**Technical Design Document**

**GAM150S14-C**

**Spring 2014**

**Producer:**

Juli Gregg

**Tech Director**:

Dan Muller

**Game Designer**:

Luke Conlon

**Project Manager**:

Kaden Nugent

Table of Contents

[Overview: 2](#_Toc379836564)

[Graphics Implementation: 2](#_Toc379836565)

[Multiplayer Implementation: 3](#_Toc379836566)

[Coding Methods: 3](#_Toc379836567)

[Debugging: 4](#_Toc379836568)

[Tools: 4](#_Toc379836569)

[Scripting Languages: 4](#_Toc379836570)

[Technical Risks: 5](#_Toc379836571)

[Appendices 6](#_Toc379836572)

[Appendix A: Interface Flow 6](#_Toc379836573)

[Flowchart 6](#_Toc379836574)

[Mockups 7](#_Toc379836575)

[Appendix B: Art Requirements 10](#_Toc379836576)

[Appendix C: Audio Requirements 10](#_Toc379836577)

[Appendix D: Technical Guidelines 11](#_Toc379836578)

Overview:

The game runs on a Game State Manager that uses function pointers to call the appropriate functions for each level. The Physics Engine will be global and easily modifiable through a rigid body system that can be added on any object that influences the physics of the world. The sprite structure will be global and creation of sprites are automatically added to the object and memory manager which will handle allocating and freeing the required memory. The AI implementation will be tied to the structure for each type of enemy and boss and will be called on the level's update function to update all enemies.

All object creation will be as automatic as possible to ensure that objects are created correctly, but many of the default values will be able to be utilized to avoid too many hard coded parameters. Simple file I/O will be used to save and load the game and save their character's progress and weapons. Audio will be handled with FMOD using an Audio Manager similar to the Object Manager to ensure we are correctly allocating and freeing memory.

Graphics Implementation:

The graphics are loaded into the Object Manager with a Z-index and drawn on the screen in that order (larger Z-index being drawn last and therefore on top). Backgrounds will be set with a Z-index of 0. The HUD will be drawn with a Z-index starting at 200 so there is enough room for more objects to be drawn in between. The starting Z-index can be increased if the need arises as well.

Animations are created using a tile-map of each frame of animation. The animation order is left to right first, then top to bottom. The sprite creation function has parameters for any number of animation rows or columns. Static sprites are created using one row and column of animation frames. Animation is handled in the sprite drawing function where it will time and calculate when the frame needs to be changed.

Dynamic text will also be available use in the game. This is handled with a font sprite sheet and altering the frame to display the correct letters. Having this will decrease the need for a lot of sprite assets just to create text on the screen. This will also make it easy to change fonts at any time as long as the corresponding sprite sheet is being used.

Multiplayer Implementation:

Multiplayer is currently a stretch goal for the team. However, the plan will be to use a keyboard and mouse for the first player and three additional controllers for the other players. The game will be able to handle up to four players with the game changing slightly based on the number of players in the game. The HUD will expand to include the other player's health, experience and usable items.

Experience will be split between players based on the percentage of damage done to each enemy. The enemies will spawn in greater numbers to account for the increase in players. For boss fights or special enemies, there will be an additional increase in health and additional abilities for group play. Money acquired will be split between all players and items will be given out in a round robin format.

Coding Methods:

The project will be divided into sub-folders based on the type of file they are. All headers will go in the “/HeaderFiles” folder. All C source files will go in the “/SourceFiles” folder. All images and textures will go in the “/TextureFiles” folder. Once sounds and music are added to the project they will be in the “/AudioFiles” folder. The Alpha Engine files are left in the default folder for the project to separate the files that the team creates and the files for the Alpha Engine.

The project will also be broken up into separate filters to organize the files. The first filter will have the same formatting and naming convention as the file structure folders. However inside the project itself there will be additional filters to separate the level headers and files, engine headers and files, object headers and files, as well as team created math headers and files.

All coding conventions including function and variable naming conventions, file header formatting, code block formatting, and special syntax for includes and header files are all located in the Technical Guidelines document. This document is also included in Appendix D. The Technical Guidelines is a living document that will expand as the project expands to account for the changes in a growing project as the need arises. The document is also informal and kept brief so information can be found easily.

Team Sausage Fox will be using a Git source control system as the primary system. There will also be a team SVN created as a backup if the first source control system has problems that the team cannot solve. The SVN will be updated by the technical director at least every week or if there is a large stable build created. The team members are encouraged to push very often. This will mean that the team has many different versions with smaller changes to go back to if there is an issue with the latest build. This is one method of debugging the team has available.

Debugging:

Debugging will be handled for the most part using print statements in a console output. This allows for easy and real time variable tracking very quickly. If more detail is needed to debug then the Visual Studio debugging tool is available to use. The team will be shown how to maximize the tool's effectiveness by utilizing break points and variable watches to step through the code that may be causing errors or unintentional behavior. To find and handle memory leaks the team will use a Visual Studio plug-in called Visual Leak Detector that can be added to the project to find any memory leaks.

The source control system can also be used for debugging to go back to an earlier version to find what changes caused the errors to appear. There is also a debug draw function available for use to see collision boxes. These can be attached and shown by any sprite that is created, and shown even if the parent sprite itself is not visible.

Tools:

The team will use Visual Studio 2012 as its primary project software development tool. All in-team testing will be done using Visual Studio and external testing will have a release standalone version of the game available. New files will also be added through Visual Studio so they are automatically put into the correct filters for the project.

Art tools that will be used are Paint.NET and Gimp 2.0. These are free image editing software that the team can use. The primary format will be a PNG file with transparency. The audio will be run and created using the FMOD software. A memory leak plug-in called Visual Leak Detector will be used for finding memory leaks.

Scripting Languages:

The project will contain no scripting languages.

Technical Risks:

One of the largest technical risks is the internet access at Digipen. Since the team will utilize the source control system for frequent pushing and pulling of the project, a stable internet connection will be mandatory for the most efficient work. A possible mitigation option for this risk is to go to a team member's apartment to use the internet there.

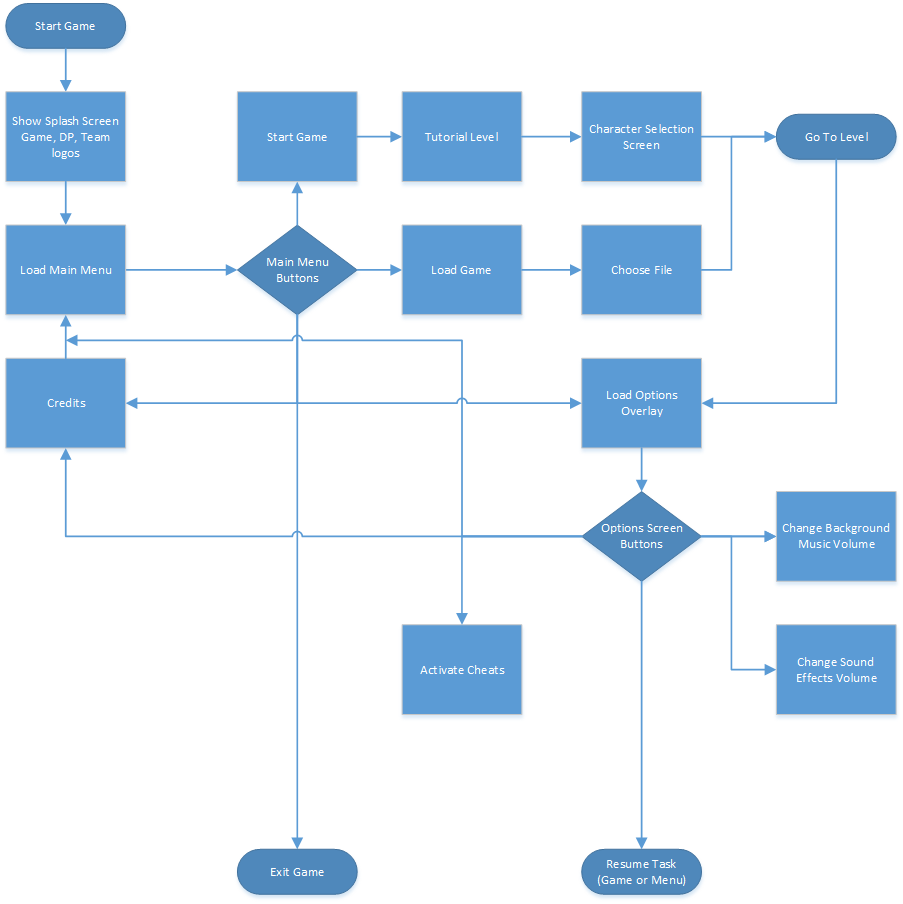
Another technical risk is to stay ahead of all milestones. Since the team does not have experience in creating a full game from scratch, some code may need to be replaced or modified for more efficient usage. To mitigate this risk, the team will keep all code separate from other code. There will not be functions that have multiple effects so all code can be easily interchangeable for other code. Keeping all of the code properly formatted and commented will help mitigate this risk as well.

Each of the team members also have different coding experience which may lead to some members doing more work than others or taking over aspects of the project by finishing their assignments and goals quicker. To mitigate this the team will have multiple avenues for open communication so the more experienced members can help the less experienced members. There will also be a weekly code review so each member can explain the code that they wrote to the rest of the team. This will allow the team as a whole to understand all the code written and other members can offer options to solve a certain problem or optimize some code or algorithm.

Appendices

Appendix A: Interface Flow

Flowchart

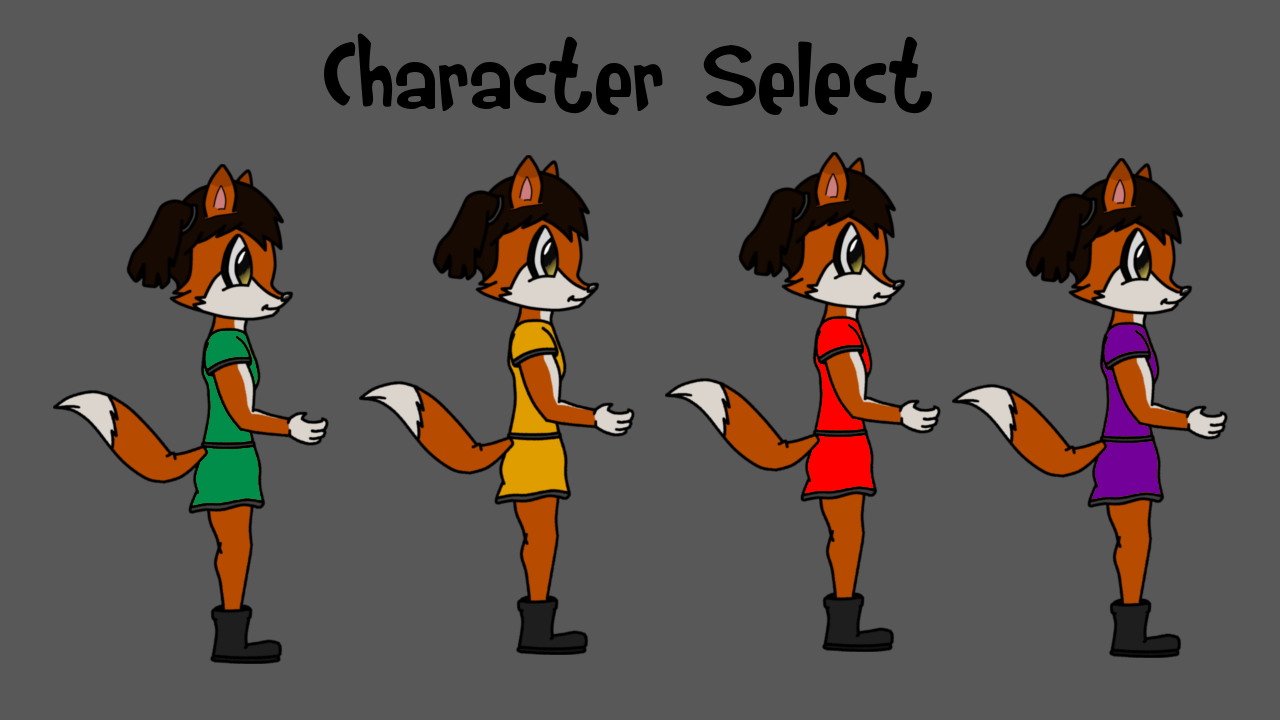


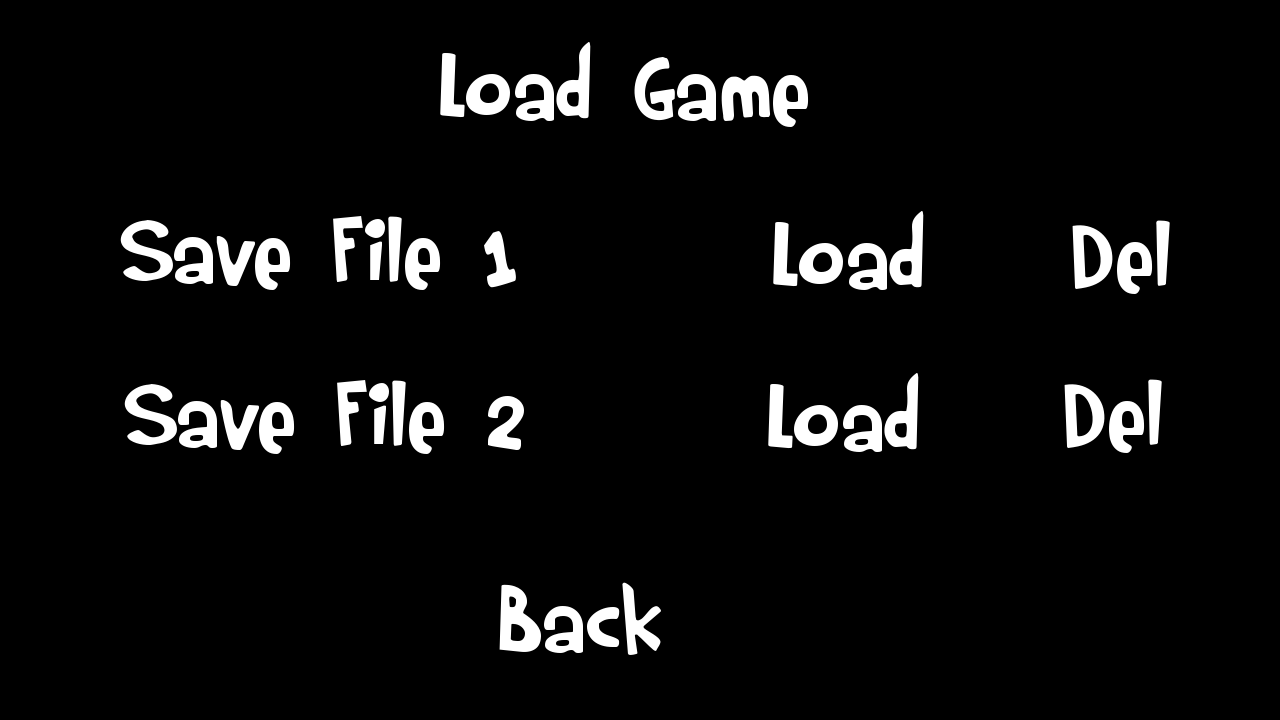
Mockups

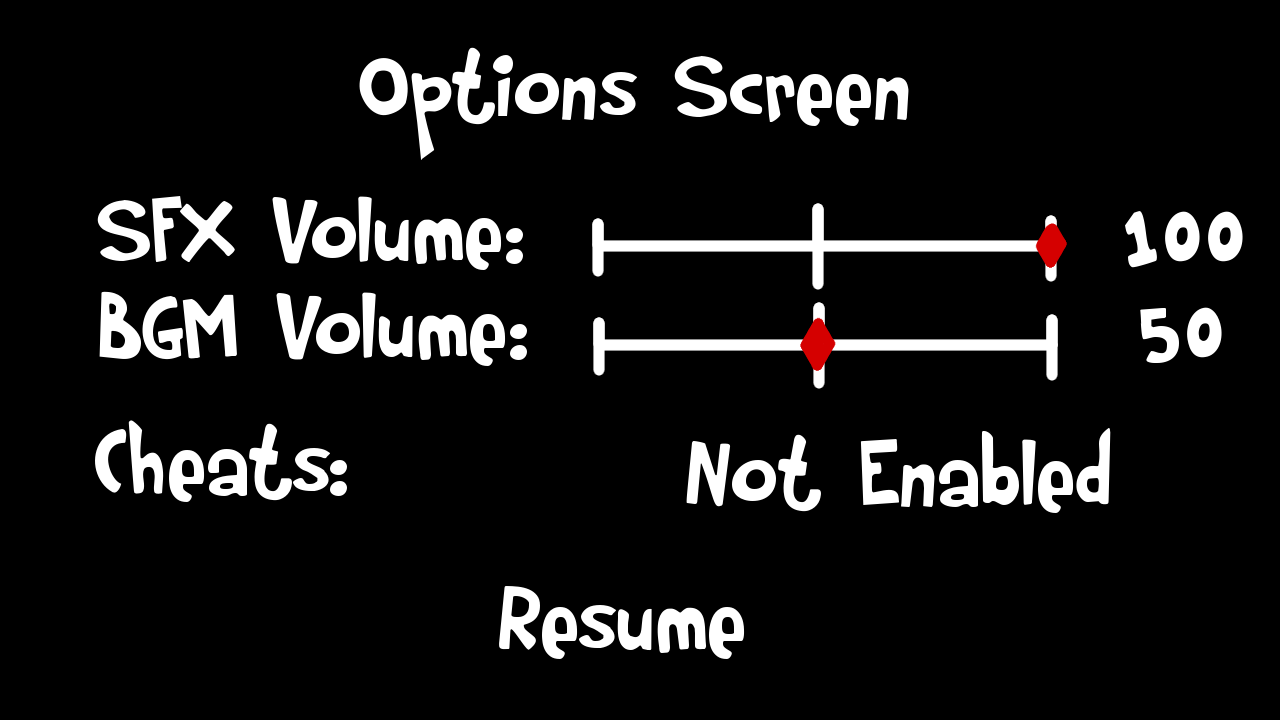
****

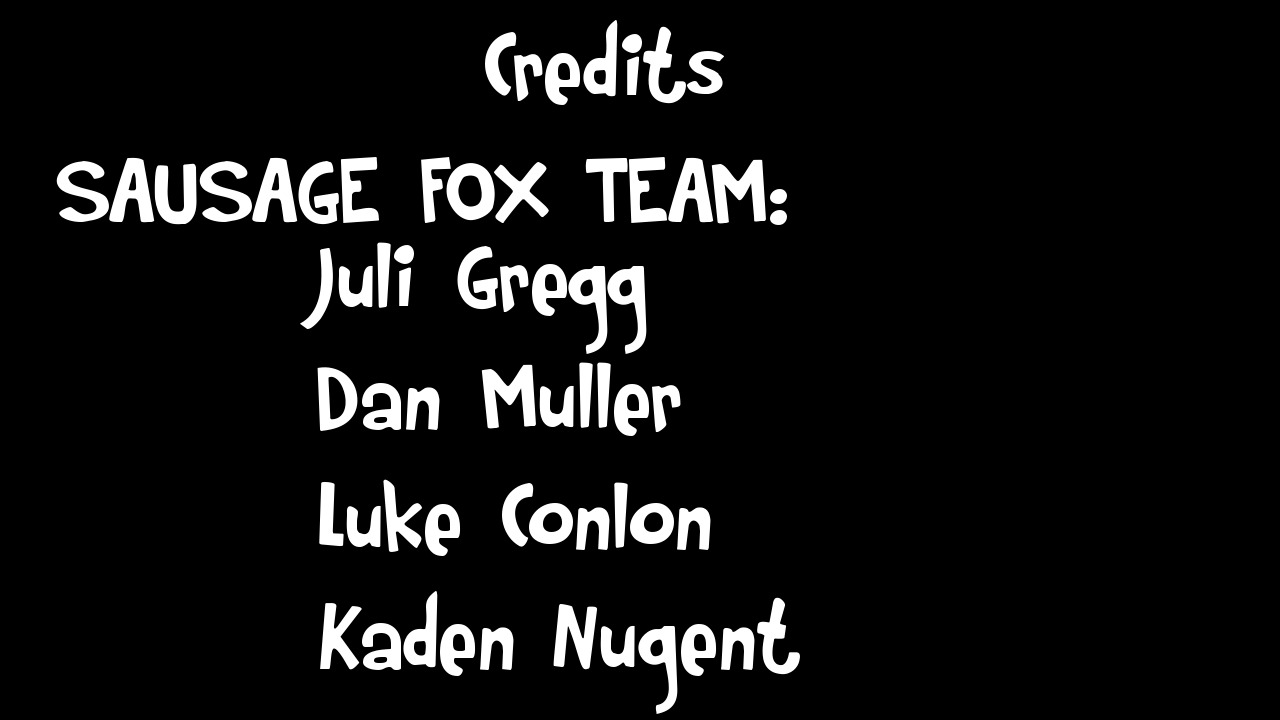
****











Appendix B: Art Requirements

All of the art will be created by the team in either GIMP 2.0 or PAINT.NET. Every file needs to have a specific name that easily differentiates the file from the others. All art assets will be placed in the “TextureFiles” folder and must be in a .png format to preserve the highest quality while supporting transparency. All finalized art assets must start with an upper-case letter, however temporary and placeholder art assets can be named in any manner as long as they are removed once it is not being used anymore.

Appendix C: Audio Requirements

All audio assets will be found either on the Digipen library or created by Jason Uritescu. The voice overs for the characters will be created by the team. The files will be in an .mp3 format so the file size stays low. All audio assets will be placed in the “AudioFiles” folder and final assets will be named with a starting upper-case letter. All audio assets will need to be normalized for consistency in the volume control.

Appendix D: Technical Guidelines

The team also keeps a short document with any and all guidelines to follow for the project. This document is an informal document and has many lists, bullet points, and examples so the information is easy to find and access if needed.

**Technical Guidelines**

**For Team Sausage Fox**

* **Global Functions** - GlobalFunctionName(void)
* **Global Variables** - int GlobalVariableName or int Global\_Variable\_Name
* **Helper Functions** - helperFunctionName(void)
* **Local Variables**  - int localVariableName or int local\_Variable\_Name

**Doxygen format documentation**

**File Header:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

\file Camera.c

\author Juli Gregg (j.gregg)

\date Jan 11, 2014

\brief Camera movement functions

\par Functions:

\li SetCamera

\li ResetCamera

\par

<b> Copyright (C) 2014 DigiPen Institute of Technology.

Reproduction or disclosure of this file or its contents without the prior

written consent of DigiPen Institute of Technology is prohibited. </b>

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Function Header:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

\brief

Sets the camera position

\param playerX

Where the player is located

\param offset

The distance from the edge of the camera the player needs to be

before changing the camera position

\param hudmain

HUD objects to draw

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Code Blocks:**

// ---------------------------------------------------------------------------

// Includes

// ---------------------------------------------------------------------------

// Globals

// ---------------------------------------------------------------------------

// Function Prototypes

// ---------------------------------------------------------------------------

// Main Program

**Header file location:** \MansionMashers\HeaderFiles

**C file location:**  \MansionMashers\SourceFiles

**Texture file location:** \MansionMashers\TextureFiles

**Audio file location:** \MansionMashers\AudioFiles

**Include syntax for header files:** “../HeaderFiles/Header.h”

**Texture syntax for texture files:** “../TextureFiles/Texture.png”

**Header defines:**

#ifndef FOX\_FILENAME

#define FOX\_FILENAME

//Header code goes here

#endif

**Fox Engine Project Filters:**

**Fox Engine Headers:** All engine headers

**Fox Level Headers:** All level headers

**Fox Math Headers:** All math headers

**Fox Level Sources:** Level source files

**Fox Math Sources:** Math source files

**Fox Engine Sources:** Engine source files

**Game Loop Format:**

**Main.c** - Load, Initialize, Loop(Input, Update, Draw), Free, Unload

Each level needs each of the functions so Main.c can call it.

Any helper functions are built off of the 7 main functions.

**Levels**  - Load<Level><Number> or Load<Name> (Ex: LoadMainMenu, LoadLevel1)

Same for Initialize< >, Update< >, Draw< >, Free< >, Unload< >