Bright-Crayon, LLC
3126 West Cary Street #407 Richmond, Virginia 23221 U.S.A.

Partners:	Laura Carter	Dr. Pamela Kiecker
+1 804 307 2729	George Kelly Flanagin	John N Pastore, Jr
www.bright-crayon.com	Susan R. Jacobs	Greg Provo

Empire Technical Spec

Compiled by George Kelly Flanagin

Abs Revision	Date of last Revision	Number of Pages
33	2004/02/12 20:14	52

1 Overview

1.1 Note to the reader

Because this document contains something of interest for many different readers, we have supplied explanations of many terms that might be confusing. For example: readers looking at the business model may already be familiar with the terms of marketing, but technically inclined readers are less so. Because we want to reduce document clutter and page flipping, the explanations of specific terms in specific disciplines have been confined to footnotes.

1.2 What is Empire?

Empire is a campaign scale, multi-player game that is modelled on the weaponry available before nuclear weapons. All actions are initiated by the players rather than by the game. It is played in an archipelago, and each world of islands is generated uniquely for each game. The game and its world are scalable for varying numbers of players, durations, speeds of play, and experience levels of the players involved.

To be brief, players enter the game on an island by themselves with little materiel with no view of the world beyond the edge of their "home island." As time passes in the game, the ability to create resources suitable for combat emerges. The player develops an economy of factories on the island. The players eventually discover each other, and compete for superiority through combat. The player with the most "stuff" at the end of the game is the winner.

Empire has been constructed so that there is no algorithmic winning strategy: one must use air, sea, and land power (and even some diplomacy) to subdue and overcome the other players in the game. Each player's role is one of commander-in-chief, rather than that of a soldier. There are no "predictable responses" from the server, and there is no way to exploit backdoors that circumvent the unpredictability of human players.

Empire is designed for network play, but it is not a game in which a player must be actively connected to the server at all times to play. The game engine controls certain automatic defences, it updates factory production, and it moves ships along their courses even while a player is not connected.

Thus, Empire is neither peer-to-peer (i.e., players' machines do not interface directly with each other), nor is it turn based (i.e., a player may make "moves" in the game whenever his accrued movement time allows him to do so, and one may use these movement units all at once, or spaced out across time). It is not strictly speaking a client-server game because [1] the server continues to maintain the world of the game whether any clients are connected or not, and [2] the players may explore some scenarios without being connected to the server at the time.

- *Empire is a unique type of military simulation.*

1.3 History of the Game

Empire is an evolution of several games of this type that evolved in university systems in the late 1970's and early 1980's. It has also been influenced by the thinking and play in other multi-player war games. It is *not* similar to multiplayer DOOM, or any other games in which a player must continually make tactical decisions. Instead, it is a game of strategy, and sometimes, patience.

In its overall plan, this particular version is most closely derived from a game by Ben Norton of Seattle, Washington. Ben's game, developed in part while he was a student at Evergreen College, was played on HP3000 minicomputers with dumb terminals and no graphics, connected with 300/1200 baud dialup modems. Clearly, much has changed.

The present version of Empire is consists of two parts: a server part that controls the players' interactions with each other, and a client part that presents the view of Empire's world to each player, and allows the player to take actions. The server's design is oriented around off-the-shelf components for Linux 7 and later. The client's design is intended for PCs running Windows™.

2 Scope

2.1 What is in this document?

The bulk of this document is concerned with explaining the game through scenarios; or as some like to call them “use cases.”

This technical part of this document contains an explanation of the fundamental game concepts, in part to make the following explanations easier through the use of a standard vocabulary.

Also explained are the highest level architectural components so that the reader can approach the examples of “design through scenario” with an understanding of the underpinnings necessary to support the playing of the game.

Finally, there is a description of the database organisation and the control language used to execute the game’s commands.

2.2 What is *not* in this document?

Low level descriptions of particular interfaces are missing. The primary interfaces lie between the major architectural components, not within them, and these primary interfaces are described herein.

2.3 Revision History of this document

Date	Nature of change
1 May 2003	Changed to the new template. Began work on fleshing out the “how it works” parts. Added information on low level routines, the “thin client,” and spelled out several concepts associated with event queuing.
31 May 2003	Began adding the description of ECL (Empire Command Language)

3 Fundamental Game Concepts

3.1 Player/Nation

Each human player is associated with a “nation” in Empire. The nation is known to the other players by a name of the player’s choosing, and the game is setup to completely conceal the name of the person associated with a nation.

Players log in to play the game using a password authentication scheme, and a player may only be logged in once to the game, thereby preventing three or four people from simultaneously engaging in combat on behalf of a single nation.

3.2 World

Empire’s world is a bit unusual. In an ordinary sense, the world contains a collection of islands and the open water between them, known as the ocean. The islands are randomly sized, and randomly placed in the world at the time the game is set up.

The coordinate system may be thought of as a grid of squares lying in the first quadrant of the plane we know from Geometry 101. The world is square, with its top and bottom edges joined, as well as the left and right edges. This apparent attempt to produce a sperical world actually results in a toroidal world, but with few playing complications and several benefits.

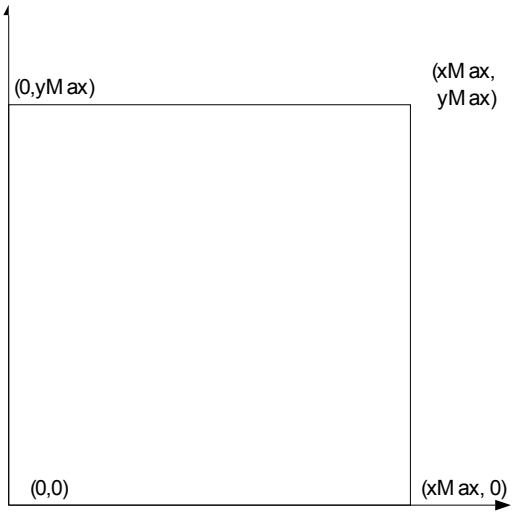

Figure: The coordinate system of Empire

One of the benefits of this system is that no player’s starting point in the game is near or far from the “edge of the world,” and no one is cramped into “polar regions” that would be found in a truly spherical world.

The Empire game engine controls each player’s view of the world. A player is informed about the overall size of the world, but he can only see the neighbourhood in which he has a presence. The situation is extreme at the beginning of the game where a player can only see the island on which he starts.

3.3 Island

The game can be configured to contain a number of islands. The islands consist of a contiguous group of sectors that lie within some rectangle. This number is generally chosen to be about twice the number of expected players. One of the benefits of this arrangement is that each player gets to start out alone on an island, and has the chance to develop the nation's strength for a while before other players are encountered and combat may begin.

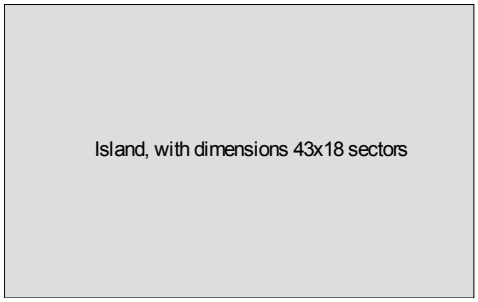

Figure: Example island region

The island shown in the example is made up of a contiguous block of sectors that lie within a rectangle 43x18, measured in terms of sectors. Islands never overlap, and the game has a parameter that controls the minimum distance between islands. The islands are randomly placed (within the constraints of size and separation) within the world.

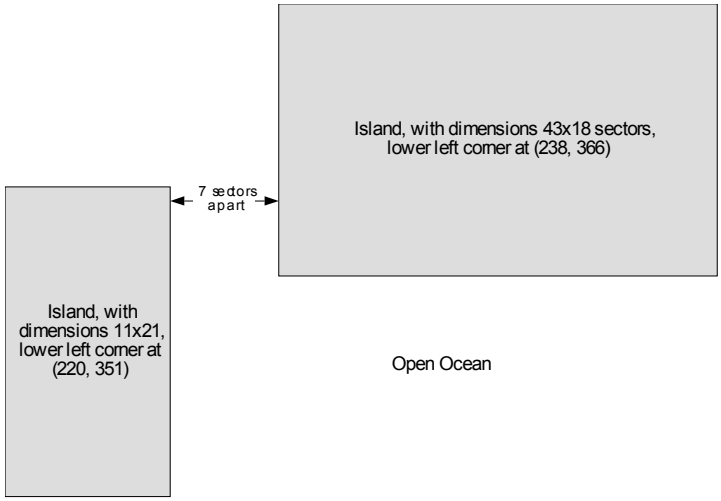

Figure: Two islands, somewhere in the great ocean

When the game is configured by the game engine, part of the process also creates each island's topography. This is a several step process whose explanation is important to the reader's understanding of certain other parts of play. The following figure shows an example.

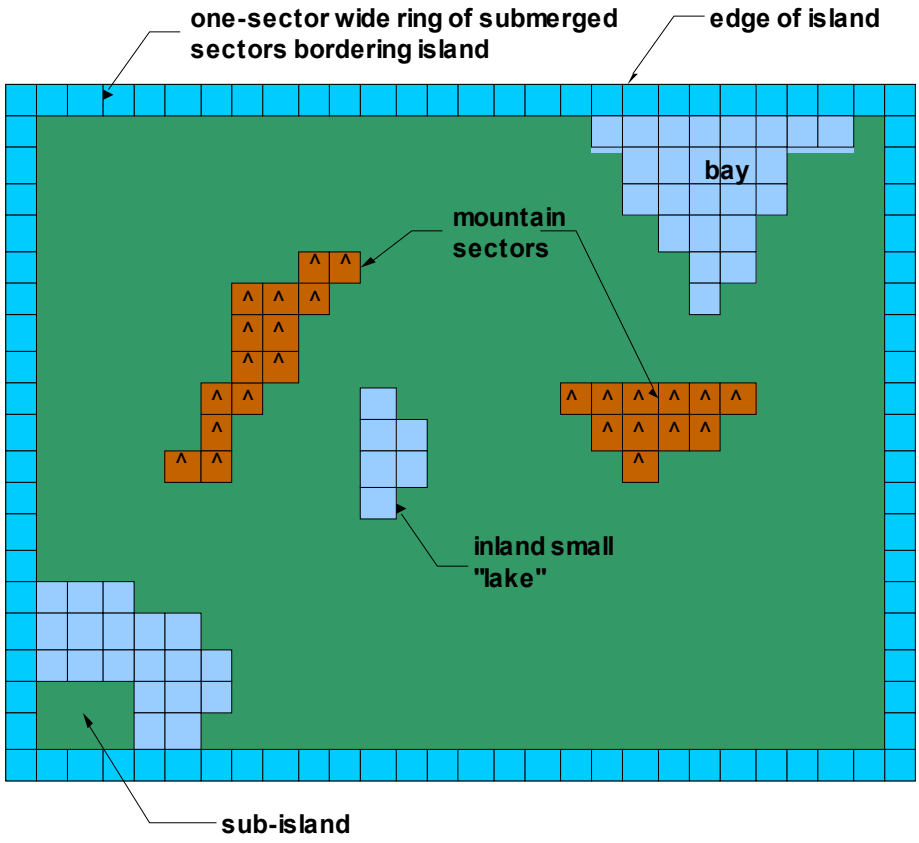

Figure: Drawing of island topographic features.

The outer ring of sectors is always submerged to create a shallow coastal (non-ocean) area around the island. Then, a fractal algorithm creates a mixture of flat land, mountain ranges, and lakes in the remainder of the island. The smoothing nature of algorithm means that most islands wind up with a somewhat irregular coastline (more so than is shown in the drawing), and it is more likely to find mountains toward the interior, particularly if the island is large.

The island on which a player starts is called the “home island,” that player who starts there has the right to name it until the player exits the island.

3.4 Sector

As we have seen, Empire’s world is made up of “sectors,” which in the case of the sectors that make up islands may be defined as the smallest unit of land area that is separately accessible to the command language of the game. Sectors are owned by one player at a time, and they have contents such as population, weaponry, factories, etc. Sectors are also the unit of sustaining damage, so when a sector is attacked the sector is damaged when the contents are destroyed.

There are quite a few types of sectors:

Capitol	Each player has at most one sector designated as the capitol sector on each island that he occupies.

Factories	These sectors are designated to produce various wares of the game, all of which are connected with warmongering. Each factory produces only one type of item. In this version of the game, they are limited to artillery pieces (“guns”), explosives (“shells”), airplanes, and ships.
Mines	Mines produce the universal substance, ore, from which the factories produce items of use. Loaded onto ships, ore becomes the fuel that lets them move about.
Special Sectors	This version of the game supports sectors designated as radar installations, forts, urban centres, airports, and canals.

Table of Sector types in Empire

3.5 Ships

Ships are necessary to move people and materiel between islands. Although the game does have aircraft, they are simply instruments of combat, and do not really carry anything resembling cargo.

Ships come in a variety of types, which correspond to “real world” ships both in name and function. A nation might have several hundred ships in a big game, and these ships may be organised into fleets for the player’s convenience.

Type of Ship	**Purpose/Function in the Game.**
PT Boat	Rapidly getting to another island to “plant the flag.”
Barge	Transporting cargo around a single island.
Ferry	Transporting civilians around a single island.
Mine Sweeper	A dedicated ship for removing mines from coastal water.
Destroyer	Depth charging subs, and laying mines.
Freighter	Transporting materiel across the open ocean.
Liner	Transporting civilians to islands to settle it.
Submarine	Stealthy exploration, and attack against enemy fleets.
Tanker	Transporting fuel, usually with a fleet, to keep the fleet fuelled.
Transport	Transporting ground combat troops across the ocean.
Cruiser	Defense of large fleets, shelling islands from offshore.
Battleship	Destructive offshore assault of islands.
Carrier	Transporting aircraft at sea.

Table of Ships used in Empire

When military are placed on ships, they become its crew. When explosives are loaded, they become the ship’s shells. When artillery pieces are loaded, they become the ship’s guns. When ore is loaded onto ships, the ore becomes fuel.

3.6 Aircraft

The aircraft model in the game is simple. There is only one type of aircraft, and it is the fighter/bomber jet. Aircraft are always located either in sectors designated as airports, or aboard aircraft carriers. Aircraft can be flown from one type of airbase to any other without use of movement time or fuel. To drop a bomb, they must have one “explosive” put

aboard them. To fly the plane, the airport or ship must have military personnel to fly the plane.

3.7 Passage of Time

When a game is set up, a rate of the passage of game time is chosen. The rate of time passage in the game is arbitrary, and the unit is referred to as a "clock tick." The game does not support day and night, nor seasons or weather. From the standpoint of game setup, experience has shown that 100 clock ticks per calendar week is a fairly slow game, and 100 per 24 hours is almost too fast to allow for fully competitive play. A frequently chosen length of a game is 2000 clock ticks.

Time is accrued by ships, and by the capitol sectors on each island. Some explanation of each concept is required, and they are gone over in detail later in this document.

3.7.1 Ship Time

Ships have a maximum speed that is measured in sectors per ship-day. A ship day is a real day multiplied by some number that the person who set up the game decides. If the factor is 1.0, then a ship day is 24 hours, and a ship with a maximum speed of "15" can travel 40 sectors in 24 hours.

Ships that are at rest accrue up to one ship-day worth of movement that can be used for interactive manoeuvring when a player is actively logged on. The game engine keeps ships that are on course moving even when a player is not currently connected to the game.

Ships burn a certain amount of fuel even when at rest, and this is a value that is determined by the length of the ship-day, and the type of ship.

3.7.2 Land Time

Land time is accumulated on a per-island basis. To accumulate any land movement time, a player must have a sector designated as a Capitol on the island. With each clock tick, the Capitol sector accumulates one unit of "movement time." The player can use these units of movement time to reposition materiel on that island, attack enemies, etc.

When a unit of movement time is expended, it becomes "update time." The purpose of update time is to increase population, cause factories to produce wares, and to cause mines to produce ore.

Land time is completely managed by the player during interactive play. However, the game does provide automatic defences from attack. For example, if a sector is bombed and has the ability to throw flak at the incoming planes, this response will take place whether or not the player is actively connected to the game.

3.8 Efficiency

Efficiency is a measure of how well the materiel of Empire works. On the land, efficiency is associated with the individual sectors of each island; each ship has its own operating efficiency.

Land efficiency actually runs from 1% to 100% rather than from 0% to 100% as one might expect. Unimproved land is 1% efficient. After the player designates what type of sector

the new land is to become, its efficiency rises one percent for each clock tick. When a sector becomes 99% efficient, the factory begins to produce its wares. If a sector is attacked, it is quite likely that the sector's efficiency will fall, and the production drops off.

The operating efficiency of ships affects everything about their operation. When first built, a ship operates at 100% efficiency. If a ship is damaged during a conflict, its efficiency drops, and with that drop in efficiency comes reduced maximum speed, and a reduced capability to withstand further attacks.

3.9 Population Growth

Update time causes civilian population to grow. The player can conscript troops on sector, island, or nationwide basis. Conscription does not alter the overall population, just the balance of civilian to military population. Once conscripted, that segment of the population is unaffected by updates.

3.10 Order of play

Empire does not support the notion of turns for players. Any player with movement time can use it at any time without regard to what other players may be doing.

3.11 News / Mail / Chat

The game engine keeps a partial log of player events. These events are stored so that they can be sorted by the players, islands, and ships involved in the incident of interest. From this data, the game engine creates a summary which amounts to a "daily newspaper" that is distributed to all the players.

As in real combat, players may want to communicate for purposes of alliance or taunting. The game provides the ability to send another player a message in an anonymous way, either through POP or instant message chat. This is in keeping with the idea that players may want to remain unidentified.

4 Primary Architectural Components

Since Empire is a multi-player game, it consists of client and server parts. The client parts support the presentation of a player's status and the choices of what to "do," and the server parts support the processing of players "moves," and keeping the game synchronised. Although the players interact via the internet, it is not a "web based" game in the usual meaning of the word.

The following drawing is a high level sketch of the major architectural components.

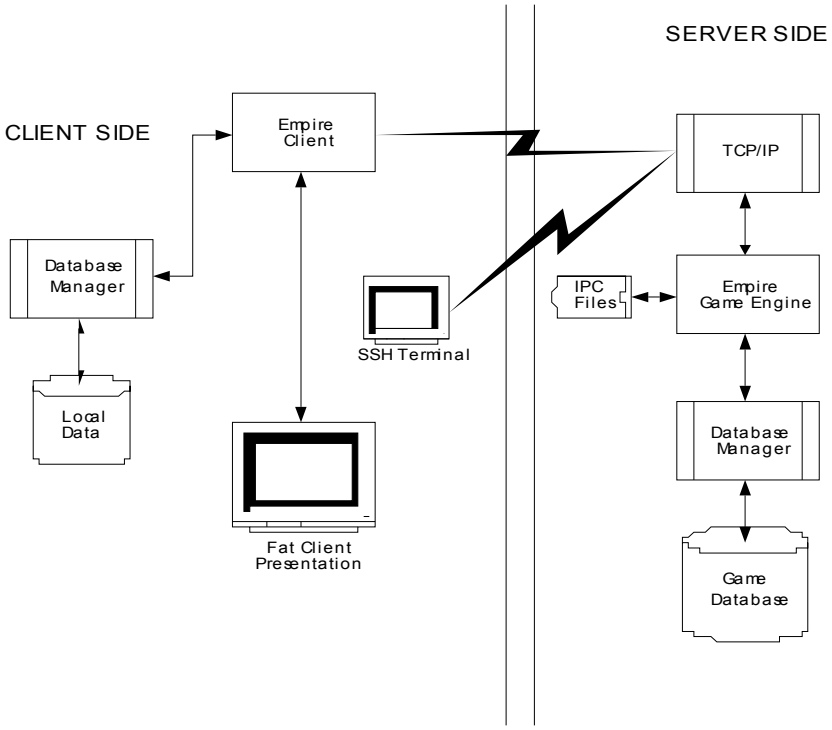

Figure: Highest level view of the architectural components

4.1 Predefined Components

As much as possible, Empire seeks to make use of off the shelf components. The trade off in doing so is that we have assumed the presence of a number of things. These are some examples:

- HTTP[1] as the data transport layer when using web clients
- SSH as the data transport when using the thin client
- XML[2] as the structuring agent for the game's data
- SQL[3] as the interface to the game's persistent data
- Some type of relatively high speed connection between the player and the game's server, whether it is DSL, cable modem, T1, etc.

[1] HTTP: Hyper Text Transport Protocol, a system for safe transport of data from a server to multiple simultaneous clients.

[2] XML: eXtensible Mark-up Language, a language that allows one to send and receive self-describing data.

- A client side programming language capable of simplifying the network access.
- A web browser for the purposes of establishing a login and connection on the client side, as well as maintenance of the game.

4.1.1 Web Browser

A web browser is a convenient wrapper around the log in function, and the Empire web site where updates are distributed, messages are posted, and other activities peripheral but necessary fro playing the game. However, the browser is not used to *play* Empire.

4.1.2 Data Interchange Format

For ease of implementation and extension, the game's data is all encapsulated in XML, as there are standard software components for both creating and reading XML documents.

XML lends itself nicely to Empire's data because there is a strong "container" aspect to the Empire game. Here is what we mean:

A player's view of the world *contains* islands, which *contain* sectors, which *contain* materiel. Because XML is oriented around nested, tagged, self identifying data, this type of common relationship within Empire is particularly easy to represent.

4.1.3 Web Server

The web server presents the public face of the Empire game. It is a direct way to support login validation by user/password pair, and it supports a way to consolidate information for visitors who might be interested in knowing more about Empire. When supporting play of the game, a web server is a type of process that handles multiple simultaneous connections, and thereby greatly assists with a multiplayer game even if it is not traditional "web pages" that are being served to be seen in a browser window.

Most web servers, notably Apache, support extensibility through some type of software plug-in. This ability to handle inter-process communication is in itself a worthwhile feature, so much so that this feature alone would encourage use of a web server to handle the incoming traffic.

4.1.4 Database Managers

Not too many years ago, there were few database managers that ran on affordable machines. The situation has changed, and using a product such as MySQL, PostgreSQL, or OpenBase makes sense. Most of the transactions in Empire are simple, and do not require complex locking strategies.

Currently, we are using MySQL, and the database schema may be seen in an appendix to this document.

[3] SQL: Structured Query Language, a standard language for getting data into and out of databases in a product independent way.

4.2 Game Specific Components

4.2.1 Empire Game Engine (Server Process)

Empire's game engine is given XML documents via the connection maintained between the web server and the player. It processes them in the order in which they are given so that there is never more than one player's command executing against the database at once. The outcome of combat or the change in positions of ships and materiel is reflected in changes to the database, and the results are repackaged in an XML document and set back to the originator.

There may also be background tasks executing while players are engaged. These include updating the game's clock, moving ships that are on course, and executing stored commands that are to be executed when the game's clock reaches a certain point.

The following major section dealing with "scenarios for use" deals in detail with the functions of the game engine.

It is important to understand that the Empire game engine is the only process connected to the database. Therefore, database level locking strategies are not employed. Rather, the queuing process takes place inside the engine.

4.2.2 Empire Game Database

Empire's database stores all the information about the islands, their sectors, and all the ships. It also contains the "static data" for the game: parameters that are somewhat arbitrary (ex: maximum speed of an aircraft carrier), but control the feel of the game.

The only connection to the database is from the game engine itself – individual players do not connect directly to the database, and SQL is limited to the interface between the game engine and the database.

4.2.3 Empire Client

Empire may be played in text mode, via terminal emulation that supports SSH protocol. In this case the client is "thin," and actually resides and executes on the server. This thin client allows for play in situations where the data exchange rate needs to be lowered, and has assisted in the development and writing of the game.

Empire's client is "fat." There are several reasons for this:

- The richness of the information presented to the players.
- The variations in how different browsers render pages.
- The need to allow players to plot strategy and test out scenarios while not connected to the game.

These factors would indicate that a traditional client application is the obvious place to start. For the initial development, we are leaning toward VB .NET in the belief that most players are likely to own a Windows PC of relatively recent vintage, and the runtime components of the .NET framework from Microsoft are freely redistributable.

4.2.4 Presentation Layer

Ordinarily, one might leave this item out of a discussion of high level architecture. However, in the case of Empire the topic is germane. From an architectural standpoint, the presentation layer is something that may need to undergo a bit of evolution, particularly as new features are added to the game. The methods for presenting complex information in a small space (such as a CRT) are always being researched, and with the research come new opinions about what system works best.

As a result, we can see that the user interface may change quite a bit more often than the underlying game, and we might even want to provide alternate displays akin to the “skin” concept that is being offered in some software.

4.3 Architectural layers

The following drawing represents the “layers” in the separation between the concepts of the ideas represented in the game, and the translation of these ideas into standard computer data structures.

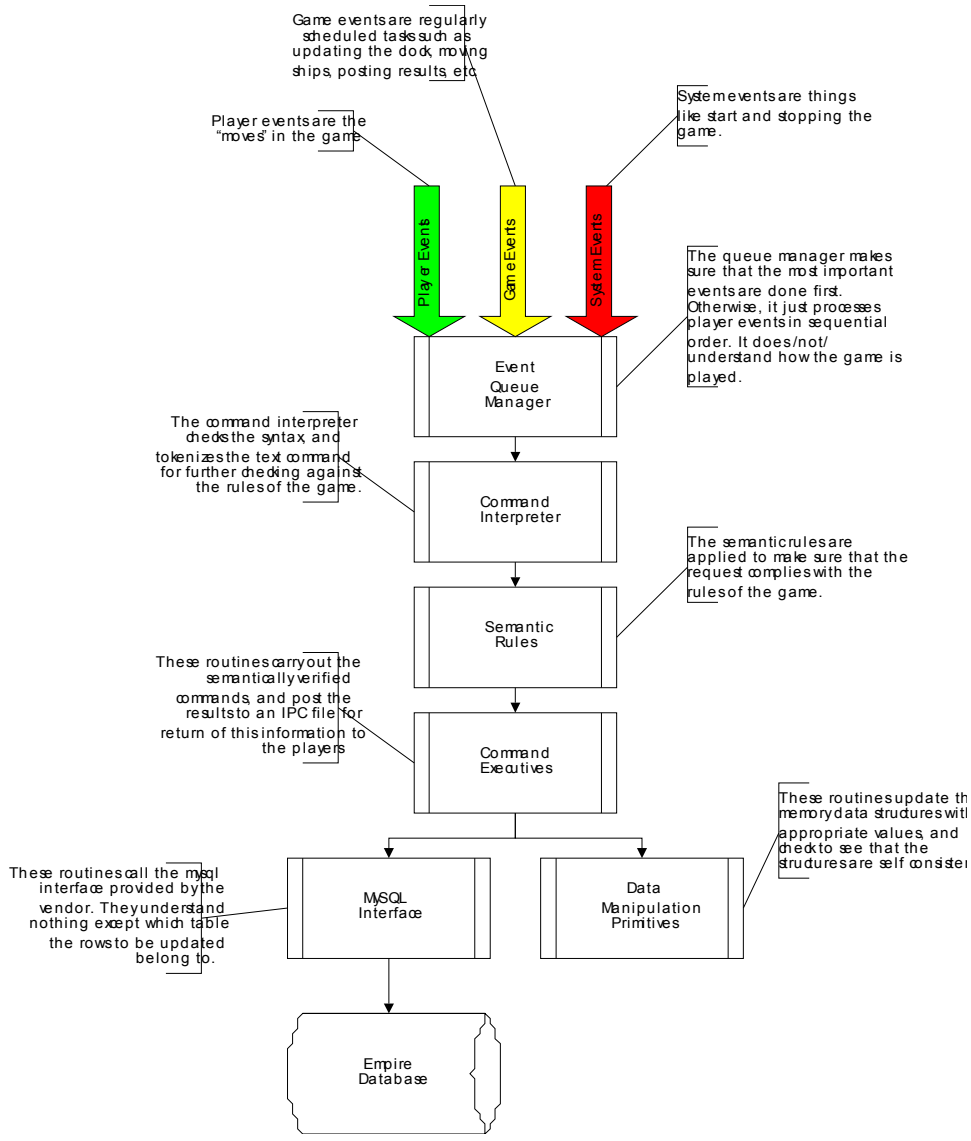

Figure: Architectural layers

The important thing to understand about the drawing is the strict partitioning of knowledge about how to play the game, as well as the mere separation of the code into modules. For example, the command interpreter is the only routine that understands the syntax of the game's language (ECL), and if that syntax is changed only this module need be modified. Similarly, the semantic rules engine contains the knowledge about what constitutes legal play, and does not need to know how the players express their moves.

5 Use Scenarios and Explanatory Text

5.1 Game Creation and Configuration

5.1.1 Create a new game.

The game administrator, historically referred to as “The Prophet” in previous versions of the game, establishes the parameters for a new game.

The SQL daemon processes the script (see appendix) that builds the database for all of Empire’s data. If the physical server is to have more than one game running on it, then the administrator may have to edit the “CREATE” statement in the SQL script.

Next, the genesis module (not shown in the architectural diagram) allows the administrator to change the default parameters of the game. The genesis module updates each changed record in the database.

When all the parameters have been chosen, the genesis module populates the game database with a record for each island, and each sector on each island. When this process is finished, the database is ready to support play.

5.2 New Player

5.2.1 New Player Signup.

The Empire web page will contain a signup button for new players. Pressing this button will take the new player to a page where he will fill in the necessary information: Nation name, email address for the game to contact the player, etc.

If this information does not conflict with the information for existing players, then a new player record is created in the player table. The game then mails a generated password to the player at the email address supplied. The game updates the player record with the message digest (hash) corresponding to the player’s password.

5.2.2 Exceptional conditions

If the maximum number of players has already been reached, new players may not be added to the game.

If all islands are occupied, but the maximum number of players has not been reached, the player is given the option of whether he wants to join this game.

5.3 Globally Available Information

Some information is globally available. These facts, available to all players at any time during the game and without qualification as to status include:

- The circumference of the world.
- The maximum number of players in the game.
- The association between nation names and player numbers.
- The current time of the game clock.

- The rate at which time is passing in the game.
- The ship-day factor.
- A topographic map of any island the player occupies.
- The island position and dimensions of any island that falls within a player's ship or land based radar range.
- The costs and maximum concentrations of materiel.
- A complete inventory of one's own materiel.

5.4 Connecting to Play

5.4.1 Login.

Player supplies his user name and password on.

The game verifies that the password matches the message digest stored in the database. It also checks to see if that player is already logged on.

The Empire client runs as an application/service on the player's machine, and this logon activity establishes a connection between the player's machine and the game.

5.4.2 Interruption of the Network

In cases of a client or server crash, or loss of network connectivity between client and server without a crash, play is interrupted, and a new logon sequence must be begun.

5.4.3 Logout.

When a player wishes to logout, the player can do one of two things. The client can issue a logout command from inside the client. Alternatively, the player can log out from the web page that is presented after login.

5.5 Inception of Play for New Players.

5.5.1 First Connection.

After a player receives a password, play may begin. A few things are different on this first connection since the player has no information in the game's database of islands and sectors.

When a player connects for the first time, the game updates the player's record in the database with his "date of birth" from the standpoint of the game.

The game engine looks through the file of islands, and finds one that is unoccupied. The game then chooses a random sector of flat land, and gives the player the default number of civilians in that sector, designates it as a Capitol, and supplies the sector with the default starting units of movement time.

5.5.2 Exceptional conditions

It is possible the between early sign up and a couple of calendar days later when a player might actually be ready to begin, that all islands are occupied. In that case, the game

engine will select from the islands with the minimum number of players (quite likely, 1 player) on the island.

5.5.3 Player's initial view of the game

The game engine controls the view of the world. After the game successfully establishes a player's initial presence on some island, the player is sent a description of the island. This information is the same "packet" that one would get regardless of the island that one is on.

In the initial state, the new player gets a sector by sector map of the island showing elevation, the one sector that is the new player's capitol, and an indication of what nations occupy other sectors on the island. A player may not see what type of sector an enemy sector is unless he owns an adjacent sector.

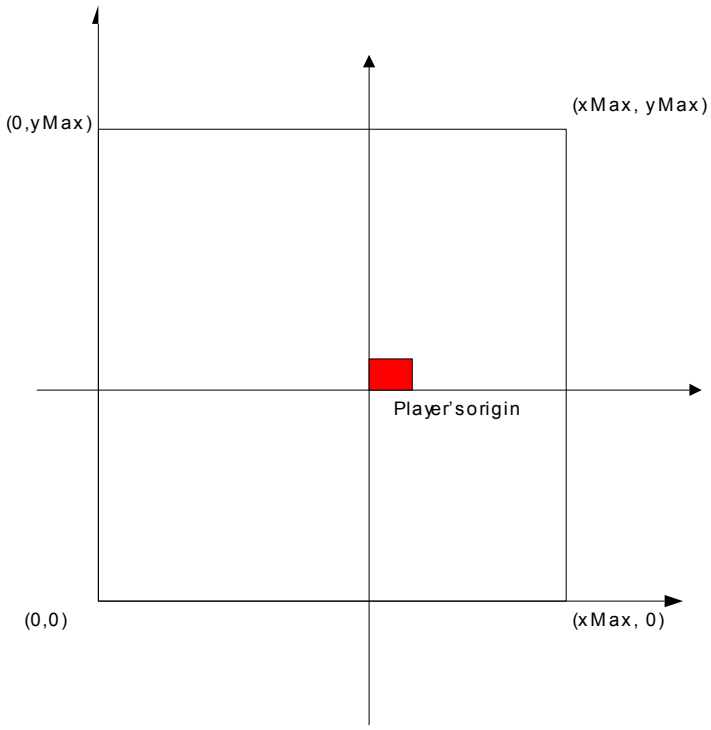

Figure: Initial view of the world.

The initial description of the island tells the player nothing about its location in the world, and in fact, the lower left corner of the player's home island becomes the "centre of the world" from his point of view. The drawing above shows how a player's coordinate system would be significantly different from the absolute coordinate system of the world.

5.6 Inter-player and Game-to-Player Communication

5.6.1 Players contacting other players.

One of the few clues offered to the player is that a new player knows he is "player number 12 of 30," for example.

A player can choose to send a message to any other player by player number. When this request is received by the game engine, the player's registered email address is looked up, and a standard email message is composed to on send.

5.6.2 Cooperative Play

The notions of alliance, treachery, deceit, and team play are all a recognised part of Empire. Empire allows for players to transfer planes, fixed materiel, people, land sectors, and ships to another players. Arrangements to do so are made via the game's communication channels. The transfer is executed by appropriate actions by the sending and receiving players in the game.

5.6.3 Game contacting players.

The game sends out a newsletter summarising recent game events to the active and deceased players. Reading this newsletter is a valuable way to determine which players are at war with each other, and which players occupy which islands.

Players may wish to note that if they are not engaging in combat, their activities will go unreported in the game's newsletter.

5.7 Passage of Time

5.7.1 At first play.

The parameters of the game will give each new player a number of movement time units to start the game. Typically, this might be 100 units. This gives a new player a considerable amount "to do" at the beginning of the game.

Movement time units accrue to the Capitol sector of the player's home island as time passes, and at the rate that is associated with the particular game.

5.7.2 Movement vs. Update time.

As a player uses his movement time units to shuffle materiel, attack neighbours, etc., these units of time are then shifted to the update side of the ledger of the Capitol sector on a one-for-one basis. For example:

Suppose a new player starts with 100 units of movement time in a game where one tick of game's clock is 15 minutes, or as one would more likely term it, "96 clock ticks per day." During the first hour of play, the player distributes some of his population across sectors near to the Capitol, using 45 units of movement time in doing so.

The player's time accrual in the Capitol sector is now 59 (55 remaining + 4 accrued as time passes) units of movement time available, and 45 units of update time.

The following table shows common game activities and the number of movement time units expended in doing so.

Game activity	Movement Cost
Move into unoccupied adjacent sector	1
Attack an occupied adjacent sector	2
Move troops or materiel into mountain sectors.	5

Move troops or materiel into or through a partially efficient sector.	1 * (1-efficiency) I.e., 0.5 units for a 50% efficient sector, 0.1 for a 90% efficient sector, and so forth.

Figure: Common game actions and associated movement time expenditures.

5.7.3 A more complete explanation of Update Time

When spent, update time causes the population to grow at the rate prescribed for the game. In most cases the population multiplier is $(1.02)^{\text{update-ticks}}$. The following table shows the resultant population for a sector of 100 civilians after the given units of update time are used.

Update Units	Population after Update
10	121
20	148
30	181
50	269
100	724

Figure: Population change with updates of varying sizes.

The expenditure of update time also causes the efficiency of each occupied sector on the island to increase by 1% if it is not already at 100% efficiency. Sectors that have just been occupied or captured from an enemy are set to 1% efficient, and thus take 100 units of time passing before they can reach maximum efficiency. Sectors that have been damaged by combat might be at any level of efficiency below 100%. These sectors are "repaired" by the expenditure of update time.

Once a factory sector is at 100% efficiency and has 100 or more civilians living there to operate the factory, it begins to produce production time units – one for each unit of update time that is expended. These production time units, or "production points," are used to build whatever the appropriate wares are for that type of factory.

Sectors designated as ore mines begin to produce ore once their sectors have reached 100% efficiency and have 100 or more civilians in residence. The game "magically" distributes this ore to all 100% efficient factory sectors where it is manufactured into appropriate wares. On any island, a player must have enough mines to produce a bit of excess ore since this excess ore is used as fuel for the ships.

5.7.4 Limits on time accrual and expenditure

Each game has a fixed limit of how many units of both movement and update time can accrue in a Capitol sector before some of it must be used. If a Capitol sector is "full," time passes in the game without any changes being made to the Capitol sector. Time accrues on a player-island by player-island basis; there is no facility for transferring movement or update time from one island to another, nor from one player to another.

Since the Capitol sector "holds" the movement time, if the Capitol is lost to an opponent's attack, the player loses all ability to execute commands that expend movement time until a new Capitol sector is designated.

One situation that arises frequently is that a player has completely occupied a home island, and there is no great need to use movement time to actually move anything about.

A player can always transfer movement time directly to the update side of the ledger without actually moving anything.

5.7.5 Game activities that consume neither movement nor update time

Not every activity requires movement time. Activities that require no movement time may be made anytime during the game, even when a player has no movement time or when his Capitol has been captured. These include:

- Firing artillery pieces.
- Loading docked ships with materiel and people.
- Building factory wares from accrued production units.
- Doing anything with ships, whether they are docked or not.

5.7.6 Efficiency and land operations

If a sector is less than 100% efficient, it is easier for an enemy to invade and conquer. From an economic standpoint, low efficiency means that factory sectors that are less than 100% efficient do not produce new wares.

5.8 Ship time

5.8.1 At creation

Each new ship is supplied with one day of movement time at creation. This block of time is equivalent to the number of sectors it can move in one day. The grant of initial time allows a player to undock a ship after it is built, and to move it some distance away from the shore.

5.8.2 Time accrual

Each ship accrues the ability to move a certain number of sectors per ship-day. Generally, this factor is set so that an average speed ship (22 sectors/ship-day) can travel between islands in about 24 hours. For many games, a ship-day is a day.

A ship cannot accrue more than one day's worth of movement time. If a ship is "on course" at maximum speed, it accrues no time because the time is being spent as it accrues.

5.8.3 Fuel Consumption

All ships consume fuel whether or not they are moving. The data for fuel consumption are contained in the game's database. If a ship runs out of fuel while it is at sea, it begins to rust. When a ship is fully rusted it is sunk, and the record of the ship is removed from the database.

5.8.4 Efficiency and ships

Efficiency of less than 100% affects most functions of the ship. For example, a ship's maximum speed at any one time is equal to the maximum speed for that class of ship times the efficiency. Efficiency affects the efficacy of the ship's guns, but not their range.

To repair a ship (i.e., restore it to 100% efficiency), it must be brought to a dock (ship factory sector) owned by the player, and there it must be refurbished using the production time units accrued in that sector.

5.9 Factory operation

5.9.1 Ore mines

Without ore, there can be no wares made, and there would be no fuel for ships. The simplification provided by this one universal substance from which everything is fabricated is important to the playability of Empire. With each expenditure of update time, the ore mines pump an endless supply of ore from the ground, and it is instantaneously transferred to factories that need it.

Each sector has a characteristic called its “sample rate.” The sample rate is a randomly generated characteristic that tells for this sector how many units of ore come up from the ground per update time unit expended. Clearly, players should designate sectors as mines where the sectors have a high sample rate.

For there to be some ore left over for the ship’s fuel, there must be a small excess of ore produced per island beyond the minimum amount necessary to operate the factories.

5.9.2 Explosive, Artillery and Plane Factories

These three factory types have in common that a player need not do any specific building of wares. These factories instantaneously change production time units into the wares they produce. In the case of explosive and artillery factories, the explosives and artillery pieces are added to the inventory of things in that sector up to the maximum concentration per sector.

Plane factories are slightly different. Plane factories produce a plane each time their accrued production time hits the cost of a plane, but the planes themselves appear as the contents of the airport sector nearest them on the same island. If a player has no airport sector on the island, the production time units increase with no planes being produced.

5.9.3 Ship Factories (Docks)

As we have seen, creating a navy involves a bit more of a selection process on the part of the player. As a result, these sectors accumulate production time units until the player issues a command to build a particular kind of ship at that sector.

When ships are produced, they are empty. They are created with a location the same as the sector that produced them. For any materiel to be place on a ship, it must either be docked (and filled from the associated land sector), or it must be in the same sector at sea with another ship that is capable of tending it.

5.10 Automatic actions provided by the game

One might ask, “What takes place while a player is not logged on?” The answer is that the Empire game engine provides a number of automatic actions consistent with sane play. Here are some examples:

Ships that are in motion remain in motion and on course unless they run out of fuel or encounter land sectors. Movement time accumulates in the capitol sectors on all islands for all players at the appropriate rate.

If a sector that contains artillery pieces, explosives, and military is attacked, that sector will return fire on the appropriate target. In other words, if the sector is bombed, the flak is directed at the offending aircraft; if it is shelled from a ship or another land sector, then that ship or that sector is the target.

An aircraft carrier will attempt to defend any ship within the flight radius of its aircraft. Likewise, cruisers will attempt to shoot down aircraft that attack any ship within its guns' range. See the section concerning sea combat for more information.

5.11 Combat

As one might expect for a war game, a tremendous amount of Empire is concerned with combat, combat resolution, and the mechanisms of letting the players know the outcome.

5.11.1 Basic principles

A player must be logged on to engage in combat. This does not seem like an unnecessary restriction on play since it would seem that the "uses and gratifications model" of play for Empire would indicate that most players would want to be actively connected to execute their own military plan.

Outcomes of combat are determined by sensible objective rules, with each rule having only a small uncertainty window.

Additionally, the game does not prevent one from attacking one's own sectors or ships. As in all cases in the real world, "friendly fire" incidents are generally devastating.

5.11.2 Concepts governing land combat

The following is a table of the basic rules that are invoked to resolve combat and a list of the scenarios that are governed by the rule.

Rule	Scenarios Governed
Efficiency of both the attacking forces and the defending forces figures into result	All except bombing runs. Individual aircraft are construed to be 100% efficient.
Defending forces have a 3:2 advantage.	Infantry and artillery ground combat. The assumption here is that the defending forces can see the point of origin of the attack.
Last military occupant of a sector cannot be eliminated by bombing or artillery.	All. Forces players to make a ground assault to capture territory.
Civilians are removed as collateral damage.	The ratio of military to civilians killed varies on the type of assault, but some civilians are always killed.
Efficiency always drops to less than 100% after an attack.	Ships and sectors.
Accumulated production in factory sectors is wiped out by any attack.	Factory sectors.

Materiel in a sector can only be captured by a ground assault. Artillery bombardment and bombing tends to destroy materiel.	Sectors.
Mountainous terrain is difficult.	All. It is difficult to attack a mountain sector; it is likewise difficult for a player to move materiel into a mountainous sector.

Rules governing basic combat.

5.11.3 Concepts governing sea combat

Sea combat is quite a bit different from land combat for two reasons: [1] the granularity level of a ship is somewhat smaller than that of a land sector, and [2] there are many different kinds of ships, and they interact in different ways.

5.11.3.1 Measuring distance on the ocean

A ship's position is measured by the game as a floating point pair of coordinates, quite unlike measurements on the land. A ship is considered to be "in" the sector named by the integer portion of this pair of floating point numbers. An unlimited number of ships of any number of players will "fit" into a water sector.

Ships of the same nation that are in the same sector can "tend" each other, meaning that the player can move materiel from one ship to another one in the same sector. The most common use of this feature is refuelling, although ships can be loaded with other materiel while at sea.

Finally, when ships are travelling across the open ocean and reach a land sector of an island they halt. However, they do not "dock" at the land sector, even if the sector in question is a sector owned by the same player that owns the ship. Thus, a ship whose location coordinates put it in a land sector may either be docked or undocked. A ship that is docked may unload its cargo; a ship that is undocked may not.

5.11.3.2 Fleet organisation, and mutual defence

The game contains a mechanism to allow a player to group several ships to simplify navigation. For example, a player might want to designate several ships to travel together to a neighbouring island for the purpose of an invasion.

[1] Ships that are within each other's gun range will mutually defend each other if attacked by land based artillery or the guns of other players' ships.

[2] Ships that are within the flight radius of carrier based planes will be defended by that carrier if the ships are attacked from the air.

[3] Ships within five sectors of a depth charge capable ship will be defended from submarine attack if the attacking submarine is also within five sectors of the destroyer.

5.11.3.3 Ship to ship shelling

Ships may fire at other ships within range. The result depends on the distance to the target, and the quality of the guns. The quality of the guns is dependent on the type of ship as well as its efficiency.

There is no such thing as a free shot at a ship unless it is unarmed. It is assumed that ships that are close enough to each other to exchange shells "exchange fire" with one of

the ships simply being the instigator. The factors of distance and gun quality also affect the return fire.

5.11.3.4 Shelling between ships and land

When ships shell a land sector, if they are with range of the sector's return fire all the guns in that sector return fire simultaneously. Some sectors are clearly much more "powerful" than others, and ships have no way of seeing into a sector to determine the consequences of shelling it ahead of time.

5.11.3.5 Air attack and flak at sea

It is common for ships to be attacked at sea by aircraft, either land based or from another player's carrier. The attack is simple from the player's point of view, and all that is required is that the source (carrier or airport land sector) have planes, pilots (military population), and explosives (to be used as bombs).

Defence is more complex. The game first determines if the target of the attack is within the flight radius of a friendly aircraft carrier. If so, that ship's planes come up to meet the intruding aircraft in numbers at most equal to the number of incoming aircraft. If any incoming planes survive, then they are hit with one round of flak from every friendly ship within flak range. Only after penetrating those two defences do the incoming aircraft drop bombs on the ship that is the subject of the attack.

5.11.3.6 Submarines and torpedos

Submarines, when submerged or on the surface, may fire torpedoes. Torpedoes are quite destructive, so players have a strong incentive to use them where possible. When a submerged submarine shoots a torpedo, its position is immediately known to all ships within five sectors, whether or not they are sonar capable.

The game engine resolves this type of combat by considering the number of depth charging friendly ships within the radius of the attack, the type of ship being attacked, and the distance of the attacking submarine. The submarine is attacked with depth charges whether or not the torpedo hits the mark.

A submarine that is stationary and submerged is construed to be resting on the bottom of the ocean, and it is therefore invisible to enemy sonar. If depth charges are dropped in the sector where a submarine is resting on the bottom, they will be less effective than if the position of the submarine were known.

5.11.3.7 Transfer of ships at sea.

While playing earlier versions of the game, a flaw was discovered in the transfer of materiel. Originally, the transfer mechanism was executed solely on the "from" side, with no corresponding actions of the part of the recipient. Clever players discovered that one could do the following: [1] Sail a ship just within the flight radius of an enemy carrier. [2] Re-flag the ship to the enemy nation. [3] Begin attacking the newly transferred ship with one's own planes.

This manoeuvre would cause the enemy carrier to begin defending this ship. Repeated attacks would gradually deplete the planes of the opponent's carrier, thus leaving the entire fleet open to attack without the resource of air defence.

Thus, any transfer of materiel now requires acceptance by the recipient. While not so necessary for land based materiel (there is no similar “back door” to land play), the transfer-accept paradigm has been found to be useful for standardisation.

5.12 User Interface Requirements

A game as complex as Empire must have a substantial user interface component. In fact, the level of effort to produce a satisfactory user interface for Empire is around three times the level of effort for the server piece. Fortunately, some stepping stones along the way reduce the need to have a “big bang” delivery of the user interface. These stepping stones are even more important when one considers the difficulty in testing a system as complex as Empire.

5.12.1 Complexity & and the Two-man Cockpit Problem

Modern passenger jets are designed to be flyable with a two man crew. Older planes used at least three people. The aircraft industry discovered that the cockpit user interface had to change when few pairs of eyes were looking at the “dials and gauges.” Empire is similarly complex, and the instrumentation necessary to play it falls into these interacting categories:

Economics: “Are there enough mines on the island to keep the factories supplied? What is the balance of production between planes, ships, artillery, explosives, population growth?”

Geographic: “What does my map of the world look like? What is the detailed view of a particular island I occupy? Where are my ships?”

Military: “What is my troop strength? What level of damage will be inflicted by an attack? What can I know about adjacent sectors occupied by the enemy?”

Forecasting and projection: “When will my ships arrive at the new island? How long will it be until my factories are productive? When will I have enough production to build an aircraft carrier?”

Redistribution and movement: “I want to distribute my artillery and explosives around the island rather than having them remain in the factories. I want to load a new ship with supplies. I want to bring my battleship close enough to shell the island.”

5.12.2 Data Representation

The primary challenge in the Empire UI is scale. As a Commander-in-Chief, a player needs a view equivalent to that of a world map. As a player takes on the role of a General, there is a need for a view of perhaps 96 sectors at a time (12 wide and 8 high on normal 4:3 displays computer displays) to resolve intense combat to a useful level.

There is also a need to have an “island scale” view of the world, and to provide a player with the ability to rapidly switch views between several occupied islands. Naval scenarios and activities are similarly scaled. Experiments conducted by DEC and Adobe have shown that rather than having continuously scaleable views, most users of visually intensive software would prefer to have a short but useful list of fixed scales. This approach would seem to be applicable to Empire.

The secondary challenge is dimension. For example: it would be useful for a player to toggle between economic specifics and combat details in order to avoid visual clutter. The UI challenge in this case is making sure there are enough visual clues that a player knows what view is currently being displayed. Extensive reliance on colour could be a problem, since 10% of the primary audience (male) has compromised colour vision.

The fat client allows a degree of freedom for the designer that is absent in web based games. The entire processing power of the client CPU may be brought to bear on rendering the images without requiring network traffic. The amount of CPU power necessary to convey to a user that he has been attacked by player #n's cruiser at sector (x, y) of island z is negligible compared with the effort to render that information visually.

5.12.3 Target Platforms for the Client

The only reasonable conclusion about the **fat client** side of Empire is that players will either have, or have access to, a Windows PC. Since gamers are likely to have up-to-date PCs, we will further assume that the client be developed for Windows 2000/XP and later editions of Windows.

The **thin client** is a program that actually runs on the game's server and handles text only. Given that at some level the data is moving back and forth via XML, any platform could make use of the thin client interface.

5.12.4 Offline Play

Under Construction

6 Low Level Routines

6.1 Database

The following are descriptions of the database primitives. Note that there are a finite number of sectors, islands, and players that are created at the beginning of the game. The creation of these database entries is only performed in the Genesis module where they operate directly on the database; therefore they are not documented here. Ships, on the other hand, are frequently being created and destroyed.

Expressed in C++, these routines consist of a family of member functions named "Transact." Each takes three parameters:

1. A reference to the data to either be (re)placed in the database.

2. An enumerated type which is one of get, put, update, or delete. This parameter defaults to "get." Unlike many database systems, there is no support for particular "modes" of reading the database – all are done by absolute key. For example, there is no reason to believe that supporting serial reads would be helpful as most operations initiated by players involve random records in the tables.

3. An integer expressing the number of records of data, which defaults to one.

The functions all return an integer stating how many records were successfully processed. In most cases, one would expect the return value to be equal to the number requested in the third parameter.

In all cases, it is the responsibility of the caller to ensure that the data is correct. Each class object representing players, sectors, ships, and islands has a Verify() member function which does an integrity check of the object. Typically, you would see something like this:

```
SECTOR Sector;
SINGLETON <EMPIREDB> Database;
bool OK = false;
 …
if (Sector.Verify())
  OK = Database.Transact(Sector, DBUPDATE, 1) == 1;
```

The data structure (object) that is used as the first argument is always subject to modification. The Transact() member function of the EMPIREDB object examines the contents of the data structure and the requested access mode to determine an appropriate value to use as a key to the database.

6.1.1 Update a sector

A sector update uses the self-referential island coordinate to locate the appropriate sector. This is particularly handy when one is executing the most common type of sector transaction which is to "update" its information because of the passage of time, or an attack, etc.

6.1.2 Create a new ship

Ship creation involves adding a record to the ship table. It is critical to call the Verify() member function prior to adding a ship to the database. The ship's number is created by

the database as its absolute key, and that number is returned inside the ship's data structure.

In terms of its function in the game's logic, the ship number is unique among the database keys in that it is actually used by the players. Player's may refer to sectors by their coordinates (Island 7, sector [23,1]) rather than what the game considers to be the absolute sector number. However, ships are referred to by their number. The numbers are never reused, thus making it safe to do so.

6.1.3 Update a ship

Updating a ship is quite similar to the process for updating a sector. However, a great deal of the updating of ships (moving them along through the ocean) is done by the game rather than by the players.

The game-level updates generally involve consuming some fuel, and moving the ship on its course if it is not stationary.

6.1.4 Delete a ship

Ships are frequently the victims of combat. They also can be scuttled by their owners, or simply rust away because they are left unattended or unfuelled. Deleting a ship does not actually remove the record from the database – it simply marks the record as inactive. This allows the game to peruse the ship table and take care of the deletes in one pass as a background task.

6.1.5 Update a player

Player updates are solely done by the game itself. Logon and logoff are the primary activities. However, players may initiate some changes on their own such as changing their email address and instant message handle.

6.1.6 Update an island

Updating an island is generally performed only to change the island's name or the "owner."

6.1.7 Update an island's clock

The update of an island because a player issues the update command is one of the most CPU and I/O intensive processes in the game. Typically, it is initiated by the player, and the only parameter is the amount of "update time" to be expended. The game engine finds every sector on the island owned by the player making the request. Then, the amount of change is calculated for each sector, and all of these sectors are updated in the database.

On a sector by sector basis, here is what happens:

If the sector is less than 100% efficient, the update time is expended enough to bring it to 100% efficiency.

Once a sector meets all of the following criteria

[1] 100% efficient

[2] contains 100 civilians

[3] is a factory

[4] is receiving ore to produce wares

… the update time is used to run the factory.

6.1.8 Create a news item

News items can be created as a result of game events, or as a result of a player's direct request to put something into the news.

6.1.9 Update news file

The news file is periodically maintained by the game to delete old records, or to summarise several records into one.

6.2 Event Queuing

As has been stated elsewhere in this document, the Empire game engine controls the serialisation of the game's events through managing a queue. One might say that the entire purpose of the client program is to post events to the queue.

There are, in fact, three queues. They are listed here in order of decreasing priority:

6.2.1 System Events

System level events may be originated in one of three ways:

[1] The game's console may post one in the case of direct intervention by the game's administrator.

[2] The game may originate its own cessation of play due to the timer. For example, cleaning up the ship and news tables are periodic tasks that need to be done perhaps once a calendar day.

[3] The ticking of the game clock. This is an event that is created by the game daemon itself.

System-level events are processed in their entirety before any events in the other queues.

6.2.2 Game Events

Game-level events are easiest to understand by examples:

[1] The need to synchronise the state of the game because time has passed. Periodically, the game needs to update ship positions on its own. This may also happen because a player sets the focus to a ship that is on course.

[2] The need to resolve conflict events before processing any other player-level events. For example: if a player attacks a neighbouring sector the outcome of the return fire and the damage done must be processed before any subsequent player events. This is

because the object of the attack might be completely destroyed, thereby altering the game's database.

[3] Any operation that causes a write to the database is a game-level event.

The game event queue must be emptied before more player events are processed.

6.2.3 Player Events

Player events are created when a player executes a command. The command need not change the state of the database; for example, the player may be requesting a map of an island. Player events are placed in an STL priority_queue data structure, where the time of posting is used to sort the events.

Events in this queue often create other events, many of which are game events, and therefore assure that the outcome of a player-level event is processed in its entirety before subsequent player-level events.

6.3 Posting Results to Players

Players obtain information about the results of their actions by monitoring an IPC[4] file. The client modules go to sleep after posting events, and await the wake up call resulting from entries being made in the IPC file.

After wake up, the messages in the IPC file are written to an appropriate device in the player's address space (for example: a console window).

[4] IPC: Inter-Process Control file. Also known in Linux as a "named pipe."

7 Empire Command Language (ECL)

The following is a list of the “keywords” or “command types” in ECL. Each keyword is described as to both its purpose, and its grammar. Additionally, an explanation of the effect of the commands is given in terms of the required changes to the underlying database.

NB: in all cases, the game engine verifies both the syntactic and semantic correctness of the commands issued. For example, if a player wishes to scuttle ship #45, the game engine verifies that the player who issued the command owns ship #45, and that it is in fact an active ship that can be scuttled.

ECL is not case sensitive. All input is converted to lower case for evaluation by the parser, and this fact is represented in the examples.

7.1 EBNF Grammar

```
# Here are some general rules I tried to follow:
# QUANTITY is always optional (default is "all") and refers
# to materiel. USE is always optional (default is "as much as
# required") and refers to clock ticks.
#
# I think the commands are all structured as
#    verb->object->qualifiers

command_string ::= command+

command ::= assault_command
 assign_command
 attack_command
 bomb_command
 depthcharge_command
 designate_command
 distribute_command
 dock_command
 emerge_command
 fly_command
 laymine_command
 load_command
 move_command
 ping_command
 refurbish_command
 sail_command
 scuttle_command
 setcourse_command
 setname_command
 setspeed_command
 shell_command
 submerge_command
 sweep_command
 tell_command
 tend_command
 torpedo_command
 transfer_command
 undock_command
 update_command

assault_command ::= ASSAULT island_id FROM ship_id
assign_command ::= ASSIGN ship_id TO fleet_id
attack_command ::= ATTACK island_posn FROM island_posn [QUANTITY
number]
bomb_command ::= BOMB target_descriptor FROM target_descriptor
[QUANTITY number]
build_command ::= BUILD ship_type FROM island_posn
```

```
depthcharge_command ::= DEPTHCHARGE ship_id FROM ship_id
designate_command ::= DESIGNATE island_posn TO sector_type
distribute_command ::= DISTRIBUTE materiel FROM co-ord TO
range_of_sectors [QUANTITY number] [USE number]
dock_command ::= DOCK ship_id | fleet_id
emerge_command ::= EMERGE ship_id
fly_command ::= FLY FROM target_descriptor TO target_descriptor
[QUANTITY number]
laymine_command ::= LAYMINE FROM ship_id [QUANTITY number]
load_command ::= LOAD ship_id WITH materiel [QUANTITY number]
move_command ::= MOVE materiel FROM island_posn TO island_posn
[QUANTITY number]
ping_command ::= PING from target_descriptor
refurbish_command ::= REFURBISH ship_id
sail_command ::= SAIL ship_id | fleet_id TO co-ord
scuttle_command ::= SCUTTLE ship_id | fleet_id
setcourse_command ::= SETCOURSE ship_id | fleet_id TO co-ord |
island_posn SPEED number
setname_command ::= SETNAME island_id | ship_id TO string
setspeed_command ::= SETSPEED  ship_id | fleet_id TO number
shell_command ::= SHELL target_descriptor FROM target_descriptor
[QUANTITY number]
spread_command ::= SPREAD population FROM co-ord TO range_of_sectors
[QUANTITY number] [USE number]
submerge_command ::= SUBMERGE ship_id
sweep_command ::= SWEEP co-ord FROM ship_id
tell_command ::= TELL player_id string
tend_command ::= TEND ship_id | fleet_id FROM ship_id WITH materiel
[QUANTITY number]
torpedo_command ::= TORPEDO ship_id FROM ship_id
transfer_command ::= TRANSFER ship_id | fleet_id TO player_id
undock_command ::= UNDOCK ship_id | fleet_id
unload_command ::= UNLOAD ship_id OF materiel [QUANTITY number]
update_command ::= UPDATE island_id | ship_id | fleet_id [USE number]

#-------------------------------------------------------------

range_of_sectors ::= island_posn SPAN co-ord
target_descriptor ::= ship_id | island_posn
island_posn ::= island_id SECTOR co-ord
ship_id ::= SHIP number
fleet_id ::= FLEET number
island_id ::= ISLAND number
player_id ::= PLAYER number
sector_type ::= DOCK | RURAL | URBAN | CAPITOL | FORT | PLANE |
AIRPORT | EXPLOSIVE | ARTILLERY | CANAL | MINE | RADAR | TRADECENTRE
ship_type ::= BARGE | PT | FERRY | SWEEPER | DESTROYER | FREIGHTER |
LINER | SUB | TANKER | TRANSPORT | CRUISER | BATTLESHIP | CARRIER
materiel ::= ORE | EXPLOSIVES | ARTILLERY | population
population ::= CIVILIANS | MILITARY
co-ord ::= number , number
number ::= [0-9]+
string ::= (standard defn of string goes here)
```

7.2 ASSAULT

7.2.1 Purpose:

Assault allows one to amphibiously arrive at an island. In general, for a ship to unload materiel of all types, the ships must be docked. “Docking” is not possible unless the sector is owned by the same player who owns the ship, or if the sector is a designated trade centre. Ergo, the assault.

7.2.2 Examples:

```
assault island 6 from ship 210
```

7.2.3 Effects:

The assault may well be repulsed. Defenders enjoy a considerable advantage against amphibious invaders. If successful,

The ownership of the sector assaulted transfers to the owner of the ship.

The sector's type is designated as a "dock" at the time of transfer.

7.3 ASSIGN

7.3.1 Purpose:

Used to associate a ship with a fleet. This is a convenience command since once a player has several ships it is general practice to keep them together for mutual defence. Fleet zero is the non-fleet; i.e., all ships are originally a part of fleet zero. To disassociate a ship from a fleet, the proper command is to assign it to fleet zero. Assign does not consume any type of game time.

7.3.2 Examples:

```
assign ship 189 to fleet 3;
assign ship 1700 to fleet 0;
```

7.3.3 Effects:

The database is searched for the appropriate ship record. Once found, the Fleet field in the record is changed to the requested value, and the database is updated. Fleet numbers are arbitrary.

7.4 ATTACK

7.4.1 Purpose:

Used to conduct ground attacks between neighbouring sectors. Attack uses two (2) units of movement time if the attack is carried out against a flat land sector, and six (6) units of movement time against a mountain sector or a fortification.

The player issuing the command may optionally specify a number of ground combat troops to use. If this amount is not given, all the military in the sector are used.

7.4.2 Examples:

```
attack island 6 8,9 from island 6 9,9 quantity 400
```

7.4.3 Effects:

The game engine verifies that the player's capitol has enough movement time to execute the attack. The capitol's time is decremented appropriately. Also verified is that the sectors involved are actually adjacent to each other.

The attack is resolved based on the efficiency of each sector, and the numbers of attackers and defenders in each sector. All of the military in the sector under attack are presumed to be employed in the defence of the sector.

In the case of attack from flat land to flat land, each sector at 100% efficiency, an attack with a 6:4 numerical advantage will result in one attacker taking over the sector under attack.

The efficiency of the sector thus captured is set to 1% if the ownership is transferred due to successful attack. The designation is unchanged. Materiel is unaffected and undamaged by ground assault, and all materiel becomes the property of the attacker.

It is worth noting that the game does not prevent attacks on one's own troops.

The following fields in the record for the attacked sector are updated: Efficiency, MilitaryPop, and possibly Owner. The sector from which the attack is launched will have its MilitaryPop changed.

7.5 BOMB

7.5.1 Purpose:

Used to drop bombs on a sector or a ship from fighters at an airport or on a ship. The player issuing the command specifies both the target and the source, and optionally a number of aircraft to use. If the number of aircraft to use is not specified, the game attempts to use the maximum possible number. *The BOMB command does not use game time*.

7.5.2 Examples:

```
bomb ship 72 from island 8 sector 18,3 quantity 10
bomb ship 72 from ship 16 quantity 50
bomb island 12 sector 32,10 from ship 16
bomb island 12 sector 32,10 from island 10 sector 2,22
```

7.5.3 Effects:

Bombing is the most complex and far reaching command within the game. As a result, the description below separates the effects into parts.

7.5.3.1 Source

Each fighter aircraft must have a pilot (military population unit) and a bomb (an explosive). If the source is an airport sector, the sector must contain military population and explosives for the command to succeed. If the source is a ship (aircraft carrier), then the ship must have military units and explosives aboard.

In either case, the number of fighters requested could be lowered because of lack of military units or explosives. For example: the player requests the use of 20, from an airport that has a military population of 500 with 10 available explosives. In this case, only ten aircraft can be used.

If the attacking aircraft are shot down, the pilots and the bombs are lost along with the planes.

7.5.3.2 Targeting a ship

If the target is a ship, the game engine determines if the target ship is under protection of a nearby aircraft carrier. If it is so protected, then the aircraft carrier scrambles its fighters which enjoy a 6:4 advantage in defence.

If any attacking aircraft remain after this air-to-air combat, then the game engine determines if there are any nearby flak-capable ships. This surface-to-air combat is then resolved.

If any attacking aircraft remain after taking flak, then the bombs are released. The bombs are reasonably accurate, although the game does generate a small amount of targeting precision error. The bombs that hit the ship inflict damage that can range from decreased efficiency of the ship to sinking.

The aircraft return to their point of origin without the bombs.

7.5.3.3 Targeting land

The game engine looks to see if the sector under attack contains “flak;” i.e., military, explosives and artillery. If so, the ground based guns attempt to shoot down the incoming aircraft, putting up one shell per gun.

If any attacking planes remain, the bombs are dropped with a 90% chance of striking the intended sector. Damage is resolved regardless of where the bomb falls.

7.6 DEPTHCHARGE

7.6.1 Purpose:

The depth-charge is the only weapon that can be used against a submerged submarine. Only ships that are depth-charge capable may be used.

7.6.2 Examples:

```
depthcharge ship 233 from ship 16
```

7.6.3 Effects:

Depth-charges use 5 explosive units from the attacking ship. Results are based on distance from the attacking ship to the submarine. The submarine does not return fire.

7.7 DESIGNATE

7.7.1 Purpose:

Used to change the type of a sector. A sector is implicitly designated as “rural” as soon as it is first occupied. Rural land has no use in the game, thus players change the sector’s type to reflect its intended use.

7.7.2 Examples:

```
designate island 7 sector 6,33 to FORT
```

7.7.3 Effects:

Designating a sector always sets its efficiency to 1%, although it has no effect on the materiel.

7.8 DOCK

7.8.1 Purpose:

For ships to load or unload materiel and crew, they must be “docked.” For unloading, docking may take place at any flat land sector, regardless of the sector’s type. To transfer materiel to a ship, it must be at a sector that is designated to be the dock type.

7.8.2 Examples:

```
dock ship 7
```

7.8.3 Effects:

The DOCK command uses one unit of ship movement time even though no actual movement of the ship takes place. The ship’s record is updated to set Docked=true, and the amount of available movement time is decreased by one.

7.9 EMERGE

7.9.1 Purpose:

This command causes the effected submarine to surface.

7.9.2 Examples:

```
emerge ship 1
```

7.9.3 Effects:

The only effect on the database is to set the submarine to be on the surface. As a side-effect, it will now begin to show up in radar reports.

7.10 FLY

7.10.1 Purpose:

7.10.2 Examples:

```

```

7.10.3 Effects:

7.11 LAYMINE

7.11.1 Purpose:

7.11.2 Examples:

7.11.3 Effects:

7.12 LOAD

7.12.1 Purpose:

7.12.2 Examples:

7.12.3 Effects:

7.13 MOVE

7.13.1 Purpose:

7.13.2 Examples:

7.13.3 Effects:

7.14 PING

7.14.1 Purpose:

7.14.2 Examples:

7.14.3 Effects:

7.15 REFURBIUSH

7.15.1 Purpose:

7.15.2 Examples:

7.15.3 Effects:

7.16 SAIL

7.16.1 Purpose:

7.16.2 Examples:

7.16.3 Effects:

7.17 SCUTTLE

7.17.1 Purpose:

7.17.2 Examples:

7.17.3 Effects:

7.18 SETCOURSE

7.18.1 Purpose:

7.18.2 Examples:

7.18.3 Effects:

7.19 SETNAME

7.19.1 Purpose:

Used by the player who has naming rights to name a ship or an island.

7.19.2 Examples:

```
setname 'borneo' island 6
setname 'john f kennedy' ship 700
```

7.19.3 Effects:

The game engine verifies that the player issuing the command does indeed own the ship, or have naming rights to the island. The name is changed in the database, but other players will only become aware of the name change the next time the island is accessed or the ship is pinged on radar.

7.20 SETSPEED

7.20.1 Purpose:

Used to change the speed of a ship or a fleet that is on course.

7.20.2 Examples:

```
setspeed 22 ship 700
setspeed 15 fleet 17
```

7.20.3 Effects:

The game engine verifies that the ship or fleet effected is on course. The game attempts to set the speed to the desired speed. If the desired speed is not available (for example: the requested speed exceeds the maximum speed for that type of ship, or for the current capabilities of the ship because of damage), the speed is set to the maximum allowable.

If the requested speed is zero, the ship or ships stop, and the course information is lost.

7.21 SHELL

7.21.1 Purpose:

7.21.2 Examples:

```

```

7.21.3 Effects:

7.22 SUBMERGE

7.22.1 Purpose:

7.22.2 Examples:

7.22.3 Effects:

7.23 SWEEP

7.23.1 Purpose:

7.23.2 Examples:

7.23.3 Effects:

7.24 TELL

7.24.1 Purpose:

7.24.2 Examples:

7.24.3 Effects:

7.25 TEND

7.25.1 Purpose:

Used to transfer materiel or personnel from one ship to another. The two ships involved must be in the same sector, and they must both be at sea or both be docked. It is not possible to tend a docked ship from an undocked ship, and *vice versa*.

The TEND command does not use ship time from either the tender or the ship being tended. If no quantity is given, all of one ship's materiel is moved.

A variant of the TEND command allows a player to tend all of the ships in the fleet of which the tender is a part. In this case, the quantity given is construed to be a total amount which is divided among the recipients as evenly as possible, or until the maximum is allowed for the recipient is reached.

7.25.2 Examples:

```
tend ship 78 from ship 77 with ore quantity 25
tend fleet from ship 77 with ore quantity 200
```

7.25.3 Effects:

7.26 TORPEDO

7.26.1 Purpose:

Used from a submarine to launch a torpedo. Torpedos may be fired at any ship within range, and they may only be fired one at a time.

7.26.2 Examples:

```
Torpedo ship 890 from ship 2
```

7.26.3 Effects:

If there is a ship friendly to the ship under attack that is capable of releasing depth charges, that ship is construed to fire up on the attacking submarine before the torpedo is released. If the submarine survives the attack

7.27 TRANSFER

7.27.1 Purpose:

7.27.2 Examples:

```

```

7.27.3 Effects:

7.28 UNDOCK

7.28.1 Purpose:

7.28.2 Examples:

```

```

7.28.3 Effects:

7.29 UPDATE

7.29.1 Purpose:

7.29.2 Examples:

7.29.3 Effects:

8 Appendix: Working Database Schema

```
/*
** This script is used to create an SQL database for playing
** the game of EMPIRE. (c) Bright-Crayon, LLC, 2002.
**
** Note that if you are playing more than one game on a server,
** then you will need to change the name of the database to be
** different from any databases already in existence.
*/

CREATE DATABASE IF NOT EXISTS empire;
USE empire;

DROP TABLE IF EXISTS globals;
DROP TABLE IF EXISTS news;
DROP TABLE IF EXISTS islands;
DROP TABLE IF EXISTS players;
DROP TABLE IF EXISTS sectors;
DROP TABLE IF EXISTS ships;
DROP TABLE IF EXISTS shipstaticdata;
DROP TABLE IF EXISTS sectorstaticdata;

/*
** There is only one row in the "globals" table. It contains
** the invariant information for "this game."
*/

CREATE TABLE globals (
 State INT,                  # bitmask actually ...
 GenesisSeed INT,            # random seed used to build this game
 GroundRadarRange INT,       # how far ground radar sees
 MinIslandSep INT,           # how far apart islands must be
 NewPlayerClockTicks INT,    # how much time does a player start with
 NewPlayerCivilians INT,     # how many civilians at the beginning
 Version DOUBLE,             # version of this game/database
 UpdateFactor DOUBLE,        # exponent for pop growth
 ShipDayFactor DOUBLE,       # speed scaling factor for ships
 ClockTicksPerDay INT,       # how fast time passes in this game vs wallclock
 MaxTimeAccrual INT,         # how much *time* can a Capitol sector hold?
 ShellCost INT,              # cost of mfg a shell ...
 GunCost INT,                # a gun ...
 PlaneCost INT,              # a plane ...
 FlightRadius INT,           # how far the planes can travel
 MaxGunsPerSector INT,       # artillery pieces per land sector
 MaxShellsPerSector INT,     # shells per land sector
 MaxPlayers INT,             # maximum players in this game
 NumIslands INT,             # number of islands, usually greater than num players
 MaxIslandDimension INT,     # how large the biggest island is allowed to be
 WorldCircumference INT,     # how large the world is
 GameStart DATETIME,         # when this game was first switched on
 GameEnd DATETIME,           # when the game will end
 GameLastStarted DATETIME,   # last time the game was started up
 GameLastShutdown DATETIME,  # last time the game was halted/suspended
 GameDailyStart TIME,        # time of day when players can first connect
 GameDailyShutdown TIME      # time of day when players will be logged out
) MAX_ROWS=1;

/*
The islands table contains a list of all the islands
in the world, and where they are.
*/

CREATE TABLE islands (
 Number INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 Name CHAR (40),
 Owner INT,
 BottomEdge INT,
 LeftEdge INT,
 XDimension INT,
 YDimension INT
) MAX_ROWS = 300; # just for planning purposes

/*
This is not a complete log of the game. Rather it is for the
purpose of analysis so that the newspaper can be updated
with word of recent conflicts.
*/
```

```
CREATE TABLE news (
 Number INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 Time DATETIME,
 PlayerInit INT,
 PlayerOther INT,
 Island INT,
 Ship INT,
 XCoor INT,
 YCoor INT,
 Description CHAR (200)
);

/*
** Everything in the game is owned by some player. This table
** contains information about logon, name, email, etc.
*/

CREATE TABLE players (
 Number INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 Name CHAR (30),
 LoggedOn INT,
 HomeIsland INT,       # What island this player started on
 MappingXOffset INT,   # For the game engine use in drawing maps.
 MappingYOffset INT,   # as above
 Email CHAR (100),
 Password CHAR (200), # Message digest of the password, actually
 IMHandle CHAR (30),   # For interplayer chatter
 Birth DATETIME,
 Death DATETIME,
 LastLogon DATETIME,
 LastActivity DATETIME,
 LastLogoff DATETIME
) MAX_ROWS = 60;       # just for planning purposes

/*
** This is a table with all the land sectors on all the islands. It
** might make sense to have one table per island, but that actually
** creates more of a problem since the configuration of the database
** would not really be known by the statically bound code. For example,
** one would have to generate the names of the tables for each island
** programmatically. One would also have to be concerned with multiple
** table locks of dynamically determined table names being performed
** in the correct order to avoid deadly embraces.

** The sectors table has NumIslands * MaxIslandDimension^2 rows, and
** the offset [first element] is set to the appropriate value that
** corresponds to: (islands.Number-1) * MaxIslandDimension^2 + ... etc.

** The Type field is set to CHAR simply to assist with human readability.
** There are only about 20 types of sectors, and it is much easier to
** debug the database without having to remind oneself of what type 11
** sectors represent.
*/

CREATE TABLE sectors (
 Offset INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 Island INT,             # What island is this sector on?
 IslandXCoor INT,        # 0 < x < IslandXDim
 IslandYCoor INT,        # 0 < y < IslandYDim
 XCoor INT,              # global x coordinate
 YCoor INT,              # global y coordinate
 Owner INT,              # Player Zero owns unowned land
 CivilianPop INT,
 MilitaryPop INT,
 Elevation INT,          # {-1, 0, 1}
 Guns INT,
 Shells INT,
 Planes INT,
 Ore INT,
 SampleRate DOUBLE,      # 0.0 .. 1.0 for ore production
 Efficiency DOUBLE,      # 0.0 .. 1.0
 Type CHAR(1),           # CHAR for readability only
 MovementTime INT,       # only if this is the capitol sector
 UpdateTime INT          # only if this is the capitol sector
);

/*
** Once again, we see all the ships in one file. This is frequently
** a very large file, and once ships are sunk their numbers are not
** reused.
```

```
*/

CREATE TABLE ships (
 Number INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
 Afloat INT,           # 0 or 1, sunk or afloat
 Rust DOUBLE,          # 0.0 to 1.0, once out of fuel, ships rust.
 Name CHAR (30),       # Most ships will be unnamed ...
 Owner INT,
 Type CHAR(4),         # corresponding to one of the types in the next table
 Fleet INT,            # Fleet numbers are arbitrary, and just for grouping
 Efficiency DOUBLE,    # 0.0 to 1.0, just as with sectors
 Fuel INT,
 Civilians INT,
 Military INT,
 Guns INT,
 Shells INT,
 HeadingXDir DOUBLE,   # -1.0 .. 1.0
 HeadingYDir DOUBLE,   # -1.0 .. 1.0
 Speed DOUBLE,
 XPosition DOUBLE,     # world coordinate
 YPosition DOUBLE,     # [ditto]
 DestinationXCoor INT, # applicable only if the ship has a course set.
 DestinationYCoor INT, # [ditto]
 MovementTime INT,     # 0 .. max speed for this ship type
 LastUpdated DATETIME, # for use by the game engine
 Docked INT,           # 0 or 1, [at sea] or [docked]
 DockedAtIsland INT,   # which island, otherwise 0
 DockedAtXCoor  INT,   # the XCoor of the island
 DockedAtYCoor  INT   # the YCoor of the island
);

CREATE TABLE shipstaticdata (
 Type CHAR (4) NOT NULL PRIMARY KEY,
 Name CHAR (30), # Note: this is the name of the /type/ of ship
 Cost INT,
 MaxSpeed INT,
 MaxFuel INT,
 IdleFuelPerDay DOUBLE,
 MaxCivilians INT,
 MaxMilitary INT,
 MaxGuns INT,
 GunRange INT,
 GunFactor DOUBLE,
 MaxShells INT,
 Flak INT,
 SonarRange INT,
 RadarRange INT,
 MaxPlanes INT,
 Capabilities SET ("SeaGoing", "DepthCharging",
                   "MineLaying", "MineSweeping",
                   "Submersible", "Shelling")
) MAX_ROWS = 15;

CREATE TABLE sectorstaticdata (
 SectorType CHAR NOT NULL PRIMARY KEY,
 Description CHAR(60),
 MaxMilitaryPop INT,
 MaxCivilianPop INT
) MAX_ROWS=15;

/*
** Let us load the static data into the globals, staticshipdata, and
** staticsectordata tables.
**
** For the globals table, these are nice default values that the
** game administrator might want to tweak on a per-game basis.
*/


INSERT INTO globals ( State, GenesisSeed, GroundRadarRange,
 MinIslandSep, NewPlayerClockTicks, Version, UpdateFactor,
 ShipDayFactor, ClockTicksPerDay,
 ShellCost, GunCost, PlaneCost, FlightRadius,
 MaxGunsPerSector, MaxShellsPerSector, MaxPlayers,
 NumIslands, MaxIslandDimension, WorldCircumference)
VALUES (0, 0, 50, 5, 100, 1.0, 1.02, 1.0, 48, 1, 10,
        50, 50, 9, 1000, 30, 60, 40, 600);

/*
** It is worth noting that the text literals here can be localised.
```

```
** They are not used for look up purposes. The character designations
** for the otherwise numeric ids for the sector types were simply
** chosen to aid in human readability of database dumps.
*/

INSERT INTO sectorstaticdata
VALUES ('A', "Artillery Factory", 1000, 1000),
       ('C', "Capitol", 10000, 10000),
       ('D', "Dock (Ship Factory)", 1000, 1000),
       ('E', "Explosives Factory", 1000, 1000),
       ('F', "Fort", 10000, 1000),
       ('M', "Ore Mine", 1000, 1000),
       ('P', "Plane Factory", 1000, 1000),
       ('R', "Rural (undesignated) Land", 1000, 1000),
       ('U', "Urban Centre", 1000, 10000),
       ('W', "Waterway/Canal", 1000, 1000),
       ('^', "Mountain", 1000, 1000),
       ('(', "Radar Installation", 1000, 1000),
       ('+', "Airport", 1000, 1000),
       ('_', "Uninhabited Flat Land", 0, 0),
       ('.', "Submerged Land", 0, 0);

/*
**
*/

INSERT INTO shipstaticdata
VALUES ("BRG", "Barge", 50, 15, 1000, 0.5, 20, 50, 1000, 0,
                        0, 10000, 0, 0, 20, 0, 0),
       ("PT", "PT Boat", 50, 40, 0.5, 10, 0, 15, 0, 0,
                          0, 10, 1, 10, 30, 0,
                       "SeaGoing" AND "MineLaying" AND "DepthCharging"),
       ("FRY", "Ferry", 75, 24, 50, 0.75, 1000, 1000, 0, 0,
                        0, 0, 0, 0, 30, 0, 0),
       ("SWP", "Mine Sweeper", 150, 22, 1.5, 25, 0, 40, 0, 0,
                         0, 300, 2, 5, 15, 0,
                      "SeaGoing" AND "MineLaying" AND "MineSweeping"),
       ("DEST", "Destroyer", 200, 30, 25, 2.0, 0, 250, 2, 5,
                             20, 200, 5, 20, 30, 0,
                           "SeaGoing" AND "DepthCharging" AND "MineLaying" AND "Shelling"),
       ("FRTR", "Freighter", 200, 22, 100, 2.0, 0, 500, 1000, 0,
                             0, 0, 0, 0, 30, 0,
                           "SeaGoing"),
       ("LIN", "Liner", 200, 15, 100, 2.0, 5000, 100, 0, 0,
                        0, 0, 0, 0, 30, 0,
                      "SeaGoing"),
       ("SUB", "Submarine", 200, 25, 75, 2.0, 0, 50, 2, 5,
                            10, 0, 0, 0, 30, 0,
                         "SeaGoing" AND "Submersible" AND "Shelling"),
       ("TNKR", "Tanker", 200, 22, 1000, 2.0, 0, 20, 0, 0,
                          0, 0, 0, 0, 25, 0,
                       "SeaGoing"),
       ("TRNS", "Transport", 200, 22, 60, 2.0, 0, 5000, 0, 0,
                             0, 0, 0, 0, 30, 0,
                           "SeaGoing"),
       ("CRU", "Cruiser", 300, 25, 50, 2.0, 0, 500, 3, 5,
                          30, 0, 30, 0, 30, 0,
                        "SeaGoing" AND "MineLaying" AND "Shelling"),
       ("BTL", "Battleship", 400, 22, 60, 2.0, 0, 800, 4, 7,
                             40, 0, 12, 0, 30, 0,
                           "SeaGoing" AND "Shelling"),
       ("CAR", "Carrier", 500, 25, 200, 2.0, 0, 2000, 1, 5,
                          20, 0, 10, 25, 30, 500,
                        "SeaGoing" AND "Shelling");

/*
** Now then, we need to put the zero-th record in each of the main
** data tables. Player zero owns everything that is unowned. Island
** zero does not exist. Ship zero is owned by player zero, and it is
** docked at island zero. Etc.
*/

INSERT INTO ships
VALUES (NULL, 0, 1.0, "Ship Zero", 0, "UNKN", 0, 0.0, 0, 0, 0, 0, 0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0, 0, 0, NULL, 0, 0, 0, 0);

INSERT INTO islands
VALUES (NULL, "Island Zero", 0, 0, 0, 0, 0);

INSERT INTO players
```

```
VALUES (NULL,
       "Player Zero",
       0,
       "root@localhost",
       NULL, NULL, NULL, NULL,
       NULL, NULL, NULL);
```

This is the end of the document

Abs Revision	Date of last Revision	Number of Pages
33	2004/02/12 20:14	52