options

- Type of Game - The sysop can choose the type of arena combat.
- Deathfest - This is an every man for himself setup.
- Capture the Flag - This groups the players into 2 teams, which try to capture each other's flag while protecting their own.
- Slaughter - Like capture the flag, except without teams.
- Environment - Choose between 10 different environments.
- Winning Condition - Choose when the game is over, if at all.
- Other Options - Set other parameters of the game.
- Team Safety - Only available for Capture the Flag games. Teammates are invulnerable to each other when Team Safety is on.
- Unlimited Ammo - Ammo is not used up.
- Power Ups - Enable selected power ups and their frequency of appearance.

Power Ups

Power ups are the same as regular space objects, except that when a player collides with one, they have various effects upon the player's ship. Planned power ups include Double Damage, Warp Shield (invulnerability), Stealth Shield (invisibility), Regeneration (energy/shield), Double Shield Strength, and Death Blossom (super weapon).

Mission System

In the game, there are various types of missions that the player can receive. Unlike many other games, the player may have multiple objectives at the same time. The player is able to accept one plot mission, one alliance mission for each alliance he is a member of or working for, and up to 3 random missions at the same time. The order the player chooses to follow these is not fixed. The player could choose to try to become a high ranking member of an alliance before even attempting the main plot, or could ignore alliances entirely.

In the multiplayer game, there are no alliance missions, and no main plot missions, only random missions. The rest of the game will be generated by player interaction with the universe and each other.

In the single player game, the scripted missions will be accomplished with "overlaid" action spheres containing items that will be added into an already existing action sphere and wait for the player to trigger them. Each mission will be responsible for setting up and cleaning up it's own objects. In addition, upon activation, the overlaid missions would have the option to "suspend" whatever was going on in that action sphere (such as a trade route), which would be reinstated when the mission is completed and the player leaves the sphere.

For a description of alliance missions, see the *NPC Alliance* section. For random missions, see the *Random Mission* section.

Random Missions

The random missions are generated by the information kiosks or by NPCs who give out missions. There are 8 different types of missions as detailed in the design document under Kiosk Generated Missions. A

Confidential

player will be able to contract out up to 3 of these missions at a time. Difficulty of the mission will be determined by the location that the mission is contracted out from. A quadrant 4 mission will almost always be much more dangerous than a quadrant 1 mission. At the same time, the offered reward will be based upon the difficulty of the mission.

Once a randomly generated mission has been accepted, the mission parameters are stored and the mission is active unless it times out. If the mission is completed, the player receives a game-mail message and the credit reward for the mission.

In the multiplayer game, when missions are offered, they can include missions that were already given to other players as well. The player who completes the mission is the only one eligible for the reward. Other players who had accepted the mission will receive a game-mail message saying that the objective has already been accomplished by another contractor and that their mission has been terminated. If a player who has not accepted the mission completes it (such as a player happens across a random combat mission, joins in, and one side is defeated), then he will get a game mail thanking him for completing the mission (which he was previously unaware of) and the reward.

To prevent missions from never being removed from the universe due to never being completed, each mission will have a timer, which begins counting down when the first player accepts the mission. If no one completes the mission within the time allotted, the mission resolves and removes itself and the players who had accepted the mission are sent a game-mail saying that they did not complete their mission in time and the mission is being terminated.

Random combat missions will be set up at the time of acceptance in an action sphere with no players, unless it is a result of enemy routes intersecting (i.e. a merchant trade route encounters a pirate patrol). If it is a result of route intersection, the combat takes place as normal. If not, the starting state of the combatants is determined at the start of the mission (i.e. A merchant ship has requested immediate help, it is in great danger. After accepting this mission, the merchant ship is set to heavily damaged, and the pirate ships are set to lightly damaged.) Both sides are invulnerable until a player enters the action sphere. When a player enters the action sphere, the invulnerability is turned off and the fight proceeds as normal. This allows us to set up scenarios and to control the type of random mission combat that occurs. If the mission times out with nobody in the action sphere, the fight is resolved in favor of the stronger side. If neither is stronger, a random outcome is determined. Either way, a game-mail is sent to anyone who accepted the mission that the mission has been terminated and the outcome is reported.

Player Alliances

Player alliances are supported in the multiplayer game.

When an alliance is created, the server starts a new alliance file. The founder of the alliance will specify all initial information including the bio and alliance icon. The server will automatically add a new game mail account for the alliance, as well as allocate a chat channel identifier.

All alliance data will be saved on the game server. The alliance file will contain the alliance bio, icon, space station information, and all member data. Members will be identified internally by their account ID on their game server, so it doesn't matter if they take their character to other servers, then come back, it will just update his data upon his return. In the event a player is purged for inactivity on a server, he will automatically be deleted from any alliances he is part of.

Confidential

NPC Alliances

NPC alliances are only in the single player game. They exist to add more depth and character to the universe. NPC Alliances are detailed in the design document under Missions and Sub-Plots.

The player has an alignment toward each NPC alliance, as well as a reputation. As reputation increases through successful finishing of alliance missions, promotions are awarded. Alignment is used to determine the reaction of NPC ships and planet side NPCs toward the player. Under this system, even if the player has risen high in a NPC alliance, if his actions cause his alignment to change, the alliance will turn against him both in space and in their dealings with him. A player can restore his alliance by performing actions beneficial to the alliance such as destroying the alliance's enemies. However, while his alignment is against the alliance, he gains no reputation from these actions until his name is restored. Each time the player falls out of favor with an alliance, it will be harder to get back in their good graces.

NPC Alliances also affect players who belong to one or more of them. These effects can be positive or negative factors affecting the player depending on the alliance. These effects typically grow in magnitude as the player advances in his alliance. These need to be individually implemented as their effects have no common denominator between them all.

The NPC alliances provide both random missions and scripted missions in addition to the main plot missions of the game. Random missions have an internal point value based on their assessed difficulty, which will be added to the player's reputation for that alliance upon completion of the mission. When the player has gotten enough reputation points to advance to the next rank in the alliance, he is given a scripted mission or series of missions. Upon completion of the scripted missions, he is raised to the next rank in the alliance.

NPCs

NPC characters in the single player game exist to give out both plot missions and random missions. In the multiplayer game, they only give out random missions, as there is no scripted plot for the multiplayer game. Each NPC will be governed in their reactions to each player by an alignment factor (i.e. a merchant NPC will not like to deal with pirates), which will also be affected by the player's reputation. In the single player game, they will also be influenced by the plot line and the events that happen therein. Plot NPCs will have a data structure containing their remarks and dialog for each plot point. Others (random mission giving) NPCs will have a data structure containing their reactions to players of varying alignment orientation relative to the NPC.

The Ship

The ship object inside of *Privateer 3* will be near identical to the ship system developed for *Wing Commander: Prophecy*. Each ship has a power supply, which the player will be able to upgrade, then all of the "powered systems" will be plugged into this power supply. Some ships can even hold multiple power plants, so if one goes out the backup comes on line. Further, having this weak link gives the player an additional way to disable another player's ship.

Confidential

Since the ship is just another 3D object it inherits all the properties of a real object including mass, speed, acceleration, and so on. Basically the ship is a "powered 3D Object". This ship receives virtual input to change its pitch, yaw, roll, and acceleration. One of the advantages here is that any "powered 3D" object can be controlled by a player. For example, a remote drone could be launched and the player could "pilot" it using the remote's virtual input control.

The Power Plant & Powered Systems

The power plant code that was used in *Wing Commander: Prophecy* will be used in very much the same way. The power plant will hold a list of powered systems like radar, lasers, shields, and scanners that are in the ship and distribute the power to them once per game update.

The power plant provides the ability to dynamically alter the percentages of power sent to the various systems. Raising the percentage given to one system will proportionally reduce the percentages sent to the remaining systems. An exception to this rule is for "locked" systems. The power plant can lock a particular system so that changes made for other systems will not effect the amount of power being sent to the locked system.

Each system that uses power will be derived from the PoweredSystem class. This way we keep the hierarchical paradigm of the power plant virtually running the ship systems. The power plant goes through each ship system changing the plant power into ship system power, so that the system can do its operation.

We are using all of the ship systems that *Prophecy* used (like engines, guns, missiles, repair, and radar) and adding on a couple more including new scanners, mine launchers, ECM units, and weapon enhancers. Each of these new systems will be coded the same way the original systems were coded.

The Cockpit & Gauges

The cockpit system displays gauges to the screen. These gauges represent variables inside of the game. This means all gauges in the cockpit HUD will mean something. A gauge could represent fuel, gun power, shield damage levels, or radar.

Most of the gauges will be flat 3D images. The changes in the gauges will primarily be reflected in a change in the 3D mesh. For example, to represent the gun power strength an artist might put together a series of little boxes to represent the full strength. When the power for the guns decrease, the program will key frame the next mesh that represents the resulting power level. In this way we do not have to incur multiple bitmap images in memory, rather we simply use the same texture with a different flat 3D mesh frame. We also inherit a number of 3D polygon attributes like sizing, rotation, and translucency.

The cockpits will be designed through the same XMIFF format that *Prophecy* used. This allows us to use the same type of HUDs they used in *Prophecy*.

Confidential

Origin, Systems Inc.

Technical Design Document

Concepts: Miscellaneous Game engines

Game Shell & Engine

This is the main loop that every instances of the engine handler goes through. Trivial as it may sound, it has to call each schedulers and handlers in correct order.

If the instances are scheduled incorrectly, the game may cause slow down, miss a VBL, timing and speed (synchronize), drawing order.

Threader

Creates and manages Windows based threads that handles such as music streaming, timing, network management, etc. If too many threads are created, Windows will slow down, and therefore, we'd have to profile and determine how much threads we can create before we begin to see loss in frame rates. Considerations of threads at this moment are:

- Network event
- Music streaming
- Sound effects
- File streaming

IOWin

IOWin, based on Jeff Grills' *Vision* game engine, gives the programmers the ease of controlling the update of each objects, items, instances, as well as drawing of these instances at an appropriate time to manage each instance individually. It manages chain of events created in manner of queue for events such as:

- Mouse events
- Joystick events
- Keyboard

These events can be passed on to the next IOWin in the queue or may be blocked in such events as update() and draw(). This makes it easier to manage instances such as hiding the mouse movements to the next queue in the list if necessary.

Confidential

IOWinManager and Event Processing

All game engine low level input events are sent to the IOWinManager and queued. The IOWinManager keeps a stack of game processes (each game process is called an IOWin). Each of the queued events are passed through the IOWin stack. The top IOWin receives the event and determines whether or not to send the event down to the next IOWin. Once control is returned to the top IOWin (assuming the event had been passed down), or the top IOWin has decided not to pass the event down, the top IOWin responds appropriately to the event.
This type of setup allows us to easily bring up an option screen by adding a new IOWin to the top of the IOWin stack. It is guaranteed that there will be no input events inappropriately processed by any of the other IOWins on the stack because the event is never passed down to them.

Item/Script scheduler

Controls the order, priorities, and incidents of items that gets created for each level. The concerns that we should watch for are issues such as number of maximum items that can be created in a level, and loss of frame rate due to too many items created in the world. Solutions such as considerations of alternating between even and odd item ID to be processed per iterations, or scheduling items such that items that are idling may be triggered only upon flags that triggers it. Each item are processed by calling the *script manager* which executes the item scripts written by the designers.

Event scheduler

Events based on synchronization and event scheduler schedules external events that are not interactive directly with the players. Event scheduler is also the interface that connects the player's input, AI, and the TCP (external input) as if they are same entities as shown in the *figure 3* below.

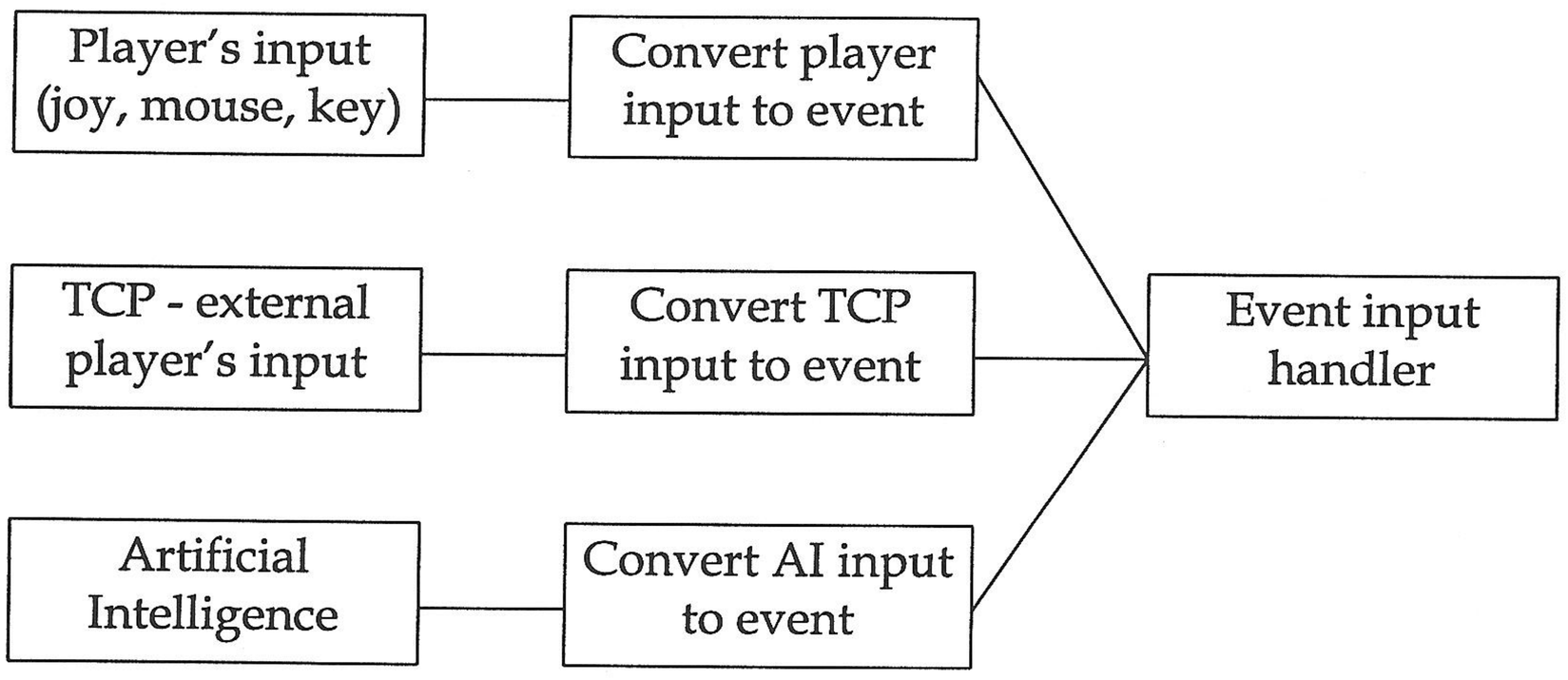


Figure 1 Event handler interfacing

As the above figure shows, by following this method, we can easily attach and detach an AI's slot for an external TCP player and vice versa, making it easy to interface and "plug in" slots for substitutions. A simple example of this may be if a TCP player looses connection for a moment, an

Confidential

AI can take over its place and soon as the player reconnects, the AI can switch over to the original state and will be totally transparent to the local player.

Script manager

Our script handler will be based on a C++ class which will be carefully constructed to allow us to write a simple C code without the knowledge of the game engine, but with the awareness of the external AI functions. With this approach, and with caution, it would allow designers to help write scripts to generate logics to make AI and similar functions useful to the game.

Scene (level) manager

A mission manager initializes mission data specific for the mission to be initiated. After memory blocks such as texture map space, static sound effects, and any blocks of memory that is necessary for the scene (or level) is allocated and initialized, mission script is launched.

Block spaces required for the level to be initiated is pre-calculated before the scene is launched. A careful method is devised so that the designers who is working on a level has the control to allocate the memory space for the level, yet be limited and controlled so that they don't over allocate the static memory space. They will be provided with a configurations file which contains status of what should be pre-allocated and/or loaded into a static memory.

Memory manager

Vision game engine memory manager by Jeff Grills' will be utilized for our memory manager. The carefully thought out memory manager has the capabilities of keeping track of static and dynamic memories that were allocated by request, and will warn the programmer if and when they have forgotten to deallocate the memory when the object has been removed.

Graphics

Texture map - 8 Megs
Wire frame - 4 to 8 Megs per wire

Music and sound

Sound effects may range from 5K to 500K per sound depending on its sampling methods. It will have to be carefully monitor which sound effects will stay static and which will be dynamic. These configurations will vary per scenes. Common sound effects such as bullet, shield hits, ambient noises, and explosions are some of the common sound effects that should be carefully sampled to its high quality yet minimal size for they will be statically allocated before each flying scene.

Music file can only be loaded one at a time, and therefore, it will attempt to allocate per music file basis.

Sound allocation - 2 to 4 Megs

Confidential

Misc.

Items such as fonts, object scripts and instances of objects are some of the other entities that should be pre-allocated before some of the scenes. There will be a cap limit as to number of objects but the cap should be dynamic according to the speed of the CPU. Faster the CPU is, more that it can handle.

Language manager

To handle multiple language support for other language then English, Hugh David and the *Prophecy* team has devised a method to handle multi-language fonts. With their language manager, it will ease the method of porting over to other language, via data driven method.

Palette manager

The basic functionality of what a palette manager should handle are:

- *Fade in and out* - fade to and from a current palette to destination palette, or basic fade to black.
- *Palette swapping* - swap from current palette to new set of palette.
- *Palette shifting* - rotate-shift from a given range of palette to give an effect of color cycle for inexpensive animation effect with inside the texture map. This feature is optional due to the fact that sprite animation are inexpensive to implement.

When the texture uses 8bit palette each pixels are indexed to the palette, while in a 16bit palette mode, each pixels are represented as a RGB palette value. No matter which method it is, the colors are adjusted when transferring the texture to the video RAM pixel by pixel (actually DWORD and WORD order).

Collision scheduler

One of the most time consumption is due to collision checking with multiple objects. More objects there are the more comparison and handling must be cared for. Even with a simple iteration, it will observe checking of:

$$\sum_{l=1}^{n-1} l$$

Where n = number *of objects to check for collision*. Object *1* is checking collisions against object 2 through n and object 2 is checking collisions against object *3* through n, and so on, all we have to do is check for one object passed itself. Hence the equation above. For example, if there are *10* objects, with calculations above, we are looking at 45 collision checks. This is better then O^2 calculation, which will result in *100* checks if it has to check for itself as well. This is probably the best that we can do without doing extra checking using 2 loops one inside the other. Our solutions should be to resolve this via algorithms and methods. One way that we can approach this is to schedule the time of collision that a single object may occur against the others, and only check when it is

Confidential

assumed to collide. This will reduce the number of items to collide for all it needs to do is check when it has predicted the collision.
Another method that we may approach this is to break the world into sections (or sectors) and only check for collisions that may occur with inside that sector.
Perhaps with both methods combined together, we might even be able to optimize more.

Physics

It is important to emulate as close to real space physics as theories may allow us to take will increase the game quality. In order to do this, each objects in space should require parameters and calculations such as:

- Velocity
- Accelerations
- Mass

An object's flight characteristics are defined by the following parameters:

- Form factors
- Mass
- Engine thrust (longitudinal trust)
- Lateral thrust
- Vertical thrust
- Mass roll rate

The parameters are unique to each ship and are assigned by the designers. Additionally, thrust can be increased when the player upgrades to more powerful engines. With these parameters and the acceleration formula, one can have maximum speed and acceleration that is inversely proportional to the speed i.e. it takes longer to accelerate from 800 to 900 mph than from 0 to 100 mph. The lateral and vertical thrust parameters specify the maneuverability of a ship. Mass effects the overall response and the maximum speed of the ship. The mass roll rate specifies how quickly the ship can be rolled.

Missiles can be modeled in the same way. Lateral and vertical thrust are applied based on the location of the target in order to steer the missiles towards the target.

Physics module will be on a thread of its own to compensate on the dead-reckoning incidents.

Control of the ship:

There are two modes to control ships:

- Fighter mode
- Weightless mode

Fighter mode is the mode in flight simulators. In this mode, the player controls pitching, yawing and rolling. The difference is that the throttle does not control the trust but sets the desired speed. Once the player sets the desired speed, the ship computer will automatically try to reach that speed as fast as possible. This makes it easier for non 'fighter jocks' to control the speed.

Confidential

The weightless mode can be best described as the 'asteroid' mode. This mode allows the player to rotate the orientation of the ship without changing the flight path. Only by using engaging the engine will the flight path change.

Collision handler

When 2 objects collide, it should react as realistically as possible. We would need to define several parameters and agree on a base class that we can implement to all objects that would inherit this. Such as coordinates, velocity, accelerations, mass, maximum velocity (this may be unrealistic, but required for collision scheduler), maximum acceleration, torque, friction, density, elasticity, etc.

Movie Player

We will be implementing EAC's UV2 codec written by Martin Griffiths for *Privateer 3*. Jeff Grills has successfully intergrated UV2 to the *Vision* engine, and thus reduces our workload of resolving the formats and methods. If time permits, we would also like to investigate on implementing EAC's MADCOW codec, which has a potential of cleaner video.

Object layered above movies

To save movie storage space, we would like to propose a method of generic movie playing in the background and have a 3D spaceship in the foreground to simulate a movie of different spaceships flying in same background. The obstacles that we may face approaching this method is that if the background movie is not scripted well, there will be problems with lighting and color matching problem.

Dynamic cinematic

Another approach to save movie storage space, and to make give an effect of wealth of cinematic is to play sequences of section movies seamlessly to simulate several different movies. In order to make sure it is very seamless, the music of the movie must be designed properly in a way that threading of movie will allow us to play second movie immediately after the first and feel like it's a single piece of movie.

Sound System

Music in our game will be interactively streamed using *Vision* game engine written by Charles Karpiak. The utility used will be Magi development system. The music and sound driver is capable of looping, but cannot jump from one MGI file to another. A simple solution to resolve this is to fade out from the current music and fade into another. This had to be mentioned since the *Design Document* mentions the use of playing different music for different sectors of the universe.

Currently, the *Vision* game engine's compression is close to 4:1 ratio. Many other strategies should be considered for data compression such as recording some sound effects in mono instead of stereo,

Confidential

experiments to see the sound quality recorded on 8bit mode rather then 16bit mode for some sound effects, recording at 22KHz rather then 44.1KHz may also be considered to reduce the storage space issue.

3DSound

According to Chuck, currently, the *Vision* game engine gets a hit of 10% loss in frame rate when 3Dsound is played.

3Dsound should be recorded in mono since the whole purpose of using this is to give an effect of switching from one speaker to another of the object that the sound is attached to.

Input Devices

The objectives of input devices is to

- Allow input from input devices that are supported by Win95 DirectInput
- Convert the input into data format that *event handler* can utilize as a form of input.

It is essential to have a virtual input mapping, a dynamic key macros (as well as mouse and joystick) so that the users can re-map their input to what they feel would help the game play much easier. It should not be permanently defined to force the players to use the input map that we define. In order to do so, we will devise a dynamic input map which has pre-defined mapping to a functions but may be redefined to player's liking.

Input Macro Recorder and player

Input macro records and plays back for purposes such as demo screen. This is very useful for tutorial purposes if we do decide to implement such features.

User Input Manager

DirectInput

The game engine uses Microsoft's DirectInput in order to obtain raw input device values. Currently, the game has support for keyboard, mouse, and two axis, two button joystick input. The Mouse, Keyboard, and Joystick classes each query the operating system for the raw values for the device, translates the raw values into appropriate game engine low level input events, and then sends the events to the IOWinManager for distribution to game processes.

Input Device Consolidation

The Gameflow portion of *Privateer3* makes use of a cursor, and is most intuitively controlled via the mouse. However, it is preferable to allow the player to use whichever input device they choose to move the cursor around the screen. This would be difficult to do if it was necessary to specifically code for every input device type, so the game engine has implemented the ability to consolidate input devices.

Confidential

In other words, the keyboard, joystick, and whatever other devices we choose to add to the game can have their input mapped to the mouse. By doing so, the player can freely use whatever device they choose to move the cursor around, while in the game code, it is only necessary to respond to mouse events. This greatly simplifies the Gameflow code, and makes the engine much more flexible.

Key Mapping

One more level of input event exists, known as the Keymap. The Keymap translates low level game input events from the keyboard, mouse, and joystick into high level game events such as Pitch, Yaw, Roll, and Fire. High level game events are queued up, and the systems responsible for responding to these events scan the queue and perform the appropriate actions.
The benefit of this type of system means that the Spaceflight game code refers only to high level game events. This makes the ship systems code much cleaner and easier to maintain. An additional benefit is the ability to re-map input devices to different game controls without the game code needing to know anything about it. As far as the Spaceflight systems are concerned, everything remains the same.

Confidential

Chapter 6

6

Origin, Systems Inc.

Technical Design Document

Task list and schedules

Schedules

Confidential

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	**Alpha**	0d	Mon 5/11/98	Mon 5/11/98		
2	*E3 Demo*	0d	Mon 5/25/98	Mon 5/25/98		
3	**Beta**	0d	Mon 7/6/98	Mon 7/6/98		
4	**Final**	0d	Mon 8/31/98	Mon 8/31/98		
5	*ECTS Demo*	0d	Mon 9/7/98	Mon 9/7/98		
6	QA Sign Off	0d	Mon 9/14/98	Mon 9/14/98		
7	**Ship**	0d	Mon 9/28/98	Mon 9/28/98		
8						
9	Screen 3d	10d	Thu 10/9/97	Wed 10/22/97		Andrew Tjew
10	Screen 2d	2d	Thu 10/23/97	Fri 10/24/97	9	Andrew Tjew
11	Event Scheduler	10d	Thu 10/9/97	Wed 10/22/97		Tony Ikeda
12	Game/Network decoupled model	7d	Thu 10/9/97	Fri 10/17/97		Allen Jackson
13	Network Game Types - Phase 1	5d	Mon 10/20/97	Fri 10/24/97	12	Allen Jackson
14	Communications Model	7d	Mon 10/27/97	Tue 11/4/97	13	Allen Jackson
15	Winsock API	3d	Wed 11/5/97	Fri 11/7/97	14	Allen Jackson
16	Asynchronous Server/Client Model	5d	Mon 11/10/97	Fri 11/14/97	15	Allen Jackson
17	Channels	2d	Mon 11/17/97	Tue 11/18/97	16	Allen Jackson
18	Channels Handling	2d	Wed 11/19/97	Thu 11/20/97	17	Allen Jackson
19	Dealing with new clients in events	1d	Fri 11/21/97	Fri 11/21/97	18	Allen Jackson
20	Network Game Types - Phase 2	3d	Mon 11/24/97	Wed 11/26/97	19	Allen Jackson
21	Item Scheduler	3d	Thu 10/23/97	Mon 10/27/97	11	Tony Ikeda
22	Timing an event on multiple clients	5d	Mon 12/1/97	Fri 12/5/97	20	Allen Jackson
23	Packet's Content	3d	Mon 12/8/97	Wed 12/10/97	22	Allen Jackson
24	Tokenization	4d	Thu 12/11/97	Tue 12/16/97	23	Allen Jackson
25	Grouping the Clients	10d	Wed 12/17/97	Fri 1/9/98	24	Allen Jackson
26	Starting a game server	5d	Tue 10/28/97	Mon 11/3/97	21	Tony Ikeda
27	Planetary Events	20d	Thu 10/9/97	Wed 11/5/97		Ala Diaz
28	Planetary Economics	5d	Thu 11/6/97	Wed 11/12/97	27	Ala Diaz
29	Hangar - Main Screen	2d	Thu 11/13/97	Fri 11/14/97	28	Ala Diaz
30	Port Government Influences	5d	Mon 11/17/97	Fri 11/21/97	29	Ala Diaz
31	Main Port Screen	1d	Mon 11/24/97	Mon 11/24/97	30	Ala Diaz
32	Commodity Exchange	4d	Tue 11/25/97	Tue 12/2/97	31	Ala Diaz
33	Main Information Exchange (1P)	2d	Wed 12/3/97	Thu 12/4/97	32	Ala Diaz
34	Planet #01	2d	Mon 10/27/97	Tue 10/28/97	10	Andrew Tjew
35						

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
36						
37						
38	Math	2d	Wed 10/29/97	Thu 10/30/97	34	Andrew Tjew
39	Camera	1d	Fri 10/31/97	Fri 10/31/97	38	Andrew Tjew
40	Control	1d	Mon 11/3/97	Mon 11/3/97	39	Andrew Tjew
41	Network Game Types - Phase 3	4d	Mon 1/12/98	Thu 1/15/98	25	Allen Jackson
42	Base Model	1d	Tue 11/4/97	Tue 11/4/97	40	Andrew Tjew
43	Various 3d Drivers	5d	Wed 11/5/97	Tue 11/11/97	42	Andrew Tjew
44	Object Hierarchy	2d	Wed 11/12/97	Thu 11/13/97	43	Andrew Tjew
45	Threader	7d	Tue 11/4/97	Wed 11/12/97	26	Tony Ikeda
46	Palette Manager	5d	Thu 11/13/97	Wed 11/19/97	45	Tony Ikeda
47	Alias Model Convert	1d	Fri 11/14/97	Fri 11/14/97	44	Andrew Tjew
48	Alias Model Viewer	3d	Mon 11/17/97	Wed 11/19/97	47	Andrew Tjew
49	Texture Bitmap & Pallette Viewer	1d	Thu 11/20/97	Thu 11/20/97	48	Andrew Tjew
50	Object Hierarchy Viewer	3d	Fri 11/21/97	Tue 11/25/97	49	Andrew Tjew
51	Game server gospel	4d	Fri 1/16/98	Wed 1/21/98	41	Allen Jackson
52	SpaceFlight - Camera	5d	Fri 12/5/97	Thu 12/11/97	33	Ala Diaz
53	Interstellar Trade Routes	5d	Fri 12/12/97	Thu 12/18/97	52	Ala Diaz
54	SpaceFlight - Meters	2d	Fri 12/19/97	Mon 12/22/97	53	Ala Diaz
55	SpaceFlight (H.U.D.) - Player Damage Display	3d	Tue 12/23/97	Tue 1/6/98	54	Ala Diaz
56	SpaceFlight - Power Distribution Display	3d	Wed 1/7/98	Fri 1/9/98	55	Ala Diaz
57	SpaceFlight - In-Flight Nav. Map	5d	Mon 1/12/98	Fri 1/16/98	56	Ala Diaz
58	Jump Points	2d	Wed 11/26/97	Mon 12/1/97	50	Andrew Tjew
59	Cockpit & Turret Screen	2d	Mon 1/19/98	Tue 1/20/98	57	Ala Diaz
60	Nav. Map	2d	Thu 11/20/97	Fri 11/21/97	46	Tony Ikeda
61	Nav. Map - Path Finding	5d	Mon 11/24/97	Tue 12/2/97	60	Tony Ikeda
62	Nav. Map - Change Views	3d	Wed 12/3/97	Fri 12/5/97	61	Tony Ikeda
63	Nav. Map - Port Info	5d	Mon 12/8/97	Fri 12/12/97	62	Tony Ikeda
64	Nav. Map - Mission Specs	3d	Mon 12/15/97	Wed 12/17/97	63	Tony Ikeda
65	Nav. Map - Mission Available	3d	Thu 12/18/97	Mon 12/22/97	64	Tony Ikeda
66	Nav. Map - Player Diary	2d	Tue 12/23/97	Mon 1/5/98	65	Tony Ikeda
67	Nav. Map - Map Rotation	2d	Tue 1/6/98	Wed 1/7/98	66	Tony Ikeda
68	Virtual Input	5d	Thu 1/8/98	Wed 1/14/98	67	Tony Ikeda
69	Virtual Input Macro	5d	Thu 1/15/98	Wed 1/21/98	68	Tony Ikeda
70						

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
106	Scene Manager	5d	Thu 3/26/98	Wed 4/1/98	105	Tony Ikeda
107	Server-Client Relationship	5d	Thu 2/12/98	Wed 2/18/98	104	Allen Jackson
108	Determining a Hit	5d	Thu 2/19/98	Wed 2/25/98	107	Allen Jackson
109	Dealing with lost clients connections	2d	Thu 4/2/98	Fri 4/3/98	106	Tony Ikeda
110	Collision Scheduler	7d	Mon 4/6/98	Tue 4/14/98	109	Tony Ikeda
111	Optimizing Space Combat Sequences (Chainin	4d	Thu 2/26/98	Tue 3/3/98	108	Allen Jackson
112	Optimizing Space Combat Sequences (holes in	4d	Wed 3/4/98	Mon 3/9/98	111	Allen Jackson
113	Physics	15d	Wed 4/15/98	Tue 5/5/98	110	Tony Ikeda
114	Using history of Coordinates	5d	Tue 3/10/98	Mon 3/16/98	112	Allen Jackson
115	using history of Coordinates (Switching AI to vi	5d	Tue 3/17/98	Mon 3/23/98	114	Allen Jackson
116	Collision Handler	5d	Wed 5/6/98	Tue 5/12/98	113	Tony Ikeda
117	Reconciliation Methods	15d	Tue 3/24/98	Mon 4/13/98	115	Allen Jackson
118	Grouping Players to Decrease Bandwidth	5d	Wed 5/13/98	Tue 5/19/98	116	Tony Ikeda
119	Using AI to decrease bandwidth	2d	Wed 5/20/98	Thu 5/21/98	118	Tony Ikeda
120	Collision	5d	Tue 12/2/97	Mon 12/8/97	58	Andrew Tjew
121	How to find a game server	1d	Fri 5/22/98	Fri 5/22/98	119	Tony Ikeda
122	SpaceFlight - Target Damage Display & Read	5d	Mon 3/16/98	Fri 3/20/98	98	Ala Diaz
123	SpaceFlight - Gun and Missle Display	4d	Mon 3/23/98	Thu 3/26/98	122	Ala Diaz
124	SpaceFlight - Target Reticule	5d	Fri 3/27/98	Thu 4/2/98	123	Ala Diaz
125	SpaceFlight - ITTS	4d	Fri 4/3/98	Wed 4/8/98	124	Ala Diaz
126	SpaceFlight - Target Acquisition Scanner	5d	Thu 4/9/98	Wed 4/15/98	125	Ala Diaz
127	SpaceFlight - Communications and Messages	5d	Thu 4/16/98	Wed 4/22/98	126	Ala Diaz
128	SpaceFlight - Defensive Measures	2d	Thu 4/23/98	Fri 4/24/98	127	Ala Diaz
129	M.E.D. Modifications	20d	Thu 10/9/97	Wed 11/5/97		Andy Sommers
130	Missiles and Bullets	5d	Tue 12/9/97	Mon 12/15/97	120	Andrew Tjew
131	Weapon #01	2d	Tue 12/16/97	Wed 12/17/97	130	Andrew Tjew
132	Weapon #02	2d	Thu 12/18/97	Fri 12/19/97	131	Andrew Tjew
133	Combat AI - Porting from WCP	5d	Thu 1/22/98	Wed 1/28/98	77	Hugh David
134	Combat AI - Enemy Recognition	3d	Thu 1/29/98	Mon 2/2/98	133	Hugh David
135	Combat AI - Optimizations	10d	Tue 2/3/98	Mon 2/16/98	134	Hugh David
136	Combat AI - Trade Route Following and Attitud	10d	Tue 2/17/98	Mon 3/2/98	135	Hugh David
137	Combat AI - Multi-Ship Maneuvers	15d	Tue 3/3/98	Mon 3/23/98	136	Hugh David
138	Combat AI - Combat Maneuvers	15d	Tue 3/24/98	Mon 4/13/98	137	Hugh David
139	Combat AI - Avoidance Maneuvers	10d	Tue 4/14/98	Mon 4/27/98	138	Hugh David
140	Combat AI - Misc. Maneuvers	5d	Tue 4/28/98	Mon 5/4/98	139	Hugh David

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
71						
72						
73	Packet Optimizations	2d	Thu 1/22/98	Fri 1/23/98	51	Allen Jackson
74	Individual Benefits/Detriments	4d	Thu 1/22/98	Tue 1/27/98	69	Tony Ikeda
75	Arena (1p)	2d	Thu 1/15/98	Fri 1/16/98		Hugh David
76	Arena (1p) Choose Opponent	2d	Mon 1/19/98	Tue 1/20/98	75	Hugh David
77	Arena (1p) Choose Arena	1d	Wed 1/21/98	Wed 1/21/98	76	Hugh David
78	Landing Sequence#01	3d	Mon 1/26/98	Wed 1/28/98	73	Allen Jackson
79	Hangar - Ship Sales	3d	Wed 1/21/98	Fri 1/23/98	59	Ala Diaz
80	Hangar - Sell	3d	Mon 1/26/98	Wed 1/28/98	79	Ala Diaz
81	Hangar - Repair Bay	3d	Thu 1/29/98	Mon 2/2/98	80	Ala Diaz
82	Hangar - Equipment Sales	3d	Tue 2/3/98	Thu 2/5/98	81	Ala Diaz
83	Hangar - Dry Dock	2d	Fri 2/6/98	Mon 2/9/98	82	Ala Diaz
84	Information Exchange (1p) - News Wire	4d	Tue 2/10/98	Fri 2/13/98	83	Ala Diaz
85	Information Exchange (1p) - Ships	3d	Mon 2/16/98	Wed 2/18/98	84	Ala Diaz
86	Information Exchange (1p) - Want Ads	4d	Thu 2/19/98	Tue 2/24/98	85	Ala Diaz
87	Information Exchange (1p) - Sports Waging	3d	Wed 2/25/98	Fri 2/27/98	86	Ala Diaz
88	Mission Builder	5d	Wed 1/28/98	Tue 2/3/98	74	Tony Ikeda
89	Mission System	5d	Wed 2/4/98	Tue 2/10/98	88	Tony Ikeda
90	Mission System - Suspend & Reinstate	3d	Wed 2/11/98	Fri 2/13/98	89	Tony Ikeda
91	Mission Generation System	5d	Mon 2/16/98	Fri 2/20/98	90	Tony Ikeda
92	Mission Timing	3d	Mon 2/23/98	Wed 2/25/98	91	Tony Ikeda
93	Mission Resolution	5d	Thu 2/26/98	Wed 3/4/98	92	Tony Ikeda
94	Main Information Exchange (Multi) - News Wire	2d	Mon 3/2/98	Tue 3/3/98	87	Ala Diaz
95	Main Information Exchange (Multi) - Personnel	2d	Wed 3/4/98	Thu 3/5/98	94	Ala Diaz
96	Main Information Exchange (Multi) - Player Info	2d	Fri 3/6/98	Mon 3/9/98	95	Ala Diaz
97	Main Information Exchange (Multi) - Alliance In	2d	Tue 3/10/98	Wed 3/11/98	96	Ala Diaz
98	Main Information Exchange (Multi) - Alliance A	2d	Thu 3/12/98	Fri 3/13/98	97	Ala Diaz
99						
100						
101						
102	Scheduling AI to sync. to clients	4d	Thu 1/29/98	Tue 2/3/98	78	Allen Jackson
103	Synchronizations (on going)	1d	Wed 2/4/98	Wed 2/4/98	102	Allen Jackson
104	Network Game Types - Phase 4	5d	Thu 2/5/98	Wed 2/11/98	103	Allen Jackson
105	Script Manager	15d	Thu 3/5/98	Wed 3/25/98	93	Tony Ikeda

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
176	Model Exhausts (Afterburner)	1d	Mon 2/16/98	Mon 2/16/98	175	Andrew Tjew
177	Derelict #1 (Damaged Ship)	3d	Wed 5/6/98	Fri 5/8/98	169	Allen Jackson
178	Polygonal Explosions	5d	Tue 2/17/98	Mon 2/23/98	176	Andrew Tjew
179	Head Turning and Shaking	2d	Tue 2/24/98	Wed 2/25/98	178	Andrew Tjew
180	Heat and Smoke Damage	5d	Thu 2/26/98	Wed 3/4/98	179	Andrew Tjew
181	Decoy (Thermal)	1d	Mon 5/18/98	Mon 5/18/98	158	Hugh David
182	Cargo #1	1d	Tue 5/19/98	Tue 5/19/98	181	Hugh David
183	Cargo #2	1d	Wed 5/20/98	Wed 5/20/98	182	Hugh David
184	Particles	5d	Thu 3/5/98	Wed 3/11/98	180	Andrew Tjew
185	Shock Wave	3d	Thu 3/12/98	Mon 3/16/98	184	Andrew Tjew
186	Space Dust	5d	Tue 3/17/98	Mon 3/23/98	185	Andrew Tjew
187	Lens Flare	2d	Tue 3/24/98	Wed 3/25/98	186	Andrew Tjew
188	Black Outs and Red Outs	1d	Thu 3/26/98	Thu 3/26/98	187	Andrew Tjew
189	Ship's log/Player Diary	2d	Fri 6/12/98	Mon 6/15/98	152	Ala Diaz
190	Dynamic Cinematic	2d	Thu 6/18/98	Fri 6/19/98	172	Tony Ikeda
191						
192						
193						
194	Running p3 on the internet	5d	Mon 6/22/98	Fri 6/26/98	190	Tony Ikeda
195	S. Ship, Nemesis (Player #3)	2d	Mon 5/11/98	Tue 5/12/98	177	Allen Jackson
196	Station, Helios (Industrial)	4d	Fri 3/27/98	Wed 4/1/98	188	Andrew Tjew
197	Ship Debri #1	1d	Thu 4/2/98	Thu 4/2/98	196	Andrew Tjew
198	Escape Pod	2d	Thu 5/21/98	Fri 5/22/98	183	Hugh David
199	Asteroid	2d	Fri 4/3/98	Mon 4/6/98	197	Andrew Tjew
200	Load/Save ATM Machine	2d	Tue 6/16/98	Wed 6/17/98	189	Ala Diaz
201	Arena	10d	Tue 5/26/98	Mon 6/8/98	198	Hugh David
202	Power Up (flag) #1	1d	Tue 6/9/98	Tue 6/9/98	201	Hugh David
203						
204						
205						
206	S. Ship, Predator (Player #2)	2d	Wed 5/13/98	Thu 5/14/98	195	Allen Jackson
207	Saving Data on Server	2d	Mon 6/29/98	Tue 6/30/98	194	Tony Ikeda
208	S. Ship, Aurora (Player #4)	2d	Fri 5/15/98	Mon 5/18/98	206	Allen Jackson
209	S. Ship, Wyvern (Player #5)	2d	Tue 5/19/98	Wed 5/20/98	208	Allen Jackson
210	L. Fighter, Demon (Player Ship #1)	2d	Tue 4/7/98	Wed 4/8/98	199	Andrew Tjew

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
211	S. Ship, Gryphon (Player #6)	2d	Thu 5/21/98	Fri 5/22/98	209	Allen Jackson
212	Cruiser, Avenger (Confed.)	2d	Thu 4/9/98	Fri 4/10/98	210	Andrew Tjew
213	L. Fighter, Dralthi V (Kilrathi)	2d	Mon 4/13/98	Tue 4/14/98	212	Andrew Tjew
214	L. Fighter, Javelin (Mercenary)	2d	Wed 4/15/98	Thu 4/16/98	213	Andrew Tjew
215	Freighter, Drayman (Merchant)	3d	Fri 4/17/98	Tue 4/21/98	214	Andrew Tjew
216	M. Freighter, Sathori (Merchant)	3d	Wed 4/22/98	Fri 4/24/98	215	Andrew Tjew
217	Station, Hermes (Mining)	4d	Mon 4/27/98	Thu 4/30/98	216	Andrew Tjew
218	M. Freighter, Intolerance (Outcast)	2d	Tue 5/26/98	Wed 5/27/98	211	Allen Jackson
219	L. Fighter, Piranha (Confed.)	2d	Thu 5/28/98	Fri 5/29/98	218	Allen Jackson
220	M. Fighter Traket (Kilrathi)	2d	Mon 6/1/98	Tue 6/2/98	219	Allen Jackson
221	H. Fighter, Dragon (Mercenary)	2d	Wed 6/3/98	Thu 6/4/98	220	Allen Jackson
222	H. Fighter, Fury (Confed.)	2d	Fri 6/5/98	Mon 6/8/98	221	Allen Jackson
223	Station, Hepheatus (Retextured for Pirates)	4d	Fri 5/1/98	Wed 5/6/98	217	Andrew Tjew
224	Cruiser, Kamekh (Kilrathi)	2d	Thu 5/7/98	Fri 5/8/98	223	Andrew Tjew
225	Destroyer, Typhoon (Mercenary)	2d	Mon 5/11/98	Tue 5/12/98	224	Andrew Tjew
226	Station, Ares (Military)	4d	Wed 5/13/98	Mon 5/18/98	225	Andrew Tjew
227	Station, Kronos (Kilrathi)	4d	Tue 5/19/98	Fri 5/22/98	226	Andrew Tjew
228	Station, Hades 5th Plot (Kilrathi)	4d	Tue 5/26/98	Fri 5/29/98	227	Andrew Tjew
229	Station, Athena (Science)	4d	Mon 6/1/98	Thu 6/4/98	228	Andrew Tjew
230	SH. Fighter, Vampire (Confed.)	2d	Fri 6/5/98	Mon 6/8/98	229	Andrew Tjew
231	H. Fighter, Vaktoth (Kilrathi)	2d	Wed 7/1/98	Thu 7/2/98	207	Tony Ikeda
232	M. Multi-Role, Hurricane (Mercenary)	2d	Fri 7/3/98	Mon 7/6/98	231	Tony Ikeda
233	H. Multi-Role, Armageddon (Outcast)	2d	Tue 7/7/98	Wed 7/8/98	232	Tony Ikeda
234	L. Freighter, Pegasus (Merchant)	3d	Thu 7/9/98	Mon 7/13/98	233	Tony Ikeda
235	H. Bomber, Devastator (Confed.)	2d	Tue 7/14/98	Wed 7/15/98	234	Tony Ikeda
236	S. Fighter, Strakha (Kilrathi)	2d	Thu 7/16/98	Fri 7/17/98	235	Tony Ikeda
237	Battlecruiser, Manowar (Confed.)	2d	Tue 6/9/98	Wed 6/10/98	222	Allen Jackson
238	H. Freighter, Tevis (Merchant)	2d	Thu 6/11/98	Fri 6/12/98	237	Allen Jackson
239	L. Fighter, Piranha (Textured for Pirates)	0.5d	Wed 6/10/98	Wed 6/10/98	202	Hugh David
240	L. Fighter, Dralthi V (Textured for Pirates)	0.5d	Wed 6/10/98	Wed 6/10/98	239	Hugh David
241	M. Fighter Traket (Textured for Kilrathi Pirates)	0.5d	Thu 6/11/98	Thu 6/11/98	240	Hugh David
242	H. Fighter, Vaktoth (Textured for Kilrathi Pirate	0.5d	Thu 6/11/98	Thu 6/11/98	241	Hugh David
243	S. Fighter, Strakha (Textured for Kilrathi Pirate	0.5d	Fri 6/12/98	Fri 6/12/98	242	Hugh David
244	H. Fighter, Dragon (Textured for Kilrathi Pirates	0.5d	Fri 6/12/98	Fri 6/12/98	243	Hugh David
245	Planet #02	1d	Tue 6/9/98	Tue 6/9/98	230	Andrew Tjew

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
246	Planet #03	0.5d	Wed 6/10/98	Wed 6/10/98	245	Andrew Tjew
247	Planet #04	0.5d	Wed 6/10/98	Wed 6/10/98	246	Andrew Tjew
248	Planet #05	0.5d	Thu 6/11/98	Thu 6/11/98	247	Andrew Tjew
249	Planet #06	0.5d	Thu 6/11/98	Thu 6/11/98	248	Andrew Tjew
250	Planet #07	0.5d	Fri 6/12/98	Fri 6/12/98	249	Andrew Tjew
251	Planet #08	0.5d	Fri 6/12/98	Fri 6/12/98	250	Andrew Tjew
252	Planet #09	0.5d	Mon 6/15/98	Mon 6/15/98	251	Andrew Tjew
253	Planet #10	0.5d	Mon 6/15/98	Mon 6/15/98	252	Andrew Tjew
254	Planet #11	0.5d	Tue 6/16/98	Tue 6/16/98	253	Andrew Tjew
255	Planet #12	0.5d	Tue 6/16/98	Tue 6/16/98	254	Andrew Tjew
256	Weapon #03	2d	Wed 6/17/98	Thu 6/18/98	255	Andrew Tjew
257	Weapon #04	2d	Fri 6/19/98	Mon 6/22/98	256	Andrew Tjew
258	Weapon #05	2d	Tue 6/23/98	Wed 6/24/98	257	Andrew Tjew
259	Weapon #06	2d	Thu 6/25/98	Fri 6/26/98	258	Andrew Tjew
260	Weapon #07	2d	Mon 6/15/98	Tue 6/16/98	244	Hugh David
261	Weapon #08	2d	Wed 6/17/98	Thu 6/18/98	260	Hugh David
262	Weapon #09	2d	Fri 6/19/98	Mon 6/22/98	261	Hugh David
263	Weapon #10	2d	Tue 6/23/98	Wed 6/24/98	262	Hugh David
264	Weapon #11	2d	Thu 6/25/98	Fri 6/26/98	263	Hugh David
265	Weapon #12	2d	Mon 6/29/98	Tue 6/30/98	264	Hugh David
266	Weapon #13	2d	Wed 7/1/98	Thu 7/2/98	265	Hugh David
267	Landing Sequence#02	1d	Mon 6/15/98	Mon 6/15/98	238	Allen Jackson
268	Landing Sequence#03	0.5d	Tue 6/16/98	Tue 6/16/98	267	Allen Jackson
269	Landing Sequence#04	0.5d	Tue 6/16/98	Tue 6/16/98	268	Allen Jackson
270	Landing Sequence#05	0.5d	Wed 6/17/98	Wed 6/17/98	269	Allen Jackson
271	Landing Sequence#06	0.5d	Wed 6/17/98	Wed 6/17/98	270	Allen Jackson
272	Landing Sequence#07	0.5d	Thu 6/18/98	Thu 6/18/98	271	Allen Jackson
273	Landing Sequence#08	0.5d	Thu 6/18/98	Thu 6/18/98	272	Allen Jackson
274	Landing Sequence#09	0.5d	Fri 6/19/98	Fri 6/19/98	273	Allen Jackson
275	Landing Sequence#10	0.5d	Fri 6/19/98	Fri 6/19/98	274	Allen Jackson
276	Landing Sequence#11	0.5d	Mon 6/22/98	Mon 6/22/98	275	Allen Jackson
277	Landing Sequence#12	0.5d	Mon 6/22/98	Mon 6/22/98	276	Allen Jackson
278	Landing Sequence#13	0.5d	Tue 6/23/98	Tue 6/23/98	277	Allen Jackson
279	Landing Sequence#14	0.5d	Tue 6/23/98	Tue 6/23/98	278	Allen Jackson
280	Landing Sequence#15	0.5d	Wed 6/24/98	Wed 6/24/98	279	Allen Jackson

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
281	Landing Sequence#16	0.5d	Wed 6/24/98	Wed 6/24/98	280	Allen Jackson
282	Landing Sequence#17	0.5d	Thu 6/25/98	Thu 6/25/98	281	Allen Jackson
283	Landing Sequence#18	0.5d	Thu 6/25/98	Thu 6/25/98	282	Allen Jackson
284	Landing Sequence#19	0.5d	Fri 6/26/98	Fri 6/26/98	283	Allen Jackson
285	Landing Sequence#20	0.5d	Fri 6/26/98	Fri 6/26/98	284	Allen Jackson
286	Landing Sequence#21	0.5d	Mon 6/29/98	Mon 6/29/98	285	Allen Jackson
287	Landing Sequence#22	0.5d	Mon 6/29/98	Mon 6/29/98	286	Allen Jackson
288	Landing Sequence#23	0.5d	Tue 6/30/98	Tue 6/30/98	287	Allen Jackson
289	Derelict #2 (Damaged Ship)	3d	Mon 6/29/98	Wed 7/1/98	259	Andrew Tjew
290	Derelict #3 (Damaged Ship)	3d	Thu 7/2/98	Mon 7/6/98	289	Andrew Tjew
291	Nebula	5d	Tue 7/7/98	Mon 7/13/98	290	Andrew Tjew
292	Black Holes	3d	Fri 7/3/98	Tue 7/7/98	266	Hugh David
293	Neutron Stars	3d	Wed 7/8/98	Fri 7/10/98	292	Hugh David
294	Power Ups #1	1d	Tue 6/30/98	Wed 7/1/98	288	Allen Jackson
295	Power Ups #2	1d	Wed 7/1/98	Thu 7/2/98	294	Allen Jackson
296	Dealing with cheating - random key	1d	Mon 7/20/98	Mon 7/20/98	236	Tony Ikeda
297	Dealing with cheating - in/out traffic	2d	Tue 7/21/98	Wed 7/22/98	296	Tony Ikeda
298	Dealing with cheating - encryption	3d	Thu 7/23/98	Mon 7/27/98	297	Tony Ikeda
299	Ship Debri #2	1d	Tue 7/14/98	Tue 7/14/98	291	Andrew Tjew
300	Ship Debri #3	1d	Wed 7/15/98	Wed 7/15/98	299	Andrew Tjew
301	Ship Debri #4	1d	Thu 7/16/98	Thu 7/16/98	300	Andrew Tjew
302	Ship Debri #5	1d	Fri 7/17/98	Fri 7/17/98	301	Andrew Tjew
303	Ship Debri #6	1d	Mon 7/20/98	Mon 7/20/98	302	Andrew Tjew
304	Ship Debri #7	1d	Tue 7/21/98	Tue 7/21/98	303	Andrew Tjew
305	Power Ups #3	1d	Mon 7/13/98	Mon 7/13/98	293	Hugh David
306	Power Ups #4	1d	Tue 7/14/98	Tue 7/14/98	305	Hugh David
307	Power Ups #5	1d	Wed 7/15/98	Wed 7/15/98	306	Hugh David
308	Power Ups #6	1d	Thu 7/16/98	Thu 7/16/98	307	Hugh David
309	Cargo #3	1d	Fri 7/17/98	Fri 7/17/98	308	Hugh David
310	Cargo #4	1d	Mon 7/20/98	Mon 7/20/98	309	Hugh David
311	dealing with cheating - Soft Ice	5d	Tue 7/28/98	Mon 8/3/98	298	Tony Ikeda
312	Derelict #4 (Damaged Ship)	3d	Tue 7/21/98	Thu 7/23/98	310	Hugh David
313	Derelict #5 (Damaged Ship)	3d	Fri 7/24/98	Tue 7/28/98	312	Hugh David
314	Power Up (Flag) #2	1d	Wed 7/29/98	Wed 7/29/98	313	Hugh David
315						

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
316						
317						
318	Main Information Exchange (Multi) - Sports Wa	1d	Thu 7/30/98	Thu 7/30/98	314	Hugh David
319	Main Information Exchange (Multi) - View Aren	5d	Fri 7/31/98	Thu 8/6/98	318	Hugh David
320	Arena (Multi)	2d	Thu 10/9/97	Fri 10/10/97		
321	Arena (Multi) - Join Game	1d	Thu 10/9/97	Thu 10/9/97		
322	Arena (Multi) - Setup Arena	2d	Thu 10/9/97	Fri 10/10/97		
323	Arena (Multi) - One on One	2d	Thu 10/9/97	Fri 10/10/97		
324	Arena (Multi) - Last Man Standing	2d	Thu 10/9/97	Fri 10/10/97		
325	Arena (Multi) - Team Battle	2d	Thu 10/9/97	Fri 10/10/97		
326	Arena (Multi) - Alliance Battle	4d	Thu 10/9/97	Tue 10/14/97		
327	Arena (Multi) - Tally Board	3d	Thu 10/9/97	Mon 10/13/97		
328	Initiate Menu - Bulletin Board	2d	Thu 10/9/97	Fri 10/10/97		
329	Initiate Menu - Disassociation	1d	Thu 10/9/97	Thu 10/9/97		
330	Member Menu - View Member Stats	3d	Thu 10/9/97	Mon 10/13/97		
331	Member Menu - Locate Member	2d	Thu 10/9/97	Fri 10/10/97		
332	Member Menu - Locate Alliance Station	1d	Thu 10/9/97	Thu 10/9/97		
333	Member Menu - Pay Dues to Alliance	2d	Thu 10/9/97	Fri 10/10/97		
334	Commander Menu - Warehouse Ledger	3d	Thu 10/9/97	Mon 10/13/97		
335	Commander Menu - Buy/Sell Station	3d	Thu 10/9/97	Mon 10/13/97		
336	Commander Menu - Add/Remove member	4d	Thu 10/9/97	Tue 10/14/97		
337	Commander Menu - Promote/Demote	3d	Thu 10/9/97	Mon 10/13/97		
338	Commander Menu - Initiate Player	3d	Thu 10/9/97	Mon 10/13/97		
339	CoPresident Menu - Change Info	3d	Thu 10/9/97	Mon 10/13/97		
340	CoPresident Menu - Retire Station	2d	Thu 10/9/97	Fri 10/10/97		
341	CoPresident Menu - Dissolve	2d	Thu 10/9/97	Fri 10/10/97		
342	President Menu - Promote/Demote Member	1d	Thu 10/9/97	Thu 10/9/97		
343	Multi-Player Arena - SysOp Option	5d	Fri 8/7/98	Thu 8/13/98	319	Hugh David
344	Multi-Player Arena - Power Ups	5d	Fri 8/14/98	Thu 8/20/98	343	Hugh David

Projections for Milestones:

Milestone #1 (Planetary Systems) *Due 11/30/97*

Basic 2d Menu System

- Planetary System (single and multi player)
- Port Screen
- Commodities Exchange
- Information Exchange
- Hangar Screen
- Basic System
- leave port to another port - for this, we immediately go to another port without spaceflight
- Screen to Screen movement
- Navigation Bar functional
- place holders for several menu screens
- Hot Recs, Postage Stamps, functional

Milestone #2 (Basic Spaceflight) *Due 1/31/98*

3d System

- Jump Points Enabled
- Multiple Action Sphere
- 2d HUD Overlay
- Navigation Points Enabled
- Joystick and Input controls
- Space Ships (1 Space Station, 1 Capital Ship, 1 Freighter, 1 AI Fighter, 1 Player Ship

Milestone #3 (Planetary Systems Completion) *Due 3/25/98*

- Completely Functional Load Out
- Commodities Exchange Fully Functional
- Random Mission Systems Functional
- Sub-Plot Missions Functional
- Information Kiosk Functional

Confidential

Milestone #4 (Alpha)

Due 5/11/98

Basic Gameplay, Servers - Single and Multi-Player

- Chat screen
- Collisions and Physics
- Missles & Weapons
- Space Battle
- AI (includes NPC)
- Missions
- PAD
- HUD (almost fully functional)

Milestone #5 (Beta)

Due 6/6/98

Repetitive importation of many items, Ships, Planet Screens, etc.

- Music
- Sound FX
- Turrets
- Shields
- Nebula, black hole, etc
- Thrusters, and other special effects to enhace the looks of ships
- Explosions
- Cinematics
- MASS importing of Space Stations, Capital Ships, Freighters, AI Fighters, Player Fighters

Milestone #6 (Final*)

- **August 31, 1998**

Confidential

Projections for Alpha:

We would like to define what would be required for *Alpha* so we can understand the priorities of scheduling. For *Alpha (E3)*, we should have one of each characteristic features implemented.

- Master server (match maker)
- One game server (up to 4-8 players)
- Chatting feature
- Single landing sequence
- One cap ship
    - defense mechanism
- Space station
    - docking
    - defense mechanism
- One freighter
- Three small ships
    - Three distinct AI for each small ship
- One or more player ships
    - HUDs for different ships completed
- One explosion
- Ship debris
- Missle & laser
- Thrusters, boosters, and afterburner
- Shield
- Damage effect
- PAD (including star map)
- Nav map
- One planet
    - Port screen (information kiosk - with place holders)
    - At least one NPC
    - Commodities (buy, sell)
    - Hangar (loadout, buy/sell weapons, repair)
    - Information exchange
    - Plot mission
    - Random mission
    - Arena
- A single mission
- Jump Points
- Front end and introductions
    - Splash
    - Logo
    - Menu
        - Options
- Cinematic
- Sound effect for AI, and crucial sounds

Confidential

- Gameflow (menu) music - the *Privateer 3* theme
- Gameplay music
- Basic gameplay speech (flight computer, port speech, some plot-specific)
- E3 Promotional Trailer - including shots from any available cinematic, current gameplay, gameflow screens, *P3* logo, original music, SFX, voice over, slick editing, and anything else that makes the product look great!

Confidential

Chapter 7

7

Origin, Systems Inc.

Technical Design Document

Appendix

Appendix A: Tools

Alias Model Converter

Description: The 3D models used in the engine are built using Alias. The Alias models must be converted first. Currently the process to convert the models requires of two programs. The first program runs on the SGI and merely extracts the model data and writes it in text format. The second program runs on a PC and
Implementation: Currently this tool consists of two programs. The first program runs on the SGI and merely extracts the model data and writes it out in text format. The second program runs on a PC and processes the Alias data and writes it out in binary format.

Alias Model Viewer

Description: This tool assists the artists with determining the final look and correctness of the models. It is basically a model viewer with camera movement capacity.

Texture Bitmap and Palette Viewer

Description: Bitmaps and palettes for the models are consolidated into a file of all bitmaps and a file of all palettes. This viewer allows one to look at these bitmaps and palettes.

Object Hierarchy Viewer

Description: This tool allows one to examine the DAG structure of objects.

Mission / Scenario Builder

Description: This is a tool to allow designers to create state driven lookup data that will be used to assign missions to the players. By having a state driven data, we can be much more flexible and less cause of code bugs caused by conditions involved when an incident happens, trigger another mission linked to the mission. This will make it much easier to upgrade and maintenance mission conditionals.

Confidential

XMIFF

One of the most essential tools used in both *Prophecy* and *Privateer 3* is XMIFF. Due to game being very data dependent, XMIFF is used to construct an extended IFF data that our game retrieves. See *Appendix E - XMIFF documentation* for more details on how to construct data.

Confidential

Appendix B: Alias Model conversion process

This section covers what we have learned so far from the Alias model conversion process. This section will be updated regularly and distributed to artists when new lessons are learned.

Confidential

Appendix C: Data handling – data driven game

Majority of the game is data driven and so the management of game upgrading will be easier to maintain and update. Since the data are the entities to be modified, once the code is written, it makes it easier to determine and maintain bugs after the core code is debugged.

This section is not about coding, but rather, how and what handles these external data that are dependent to the game.

Because of the data being open to the user, we need to consider the fact that the data can be hacked and modified to cheat as well.

Audio (Music and sound)

Music files will be MGI files and the sound and speech files will be in IFF file format. These data are created by the music department using the tools created by Chuck Karpiak.

Fonts

Fonts are IFF files that are generated via tool written by Hugh David which converts an TrueType fonts into bitmap format.

Mesh

Mesh data are created by Andrew Tjew's converter routine.

Space/Mission

Mission data are currently considered to be generated by MED.

Confidential

Appendix D: CD and Media storage

There are many things that has to be considered when streaming from a media other then the hard drive. Items that should be carefully observed are:

- *Media streaming speed* - Speed of the media that it can stream from may and can cause music and movies to stutter. Hardware such as IDE CDROM drives and old 1x CD drives should be tested to make sure that they don't cause stuttering effects.
- *Media size* - Size of the media is crucial to how the file tree structures will be set up so the music, sound effects, movies, and missions will be found appropriately.

Music and Sound effects

Our current engine can support close to 4:1 compression ratio. Going to 16bit 22KHz instead of 44.1 kHz can reduce the size of data. Going from stereo to mono can also reduce the size of the data. Carefully selecting sound effect to go from 16bit to 8bit sampling as well can be simple solutions to reduce the size of data.

Currently, Direct3Dsound are only in mono, yet the hit we take is about 10% loss in the frame rate, and thus we would have to also have to attempt to speed up the method so we can use these sound effects efficiently.

With the amount of music that needs to be streamed for different sectors of the universe, Jason Cobb has calculated about 260 minutes of music, which can fill up 3 red book audio CD's. Many methods should be considered to resolve storage spaces and streaming speed, such as considerations of *MPEG 1 Layer 3* which allows 12:1 compression ratio. According to Jason Cobb, the sound quality is far better then 16 bit 22.05 kHz sample rate, and almost sounds as good as the red book audio CD. This will allow us to consider storing approximately a 500 Megs audio data into 100 Megs storage space as well. Currently, the method had been tested and was found that it caused some pops and stutters which may need to be investigated further.

Movie

Although we are being careful and efficient about re-using the generic movies to save media spaces, there are movies that are unique to its specifics. Currently, the movie compression using the EAC's UV2 player written by Martin Griffiths takes approximately 3:1 compression ratio with audio and video weaved together. An average movie files may take about 15 Megs (some movies may go as high as 70+ Megs depending on its contents). Since this is an external project that we are adapting to our game due to our timeline, and at the same time, we'd like an high quality standard movie player, we may have to compensate our media space accordingly to this.

Confidential

DVD

If the possibilities exists that we may go to DVD, we need to make sure of the plans on how to go about streaming from it. The file tree structure would have to change as well.

Confidential

Appendix E - XMIFF documentation

XMIFF - Extended MIFF compiler

Author: Jason Yenawine

Command line switches: (using '/' or '-')

-d[symbol] define a symbol before compiling data
-i include path, e.g. -i..\dpmi;..\lib\inc
-big use Big Endian word arrangement
+,@ response file

The include path determines the directories that are searched for C/C++ include files using #include, and for including raw data files with #insert.

Big Endian word arrangement is useful for compiling an IFF file for the Mac or 3DO machines.

Use a response file to compile multiple IFF files. Example:

```
/i..\include;d:\my_prgram\include
world.mif
tummerik
xan  output.fil
```

This file will compile three IFF source files. If no file extension if supplied, the .XMF extension is assumed. If a second name is supplied, it is used as the output filename. Otherwise, the base name of the source is used, with .IFF added as the extension.

XMIFF contains much of the same functionality of the original MIFF program, but with a few extra features thrown in to add flexibility to the MIFF language. In general, syntax changes were made to make MIFF more like the C programming language. This document assumes that you are familiar with the old MIFF program and its syntax.

#include

syntax: #include <filename>, or #include "filename"
notes:

Confidential

Use #include to bring C/C++ defines and enumerations into your source file. Use the angle brackets to have XMIFF search for the include file in the include path only. Use the quotation marks to have XMIFF search in the directory of the source file first, then search the include path.

Conditional Compiles:

#if, #ifdef, #ifndef, #else, #endif, and #elif function just as they do in C.
note: #if defined() is not supported in XMIFF.

Labels:

A label is a variable name followed immediately by a colon. The value of a label is its position in the file. Labels can be used in expressions. The most common use of a label is to get the length of a data structure. (See Macros for an example of using labels.)

When a label is declared inside a CHUNK, its name is "scoped" within that CHUNK. It cannot be referenced from outside that CHUNK.

For example:

```
CHUNK dat1
{
my_label:
   long   my_label
}
CHUNK dat2
{
   long my_label
}
my_label:
```

In this example, the long value written in CHUNK dat1 uses the local label inside the chunk, while the value referenced in CHUNK dat2 uses the global label.

There are 3 special labels that are predefined by XMIFF.

@@: A nameless label that can be used again and again. Use @f in an expression to get the value of the next @@ label in the file. Use @b to get the previous @@ value.

$ Predefined to be the value of the current position in the file.

Confidential

Macros:

The macro offers a flexible way to define commonly used values and data structures. The simplest way to use a macro is to define a constant value. For example:

```
#define MAX_ROCKS   20
...
long   MAX_ROCKS                    //   long   20
```

Another way to use macros is for defining a data structure. Take the following data structure for example:

```
char  "Jason Yenawine"      // employee name
long   1968                 // birth year
short  180                  // height, in cm.
char    -1                  // sex, 0=NO, 1=YES, -1 =assexual
budding
```

This structure can be written more clearly with a macro.

```
#define    EMP_RECORD(name, birth_year, height, sex)        \
                        char[80]  name                      \
                         long   birth_year                  \
                        short    height                     \
                        char    sex

#define YES 1
#define NO  0
#define BUDDING -1
```

Invoke the macro like this:

```
EMP_RECORD("Jason Yenawine", 1968, 180, BUDDING)
```

In the example above, the definition of EMP_RECORD extends over more than one line. Use the '\' character to extend the macro. Macro definitions end on a comment (either // or /*) or a newline.

You can use macros to define a string type. Strings are not NULL terminated in XMIFF. To get a NULL terminated string, you must add a zero to the end of the string...or you can make a macro to do it for you.

Confidential

```
#define string(x)    char      x,0
```

Also, you could change the definition of string from the C/C++ convention to the Pascal convention for strings:

```
#define string(x)                       \
                char    @f - $ - 1      \
                char    x               \
                @@:
```

The first line of this macro calculates the length of the string by subtracting the current file position from the file position after the string.

You can use '##' to concatenate two symbols together in a macro. For example:

```
#define ROCKS_EVERYWHERE 200000
#define ROCKS_IN_THE_CITY   1250
#define ROCKS_IN_FRANCE     512
#define X(x)            ROCKS_##x

        long    X(EVERWHERE)            // 200000
        long    X(IN_THE_CITY)          // 1250
        long    X(IN_FRANCE)            // 512
```

NOTE: XMIFF does not detect circularly defined macros.
For example:

```
#define X X
long X
```

will cause XMIFF to loop until it runs out of memory.

Enumerations:

XMIFF supports C/C++ enum types.

FORM:

The keyword FORM must be followed by a FORM name. Up to 4 letters can be used. If less than 4 letters are used, 0's are used to pad the name to 4 characters. You can put the name in quotations if you want to pad the name using spaces.
For example:

```
FORM AI             // <- padded using 0
FORM "AI  "         // <- padded using ' '
```

Confidential

CHUNK:

The keyword CHUNK must be followed by a CHUNK name. Up to 4 letters can be used. If less than 4 letters are used, 0's are used to pad the name to 4 characters. You can put the name in quotations if you want to pad the name using spaces.
For example:

```
CHUNK AI                // <- padded using 0
CHUNK "AI  "            // <- padded using ' '
```

Data types:

XMIFF has 5 defined data types:

char - 8 bit signed data
short - 16 bit signed data
long - 32 bit signed data
float - 32 bit single precision
double - 64 bit double precision

Fixed point math:

type.bits where type is either char, short, or long. 'Bits' is the number of fraction bits. For example:

```
long.8          2.5          //      640 actually stored in file
```

Data length:

type[len] where type is either char, short, or long; len is the number of those types to store. You can specify all of the values to be stored, or enter only one.
For example:

```
long[4]         10,
                20,
                30
```

This would result in 4 longs being stored. The 4th long would be padding (0). Note also that a line can be continued using the comma. In the above example, if we had entered more than 4 values, the extra numbers would not have been used. Fixed point math and fixed length can be used at the same time. For example:

```
long.16[16]             2.5, -0.009, 0
```

Expressions:

Expressions in XMIFF are terminated by a newline or a comma.

Confidential

Unary Operators:

~ bitwise NOT.
! logical NOT
- negative

Binary Operators:

% remainder, integer divides
& bitwise AND
| bitwise OR
* multiply
- subtract
+ addition
/ division
|| logical OR
&& logical AND

Order of precedence:

1) Unary operators,
2) *, / (left to right),
3) all other operators, in left to right order

Numbers:

floating point:	[num].[num]	ex. 5.387
hexadecimal	0x[num]	ex. 0x5A92
octal	0[num]	ex. 052
decimal	[num]	ex. 105662

Confidential

Variables:

Variables in XMIFF operate in much the same way as in the BASIC programming language. They are global in scope and their type depends on what value they are assigned. Variables can be assigned to an integer type, a quoted string, or a floating point number. For example:

```
x = 0
y =  "Text string"
z = 2.5
CHUNK DATA
{
        long x
        x = x + 1
        char y
        long z
        z = z + x
}
CHUNK OBJ1
{
        long x
        x = x +1
        long z
}
```

Messages and Errors:

You can cause lines of text to be printed while your file is compiling. This is useful for determining how far along your compile is, and for determining the value of variables at various stages of compiling.

#print: print a message to stdout.
#fatal: print a message to stdout, then terminate compilation.

Confidential

examples:

```
x = 0
y =  "Text string"
z = 2.5

#print "Compiling data now..."
@@:
CHUNK DATA
{
        long x
        x = x + 1
        char y
        long z
        z = z + x
}

#print "Done. Data size = ", $ - @@, ".\n"
#if !x
   #fatal "Error: x is zero!\n"
#endif
```

Comments:

XMIFF allows C/C++ comments, including nested comments.

#insert:

syntax: #insert <filename>, or #insert "filename"

To include data directly into an IFF file, use #insert or include (without #).
For example:

```
CHUNK DATA
{
        #insert "rawdata.dat"
}
```

XMIFF uses the include path to find the data file.

Confidential

Sound document

Author: Jason Cobb

Overview

Sound in Privateer 3 should be an involving, immersive experience from the moment of auto play onward. Stereo spacialized samples should be used throughout to develop a rich dimensional soundfield. Even during load times, a small looping sound can be played to fill in the void. In order to assure complete control of the audio experience, all of the sound in Privateer 3 will be digital. All of the music will be streamed, no GM or FM will playback be used. Where possible, audio will be compressed with the XA audio codec (3.7:1). All frequently used sound effects will be loaded into RAM with options for varying quality and size of samples. Privateer 3 should take advantage of support for sound card RAM and 3D audio hardware acceleration through Direct Sound 3D and manufacturer drivers. Speech will be streamed off the CD or full install. Movie audio will be woven with video and stream from the CD or full install. All of these issues and more are detailed in the body of this document.

Music

Music is an artistic element that is crucial to creating a unique style and sound of a computer game. Privateer 3 is all about the gritty future in the fringes of the Confed controlled Wing Commander universe. It is intended to be a ground-breaking game that will be the measure of standard for the next few years. Therefore, it is very important to have a innovative, cutting edge musical soundtrack that will create a fresh, unique style for the Privateer property. For the purposes of this document, the encompassing genre of Privateer 3 music is defined as Techno. Techno is a broad term describing electronic music with even meter, heavy percussion, synths, and samples. From Techno, we can derive many sub-genres and styles that will lend a great variety and replayability to the game soundtrack.

Programming

- Music volume setting in game options screen.
- Music interactively streamed using Magi and XA (3.7:1) compression
- Possible MPEG layer II or III support.
- Game songs are linked to each of the 4 quads by default. They will be stored on the same CD as a given quad.
- Arena songs will be stored on every CD.
- Jukebox mode allows player to hear any song on the current CD they choose.

Confidential

- User may play red-book audio CD instead of game music with full install in multiplayer mode.
- Approximately 102 minutes of music not counting movie music. ~ 375 megabytes

Confidential

Songs

- **Menu** ~ 3 minutes, intro, loopable main section, outro. This is the Privateer 3 theme song. It is heard during all of the menus outside of game play.
- **Credits** ~ 3 minutes, loopable main section, outro. This is heard during the game credits.
- **Game** ~ 20 minutes, see branching diagram for details. There are 4 of these large songs, each comprised of may tracks that play depending on what is happening in the game. The 4 game songs relate stylistically to the 4 quadrants of space which are occupied by Confed, Kilrathi, Pirates, and Outcasts.
- **Arena** ~ 4 minutes, intro, loopable main section, outro. There are 4 arena tunes that are played exclusively in combat arenas.
- **Movies** - These songs will be composed based on the timing and emotion of a given animation. Movie music will be a crucial element in the soundtrack which also includes dialog, sound effects, ambience, and foley.
- **Trailers** ~ 45 seconds to 1.5 minutes. These songs will be intended for use with promotional trailers. They could be originate from selected tracks of game songs, the menu tune, or selected movie music.

The 4 game songs are divided into 14 tracks each that are interactively played according to game events. Switching between tracks is assisted by 54 short segues and follows the interactive music matrix diagram.

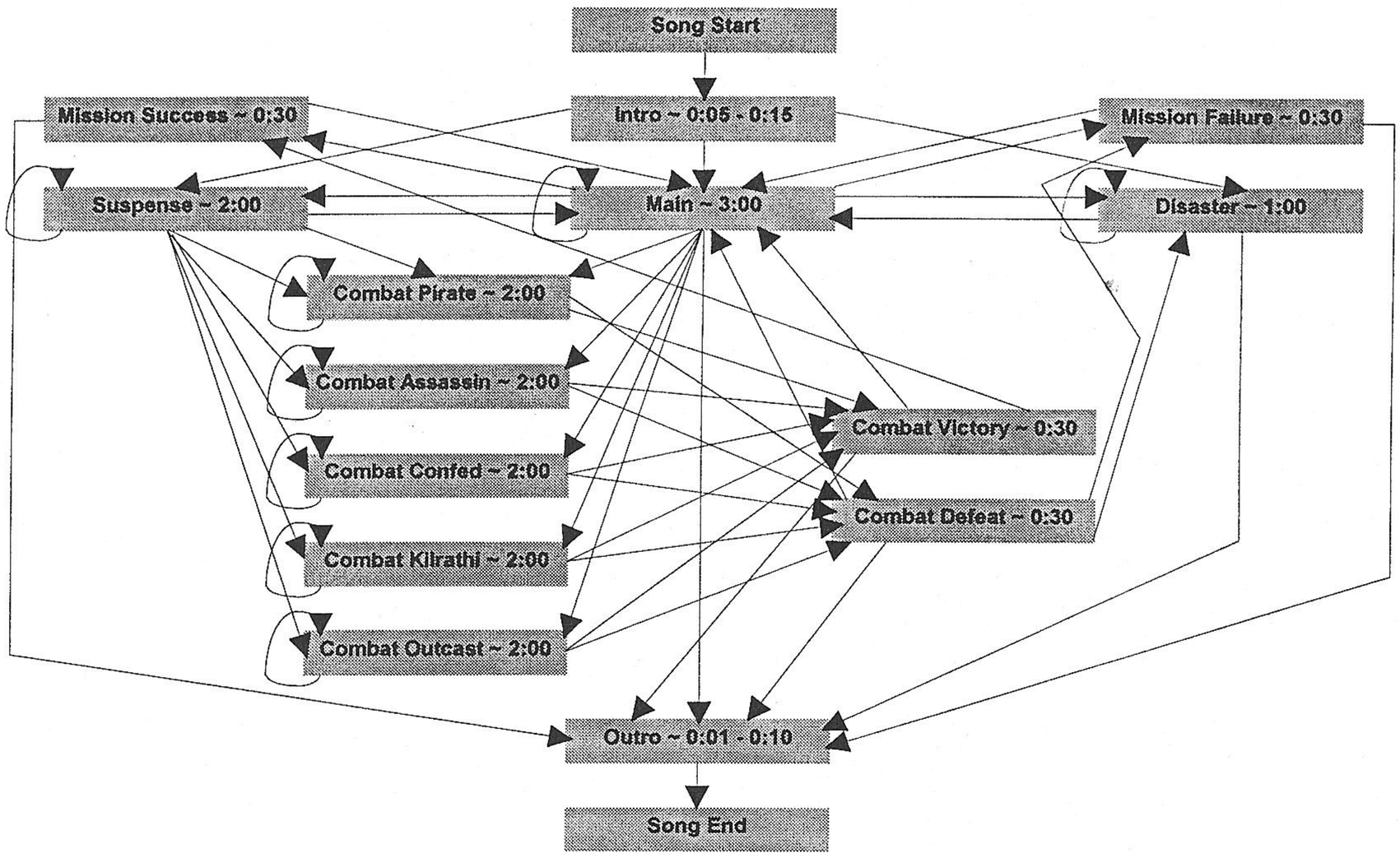
- **Intro** - ~ 5 to 15 seconds - played only once as the song is started for the first time
- **Main** - ~ 3 minutes, loopable - the main part of the song that plays while no major game events are occurring. Also heard when docked at a port.
- **Mission Success** - ~ 30 seconds - Plays when the player accomplishes a mission
- **Mission Failure** - ~ 30 seconds - Plays when a player fails to accomplish a mission
- **Disaster** - ~ 1 minute, loopable - Plays when a cataclysmic event occurs outside the player's action sphere. (another ship self destructs, part of the universe collapses, player's alliance star port is destroyed)
- **Suspense** - ~ 2 minutes, loopable - Plays when the player is about to be ambushed or attacked by unseen enemies
- **Combat Pirate** - ~ 2 minutes, loopable - Plays during combat with pirates
- **Combat Assassin** - ~ 2 minutes, loopable - Plays during combat with assassins or bounty hunters

Confidential

- **Combat Confed -** ~ 2 minutes, loopable - Plays during combat with Confed
- **Combat Kilrathi -** ~ 2 minutes, loopable - Plays during combat with Kilrathi
- **Combat Outcast -** ~ 2 minutes, loopable - Plays during combat with outcasts
- **Combat Victory** - ~ 30 seconds - Plays when the player defeats an enemy in combat
- **Combat Defeat** - ~ 30 seconds - Plays when the player loses in combat
- **Outro** - ~ 1 to 10 seconds - Played only once as the song is ending and before the next song is played. Basically, a way to end a song short of just cutting it off. Songs will be ended whenever the player docks, dies, or leaves the quad.
- **Segues** - ~ 1 - 5 seconds each - The 54 possible segues are used for looping an individual track on itself or to get from one track to another within a song. Depending on the tracks, segues can be anything from a musical riff, a drum fill, a sample, or the decay of a cool multi-effect. The exact nature of segues will vary depending on the circumstances. In the case of two tracks being able to be cut together smoothly, a segue may not be needed. In the case of segues to the outro, if preferred, the segue could just end the song itself, rather than branching to the outro to end. Segues to allow track looping may just be small bits to smooth over the loop area, or they may not even be needed.

Confidential

Interactive music matrix diagram
(Segues are represented as arrows →)



Confidential

Sound Effects

Programming

- Sound effects volume setting in game options screen.
- Direct Sound 3D for real-time panning of crucial mono sound effects for objects outside the player's ship (ships, explosions, weapons).
- Support for sound card hardware audio effects acceleration and on-card RAM via Direct Sound 3D / third party drivers.
- Stereo spacialized samples for many sound effects (interior computer noises, mechanical noises, menu / interface sounds).
- 3 meg RAM minimum for frequently used sound effects, 4 or 6 meg RAM option for higher quality samples.
- Some sound effects loaded only as needed.

Menus / Interface

- Autoload sounds to kick-off the immersive sonic experience from the insertion of the CD-ROM onward.
- Item selections.
- Pop up menus.
- Screen transitions.
- Small looping sounds to play while loading data.
- Specific event cues.
- Command responses.

Gameplay

- Flight, targeting, navigation computers / interface (switches, buttons, alert beeps, warning beeps, status cues, interface selections).
- Communications sweeteners (static, alert beeps, sos beacon, message / chat waiting alerts, distinct enemy / friendly / neutral communication and message waiting beeps).
- Power plants (pitched running loop, changes pitch upward when overtaxed or before self-destruct, changes pitch downward when low on fuel).
- Ship engines (power up / down, pitched running loop, fixed low end rumble, damaged loop, internal and external).
- Fly-by sweeteners (whoosh of passing a large asteroid, etc.…).
- Weapons (internal and external versions with mechanical activation, operational and damaged versions).

Confidential

- Equipment (drones, electronic counter measures, tractor beams, cargo holds, operational and damaged versions).
- Shields (impacts, weak or overtaxed, power up and down).
- Explosions (chemical, nuclear, anti-mater, large, medium, small, near, distant).
- Physical impacts (three distinct ship alloys, ship landings, rock, large, medium, small, internal and external).
- Space port ambience and machinery (doors, creatures, etc.…).
- Ship maintenance and customization (add weapons, etc.…).

Movies

- Plot specific and generic cut-scenes.
- Sounds must match those used in gameplay.
- Foley (clothing rustle, body movement, prop noises).
- Some sound effects could be interpreted as stylistic musical cues.

Confidential

Speech

Programming

- Speech volume setting in game options screen.
- Subtitles for all speech.

In-flight computer (male and female versions)

- Simple two word phrases parsed on the fly from RAM (not streamed)
- Targeting feedback
- Ship, equipment status / damage information
- Navigation response
- Reconnaissance information
- Drone status information
- Creature comforts (please fasten your seatbelt!)
- Augment vocabulary incrementally as player upgrades equipment
- Speech becomes slow (pitch shift) and stutters (random re-trigger) when computer is damaged
- May be turned off in audio options menu

Mission Specific

- Streamed audio with XA (3.7:1) compression
- Game play plot line - radio communications
- NPC attack phrases
- Movie script

Space Port

- Streamed audio with XA (3.7:1) compression
- Ambient crowd walla
- Ship bay salesman
- Information kiosk
- Commerce computer
- Landing clearance

Movies

- Principal scripted dialog recorded and edited prior to animation to provide sync
- Miscellaneous extra speech for world enrichment

Confidential

- Various ambient crowd walla based on quadrant specific populations

Translations

- All speech translations by Origin
- All speech recorded in out-of-house studios under Origin supervision / direction

Confidential

Movies

Scenes

- Game intro
- Plot-specific movies (including endgame)
- Generic cut scenes (planet landings, dockings)

Production formats

- Mixes: stereo / 4-2-4 surround sound, AC3
- 16 bit, 48kHz master resolution, DVD resolution
- 16 bit 22.050 stereo CD game resolution using XA compression (3.70:1)
- Stems: 6 dialog, 6 ambience, 6 foley, 6 hard effects, 4 pan effects, 4 music
- 8 track master DA-88 reels (Lt, Rt, LF, LS, RF, RS, C, S)
- Ambience, hard effects, pan effects, dialog, and music stems produced at Origin
- Foley produced out-of house under Origin supervision / direction

Production flow

- Storyboards created.
- Movie dialog recorded and edited to storyboards.
- Animations done to timing of dialog edit.
- Rough cut of music and composed to composite storyboard / animations.
- Rough cut of sound effects edited to composite storyboard / animations.
- Final cut of animations.
- Sound effects, ambience and foley edited to finished animations.
- Final music cut to finished animations.
- All stems mixed.

Confidential

Appendix G: 2D and 3D Interface

Using Objects

Most object must be created first before it can be used. As such every object has a creation and destruction method. The creation method Create() allocates all resources necessary for the object to exist. The destruction method Release() frees all allocated resources. An object becomes invalid after calling Release(). Note that object creation and destruction are not implemented using the C++ constructors and destructors.

Creating an Object

There are two ways to create an object:

- The object can create itself.
- The object is created by a class factory or a class manager.

Example of an object that create itself:

```
OBJECT Object;
Object.Create();
```

Example of an object that is created by a class factory:

```
FACTORY Factory;
OBJECT* Object;
Object=Factory.CreateObject();
```

Destroying an Object

Analogously, there are two ways to destroy an object:
The object has it own destruction method.
The object must be destroyed by a class factory or manager.

Example of an object that can destroy itself:

```
Object.Release();
```

Example of an object that must be destroyed by a class factory:

```
Factory.ReleaseObject(Object);
```

The Release() method for objects that can destroy itself is automatically called by the objects go out of scope. Objects created by class factories or managers and not explicitly destroyed are destroyed when the class factories or managers go out of scope.

Concrete Objects

These are objects without Create() and Release() methods. They never demands any resourses other than physical memory for instantiation. They should be considered more as types than objects. They are almost never used as a base class. Example is a

Confidential

vector class with only components as member data. [Stroustrup, The C++ Programming Language]

Confidential

CS_DATA.HPP

Template CS_ARRAY<DATATYPE>

Status: Final
Description: This template can be used to define arrays. It is optimized for speed

Example:

```
struct POINT
{
  float X;
  float Y;
  float Z;
};
CS_ARRAY<POINT> Array;
DWORD i;
DWORD Size;
POINT* Point;
//--- Slow method to set points to (0, 0, 0) ---
for (i=0;i<Array.GetSize();i++)
{
  Array[i].X=0.0f;
  Array[i].Y=0.0f;
  Array[i].Z=0.0f;
}

//--- A faster but less elegant way to clear elements to zero ---
Size=Array.GetSize();
for (i=0;i<Size;i++)
{
  Point=Array.GetElement(i);
  Point->X=0.0f;
  Point->Y=0.0f;
  Point->Z=0.0f;
}
```

Arguments: <DATATYPE>: the element type of the array.

EC Create(DWORD Size);

Description: Creates array of specified size (number of total elements).
Argument: DWORD Size: Size of array.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

EC Release();

Description: The standard Release() method.
Argument: DWORD GetSize()const;
Description: Get size of array.
Return Value: DWORD: Size of array.

Confidential

DATATYPE* GetArray()const;

Description: Returns block of memory where the elements are stored.
Return Value: DATATYPE*: Pointer to memory.

DATATYPE& GetElement(DWORD Index);

Description: Get specific element. No specific out of range notifications. Out of range is trapped in debug mode.
See Also: DATATYPE& operator[](DWORD Index);

EC GetElement(DWORD index, DATATYPE*& Element);

Argument: DWORD Index: Element index into the array.
Return Value: DATATYPE&: Element requested.

DATATYPE& operator[](DWORD Index);

Description: Get specific element. No specific out of range notifications. Out of range is trapped in debug mode.
See Also: DATATYPE* GetElement(DWORD Index);

EC GetElement(DWORD index, DATATYPE*& Element);

Argument: DWORD Index: Element index into the array.
Return Value: DATATYPE&: Element

EC GetElement(DWORD index, DATATYPE*& Element);

Description: Get specific element with out of range notification.
See Also: DATATYPE* GetElement(DWORD Index);

DATATYPE& operator[](DWORD Index);

Argument: DWORD Index: Element index into the array.
Return Value: EC: EC_OK on success, EC_FAIL on failure.
DATATYPE*& Element: Element

Confidential

CG_DISPLAY_HPP

class CG_DISPLAY

Status: Alpha
Description: This object is a class factory. Specifically, it makes those classes that depends on the graphics API (Glide, DirectX, etc.). This object is to shield the underlying graphics API and provides device independence at the source level.

CG_ARRAY_DISPLAYDRIVER* Create(HWND Window);

Description: The method creates CG_SCREEN and returns an array of valid display drivers.
Arguments: HWND Window: Window handle.
Return Value: CG_ARRAY_DISPLAYDRIVER*: Display driver.

EC Release();

Description: The standard Release() method.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

CG_SCREEN* CreateScreen(DWORD MaxObjects, DWORD MaxTriangles, DWORD MaxVertices, DWORD MaxPositions, DWORD MaxBitmaps, DWORD MaxPalettes, CG_DISPLAYDRIVER& DisplayDriver, BOOL WindowMode=FALSE);

Description: Creates a screen.
Arguments: DWORD MaxObjects: Maximum objects on screen.
DWORD MaxTriangles: Maximum number of triangles that an object can have.
DWORD MaxVertices: Maximum number of vertices that an object can have.
DWORD MaxPosition: Maximum number of positions that an object can have.
DWORD MaxBitmaps: Maximum number of texture bitmap pages.
DWORD MaxPalettes: Maximum number of texture palette pages.
CG_DISPLAYDRIVER& DisplayDriver: Display driver.
BOOL WindowMode: Default is full screen mode.
Return Value: CG_SCREEN*: Created screen. NULL on failure.

Confidential

Remarks: Currently only one CG_SCREEN can be created.

EC ReleaseScreen(CG_SCREEN* Screen);

Description: Releases a CG_SCREEN created with CreateScreen();
Arguments: CG_SCREEN* Screen: CG_SCREEN to be released.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

Confidential

CG_SCREEN

class CG_SCREEN

Description: This class abstracts the physical screen and provides methods to draw to the screen.

Remarks: For device independence, this object should be created by the CG_DISPLAY class factory.

EC Create(DWORD MaxObjects, DWORD MaxTriangles, DWORD MaxVertices, DWORD MaxPositions, CG_CACHE_TEXTUREBITMAP& TextureBitmapCache, CG_CACHE_TEXTUREPALETTE& TexturePaletteCache, LPDIRECTDRAWSURFACE DirectDrawFrontBuffer, LPDIRECTDRAWSURFACE DirectDrawBackBuffer, LPDIRECTDRAWSURFACE DirectDrawDepthBuffer, LPDIRECT3DDEVICE2 Direct3DDevice2, LPDIRECT3DVIEWPORT2 Direct3DViewport2);

Description: The standard Create() method().

Remarks: For device independence, this object should be created by the CG_DISPLAY class factory.

EC Release();

Description: The standard Release() method.

Return Value: EC: EC_OK on success, EC_FAIL on failure.

Remarks: Do not call this method when CG_SCREEN is created by a class factory.

EC LockTextureBitmap(DWORD TextureBitmapIndex, VOID*& Surface, DWORD& Pitch);

Description: Locks a texture bitmap to access the bitmap surface.

Remarks: Do not assume the bitmap surface and pitch to be the same between calls. For every call to LockTextureBitmap(), there must be a call to a corresponding UnlockTextureBitmap()

Arguments: EC: EC_OK on success, EC_FAIL on failure.
DWORD TextureBitmapIndex: Index of bitmap.

Return Values: VOID*& Surface: Bitmap surface.
DWORD& Pitch: Pitch of bitmap. This value is not necessary the same as the width of the bitmap.

Confidential

VOID UnlockTextureBitmap(DWORD TextureBitmapIndex);

Description: Unlocks a texture bitmap locked previously with LockTextureBitmap();
Arguments: DWORD TextureBitmapIndex: Must be the same index used to lock texture bitmap.

EC LockTexturePalette(DWORD TexturePaletteIndex, CG_TEXTUREPALETTE*& Palette);

Description: Locks a texture palette to access the palette entries.
Remarks: Do not assume the CG_TEXTUREPALETTE to be the same between calls. For every call to LockTexturePalette(), there must be a call to a corresponding UnlockTexturePalette()
Arguments: EC: EC_OK on success, EC_FAIL on failure.
DWORD TexturePaletteIndex: Index of palette.
Return Values: CG_TEXTUREPALETTE*& Palette: Texture palette.

VOID UnlockTexturePalette(DWORD TexturePaletteIndex);

Description: Unlocks a texture palette locked previously with LockTexturePalette();
Arguments: DWORD TexturePaletteIndex: Must be the same index used to lock texture palette.

EC LoadTextureBitmapsFromFile(CHAR* Filename, DWORD BitmapIndexOffset, DWORD& BitmapsLoaded);

Description: Loads texture bitmaps from a file.
Arguments: CHAR* Filename: Filename.
DWORD BitmapIndexOffset: Internally, all bitmaps are stored in an array. This parameter specifies where the bitmaps from the file should be stored in the array.
Return Values: EC: EC_OK on success, EC_FAIL on failure.
DWORD& BitmapsLoaded: Number of bitmaps loaded.

EC LoadTexturePalettesFromFile(CHAR* Filename, DWORD PaletteIndexOffset, DWORD& PalettesLoaded);

Description: Loads texture palettes from a file.
Arguments: CHAR* Filename: Filename.
DWORD PaletteIndexOffset: Internally, all palettes are stored in an array. This parameter specifies where the palettes from the file should be stored in the array.

Confidential

Return Values: EC: EC_OK on success, EC_FAIL on failure.
DWORD& PalettesLoaded: Number of palettes loaded.

EC SetViewport(DWORD Left, DWORD Top, DWORD Width, DWORD Height);

Description: Sets the size of the viewport.
Arguments: DWORD Left, DWORD Top, DWORD Width, DWORD Height:
Position and size of viewport.
Return Values: EC: EC_OK on success, EC_FAIL on failure.

VOID Clear(DWORD Color=0);

Description: Clears the screen.
Argument: DWORD Color: Currently in B8G8R8 format. Subject to change.

VOID Flip();

Description: Flip between back and front buffers.

VOID SetUpFor2DScene();

Description: Sets up the internal state for 2D drawing (Draw2DSprite()).

VOID SetUpFor3DScene();

Description: Sets up the internal state for 3D drawing (RenderObject(), RenderEmisiveObject()).

VOID Draw2DSprite(CG_2DSPRITE& Sprite);

Description: Draws a CG_2DSPRITE.
Argument: CG_2DSPRITE& Sprite: Sprite to be drawn.

VOID RenderObject(CG_OBJECT& Object, CG_4X4MATRIX& ObjectToWorldMatrix, DWORD TextureBitmapIndex, DWORD TexturePaletteIndex);

Description: See SetLightManager()

VOID RenderEmissiveObject(CG_OBJECT& Object, CG_4X4MATRIX& ObjectToWorldMatrix, DWORD TextureBitmapIndex, DWORD TexturePaletteIndex, CG_COLOR& Color);

Description: See SetLightManager()

Confidential

VOID SetLightManager(CG_LIGHTMANAGER& LightManager);

Description: Sets the CG_LIGHTMANAGER to be used for light calculations in 3D rendering.

Remarks: CG_SCREEN only takes the pointer to CG_LIGHTMANAGER, so the CG_LIGHTMANAGER passed to CG_SCREEN must exist at all times CG_SCREEN is in scope.

Parameters: CG_LIGHTMANAGER& LightManager: Light manager.

VOID SetCamera(CG_CAMERA& Camera);

Description: Sets the CG_CAMERA to be used for 3D rendering.

Remarks: CG_SCREEN only takes the pointer to CG_CAMERA, so the CG_CAMERA passed to CG_SCREEN must exist at all times CG_SCREEN is in scope.

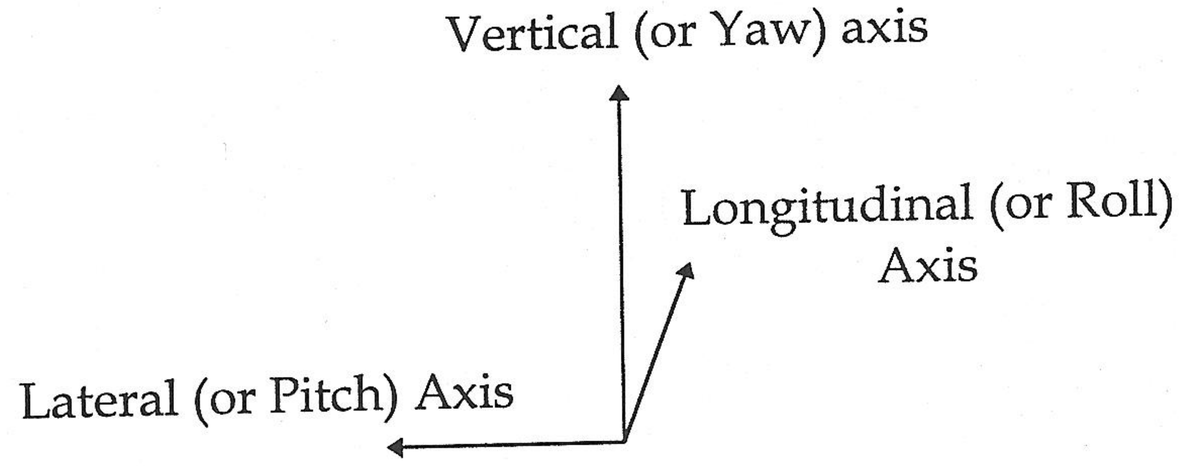
Parameters: CG_CAMERA& Camera: Camera

Confidential

CG_CONTROL

class CG_CONTROL

Status: Alpha
Description: This class provides the functionality of a (control) point in space with local orientation axis (local coordinate reference system). The forward direction is the longitudinal direction.



EC Create(DWORD NumberOfUpdatesBeforeNormalization=4, DWORD InitialNormalizationUpdateCount=0);

Description: The standard Create() method.
Arguments: DWORD NumberOfUpdatesBeforeNormalization: Due to accumulation of floating point round off errors, the axis looses its orthogonality after each Update() call. However renormalization is a costly operation. This parameter specifies how many times Update() should skip normalization. A good number is 1000.
DWORD InitialNormalizationUpdateCount: This parameter allows the normalization process to be staggered for multiple CG_CONTROLs with synchronized Update() calls.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

EC Release();

Description: The standard Release() method.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

Confidential

VOID Update();

Description: Updates the internal values. Getters are only valid after this method is called.

VOID Normalize();

Description: Explicitly normalize the axis. Normally Update() will automatically call this method after it has been called a specific number of times.

VOID Reset();

Description: Resets CG_CONTROL to world coordinate (0, 0, 0) facing upright in the x direction.

CG_4X4MATRIX& GetMatrix();

Description: Get the internal CG_4X4MATRIX that is used to represent CG_CONTROL's position and axis.
Return Value: CG_4X4MATRIX&: Internal matrix.

VOID SetPosition(FLOAT X, FLOAT Y, FLOAT Z);

Description: Setters for the position of CG_CONTROL.

VOID GetPosition(FLOAT& X, FLOAT& Y, FLOAT& Z)const;

Description: Getters for the position of CG_CONTROL.

FLOAT GetX()const;

Description: Getters for the position of CG_CONTROL.

FLOAT GetY()const;

Description: Getters for the position of CG_CONTROL.

FLOAT GetZ()const;

Description: Getters for the position of CG_CONTROL.

VOID SetVelocity(FLOAT LongitudinalVelocity, FLOAT LateralVelocity, FLOAT VerticalVelocity);

Description: see SetVerticalVelocity()

Confidential

VOID SetLongitudinalVelocity(FLOAT LongitudinalVelocity);

Description: see SetVerticalVelocity()

VOID SetLateralVelocity(FLOAT LateralVelocity);

Description: see SetVerticalVelocity()

VOID SetVerticalVelocity(FLOAT VerticalVelocity);

Description: Setters for the velocity in the longitudinal, lateral and vertical direction.

VOID SetAngularVelocity(FLOAT RollVelocity, FLOAT PitchVelocity, FLOAT YawVelocity);

Description: see SetAngularYawVelocity()

VOID SetAngularRollVelocity(FLOAT RollVelocity);

Description: see SetAngularYawVelocity()

VOID SetAngularPitchVelocity(FLOAT PitchVelocity);

Description: see SetAngularYawVelocity()

VOID SetAngularYawVelocity(FLOAT YawVelocity);

Description: Setters for the angular roll, pitch and yaw velocity.

Confidential

CG_CAMERA

class CG_CAMERA:public CG_CONTROL

Status: Alpha
Description: This class provides the functionality of a camera.
See Also: CG_CONTROL

EC Create(FLOAT NearDistance, FLOAT FarDistance, FLOAT FieldOfView, DWORD ViewportWidth=640, DWORD ViewportHeight=480);

Description: The standard Create() method.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

EC Release();

Description: The standard Release() method.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

CG_4X4MATRIX& GetWorldToScreenMatrix();

Description: This matrix converts from world positions to screen coordinates.
Return Value: CG_4X4MATRIX&: World to screen coordinate matrix.

CG_4X4MATRIX& GetBackgroundToScreenMatrix();

Description: This matrix converts the world background positions (those that are 'infinitely' far way) to screen coordinates.
Return Value: CG_4X4MATRIX&: Background to screen coordinate matrix.

VOID Update();

Description: Updates the internal values. Some Setters defers their operation to this method. Getters are only valid after this method is called.

VOID SetPivot(FLOAT X, FLOAT Y, FLOAT Z);

Description: Sets the pivot point for the PointToPivot(), RotateHorizontallyAroundPivot(), RotateVerticallyAroundPivot(FLOAT Angle) methods.
Arguments: FLOAT X, FLOAT Y, FLOAT Z: Pivot point.

Confidential

VOID PointToPivot();

Description: Rotates camera such that it points to the pivot point.

VOID RotateHorizontallyAboutPivot(FLOAT Angle);

Description: Rotates the camera in the local horizontal plane around the pivot point.
Arguments: FLOAT Angle: Angle in radians to rotate.

VOID RotateVerticallyAboutPivot(FLOAT Angle);

Description: Rotates the camera in the local vertical plane around the pivot point.
Arguments: FLOAT Angle: Angle in radians to rotate.

VOID MoveLongitudinallyWidthPivot(FLOAT Distance);

Description: see MoveVerticallyWidthPivot()

VOID MoveLaterallyWidthPivot(FLOAT Distance);

Description: see MoveVerticallyWidthPivot()

VOID MoveVerticallyWidthPivot(FLOAT Distance);

Description: Moves the camera in the longitudinal, lateral or vertical direction. Moves the pivot also.
Arguments: FLOAT Distance: Distance to move.

Confidential

CG_2DSPRITE

class CG_2DSPRITE

Description: This class provides the functionality of a 2D sprite. Each sprite has four vertices. Each vertex has blending values for red, green, blue and alpha channels.

EC Create();

Description: The standard Create() method.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

EC Release();

Description: The standard Release() method.
Return Value: EC: EC_OK on success, EC_FAIL on failure.

VOID SetTextureBitmapIndex(DWORD TextureBitmapIndex);

Description: Sets the texture bitmap index to be associated with current sprite.
Arguments: DWORD TextureBitmapIndex: Texture bitmap index.

VOID SetTexturePaletteIndex(DWORD TextureBitmapIndex);

Description: Sets the texture palette index to be associated with current sprite.
Arguments: DWORD TexturePaletteIndex: Texture palette index.

VOID SetUnscaledPosition(FLOAT X, FLOAT Y, FLOAT Width, FLOAT Height, FLOAT U, FLOAT V,);

Description: Sets the positions sprite to be rendered without any scaling.
Arguments: FLOAT X, FLOAT Y: X, Y coordinates of sprite.
FLOAT Width, FLOAT Height: Width and height of sprite.
FLOAT U, FLOAT V: The U, V coordinates of the bitmap. U, V values are between 0 and 1.

CG_2DSPRITEVERTEX& GetVertex(CG_2DSPRITE::VERTEX Vertex);

Description: Gets a specific CG_2DSPRITEVERTEX.

Confidential

Arguments: CG_2DSPRITE::VERTEX: Specifies the index. Choose one of VERTEX_TOPLEFT, VERTEX_TOPRIGHT, VERTEX_BOTTOMLEFT, VERTEX_BOTTOMRIGHT.

Confidential

CG_2DSPRITEVERTEX

class CG_2DSPRITEVERTEX

Status: Pre pre-alpha.
Description: Defines a 2D sprite vertex. This is a concrete class.
Methods: VOID SetPosition(FLOAT X, FLOAT Y);
Description: Sets the vertex position.
Arguments: FLOAT X, FLOAT Y: Position of vertex. Unit is screen pixels.

VOID SetVertex(FLOAT X, FLOAT Y, FLOAT U, FLOAT V, FLOAT RedValue, FLOAT GreenValue, FLOAT BlueValue, FLOAT AlphaValue=0);

Description: see SetValues()

VOID SetColorValues(FLOAT RedValue, FLOAT GreenValue, FLOAT BlueValue);

Description: see SetValues()

VOID SetValues(FLOAT RedValue, FLOAT GreenValue, FLOAT BlueValue, FLOAT AlphaValue);

Description: Setters of vertex values.
Arguments: FLOAT X, FLOAT Y: Position of vertex. Unit is screen pixels.
FLOAT U, FLOAT V: UV values. Texel coordinates. Must be between 0 and 1.
FLOAT RedValue, FLOAT GreenValue, FLOAT BlueValue: Vertex RGB values. Effect depends on what the renderer mode. A common usage for these values is for color modulation.
FLOAT AlphaValue: Effect depends on renderer mode. A common usage is for blending in which case this value controls the opacity of the sprite.

VOID SetTexelPosition(FLOAT U, FLOAT V);

Description: FLOAT U, FLOAT V: UV values. Texel coordinates. Must be between 0 and 1.

Confidential

CG_TEXTURE

class CG_TEXTUREPALETTE:public
CS_ARRAY<CG_TEXTUREPALETTECOLOR>

Description: This class managers the texture palette. There are 256 entries in the palette.
CS_ARRAY<CG_TEXTUREPALETTECOLOR> Methods:
All

VOID SetColor(DWORD Index, BYTE Red, BYTE Green, BYTE Blue);

VOID SetColor(DWORD Index, BYTE Red, BYTE Green, BYTE Blue, BYTE Alpha);

Description: Sets the color and alpha channel values.
Arguments: DWORD Index: Index of palette entry to be set.
BYTE Red, BYTE Green, BYTE Blue, BYTE Alpha: Color and alpha channel values. Range is 0 to 255.

VOID SetColor(DWORD Index, FLOAT Red, FLOAT Green, FLOAT Blue);

VOID SetColor(DWORD Index, FLOAT Red, FLOAT Green, FLOAT Blue, FLOAT Alpha);

Description: Sets the color and alpha channel values.
Arguments: DWORD Index: Index of palette entry to be set.
BYTE Red, BYTE Green, BYTE Blue, BYTE Alpha: Color and alpha channel values. Range is 0 to 1.

Confidential

Index

3
3D 13
3Dfx 8, 9, 12, 14

A
AI 49, 51

D
DAG 13, 14, 59

G
Game server 16, 49, 51

M
Master Server 16, 49, 51
MPath 16

S
Server 16, 49, 51

T
TCP 8, 50

V
VISION game engine 49, 51

W
Winsock 9, 18, 19

Confidential