

Exodus Design Document

2005.04.24

By Brian Draeger and Bradley Tetzlaff

3D7 Software

www.3d7software.org

“For the furtherance of God’s kingdom”

Contents

EXODUS DESIGN DOCUMENT.....	1
CONTENTS.....	2
<i>Design Document Modification Rules.....</i>	<i>4</i>
1 PROJECT INFORMATION.....	4
1.1 INTRODUCTION & OVERVIEW.....	4
1.2 GOALS.....	5
1.2.1 To Glorify God.....	5
1.2.2 To Make an Interactive Way in Which People Can Learn of the Bible.....	5
1.2.3 To Make an Enjoyable and High-Quality Program.....	6
1.2.4 To Learn How to Work as a Team.....	6
1.2.5 To Improve Our Skill in the Areas in Which We Work.....	6
1.3 THE TEAM.....	6
1.3.1 Team Members and Assignments.....	6
1.3.2 Development Rules.....	7
1.4 VERSION CONTROL.....	8
1.5 RESEARCH AND SOURCES.....	8
1.5.1 Useful Internet Sources.....	9
1.5.2 Useful Bible Sources.....	9
2 COMMON ELEMENTS.....	10
2.1 UTILITIES.....	10
2.2 IO.....	10
2.3 GRAPHICS.....	10
2.4 AUDIO.....	11
2.5 GRAPHICAL USER INTERFACE (GUI).....	11
3 ISOMETRIC GAME ELEMENTS.....	13
3.1 THE TILE MAP - EXODUSMAP, GAMEMAP.....	13
3.1.1 Purpose.....	13
3.1.2 Map Margin.....	13
3.1.3 Data Structures.....	13
3.2 THE ACTIVE DATA - EXODUSACTIVEDATA, GAMEACTIVEDATA.....	14
3.2.1 Purpose.....	14
3.2.2 Data Structures.....	14
3.3 THE AI - EXODUSAI, GAMEAI.....	16
3.4 TILES - MAPTILE, MAPTILETYPE.....	16
3.4.1 Elements.....	16
3.5 BUILDINGS - GAMEBUILDING, BUILDINGTYPE.....	17
3.5.1 Elements.....	18
3.6 TERRAIN FEATURES - GAMETERRAFEATURE, TERRAFEATURETYPE.....	19
3.7 CHARACTERS - GAMECHARACTER, CHARACTERTYPE.....	19
3.7.1 Characteristics.....	19
3.7.2 Game Actions and Animation.....	20
3.7.3 Health and Experience.....	22
3.7.4 Flying.....	23
3.7.5 Carrying.....	23
3.8 WEAPONS - GAMEWEAPON, WEAPONTYPE.....	23

3.9 INVENTORY - GAMEINVENTORY, INVENTORYTYPE.....	23
4 EXODUS GUI.....	24
4.1 OUT-OF-GAME GUI.....	24
4.1.1 Main Menu.....	24
4.1.2 Options Screen.....	24
4.1.3 Save/Load Game Screen.....	25
4.1.4 Information Screen.....	25
4.1.5 Credits/Sources Screen.....	25
4.1.6 Briefing Screen.....	25
4.1.7 Debriefing Screen.....	26
4.2 IN-GAME GUI.....	26
4.2.1 Controls and Displays.....	26
4.2.2 Indicator Overlays.....	28
4.2.3 Speech & Message Display.....	28
5 GAME PLAY.....	29
5.1 TRANSITIONS.....	29
5.2 INPUT.....	29
5.3 MULTIMEDIA.....	29
5.3.1 Graphics.....	29
5.3.2 Audio.....	30
5.3.3 Music.....	31
5.3.4 Cursor.....	31
5.4 THE CAMERA.....	31
5.5 MISSIONS.....	32
5.5.1 Purpose and Operation.....	32
5.6 CUTSCENES.....	32
5.6.1 Purpose.....	32
5.6.2 Overview of Implementation.....	32
5.6.3 Script Usage in Cut-Scenes.....	33
6 AI & SCRIPTS.....	34
6.1 ARTIFICIAL INTELLIGENCE.....	34
6.1.1 Character Routines.....	34
6.1.2 Movement Algorithms.....	34
6.1.3 Character Relations and Battle AI.....	38
6.2 SCRIPTS.....	41
6.2.1 Purpose and Use.....	41
6.3 SCRIPT VARIABLE TYPES.....	41
6.4 GENERAL SCRIPT FUNCTIONS.....	42
6.5 EXODUS SCRIPT FUNCTIONS.....	42
6.5.1 Cut-scene Functions.....	43
7 THE GAME CYCLE.....	44
7.1 CLASS INTERACTION.....	44
7.2 MODES.....	44
7.3 ORDER OF OPERATIONS.....	45
7.3.1 Input Handling.....	45
7.3.2 Drawing Order.....	45
7.3.3 AI & Script Activities.....	45
7.3.4 Complete Order.....	45
8 EXODUS MAP/ACTIVE DATA EDITOR.....	47
8.1 MODES AND INTERFACE.....	47

8.1.1 Toolbar.....	47
8.1.2 General Parameters Setup.....	47
8.1.3 Map Tile Editing Mode.....	47
8.1.4 Terrain Feature Editing Mode.....	47
8.1.5 Building Editing Mode.....	47
8.1.6 Map Region Editing Mode.....	47
8.1.7 Character Editing Mode.....	47
8.1.8 Projectile Weapon Editing Mode.....	48
8.1.9 Inventory Editing Mode.....	48
8.1.10 AI Editing Mode.....	48
8.1.11 Console/Text Display.....	48
8.2 AUTOMATIC TERRAIN GENERATION.....	49
8.3 MAP TESTING.....	49
9 THE EXODUS WEBSITE.....	50
9.1 PURPOSE.....	50

Design Document Modification Rules

1. Edit this document with Microsoft Word or a compatible program.
2. Use the “Reviewing” features of Microsoft Word whenever you modify this document. Turn on the “Reviewing” toolbar and press the “Track Changes” button before doing any modifications. This will enable versioning of the document.
3. Press the “Styles and Formatting” button on the “Formatting” toolbar to see the list of available styles. Always use those styles when making new headers, dividers, or tables. The contents will only include headings that are rightly marked as such. Also use cross-references when labeling figures and tables.
4. Have others review the changes if necessary.

1 Project Information

1.1 Introduction & Overview

The Exodus of the Israelites is one of the most important events of human history. By reading the record of it in the Bible (in the Pentateuch), we can learn very many essential lessons about God, our relation to God, love, law, sin, the ancient Hebrew culture, and much more. Furthermore, there is a great need for people to know the lessons of the Bible, especially in our current time. Many people do take the time to read it, but there are also many people who have misconceptions about the Bible or are too lazy to read it. However, many people do enjoy playing games, solving puzzles, and watching animations. One way to evangelize to those who would not read the Bible otherwise would be to design an interactive simulation of the Exodus of the Israelites as a computer program. A program like this could present the lessons of the Bible in a way that many people—both Christians and non-Christians—would enjoy using. The Exodus project has started for this reason and others. (Please review the goals below.)

The title of this project is “Exodus,” the code name is “Exodus” and title the finished program is titled “Exodus: The Path of Promise.” Exodus will be an educational computer game that depicts the Exodus of the Israelites in an accurate manner. The player can follow the Israelites as they journey through the deserts to the Promised Land. The player will learn of God's will, His laws, His blessings, and the consequences of sin. It will include the message of the Gospel at the conclusion of the game. The game is primary a strategy game with role-playing elements such as cutscenes. The game will start as the Israelites are fleeing from Egypt and from Pharaoh's army. It will follow the books of Exodus and Numbers closely. Parts of Leviticus and Deuteronomy will also be present.

The game will be divided into cutscenes and missions. Whenever Moses talks with God or other Israelites in the Bible, then a cutscene will play. When the Bible tells generally what the Israelites did (such as gathering manna or fighting a battle), then a mission will occur. The player will need to succeed at bringing out the Biblical outcome to each mission. Each mission will include a briefing that gives a description of the situation, references to the story as it appears in the Bible, and a list of objectives for the player to achieve. Some of the missions will have opportunities for the player to answer questions that test how well they know the God's will. Finally, Exodus will include a historical perspective on how it was for the Israelites to make the journey to the Promised Land. The game ends when the Israelites have reached the Western shore of the Jordan River.

1.2 Goals

The goals of the project are very important. The team should always have these goals in mind when developing Exodus. If there is a loss of focus, the team should meet for prayer and guidance from God. The project can only be completed successfully if the Lord is the guide of it.

1.2.1 To Glorify God

God is the Creator and Ruler of all things. He gives to us and he takes from us. We must use the gifts we have received from Him to glorify Him above all else. He must be our guide in this project and we should give thanks to Him. We should work on the Exodus project as though we are working for God, for this is the case.

“I will praise God's name in song and glorify him with thanksgiving” (Psalm 69:30).

1.2.2 To Make an Interactive Way in Which People Can Learn of the Bible

One of the primary reasons for making Exodus is to provide everyone with a very easy way to learn about God's Word, the Bible. The program should be able to be used for evangelism. It should be an accurate and historical depiction of the Exodus of the Israelites. It should also include some way to learn of the Gospel message of Jesus Christ. The user of this program should also be able to learn interactively of what is written in the Bible through input, text, graphics, and audio.

1.2.3 To Make an Enjoyable and High-Quality Program

This program should be not only a simulation of the Exodus, but also a game that allows a person to have fun while learning God’s Word. In order to spread the Word of God effectively, this program should also express the thoughts and emotion of the individuals of the Exodus. When someone plays this game, it should stimulate the player’s senses in every way necessary for the story to be conveyed well. It should also be a high-quality program that does not include bugs, inaccuracies, or difficulties for the user.

1.2.4 To Learn How to Work as a Team

Since Exodus is a large project and cannot be easily completed alone, we must work together (and with God) to have success in this project. We should be respectful to one another and we should follow the development rules upon which we have decided. We should encourage one another and be good examples of the life we have in Jesus.

“For God did not appoint us to suffer wrath but to receive salvation through our Lord Jesus Christ. He died for us so that, whether we are awake or asleep, we may live together with him. Therefore encourage one another and build each other up” (1 Thes. 5:9-11a).

1.2.5 To Improve Our Skill in the Areas in Which We Work

As we develop Exodus, we should learn how to perform our tasks better than we knew before we started. We should improve and test our skill. We should be willing to take on a challenge and we should do so only when we have prayed to God about it.

[Matthew 25:14-30](#)

1.3 The Team

The Exodus development team is responsible for the development and success of the Exodus project.

1.3.1 Team Members and Assignments

<i>Name</i>	<i>E-Mail Address</i>	<i>AIM</i>	<i>Phone</i>	<i>Main Role</i>
God				Leader
Brian Draeger				Programmer, Manager
Chris Gullicksen				Graphics Designer
Bradley Tetzlaff				Programmer, Organizer
<i>Additional Helpers</i>				
Eric Slawyk				CD Designer
Kyle Stittleburg				Server Manager

Table 1.3.1.A Team Member Listing

- ◆ The Exodus development forum at <http://3d7software.org/forums/viewforum.php?f=11> should be used for discussion of the plan, progress, problems, and implementation.

1.3.2 Development Rules

These rules are designed to ensure smooth development and cooperation of the involved team members.

All

1. The conduct of all members and the content that they produce must be according to God's will. All content should be produced in accordance with Scripture. Artwork accuracy, programmed game features, etc. should not conflict with what the Bible states.
2. Any conflicts that arise should be handled with (1.) prayer to God, (2.) discussion between those involved, (3.) action to end the conflict, and (4.) prayer to thank God.
3. Major changes in plan should be discussed in a meeting, on a forum, or by AIM. Everyone involved should be aware of major changes in the plan within a week of it being made known to one other person.

Programmers

4. The planning of any module of a program can be done as a team effort or individually. If planning is done individually, then the plan must be proposed to the team shortly after. Work may be done on a module before the plan is known by the team, but it is not recommended. Those who do so must be accepting to modification suggestions, provided the proper evidence is provided.
5. The assignment of classes should be undertaken by all of the involved members. However, a new class is initially assigned to those that the original class creator believes it should be assigned to.
6. Members must be respectful to one another. If someone has a good suggestion for improvement and supports this with evidence, then the team should respect this suggestion. This means that the team evaluates it and decides (possibly votes) as to whether action should be taken for it.
7. Only those who are the designated author of a class may change the meaning of the existing code.
8. If code causes a compilation or linking error or if it has a runtime error, this code may be changed by anyone who finds this problem. This person should understand the code fully before attempting to fix the problem. Preferably, the author should fix the problem and it is may be best to wait for the author to fix the problem if someone other than the author found it. (Then wait for the author to decide whether to change it.) This is to honor the original author's intentions and work respectfully in a group project.
9. Code (functions and fields) may be added to a class by a member that is not assigned to that class as long as the member understands the workings of the class, cannot easily contact the author, and can justify the addition with good reason. No existing functions or member fields should be removed or edited.
10. If a member assigned to a class finds code that has been added by a member that has not been assigned to it, then this code may be removed if the member finds that it is

unnecessary. The person who added it should be notified before it is removed, but this is not essential if communication is unavailable at the time.

11. Optimizations to code of a class that is not assigned to a member that would like to make the optimization should be handled by writing an optimization comment description in the code. The assigned member should be notified.
12. If a member who is not assigned to a class finds code that must be changed in that class, then appropriate comments should be placed either in the code or in a separate file. A member assigned to that class should be notified of the change request.
13. Proper commenting and documentation should be present in all completed classes.
14. If any code does not follow the coding guidelines described in “Exodus_Coding.doc”, then the code should be commented, unless it causes a compilation or runtime error in which case it can be changed to work properly. The code’s meaning should not change. These guidelines should be followed at least by 95% in all members’ work. If there is a problem meeting this, use the automatic code formatting features of the IDE.

1.4 Version Control

The Exodus project will be under version control by the Subversion VCS. The server is located in West Allis and it is run by Kyle Stittleburg. The address of the server on 2005.04.27 is `svn://melonballs.com/exodus`. The Tortoise SVN program can be used to access the repository. This program can be downloaded at <http://tortoisesvn.tigris.org/>. Each developer should have a user account. Care should be taken to ensure the integrity of the Subversion repository.

1.5 Research and Sources

In order to provide an accurate depiction of the Exodus of the Israelites, we must perform accurate research. Of course, we should use the Bible as the first and foremost source of information about the Exodus. It certainly has the best possible account of it. The Bible, however, does not provide all of the information we need to depict a visual representation of the Exodus. Therefore, we must also conduct historical research on the cultures present in the Exodus account and the historical terrain of the Exodus route. We should also perform some research to ensure that we properly interpret the Bible.

Please follow these guidelines when perform research so that the integrity of the Exodus program can be ensured to a high degree:

1. Use reliable sources. Perform verification on the author if possible.
2. Use care when documenting your findings. Ensure that any quoted material is correct.
3. Always document the sources you use in “Exodus_Sources.doc” in the “Information” folder. Use MLA formatting.
4. Always obey copyrights and legal licenses when working with sources of information.
5. Try to verify your findings with experts and other sources.

1.5.1 Useful Internet Sources

- ◆ Christian Coders Network - <http://www.christiancoders.com/>
- ◆ Game Developers Network - <http://www.gamedev.net/>
- ◆ Flipcode Game Developers - <http://www.flipcode.com/>
- ◆ The Bible (with audio for the NIV) - <http://bible.gospelcom.net/cgi-bin/bible>
- ◆ The Old Testament Gateway - <http://www.otgateway.com/exodus.htm>
- ◆ Historical Information on Israel - <http://www.bible-history.com/>
- ◆ Israelite Exodus Information - http://www.hum.huji.ac.il/Dinur/Internetresources/historyresources/biblical_studies.htm
- ◆ A Map of Israelite Exodus Travels - <http://scriptures.lds.org/biblemaps/2>

1.5.2 Useful Bible Sources

- ◆ 2 Cor 3:7
- ◆ Heb 3:3-4
- ◆ 1 Kings 9:26
- ◆ 1 Kings 8:56 - “Exodus: The Path of Promise”

2 Common Elements

In order to construct Exodus, many smaller component classes must be used to build the larger components. This chapter reviews these components. Most components are serializable and deserializable using the insertion or extraction operators and FileAccess.

2.1 Utilities

TextString: A flexible version of a string.

ArrayList: A flexible array structure that can automatically resize and shrink. The array index must be consecutive.

ArrayKeyList: Like the ArrayList, but each element contains a unique ID. It should be searchable using a binary search.

Array2D: A dynamically allocated 2D array. This class is small and simply allows for less code reduplication when using 2D arrays.

ChainList: A unidirectional linked-list. Each element has a unique ID.

BinaryTree: A tree of nodes with unique IDs and two branching pointers.

RangeList: A hash table structure that requires an extension structure. A set of ranges is used to determine which extension node should be used. It supports a key on the range and a different key on the extension structure.

Timer: A basic means of recording a start and end time.

2.2 IO

FileAccess: A standard way to easily write and read data to or from a file. It can handle object serialization.

IDataConnector: An interface used during serialization to request the connection of a previously set pointer.

IIOSupport: A class that can connect a pointer during deserialization.

2.3 Graphics

GraphicsDevice2D: The graphics drawing device.

GraphicsFont: Draws text to the screen in a certain style. It can also draw a bitmap font if one is given. It can draw multiple lines of wrapped text.

ColorARGB: A generic color that can be cast into an unsigned integer implicitly.

GraphicSurface: A loaded bitmap file.

SGraphic: A single graphic image.

AGraphic: This animated graphic stores the offset positions of each frame and it assumes that they are all the same size. It also contains the height and width of a single frame. It contains the frame duration, a looping property and a possibly a stopped property. It should contain a way to generate the frame positions automatically based on the number of frames per line and the number of pixels between each frame.

AGPointer: The AGPointer class contains a pointer to an AGraphic. It also contains the actual duration into the animation in milliseconds or the start time of the animation (in millisecond mission time). If the millisecond mission time is given, there is no need to increment the

number of elapsed milliseconds on each game cycle—the current frame is found by subtracting the current mission time from the mission time when the animation started.

AGPosition: Inherits AGPointer and adds another variable that stores the frame position also. This type of AGPointer is used when multiple objects (such as tiles) are all animated in the same way. Saving the frame position prevents need for recalculation.

Coord2: An (x, y) coordinate.

Coord3: An (x, y, z) coordinate. (Not used often, but can be used for 3D audio.)

Coord4: A set of a top left coordinate and a bottom right coordinate, can be used for defining rectangles.

2.4 Audio

AudioDevice: The master control of audio operations.

AudioSample: An audio file that is loaded into memory, can be looped and played in 3D if the position of the object is given in relation to the player.

AudioStream: An audio file that is streamed when it is played. This is used for music.

AudioChannel: A channel of a currently playing AudioSample.

AudioProperties: Extension properties that can be used for more control over a sound when they are used in conjunction with an AudioSample or AudioStream.

2.5 Graphical User Interface (GUI)

UITheme: A user interface theme data set. This class enables easy control over how the GUI looks. It provides information about how components should look and what sound should play when working with them. It can also be used to draw an adjustable plane to the screen. It will scale the center image piece according to a given size and it will use eight small image pieces for a border. It could include type information in an ArrayList for each type of user interface control.

UIWindow: An actual window of the native operating system. This class does not actually contain any visual data other than a background. This class is an easy way to make a new window. A form should be placed in the window to add controls. The first form to be added to the window can only be closed when the window is closed.

UIRegion: This generic class gives common functionality to all of the other controls. It gives a region of the screen events for a click (from UIComponent), double-click, right-click, mouse release, mouse hover, mouse exit, drag (with direction given), focus, lost-focus, and key press on focus. However, these events are stored in an ArrayList and therefore they are all optional.

UIForm: A UIRegion that can contain UIRegions. It acts like form if necessary. It can be given a particular background. It should have an option to display a title bar or not.

FramedLabel: A label with an adjustable background. It can be displayed when a control is hovered over. It is not a UIRegion, which is why it can be used to construct other controls. It should be used as a tool tip if necessary.

Label: A positioned piece of text. It can be either drawn using a bitmap font or by a vector font.

Button: A clickable part of the screen that can change its graphic based on mouse hover and mouse press. It can be toggled in order to work like an option button or check button.

SliderBar: A bar that has a button that can be slid from one side to another. It can be horizontal or vertical.

UIImage: An image that inherits UIRegion and can be placed on a Panel.

TextBox: A text box that can display multiple lines of text and can perform text entry.

Menu: A menu or list box control. It will display a list of text items and either show a slider bar or grow if there are more items than those that fit in the given region.

DialogBox: An advanced control used to show a message to the user. It can have a list of buttons with descriptions next to them.

EventHandler: A class that represents a handler of a particular event. The event can be of many different types.

Events: (This is a title, not a class name.) Some delegates for events are as follows.

- ◆ `UIEvent(UIRegion sender);`
- ◆ `MouseEvent(UIRegion sender, Coord2<PIXELS> pos, float dir);`
- ◆ `KeyboardEvent(UIRegion sender, unsigned short keyCode, MetaKey meta);`

3 Isometric Game Elements

These are found in the Isometric namespace of the Games namespace. Most objects on the map inherit a general MapObject class and mobile versions inherit a MobileMapObject class. The types should inherit a GameObjectType class. In terms of graphics, the game objects appear at 30-degree angle from the top with the viewer being at the southeast. This means that there is a (semi-)isometric look to the objects. They also are drawn in layers starting from the upper left corner to the lower right corner. The layers can generate a 3D effect.

3.1 The Tile Map - ExodusMap, GameMap

3.1.1 Purpose

The tile map is meant to provide a layered backdrop for the game. It includes a large array of tiles that should always be drawn first as the default backdrop. The map is to be constant at game-time—changing data should be in the active data class. The division of constant map data and active data allows for faster game reloads since the map data can remain the same if the loaded file is still on the same map. The map also provides elevation and regions for AI purposes.

3.1.2 Map Margin

The tile map should include a one-tile margin around its border. The margin is used to simulate a smooth continuation with imaginary terrain. It should not be able to be seen when the game is running, so its main purpose is to supply smooth texture rendering on the side and to allow trees or rocks to continue past the edge of the map. In the map editor, possibly the only way to draw on the margin tiles would be to use a spray nozzle larger than one tile.

All AI navigation tasks should do their range validation from one tile in from the sides of the map. The real advantage to the margin is that it allows characters to come in from the sides of the map (such as a convoy) without a need for them to suddenly appear on already visible tiles. Since movement AI does not consider the margin, the characters in the margin are only allowed to move to the adjacent tiles on the visible map.

3.1.3 Data Structures

<i>Data</i>	<i>Structure, ID, Size, Sort</i>
Map Tile Array ^P (MapTile)	Array2D on <i>x</i> and <i>y</i> (TILES), size is map width by map height
Map Tile Types ^{SP} (MapTileType)	ArrayList on type ID (SMALL_ID), ascending sort
Buildings ^P (GameBuilding)	ChainList on building ID (MEDIUM_ID), ascending sort
Building Positions (BuildingRange)	RangeList on <i>y</i> (TILES), size is map height; extended by ArrayList with generic ID (MEDIUM_ID), ascending sort
Building Types ^{DP}	BinaryTree on type ID (MEDIUM_ID)

Terrain Features ^P (GameTerraFeature)	RangeList on y (TILES), size is map height; extended by ArrayKeyList on x (TILES), ascending sort
Terrain Feature Types ^{SP} (TerraFeatureType)	ArrayList on type ID (MEDIUM_ID), ascending sort
Inter-Region Positions ^P (Coord2<PRECI_PIXELS>)	Array2D on region ID “from” and region ID “to” (SMALL_ID), size is region count squared
Region-to-Group Data ^P	ArrayList on region ID (SMALL_ID), size is region count, ascending sort
Region Groups ^P (SMALL_ID)	RangeList on region group ID (SMALL_ID), size is region group count; extended by ArrayList on region ID, ascending sort
Bitmap Surfaces ^{DP} (DxSurface)	BinaryTree on bitmap file name (TextString&)
Audio Samples ^{DP} (AudioSample)	Optional BinaryTree on audio file name (TextString&)

Table 3.1.3.A Map Data Structures. The ^S indicates that the types are in a set and are not loaded on demand, ^D is on-demand external loading. The ^P indicates that the value may have persistent pointers to its data. Optional structures are loaded when needed, null otherwise.

3.1.3.1 Other Data

Map Data: Record any special information about the map.

Type File Data: Record which type files should be loaded.

Tile Type Bitmap: Use a single graphic for the tile set.

3.2 The Active Data - ExodusActiveData, GameActiveData

3.2.1 Purpose

The active data is meant to represent the active portion of the game. It contains all of the data structures that are likely to change during the game. It is likely to be manipulated by the AI frequently. A save of a player’s game saves this portion of the data.

3.2.2 Data Structures

<i>Data</i>	<i>Structure, ID, Size, Sort</i>
Characters ^P (GameCharacter)	RangeList on character ID (MEDIUM_ID), size is 12; extended by BinaryTree on character ID (MEDIUM_ID)
Live Character Positions (GameCharacter*)	RangeList on y (TILES), size is map height; extended by BinaryTree on x position (TILES)
Dead Character Positions (GameCharacter*)	RangeList on y (TILES), size is map height; extended by BinaryTree on x position (TILES)
Flying Character Positions (GameCharacter*)	ChainList on character ID (MEDIUM_ID), unsorted
Character Types ^{DP}	BinaryTree on type ID (MEDIUM_ID)

(CharacterType)	
Character Groups ^P (GameCharacter*)	BinaryTree on group ID (MEDIUM_ID); extended by ArrayList on generic ID (MEDIUM_ID), unsorted
Carried Health/Exp. (CharacterStatus)	ArrayKeyList on character ID (MEDIUM_ID), ascending sort
Game Action Types ^{DP} (GameActionType)	RangeList on type ID (MEDIUM_ID), size is 6, extended by BinaryTree on type ID (MEDIUM_ID)
Weapons ^P (GameWeapon)	ChainList on weapon ID (MEDIUM_ID), unsorted
Weapon Types ^{DP} (WeaponType)	BinaryTree on type ID (MEDIUM_ID)
Inventory on Map (GameInventoryItem)	RangeList on y (TILES), size is map height; extended by ArrayKeyList on x position (TILES), ascending sort
Inventory Store (GameInventoryItem)	ArrayKeyList on inventory ID (MEDIUM_ID), ascending sort
Inventory Types ^{SP} (InventoryItemType)	ArrayList on type ID (MEDIUM_ID), ascending sort
Region Visitation ^P (RegionVisitation)	ArrayList on region ID (SMALL_ID), ascending sort
Team Status/Data ^P (TeamStatus)	ArrayList on team ID (SMALL_ID), ascending sort
Global Script Variables (ScriptVar)	BinaryTree on script variable ID (VARIABLE_ID)
Positioned AGraphics ^{DP} (PositionedAGraphic)	ArrayList on layer ID (SMALL_ID), size is layer count; extended by Optional ChainList on graphic ID (MEDIUM_ID), unsorted
Positioned AudioSamples ^{DP} (PositionedAudioSample)	ChainList on audio ID (MEDIUM_ID), unsorted
Bitmap Surfaces ^{DP} (DxSurface)	RangeList on initial character (char), size is 6; extended by BinaryTree on bitmap file name (TextString&)
Audio Samples ^{DP} (AudioSample)	RangeList on initial character (char), size is 6, extended by BinaryTree on audio file name (TextString&)

Table 3.2.2.A Active Data Structures. The ^S indicates that the types are in a set and are not loaded on demand, ^D is on-demand external loading. The ^P indicates that the value may have persistent pointers to its data. Optional structures are loaded when needed, null otherwise.

3.2.2.1 Other Data

Mission Data: Record the current mission/cutscene number/name and the briefing data file name.

Type File Data: Record which type files are loaded. Also record which map is loaded.

Script Data: Include a link to the current script file and include the ability to have a script function be executed upon mission save and load. This allows for special customization (such as setting the counter icons) when a game is loaded or saved. Allow a script to save some variable data for faster processing. Include the ability to run a script at a certain point in the game cycle.

Music: Have the currently loaded streaming music.

Current Character Selection and Action List: Must store the displayed character selection.

Counters: Record the current value of the mission time and the other counters.

Status Bar: Include the string that is currently displayed on the status bar and the amount of time remaining before it should disappear.

AI Status: Include any special information such as if the player is locked out of all interaction (as for a cutscene), etc.

Weather: Include a byte that tells how the weather conditions currently are. The following table describes the conditions.

<i>Bits</i>	<i>Value</i>	<i>Description</i>
Lightness 0, 1	0	Morning
	1	Before Noon
	2	Afternoon
	3	Night
Temperature 2, 3	0	Cold
	1	Normal
	2	Hot
	3	Desert
Precipitation 4, 5	0	None
	1	Clouds
	2	Light Precipitation
	3	Storm
Reserved 6, 7	0	
	1	
	2	
	3	

Table 3.1.3.B The Weather Bit Field

3.3 The AI - ExodusAI, GameAI

These classes handle the movement AI and the battle AI of the characters in the game. GameAI is a general class for common AI tasks. These classes only store temporary AI processing data. They are mainly libraries of functions.

3.4 Tiles - MapTile, MapTileType

Tiles are the foundation for position and graphics. They are to be 32x32 pixels. Each map is composed of a 2D array of tiles. The tile allows for better efficiency by dividing the map into a grid. Tiles also allow for better drawing capabilities and combinations. The MapTile class should take only 4 bytes.

3.4.1 Elements

Graphic: The graphic should be 64x64 pixels, but only the center 32x32 pixels should be fully opaque. The 16 pixel border should be used to blend the tile graphic with the surrounding tiles. This blending will allow for more flexibility and effect. Graphics may be animated, but the animation frame is universal to all tiles of that type. A single bitmap should be used to store all of the tile graphics.

Variation and Tint: Each tile type may have a set of variations (ID as a `SMALL_ID`) that is used to give a more realistic terrain appearance. When placing tiles in the map editor, the editor can automatically choose a certain variation on a tile type. Another possible use for this feature is with water-shore matching. Water and shore should not be blended, so using a variation may allow for a correct appearance.

The tint and the variation should be stored in the same byte, so there are able to be 8 variations (bits 0 to 2), and 16 tints (bits 4 to 7). Bit 3 is a special bit that represents whether a building is present at that location. This allows the AI to know quickly when a character has walked into or out of a building. If the value changes to a 1 or from a 1, then the building data structure should be searched.

The tint of a tile allows for hill effects. There are four levels of tint. Although it could be calculated by the elevation differences, it should be stored with the elevation to improve performance. Tint should change for night missions. The actual tint colors should be stored in an array of color values.

<i>Value</i>	<i>Description</i>	<i>Value</i>	<i>Description</i>
0	No tint	8	Darker 80%
1	Lighter 20%	9	Green 20%
2	Lighter 40%	10	Green 40%
3	Lighter 60%	11	Brown 20%
4	Lighter 80%	12	Brown 40%
5	Darker 20%	13	Special tint 1
6	Darker 40%	14	Special tint 2
7	Darker 60%	15	Special tint 3

Table 3.4.1.A Tint Values

Elevation: The z dimension of the tile. Each tile has a height to simulate changes in elevation and impassable terrain (from large differences). Character speed can be affected by it. Elevation may span from -128 to 127. Negative elevations are below sea level and should cause the bottom of non-floating characters to be cut off. All tiles that are at -8 in elevation or less are considered to be water.

Region: Each tile belongs to a particular region of the map. Region IDs can be used to group tiles (determine rooms), allow for darkness/fog and clearness, provide movement AI, mark hotspots for AI, and to mark special regions that trigger a script when entered.

3.5 Buildings - *GameBuilding*, *BuildingType*

Buildings (“structures” may be a better name) provide a unique feature that is not present with other game element—multi-tile converge. A building like an advanced `GameTerraFeature`. It allows for both an exterior perspective and an interior perspective. Characters may enter buildings too. Unlike buildings in other games, the Buildings of Exodus are fixed at design time and cannot be destroyed or built during game play. The main reason for this is that rarely buildings can be built so quickly or destroyed completely. Furthermore, the technology level of the Exodus does not permit large buildings from being changed in any drastic way by weapons or construction. Fixed-position characters should be used if there is a need for a constructible/destructible building.

3.5.1 Elements

Base Position: This tile position marks the upper left corner where the building's floor or base starts. It allows for a true 3D effect since it marks the start and end of the overlap. Characters above and to the left of this position are to be covered by the building's graphic.

Graphic Offset: This pixel position marks the base position offset to the upper left corner from which all graphics are drawn. Unlike other map objects, this part of the building is allowed to overlap with other objects, including buildings. It is not actually part of the building.

Building Size: The height and width of the building's graphic.

Exterior Graphic: This graphic is displayed whenever no characters are in the building. It shows the outer parts of the building. It is not displayed at all when characters are within the building.

Interior Back Graphic: This graphic is displayed in the back of characters that are within the building. It normally represents the back north and west walls of the building. It should be drawn when all of the characters to upper left of the base position are drawn.

Interior Front Graphic: Optional. This graphic is displayed in front of the characters that are within a building. It normally represents the south and east walls or the ceiling. It should be drawn when all of the characters to the upper left of the lower right corner of the graphic have been drawn.

Parameters: One byte of parameter flags should be in the BuildingType class for extensibility. This could indicate the rotation of the building so that the far corner would be something other than the upper left.

Visitor Count: The number of characters that are present in the building at the time. (This can be regenerated when the active data is reloaded.) The interior is shown if this value is above zero.

Tile Change Map: Optional. This 2D array of tile type IDs is only needed in map editing mode. It is used to set the tiles quickly under the building when it is placed on the map. Its size is made from the base position to the graphic's lower right corner. If tile type 255 is given, then no change in the type should be made but the tile is considered an interior square. If tile type 0 is given, then no change in the tile type is to be made and the tile is not considered part of the building.

255	43	34	53	255
23	23	34	21	25
66	34	34	23	35
5	33	0	0	23

Figure 3.5.1.A Tile Map Diagram

Elevation Change Map: Optional. This 2D array of elevation values is only needed in map editing mode. It is used to set the tile elevations quickly under the building when it is placed on the map. Its size is made from the base position to the graphic's lower right corner. If elevation -128 is given, then no change in elevation is made. Note that these are strict values, not offsets of the current elevation.

-6	128	128	32	-128
128	-10	0	0	128
128	0	0	0	128

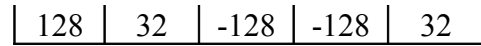


Figure 3.5.1.B Elevation Map Diagram

3.6 Terrain Features - GameTerraFeature, TerraFeatureType

Terrain features are a primitive way to simulate a 3D effect in a 2D environment. These features normally represent trees, bushes, and rocks. They should be drawn over characters that are above them on the screen. They are only able to be 32x32 pixels at most because they only are allowed a single position on the map. (If larger graphics are needed, use buildings.)

The ID of the terrain feature type should be used to determine if the feature changes the elevation of the tile when it is placed on the map. All even IDs should make the terrain impassible when the feature is placed, all odd IDs should keep it the same as it is.

3.7 Characters - GameCharacter, CharacterType

3.7.1 Characteristics

Every character is slightly different and most of these differences are binary (true/false). The parameters of a character allow special information to be provided. Parameters are stored in bit fields to save memory. The parameters are as follows:

Bit(s)	False Meaning	True Meaning
0	Does not cast a shadow	Casts a shadow
1	Do not draw carried character	Draw carried character on top
2	Does not float on water	Floats on water (no sinking below 0)
3	Cannot swim	Can swim/Is ship
4, 5	00 = Is a person, 01 = Is an object, 10 = Is an animal, 11 = Is a vehicle	
6	Is male	Is female
7	Is young	Is old
8-15	Reserved	Reserved

Table .A CharacterType Parameters

Bit(s)	False Meaning	True Meaning
0, 1	00 = Is idle, 01 = Is retreating, 10 = Is attacking, 11 = Is confused	
2	Is not waiting	Is waiting (for another character to move)
3	Is not carrying a character	Is carrying another character
4	Is on the ground	Is flying
5	Is visible to enemy AI	Is hidden to enemy AI (not graphically hidden necessarily)
6	Is not sick	Is sick, health will slowly decrease until death
7	Has light off	Has light on (torch is lit)
8	Does not have its data carried to next mission	Does have its data carried to next mission

9-15 Reserved	Reserved
-----------------	----------

Table .B GameCharacter Parameters

3.7.2 *Game Actions and Animation*

A character is always performing some kind of action. Actions in terms of characters are very much like animations that can be played in eight variations based on character direction (it is also possible to have just four or one animation defined). An action also serves semantic and AI purposes. Each action has an ID that is universal to all characters. If a character switches to a new action, then the character should change appearance and the action animation should play. Certain actions also include events like firing a weapon or picking up an object.

Game actions should include:

1. The universal action ID (as a `short` integer).
2. The next required action ID. This action must be performed before the next requested action can be performed. For instance, if a character is crawling on the ground, then this required action ID should be set to another action that displays the character getting up before the next requested action is started. If an adjacency list is set for a required action, it should be ignored. If this is zero, then the next requested action should take place immediately.
3. An AGraphic list in 16 directions, with 0 as N and moving around clockwise to NW at 16. Normally, only 8 graphics will be given and the algorithm should shift all directions to the nearest value. If only four AGraphics are given, then diagonal direction should be shifted clockwise to the nearest straight direction (e.g. NE >> E). If only one graphic, then this should be displayed regardless of direction.
4. An optional sound that should be played when the special action time is reached. This sound should be stopped at the end of the current action if it loops, but it should not be stopped explicitly if it does not loop.
5. The time of the special action in terms of milliseconds. For instance, if a character is firing a weapon, then the projectile will appear and the sound will play when the animation reaches (or just passes) this time. It enables an animation to have a start and end phase. If no special action should ever be taken, set this to a time that is beyond the animation's total duration.
6. An optional adjacency list. This list includes a character or inventory type ID, a new character type ID, and a new action type ID for each element. This list serves to show which character types must be adjacent to this character in order for the action to be performed. It is most often used for determining which characters can be carried by another character.

The new type ID and new action ID are to be used only if this character is to change to a new character type or to a new action when the action is performed with the given character type (such as when an animal lets a person ride on it). If the character type ID is 0, then no change should be made to this character's type. If the action ID is 0, then no change should be made to this character's current action. If both are non-zero, then the character should switch type and then switch to the given action. Just the character's action or type should change if only one is non-zero.

The game character data should include:

1. A current action ID to mark the action that is currently visible. This animation must complete before other actions can be done. (For looping animations, the action should continue until a requested action is set.)
2. A default motile action ID should be the action that is shown if the character must move somewhere. If a character is performing an action that cannot be done while the character is in motion but is required to move, then this ID should be set to the current action so that the character is not moved while looking like he or she should be still. This could be set equal to the current action ID to override any change and motile actions should do this. This should not be set to an adjacency-dependent action.
3. The requested action ID. Once the current animation and any required actions are complete, the character will switch to this action. For instance, if the player commands a character to stand up from a sitting state, then the getting up action should be performed and then the next requested action should be set to the current action. It should be set to the current action ID to keep the character on the same action.

3.7.2.1 Action Listings

While actions are unique and must be coded specifically, some ranges are allowed to support the addition of more actions that require similar effects. The ranges are described in the table below.

<i>Range</i>	<i>Description</i>
Even IDs	Actions that can be done while moving (motile actions). The default motile action ID of the character should be set to these unless the action has an adjacency list.
Odd IDs	Actions that must be done while still (sessile actions). If a character is required to move while performing a sessile action, then the action should switch to a motile action.
0 – 15000	Common actions, no special treatment
15000 – 16000	Prepare to fire and then fire the character's weapon
16000 – 18000	Pick up the object that is either in the followed character pointer, or directly in front of the character as long as it is of one of the type in the adjacency list. Change this character's type or action ID according to the adjacency list.
18000 – 20000	Put down the object that is being carried, if there is one. Change this character's type or action ID according to the adjacency list.
20000 – 22000	Pick up the inventory item that is directly below the character or directly in front if there is not one below, granted that it is in the adjacency list.
22000 – 24000	Put down an inventory item that is given in the adjacency list and change the character's action if requested by the adjacency list.
24000 – 30000	Execute a script
30000 – 32767	Reserved

Table 3.7.2.1.A Action Type Ranges

The following table shows a list of the possible actions a character may perform. They may be listed in the GUI of the game so that the player can select what actions the character should perform.

<i>ID</i>	<i>Description</i>	<i>Next Required ID</i>	<i>Comments</i>
0	Ignored action		The action is ignored and no change is made
2	Invisible action		No visibility
10	Move normally		Required, the standard action of all characters
401	Generic death	402	
402	Generic dead		
15000	Use weapon while moving		A character should have 15000 or 15001, not both. It depends on the weapon.
15001	Use weapon while standing		A character should have 15001 or 15000, not both.
16001	Pick up an object on the ground		Used for wood or brick on the ground
16003	Ride on another character		Used for horses and camels
18001	Put down an object on the ground		
18003	Get off of another character		
20001	Pick up an inventory item		The inventory item will go into the general inventory list.
20003	Pick up and use inventory item		The inventory item will be used on that character immediately.
22001	Put down an inventory item		The inventory item is generated by the adjacency list's first ID.

Table 3.7.2.1.B Common Actions

Possible Additional Actions

swimming, jumping, climbing, hiding, sacrificing, praying, eating, sleeping, sitting, idle actions, wandering

3.7.3 Health and Experience

Each character has a health level and an experience level. Health is lowered upon injury and it can be restored by a medical care or inventory. Since Exodus does not contain any standard way to get more characters, a character's health should *very slowly* increase over time. A character with low health could walk slower or be less like to hit a target. Each character type can have a greater or lesser maximum of health points than another.

Experience is a special factor that is used for many AI purposes. It is increased when a character successfully hits a target or performs an action. A higher experience level allows a

character to be more alert (the situation is check on more game cycles) and better at using a weapon. Morale can also affect experience.

3.7.4 Flying

Flying is the ability to move into the air. In terms of the game, it means that two characters can be in the same location at once. This also means that the flying characters must be positioned in a different structure and drawn later. Flying characters are those that have mobility greater than 32 and each value above 32 means that the character is more likely to fly than walk on land. Any flying object should not be allowed to land on a tile that is marked as a building tile. Any character that cannot swim or float should not be allowed to land on water. A character that can fly should be able to have flying and landing actions.

3.7.5 Carrying

Carrying is a special feature of characters. It allows one character to pick up another and to drop that character at another location. Carrying is only possible between certain characters, so each character should contain a list of what character types he or she can pick up. Along with the type of character that can be picked up should also be another character type ID that can be nonzero. The pickup of that character type can then change the type ID of the carrying character so that it looks appropriate. There should also be a action ID that is used to tell what action the carrying character should switch to when a character of the listed type is picked up.

In order to improve efficiency, script functions should be able to be set to be executed when a character picks up an item or puts down an item. This prevents a need to check for this event on every game cycle. A carried character should not be displayed on the map, but its position and other factors should be updated to prevent problems with scripts unless it can be proven to be safe. In terms of game play, carrying allows an immotile character to be moved.

3.8 Weapons - GameWeapon, WeaponType

The two main types of weapons are projectile weapons (spears, rocks) and short-distance weapons (sword). The GameWeapon class is only representative of projectile weapons since those are the only ones that actually need a visible object. Projectile weapons can launch and can travel in a straight path until a collision occurs. The weapon hits a character and then decreases the health and increases the experience of the character that was responsible for launching the projectile. A weapon's rate of fire could be determined by the character's graphic.

3.9 Inventory - GameInventory, InventoryType

Inventory is one way to add a true game element to Exodus. Every map has a small amount of special objects strewn about over the landscape or in buildings. Inventory items may be picked up by the Israelites (although possibly the enemy could pick them up too so there is more of a challenge). Once picked up, inventory goes into a global inventory list. Inventory can then be used or applied to any character when it is selected in the inventory list. Inventory normally does good things like increases health or experience.

Inventory should carry over from the previous mission to the next. This allows for resourcefulness.

4 Exodus GUI

4.1 Out-of-Game GUI

4.1.1 Main Menu

The main menu should include the Exodus title, a background image of a map of the locations in Exodus, a list of all the missions and cutscenes, and a list of these menu items:

1. Save/Load Game
2. Options
3. Information
4. Credits & Sources
5. Exit

In addition to those options, the list of missions and cutscenes can be used to start a new game or replay a previously played mission or cutscene. The missions and cutscenes should be disabled if the player has not reached them yet, but the very first one is always allowed so that a new game can be started. A player can also restart the current mission from here. A confirmation dialog should be displayed if the player chooses to replay an old mission when another is already in progress.

4.1.2 Options Screen

The options screen should include the ability to set the following features. The default range and the default value are shown in parenthesis. The player should be able to make these changes any time and they should take affect as soon as possible.

1. Graphics
 - a. Brightness (5% - 100%, 50%)
 - b. Scroll rate (5% - 100%, 50%)
 - c. Show corpses (on/off, on)
2. Audio
 - a. Music volume (0% - 100%, 40%)
 - b. Sound volume (0% - 100%, 50%)
 - c. Voices (on/off, on)
3. Game Play
 - a. Game speed (5% - 100%, 50%)
 - b. Carry health, experience, inventory to next mission (on/off, on)
 - c. Require key press to continue after character speech (on/off, off)
 - d. Skill (Easy - Medium - Hard, Medium)

4.1.3 Save/Load Game Screen

This screen should contain a list-box containing the previously saved games and a file name entry. The user will be able to select one of the previously saved games and save over it or create a new saved game by typing in a name for the game. If the player would like to load a game, a button for loading may be pressed and the currently selected game will be loaded. The default file name should use the mission name and a mission timestamp like “Mission Name mmm.ss”. This can help to save time when saving a game. A game delete function should also be allowed.

The possible functions are optional:

- ◆ We could save an image of the game screen when the game was saved and display that somewhere on the screen when the saved game is highlighted (This is easy to do with DirectX)
- ◆ If we have a background image of a map, we could place an object on the location the saved game is. (This would be harder to do, but not impossible)

4.1.4 Information Screen

The information screen is used to provide extra information to the player using tabbed pages. The topics should include a listing of useful resources for how the player can learn more about Christianity. It should also include general information (background information) on the Exodus of the Israelites. It could also include some help for how to use Exodus. A purpose statement may also be helpful so that the player knows the goals of the program and can know our justifications for it.

Note that the briefing screen design may be reused here.

4.1.5 Credits/Sources Screen

This screen should contain a textbox or an automatically scrolling list of credits for those who have contributed to the Exodus project. It should use MLA style citations.

4.1.6 Briefing Screen

The briefing screen is the primary means of informing the player what the next mission involves. The briefing screen is to be a set of tabbed pages. It is used to provide elements such as the mission name, the mission time/location, a summary of the situation, the objectives, and some Bible quotes relevant to the mission. It should also include a Q & A section for answering likely apologetics questions. It could relate the mission’s moral significance to present day circumstances. If possible, it would be good to include historical evidence regarding the mission’s event. If the user views the briefing again during the game, the objective list should mark whether an objective has been completed.

Note that the information screen design may be reused here. Also, note that it may be possible to load the map and initial active data when the briefing is being read.

4.1.7 Debriefing Screen

The main purpose of the debriefing screen is to show the statistics of the completed mission. It could include a brief remark on the historical significance of the mission's event. A table should be present that lists the teams (affiliation groups) with their name and color and these statistics:

1. Deaths to total characters ratio
2. Total experience point increase
3. Total inventory items found
4. A summation of the three above statistics as a full total
5. Total mission time
6. Averages of the final experience and health values could be included too

4.2 In-Game GUI

The in-game GUI is the interface in which missions and cutscenes take place. Moving the cursor to the side of the screen scrolls the map.



Figure 4.2.A The Layout of the In-Game GUI. Buttons will appear as icons in the final version.

4.2.1 Controls and Displays

If necessary, tool tips should display to the side of some controls when it is hovered over to provide a better description.

Character Information

The character information box is shown whenever the cursor hovers over a character. It displays the character's health as a percentage, the experience points, the character's name, the current action name of the character, and any special states that the character. The background should be tinted according to the character's team.

Inventory

Inventory: Pressing on this button will cause a popup menu to be displayed on the right side of the screen. The player can select an inventory item to apply to a character.

Character Controls

Action List: This popup menu can be displayed whenever at least one character is selected and the right mouse button is pressed. Its background should be the same as the color of the character's team.

Independence: Sets one of the four independence settings for the character from no independence to full independence. It is disabled if no characters are selected.

Path Controls

Set Path: Press to program a path (using a CharacterRoutine). Each map press will make add a point to the path. When a path point is right-clicked, it displays the action menu of the selected characters. If an action is selected from this list, then that action will be placed in the routine for that point. Depress the set path button to have the character start to follow the path. If the characters selected already are following a path, then pressing this button will add to the current path.

Path Loop: This is only needed during path setup. If depressed, it makes the path loop from the first point to the last point.

Path Index: This takes the place of the zoom level indicator when a path is being made. It shows the current path index.

Next Point/Previous Point: Takes the place of the zoom in/out buttons during path setup. Allows for iteration through the path points. This may help when more than one path point resides at the same tile.

Visual Controls

Zoom Level: Marks the current zoom level. One for farthest zoom, 3 for highest zoom.

Zoom In/Zoom Out: Control map zoom level.

Camera Follow/Go to Hotspot: If there is one character selected, it allows the player to have the camera follow that particular character until the player manually scrolls the screen or depresses the camera follow button. If there is a multiple selection or no selection, then this button has a different appearance. It can then be used to center the screen on wherever the most action is happening. The most action is determined by the highest region character count.

Transition Controls

Load/Save: Load or save the current game.

Briefing: Allows the player to view the briefing again and to review which objectives still need to be completed. It is disabled for cutscenes.

Main Menu: Shows the main menu with a list of available missions and cutscenes. The player can restart the current mission or start a previously completed mission.

Continue: Allows the player to continue to the next mission or to the next phase of the mission. For instance, some missions span multiple days, so this allows the next day to begin. It is disabled until the mission (or mission segment) is complete. When a new mission or cutscene is available, a message will display on the bottom of the screen, and this button will be enabled.

The status bar displays a status message such as button rollover help, status, objective completion notification, etc. If necessary, it could be used as a text-entry box. The mission time is shown to let the player know how much time has elapsed for timed missions. This should be formatted as mmm:ss, minutes are allowed to be more than two digits. Two counters are present, but they do not need to be used. They can have small indicator icons set so that the player knows what the count represents. The counters would be used in a population-counting or food gathering mission.

4.2.2 Indicator Overlays

Indicator overlays allow the player to know the current character selection and path points. For each selected character in a mission, a white square selection box with corners only should be placed around the character. A small indicator bar in red/orange/green should display health level percent over the character. A small indicator bar for experience percentage should display in navy/blue/light blue below the character. However, cutscenes never show any selection indicators on the map. The multiple selection box drawn by the player should be made of white lines.

Path points are displayed as glowing circles. The currently selected point should be red while older points should be a blue color. If possible, a white line should connect the points. The current index of the path point should be shown on it (1-based).

4.2.3 Speech & Message Display

When a character speaks, a panel should display at the bottom of the screen and the currently speaking character should be selected. It shows a portrait of the character in the upper left corner with the speaker's name. White text should appear on a partially transparent reddish background (if possible). If voice files are present, they should be played at the beginning. A player should be able to have the option of either having the dialog continue automatically or to press a button to continue. (Possibly the number of characters in the speech text could be used to generate a delay.) The previously selected character(s) should be selected in missions once the speech is complete.

If the player must answer a question, a dialog box similar to the one used for speech should display a question followed by a list of buttons. Answers should be displayed to the right of their corresponding button. The dialog box should disappear once the player selects an answer.

5 Game Play

5.1 Transitions

The game will progress in a linear fashion. The game will notify the player at the bottom of the screen when a cut-scene or mission is available. The player will then be able to choose when to play the cut-scene or mission through the “missions” button. The game will not advance until the player watches the cut-scenes or completes the missions. When the game is waiting for the player to do this, the player will be able to scroll around the current camp and watch the Israelites go about their daily business. New cut-scenes or missions will be available after the player finishes the current cut-scene or mission.

5.2 Input

The game should be playable using only the mouse. The keyboard should only be used when text is needed from the player (i.e. saving and loading games). The keyboard may be used for some shortcuts during the game, but this is still optional. (All input abilities should be controlled by means of the current screen situation so modes of input should be used in the implementation.)

<i>Key(s)</i>	<i>Use</i>
Esc	Exit the game (display a confirmation)
F2	Save/load game
Pause/Break	Pause game
-	Zoom out
+	Zoom in
Ctrl + (1-5)	Set up group 1-5 using the current selection
1-5	Select a group 1-5
Shift + Mouse 1	Additive character selection

Table 5.2.A In-Game Input Keys

5.3 Multimedia

5.3.1 Graphics

Graphics are one of the most important parts of a game. The artwork should look as realistic as possible within the limits of artists' abilities. It should give the feel of an ancient Hebrew desert journey. The graphics are also very critical in terms of representation. Historically accurate attire, structures, landscape, and vegetation should be used. This means that the artists should be well educated in ancient Hebrew and Egyptian culture. The artwork should be 3D isometric rendered to a 2D plane. The graphics can be low-resolution in general. If possible, resolutions higher than 640x480 should be supported (out-of-game GUI could be enlarged).

Reminders

1. Aaron was 123 years old 14 years after leaving Egypt (Numbers 33:38).
2. Single-hump camels are the only types of camels found in the Sinai region.
3. Museums, bookstores, and National Geographic magazine are all possible sources of art information.
4. Since the Hebrews spent 400 years as slaves in Egypt, they probably wore Egyptian clothing for at least the beginning of their journey. Fortunately, ancient Egyptian culture studies are in great quantity.
5. The artwork must give the correct perspective (perhaps 30 degrees from the top at an angle facing towards the upper left). Using mixed perspective would be very odd.
6. Possibly try a blue screen or magenta screen to film character actions. Elmbrook Church has a blue screen behind the sanctuary.
7. Blender 3D could be used to make the buildings and objects.
8. Characters must be about 32x32 pixels in size, larger characters may not be drawn when at the sides of the screen otherwise.
9. Since vegetation only need to be seen from one direction, do not spend the time designing it in 3D unless that has some noticeable advantages.
10. Bitmap file dimensions can only be in powers of two and the image should be square (although non-square bitmaps do work on most/all modern video cards). Magenta should be used for transparency and one pixel cyan dividing lines should be used to separate each frame of an animation.
11. Look at Exodus_Notes.doc.

5.3.1.1 Weather & Environmental Effects

The arid desert of the Exodus should be felt in the game. Environmental dangers like falling rocks (on Mt. Sinai), heat, drowning, insects (could cause sickness), and animals are all things that the player should need to deal with. Total darkness should also be present in some missions such as the Canaan spy mission. Characters should be able to have torches for missions with total darkness. The screen could be tinted in darkness and a light transparent circle could be drawn for a torch glow afterwards.

Another advantage to the use of regions is the ability to have shrouded regions. The use of shrouded regions will add a sense of mystery to the map. A region where Israelites are present should be completely visible, but regions that have not been visited by Israelites should be shrouded in specked darkness or be blurred to prevent knowledge of what is present. (Note that difficulty of the game could be a factor here: easy: no shroud, medium: shroud until visit, hard: shroud recovers after region is exited). Note that region groups should be used to open more than one region at a time. A scan of three tiles ahead of a character could be used to open an adjacent region. Since each region group keeps track of the number of Israelites in it, there is no problem knowing when it should be revealed.

5.3.2 Audio

Audio should be used whenever it would realistically occur. The main times when audio should be heard are upon character selection (the character says hello in some way), upon command (the character responds to the command), and upon action events. It could be used in the user interface as well. If dialogue speech is available, it should be used when the caption is shown at the bottom of the screen. (Remember not to allow more than one speech sound to play

at the same time.) All sounds that occur within the map should use a 3D effect to simulate distance. (Use the BASS library for the 3D effect and audio features.)

5.3.3 Music

Music is a very important element of game play since it sets the mood. The music should be an orchestral composition that fits the scene and the time. It should include elements of Hebrew, Arabic and Egyptian music. It should also have a recognizable set of themes—a theme for the Israelites, a theme for the enemy, and a theme for God. Music should play throughout the whole game and should switch on demand in cutscenes, but at random in missions. Possibly, it could be lowered in volume automatically during dialogue.

5.3.4 Cursor

The cursor should change to indicate what the player's click or drag will do. When there is a character selection and the map is under the cursor, the cursor should be like four inward arrows to indicate that the character will move to that location. If the cursor is over anything that can be clicked, it should turn into a hand to indicate activation ability. Special actions like "pick up" and "put down" could use a different cursor when it is over a character that can be the target. (Unless those actions always assume that the target is in the tile directly in front of the character.)

5.4 The Camera

The camera is the perspective of the player. A camera acts like a character but its current position represents the center point of the player's view and it does not have any appearance. A GameCharacter object ("The Player") should be used for the camera and it should be able to be given a CharacterRoutine object to tell it what to do. The duration of the action provides a way to tell the speed of a zoon, fade, etc. Any dialogue is regarded as being spoken by the narrator and they should act the same as for character. Setting a negative x coordinate causes no movement to occur and setting a negative y coordinate allows for simultaneous actions. (This enables zooming while fading for example.) The cameras actions are as follows:

<i>ID(s)</i>	<i>Description</i>
0	Wait, do nothing different
2	Stop current action after duration
10	Center on given position
12	Center on place of action (determined by AI and region)
14	Center on selected character
16	Follow the given character (set by a script, character ID in a certain variable)
20	Center on random position
50	Pan camera by position x, y
51	Pan camera by position -x, y
52	Pan camera by position x, -y
53	Pan camera by position -x, -y
1000	Zoom to 0%
1050	Zoom to 50%
1100	Zoom to 100%

2050	Zoom in by 50%
3050	Zoom out by 50%
6000	Fade in from black
6002	Fade in from white
6500	Fade in from given color (set by a script, color in a certain variable)
7000	Fade out to black
7002	Fade out to white
7500	Fade out to given color (set by a script, color in a certain variable)
8050	Rotate clockwise 50%
9050	Rotate counter-clockwise 50%
10000	Tint screen to given color (set by a script, color in a certain variable)
20050	Earthquake 50%

Table 5.4.A Camera Actions. Gray actions do not need to be implemented for Exodus.

5.5 Missions

5.5.1 Purpose and Operation

Missions are the in-game scenarios where the player can interactively engage in the events. The player is to be given a set of objectives to complete after reviewing the mission briefing and any relevant cutscenes. The player must complete the objectives in order to continue to the next mission. Failure in any objectives requires that the player replay the mission entirely or from a saved game.

5.6 Cutscenes

5.6.1 Purpose

Cut-scenes will be used in Exodus to share major parts of the story of the Israelites traveling from the Red Sea to the Promised Land. The player will only interact with cutscenes by advancing text displayed on the screen.

5.6.2 Overview of Implementation

Cut-scenes will be made up of many scenes. Cut-scenes will progress linearly until all the scenes are completed, and thus the cut-scene is over.

Scenes are made up of one or more of the following events:

1. Character Dialogue: Text is displayed in the dialog box and the voice file associated with this dialog is displayed. This event is ended when either the voice file is done playing or the player tells Exodus to continue.
2. Character Movement: List of characters are moved to new locations. When all the characters have moved to their new locations, the event is ended. Characters can either move progressively (as in the player sees them move) or absolutely (as in the character is

placed at that position) This will be helpful when we need to load a new map or character in the middle of a cut-scene.

3. Character Action: List of characters are given new actions (animations) to be played. When all the characters have finished playing their actions, the event is ended.
4. Camera Movement: The camera is given a new location and a speed. When the camera reaches the new location, the event ends.

There will be an indicator in the current scene as to which event must be finished before the scene is finished. For example, we may want the current scene to end when the camera moves to a new location, but we also want characters to be walking while the camera is moving. So we set the indicator to the Camera Movement scene, and the scene will wait until the Camera Movement event is finished. The scene does not care how far along the Character Movement is. (The indicator could simply be a “char” with a number telling which event to wait for to finish the scene)

5.6.3 Script Usage in Cut-Scenes

A cut-scene will be stored as one long script. The script will contain a list of scenes and the events in each scene. The script will be loaded in at the start of the cut-scene, and the data will be stored in a cut-scene class. The cut-scene class will then execute the cut-scene from start to finish.

The following functions will be used in the script to create the cut-scene:

1. `SetScene(int sceneNumber)` : Sets which scene the following data will be used in.
2. `AddDialogue(TextString dialogue, TextString voiceFile)` : Adds a dialogue event to the current scene. Dialogue is the text to display on the screen, voiceFile is the audio file that will play during the dialogue.
3. `AddCharacterMovement(MEDIUM_ID charID, PRECI_PIXELS newXPos, PRECI_PIXELS newYPos)` : Adds a character movement to the current scene. CharID is the character ID to move, newXPos and newYPos are the new coordinates to move the character to.
4. `AddCharacterAction(MEDIUM_ID charID, MEDIUM_ID charActionID, char repeat)` : Adds a character action to the current scene. CharID is the character ID to apply the action to, charActionID is the action ID to apply, repeat is the # of times to play the action.
5. `AddCameraMovement(PRECI_PIXELS newXPos, PRECI_PIXELS newYPos, PRECI_PIXELS speed)` : Adds a camera movement to the current scene. newXPos and newYPos are the new coordinates to move the camera to, speed is the speed to move the camera at.
6. `SetIndicator(char ID)` : Sets which event should end the scene. ID can be a number (1-4) which identifies the event. (1: Dialogue, 2: Character Movement, 3: Character Action, 4: Camera Movement)

6 AI & Scripts

6.1 Artificial Intelligence

6.1.1 Character Routines

Character routines provide a foundational AI component of cutscenes and some mission events. They are like paths for the character to follow, but they also include the ability to give the character action commands and to run scripts at any checkpoint. Character routines can be programmed graphically by placing points on the map and then assigning any special event to that position. A CharacterRoutine is defined as follows.

CharacterRoutine

Repeat Count: The number of times to run the whole routine, or zero for infinity.

List of Events: The list of events along the path.

CharRoutineEvent

Event ID: The ID of the event.

Position: The coordinate position on the map at which this event should occur.

Action ID: Optional. The action that the character should at the given location.

Duration: The amount of time that the character should perform the action for—this cannot be less than the animation time if it is a sessile action.

Script Function: Optional. The name of a script function that should be triggered when this event starts.

Speech Text: Optional. Dialogue text that should be displayed at this event.

Speech Audio: Optional. Dialogue text that should be heard at this event.

Sync Character ID: Optional. The ID of a character that is used for synchronization with the character that has this event. Its primary purpose is to prevent one character from doing something before another has been able to get to a certain location or to finish a certain action.

Sync Character Routine ID: The CharacterRoutine event ID that the sync character must be currently at for the next event to occur.

6.1.2 Movement Algorithms

Efficient movement of a character or some other object is essential to fast and reliable performance. Common movement tasks involve following paths, following other characters, simple linear movement to a near location, movement around an obstacle, movement to allow another character to pass, and knowing when to stop trying to reach an unreachable zone. The AI of Exodus should be able to handle all of these very well.

Maps are arranged as two-dimensional tile arrays with tiles being 32x32. No two characters can occupy the same tile at once unless one character is dead or flying. The elevation of the map determines how fast a character can move over the terrain and if it is even possible.

6.1.2.1 Basic Character Destination Stack & Followed Character Pointer

Movement of a character (or MobileMapObject) is always made in a linear fashion toward the location that was last pushed on to the movement destination stack (`movePath`). There is also a “followed character pointer,” which can be used to tell the AI to have the character always move toward another character. A character should stop moving once there are no more destinations in the stack and a special followed character pointer is not set. A destination is removed once the character has the same position as it. Positions should be stored as `PRECI_PIXELS` (Precision Pixels). The followed character pointer should only be set to `null` upon a special action by a script or by the user, but it could also be changed if there is a change in action.

<i>Index</i>	<i>Description</i>
0	Final destination (Required for movement.)
1	Regional goal destination (Not present if in destination region.)
2	Long-term path finding destination
3	Short-term path finding destination
4-n	Any other short-term destinations

Table 6.1.2.1.A The Destinations Stack

6.1.2.2 Goals

The movement of characters over the map should satisfy these goals:

1. A character should be able to be told to move to any location on the map. The character should be able to move in eight directions (four diagonal, four straight).
2. If a character cannot reach a destination, the character should stop as close as possible to the destination and should not move more than necessary to get close.
3. Movement to a very distant location should be handled intelligently. The character should not take a very wrong route only to have to turn around completely. (However, the experience of the character should affect this.)
4. Movement over a short distance should have the character pick one of the shortest and quickest routes, but it should also be random in some extent. A character should be able to move around small objects without always picking one side unless factors strongly favor that side.
5. If a character must pass other characters, the AI should cause the other characters to move out of the way if they happen to be stationary. Care should be taken to check that the characters has a speed greater than zero to prevent AI problems. A character should stop if there is no way to get through.
6. Characters should move slower when moving from a tile of lower elevation to one of higher and slightly faster when moving from a tile of high elevation to one of lower elevation. Speed should always be kept above zero for these changes.

6.1.2.3 Regional Movement Instructions (RMI)

The regional movement instruction system should be used to guide a character around large terrain obstacles. The RMI system is designed to allow the AI processor to work less because long-distance path finding is done at map editing time rather than during game time.

Each part of the map is designated to a region. These map regions should have no major obstacles in them. Rather, regions should normally border large obstacles (like a line of cliffs) and every part of the map must be part of a region. The AI uses these regions to direct a character around the map if the character's final destination is not in the same region as the character.

The regions should be editable with the map editor of Exodus, but possibly an algorithm could be developed to automatically find the best regions and destination positions.

The AI uses the regions like so:

1. If a character is given a command to move to the other side of the map (and large obstacles prevent a direct linear route), then the region lookup table should be used. The lookup table is always square and it matches the region of the character's current location with the region of the character's destination location. It may look like this if the map has five regions:

	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
<i>0</i>	(0, 1)	(23, 53)	(90, 12)	(32, 6)	(23, 10)
<i>1</i>	(4, 45)	(1, 1)	(93, 3)	(7, 5)	(3, 56)
<i>2</i>	(43, 12)	(23, 21)	(4, 2)	(34, 23)	(34, 53)
<i>3</i>	(56, 19)	(23, 43)	(90, 9)	(5, 3)	(43, 6)
<i>4</i>	(87, 5)	(67, 55)	(12, 54)	(66, 45)	(1, 3)

Table 6.1.2.3.A The Inter-Regional Target Location Table. The current region is on the x axis, and the destination region is on the y axis. Note that the coordinates should be PRECI_PIXELS, so they are actually floating-point.

2. The location given for the match of the two regions should be placed onto the character's movement stack. This location should be chosen to direct the character either directly to the destination region or to divert the character to another region that indirectly or directly links to the destination region. The character will then head toward that location.
3. Whenever the character moves to a new region, the lookup table should be used again and the current region destination should be replaced with the one found in the lookup table. By doing this, it allows the character's path to be freer. The destination location given in the lookup table is only used to direct the character to a new region; it is not a requirement for the character actually to be at the destination location.
4. This process continues until the character is in the destination region.

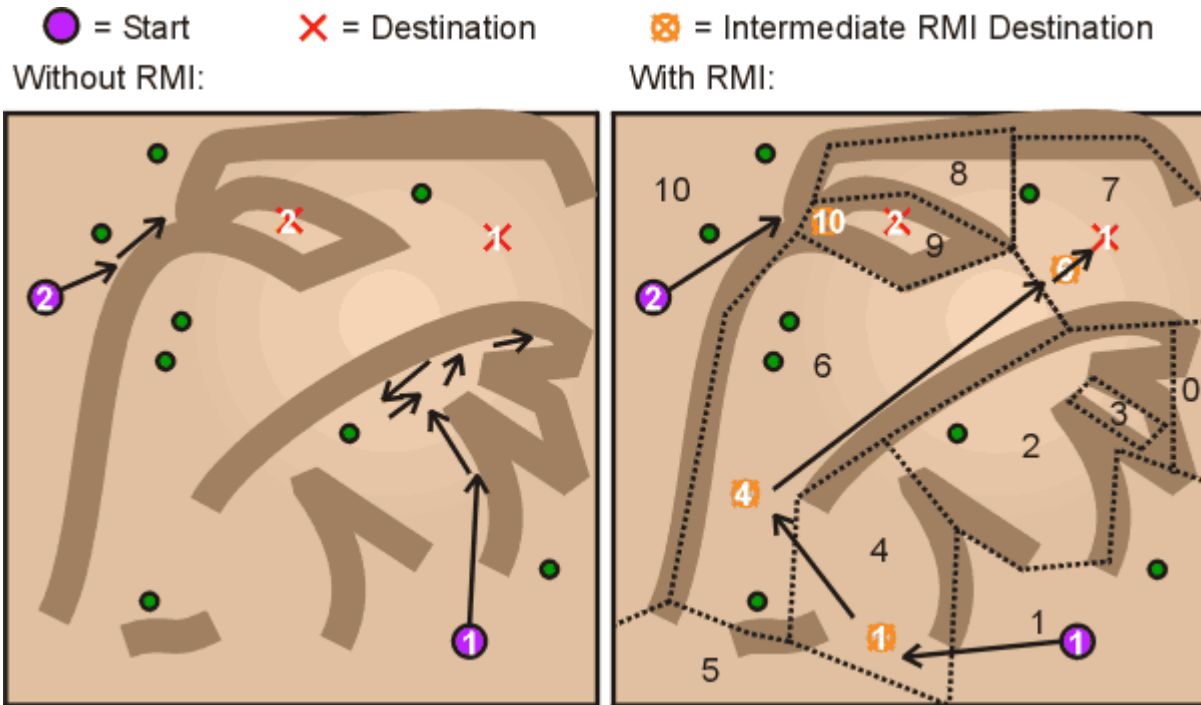


Figure 6.1.2.3.A RMI Diagrams

Inter-region coordinate tables should have its diagonal defined in as such:

- The x coordinate represents region impassability (zero) or a positive value to represent that it can be crossed easily. Higher values could represent better traversal.
- The y coordinate represents the region group to which this region belongs. (This must be cast to an integral value.) It can be used to locate the other regions in the region group quickly. A negative value means that the region is not in a group.

6.1.2.4 Short-Distance Path Finding

Short-distance path finding should be used to guide the character around small obstacles such as other characters, vegetation, and rough patches of terrain. The AI should scan the tiles toward the character's destination by the character's sight range, or at least two tiles. It should also scan the peripheral tiles directly in front of the character. If the path appears to be clear, then the character should turn as necessary and move forward linearly. If, however, there is an obstacle in the path, then more scanning should be done around the character. The scanning should always be done at the locations that are toward the destination first. The scanning should switch the opposite direction only if no good path can be found toward the destination. The scan should radiate outward from the character once an open adjacent tile is found.

	D		
		5	
	3	1	4
9	2	C	
7	6	5	
	8		

	D		
	9	8	
	6	5	
7	4	3	
	2	1	
		C	

Figure 6.1.2.4.A Possible path finding tile choices. D is the destination, C is the character. Yellow marks cumbersome terrain. Green shows the selected path.

In order to implement the scan, a set of diagonal offset and straight offset arrays could be iterated through in order to ensure that all of the tiles around the area are scanned in a radial fashion without any repeat. A tile is scanned by a function that takes tile elevation, distance, speed and the location of other characters into account. It should return a heuristic value that is negative for impossible movement and greater in value for a better movement choice. At least four tiles should be sampled in the direction that the character could move. The heuristics should be compared and a direction should be generated from them. This is similar to the A* algorithm. If the character's speed allows the character to reach the exact destination location, then the character's location should simply be set to the destination location.

6.1.2.5 Unreachable Zones and Halting

Halting of a character can be done in two ways and it is the processes of knowing when a character should stop trying to reach a destination. The first way is for short-distance halting. When the character is within six tiles of the destination, a full path should be scanned to it using A*. If no path can be found, then the character should halt. In order to determine when a character is within a short distance of the destination efficiently, a simplified distance formula may be used. The simplified formula is $(x_2 - x_1)^2 + (y_2 - y_1)^2 < 72$. This does not tell the actual distance, but value less than 72 is within 6 tiles of the destination.

The second method involves unreachable regions. If there is a large region that cannot be entered, then it should be marked as "impassable" by setting its diagonal x coordinate to be zero in the inter-region movement table. The region table should be programmed to allow the character to move to a close as possible to the destination. The character should halt as close as possible to the unreachable region.

6.1.2.6 Character Volume Sphere

Most characters need a volume to keep other characters from getting closer than what looks realistically possible. A basic form of clipping should be implemented to prevent a character from getting too close. Each character should be given a distance at which another character is not allowed to enter. This distance may be anywhere from 0 to about 32 pixels. A character must check if there is another character directly in front before moving forward. If there is a character, a distance check should be done to verify that the next position is not going to enter the volume sphere of the adjacent character.

6.1.3 Character Relations and Battle AI

The character AI is determined by a number of factors. The most common factors include the character's affiliation ("what side" the character is on), the morale of the affiliation group (team), the experience of the character, the health of the character, and the actions that the player can perform.

6.1.3.1 Affiliations and Teams

Character affiliations are used to determine which group of characters a particular character is present as a member. The setup for character relations allows for precise control over how one character should react to another. The primary means of controlling affiliation is with a byte of information stored in a `GameCharacter`. Each active bit of the affiliation byte represents an affiliation with other characters that have that bit active also. (Therefore, there are 256 different possible affiliation configurations.) In order to know if a character is friendly with another, the two affiliation bytes should be ANDed together. If this results in 0 as the result, then they are enemies. If the value is positive, then they are affiliated with one another in at least one way. A matching affiliation means that the characters are on the exact same team.

Example of affiliation matching:

```
0010 0010 & 0000 0010 = 0000 0010 = Friends
0000 0001 & 0010 0100 = 0000 0000 = Enemies
```

In order to match a particular group of affiliations with the `TeamStatus` array, the affiliation should be ANDed with a `0x1` for two teams, `0x3` for four teams, `0x7` for eight teams, or more depending on the number of teams. (Note that is AND operation is equivalent to the modulo of 2, 4, or 8.) The affiliation should then be able to match precisely with a certain team. The typical format is as follows:

<i>Affiliation Bits</i>	<i>Description</i>	<i>Team when 2</i>	<i>Team when 4</i>	<i>Team when 8</i>
0000 0000	Complete hostile (even with its own team)	0	0	0
0000 0001	Player's affiliation	1	1	1
0000 0010	Enemy's affiliation	0	2	2
0000 0111	Neutral's affiliation	1	3	7
0000 1000	Extra affiliation, enemy even to neutrals	0	0	0
0001 0000	Animal type 1's affiliation	0	0	0
0010 0000	Animal type 2's affiliation	0	0	0
0100 1111	Animal type 3's affiliation	1	3	7
1111 1111	Complete friend	1	3	7

Table 6.1.3.1.A Typical Affiliations. Note that the "Team when #" means that is the team number when there are # teams. Normally four teams will be used.

6.1.3.2 Character Groups

Characters may be part of one or more groups. These groups are implemented simply as lists of character pointers with numeric identifiers. A group is used to select a set of characters quickly and in terms of AI, it can be used to work easily with multiple characters. The player may set the first five (1-5) groups, so no AI should use these IDs. The player sets the groups by making a selection and then pressing Ctrl + (1-5). These can be selected quickly by pressing one of these numbers.

6.1.3.3 Sight and Character Searches

While characters move about the map, they must have a way of detecting characters that are within their proximity and sight. A character can only react to characters that it knows from proximity. A character always sees in the direction that the character is facing and therefore can react sooner to objects coming from in front. A character should be given a sight range in tiles. The search for a close character should radiate out from the character when trying to find the best target. The search for other characters should progress as follows:

1. Scan the three tiles directly in front of the character. If more than one character is found, choose the closest character.
2. Scan a square around the character that is half the distance of the total sight. Choose the closest character if possible.
3. Scan a rectangle out from the character's direction. Choose the closest character.

Because of the format of the character data structure, diagonal searches are rather difficult. One possible solution is to request two squares like so:

```

2222
2222
222211
222211
 1111000
 1111000
   00C00
   00000
   00000

```

Figure 6.1.3.3.A Diagonal Search with a Sight of 6 Tiles

6.1.3.4 Independence

Independence is a character property that regulates how a character responds to commands and hostility. In addition to independence levels, a character can be told to wander around the map which, when combined with independence, provides automatic scouting and automatic enemy track-down. The following independence levels are present.

<i>ID</i>	<i>Name</i>	<i>Description</i>
0	Avoid	If an enemy comes near, the character should move away especially if attacked. The character should not use a weapon if he or she has one. The character should try to hide.
1	Guard	The character should stand still and engage an enemy that comes in range, but should not do any pursuit.
2	Harass	The character should engage and pursue the enemy a short distance and then give up.
3	Destroy	The character should engage and pursue an enemy until the enemy is killed. The character should act fearlessly and should fight as much as possible.

Table 6.1.3.4.A Independence Levels

6.2 Scripts

6.2.1 Purpose and Use

Scripts will be used to set up different situations in Exodus. This will involve setting different variables such as what map to load, what dialogue for the character to speak. Scripts will have to be able to access active data and choose different paths based on this information.

6.3 Script Variable Types

The following contains the types of variables the scripts will implement. PLEASE keep this list as small as possible. If a variable type can be created by combining basic variable types, then just use the basic variable types. For example, a Coord2<PIXELS> can be created by having two PIXELS variables. This will reduce the work I need to do.

<i>ID</i>	<i>Type</i>
1	Int
2	Bool
3	Float
4	TextString
5	SMALL ID
6	MEDIUM ID
7	LARGE ID
8	PIXELS
9	PRECI PIXELS

Table 6.3 Script Variable Types

<How about this list for ScriptVar, where grayed values are not implemented:

<i>ID</i>	<i>Type</i>
0	NULL type
10	bool
11	signed char
12	unsigned char/BYTE
13	signed short
14	unsigned short
15	signed int
16	unsigned int
17	signed long
18	unsigned long
19	signed __int64
20	unsigned __int64
21	float
100	void*
110	TextString*

510	Coord<PIXELS>*
511	Coord<PRECI_PIXELS>*

Note that DLL scripts could use their own global variables of any desired type to store data over more than one game cycle. Then we would just need to be sure that the data can be reloaded when the script is reloaded. For instance, it would be more efficient to store a pointer to a GameCharacter rather than to have the system re-search the data structure again (albeit, it would be fast). We could have it get the pointer only if the global variable is `NULL`, otherwise we could reuse it for the whole time. That reduces a whole data structure search into just one `NULL` check on all subsequent runs. This value does not need to be serialized since the script can just ask for the character again if it is reloaded.

Another useful feature we should implement in scripts is a small header that allows us to put a mnemonic on a `void*` so that we do not confuse them if we do not want to include the definition of the class in the file. The `void*` allows us to reference a GameCharacter even though the script does not know about it (but Exodus does). Here is an example:

```
typedef void *GAME_CHARACTER;

GAME_CHARACTER savedCharPtr;
```

Finally, we should decide what we are going to use for the script variables ID type. I would say that using an unsigned int should be good enough, but an actual TextString would work well enough too. If we use an integer, we could use syntax like:

```
x = iex.GetVariable('x');           // Use the ASCII code as a variable ID

// Or we use comments as names: (This one is "fooBar")

iex.SetVariable(2345, someVal);    // 2345 = fooBar
```

If we use a TextString, then it would be more expensive on resources, but not bad. It may be better that way for extensibility.

```
// Must define the name so that it is not reallocated each time we write "x".
TextString X_VAR_NAME = "x";

x = iex.GetVariable(X_VAR_NAME);
>
```

6.4 General Script Functions

<ADD: Add these to an Excel file?>

6.5 Exodus Script Functions

The following will contain all the script functions we will need that are specific to Exodus.

This will help me decide the best way to implement the scripts. We can assume that script functions will contain a name, and a list of variables. All variables will be passed by reference.

6.5.1 *Cut-scene Functions*

`SetScene(int sceneNumber)` : Sets which scene the following data will be used in.

`AddDialogue(TextString dialogue, TextString voiceFile)` : Adds a dialogue event to the current scene. `Dialogue` is the text to display on the screen, `voiceFile` is the audio file that will play during the dialogue.

`AddCharacterMovement(MEDIUM_ID charID, PRECI_PIXELS newXPos, PRECI_PIXELS newYPos)` : Adds a character movement to the current scene. `CharID` is the character ID to move, `newXPos` and `newYPos` are the new coordinates to move the character to.

`AddCharacterAction(MEDIUM_ID charID, MEDIUM_ID charActionID, char repeat)` : Adds a character action to the current scene. `CharID` is the character ID to apply the action to, `charActionID` is the action ID to apply, `repeat` is the # of times to play the action.

`AddCameraMovement(PRECI_PIXELS newXPos, PRECI_PIXELS newYPos, PRECI_PIXELS speed)` : Adds a camera movement to the current scene. `NewXPos` and `newYPos` are the new coordinates to move the camera to, `speed` is the speed to move the camera at.

`SetIndicator(char ID)` : Sets which event should end the scene. `ID` can be a number (1-4) which identifies the event. (1: Dialogue, 2: Character Movement, 3: Character Action, 4: Camera Movement)

<Must have a way to show a small dialog box with a title, a message, and a varying amount of selectable options. The index of the selected option must be returned to the script. This dialog box allows a user to respond to a question.>

<ADD: Add these to an Excel file?>

7 The Game Cycle

The game should be run on a single thread to ensure that nothing goes out of sync. A main loop should progress indefinitely as the game runs. The cycle is responsible for an ordered checking of certain states and input as well as all of the actions that occur during the game. In order to reduce processing requirements and increase speed, not every cycle should do a complete run though all activities.

7.1 Class Interaction

Classes should be designed to be as independent from one another as possible. Exodus should have an object-oriented design to allow for easy, categorized programming. Each class should be able to contribute its own special features to the game [cycle].

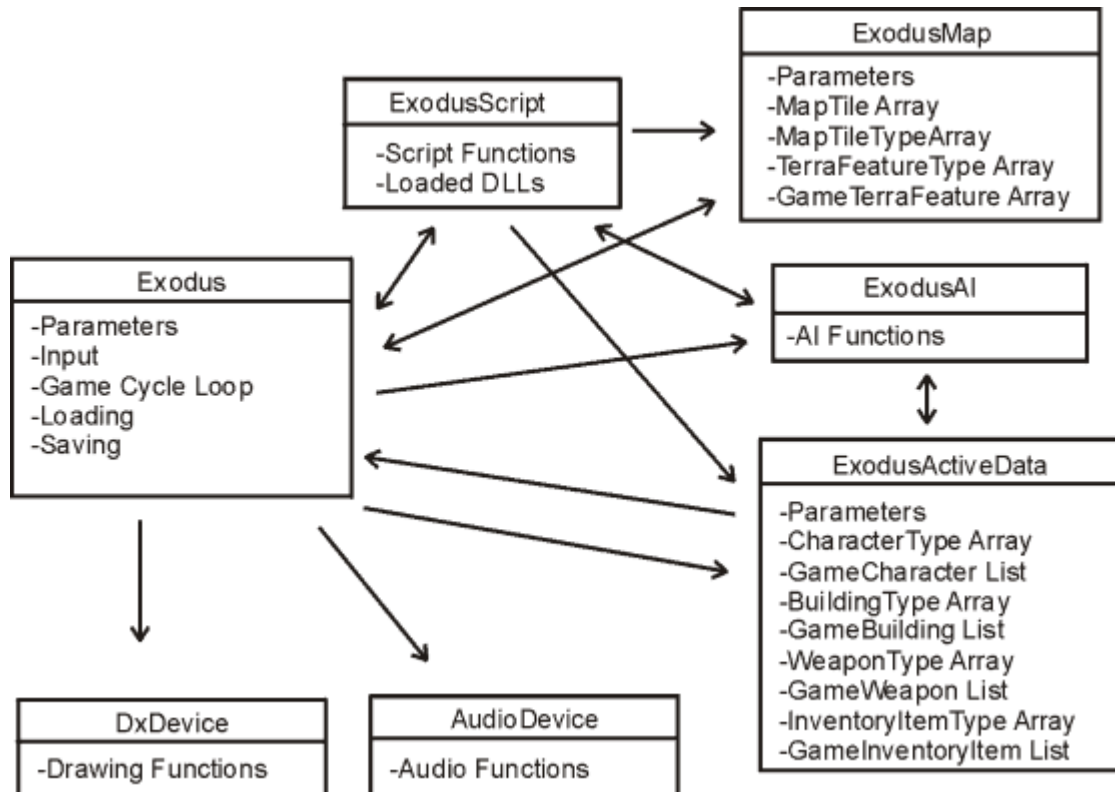


Figure 7.1.A Class Interaction Diagram. Member listings are rough.

7.2 Modes

The three main modes of Exodus are out-of-game mode, in-game mode, and map editor mode.

<ADD>

7.3 Order of Operations

In general, operation should progress in a definite, deterministic order. Synchronization of every game element is essential to proper game play. Map tile and map object ordering should be from the upper left to the bottom right. Progression should move from left to right in rows.

7.3.1 Input Handling

Input handling should be done using the user interface classes. A UIRegion should mark the visible map region and the events that come to that region should be processed further. Input order is very important since it must accurately reflect what is on each layer and what level each layer has. Input should always filter down from the top objects to the bottom objects. For instance, a menu over a character should be activated rather than character selection.

7.3.2 Drawing Order

Drawing order is one of the most complex tasks of the game, but it is also very orderly. Each object on the map must be drawn from bottom to top and from upper left to lower right. If this is not adhered to, then the graphics will not have a valid 3D appearance. One of the more complex parts of drawing is dealing with buildings since characters can enter them. Building drawing requires that all of the characters to the upper left be drawn before an interior building graphic is drawn. User interface should always be drawn last since it is overlaid.

7.3.3 AI & Script Activities

AI and script activities should be able to occur at certain positions throughout the game cycle, but scripts should not be able to run in dangerous sections of partial manipulation. If someone picks up an object, it should be removed off the map and set as such. If someone less tries to pick up that object, it should be prevented since it was already done. Also, remember that performing AI operations on a character sooner may give that character an advantage in a battle.

7.3.4 Complete Order

The complete order of operations is given below.

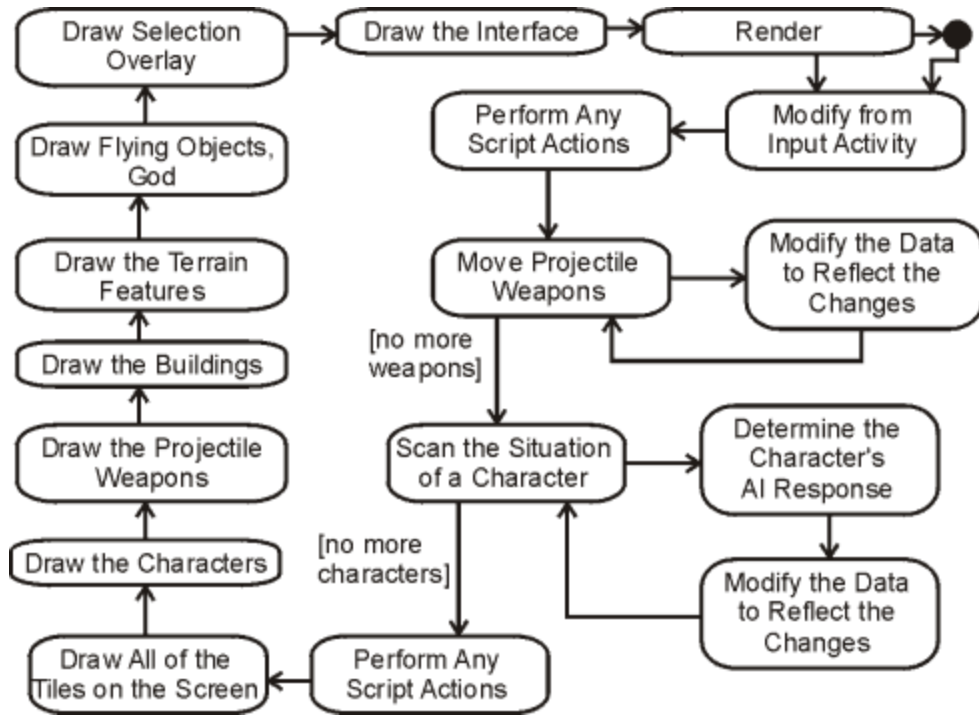


Figure 7.3.4.A A Flow Chart of the Game Cycle Operations

8 Exodus Map/Active Data Editor

The Exodus map and active data editor is a very important component since it there is no way to build an Exodus mission or cutscene without it. It does not need to be used by the user so there should be more focus on function than form. It is an integrated development environment that allows a map to be made in the same place as the active data. It can also be used to engineer a mission or cutscene graphically using a select-and-place technique.

8.1 Modes and Interface

8.1.1 Toolbar

This toolbar is used to launch into the other map editing modes. It can be used to start a new map. It should always be on screen when in map editing mode.

8.1.2 General Parameters Setup

<ADD>

<Set up type files here.>

8.1.3 Map Tile Editing Mode

<Allow for spraying of tiles onto the map.>

<Elevation Painting Mode>

8.1.4 Terrain Feature Editing Mode

<ADD>

8.1.5 Building Editing Mode

<ADD>

8.1.6 Map Region Editing Mode

<ADD>

8.1.7 Character Editing Mode

<ADD>

<This is very complex, may need to be divided.>

8.1.7.1 Path/Character Routine Editing Mode

<ADD>

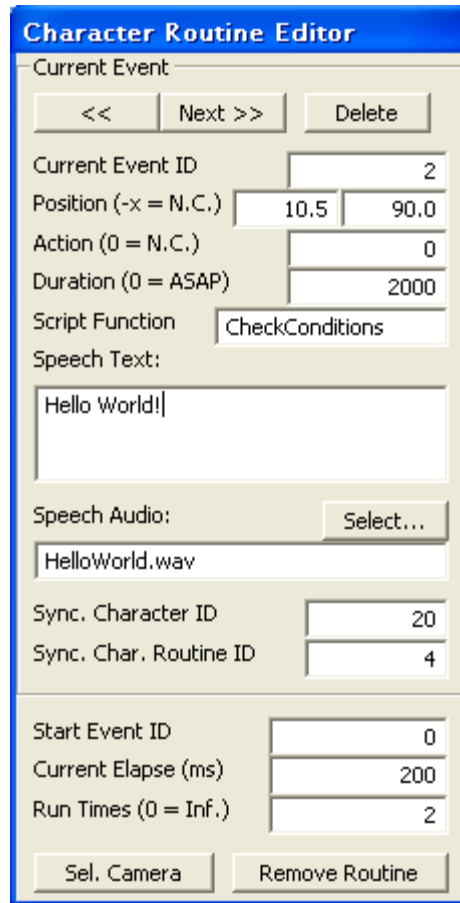


Figure 8.1.7.1.A The Character Routine Editor

8.1.8 Projectile Weapon Editing Mode

<ADD>

8.1.9 Inventory Editing Mode

<ADD>

<Allow for editing of the actual inventory list?>

8.1.10 AI Editing Mode

<ADD>

8.1.11 Console/Text Display

The console text display is used to view debug output. It should also include a way to enter text commands that have been preprogrammed to output status information or to run certain debugging operations. It could be activated using the '~' key.

8.2 Automatic Terrain Generation

Automatic terrain generation could be used to increase reduce the amount of time taken to make a map's terrain features. Possibly the Perlin Noise could be employed to generate the effects. (http://freespace.virgin.net/hugo.elias/models/m_perlin.htm) The parameters of automatic terrain generation should include a randomization seed, the amount of grain, the water level, the amount of mountains, and the amount of vegetation. Possibly allow certain textures to be assigned to certain elevations.

8.3 Map Testing

Maps should be able to be tested from the same location as the map editor.

<ADD>

9 The Exodus Website

The Exodus Website is <http://3d7software.org/exodus/index.php>.

The Exodus fan forum is <http://3d7software.org/forums/viewforum.php?f=13>.

9.1 Purpose

The Exodus Website should be used to post information about the Exodus computer game. As we develop it, the site should include some information about our progress. The site should provide a download for the game once it is complete. Visitors should be able to learn about the Exodus a little as well. It should also provide a way for us to tell others about God, perhaps by means of forum discussions or e-mail.