PART II

**SOFTWARE REQUIREMENTS SPECIFICATION**

****

**California State University Northridge**

**Computer Science Department**

**Scorch3d Earth**

By: Matthew Hoggan

Hangul Lim

David Schaffer

Davit Stepanyan

Author: Matthew Hoggan

Creation Date: 9/25/2010

Last Revised: 5/11/2011

Version: 1772

Contents

[Section 1: Preface 3](#_Toc290330530)

[1.1 *Revision Notes*: 3](#_Toc290330531)

[1.2 *Purpose of SRS Report*: 3](#_Toc290330532)

[Section 2: Introduction 4](#_Toc290330533)

[2.1 *Purpose for Re-implementing Scorched Earth* 4](#_Toc290330534)

[Section 3: Glossary 5](#_Toc290330535)

[Section 4: User Requirements 8](#_Toc290330536)

[4.1 *User Definition* 8](#_Toc290330537)

[4.2 *Services Offered By the Game “Scorch3d Earth”* 8](#_Toc290330538)

[4.3 *Non-Functional Requirements* 8](#_Toc290330539)

[4.1.a *Product Requirements* 8](#_Toc290330540)

[4.1.b *Organizational Requirements* 9](#_Toc290330541)

[Section 5: Functional Requirements 10](#_Toc290330542)

[5.1 *List of Functional Requirements* 10](#_Toc290330543)

[Section 6: System Architecture 14](#_Toc290330544)

[6.1 *Basic Diagrams for System Architecture* 14](#_Toc290330545)

[Section 7: System Evolution 20](#_Toc290330546)

[7.1 *Concerns for Future Expansion of Scorch3d Earth* 20](#_Toc290330547)

[References 21](#_Toc290330548)

# Section 1: Preface

1.1 *Revision Notes*:

As of Revision 1760, this document now reflects the current changes to the game, as well as, presents information not available in previous versions of this document. We have also removed the gas can from the game due to time constraints. Additionally, we have also slightly modified the controls. Finally, it should be noted that we have addressed one of our concerns (found at the end of this document). That is, we have moved off the fixed functional pipeline, and into the use of shaders to render and modify the appearance of our objects.

As of Revision 800 of this SRS report, is a drastic change from the initial document entitled, “Software Requirements Specification – 3D-Scorched Earth” version one. The reason for such change consists of inadequacies in the initial document, and the need for a modified, version tracking system. Thus, the version numbering system used for this document will be based on the Content Management System (CMS) version number. Furthermore, this report will provide more complete information about both system and user requirements.

1.2 *Purpose of SRS Report*:

This document will outline the services offered to the players of our game, entitled “Scorch**3**d Earth”. For the purposes of this document, the game in its entirety will be labeled as “the system”. Thus, this document will explain what “the system” is expected to provide to the end user, as well as the constraints under which it will operate. Additionally, this document will give a detailed description of the software system’s functions, services, and overall general structure and implementation.

Section 2: Introduction

2.1 *Purpose for Re-implementing Scorched Earth*

According to Wikipedia, the original “Scorched Earth” is or was a popular shareware artillery video game in the early 1990s. The game was originally developed for DOS and consisted of a relatively primitive 2D environment (Diagram 1 below). The game is a multiplayer game, in which each player takes a turn on the same computer, attempting to blow up their opponent’s tank. Furthermore, the difficulty of the game is increased by manipulating the flight path of the projectile through the simulated physics of wind drag and propulsion, as well as gravity on a tank’s projectiles.

The purpose of this project is to remodel the original 2d Scorched Earth into a 3D version. With the advent of graphics libraries, such as OpenGL and Microsoft’s Direct3d, games such as Scorched Earth can gain new potential. This project will attempt to use the same general control system, random terrain generation, economic, physics, weapons, and environmental systems that were present in the original game. Thus, this project will attempt to reincarnate the original 2D game back to life in a new 3D body.

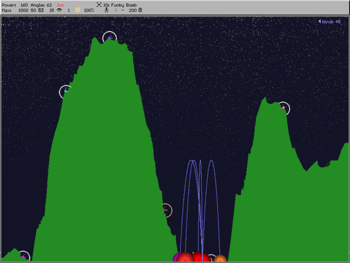


Diagram 1: An early version of the original 2D Scorched Earth

# Section 3: Glossary

**AC3D** – This is a 3D design program which has been available since 1994. The software is used by game and model designers for modeling 3D graphics for games and simulations.

**Game Loop** - The central component of any game, from a programming standpoint, is the game loop. The game loop allows the game to run smoothly regardless of a user's input or lack thereof. Games must continue to operate regardless of a user's input. The game loop allows this. The game loop may be refined and modified as game development progresses, but most games are based on this basic idea. Game loops differ depending on the platform they are developed for. For example, games written for DOS and most consoles can dominate and exploit available processing resources without restraint.

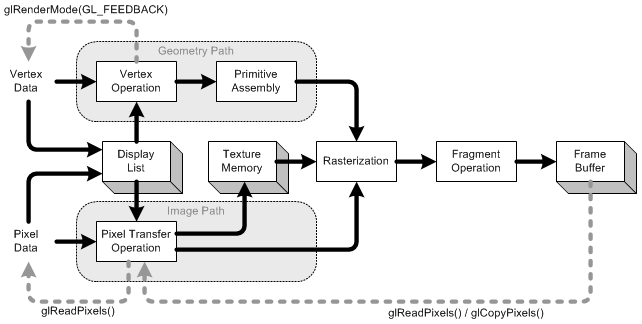
**Graphics Language Utility Toolkit (GLUT)** – GLUT is a windowing system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms. GLUT is designed for constructing small to medium sized OpenGL programs.

**Normal Vector** – This vector is often referred to as "normal," to a surface and is defined as a vector perpendicular to said surface. Often, the normal unit vector is desired as the "unit normal." The terms "normal vector" and "normalized vector" should not be confused, especially since unit normal vectors might be called "normalized normal vectors" without redundancy. The normal vector is commonly denoted as N or n.

**OpenGL** - OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used/supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

**Polygon Mesh** - A polygon mesh or unstructured grid is a collection of vertices, edges and faces that defines the shape of a polyhedral object in 3D computer graphics and solid modeling. The faces usually consist of triangles, quadrilaterals or other simple convex polygons, since this simplifies rendering, but may also be composed of more general concave polygons, or polygons with holes. The study of polygon meshes is a large sub-field of computer graphics and geometric modeling. Different representations of polygon meshes are used for different applications and goals.

**Rendering Pipeline (Fixed Functional)** – OpenGL’s Fixed Functional Pipeline has a series of processing stages in order. Two forms of graphical information, vertex-based data and pixel-based data, are processed through the pipeline, combined together then written into the frame buffer.



http://www.songho.ca/opengl/gl\_pipeline.html

Display List - Display list is a group of OpenGL commands that have been stored (compiled) for later execution. All data, geometry (vertex) and pixel data, can be stored in a display list. It may improve performance since commands and data are cached in a display list.

Vertex Operation - Each vertex and normal coordinates are transformed by GL\_MODELVIEW matrix. Also, if lighting is enabled, the lighting calculation per vertex is performed using the transformed vertex and normal data. This lighting calculation updates new color of the vertex. (See more details in Transformation)

Primitive Assembly - After vertex operation, the primitives (point, line, and polygon) are transformed once again by projection matrix then clipped by viewing volume clipping planes; from eye coordinates to clip coordinates. After that, perspective division by w occurs and viewport transform is applied in order to map 3D scene to window space coordinates. Last thing to do in Primitive Assembly is culling test if culling is enabled.

Pixel Transfer Operation - After the pixels from client's memory are unpacked (read) the data are transformed by scaling, bias, mapping and clamping. These operations are called Pixel Transfer Operation. The transferred data are either stored in texture memory or rasterized directly to fragments.

Texture Memory - Texture images are loaded into texture memory to be applied onto geometric objects.

Rasterization - Rasterization is the conversion of both geometric and pixel data into fragments. Fragments are defined as, a rectangular array containing color, depth, line width, point size and antialiasing calculations.

Fragment Operation – This is the final process in which fragments are converted to pixels and mapped into the frame buffer. The first process in this stage is texel generation; A texture element is generated from texture memory and it is applied to the each fragment. Then fog calculations are applied. After that, there are several fragment tests which follow, namely Scissor Test ⇒ Alpha Test ⇒ Stencil Test ⇒ Depth Test. Finally, blending, dithering, logical operation and masking by bitmask are performed and actual pixel data are stored in frame buffer.

**Scorched Earth** - is a popular shareware artillery video game, which is a subgenre of strategy games. The game was developed in the DOS era, originally written by Wendell Hicken (using Borland C++ and Turbo Assembler), in which tanks do turn-based battle in two-dimensional terrain, with each player adjusting the angle and power of their tank’s turret before each shot.

**Simple Direct Media Layer** - Simple DirectMedia Layer is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer.

# Section 4: User Requirements

## 4.1 *User Definition*

“Scorch3d Earth” will not only attract those players who played the original “Scorched Earth”, but also will be interesting enough to attract new comers to the game. According to statistics, players of the game will be young males between the ages of twelve to thirty. We are planning on reaching out to future players by distributing the game freely over the internet.

## 4.2 *Services Offered By the Game “Scorch3d Earth”*

“Scorch3d Earth’s” new 3D graphics will offer a more visually stimulating environment for game play.

“Scorch3d Earth” will provide an easy-to-use menu and control system to facilitate game play.

“Scorch3d Earth” will replicate the random terrain generation, economic, physics, weapons, and environmental systems that were present in the original game, in a new dimension.

4.3 *Non-Functional Requirements*

### 4.1.a *Product Requirements*

Usability Requirements: Up to this date, the only usability concern is that both players of the original 2D version of the game, as well as new comers, can quickly familiarize themselves with the controls and game play within a 3D environment. Additionally, the goal is that users will be able to adjust to a new and graphically superior menu system.

Efficiency Requirements: Up to this date, we are developing the game to run “smoothly” on systems that have the following system specifications and higher.

|  |  |
| --- | --- |
| System Component | Requirement |
| CPU | 1.5 GHz or Faster |
| CPU Type | Intel Pentium 4 or AMD Athlon |
| Memory | 256 MB Internal RAM |
| OS | Any |
| Graphics Card | >= NVIDIA GeForce FX 5200  >= ATI Radeon 9600 |
| Graphics Card Memory | 128MB or greater |
| Hard Drive Space | 256 MB of free disk space |
| Mouse | Any |
| Keyboard | Any |

Dependability Requirements: Presently the only dependability requirement is that the game does not crash under any circumstance produced by the game itself. This requirement does not have to hold true if the system itself crashes for other unknown reasons.

### 4.1.b *Organizational Requirements*

Development Requirements – The following table depicts the development requirements of the programming team for “Scorch3d Earth”. All tools will be provided by Matthew Hoggan from his servers at home (apart from the specified IDE).

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Reason |
| Subversion | A Content Management System (CMS) | Due to the complexity of the code and the normal structuring of a module when programming in OpenGL, the team will need group access to a consistent code base. |
| Microsoft Visual Studio 2008 | A Integrated Development Environment produced by Microsoft© | In order to maintain consistent project settings, and a uniform build environment, the team will be using Microsoft Visual Studio to build “Scorch3ed Earth” |
| OpenGL | A Set of Free Cross Platform Graphics Libraries | OpenGL will be at the heart of our program. Without OpenGL this game would not be possible. We chose OpenGL for our main tool because it is relatively comfortable for our team to use and provides an easy access of use at the computer lab 1618. |
| GLUT | Graphics Language Utility Toolkit (see glossary) | GLUT will be used to handle windowing, and user input via the keyboard or mouse. Furthermore, GLUT will provide our application with a timer function that will coordinate the game loop and the graphics pipeline. |
| SDL | Simple Direct Media Layer | Simple DirectMedia Layer is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer. |
| AC3D | A 3D Design Program | AC3D will be used to generate models. Without AC3D we would have to render all models by hand (vertex by vertex), which would be extremely time consuming, if not impossible. |
| Doxygen | Doxygen is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extent D. | It can help you in three ways:  It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in ) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. You can configure doxygen to extract the code structure from undocumented source files. Doxygen is developed under Linux and Mac OS X, but is set-up to be highly portable. As a result, it runs on most other Unix flavors as well. Furthermore, executables for Windows are available. |
| Bugzilla | Bugzilla is server software designed to help you manage software development. | Bugzilla is a "Defect Tracking System" or "Bug-Tracking System". Defect Tracking Systems allow individual or groups of developers to keep track of outstanding bugs in their product effectively. Most commercial defect-tracking software vendors charge enormous licensing fees. Despite being "free", Bugzilla has many features its expensive counterparts lack. Consequently, Bugzilla has quickly become a favorite of thousands of organizations across the globe.` |

# Section 5: Functional Requirements

## 5.1 *List of Functional Requirements*

5.1.a **Function**: *Mimic Initial Controls with Slight Modifications to Adjust to the “New”*

*Dimensionality*

5.1.a.i

If the user presses the left arrow key , the turret on the tank will yaw left.

5.1.a.ii

If the user presses the right arrow key , the turret on the tank will yaw right.

5.1. a.iii

If the user presses the up arrow key , the turret on the tank will pitch the barrel up to a maximum of .

5.1.a.iv

If the user presses the down arrow key , the turret on the tank will pitch the barrel down to a minimum of .

5.1.a.v If the user presses the less than key the velocity of the projectile will decrease before being fired.

5.1.a.vi If the user presses the greater than key the velocity of the projectile will increase before being fired.

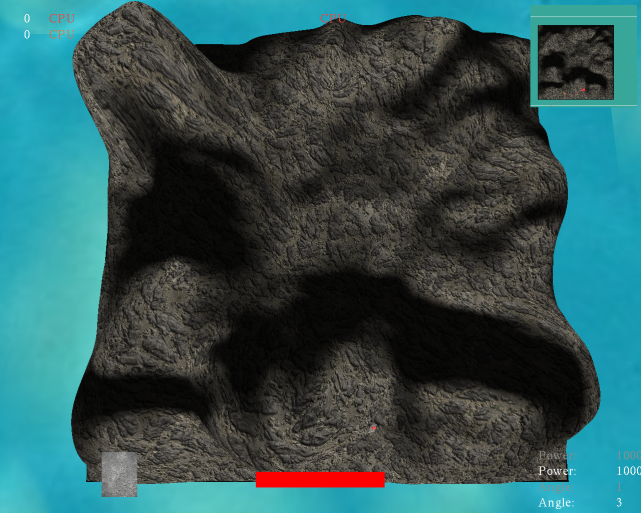
5.1.a.vii If the user presses the “i” button and then clicks the or or or buttons the select cursor will move in that direction. If the player type is human.

5.1.a.viii If the user presses the “i” button on the keyboard the inventory menu will open only if in the “Game Play State” is active (discussed below). If the menu is already open pressing the “i” button will close the inventory.

5.1.a.iv If the user presses the “space bar”, the player whose turn it is will use the item selected in their inventory. This item could be a projectile, or any other item provided by “the system”

5.1.b **Function**: *Generate Terrain Randomly*

5.1.b.i At the start of each game, while in the “Main Menu State” players of the game will have the option to determine the type of terrain they desire to play on. Subsequently, a random terrain generation algorithm will be deployed based on the parameters specified by the user. Examples of randomly generated terrain by David Schaffer’s random terrain generating algorithm can be seen below for both flat and mountain terrains.

5.1.c **Function**: *Provide a Turn Based Play Style*

5.1.c.i During the “Game Play State” multiple players, up to 10, will be able to take turns attempting to blow up one of their opponents tanks (or their own). During a players turn they will be able to use a subset of the controls listed above, and they will be able to select from their inventory, the item they want to use for that round. The whole purpose of the game is to be the last tank or team of tanks left standing.

5.1.d **Function**: *Support an Economic System to Purchase New Items After Each Subsequent*

*Round*

5.2.d.i From a functional perspective, the economic system of the game will be governed by the ranking of tanks at the beginning of a round. If the game just started, each tank will be given rank zero, and therefore, each tank will have equal access to the same items, such as missiles, rockets, gas cans, and others used to destroy other enemy tanks, which they can place into their inventory while in the “Main Menu State” of the game. If players decide to play multiple rounds, the higher rank tanks will receive bonus cash to spend on the subsequent round, which will give them an advantage in that round.

5.1.e **Function**: *Provide an Adjustable Physics Engine for Each Round*

5.2.e.i The purpose of this is to mimic real life occurrences such as wind and gravity. By doing so the game will not only be more enjoyable, but it will also recreate the same physics that were experienced in the original game.

5.2.e.ii Just as in the functional requirement 5.1.b, users will also be able to govern the physics in the environment, such as wind velocity and gravity. Flying distance as well as the direction of projectiles fired from the tanks will be affected according to the physics formula. Furthermore, you can set the option to allow tanks to fall or not to fall if the ground underneath them collapses by the impact of a projectile.

# Section 6: System Architecture

## 6.1 *Basic Diagrams for System Architecture*

This section of the document will describe via diagrams and brief explanations, the overall general architecture for “Scorch3d Earth”. By nature, games, specifically games produced using OpenGL, are extremely state based. Secondly, OpenGL consists of two parts, the server (OpenGL libraries) and the client (the API that OpenGL exposes to programmers). Finally, an important feature of games is that they center themselves on a game loop (see glossary). Thus, the overall picture for the game and its loop appears as follows (see Diagram 2):

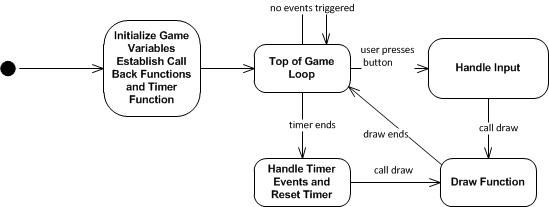


Diagram 2: General Overall Game Loop

Everything that *is* OpenGL would occur in the *draw* function above. Essentially, the programmer would set a series of states within the OpenGL fixed pipeline (see glossary), which in turn sets textures, lights, materials, etc. for the next scene to be rendered. Next the programmer would set his/her camera, and then draw the desired object(s) onto the screen. What is drawn and how it is drawn to the screen is determined by the programmer.

In order to better explain the overall process in designing the game we have derived the following picture (See diagram 3 on next page) to describe the two general sub-systems which make up the game (“the system”). Namely, these two sub-systems are the “Main Menu Sub-System” and the “Play Game Sub-System”.

Diagram three, below, depicts the overall general picture of “the system” and its two main sub-systems. These sub-systems can be considered as states within the program, which can both be reached from each other after one full game loop iteration.

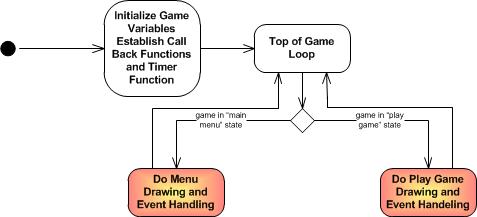


Diagram 3: The Two Main Sub-systems or “Game States”

It is important to note that these diagrams are extremely general and are only meant to give a basic overall picture of “the system”.

Next we will take a deeper look into each of the two sub-systems discussed above. The table below breaks these two sub-systems (“The Main Menu” and “The Play Game”) into sub-sub-systems.

|  |  |
| --- | --- |
| The Main Menu Sub-system | The Play Game Sub-system |
| The Windowing/Controls Sub-sub-system | The Terrain Sub-sub-system |
| The Data Structure Sub-sub-system | The Inventory Sub-sub-system |
| The Player Sub-sub-system | The Tank Sub-sub-system |
| The Environment Setting Sub-sub-system | The Physics Sub-sub-system |
| The Economic Sub-sub-system | The Background Sub-sub-system |
|  | The Projectile Sub-sub-system |

In diagram four (on next page) the game loop has been removed, and the directed arrows demonstrate relationships between each of the sub-sub-systems within their respective sub-systems. To keep the diagram clear, we have avoided drawing lines between the Sub-sub-systems of the “Main Menu Sub-system” and those in the “Play Game Sub-system”. However, we anticipate that it is obvious to the reader that the “Environment Sub-sub-system” would set the parameters for both the “Physics Sub-sub-system” and the “Terrain Sub-sub-system” in the “Play Game Sub-system”. These relationships are saved for diagram five, which immediately follows diagram four.

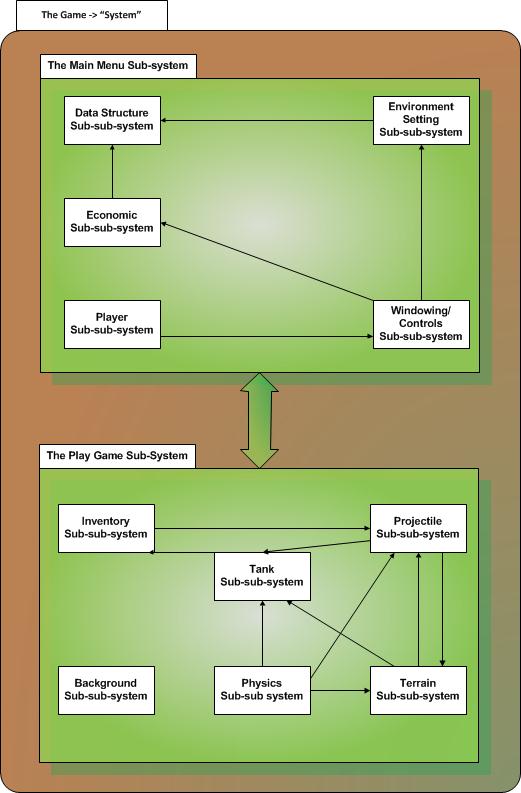


Diagram 4: Relationships between Sub-systems and Sub-sub-systems

The definitions of these sub-systems and sub-sub-systems along with their respective relationships can be seen in the tables below:

Table for Main Menu Sub-system and Sub-sub-systems

|  |  |
| --- | --- |
| Sub-sub-systems of  Main Menu Sub-system | Description |
| The Player Sub-sub-system | Because this is a multiplayer game, each player will need his/her own Data Structure Sub-sub-system (see below) as well as access to the Windowing/Controls Sub-sub-systems (see below). A “FIRST” player will be maintained by the “Main Menu Sub-system”, who will be responsible for setting the parameters in the Environment Settings Sub-sub-system (see below). |
| The Windowing/Controls Sub-sub-system | The Windowing/Controls Sub-sub-system will be responsible for facilitating user input when establishing the parameters for the “Play Game” state. It will display and manage each of the corresponding windowing items, and menu items for all the sub-sub-system menu panels of the “Main Menu Sub-system”. It is through this sub-sub-system that the user will be able to select and control his gamming environment and purchase equipment to be used in the next round. |
| The Environment Setting Sub-sub-system | This Sub-sub-system will govern and provide the parameters to set the properties of physics and type of terrain for the next game. All properties that are set will be written to the Data Structure Sub-sub-system. |
| The Economic Sub-sub-system | This Sub-sub-system will govern and provide the parameters to aid the players in selecting and purchasing ammunition as well as upgrades for the following round. All properties that are set will be written to the Data Structure Sub-sub-system for each player. |
| The Data Structure Sub-sub-system | The Data Structure Sub-sub-system will be responsible for holding all information selected by a given user. Through the Windowing/Controls Sub-sub-system. It is through this sub-sub-system that the sub-sub-systems in the “Game Play Sub-system” will obtain their data. |

Table for Game Play Sub-system and Sub-sub-systems

|  |  |
| --- | --- |
| Sub-sub-systems of  Game Play Sub-system | Description |
| The Terrain Sub-sub-system | This will be the terrain the tanks are on, and it will be deformed by The Projectile Sub-sub-system. |
| The Tank Sub-sub-system | The Tank Sub-sub-system will be controlled by a player, and will shoot projectiles through the inventory sub-sub-system. |
| The Projectile Sub-sub-system | This is determined by the inventory sub-sub-system and is responsible for destroying both the tank and terrain sub-sub-systems. |
| The Physics Sub-sub-system | Users will have to fight against wind and gravity, to destroy their opponents. |
| The Background Sub-sub-system | This is just to make things look beautiful. |
| The Inventory Sub-sub-system | This will determine the type of ammunition shot or other game based items used by each player. |

The interactions between the sub-sub-systems of the two sub-systems are depicted in diagram five on the next page. The directed arrows indicate which components affect other components. From the diagram we can see that the Data Structure Sub-sub-system of the “Main Menu Sub-system” is the bridge between the two Sub-systems for the overall “Game System”.

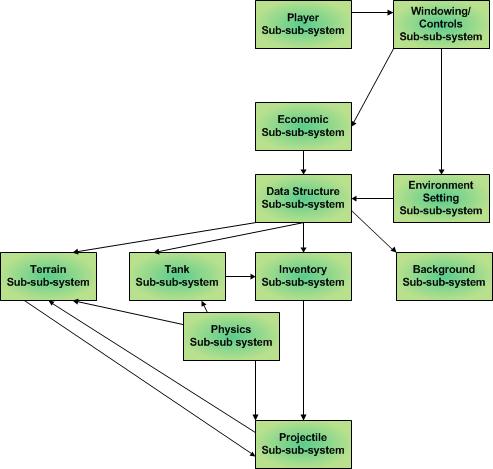


Diagram 5: Relationships between Sub-sub-systems of Sub-sub-systems

# Section 7: System Evolution

## 7.1 *Concerns for Future Expansion of Scorch3d Earth*

Game systems over the last couple of decades have changed dramatically. Just by looking at the blog post at the following website, http://www.webdesignerdepot.com/2008/12/video-game-desig n-between-1990-2008/, it is incredible how the increase in our technological capabilities have allowed for insane advancements in the video game industry. This awe inspiring progress is also a deep concern for us as game developers. In order to compete on the market, games must be done at a professional level, and released into a market where most games just don’t make it. Thus, this section briefly describes changes that would need to be made to the game in order for it to be released on the market.

According to the internet, most games today are programmed initially in C++ using the Win32 API for Windows, or an equivalent for other operating systems. These operating system (OS) specific APIs facilitate improvements in game speed and quality. Thus, one future evolution for our version of 3D Scorch3ed Earth would be to move away from GLUT and into the windowing system which is native to each OS.

Furthermore, the use of a programmable graphics pipeline would enhance the quality of our program relative to a fixed functional graphics pipeline used by older versions of OpenGL. Things such as shaders and texture mappers are the byproduct of a programmable pipeline, and allow for custom shading and enhanced visual quality within a gamming environment.

Just these two changes alone would enhance the quality of Scorch3d Earth. However, these would probably not make the game stay out on the market for long. Graphics and games will continue to advance. For example, Microsoft is in the process of promoting their new graphics and windowing system, Windows Presentation Foundation, in which it incorporates Direct3D 11 into its day to day windowing system. Innovations such as these will continue to change how Scorch3d Earth “should have” been programmed.

# References

"AC3D." *Wikipedia, the Free Encyclopedia*. Web. 26 Nov. 2010. <http://en.wikipedia.org/wiki/AC3D>.

"Game Programming." *Wikipedia, the Free Encyclopedia*. Web. 26 Nov. 2010. <http://en.wikipedia.org/wiki/Game\_programming>.

"Normal Vector -- from Wolfram MathWorld." *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource*. Web. 26 Nov. 2010. <http://mathworld.wolfram.com/NormalVector.html>.

*OpenGL - The Industry Standard for High Performance Graphics*. Web. 26 Nov. 2010. <http://www.opengl.org/>.

"Polygon Mesh." *Wikipedia, the Free Encyclopedia*. Web. 26 Nov. 2010. <http://en.wikipedia.org/wiki/Polygon\_mesh>.

Web. 26 Nov. 2010. <http://www.songho.ca/opengl/>.