KRYPTA

Krypta

Technical Manual

Table of Contents

1. Introduction and Information

1.1 – Purpose

The purpose of this technical manual is to elaborate technical aspects and provide an understanding of the workings of the Krypta project and all of its components. This will include design details regarding the game client, the map editor as well as the Krypta 2D Framework.

This document will serve as both a technical document and reference manual for ourselves throughout development and for those who wish to further develop using Krypta 2D. The following will cover the details of the system architecture, the core technologies and operative environment that is used, descriptions of file structuring and design, insight into libraries and interface design and justification for design decisions.

1.2 – Overview – What is Krypta and Krypta 2D?

The Krypta project is comprised of multiple components which will be referenced throughout this manual.

Krypta is an interactive logic-puzzle based game which tests the user's foresight and logic skills against objectives in a fun sandbox environment that aims to appeal to players of different skills and ambitions. The Krypta game is a by-product created from a custom built framework used to display its power and potential.

Krypta 2D is the custom built framework that was created in order to complete the original Secure Dungeon project. The reasoning on our decision to build Krypta 2D will be discussed in section 1.5.

1.3 – Project Context

The Krypta team consists of five team members studying a Bachelor of Computer Science at the University of Wollongong. The team consists of mainly Multimedia and Game Development majors, but also consists of Software Engineering and Mobile Computing majors.

The original Project Specification is shown below.

> **Proposed Title:** Secure Dungeon: A physical security tool.
>
> **Project Description:** The aim would be to design software capable of representing the physical security of a dungeon, or more generally a building. The reason for saying a dungeon is to initially limit the technological resources. The software should be able to assist in the management of security systems. For example, by illustrating the illumination of torches, the presence of pressure plates, or the spread of gas trigged by a tripwire or pressure plate. The security components could be static or dynamic (patrol dog) and a time dependent representation would be helpful.

Krypta was developed to emulate the idea of a "Secure Dungeon" requirement but allowed the group to play to the strengths of the group and fields of study.

Krypta is designed with a large target audience in mind – from young teenagers and up and anyone who generally likes puzzle games and quick logic puzzles. It provides a single player campaign and multi-player community aspects through the sharing of player made maps.

The structure of the Krypta project can be divided into four main sub-systems which will be discussed below.

1.3.1 – The Framework – Krypta 2D

The Krypta 2D Framework is the most important piece of the Krypta project. A software framework is essentially a building platform for an application that is intended to serve as a support or guide for the building of a program.

This is the main focus of what will be created during the project, which will be demonstrated through the use of the Krypta game and its accompanying editor.

1.3.2 – The Game

The Krypta Game Client will be the second most important piece of the Krypta Project. The game client is the piece of software that will connect the user to the game. It will perform as an interface that relays data between the user and the program. The user will be able to select a variety of maps, load in custom maps that have been created by the Map Editor, and play the game using the selected map.

1.3.3. – The Map Editor

The Map Editor will act as an extension for the game itself. The initial game is based on pre-made maps made by the creators of the game for users to play. The purpose of this function is to add a community aspect by allowing users to create their own maps

1.3.4 – The Online Community – Krypta Online

The Krypta Online Community for the Krypta game will allow users to upload their custom built maps for other users to try out their skills. This will also encompass a ratings system as well as a high scores leader board, involving a database that stores all user and map information. The website can be found at: kryptagame.com

1.4 – Project Goals

The original specification of Krypta was to develop software that emulates and demonstrates the aspects of a secure dungeon. As stated previously in *1.3 – Project Context* – we modified the original specifications and proposed to take a Game Development approach to the project.

With this new approach to the specification, we aimed to take it a step further and develop a highly customisable tool that not only shows use of static and dynamic physical security components, but also develop a highly customisable framework that can be deployed in a large variety of environments.

The Krypta game aims to embody the "Secure Dungeon" theme by involving:

- Users traversing dungeons
- Avoiding pressure and time-interval activated traps
- Logic based puzzles
- Static and dynamic path patrol enemies
- Illumination of surroundings through torches and Fog of War.

Aside from achieving the specification related goals, the goal of the CSCI321 project is to provide an introduction and first-hand experience to the Software Development process through the complete documentation, design, team-based development and deployment of Krypta.

1.5 – Why develop a framework?

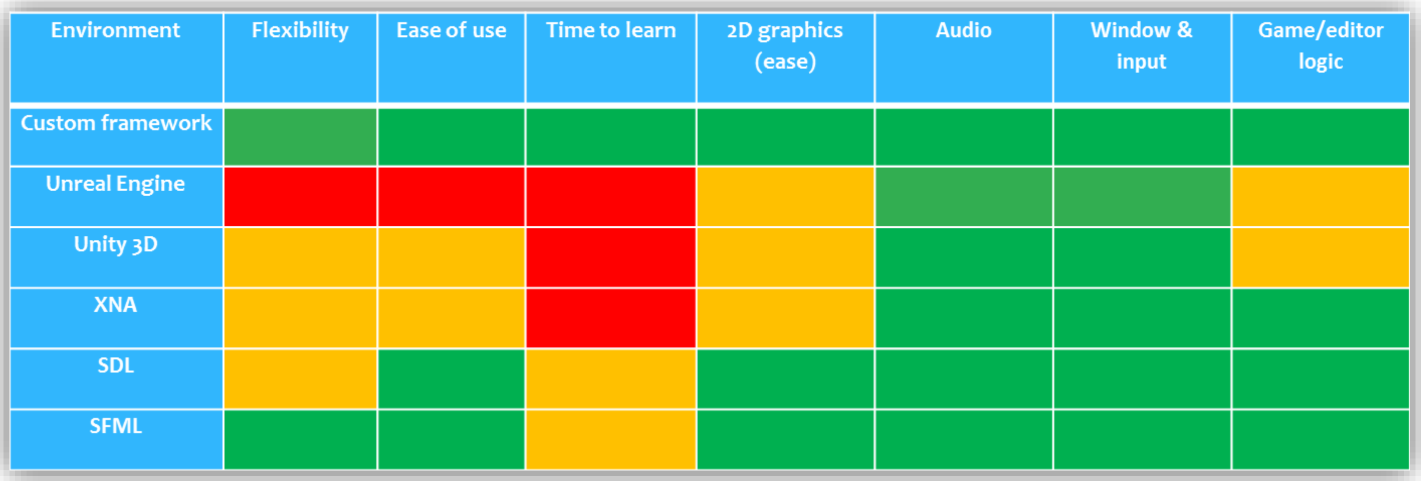
The reason that the Krypta 2D framework was created is so that development would not be reliant on other bulky engines that wouldn't be suited to the requirements of the project.

The application and products that were designed were primarily light weight so it would be unnecessary to use a larger framework that is "fatty" by providing a large amount of unnecessary functions.

At the beginning of this project we wanted to know what was behind the scenes and know what is available for our product to allow greater control. With this in mind, we analysed and reviewed a multitude of current gaming engines available at our disposal taking into account including the flexibility, ease of use, and learning curve.

The table below shows our findings from reviews of current development environments that influenced our design decision to create Krypta 2D:

- Red boxes indicate a lot of effort.
- Orange boxes indicate a moderate effort.
- Green boxes indicate a light amount of effort.



Environment	Flexibility	Ease of use	Time to learn	2D graphics (ease)	Audio	Window & input	Game/editor logic
Custom framework	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Unreal Engine	🟥	🟥	🟥	🟧	🟩	🟩	🟧
Unity 3D	🟧	🟧	🟥	🟧	🟩	🟩	🟧
XNA	🟧	🟧	🟥	🟧	🟩	🟩	🟩
SDL	🟧	🟩	🟧	🟩	🟩	🟩	🟩
SFML	🟩	🟩	🟧	🟩	🟩	🟩	🟩

Figure 1: Framework Review Table

As shown in Figure 1 the results that were gathered from the review favoured the development of a custom framework. With this in mind, the aim was to develop an easy to learn, light weight framework that could be deployed in many different environments and applications.

1.6 – Users of Krypta

The users of Krypta refer to the users that will be interacting solely with the game that has been built.

Krypta Game and Editor

Standard User – The standard user will be the main users of the system i.e. the player base. This user will only be allowed to play the main game with the pre-built maps and game functions.

Registered User – The registered user will be an extension from the main users of the system. Registered users will join as a member of the community and allow users to upload their own maps to be ranked and played and download maps to try their skills.

Krypta Online Community

Administrator – The administrators will have control over user accounts and user-submitted content such as the reviewing of created maps and the ability to remove distasteful content.

1.7 – Constraints

Network Connection – The online play and sharing of maps for registered users will be dependent on an internet connection for the uploading and downloading of content.

Hardware – Besides a functional computer than can handle running a few complex processes at a time and Supports Visual Studio 2013 at the least, no other special hardware is required. The hardware needed for the server depends on the scale of the database, and the number of users of the product (and the website).

2. Development Environment

2.1 Operating System(s)

Windows 7 (32bit/64bit) is the main development and target platform for the product. Due to time constraints, there will not be a port to other platforms before release, however, ports upon post release are not ruled out.

2.2 Development Tools

Microsoft Visual Studio 2013 – Software Development:

Visual Studio 2013 is the main development tool being used for the Krypta based on personal preference. The compiler used is the visual C++ 2013 compiler. The reason this was chosen is because the compiler offset support of C+11 has great debugging tools and is the best compiler for Windows.

Microsoft Visual Studio 2013 can be found at: http://www.visualstudio.com/

GLEW:

GLEW is an OpenGL Extension Wrangler Library (GLEW) that is an open source cross platform C/C++ extension loading library. It allows for the efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform.

GLEW can be found at: http://glew.sourceforge.net/

Notepad++:

Notepad++ is a free source code editor and Notepad replacement that supports several languages. Notepad++ is the development environment used for the development on the website and online community aspects.

Notepad++ can be found at: http://notepad-plus-plus.org/

Git – Version Control:

Git is the version control technology that has been used to ensure that all code and progress is safely secured, saved and can be retrieved at any time. GitHub allows multiple members to also access the code giving it a centralised location making sharing easier.

Git can be found at: http://git-scm.com/

GitHub can be found at: https://github.com/

2.3 Programming Languages

C++:

Three sections of the product are developed in C++, being: the framework, the game, and the editor. As C++ is highly flexible and efficient, it is well suited for providing the core, and front end, functionality. Considering that C++ is the most well understood language amongst the team as well, it is the best candidate for the product's development.

PHP:

The back end of the product, being the server, is written in PHP, as it is a well-documented and uncomplicated language designed for web development. It is used to handle the MySQL (explained under Libraries and Frameworks) requests and updates sent to the server.

JavaScript:

JavaScript and HTML are both proven web development tools, and are used in the development of the website.

SQL:

All of the databases are stored and managed through Structured Query Language (SQL). The data that will be stored includes user accounts, maps, comments, ratings and statistics.

2.4 Communication and Project Management

Trello – Project Management:

Trello was used as a Project Management tool that could be used to ensure that the project was properly managed and organised.

Trello can be found at: https://trello.com/

Skype – Online Communication:

Due to our timetable and not living within close proximity of each other, Skype was used as the communication tool for online meetings on a weekly basis.

Skype can be found at: http://www.skype.com/en/

2.5 Asset Development

Microsoft PowerPoint – Art Shape Development:

PowerPoint was used to develop the basic sprite and art tiles with the use of its range of pre-defined shapes.

Ulead PhotoImpact X3 – Image Editing Software:

Photo Impact was used to clean up, create textures and further develop art assets that were designed in Microsoft PowerPoint.

PhotoImpact X3 can be found at:
http://www.paintshoppro.com/en/products/photoimpact/default.html

Adobe Photoshop CS5 – Image Editing Software:

Photoshop was used in the development of the Krypta logo's and art assets.

2.6 Documentation

Microsoft Office Suite:

The Microsoft Office Suite has been used extensively for a range of purposes. Word has been used for documentation, note taking and PowerPoint has been used for art development and slide show presentations.

3. Frameworks and Libraries

Win32:

Windows' native C API allows users access to medium level sections of the Windows operating system. As Krypta 2D is written in C++ and only supports Windows, naturally Win32 shall be used for system specific functionality, including threading, file system management, window creation, and networking.

OpenGL:

OpenGL is a multi-platform, state-machine, graphics, C API. Krypta 2D uses the fixed pipeline versions, being 2.0 and below, for simplicity in development of 2D games. The use of OpenGL within Krypta 2D allows for the development of all of the graphical components of the Krypta game.

For information and downloads visit: https://www.opengl.org/

OpenAL Soft:

OpenAL Soft is similar in naming conventions to OpenGL. It is an easy to use, multi-platform, C API designed to handle the playing, mixing, and manipulation of audio. Krypta 2D uses OpenAL to play and manipulate audio in a simple fashion for ease of use in game development.

OpenAL Soft can be found at: http://kcat.strangesoft.net/openal.html

FreeType 2:

FreeType is a C API that handles loading and rasterizing of true-type fonts. Krypta 2D implements FreeType 2.0 for simple font loading, rendering and OpenGL contexts.

FreeType 2.0 can be found at: http://www.freetype.org/freetype2/

Stb_image:

Stb_image reduces the complexity of loading images for the game. Stb image implements a PNG, JPEG and TGA loader into a single C file without dependencies.

Stb_image can be found at: https://github.com/nothings/stb

mpg123:

mpg123 is a real time MPEG 1/2/2.5 audio decoder for layers 1/2/3. Krypta 2D employs mpg123 specifically for its support for MP3, for use with OpenAL.

Mpg123 can be found at: http://www.mpg123.de/

QT Visual Studio Add-In:

Qt is a cross platform application and USI framework for C++. This allowed for the creation of all user interfaces throughout the Map Editor. This was realized through the use of a plug-in in Microsoft Visual Studio. At the current time the user must have QT 5.3.1 installed including the QTR creator.

QT Visual Studio Add-In can be found at: http://qt-project.org/wiki/QtVSAddin

jQuery:

jQuery is a JavaScript API that simplifies HTML manipulation and event handling. The website's code involves the use of jQuery to simplify its development.

jQuery can be found at: http://jquery.com/

PDO:

PHP Data Objects is a PHP extension that provides easy access to generic database features in a secure fashion. PDO is used to access the MySQL database used in the server.

PDO can be found at: http://php.net/manual/en/book.pdo.php

Lighttpd:

Lighttpd ("lighty") is an open source web server optimised for speed-critical environments while remaining standards-compliant. It provides a small memory footprint and provides effective management of the cpu-load.

Lighttpd can be found at: http://www.lighttpd.net/

FastCGI:

FastCGI is a binary protocol for interfacing interactive programs with a web server. It's very simple as its simple CGI with a few extensions. This was used to tie everything together to the server.

FastCGI can be found at: http://www.fastcgi.com/drupal/

4. System Requirements

4.1 Krypta 2D Framework

These recommendations are specific to the beta release and are subject to change:

- Processor: 1 GHz 32-bit or 64-bit processor
- Memory: 1 GB of system memory
- Hard drive: 16 GB of available disk space
- Video card: Support for DirectX 9 graphics with 128MB memory

4.2 Krypta Game and Map Editor

These recommendations are specific to the beta release and are subject to change:

- Processor: 1 GHz 32-bit or 64-bit processor
- Memory: 1 GB of system memory
- Hard drive: 16 GB of available disk space
- Video card: Support for DirectX 9 graphics with 128MB memory

4.3 Web Server

- Web server that is running PHP
- A web browser that supports css and jQuery

5. Development Environment Installation

Library Dependencies:

In order to set up the Krypta 2D Framework for compilation, the following libraries and frameworks will be required:

- glew32
- openGL32
- freetype253
- openAL32
- libmpg123
- std_lib

Instructions:

A Visual Studio project/solution file will be provided with all dependencies and compiler flags set-up.

The dependent libraries are will need to be dynamically compiled in order to create the .dll files following their respective compilation guides. Place each external library's .lib files in the IDE's "/lib" directory and the include files in the IDE's "/include" directory.

However, the dependent libraries will need to be dynamically compiled (creating .dll files) following their respective compilation guide. Place each external libraries .lib files in the IDE "/lib" directory and the include files in the IDE "/include" directory.

The .dll files (of each library) will need to be in the same directory as any executable compiled with Krypta2D. Some libraries used do not require pre-compilation and may be directly compiled with the framework.

Download:

The Krypta2d framework can be downloaded either from our website, or from the GitHub. The files can be found at: kryptagame.com/download

Usage:

To include all modules of the framework, krypta2d.h provides includes to each module.

6. Architecture Overview

6.1 – Project Structure

Krypta is aimed towards a community involved, framework-based project. The project is comprised of four components: The Krypta 2D Framework, The Map Editor, The Game and the Online Technologies.

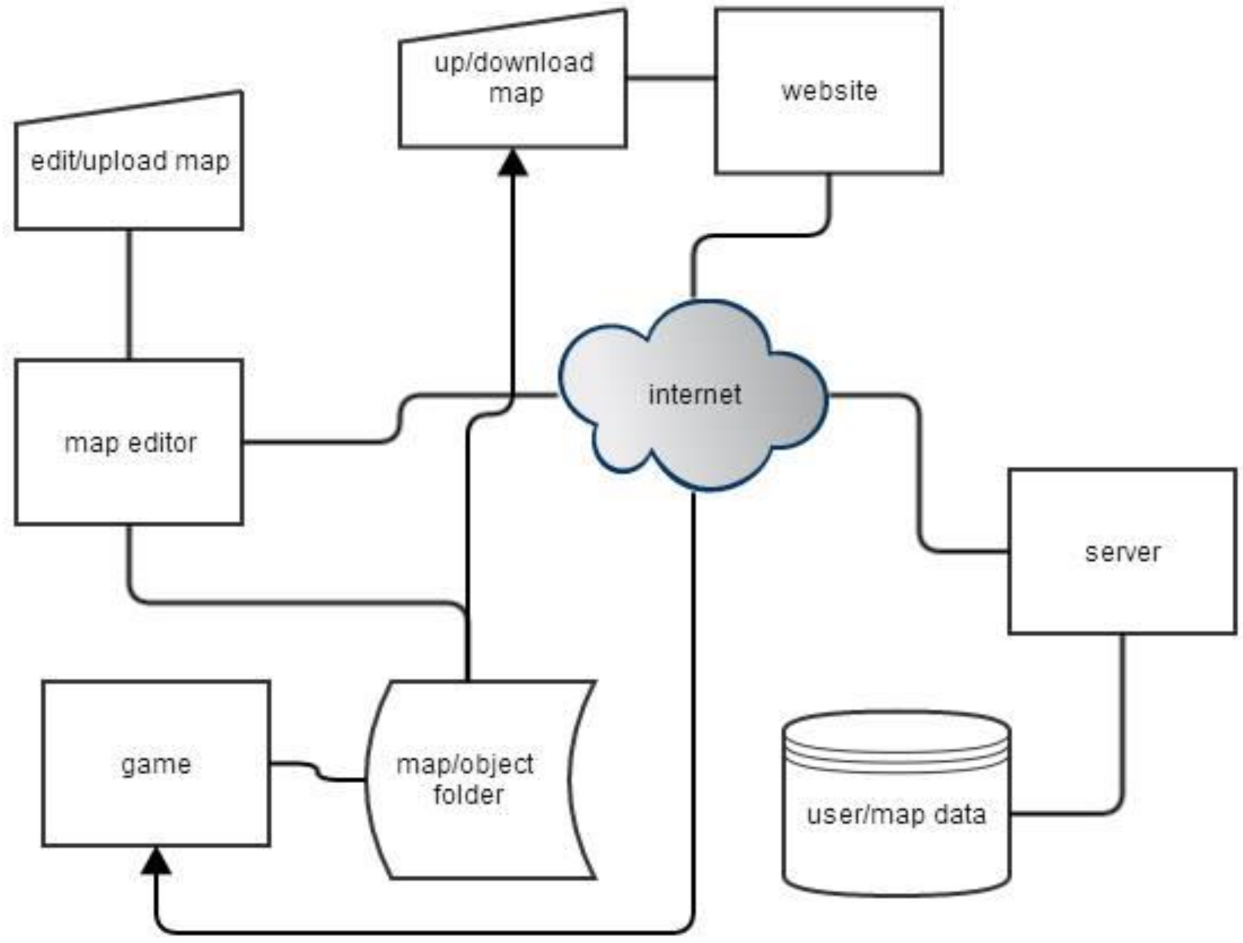


Figure 2: UML flow diagram of the system

6.2 – Project Subsystems

6.2.1 – Krypta 2D Framework

The Krypta 2D Framework provides the basic skeletal support for the Krypta game, the Map Editor and the website through network support. Krypta 2D is divided into seven separate modules each providing a specific functionality:

- **Utilities:** provides the basic tools that aren't group with any of the other modules.
- **Process:** provides processing support and multi-threading functionality.
- **System:** provides system specific functionality such as window creation and file system handling.
- **Media:** provides the handling of media of audio and image files and formats.
- **Audio:** provides and manages audio playback and manipulation.
- **Graphics:** provides graphics rendering and manipulation.
- **Network:** provides handling of network connections of different types across LAN/WAN networks.

The framework can also be used to create simple applications including those not game specific.

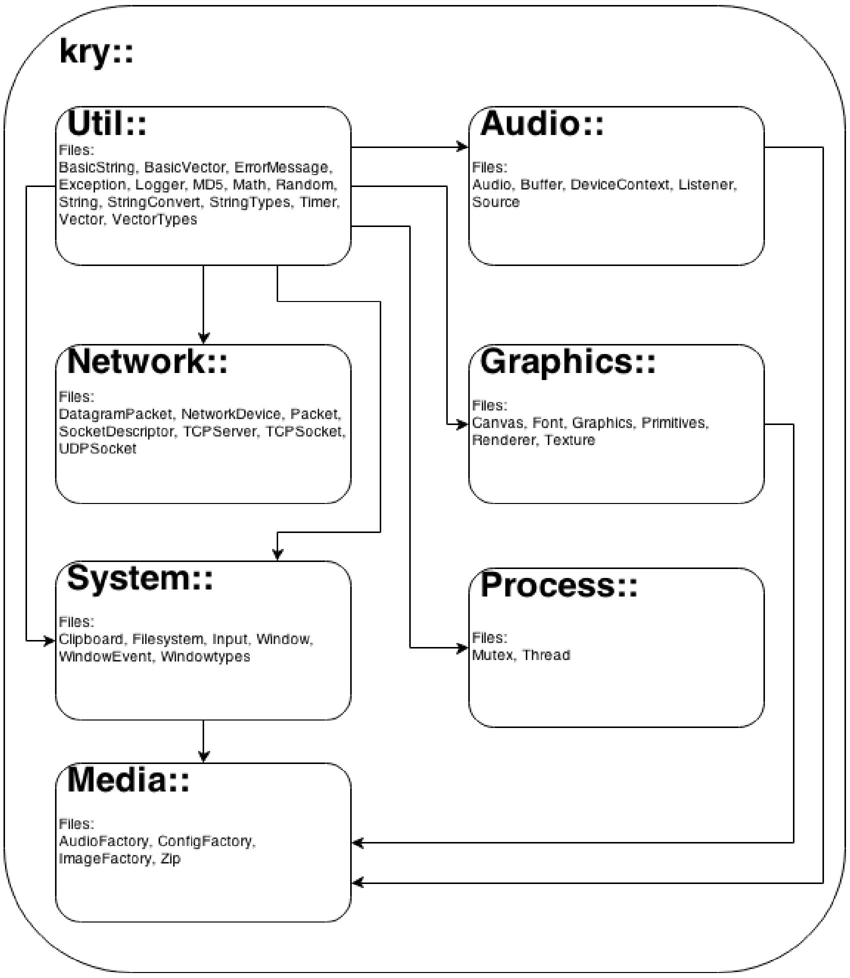

kry::
Util::
Files:
BasicString, BasicVector, ErrorMessage, Exception, Logger, MD5, Math, Random, String, StringConvert, StringTypes, Timer, Vector, VectorTypes
Audio::
Files:
Audio, Buffer, DeviceContext, Listener, Source
Network::
Files:
DatagramPacket, NetworkDevice, Packet, SocketDescriptor, TCPServer, TCPSocket, UDPSocket
Graphics::
Files:
Canvas, Font, Graphics, Primitives, Renderer, Texture
System::
Files:
Clipboard, Filesystem, Input, Window, WindowEvent, Windowtypes
Process::
Files:
Mutex, Thread
Media::
Files:
AudioFactory, ConfigFactory, ImageFactory, Zip

Figure 3: Namespace Diagram

6.2.2 – Map Editor

The Map Editor will allow the user to create, modify and upload maps that are used within the game. The Map Editor was introduced to tie into the community aspect to increase marketability of the product by allowing users to extend the lifetime of the Krypta game content with the user's own content.

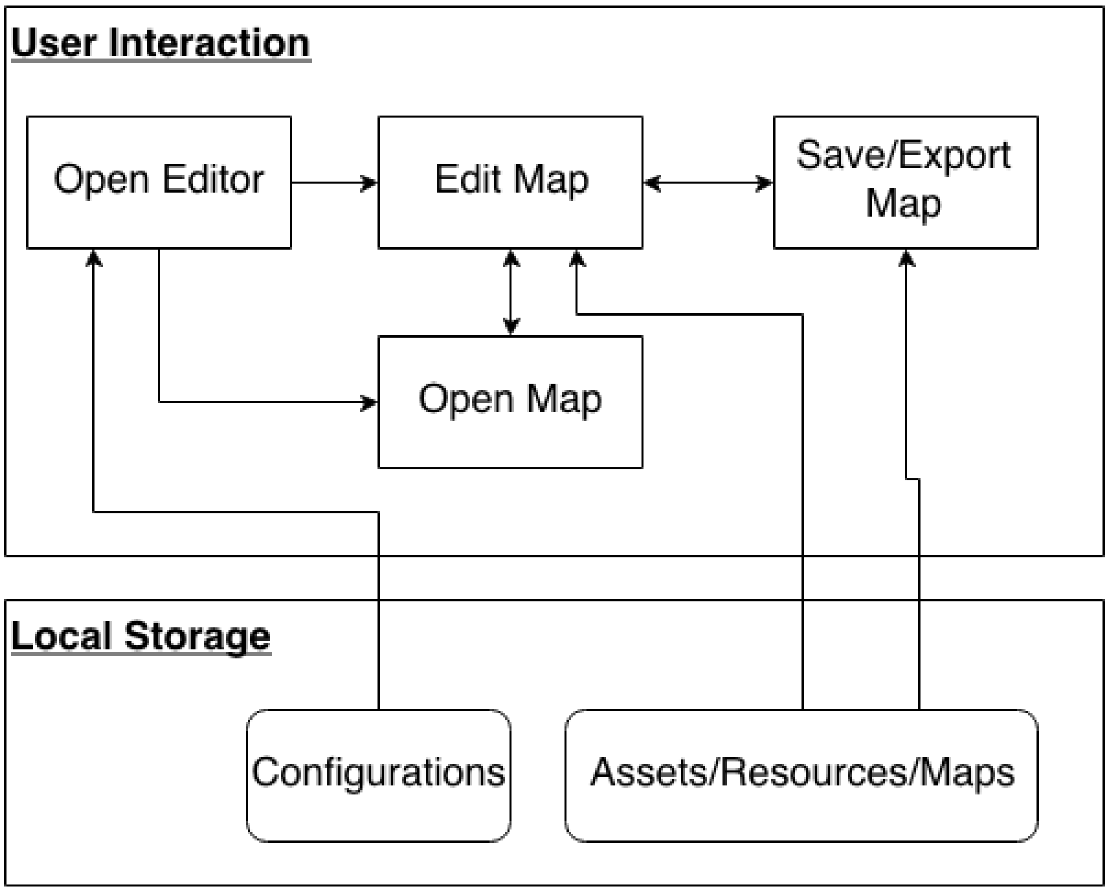


Figure 4: Map Editor Diagram

Upon opening the editor, the user is presented with a blank map. The map may be edited and saved, or the user may wish to create a new map. Most significant changes either require a prompt or confirmation, depending on the event.

A map's contents are managed by tool interfaces and a viewport in the centre which displays the current map design. The viewport uses Krypta 2D's Graphics module to render its contents. The tools interfaces contain assets read from an asset pack, and display them inside the interfaces for easy visual recognition.

The user is able to place assets around the map by various convenience input methods (keyboard shortcuts, mouse clicks), and edit their properties in the tool box interfaces around the viewport.

All assets are configurable, and the global map options themselves are also configurable, via separate tool interfaces. Objective editing is also available from these interfaces.

Uploading a map from the editor requires the user to enter their log-in details, ensuring that they actually have an account to tie the map to. Map uploading requires Krypta 2D's Network module, and blocks (seemingly, updates a progress bar) until the map has finished uploading, or until an error occurred (a visual indication is provided).

Saving a map writes it to a file in a human readable format for easy editing and reloading. Exporting the map writes it to a binary file (saves space and size) along with additional information (such as an icon) for use in the game.

The editor itself also has options that can be configured at any time, using Krypta 2D's Media module, which is saved and loaded to and from a config file located in the editor's local path.

6.2.3 – Game

The Krypta Game provides the entertainment value for the product and allows the user to experience their own and other user’s maps. An online connection is not a requirement to play the game as the game comes with the campaign but content such as online maps may not be available.

The game is designed in such a way that users may edit its contents including art and audio or create their own to be used.

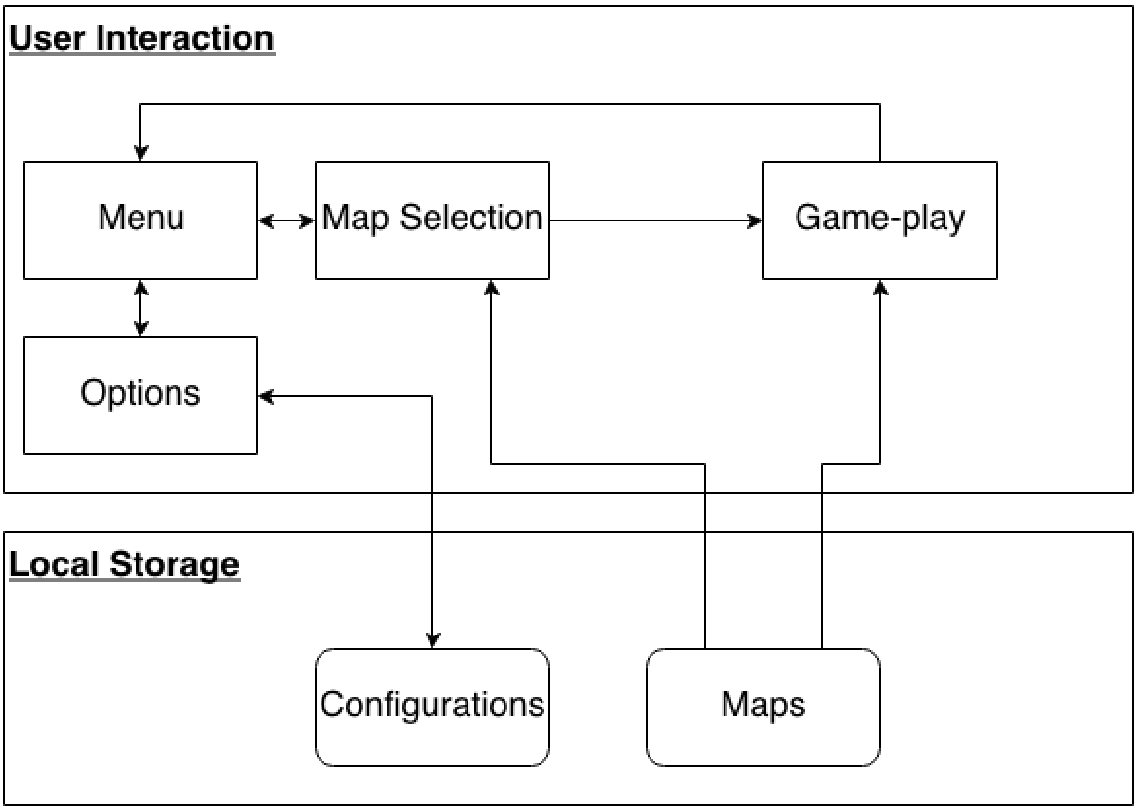


Figure 5: Game Diagram

Krypta's core development stems from Krypta 2D's System module, which handle the creation and management of Krypta's main window.

The user is presented with a new interface providing maps to play on from the user's local map folder. Each map is represented by an icon and a map title. Selecting a map, and continuing on, will load the map in a new state and allow the user to begin playing.

Loading Krypta's assets are handled by Krypta 2D's Media module (for loading of textures/audio/text), and passed to its respective modules (Graphics/Audio) from there.

At all times, the user has the chance to change the options (some options may be omitted based on the fact that the option is already in use or may require Krypta to restart, see the user manual for information on the available options), and may return from the options interface with or without saving them. The options are saved to a configuration file on the user's local game directory. Most configuration handling comes from Krypta 2D's Config class in its Media module.

Beginning the game, the user may quit/check options/resume from a single in-game menu interface. The currently available tools/items may display on a separate interface, as well as controls, such as map viewing. Opening the menu interface may pause the game, while the game will continue during the viewing of any other interface.

The game uses Krypta 2D's Graphics and Audio modules for the rendering of most of Krypta's interfaces, as well as the game's graphics and music/sounds.

Upon completing a map, the user will be revealed and will be returned to the map selection interface.

6.2.4 – Web / Server

The website is the front end section and provides the users with a quick and simple method to the server. The website will allow users to upload and download custom maps, discuss maps with other users, review maps and look at information about the product.

The downloading/uploading of maps requires the user to be registered to the website. The rest of the website including information regarding the product doesn't require the user to log in to their account.

The server manages all of the community data and user information using Lighttpd and MySQL. Data is stored and retrieved at will and can be managed by a system administrator by accessing the databases and code manually.

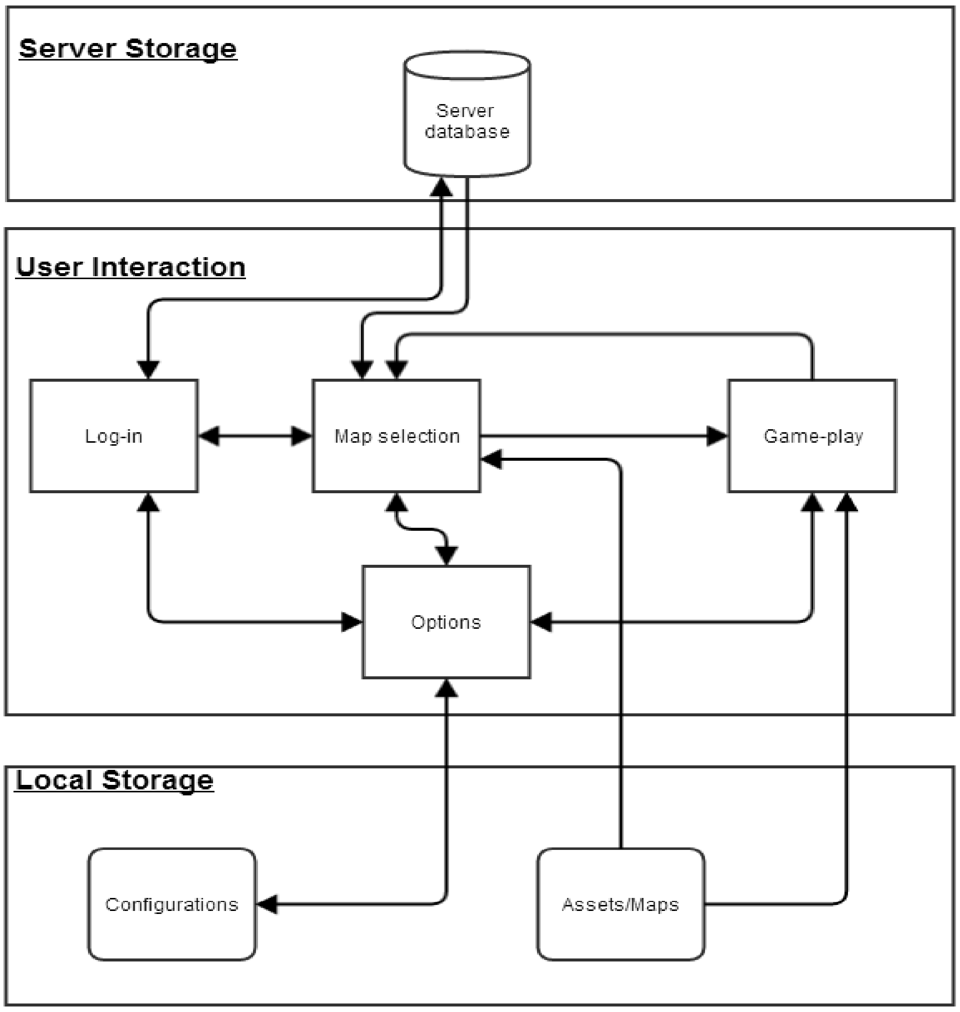


Figure 6: Data Flow Diagram

6.3 – Data Storage:

Local map data is stored in a single folder on the user's hard drive, which is accessed by both the game and the map editor to use during gameplay/map editing. Local data can be moved around however, and users are free to manually pass on data to one-another. Local data may also be sent to the server via the map editor and the website.

Data from the server, such as user/map statistics and data may be retrieved via the website or the game. The game may request user information, or maps that are available from the server and the website may request all types of data from the server to be displayed or downloaded. The server stores all the data in a database, including map information/data, user information/data, and the website's discussion page's information/data.

A more technical explanation of the database storage is provided further on in the manual via the Data Dictionary.

7. Structures – Files and Classes

7.1 – Framework:

Krypta 2D is categorised into seven modules which are all under the same (and their module name) namespace, 'kry'. The seven modules are:

- **Process:** contains functionality regarding multithreading.
- **System**: provides a much cleaner, and simpler, method of creating, handling, threading, and destroying, windows, as well as a few Operating System specific methods and functionality.
- **Graphics:** helps manage rendering to a context using primitives, fonts, or textures.
- **Media:** provides functionality for the loading of different file formats, and returning appropriate data for those formats.
- **Network:** contains the ability to establish LAN/WAN connections, and send/receive data across the internet.
- **Audio:** similar to Graphics, but all about managing audio in a simple manner.
- **Utilities:** the module that contains helper methods and functionality, and isn't specific to any other module.

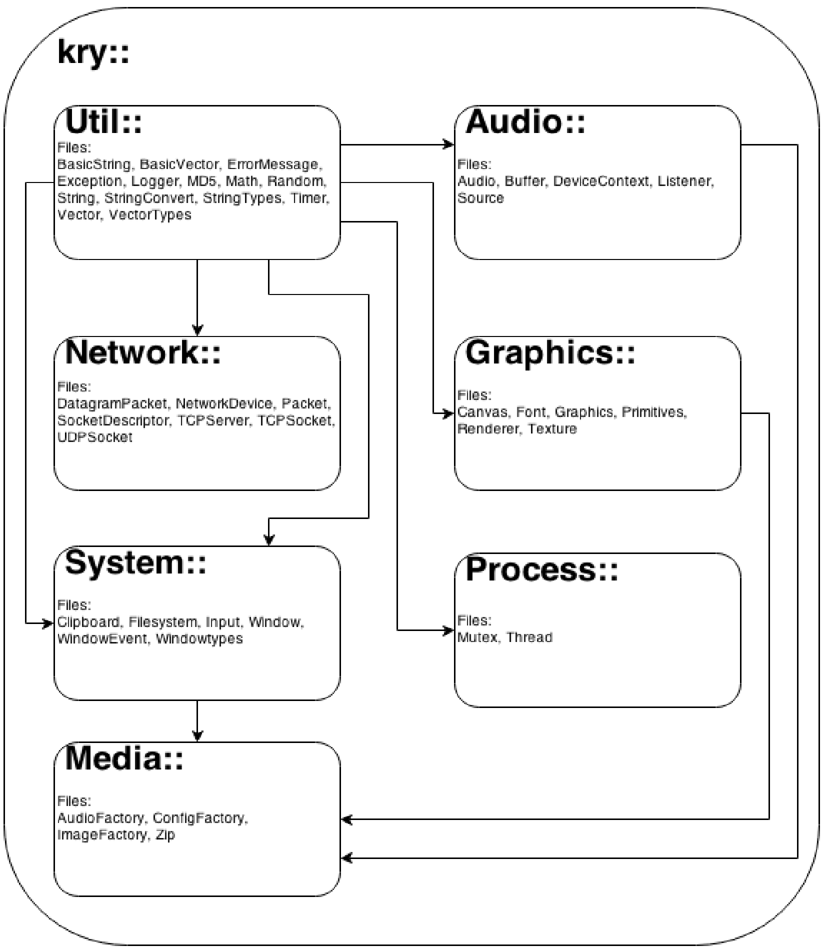

kry::
Util::
Files:
BasicString, BasicVector, ErrorMessage, Exception, Logger, MD5, Math, Random, String, StringConvert, StringTypes, Timer, Vector, VectorTypes
Audio::
Files:
Audio, Buffer, DeviceContext, Listener, Source
Network::
Files:
DatagramPacket, NetworkDevice, Packet, SocketDescriptor, TCPServer, TCPSocket, UDPSocket
Graphics::
Files:
Canvas, Font, Graphics, Primitives, Renderer, Texture
System::
Files:
Clipboard, Filesystem, Input, Window, WindowEvent, Windowtypes
Process::
Files:
Mutex, Thread
Media::
Files:
AudioFactory, ConfigFactory, ImageFactory, Zip

Figure 7: Namespace Diagram

7.2 – Framework Dictionary – All under namespace – kry::

7.2.1 – Namespace Process:

Class Thread:

- Functionality for creating/destroying/joining/pausing/resuming threads.
- Contains a thread ID for thread comparison.
- Contains a function callback as the new thread.

Class Mutex:

- Create/destroy.
- Locks/unlocks from any thread, to prevent races.

7.2.2 – Namespace System:

Class Window:

- Can create multiple windows in different threads.
- Create a window based on a title and dimensions.
- Prompt the window, checking if it is open/closed.
- Event handling based on three types of obtaining events (requires WindowEvent):
    - Peek - get the next event, but dont remove it from the queue.
    - Get - get the next event and remove it from the queue.
    - Wait - block until an event arrives, and remove it from the queue.
- Show/hide the window.
- Minimize/maximize/restore the window.
- Toggle fullscreen/borderless-window mode.
- Contains a graphics context for OpenGL, and a window process callback.
- Window set/get position/dimensions
- Clip/hide cursor0

Class WindowEvent:

- Contains an event type, and any event details.
- Event types:
    - Keyboard input
    - Mouse input
    - Window focus/resize/move

FileSystem (file):

- File/directory copy/move/delete/exists/create.
- Read/write all lines from/to a file (delimiter as '\n' by default).
- Read/write all bytes from/to a file (1 byte chunks by default).
- Get all file names from a directory.
- Get all directory names from a directory.

Clipboard (file):

- Open and close the Windows clipboard.
- Read from/write to the clipboard as a buffer of bytes.
- Clear the clipboard.

Input (file):

- Mouse button/keyboard key definitions.

WindowTypes (file):

- Type definitions for different types used by Window.

7.2.3 – Namespace Graphics:

Class Texture:

- Contains an OpenGL texture handle.
- Create (pass a pixel buffer) and destroy a texture.

Class Canvas:

- Holds a container full of primitive (shapes, fonts, textures) render types for render call.
- Apply transformations to the canvas.

Class Sprite:

- Contains a Texture.
- Apply transformations to the sprite.

Class Font:

- Contains true-type font information and metrics.

Class Text:

- Contains a string of text and a Font.
- Create by passing a Font and a string.
- Apply transformations to the text.

Primitives (file):

- Primitive shape classes.
- Each class contains vertices.

Class Renderer:

- Holds a container of canvas' and shaders.
- Add a canvas or shader (consider rendering order).
- Render all of the renderer's contents.
- Clear all contents.

7.2.4 – Namespace Media:

ImageFactory (file):

- Load and convert image files/data to Textures/RGBA data.

AudioFactory (file):

- Load and convert audio files/data to PCM data.
- Load and stream MP3 files.

Class Config:

- Contains a map of values to options.
- Check if a key/section exists.
- Get value from key.
- Read and write to and from a file (requires a specific format).

Class Zip:

- Read/write data to/from ZIP files.
- Store data in a directory format.

7.2.5 – Namespace Network:

Class Packet:

- Contains an array of bytes.
- The first four-eight bytes stores the length of the array.
- Fill the packet with any type, stored in the array.
- Clear the packet of contents.

Class DatagramPacket:

- Inherits Packet.
- Contains an address and a port.

Class TCPServer:

- Contains a TCPSocket and connection information.
- Contains a callback for accepting connections.
- Create by passing a port.
- Start/stop listening for connections.
- Pass newly created connections to the callback.
- Prompt for backlog count.
- Toggle reusing the address to bind with.
- Close the server.

Class TCPSocket:

- Contains an address and a port.
- Contains a socket descriptor.
- Contains callbacks for connection and disconnection.
- Create by passing a port to the socket.
- Open and close the socket for an address and a port.
- Connect and disconnect to the opened socket's data.
- Prompt the socket, checking if it is still connected.
- Set timeout values for the socket.
- Send and receive packets.
- Send and receive primitive types (creates a packet for them, convenience).
- All send/receives are blocking calls.
- Set callbacks for connection and disconnection events.

Class UDPSocket:

- Contains a socket descriptor and a socket address regarding the client.
- Contains a boolean as to whether or not to reuse the bound address.
- Create by passing a port to the socket.
- Bind the socket to its address.
- Send and receive DatagramPackets, or normal Packets to an address and port.
- Send and receive primitive types (creates a packet for them, convenience).
- All send/receives are blocking calls.
- Open and close the socket.
- Set timeout values for the socket.

Class NetworkDevice:

- Singleton pattern.
- Contains winsock/system version/information.
- Start and stop winsock.

7.2.6 – Namespace Audio:

Class DeviceContext:

- Singleton pattern.
- Contains a handle to an OpenAL device and context.
- Destruction destroys the handles. Calls to OpenAL will cause errors.

Class Buffer:

- Contains a handle to an OpenAL buffer.
- Create by passing a sample buffer and a channel format.

Class Source:

- Contains a handle to an OpenAL source.
- Create by passing a Buffer.
- Start/stop/pause the audio from the source.
- Set pitch/volume for the source.
- Get and set current playback offset.
- Get current state of the source (playing, stopped, paused).
- Toggle looping the source.
- Queue/unqueue another buffer.

Class Listener:

- Singleton pattern.
- Get and set position and direction of the listener.
- Get and set the volume that the listener receives.

7.2.7 – Namespace Utilities:

Class BasicString:

- Same functionality as std::string.
- Explode the string based on a delimiter.
- Convert the string to an upper/lower case equivalent.
- Starts/ends with convenience methods.
- Trim white space from both ends of the string.
- Reverse the strings contents.
- Template arguments provide size conversions (create wide strings).
- Invoke on string literals without counting the length of the literals.

Class BasicVector:

- Template class, determines underlying data's type.
- Operator overloads for convenience.
- Get the vector length of the data.
- Normalise the data.
- Invert the data.
- Get the dot value of the data and another vector's data.
- Get the distance between the data and another vector's data.
- Get the cross product of the data and another vector's data.
- Typedef's of the vector and its different types (e.g. Vector2i, Vector3f, etc.).

Class Exception:

- Inherits std::exception.
- Contains a String as the message.
- Create by passing a message, the function signature, the line, and the file.
- Implements a virtual 'what()' method.

ErrorMessage (file):

- Convert Windows error messages to BasicStrings.

Maths (file):

- Constants such as PI, PI squared, half PI, etc.
- Radians/degrees to degrees/radians conversion functions.
- Clamp template function.
- Box intercept for checking a point intersects a box.
- Radial intercept for checking a point intersects an ellipse.
- Radius for getting radius of given ellipse dimensions and an angle.
- Angle between two points.
- Trajectory returning a position given starting position, angle and distance.
- Direction vector given from one point to another.
- Angle to direction, converts angle to direction vector.
- To isometric, converts orthographic coordinates to isometric coordinates.
- To orthographic, converts isometric coordinates to orthographic coordinates.
- Absolute box, forces first point top-left and second point bottom-right.

Class Random:

- Contains a seed.
- Contains a random generation engine from the standard library.
- Generate random values of different sizes/types.
- Generate random values between a min and a max.
- Re-seed the engine.

Class Timer:

- Contains a duration to check for.
- Create by setting a duration.
- Start and stop the timer.
- Prompt the timer, checking if the time has elapsed.
- Get the current elapsed time.

Class Logger:

- Change logging output stream.
- Log strings and exceptions.
- Get the current date and time as a string.

MD5 (file):

- Convert strings, data, and files to MD5 hashed strings.

StringConvert (file):

- Convert Strings to number values and vice-versa.

StringTypes (file):

- Type definitions for useful BasicString types (String, WString, etc.).

VectorTypes (file):

- Type definitions for useful BasicVector types (Vector2i, Vector4f, etc.).

8. Data Dictionary

8.1 – Map File Dictionary

The map file is organised similar to an ".ini" file.It comprises of sections, keys and values.

- Each section is defined in square brackets like "[Section]".
- Each key is define on a new line like "key = ".
- Each value must proceed the key and only uses the rest of the line like "key = value".

8.1.1 – Settings Structure

Order of sections and keys are not strict. Sections may be fragmented (keys belong to the last section defined). The sections required for Krypta include:

- Settings
- Items
- Entities
- Tiles

Data Name	Data Type	Data Description
name	String	The name of the map
iconImage	String	The local path to the icon image
checksum	String	The checksum of the map file
fogOfWar	Boolean	Whether to use fog of war
revealOfWar	Boolean	Whether to use reveal of war
fogTint	Vector4f	The TGBA tint applied to objects in the fog of war
fogThroughWalls	Boolean	Whether the fog is not restricted by line of sight
fogTillLastWall	Boolean	Whether the line of sight continues to reveal walls until the last wall
tileDimensions	Vector2f	The pixel dimensions of a tile for the isometric grid
floorFadeTime	Float	The milliseconds to transition to next floor on warp
cameraScale	Float	The initial camera scale/zoom

deathFadeTime	Float	The milliseconds to fully transition the game over overlay
gameOverSkin	String	The name of the game over overlay in the “overlaySkinConfig” file
lifeSkin	String	The name of the skin overlay in the “overlaySkinConfig” file
soundtrackSize	String	The number of track keys to look up (index: 0)
Soundtrack#	String + int	The local path to the sound file
randomizeSoundtrack	Boolean	True/False to randomize the next track from the soundtrack list.
inventoryTextSize	Int	The font size of the texts for each inventory item
inventoryIconDimensions	Vector2f	The pixel dimensions of an inventory item.
inventoryIconGap	Vector2f	The gap between icons. X: gap from left side and top of screen. Y: gap between next icon vertical.

8.1.2 – Item Structure

For each item you wish the item factory to load, you must specify the name of the section for the item under the "Items" section, with its key a unique int ID i.e.

```
[Items]
myItem = 0
```

The types of items include: (These are case sensitive)

- key
- loot
- weapon

Data Name	**Data Type**	**Data Description**
type	String	The name of the type as found above
inventoryName	String	The name of the item.
showInInventory	Boolean	Make item visible in the inventory.

8.1.3 – Floor Structure

For each floor you wish the floor factory to load, you must specify the name of the section for the floor under the "Floors" section, with its key a unique int ID i.e:

```
[Floors]
myFloor = 0
```

Data Name	Data Type	Data Description
dimensions	Vector2i	The grid dimensions (columns / rows)
floorBinary	String	The local path to the floor file

8.1.4 – Entity Structure

For each entity you wish the entity factory to load, you must specify the name of the section for the entity under the "Entities" section, with its key a unique int ID i.e.:

```
[Entities]
myEntity = 0
```

The types of entities include: (These are case sensitive)

- static: Static
- player: Dynamic
- trigger: Dynamic
- text: Passive
- portal: Static
- item: Static
- trap: Static
- enemy: Dynamic
- attack: Dynamic
- chest: Static
- door: Static
- checkpoint: Static
- spawner: Static
- fogRevealer: Static

A passive entity only requires the "type" key, because it does not exist in the world space.

Data Name	Data Type	Data Description
type	String	The name of the type as found above
group	Int	The group ID for this entity, used like a filter
floor (optional)	Floor ID	The ID of the floor to spawn on – if the floor is not specified the entity will not be added to any floor and technically is not spawned.
position	Vector2f	The spawning grid cells coordinates for the entity
dimensions	Vector2f	The dimensions of the entity in grid cell dimensions, based around the position as the centre
direction	Float	The entity's initial rotation when spawned
seeInFog	Boolean	Whether the entity is rendered if in the fog
directions	Int	Either 1, 4 or 8 is accepted. Specifies the number of directions the entity can render in (with each direction having a different skin). 4 directions is equivalent to north, south, east and west.
skinConfig (optional)	String	The local path to the skins file used for this entity.

8.1.5 – Dynamic Entity Extras

Data Name	Data Type	Data Description
maxHeuristic	Int	Used by path-finder to ignore any cells with a heuristic greater than this
"heuristic TYPE"	Int	"heuristic" followed by the TYPE which is the type name of any entity or tile type. The value of this entity's heuristic to this type.
"heuristicTile ID"	Int	"heuristicTile" followed by the ID of a tile. The value is this entity's heuristic to this particular tile.
"heuristicEntity ID"	Int	"heuristicEntity" followed by the ID of an entity. The value is this entity's heuristic to this particular entity.

8.1.6 – Tile Structure

For each tile you wish the tile factory to load, you must specify the name of the section for the tile under the "Tiles" section, with its key a unique int ID:

```
 [Tiles]
myTile = 0
```

The types of tiles include: (These are case sensitive)

- void
- solid
- wall

Data Name	Data Type	Data Description
Type	String	The name of the type as found above
skinConfig	String	The local path to the skins file used for this tile
Skin	String	The name of the skin used for the tile in the "skinConfig" file
Heuristic	Int	The base or default heuristic to this tile, used in path-finding
sortDepth	Int	The other of which this tile is rendered. A depth of less-than-or-equal-to 0 tile is sorted in the same canvas with the entities
sortPivotOffset	Vector2f	The grid cell corrdinate offset from the pivot of the tile (being its centre of an isometric diamond). This allows the tile to be sorted in the world space with entities; such as walls which may need to obfuscate entities behind it.

8.2 – Entity Specific Structures

8.2.1 – Trigger

Data Name	Data Type	Data Description
targetEntities	String	A list of entity ID's (split by commas). On trigger, each entity in this list will receive a trigger event.
oneUse	Boolean	Whether the trigger only executes its triggers once
delay	Float	Milliseconds delay before execution
triggerOnTouchGroups	String	A list of entity groups (split by commas). On touch (collision with the trigger based on its position and dimensions), if the entity touching belongs in one of the groups defined here, then the trigger is executed.
triggerSound(optional)	String	The local path to the sound file that will play once the trigger is touched by an entity belonging to the groups found in "triggerOnTouchGroups"
skinIdle(optional)	Sring	The name of the skin found in the "skinConfig" file, for the trigger at all times.

8.2.2 – Trap

Data Name	Data Type	Data Description
health	Float	The amount of health the trap initializes with
oneUse	Boolean	Whether the trap is used once. A use is after the "resetting" state of the trap.
stayTriggered	Boolean	Whether the trap stays in the triggered state. This is different from oneUse because it may initiate again.
triggerOnTouchGroups	String	A list of entity groups (split by commas). On touch (collision with the trap based on its position and dimensions), if the entity touching belongs in one of the groups defined here, then the trap goes into the "triggering" state
attackGroups	String	A list of entity groups (split by commas). Any entity that has collided with this trap will be attacked if the entity belongs to one of the groups defined here.
touchDelay	Float	Milliseconds of delay before the trap enters the "triggering" state, if not already in the "triggered state"

autoIntervalDelay	Float	Milliseconds of delay between “idle” state and “triggering” state.
triggeringTime	Float	Milliseconds delay between “triggering” state and “triggered” state.
triggeredTime	Float	Milliseconds delay between “triggered” state and “resetting” state.
resettingTime	Flat	Milliseconds delay between “resetting” state and “idle” state.
dyingTime	Float	Milliseconds delay between “dying” state and “dead” state
idleDamage	Float	The amount of damage to apply to any entities touching the trap and group is one of the groups defined in “attackGroups”, when the trap is in the “triggering” state.
triggeredDamage	Float	The amount of damage to apply to any entities touching the trap and group is one of the groups defined in "attackGroups", when the trap is in the "triggered" state
resettingDamage	Float	The amount of damage to apply to any entities touching the trap and group is one of the groups defined in "attackGroups", when the trap is in the "resetting" state
dyingDamage	Float	The amount of damage to apply to any entities touching the trap and group is one of the groups defined in "attackGroups", when the trap is in the "dying" state
deadDamage	Float	The amount of damage to apply to any entities touching the trap and group is one of the groups defined in "attackGroups", when the trap is in the "dead" state
idleSound (optional)	String	The local path to the looping sound when the trap is in the "idle" state
triggeringSound (optional)	String	The local path to the sound when the trap is in the "triggering" state
triggeredSound (optional)	String	The local path to the sound when the trap is in the "triggered" state
resettingSound (optional)	String	The local path to the sound when the trap is in the "resetting" state
dyingSound (optional)	String	The local path to the sound when the trap is in the "dying" state
deadIdleSound (optional)	String	The local path to the looping sound when the trap is in the "dead" state
skinIdle(optional)	String	The name of the skin in the "skinConfig" file when the trap is in the "idle" state
skinTriggering (optional)	String	The name of the skin in the "skinConfig" file when the trap is in the "triggering" state

skinTriggered(optional)	String	The name of the skin in the "skinConfig" file when the trap is in the "triggered" state
skinResetting (optional0	String	The name of the skin in the "skinConfig" file when the trap is in the "resetting" state
skinDying (optional)	String	The name of the skin in the "skinConfig" file when the trap is in the "dying" state
skinDeadIdle	String	The name of the skin in the "skinConfig" file when the trap is in the "dead" state

8.2.3 – Text

Data Name	Data Type	Data Description
Message	String	The message to display. "\n" will create new lines
hoverTime	Float	Milliseconds time to show the message

8.2.4 – Static

Data Name	Data Type	Data Description
idleSound (optional)	String	The local path to the looping sound
skinIdle (optional)	String	The name of the skin found in the "skinConfig" file

8.2.5 – Spawner

Data Name	Data Type	Data Description
oneUse	Boolean	The local path to the looping sound
spawnNextDelay	Float	The name of the skin found in the "skinConfig" file
spawnEntities	String	A list of entity ID's (split by commas). On spawning, each entity in this list will be spawned periodically based on the "spawnNextDelay"
skinIdle (optional)	String	The name of the skin in the "skinConfig" file when the spawner is in the "idle" state
skinSpawning (optional)	String	The name of the skin in the "skinConfig" file when the spawner is in the "spawning" state
idleSound (optional)	String	The local path to the looping sound when the spawner is in the "idle" state

spawningSound (optional)	String	The local path to the looping sound when the spawner is in the "spawning" state
spawnedSound (optional)	String	The local path to the sound when the spawner spawns its next entity
spawnLocation	Vector2f	The grid cell coordinate of where the entities will spawn

8.2.6 - Portal

Data Name	Data Type	Data Description
targetFloor	Int	The ID of the target floor the entity (target entity) will be warped to; may warp to same floor
targetPosition	Vector2f	The grid cell coordinate of the target location the entity (target entity) will be warped to
oneUse	Boolean	Whether the portal only has one use, a use is after its first target entity is warped, after use it is in the "disabled" state
warpGroups	String	A list of entity groups (split by commas). On touch (collision with the portal based on its position and dimensions), if the entity touching belongs in one of the groups defined here, then this entity is warped to the target floor and location
skinIdle (optional)	String	The name of the skin in the "skinConfig" file when the portal is in the "idle" state
skinSpawning (optional)	String	The name of the skin in the "skinConfig" file when the portal is in the "spawning" state
idleSound (optional)	String	The local path to the looping sound when the portal is in the "idle" state
skinDisabled (optional)	String	The name of the skin in the "skinConfig" file when the portal is in the "disabled" state
useSound(optional)	String	The local path to the sound when the portal is used

8.2.7 - Player

Data Name	Data Type	Data Description
viewDistance	float	The radius for revealing fog/reveal-of-war in grid cell measurements
moveAcceleration	float	Tiles per second acceleration of the movement
turnAcceleration	float	Rotations per second acceleration of the turning

maxMoveSpeed	float	The max tiles per second of the movement
maxTurnSpeed	float	The max rotations per second of the turning
health	float	The initial health of the player
inventory	String	A list of item ID's (split by commas). Each item will be added to the initial inventory
invincibilityTime	float	The milliseconds of invincibility from receiving attacks after the last attack
invincibilityFlickerTime	float	The milliseconds of rendering on/off of the player, to signal the invincibility state
dyingTime	String	Milliseconds delay between "dying" state and "dead" state
idleSound(optional)	String	The local path to the looping sound when the player is in the "idle" state
moveSound(optional)	String	The local path to the looping sound when the player moves
deadSound(optional)	String	The local path to the looping sound when the player is in the "dead" state
hurtSound (optional)	String	The local path to the sound when the player is attacked
dyingSound (optional)	String	The local path to the sound when the player is in the "dying" state
skinMove(optional)	String	The name of the skin in the "skinConfig" file when the player moves
skinDead(optional)	String	The name of the skin in the "skinConfig" file when the player is in the "dead" state
skinDying(optional)	String	The name of the skin in the "skinConfig" file when the player is in the "dying" state
skinIdle(optional)	String	he name of the skin in the "skinConfig" file when the player is in the "idle" state
lifeSkin(optional)	String	The name of the skin in the "overlaySkinConfig" file defined in the "Settings" section of the map. The skin pivot is snapped to the bottom-left of the screen, and may be offset using the pivot settings in the skin. The frame in the skin is chosen by the percentage of health of the player. 0% is the first and 100% is the last frame

8.2.8 – Item

Data Name	Data Type	Data Description
targetGroups	String	A list of entity groups (split by commas). On touch (collision with the item based on its position and dimensions), if the entity touching belongs in one of the groups defined here, then the item is added to its inventory (if it holds one, otherwise the entity disappears)
idleSound (optional)	String	The local path to the looping sound when the item is in the "idle" state
pickupSound(optional)	String	The local path to the sound when the item is touched by an entity
skinIdle(optional)	String	The name of the skin in the "skinConfig" file when the item is in the "idle" state

8.2.9 – Fog Revealer

Data Name	Data Type	Data Description
revealFogOfWar	Boolean	Whether the revealer reveals the fog-of-war
revealRevealOfWar)	Boolean	Whether the revealer reveals the reveal-of-war
radius	Float	The radius for revealing fog/reveal-of-war in grid cell measurements

8.3 – Server Database SQL Data Dictionary

8.3.1 - Users

Data Name	Data Type	Data Description
user_ID	Int	The user's unique identifier.
username	String	Publically displayed username.
password	String	Bcrypt, salted password
email	String	Email address for password reset
token	String	The user's unique token
registered	Int	The unix time of registration
ip	String	The user's IP address
active	Int	Unix time of last activity

8.3.2 – Map Information

Data Name	Data Type	Data Description
map ID	Int	The unique map identifier
user_ID	Int	The user's unique identifier
map	String	The location of map for download
title	String	The maps title.
description	String	The maps description.
version	String	The maps version
checksum	String	The maps checksum (md5)
created	Int	Unix timestamp, date of creation.

8.3.3 – Map Comments

Data Name	Data Type	Data Description
comment ID	Int	Unique comment identifier
user ID	Int	User ID whoever commented
map ID	Int	Map id, map commenting on
comment	String	Comment on map
time	Int	Unix timestamp

8.3.4 - Downloads

Data Name	Data Type	Data Description
map ID	Int	The map's unique identifier
user ID	Int	The user's unique identifier

8.3.5 - Votes

Data Name	Data Type	Data Description
nap ID	Int	The map's unique identifier
user ID	Int	The user's unique identifier
score	Int	The user's rating for this map

9. API Reference

9.1 – API Details

URL:	http://kryptagame.com:3000/api
Input:	GET, POST
Output:	JSON
Server:	Node.js + express
Methods:	17
Description:	This API allows access to Krypta Online features.

JSON: JSON stands for JavaScript Object Notation, an output format that is easily readable by humans yet easy to parse and generate on computers.

POST: Post data is encoded within the message body, for the Krypta API we use the application/x-www-form-urlencoded header.

GET: Get data is encoded into the URL. For example /api/maps?foo=bar

token: The token is the user's unique identifier and also unique API access key.

9.2 – Accounts

Command	POST /login
Description	Login to Krypta Online and receive user unique token.
Inputs	username, password
Constraints	user must be registered
Outputs	JSON Object, unique user token

Command	POST /register
Description	Register a user with Krypta Online.
Inputs	username, password, email
Constraints	• username between 3-15 characters • password more than 6 characters • email valid
Outputs	JSON Object

Command	POST /changepass
Description	Change your password
Inputs	Newpass, oldpass
Constraints	Newpass more than 6 characters
Outputs	JSON Object, unique user token

Command	POST /changeemail
Description	Change your email
Inputs	newemail
Constraints	Newemail must be a valid email
Outputs	JSON Object, true or false

9.3 – Maps

Command	GET /maps
Description	Returns a list of maps a user has subscribed to
Inputs	Token as URL parameter
Constraints	Token must match with user
Outputs	JSON Object, array with user information

Command	GET /maps/id
Description	Returns details about a specific map.
Inputs	id as maps unique int identifier
Constraints	ID must match an actual stored map.
Outputs	JSON Object, array with map information

Command	GET /maps/id/rating
Description	Returns the average rating for this map
Inputs	ID as maps unique integer identifier
Constraints	ID must match an actual stored map.
Outputs	JSON Object with rating (float)

Command	GET /maps/id/comments
Description	Returns all comments left on this map.
Inputs	id as maps unique int identifier
Constraints	ID must match an actual stored map.
Outputs	JSON Object, array with map comments

Command	GET /maps/id/checksum
Description	Returns the map files checksum
Inputs	id as maps unique int identifier
Constraints	ID must match an actual stored map.
Outputs	JSON Object with checksum

Command	GET /subscribe/id
Description	Add maps to a user's collection
Inputs	id as maps unique int identifier
Constraints	ID must match an actual stored map.
Outputs	JSON Object true or false

Command	GET /unsubscribe/id
Description	Removes a map from a users collection
Inputs	id as maps unique int identifier
Constraints	ID must match an actual stored map.
Outputs	JSON Object true or false

Command	GET /rate/id/score
Description	Adds a rating to the map
Inputs	• :id as maps unique integer identifier • :score as map score • token as URL parameter
Constraints	• id must match an actual stored map • token must match with user • score must be between 1 and 5
Outputs	JSON Object true or false

Command	POST /comment/id/
Description	Adds a comment to the map
Inputs	• :id as maps unique integer identifier • :score as map score • token as URL parameter
Constraints	• id must match an actual stored map • token must match with user • comment must be between 1 and 480 characters
Outputs	JSON Object true or false

9.4 – Site

Command	GET /active
Description	Returns a list of active players in the last 24 hours
Inputs	• N/A
Constraints	• N/A
Outputs	JSON Object

Command	GET /game
Description	Returns game name name, checksum and version
Inputs	• N/A
Constraints	• N/A
Outputs	JSON Object

Command	GET /editor
Description	Returns a list of active players in the last 24 hours
Inputs	• N/A
Constraints	• N/A
Outputs	JSON Object

Command	GET /online
Description	Returns a list of currently active players
Inputs	• N/A
Constraints	• N/A
Outputs	JSON Object

Command	GET /stats
Description	Returns a count of players and unique maps
Inputs	• N/A
Constraints	• N/A
Outputs	JSON Object

10. Game Design

10.1 – The Player

The character controlled by the user. The player interacts with elements within the environment to achieve a goal. Elements provide as obstacles to challenge the player. The attributes of the player are:

- Speed (Rate of movement)
- Health (Alive, Dead, Poisoned)
- Vision (Radius)

Other attributes:

- Gold (Score)
- Keys (Generic interaction with elements)
- Tools (Specific interaction)

Potential attributes:

- Noise
- Food
- Ammunition (Bow and Arrow / Slingshot Concept)

10.2 – The Tools

Tools are items that are used by the player and have specific functions. They are limited to the player, a maximum of two:

- Torch (Increase Vision Radius, Interact with specific elements e.g. Burn spider web, light wall torches, etc.)

- Boots (Increase rate of movement, i.e. Speed. Another benefit, if noise implemented, reduce tile creak, OR increase gold chance factor, call Lucky Boots)

- Shield (Defensive combat tool, safeguards for one lethal interaction, then destroyed. Isn't destroyed by projectiles.)

- Sword / Bow / Slingshot (Offensive combat tool, kill on successful interaction. Needs testing and balancing. Can interact with certain elements, e.g. Switches, Webs etc.)

- Bird (Deployable Scout with a time limit of flight. Can interact with certain elements, ignores floor based traps, perishable to combat, poison gas, etc) [Chirp on click to make noise and distract if noise implemented?]

10.3 – Tiles

Tiles are the base level element which supports a level of gameplay. Its current attribute is only:

- Texture

If noise is implemented, potential to have a CREAK value associated. This can easily be done with Tile/Entity collision handling, on first collision, have a chance to creak.

10.4 – Wall

Walls are the physical barrier that shapes the skeleton of the gameplay environment. Current attributes:

- Texture

Future attributes:

- Transparent / Window (Allow vision travel through the wall to reveal fog of war)

10.5 – Traps

Traps are essentially mostly static elements which negatively impact the player. They rely on a TRIGGER to take on an EFFECT. They can be triggered in numerous ways, for example on touch, or being switched on elsewhere (pressure plate). Effects can range as follows:

- Immediate Kill (Bear Trap)
- Projectile (Darts)
- Delayed Kill (Floor spikes, have a fuse before rising for example)
- *** Noise Alarm (pot smashes onto the floor)
- Pit of Death, similar to bear trap, but without an end state, always live. Maybe warp you to floor below, then die, unless you have boots. Or warp to lower floor on top of spikes.
- Falling Objects, directional death. React to environment. E.g. Wall collapses.
- Risk Choices, e.g. Snake basket.
- Poison gas released.

10.6 – Items

Elements that (for the most part) aid the player, or provide some game dynamic.

- Poison (X seconds to live, reverse movement control, blind player [Fog of war is restored and cannot be revealed for the duration of the poison])
- Potion (Cure poison, increase attribute, indestructible, ghost mode [move through walls]
- Switch / Pressure Plate (Interact with to alter another unknown element)
- Chests (Contain benefit, like gold, but require unlocking)
- Risk Chest (Do not require unlocking but may have a negative effect, need to pay close attention to value correctly, e.g. snake basket rustle on first view)
- Door (Progress to another area, but requires unlocking)
- Boulder (Push to encumber pressure plates / block routes)
- Wall Torch (may or may not be pre-lit, but will keep the corresponding fog of war revealed. Acts as a vision tower. Torch tool can light them)

10.7 – Enemies

Have a number of different purposes, each with the purpose to challenge the player in some combat scenario. The three basic enemies for the game will be:

- Mummy (Slow, have Roam and Hunt behaviour, one hit kill on player)
- Snake (Result of risk, immediate non-lethal effect)
- Scarab (Quick, triggers traps and steals gold on touch. Drops stolen gold on death.)

Each of the three basic enemies can be used in different situations to continue challenging the player. For example, some enemies might have a static path that doesn't hunt the player down but moves faster, or a dynamic roaming enemy.

10.8 – Puzzles

Abstract interactions that challenge the player need to be balanced. Not too easy, or difficult. It should provide the user with a sense of accomplishment.

- Distant trap triggers
- Multi-floor interaction (i.e. not immediately visual)
- Timing based actions, e.g. avoiding patrols
- Mini-games, i.e. scarab key idea
- Combat scenarios, lure into traps, or use offensive tools.

- Search objectives, commit player to a section of decision tree with a fork in the road etc.
- Escape poison gas that's spreading
- Boulder moving puzzle.

11. User Interface Design

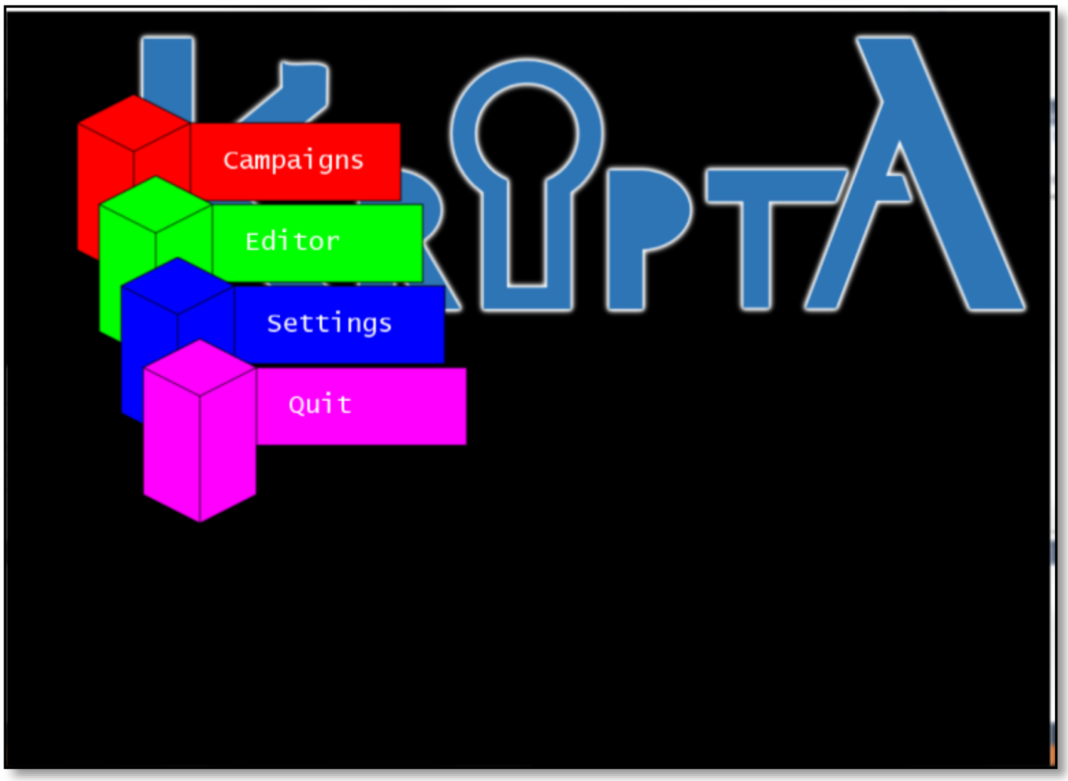
The aim for the user interface design for Krypta was to keep everything as simple and minimalistic as possible. This concept was to be applied to all aspects of Krypta in order to attempt to make it simple, quick and easy to use.

For user interface design the aim is to keep it simplistic and minimalistic for the implementation within the Krypta game.

11.1 – Krypta Game:

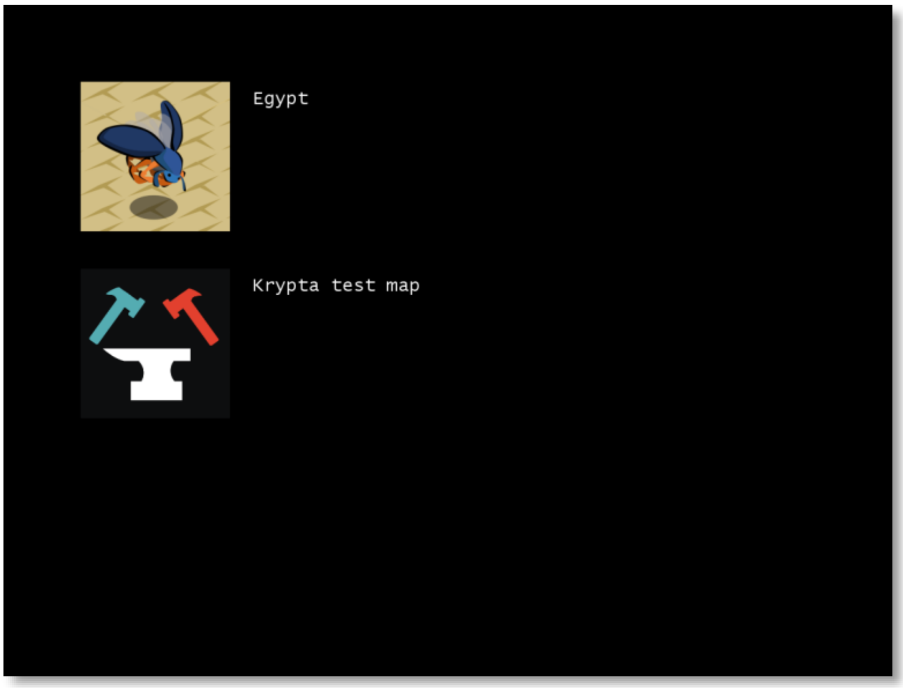
Main Menu

For the main menu screen, the user interface is very simple. The user is presented with four options, when the user hovers over an option the four towers a ribbon will expand out to the right with the option title as pictured.



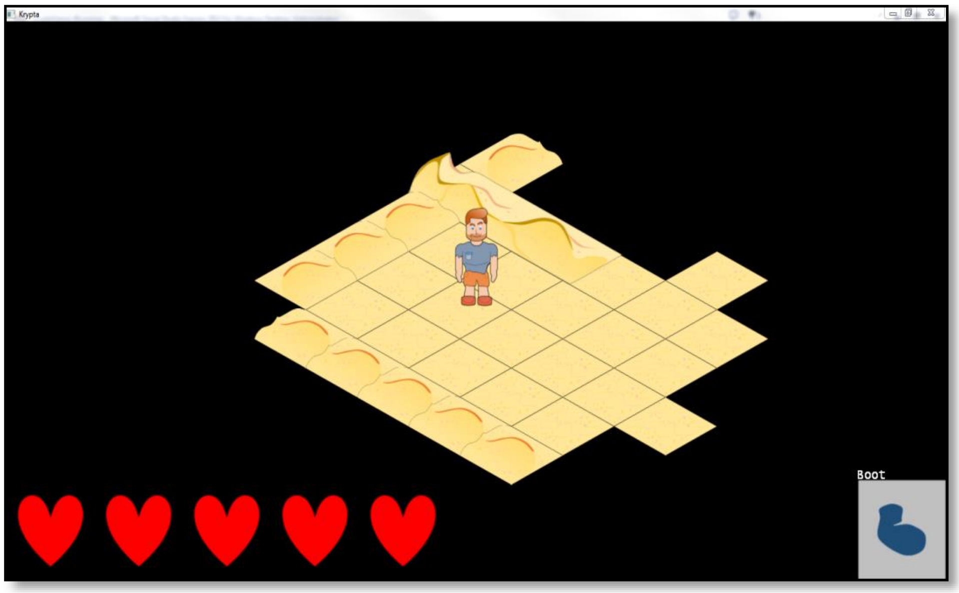
Map Select

The map select screen was achieved through a simple system that acts like a "table". The interface is designed with a scrollable list of each campaign that is found. For each campaign is an icon and a title / description of the campaign package.



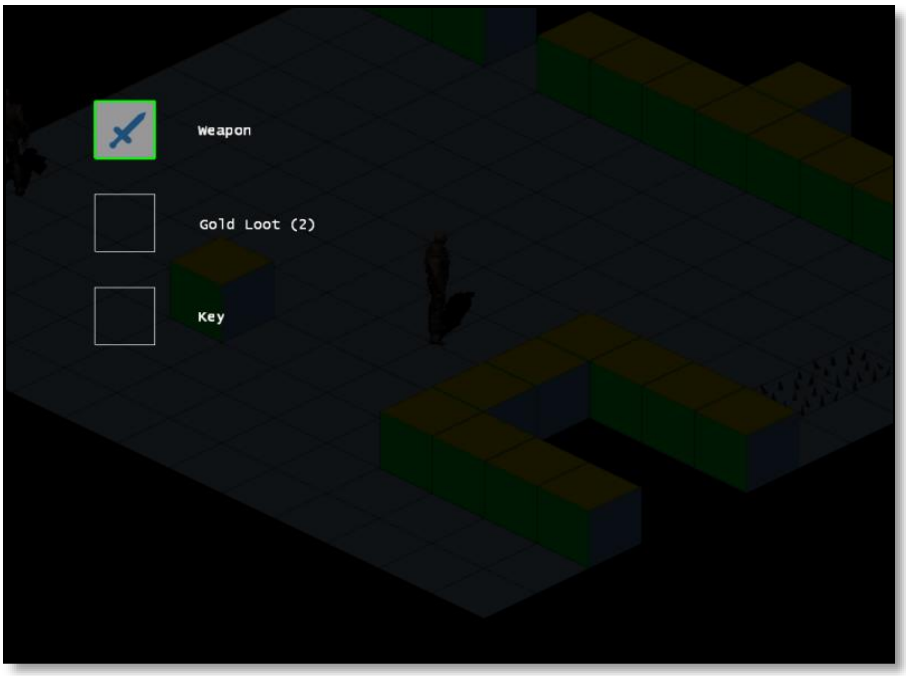
In Game GUI

The only necessary pieces of information that will be required to be displayed are health and the equipped tool for the user. The bottom right hand corner provides you with a view of your tool that you currently have equipped and the bottom left displays the player's health.



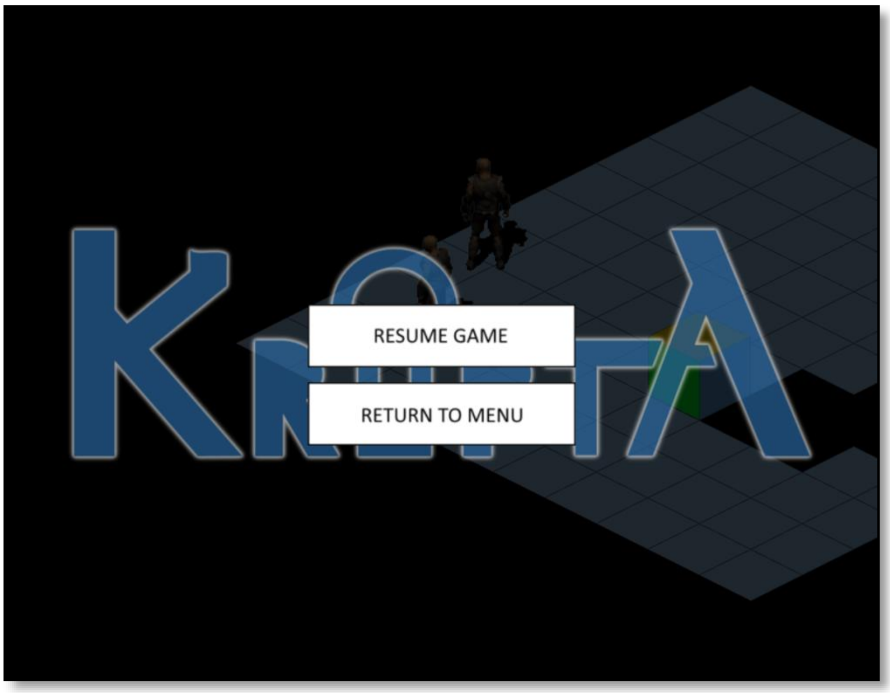
Inventory

The user can also access their inventory simply by pressing TAB which simply displays the item icon and the quantity that they currently have in their possession.



Pause Menu

If at any time the user wishes to pause the game they can do so by simply clicking ESC. This will bring up a simple interface with two options of either resuming the game or returning to the main menu.



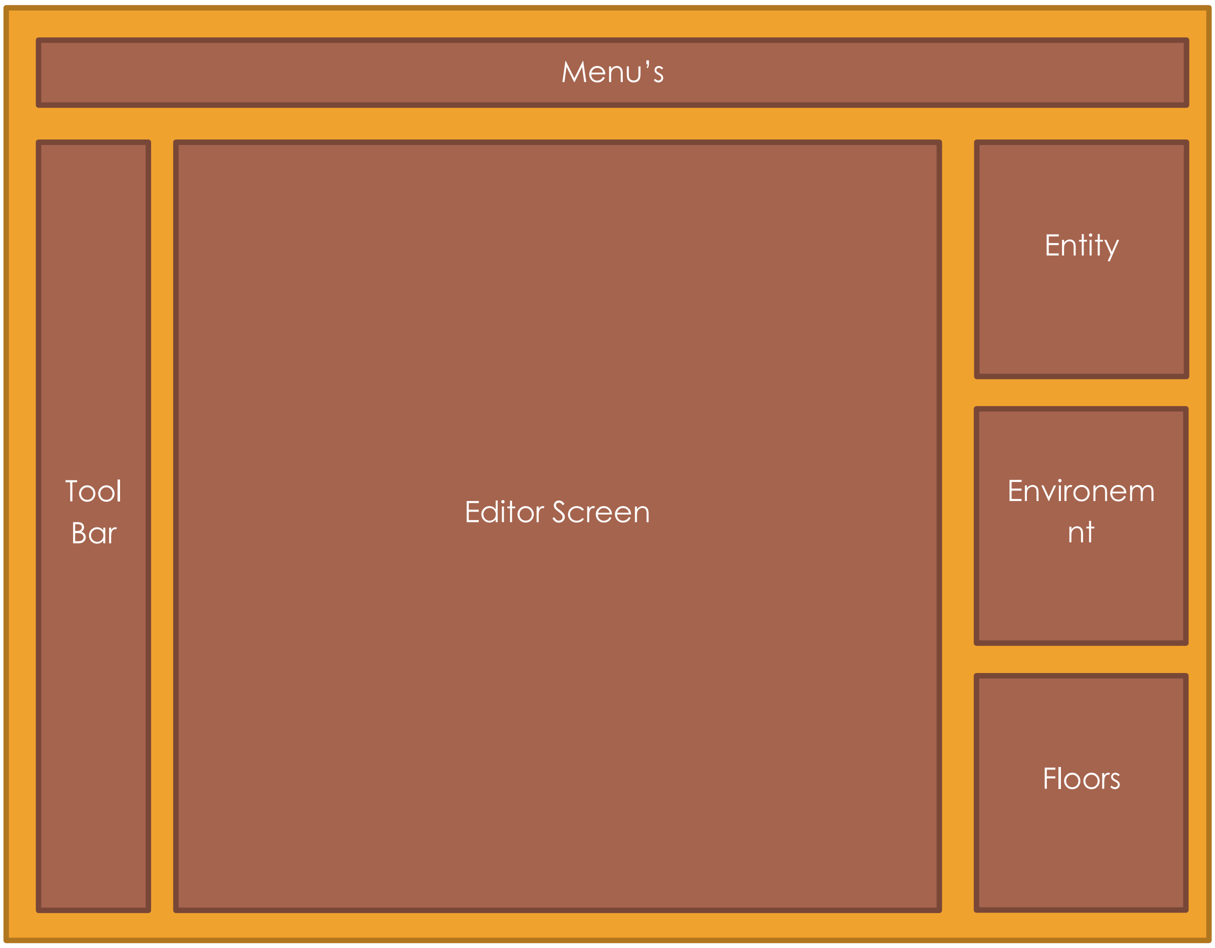
11.2 – The Map Editor:

The overall interface design of the map editor was set to be very simplistic and aimed to achieve a very standard layout that would be implemented by a lot of programs so that users would find navigation very easy.

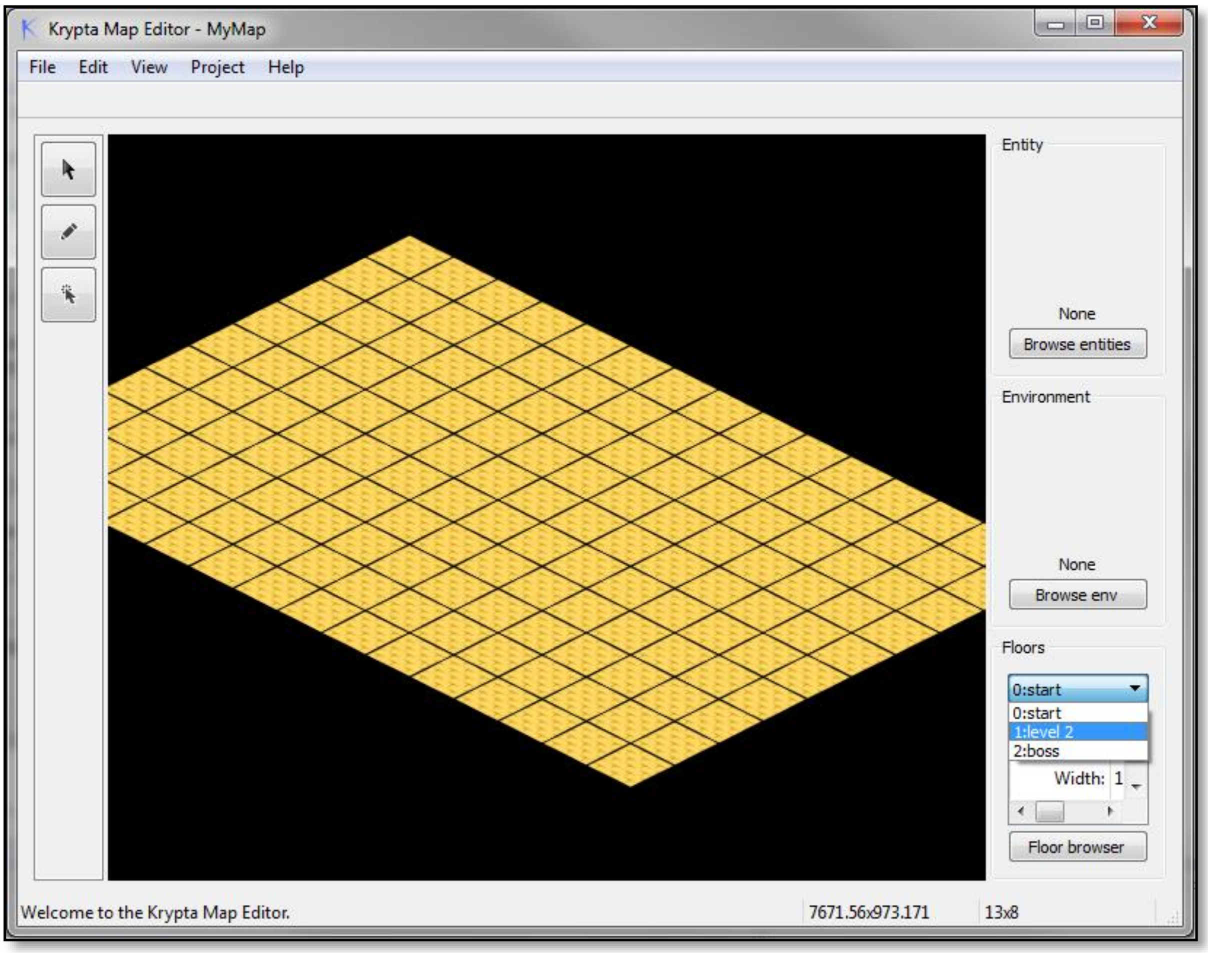
The base Map Editor is very simplistic and is able to be navigated like most standard programs such as Notepad. To create a new map, simply click File → New map. Just like creating any new document. The GUI houses many "standards" of a GUI including:

- Exit in right hand corner.
- Drop down boxes
- Buttons
- Text fields

The prototype editor layout that was designed was to be laid out as follows:



The final layout and design of the Map Editor is very close to the prototype dratted layout.



Welcome to the Krypta Map Editor. 7671.56x973.171 13x8

The Menu's

At the very top along the screen is the menu bar for the program. This is where all of the important and general functions of the program are kept. As one would expect under File are options such as New File and Save. All functionality that effects the Editor as a whole as well as general functions can all be found under one of the five drop down menus available.

The Tool Bar

The Toolbar shown down the left hand side of the editor provides three options that can be used to manipulate the map:

- **Pointer:** basic click tool.
- **Pain**t: paint something from the environment / entity browsers.

- **Select:** select large groups of entity's and environments to mass copy, edit or remove.

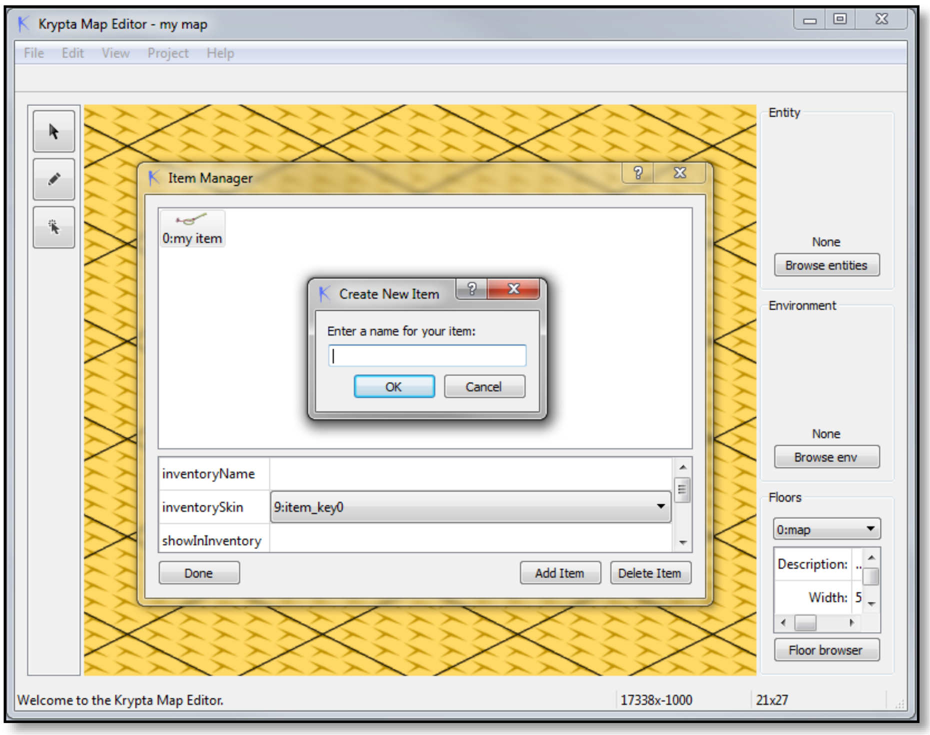
These options act very similar to Microsoft Paint or Photoshop when manipulating the map.

Editor Screen

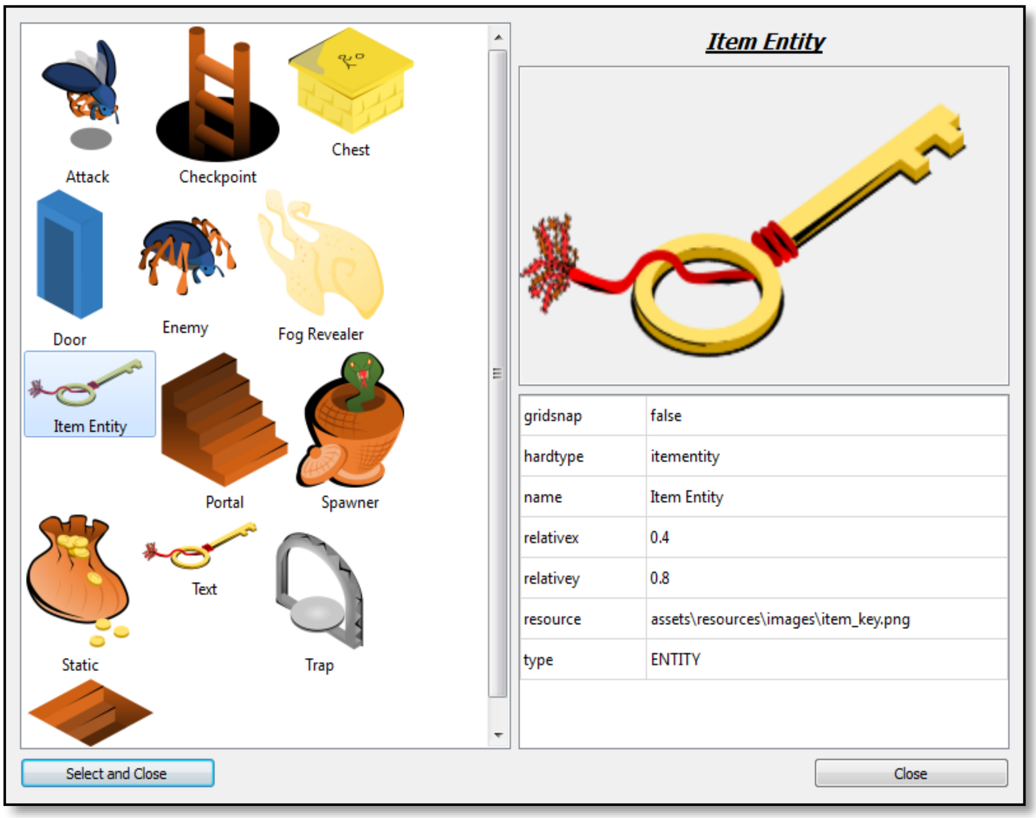
The Editor Screen is the visual representation of what is currently placed on the custom map. Users are able to manipulate data through this screen by dropping, dragging, and selecting entities, environments and items to be placed down. On this screen the user can right click anything that has been placed down and modify its properties and adjust them how they wish.

The Browsers

The Entity and Environment boxes on the right hand side of the screen provide a preview on the user's current selected entry. There are multiple browsers that have been created for the entities, the environments and the items that are available for use within the game. Each browser will allow the user to either select one of the pre-existing entry's or create a new entry with all of their own assets.



Below is an example of one of the multiple browsers that is featured throughout the editor. If the user wishes to add the item to their map, they simply click Select and Close.



gridsnap	false
hardtype	itementity
name	Item Entity
relativex	0.4
relativey	0.8
resource	assets\resources\images\item_key.png
type	ENTITY

Floors

The floors component of the user interface in the bottom right hand corner allows the user to see the current hierarchy of the floors that are associated with their map. Users are able to use this to jump between the different floors in their map through this option.

12. Algorithms

All of the code throughout the project has been standard in regards to implementation and has been developed solely by the Krypta team. The one instance in which another algorithm has been researched and adapted for the Krypta project is from a research paper titled "A Fast Voxel Algorithm for Ray Tracing" by John Amanatides and Andrew Woo.

The reason for use of this algorithm was to ensure that the field of vision wouldn't become visible around corners and projected from the player's line of sight.

The abstract of the paper is as follows:

> "A fast and simple voxel traversal algorithm through a 3D space partition is introduced. Going from one voxel to its neighbour requires only two floating point comparisons and one floating point addition. Also, multiple ray intersections with objects thatare in more than one voxel are eliminated."

The algorithm that was used from this paper is as follows:

The incremental phase of the traversal algorithm is very simple. The basic loop is outlined below:

```
loop {
        if(tMaxX < tMaxY) {
                tMaxX= tMaxX + tDeltaX;
                X= X + stepX;
        } else {
                tMaxY= tMaxY + tDeltaY;
                Y= Y + stepY;
                }
        NextVoxel(X,Y);
        }
```

We loop until either we find a voxel with a non-empty object list or we fall out of the end of the grid. Extending the algorithm to three dimensions simply requires that we add the appropriate z variables and find the minimum of tMaxX, tMaxY and tMaxZ during each iteration. This results in:

```
list= NIL;
do {
        if(tMaxX < tMaxY) {
                if(tMaxX < tMaxZ) {
                X= X + stepX;
                if(X == justOutX)
                return(NIL); /* outside grid */
                tMaxX= tMaxX + tDeltaX;
        } else {
                Z= Z + stepZ;
                if(Z == justOutZ)
                return(NIL);
                tMaxZ= tMaxZ + tDeltaZ;
        }
} else {
        if(tMaxY < tMaxZ) {
                Y= Y + stepY;
                if(Y == justOutY)
                return(NIL);
            tMaxY= tMaxY + tDeltaY;
            } else {
                Z= Z + stepZ;
            if(Z == justOutZ)
                return(NIL);
            tMaxZ= tMaxZ + tDeltaZ;
            }
        }
        list= ObjectList[X][Y][Z];
        } while(list == NIL);
return(list);
```

The complete paper can be found at: **http://www.cse.yorku.ca/~amana/research/grid.pdf**

13. Testing

Testing is a huge fundamental component of any Software Development cycle and often one of the most expensive. Due to the nature of the Krypta project a whole variety of different testing methods will need to be applied.

13.1 – The Game

During the development of the game, an incremental testing approach was adopted. Every time a new feature was added to the game extensive testing was performed using a test map to ensure that the new feature was performing correctly. Not only that but testing was focused on making sure the implementation had not compromised the functionality of previous features in any way.

13.2 – The Map Editor

In order to test the maps that are created are correct there are testing methods that have been designed to prevent users from creating their own maps that are unsolvable. Due to the nature and high complexity of user generated maps, automated testing cannot always be applied.   Three methods have been designed in order to aid in the testing and filtering of defective maps.

13.2.1 – Path Finding

The first implementation for testing that was used was by creating an AI script that used path finding to see if it was able to find the end of the dungeon. This process is beneficial as it can easily be automated but taking account for multiple floors and requirements such as keys becomes rather troublesome and difficulty compromises the effectiveness of this script as it increases.

The second implementation for testing that has been designed, especially in the case of user uploaded maps for the Krypta Community is that all maps must first pass a user test. The basis of a user test is that after the user has completed their map that they wish to upload they must first complete the map as a means of verification to prove that the map is indeed solvable.

13.3 – The Web Server

In order to ensure that the Web Server could handle traffic and performed as necessary for for the Krypta project, the network was tested extensively.

Multiple stress tests were performed against the website in order to gain statistics on performance and how it handled high demand and load.

The following results are from a stress test:

- Requests: 91
- Errors: 0
- Requested URL Avg Response Time (Secs)
    - http://kryptagame.com:3000/api/login ------ 0.324
    - http://kryptagame.com:3000/api/editor ----- 0.003

A big method such as the login function takes longer than the smaller request of the Editor which is almost instantaneous but can still handle a large number of requests. It's important to note that one of the reasons the log in function takes longer to respond is because it has to use bcrypt.

14. Requirements

14.1 – Scale

Core – The core requirements are mandatory before any by-product of the framework can be initiated.

High – The requirements labelled as a high priority must be completed for the Krypta project to run correctly. It is vital to the project.

Medium – The medium requirements aren't necessary for the game or project to work but they're an addition that will be required to make the game interesting and have a larger range of functionality.

Low – The low requirements aren't important to the project but would be nice additions that would extend onto the project.

Post Release – Post release goals are highly unlikely to make it into the final product, but are ideas that can be worked on after release in order to keep the product fresh and supported.

Requirement ID – The requirement ID is a quick key guide to the component of the system and the number of the requirement in the following requirements.

14.2 – Summary

The following tables are an overall summary of our requirements for the project and their status:

- Green indicates completed
- Yellow indicates a work in progress
- Red indicates not implemented
- Black indicates removal from project

14.2.1 – Framework

Req. ID	Requirement	Priority	Completed
F1	Graphics	Core	🟩
F2	Networking	Core	🟨
F3	Threading	Core	🟩
F4	Windows Support	Core	🟩
F5	Audio	High	🟩
F6	File System Support	Medium	🟩

14.2.2 – Map Editor

Req. ID	Requirement	Priority	Completed
M1	Export and Import	High	🟩
M2	Editor Entity Settings	High	🟩
M3	Editor Global Map Settings	High	🟩
M4	GUI for Editor	High	🟨
M5	Visual Click Editing	High	🟩
M6	Basic Artificial Intelligence	Medium	🟩
M7	Upload and Download from Editor	Medium	🟩
M8	Mods, Extensions and Plug - Ins	Low	🟩
M9	Team Based Editing	Low	🟥
M10	Heat Map for Statistics	Post Release	🟥
M11	Live Play	Post Release	🟥

14.2.3 – Game

Req. ID	Requirement	Priority	Completed
G1	Enemies	High	🟩
G2	Game Settings	High	🟥
G3	Grid System	High	🟩
G4	Movement	High	🟩
G5	Obstacles and Traps	High	🟩
G6	Path Finding	High	🟩
G7	Pre-made Maps	High	🟨
G8	Save and Load Game-Play	High	🟥
G9	Achievements	Medium	🟥
G10	Collectables	Medium	🟥
G11	Consumables	Medium	🟥
G12	Experience System	Medium	🟥
G13	Fog of War	Medium	🟩
G14	Game Play Statistics	Medium	🟥
G15	GUI for Game	Medium	🟨
G16	Items and Weapons	Medium	🟥
G17	Lighting Systems	Medium	🟥
G18	Map Replays	Medium	🟥
G19	Money and Points System	Medium	🟥
G20	Objectives	Medium	🟥
G21	Particle Effects	Medium	🟥
G22	Screen Effects	Medium	🟥
G23	Smart Artificial Intelligence	Medium	🟥
G24	Vendor NPC	Medium	🟥
G25	Controller Support	Low	🟥
G26	Cut Scenes	Low	🟩
G27	Multi-Platform	Low	🟥
G28	Multi-Player	Low	🟥
G29	NPC Interaction and Dialogue	Low	🟥
G30	Physics	Low	🟥
G31	Story	Low	🟥
G32	Theme Mod	Low	🟥
G33	AI that Learns	Post Release	🟥
G34	Production Music and Theme Song	Post Release	🟥

14.2.4 – Online Community [ask sam for this stuff]

Req. ID	Requirement	Priority	Completed
W1	Server	High	🟩
W2	User Accounts	High	🟩
W3	Wiki for Framework	High	🟩
W4	Data Resource Archiving	Medium	🟩
W5	In Game Browser	Medium	🟥
W6	Link to Website	Medium	🟩
W7	Website Community Features	Medium	🟩
W8	Wiki for Game	Medium	🟩
W9	Website Forum	Low	🟩
W10	Merchandise	Post Release	🟥
W11	Twitter Integration	Post Release	🟥

Functional Requirements

The following functional requirements are requirements that must be met in order for the software to work.

14.3 – Framework

F1	**Graphics**	**Core**
Graphics include textures and rendering methods, used in both the game and user interface elements.		

F2	**Networking**	**Core**
Networking provides the ability to send data from system to system via a local or internet connection. It will allow the transferring of maps and community data.		

F3	**Threading**	**Core**
Threading involves the use of the Windows API for areas of the framework that may need to make use of threading, such as file I/O operations, or networking.		

F4	**Windows Support**	**Core**
Basic window creation/destruction with event handling, used to display everything, from user interfaces to the game’s graphics.		

F5	**Audio**	**High**
Audio involves the loading/playing of audio files.		

F6	**File System Support**	**Medium**
An extension to C++’s standard library’s file handling, allowing basic interaction with files/directories without using the standard library (e.g. checking existence, moving files, etc.).		

14.4 – The Game Client

G1	Enemies	High
Enemies involve NPCs (non-player characters) who, like obstacles or trap, may kill or hinder the player. However, enemies are, unlike traps, non-static, and may move around in pre-determined/random paths.		

G2	Game Settings (Video, Audio, Key Bindings)	High
Provide a game setting menu where the user can control video and audio output, as well as choose key bindings for controls.		

G3	Grid System	High
The grid system will store tile based information such as structure type (floor/wall/door), and any other items that may layer on top of the tile such as traps/items/guards/goals. This system will be used for navigation, path finding and visual representation of the isometric dungeon.		

G4	Movement	High
The movement for the player will be real-time and free roam via the mouse, clicking will move the character toward the target position after path finding.		

G5	Obstacles and Traps	High
Obstacles and traps are the in-game, static, complications that a player must deal with. They can kill (force to restart) or hinder the player, depending on the severity of the obstacle or trap.		

G6	Path Finding	High
This feature may be used by guards who can respond to any points of interest on the map. It may also be used for evaluating the map for the editor, to test if it is possible to reach the end/goals.		

G7	Pre-Made Maps	High
Pre made maps are a select few maps that are available for all users to play as default. These maps are bundled in the program and are available in offline play.		

G8	Save and Load Game-Play	High
The ability to be able to start and stop gameplay without any loss of information, i.e. save and load game states.		

G9	Achievements	Medium
Implement an achievement system to add a more challenging aspect to the game i.e. Money Bags – loot $100,000 gold.		

G10	Collectables	Medium
Collectables may serve as items that award points for collection, or pure aesthetic appeal. Collectables are only provided by the maps provided by the developers and are not available in the dungeon editor. Collectables can be tracked in the menu, for players to view their collectable progress.		

G11	Consumables	Medium
Consumables are items that the player can pick up and store temporarily in their inventory (these do not carry over to other plays). Consumables are items that may be used once, such as health.		

G12	Experience System	Medium
A representation of a player's collected points in a single map (a high score) for completing objectives. Reset at the beginning of each play-through.		

G13	Fog of War	Medium
To enhance suspense of game-play, this feature will provide a limited radius of vision that will reveal the dungeon as it is explored, and leave any unexplored areas black. However, guards may only be visible within the radius of the player.		

G14	Game Play Statistics	Medium
The tracking, calculating, storing and displaying of any game related statistic to the user. Depending on the statistics at hand the method for displaying may be via a table of values, raw value, or part of the interface.		

G15	GUI for Game	Medium
The game may contain a user interface to represent items that the main character is carrying, or effects that have been imposed on the character, but more importantly, menu's and options for the game.		

G16	Items and Weapons	Medium
You may be able to gather items for points, or they may have specific functionality such as traps. Weapons may be used to attack guards. You may choose certain items and weapons before each attempt at the dungeon that you may use.		

G17	Lighting Systems	Medium
A dynamic lighting system which changes how far the user can see depending on objects such as torches and other light emitting entities.		

G18	Map Replays	Medium
Allow the user to replay maps that they have already completed.		

G19	Money and Points System	Medium
A form of (in-game) currency to be used to buy (in-game) disposable items/equipment for an advantage in the current map.		

G20	Objectives	Medium
The dungeons may have various objectives in order to complete the dungeon. These options are provided to the dungeon editor to decide how the dungeon goal should be achieved. These objectives may include items that require collecting (before the exit allows you to finish), guards that need neutralizing, a minimum amount of points required, or simply to get to the exit set by the dungeon designer.		

G21	Particle Effects	Medium
Particle effects can increase the games look with many small sprites simulating things such as fire, or damage taken from a trap.		

G22	Screen Effects (Shader)	Medium
Develop and implement graphical effects such as shaders to implement into the game for aesthetics and representation purposes.		

G23	Smart Artificial Intelligence	Medium
A smart AI is developed to help test a map to make sure that a map is not impossible to complete, and could also be integrated into NPCs as more of a challenge to the player.		

G24	Vendor NPC	Medium
An in-game entity which provides the user with options to trade in game items/currencies, to create a more user defined game play experience.		

G25	Controller Support	Low
Allow controller to be used in place of keyboard and mouse, as some users may prefer to use a controller and in general increases our accessibility.		

G26	Cut Scenes	Low
Pre-rendered videos that may use assets from the game, while taking less processing power, and giving the game a more polished look, while real-time cut-scenes (in-game) are easier to script, with the same polish as pre-rendered cut-scenes.		

G27	Multi-Platform	Low
Dungeons may be able to play cooperatively with other players; this may involve solving puzzles specifically designed for more than one player.		

G28	Multi-Player	Low
Audio involves the loading/playing of audio files.		

G29	NPC Interaction and Dialogue	Low
The communication with in-game entities, with the purpose to convey information, offer decision trees, or to propel potential plotlines. The thought on communication here is via on screen text, not pre-recorded audio.		

G30	Physics	Low
Item and entity physics to act similar to real world scenarios.		

G31	Story	Low
A piece of fiction that sets the tone and environment for the product, whilst also offering a plotline for the user to follow, progress and navigate through. The thought here is that a dynamic plot would be used, being that the actions/decisions of the user have an impact on the plot.		

G32	Theme Mod	Low
Giving the user the ability to be able to implement their own custom theme/skin to the interface of the product. There will be a set standard/guideline for this feature described within the Wiki for the game.		

G33	AI that Learns	Post Release
An AI that learns how to complete the map with the best possible solution.		

G34	Production Music and Theme Song	Post Release
Audio of our own design to give a unique touch to the product, an attribute in a different medium that the user can identify with the product easily.		

14.5 – Map Editor

M1	**Export and Import**	**High**
The ability for the Map Editor to both import and export our custom file extension associated with data necessary to construct a map within the Map Editor.		

M2	**Editor Entity Settings**	**High**
Entities can be set with specific settings to alter how they work, for example: a door may have a setting that requires a key to open it.		

M3	**Editor Global Map Settings**	**High**
Implementation of global map settings including objective options and gameplay settings i.e. fog of war, speeds etc.		

M4	**GUI for Editor**	**High**
Considering the editor is a visual representation of a map's design, a user interface is required to provide the tools to help the user develop and edit maps.		

M5	**Visual Click and Drag Editor**	**High**
Implementation and functionality of a visual drag and drop function for the map editor for ease of use for building maps.		

M6	**Basic Artificial Intelligence**	**Medium**
Within the Map Editor, users will be able to confirm their custom map as valid with the use of this tool. This tool will utilise basic path finding and brute force techniques to function. No heuristics are planned for this tool.		

M7	**Upload and Download from Editor**	**Medium**
The ability to browse/add and get user created maps from within the Map Editor interface. This involves the user uploading to a server, downloading from said server, and viewing other stored map within the server via the interface.		

M8	**Mods, Extensions and Plug Ins**	**Low**
Ability to allow users to create extensions to our game, such as new traps, entities and themes etc.		

M9	**Team Based Editing**	**Low**
Multiple people working on a map at the same time, either on a LAN or WAN network, and being able to see each other's contributions to the development in real-time.		

M10	Heat Map for Statistics	Post Release
Implementation of a “heat map” to provide a visual representation of where users are being caught / running into traps on user created maps.		

M11	Live Play	Post Release
An integration of the game into the editor. A player may edit a map, and start to play the game from a desired position to quickly test a feature.		

14.6 – Online Community

W1	**Server**	**High**
The server controls the database that stores all user and map information related to the game and web-site.		

W2	**User Accounts**	**High**
Provide storage and creation of user accounts so users can join the community, create maps, progress on their character etc.		

W3	**Wiki for Framework**	**High**
An online hub for documentation and discussion in regards to the custom built framework associated with our 'Krypta' product. This tool is directly linked to the website of the product.		

W4	**Data Resource Archiving**	**Medium**
The server's ability to ensure that data is being saved and archived correctly and be accessed correctly.		

W5	**In Game Browser**	**Medium**
A small, online map browser within the game encouraging users to join the community.		

W6	**Link to Website**	**Medium**
Provide a live website and links to the website within the game to encourage users to join the community.		

W7	**Website Community Features**	**Medium**
The game may contain a user interface to represent items that the main character is carrying, or effects that have been imposed on the character, but more importantly, menu's and options for the game.		

W8	**Wiki for Game**	**Medium**
An online hub for documentation and discussion in regards to all aspects of gameplay associated with our 'Krypta' product. This tool is directly linked to the website of the product.		

W9	**Website Forum**	**Low**
Community based discussions can take place on the website in an organised space.		

W10	Merchandise	Post Release
Turn the game into a brand name with plushies, t-shirts and other goodies.		

W11	Twitter Integration	Post Release
Post high-scores and rare collectables notices to twitter to expand coverage.		

15. Code Conventions

15.1 - Formatting

Most conventions under formatting are purely cosmetic, and are either chosen as a preferred style for personal reasons, or for readability reasons.

15.1.1 – Spaces or Tabs:

Tab characters should be used instead of space characters, and set to an indentation width of 4 spaces.

15.1.2 – White Space:

Generally, white spaces increase the readability of code, and should be used as often as it is sensible. New-lines should be used to separate code only around functions, classes, or large blocks of code, in the same way as you might use them to separate paragraphs of text. Examples:

```
Good:

int foo = 1 + 3 * MYMACRO, a, b, c;

class MyClass : public MyDerived
{
};

void bar()
{
    if (foo > 10)
        cout << "text\n";
}

Bad:

int foo=1+3*MYMACRO,a,b,c;

class MyClass:public MyDerived
{
};

void bar()
{
    if(foo>10)
        cout<<"text\n";
}
```

15.1.3 – Bracing:

The style of bracing depends on the situation in which it is used. Inline functions should have their braces placed on the same line as the declaration, if the function is defined where it was declared. In all other cases, each brace is placed on its own line, starting from the line after the declaration/definition. This helps to avoid clutter where it makes sense, and improves readability for large spanning sections of code.

Example:

```
inline void print() { cout << "text\n"; }

class MyClass // each on their own line
{
};

void foo()
{
    cout << "bar\n";
    // imagine a lot more code here
}
```

15.1.4 – Comments:

Single line comments are denoted using //. Multi-line comments are denoted using /* */. And descriptions are denoted with /** */.

15.1.5 – Naming styles:

Upper camel case dictates how namespaces, classes, template arguments, and enums, are to be named, while lower camel case dictates how functions/methods are named. Local or member variables should be all lower case, while any constants (including macros/enum values) should be all upper case. Whether a method, or data, is private/protected/public or not does not influence its notation. Spaces should be replaced with underscores.

All names should be based on their respective functionality, without becoming too verbose. Examples:

```
#define MY_MACRO
const int MY_CONSTANT = 0;

namespace MyNamespace
{
    class MyClass
    {
        private:
            int privatedata;
        protected:
            int protecteddata;
        public:
            int publicdata;

            void publicMethod(int param)
            {
                int localvar;
                const char LOCAL_CONSTANT;
            }

            inline int getPublicData() { return publicdata; }
            inline int getPublicDataFromMyClass() { return
publicdata; } // too verbose
    };

    enum class MyEnum
    {
        MY_ENUMVAL // also a constant
    };
}
```

15.1.6 – Namespaces/Classes:

Namespaces should indent anything inside it, to help separate globals from code that resides inside namespaces. Nested namespaces should also be indented/indent. This helps to increase readability. Classes should do the same, as well as their access specifiers. Example:

```
namespace NamespaceOne
{
    void doOne();

    namespace NamespaceTwo
    {
        void doTwo();

        Class MyClass
        {
            private:
                int mydata;
        };
    }
}
```

15.1.7 – Functions/Methods:

In some cases, functions may contain more than a few parameters, and may stretch past the edge of the screen/editor. In these cases, wrapping the parameters neatly, with some alignment, should help to increase readability. Placing each parameter on its own line is unnecessary. Example:

```
void doSomething(int param1, int param2, int param3, int param4
                 int param5, int param6, int param7, int param8)
{
}
```

15.1.8 – Switch statements:

Each case should stand on its own line, followed by their respective code. If a case requires a block using braces, the braces should also be indented, as well as code inside the brace label. Example:

```
switch (myint)
{
    case 1:
        doSomething();
        break;
    case 2:
        {
            char var = 'a';
            doElse(var);
        }
     break; // break on this line, indicating end of block and case
}
```

15.1.9 – Empty loops:

Empty braces should follow an empty loop to help increase its visibility, and so that it is not mistaken for a non-empty block. Example:

```
for (int i = var; isSingleDigit(i); ++i, ++count) {}
```

15.1.10 – Pointers/References:

A white space is placed after the pointer/reference operator whenever it is used (not to be confused with a logical/bitwise AND operator, or multiplication operator). Example:

```
int* intptr;
void doSomething(const MyClass& myclass);
```

15.1.11 – Templates:

Template arguments should follow the same conventions as function parameters. Each argument should be placed on the same line, unless too many arguments are present.

In the case of whether or not to use "typename" or "class", "typename" should be used, to avoid any possible problems along the way. The entire template part should be on its own line(s). Example:

```
template<typename T>
void foo(T & var);

template<typename Arg1, typename Arg2, typename Arg3,
         typename Arg4, typename Arg5, typename Arg6>
class MyClass
{
};
```

15.2 – Features

Features explain the use, or lack of use, of features specific to C++ and C++ compilers.

15.2.1 – Sized integer types:

Types of specific sizes, such as int8_t, etc., come from the standard library, and should not be mixed with types from external libraries for safety reasons.

15.2.2 – Run-Time Type Information (RTTI):

RTTI should not be enabled during compilation as most code can be achieved without it (in the case of this framework at least). It is also an unnecessary increase in the size of the compiled executable.

15.2.3 – 64-bit Compatibility:

As the framework is exclusive to 32-bit architecture, 64-bit compatibility is not imperative. However, it may help make code more portable in the future if 64-bit is kept in mind.

15.2.4 – C++11 or C++03<:

As C++11 is the latest version of C++ (bar C++14, as it is not a major update), and comes with a host of helpful features and improvements to the standard library, code written in the framework is to be compiled using the C++11 standard. However, some compilers still do not include every feature of C++11, and so caution should be taken in making sure of cross-compiler compatibility.

15.2.5 – Pre-processor macros:

Macros can cause hidden errors and scope issues, and so, are not to be used. Instead, consider using inline/constexpr functions and/or constants. Note that some compilers may not yet support constexpr.

15.2.6 – Null Pointers:

Considering the advantages of nullptr over NULL (NULL being an integer type while nullptr being a nullptr_t type), nullptr should be used in any case where pointers are involved, and NULL should not be used at all.

15.2.7 – Function Overloading:

Overloading of functions should be used only where appropriate. In cases where only the types of the function parameters change, consider renaming the function instead of overloading it. Also consider that default arguments can prevent the use of an overloaded function. Examples:

```
void doSomething(int param1, int param2);

void doSomething(short param1, short param2); // could cause
undesired behaviour, rename this function

void somethingElse(int param1, int param2, char c = 'A'); // default
arg can replace an overload
```

15.2.8 – Default arguments:

As long as default arguments and their functionality are well documented, they can be used. If documentation is too much for the default argument, consider removing it as it may cause ambiguity.

15.2.9 – Error handling:

Exceptions should be used and maintained as they can carry important information, and escape problem code in a simple fashion. Exception should not be used to return from a function, or skip code, or carry data around (exclusively). Where appropriate, throw an exception with a clear and informative message, as well as any other useful information (problem variables, names of variables that went wrong, error codes, etc.).

15.2.10 – Casting:

C++ style casts should be used in replacement of C-style, as they may provide better error information, and make it easier to understand what is being converted to what. Casting all together should not be avoided (especially in a language such as C++), but only done when necessary.

15.2.11 – Boolean Expressions:

Any boolean expression may not include a comparison to true or false. However, non-boolean expressions should not be cast to true or false.

```
Example:

Good:
bool foo = false;
int a = 1;
if (foo)
    cout << "bar\n";
if (foo == false) // unnecessary
    cout << "foo\n";
if (a == 1)
    cout << "foobar\n";

Bad:
int a = 1;
if (a)
```

```
    cout << "bar\n";
if (a == false)
    cout << "foo\n";
```

15.2.12 – Enums:

Enums are to use the enum class syntax (available in C++11), which solves enum scope issues, and namespace/class collisions/pollution. Example:

```
enum class Foo
{
    BARONE,
    BARTWO
};
```

15.3 – Scope

Scope refers to C++ scope, as well as file scope (static/includes).

15.3.1 – Global scope:

Globals should not be used at all. If something must be used in a global sense, consider wrapping it in a namespace or class in a sensible fashion. Globals can be hard to track down during debugging, and cause clutter.

15.3.2 – Namespaces:

Namespaces help separate code and avoid disambiguity from external sources. Using directives must be avoided in headers, and should be used sparingly, as they may make code harder to track.

15.3.3 – Classes:

Classes should be nested, or separated, depending on the situation. Classes that are used privately by other classes (such as a linked list node) should be nested. Other than that, it really does depend on the situation.

15.3.4 – Includes:

Forward declarations should be created in header files as often as possible, unless the type comes from the standard library, in which case, it may be more of a hassle to forward declare than not to.

15.3.5 – Headers:

Each header must include its guards to prevent multiple inclusions. Guard names are defined as follows:

```
MODULE OR SUBSECTION OR FRAMEWORK ABBREVIATION]_FILENAME_H
```

15.4 – Classes

This section refers to the class/struct/union types (although unions are used rarely).

15.4.1 – Constructors:

Each class should contain a default constructor, copy constructor, and an equal's operator, due to some compilers implementing them automatically, which may cause undesired behaviour. They may have any appropriate access modifier, however.

15.4.2 – Functionality:

Classes and structs should be used based on their level of functionality. Classes should define a type that can have operations and algorithms performed either using it, or from it. Structs should be considered as Plain Old Data (POD) types, and contain no methods other than the constructors/equals operator. Unions should behave in the same fashion as structs.

15.4.3 – Access modifiers:

Access modifiers should always be stated, regardless of the type of class (class/struct). The only exception to this is when the class has no methods/data of that specifier. The order of modifiers in a class is public, protected, and then private. Example:

```
class MyClass // protected: is omitted because it is not used
{
    public: // stated anyway, even though classes are private by
default
        int foo;

    private:
        void doSomething();
};
```

15.4.4 – Interfaces:

Interfaces can be useful, and so are allowed to be used. Interface names should be prefixed with an 'I' to make it obvious that they are actually interfaces, and must contain a virtual destructor (non-pure). Also, all methods in an interface must be public and pure virtual (struct's may be used for this). Interfaces may not include constructors or the equals operator. Example:

```
class IMyInterface
{
    public:
        virtual ~IMyInterface() {} // destructor is non-pure virtual
        virtual void doSomething() = 0; // pure virtual
};
```

15.4.5 – Templates:

Templates declared and defined in headers should be split into declaration and definition, in the same file.

16. Glossary

Achievements – Trophies that are obtained by the user for completing certain requirements i.e. walk10,000 steps.

Algorithms – A process or a set of rules to be followed in calculations or other problem-solving operations.

Applications – The action or use of something into operation.

Architecture – Software architecture is the high level structure of a software system.

Artificial Intelligence – AI refers to the creation of either enemies or testing components that exhibit a thought like process to help in debugging or as an enemy during the game play.

Client – This refers to the game client where the user plays the game.

Communication – This refers to the relaying of information from one source to another.

Community – This refers to the online web forum aspect where the users form the gaming community.

Configuration – This refers to how the user can change the settings.

Cross Platform – The program and framework being available on more than one platform i.e. both Windows and Mac.

Database – The structured set of data maintained to hold the details of the registered users for Krypta.

Deployment environment – The deployment environment refers to the processes that make the software available for use i.e. hosting, release, installation and activation etc.

Design patterns – In software engineering, a design pattern is a general reusable solution to a commonly occurring problem.

Development Environment – The development environment is the set of processes and programming tools that are used to create the software or product.

Editor – This refers to the Map Editor that will allow the users to build their own custom maps and edit them accordingly.

Events – This refers to an action that has occurred i.e. on touch – that will warrant another action to occur i.e. pick up.

Framework – The basic structure underlying a system that is used as a supporting structure in the aid of building i.e. the Krypta 2D framework was used to build the Krypta game.

Functionality – The range of operations that can be run or performed by the program.

Graphics – This refers to the visuals displayed by the game..

Level – This refers to the levels or different floors of the dungeon.

Libraries – A library is a set of predefined functions and programs that are often preloaded onto the computer i.e. iostream, cmath, etc.

Logic – reasoning conduction or accessed according to strict principles of validity which can require forethought and preplanning.

Map File – The map file refers to the file containing the map data for custom and prebuilt maps.

Operating Environment – The system and where the program is run i.e. Windows 7.

Protocol – The procedure or system of rules that is followed.

Prototypes – This refers to preliminary versions of the program or early concepts.

Scenario – a sequence or development of events.

Server – The web hosted server responsible for managing the data.

Software Components – The software components refer to the structure of the system i.e. the game client, the framework, the map editor.

Structures – Structures refer to user defined variables.

Technologies – What types of technologies were used i.e. computer.

Test – a procedure to establish quality, performance or reliability.

Test Cases – Test cases are different instances of the program to determine quality, performance and reliability.

Test Suite – A test suite is a collection of test cases that are used to test a software program that it has some specified set of behaviours.

User Interface – The means by which the user and the program interact i.e. the way the user navigates the program.

View – This refers to what Is visible to the user.

World – The world refers to different packages that the user can create for the game i.e. a set of maps and levels.