CIS 422

Panda Engine

**Zachary Bower, Chase Craig, Linshu Huang, Benjamin Yin, Luyao Wang**

Table of Contents

[**Project Plan**](#_abossxldbx6u) **3**

[Purpose and Audience](#_3y69s0lc6itd) 3

[Important Notice](#_sgx6f9l783bb) 3

[Team Roles and Responsibilities](#_x19dwqcjknqu) 3

[Team Responsibilities](#_nyi9k4ssvpsc) 3

[Documentation Responsibilities](#_fzw923jurxin) 4

[Project Development Responsibilities](#_dgf6i81ano0x) 4

[Risks and Risk Mitigation](#_ahmcisw1iq22) 6

[Project Risks](#_37al912h5u0o) 6

[Risk Assessment and Mitigation](#_f4ai7er39qrq) 6

[Process](#_72lqlb47sxod) 8

[Project Overview and Constraints](#_sw931drurdr8) 8

[Addressing specific developmental objectives](#_7eomqsuy26nt) 9

[Mechanisms, Methods, Techniques](#_sg90kjh8oohc) 9

[Detailed Schedule and Milestones](#_jbpwoc9hj62r) 9

[Resources and References](#_r2klyvs1p90a) 10

[Meeting Notes](#_od2mhrf5yni) 10

[**Software Requirements Specification (SRS)**](#_kzfrj4hnb0yg) **11**

[1. Introduction](#_okbkt3xak7ya) 11

[1.1. Intended Audience and Purpose](#_9d63hp57xa32) 11

[2. Concept of Operations](#_t8nyr6n5m0hw) 11

[2.1. System Context](#_xlb5grki6w8r) 12

[2.2. System capabilities](#_5vfc1gxji7jr) 13

[2.3. Use Cases](#_xrg9uyrav3m) 13

[3. Behavioral Requirements](#_71n34ip904nw) 14

[3.1. System Inputs and Outputs](#_qlgk7yac3xx1) 15

[3.1.1 Inputs](#_bt4edj2whmwk) 15

[3.1.2 Outputs](#_ahh8ny8hb4ne) 15

[3.2. Detailed Output Behavior](#_gytlmoqy277b) 16

[4.Developmental Quality Attributes](#_5kej2ltw9i9u) 16

[Behavioral qualities based on runtime :](#_rgg6nv3lmwt5) 16

[Developmental qualities based on design-time:](#_k1yvn7d0vei3) 16

[5.Fundamental Assumptions](#_5oluxphaexfv) 17

[6.Definitions, Abbreviations, References](#_b6zzbjjtxne2) 17

[6.1 Definitions](#_5syot7c1u6l9) 17

[6.2 Acronyms and abbreviations](#_7fphdtuqzunj) 19

[6.3 References](#_qhp9g2pggo49) 20

[**Software Design Specification**](#_sw9vgz9tmhq9) **21**

[1.Software Architecture](#_f2e8tpdpog98) 21

[Static Diagram with major interactions](#_lnth0ld1nw9) 21

[Major Functionality of Components](#_bpl9p8ci968g) 21

[Dynamic Diagram](#_wetrnbas2ej5) 22

[Module Interface Specification](#_y5av1bnekbnx) 22

[How to read this section](#_7ofs65hylcir) 22

[Data Types:](#_90w7tt7pkkys) 23

[Available Modules and Functions](#_i0yqdk8s0m7q) 23

[Event Dispatcher Module](#_m4pett2nrtai) 23

[Game Management Module](#_s4fvspob3d6x) 24

[Logging Module](#_v8tabfm2eujy) 25

[**Quality Assurance Plan**](#_6dx03en7j7xx) **26**

[Reviews](#_wsn8uv28lv7p) 26

[Testing](#_xp9cxoixfsx3) 27

[Software Documentation](#_psb0k0czt8vg) 29

[**Developer Logs**](#_3fyzz2a4na1x) **29**

# 

# 

# 

# 

Project Plan

|  |  |  |
| --- | --- | --- |
| Date | Author | Description |
| 02/19/2019 | Zachary Bower | Initial Project Plan |
| 03/01/2019 | Zachary Bower | Updated plans to reflect change in schedule |

## Purpose and Audience

This document is intended for two separate audiences:

* Developers: This document will be used to guide developers of Panda Engine towards understanding the project. This document shall outline individual workloads of member and should be used as a measure of the project as a whole.
* Stakeholders: Any person with a vested interest in the outcome of Panda Engine. This may include: future team members, evaluators, and investors. This document shall outline information needed to make decision regarding the project state including programming and documentation.

## Important Notice

An important definitions section can be found further below in this document. These definitions will be referenced before the section.

## Team Roles and Responsibilities

### Team Responsibilities

* A team responsibility is defined as ensuring all members assigned with a task are on schedule, reallocating available resources as needed.

|  |  |  |
| --- | --- | --- |
| Role | Team Member | Responsibilities |
| Resource Manager | Chase Craig | Ensures the team is on schedule, adjusts roles |
| Checks and Balances | Zachary Bower | Ensuring the code base maintains quality.  Resolving merge conflicts. |
| Documentation Manager | Linshu Huang | Ensures the documentation is consistent and on schedule |

### Documentation Responsibilities

* A documentation responsibility is defined as the primary maintainer(s) of a document

|  |  |
| --- | --- |
| Document(s) | Team Member(s) |
| Project Plan | Zachary Bower |
| Software Requirement Specification | Linshu Huang, Luyao Wang |
| Software Design Specification | Chase Craig, Benjamin Yin |
| Developer Documentation | Zachary Bower |
| User Documentation | Luyao Wang, Linshu Huang |

### Project Development Responsibilities

* A project development responsibility is defined as the primary contribution of a team member to the project.

|  |  |  |
| --- | --- | --- |
| Role | Team Member | Responsibilities |
| Designer | Chase Craig  Zachary Bower | Designing the system including: components, event handling, design architecture |
| Developer | Zachary Bower  Luyao Wang | Maintaining code and repository integrity.  Assisting other developers with questions |
| Editor | Luyao Wang  Chase Craig | Reviewing documentation ensuring the quality and cohesion of each document |
| Testing | Benjamin Yin  Linshu Huang | Interacting with the system as an end user with the highest expected competence. Reporting, logging, and assigning/correcting issues. |
| Quality Assurance | Linshu Huang  Benjamin Yin | Interacting with the system as an end user with either malicious intent (defined as the desire to break the system) or the lowest expected competence. Reporting, logging and assigning/correcting issues. |

Project Responsibilities

* A project responsibility is defined as the main responsibility of a team member towards the code base.

|  |  |  |
| --- | --- | --- |
| Role | Team Member | Responsibility |
| Engine | Zachary Bower  Luyao Wang | Create the basic components of a three dimensional game engine. This include communication with hardware(keyboard, mouse, GPU) setting up back end to speak with components. |
| Entity Creation | Linshu Huang  Benjamin Yin | Allow the system to create entities, or objects within a created world. These may or may not be interacted with |
| Event Delegation | Chase Craig  Zachary Bower | Allow the system to communicate with components under specific conditions |
| File Management | Benjamin Yin  Luyao Wang  Linshu Huang | Allow the system to take as input multiple file types depending upon the need of the system. These may include vertex files, image files, sound files, and map files |
| Game State Management | Chase Craig  Benjamin Yin | Allow the system to manage multiple states. A state is defined as a distinct circumstance. |
| Math Functions | Linshu Huang  Luyao Wang | Allow the system to utilize a variety of mathematical functions needed for an assortment of calculations. |
| Threading | Zachary Bower  Chase Craig | Allow portions of the system to communicate in an asynchronous manner |

## Risks and Risk Mitigation

### Project Risks

Major risks determined for this system include:

* Late delivery of milestones
* The system will not meet expectations
* Deviation/Contradictions from documentation
* Poor modularity/code reuse
* Poor commenting on source code
* Project scope too large

### Risk Assessment and Mitigation

**Late Delivery**

The cost associated with late delivery is crucial. Late delivery will result in the project as a whole falling being schedule. This will cause stakeholders to lose confidence in the ability of the team to produce a product and will result in a poor grade for the course.

A schedule has been established to monitor project status. The schedule will be followed during all stages of project development. The team shall build in slack time into the schedule, reallocate resources and construct a Gantt chart to identify critical paths of the development stage.

**System Expectations Not Met**

The end system not meeting expectations would result in the system not working as intended. This would cause stakeholders to lose confidence in the team's ability to produce a product resulting in a poor grade.

A clear requirements documentation has been created outlining the expected functional requirements and non-functional requirements. These requirements were picked as a team and selected based on the time given to complete the system. Additionally, a clear line of communication has been established for any team member who seeks clarification.

**Deviation/Contradiction in documentation**

The end system containing contradictions or deviations will result in a system not meeting the expectations set in the beginning. This would cause the system to have not delivered on the goals laid out in the documentation.

A definition of what the system should delivery has been agreed upon by team members prior to any work on the system taking place. Additionally, a communication channel is available to all team members who seek clarification. The team editor will also ensure this risk has been minimized by conducting periodic reviews of the documentation.

**Poor Modularity/Code Reuse**

The end system possessing high coupling will make the testing and maintenance stages of development more difficult. Poor modularity will also result in difficulting working between members with version control software.

The software design specification has been created with modularity in mind. Additionally a team member has been assigned to conduct checks and balances with the code base to ensure the end system is meeting the specification of loose coupling.

**Poor Commenting on Source Code**

The end system lacking code documentation will make maintenance, bug fixes, and logic flow difficult to follow. This will result in extra work during the maintenance and testing stages, leaving less time for other tasks.

The code base will go through a periodic review to ensure comments are regularly added and changed. The team has agreed to a general commenting style for each source file.

**Project Scope too Large**

The scope of the project involves working with complex libraries and other systems in an efficient and flexible way. Many team members are not familiar with these libraries and will have to learn as the project progresses. The scope may be large means the system may be considered successful.

The project was chosen as a challenge for all team members. The team members have chosen to incrementally build the system, reusing as much code as possible from one stage to the next. Goals have been strictly set to bring the scope of the project into more manageable pieces.

## Process

Our team meets at minimum two times per week, currently Mondays 12 - 1, and Fridays 3:30 - 5. During the meeting, the project manager discusses any pressing issues, deadlines, or concerns facing the team. After the project manager has gone through concerns, each team member gives a progress report and any concerns facing the team member.

The team has set up a communication server all member have access to. This server is used in an informal sense and allows team members to ask for assistance, share progress, and inform team members of any applicable information.

### Project Overview and Constraints

The goal of the project is to create an initial version of a three-dimensional rendering engine. This application is designed in a manner that it can be improved upon well after the team members are no longer in the CIS422 class. That is some goals are intentionally left until after the term is over.

The two primary constraints of the project are the short time for the initial development cycle. The second constraint involves the team members knowledge of the development libraries used in the project.

### Addressing specific developmental objectives

Our team has compiled a list of milestones to reflect the amount of time available for project scope as described by the Project 2 requirements for CIS422.

## Mechanisms, Methods, Techniques

The current plan is to design the system using the prototyping method .

* Our team plans to implement incremental systems building upon previous version to work towards a more complex system.
* At each stage a specific goal will be set out, such as ‘the system will perform x’. The judgement of success will be on the system's ability to perform bug free.
* After the testing of any prototype, the code base will be updated to reflect any feature(s) the system may contain

The following tool are utilized to during the implementation of the system

* Github for version control and change logs, and issue tracking
* Email for announcements and official communications
* Excell for creation of Gantt charts

## Detailed Schedule and Milestones

Dates of milestones and meetings

[Gantt Chart](https://docs.google.com/spreadsheets/d/162KMobmWxzCHWtQR8V7vKhSSt06m_RMF0v9VQr1bygI)

02/15 - Initial Requirements Agreed Upon

02/15 - Initial Roles Assigned

02/15 - Documentation Started

02/18 - First Meeting, Roles set

**02/18 - First prototype (basic hardware communication)**

02/20 - Documentation Completed (without editing)

**02/21 - Documentation gone through editing process and submitted**

**02/22 - Second prototype (full 3D model with texture)**

02/22 - Second Meeting

02/22 - Developer Documentation Created

02/22 - User Documentation Created

**02/24 - Second prototype completed testing**

02/25 - Third meeting

**02/25 - All supported input files are parsed correctly**

02/27 - Tutorial documentation created

**02/28 - 3D world can be rendered from file**

03/01 - Fourth meeting

**03/01 - Event Handling Complete**

03/04 - Fifth meeting

**03/04 - Tutorial for creating prototypes 1 - 2 complete**

03/08 - Sixth meeting

03/11 - Seventh meeting

**03/14 - Project Complete**

**03/14 - Presentation Complete**

## Resources and References

This project will make use of multiple resources. These resources include OpenGL, SDL2, and GLM. These resources were chosen as they are long established libraries, compatible with multiple operating systems and well documented.

* OpenGL: <https://khronos.org/registry/OpenGL-Refpages/gl4/>
* SDL2: <https://wiki.libsdl.org/FrontPage>
* GLM: <https://glm.g-truc.net/0.9.4/index.html>

## Meeting Notes

[2/13 Meeting](https://docs.google.com/document/d/1PZfGOx2zmBy_7Tq-mbmUAzpCqkEAIXNkXBovv86G3A0/edit)

[2/15 Meeting](https://drive.google.com/open?id=1bpSbeDIjkd1h9W3I-nZsZ0FGtk1iwlgawvi1AlFh35A)

[2/18 Meeting](http://docs.google.com/document/d/1cxpHYT5edLHwJncAxKpR662XYilBKTx9yliFkNNXOd8/)

[2/22 Meeting](https://docs.google.com/document/d/1mkA9gkJhjFcg7yIDmtg0hm9ADaiqPZnjfQZqSUNinBg/)

[3/4 Meeting](https://docs.google.com/document/d/1iFfXVCsoQWOTwDZRSizUihv3htTq5o0kIHbl3BIfJBQ)

[3/8 Meeting](https://docs.google.com/document/d/1ou94hzf0dEhH1HhM6yDN1Cio9soEQ696Ib0BDI98fys)

[3/9 Meeting](https://docs.google.com/document/d/1v6I7h8wN3ghtKaDRCk_5W4IrfjgfXOmEl14QfolncXM)

[3/11 Meeting](https://docs.google.com/document/d/1TVfzwLq2n5aVDNkfxPdU0GCNK5IY_e78dIp9MA-XkVI)

# 

# Software Requirements Specification (SRS)

Revision History:

|  |  |  |
| --- | --- | --- |
| Date | Author | Description |
| 2/20/2019 | Linshu Huang | Revised section 1 - 3 |
| 2/20/2019 | Luyao Wang | Revised section 4 - 6 |
| 02/28/2019 | Zachary Bower | Revised static models |
| 3/11/2019 | Chase Craig | Overall revision |
| 3/12/2019 | Luyao Wang | Overall revision |
| 3/12/2019 | Benjamin Yin | Diagrams revision |

## 1. Introduction

### 1.1. Intended Audience and Purpose

End users of the system:

An end user is any user who intends to use the functionality of the system or any component of the system. This document describes what the system shall accomplish for such a user.

Developers future and present:

This document outlines the goals of the system as a whole and each component. All developers working on the project will need to understand these goals to deliver the correct end product.

## 2. Concept of Operations

The end user is expected to have knowledge of C++ and common programming paradigms. The user will directly interact with the system through source code.

The end user is not expected to have knowledge of graphics, hardware configuration, operating system communication, or three-dimensional rendering techniques. Knowledge in one or all of these is encouraged, but not necessary for an end user to interact with the system.

### 2.1. System Context

The system utilizes three third party libraries:

OpenGL: A cross platform library used to interact with graphical hardware.

SDL2: A cross platform program to communicate with common peripheral devices

GLM: An OpenGL mathematics library



The system will utilize OpenGL in the following manner.

* The system will use OpenGL to communicate with the GPU and CPU.
* The system will use OpenGL to create a depth buffer
* The system will use OpenGL to create a GPU Context
* The system will use OpenGL to draw polygons on the screen

The system will utilize SDL in the following manner.

* The system will use SDL to communicate with common peripheral devices including: keyboard, mouse, sound device(s), joysticks, controller and more.
* The system will use SDL in the creation of an interactable window and map user input to a position in the window.
* The system will use SDL in the creation of event handlers based off user input.

The system will utilize GLM in the following manner.

* The system will use GLM to provide data structures that is helpful for parsing the multimedia files, textures, and shaders.
* The system will use GLM to increase the efficiency and code clearance on math calculation.

### 2.2. System capabilities

This engine allows the user to construct an interactive program by delegating management of common tasks (rendering, interaction, user input, file I/O, object management, event handling) to the system.

### 2.3. Use Cases

2.3.1 Create an object in a specified location

* The user creates an object with settings
* The user informs the system of the location of the object
* The System updates its state
* An object is created

2.3.2 Display a game object

* The user gives the system the settings of the object
* The System updates its state
* An object is displayed

2.3.3 Assign textures to the object

* The user gives the system the settings of the texture
* The user gives the system the location of the texture application
* The system updates its state
* A texture is assigned to an object

2.3.4 Adding shading effect to the object

* The user gives the system the settings of a shader
* The system parses the shader settings
* The system links and compiles the settings
* The system updates its state
* Shading settings are applied to objects

## 3. Behavioral Requirements

The system will allow the user to incorporate any component(s) into an end users system. The components are separated into the following groups

Display:

* The system shall allow the creation of a display using settings defined by the end user
* The system shall remove the display when requested by the end user

Drawing:

* The system shall create a set of vertices at a location specified by the end user
* The system shall load and parse files specified by the end user
* The system shall map file coordinates to vertices as a texture
* The system shall support both user defined and default shaders
* The system shall support 3D and 2D graphics
* The system shall support 32 bit color

Events:

* The system shall support object creation defined by the end user
* The system shall support event creation defined by the end user
* The system shall support communication with peripheral devices
* The system shall support movement of world object and view objects

Game State

* The system shall keep track of ticks
* The system shall update object states
* The system shall support camera location manipulation
* The system shall support perspective location
* The system shall support interrupts
* The system shall support updating the view displayed to user

System:

* The system shall be modular with at least 22 files
* The system shall allow all or selected components to be used
* The system shall communicate with an end users system

### 3.1. System Inputs and Outputs

Due to the nature of the system, the inputs are defined as user settings.

#### 3.1.1 Inputs

The input is determined by the module called by the user

Display:

* Settings for window creation

Drawing:

* Settings from the end user

Event:

* Settings from the end user

Game State:

* Previous game state

#### 3.1.2 Outputs

The output is determined by the module called by the user. Each major subsystem may output any of the following items.

Display:

* An interactable window matching the end user’s settings

Drawing:

* Any number of polygons defined by the users settings, adjusted to OpenGL coordinates
* Lighting effects defined by the user
* A mapping of textures to polygons

Event:

* Event listener with user settings

Game State:

* Updated game state

#### 3.2. Detailed Output Behavior

The program is a game engine that takes users’ mouse click and keyboard input as the controller of the object behaviors.

|  |  |
| --- | --- |
| Action | Output Behavior |
| Display (set up the dimension of windows) | Window Creation/Deletion |
| Drawing (set the coordination of vertices; texture and shading of the objects) | Render objects |
| Event Handler | Listen to the user’s module call |
| Update | Inherited from previous state and output a new state |

## 

## 4.Developmental Quality Attributes

#### Behavioral qualities based on runtime :

* Performance: The Elapsed time between user inputs and system response should be less than 500 ms.
* Reliability: System should be able to parse the coordinates that been created by users without losing data or errors.
* Standards: Ensure the data entered is transferred by proper coordinates
* Consistency: Editing the 3D-objects without losing associated data.
* Usability/Ease of use: User will be able to navigate through the Engine by the User Guide, Installation Guide, and game prototyping.

#### Developmental qualities based on design-time:

* Modifiability: Ability that is able to modify to correct faults without a redesign, accomplished by the development of individuals module.
* Maintainability: The program should be able to update to new changes with new features and the intention of optimizing performance that satisfies the user requirements
* Reusability: The program should be able to use of existing assets in some form within the software product development process; these assets are products and by-products of the software development life cycle and include code, software components, test suites, designs and documentation
* Understandability: The software should be understood for a given development team for integrating into new software with the use of its libraries and software components

## 5.Fundamental Assumptions

The system will be developed to run on a computer running the GNU/Linux operating system. Due to the nature of the project the end user may use any number of components within the system.

To test the system a number of prototypes will be available. To run the prototypes, the following assumptions are made: A computer running the GNU/Linux operating system with a GUI (we recommend Xorg). Further assumptions for each prototype are listed below.

* Hello Cube: Minimum requirements
* World creation: Minimum requirements
* World with movement: A mouse and keyboard
* World with movement and entity creation: A mouse and keyboard

## 6.Definitions, Abbreviations, References

### 6.1 Definitions

|  |  |
| --- | --- |
| **Keyword** | **Definitions** |
| Client | Any third party system that will use this system to produce a final product. |
| Peripheral Devices | A hardware device (although can be emulated through the use of software) that can interact with the machine. Typical examples include mouse or keyboards. |
| Shader | A program that is able to change how a model will render given a specific openGL state and texture. |
| End User | An end user is an entity (e.g. a person, a computer system) that will be using the product built from the client. |
| Game | A game in this document will refer to the piece of software capable of providing a service to an end user developed by the client. |
| Engine | This piece of software. |
| Object | An object is an abstraction of anything that can be displayed/”drawn” to the end user’s screen and that will be managed by the engine. |
| Screen | A screen is a window displayed to the end user. |
| Entity | An object that also is considered to be non-static and interact-able. |
| Xorg | X.Org Server is the free and open-source implementation of the display server for the X Window System stewarded by the X.Org Foundation. |
| Event firing | The action of when a trigger for a given event occurs. |
| Event listeners | A function that is a part of a piece of software that is called when a particular trigger has occurred in the engine. |
| Frame (timing) | A frame is when the screen is updated with the next picture to be displayed to the user (typically every 1/60th of a second, but may fluctuate). Usually when drawing to the screen is done. |
| Tick (timing) | A tick is when the system’s internal logic loop finishes, effectively when the game goes from one state to the next (typically every 1/60th of a second, but may be faster and may also fluctuate). Usually when physics or other world manipulation is done. |
| Vertex | A point where higher-dimensional geometric objects meet |
| Mesh | A collection of vertices and polygons |
| GPU | Graphical Processing Unit |
| CPU | Central Processing Unit |
| GPU Context | A direct communication line with the GPU |

### 6.2 Acronyms and abbreviations

|  |  |
| --- | --- |
| **Acronym or Abbreviations** | **Definitions** |
| API | An application programming interface is a set of subroutine definitions, communication protocols, and tools for building software |
| GUI | A graphic user interface is an interface of human interacts with a computer that uses windows, icons, menus and which can be manipulated by mouse or keyboard. |
| GNU | GNU is an operating system and an extensive collection of computer software. |
| OpenGL | Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics |
| SDL2 | Simple DirectMedia Layer is a cross-platform development library designed to provide low-level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D |
| GLM | OpenGL Mathematics (GLM) is a header only C++ mathematics library for graphics software |
| I/O | Input / Output |
| UUID | Universally Unique Identifier |
| IMGUI | Immediate Mode GUI, an interface for immediately rendering to the displayed screen. |

## 6.3 References

For more information on Software Quality Attributes: <http://www.ieee.org.ar/downloads/Barbacci-05-notas1.pdf>

For more information on Reusability: <https://en.wikipedia.org/wiki/Reusability>

For more information on GLM: <https://glm.g-truc.net/0.9.9/index.html>

For more information on SDL2: <https://www.libsdl.org/>

For more information on OpenGL: <http://openglbook.com/chapter-0-preface-what-is-opengl.html>

For more information on GNU:<https://www.gnu.org/home.en.html>

For more information on API: <https://en.wikipedia.org/wiki/Application_programming_interface>

# Software Design Specification

## 1.Software Architecture

### Static Diagram with major interactions

#### 

### Major Functionality of Components

The GameManager handles the initialization of SDL and OpenGL. If these libraries have been initialized a SDL window is created. The third party program also creates a Run function on initial game manager call. The run function creates an event dispatcher which manages registering, unregistering, and executing events.

The event dispatcher is given the logic and which events to listen to (game, mouse, keyboard) from the third party program. These events are triggered by the game manager and the event dispatcher is notified accordingly.

The Display is hidden or shown depending on events given from the game manager to the display. Events may include: hide, show, swap, clear or render. The render information is given from the third party program.

The third party user inputs events in the form of keyboard presses, mouse movement, or events defined by the third party program. These events are given to the game manager which contains the logic to handle these events. Any components needing to be updated are updated accordingly.

### Dynamic Diagram

#### 

## Module Interface Specification

### How to read this section

The module interface specification section is broken down into two parts: classes and data structures, and sub-modules available to the user. The classes and data structures are made available to the user and will be referenced frequently below as parameters to functions. The available modules and functions list all functions that are available for a third party to use to manipulate the execution of the engine.

### Data Types:

**EventListener:** A class template with specific event hooks (functions to be called) predefined for when the event is fired.

**EventDetails:** A data structure holding information about a given event, include the time, the event ID, event name, if it is cancellable or not, and if it was cancelled..

**Textures:** A data structure capable of holding image information ready for displaying or manipulation.

**Camera:** An object that is capable of transforming the “world” the user sees. This is very similar to an actual camera in how it takes a scene in the real world and “renders” it to a flat image.

**Shader:** An object/program that takes in information provided from the graphics library about the world and returns how the particular object should be rendered to the user. This would be the “shading” or coloration of an object. Shaders are programs themselves and are compiled by the user’s graphics library on runtime.

**Mesh:** A data structure containing a buffer of Vertex objects, these Vertex objects defined triangles that could be rendered to the screen.

**Object:** A generic class/data structure that will be stored in the world management module.

**Vector*N*:** A data structure for storing ***N*** objects representing a value in ***N*** dimensional object space. Typically used with numbers to represent a point in ***N-***D space.

### Available Modules and Functions

This section shall document required modules and actions (“functions”) available within each module.

#### Event Dispatcher Module

Handles the management and execution of all event listeners.

Required available functions with minimum required information passing:

void RegisterListener(EventListener\* lis, Priority p) - Registers the event listener lis to receive events with priority p. The priority dictates what order the listeners will be executed.

void UnregisterListener(EventListener\* lis) - Unregisters the event listener lis from all events.

int RegisterUserDefinedEvent(string eventName) - Registers an user defined event for listeners to listen for. Will return an integer as the unique ID of the custom event. For any listeners already waiting for the user defined event (from using below), they will be automatically registered when calling this function.

void RegisterUserDefinedListener(GenericEventListener\* lis, string eventName, Priority p) - Registers an event listener lis to receive eventName events with priority p. The priority dictates what order the listeners will be executed. Will cause an log a warning (with a unique id) if the eventName doesn’t yet exist. If the game starts before the user defined event is registered, the system will immediately abort (see below).

void UnregisterUserDefinedListener(GenericEventListener\* lis, string eventName)

Unregisters the event listener lis from the given event eventName.

bool ExecuteUserDefinedEvents(string eventName, bool cancellable) - Fires the event eventName, calling all listeners registered to the event. The event is cancellable is cancellable is true. If the event was cancelled, this will return false, otherwise it will return true.

#### Game Management Module

Handles the management of all the game operations.

Required available functions with minimum required information passing:

void StartGame(bool startPaused) - Flags the Game Management module to start the game. If startPaused is true, then the game will start paused (see below). Will fire the game starting event. Cancellable. Requires the game to not be running.

void StopGame(bool forced) - Flags the Game Management module to stop the game and close normally. Will fire the game stopping event. Cancellable only if forced is false. Requires the game to be running.

void PauseGame() - Flags the Game Management module to stop firing tick events for the game. All functionality should still be available besides for those listening to tick events. Requires the game to be running.

void ResumeGame() - Flags the Game Management module to resume firing tick events for the game. All functionality should be restored to listeners listening to tick events. Requires the game to be running.

void Abort(string reason) - Flags the Game Management module to stop as soon as possible, and closing the application with little safety. Will ***not*** fire the game stopping event (and hence not cancellable). Will log the reason with the logger with a critical status.

#### Logging Module

Handles the management of an internal logging system to debug the application in an organized manner.

Required available functions with minimum required information passing:

void Log(int debugLevel, string str) - Adds the string str to the logger with the severity level as stated in debugLevel. All debugs will automatically have the time and severity should be appended to the string.

void PurgeLogs(int debugLevel) - Clears the buffer of strings from the logs. This effectively deletes the logs with the given severity level debugLevel, and saving the logs should be done first before executing this function.

bool WriteLog(int debugLevel) - Saves the buffer of strings from the logs of severity level debugLevel to the local filesystem. This should be automatically called when the system enters the stopped state. This returns false if the logs can not be saved, while a result of true means the logs succeeded in saving.

# 

# 

# Quality Assurance Plan

To test whether the program valid for all of the requirement, the team would make at least one game using this gaming engine. In the game, all of the modules would be called at least once based on what listed in Software Design Specification.

To determine if the design satisfies all the necessary functional and quality requirements, the team would evaluate the game performance. After successfully call all of the modules, the gaming objects would display on the screen. The quality requirements satisfaction evaluation would base on the model’s performance.

The line of code and coupling, cohesion between the codes would be evaluated. The goal of the project is the loss coupling and strong cohesion. The code would go under the peer review process and series round of the test.

## Reviews

For the robustness, the engine is able to be fully function in Linux operating system, and potentially met some issue while running in MacOS since one of the library used on MacOS system probably not compilable for the other part of the engine. With the MacOS system we have and test so far, the demo games are successfully run, but due to the different built-in library version in MacOS system, some function may still not able to be functional.

The team had to change the plan due to two reasons. The first one is the weather issue. Snow in the half way of implementation progress cause the lost of internet connection for some members, therefore the milestone got delay. Furthermore, after talked to the professional in the gaming field, the team shifting the goal into more focusing on the asset pipeline instead of providing more freedom for the user while using the engine. The engine finally are able to accomplish the wide range of the multimedia input such as 2D and 3D objects files, mouse and keyboard input. For the administrative users, the engine using few outside libraries that out of markets for a period of time, therefore the potential errors had been discussed at the online community for a while. Aiming to the common end user, the developer documentation includes all of the libraries that the team developed with the function description. The team also plan to keep implement this engine to provide more features and making it more user friendly in future.

All of the documentations are located at the github. The README at the homepage of the repository function as the content for all of the documents, user is able to access these documents by directly clicking it.

## Testing

1, What are the overall testing goals?

The testing goals are aiming to test both each individual module and the inter-relations between the modules. There need to have a stable data transformation between modules. The testing also aiming to input the abnormal cases and expect engine would have error handler to deal with invalid input and notify the user.

2, What kinds of tests will be used to meet those goals?

1. Multiple types of the number such as int and double
2. An invalid data type for a function.
3. No input for the function that takes parameters
4. Asking result for the function that does not return
5. Invalid mouse and keyboard input, and expecting do not cause the error of the engine.

3, What tests were actually run?

|  |  |  |
| --- | --- | --- |
| Test Input | Expected Output | Results (Actual Output) |
| Display | | |
| Command to “Remove” Display | Close the window | Closes the window after a slight delay. Traced delay to hardware and disposing of the window. |
| Drawing | | |
| Function to set the vertex location and color | “Draw” (display) the object on the screen with specified location | Object/Mesh is rendered on the screen at the specified location. |
| Assign the texture at the top of the object | The object is assigned to a certain texture. | The texture can be seen on the object, however for complicated models the textures binds interestingly (though that is because the texture coordinates are not set properly). |
| Set both 2D and 3D vertex | View the 2D and 3D object respectively | Able to view 3D objects, and 2D “pseudo-objects” (technically a 2D object in 3D space). |
| Provide files ask the engine to read. Some file name is valid and some of them do not. | The engine should prompt error message when the file name is valid or the file is broken. | The engine will output an error message if the object is invalid or not found. |
| Events | | |
| Invalid object or entity type | Prompt warning and exit the program | Segfaults or fails to compile. Known “bug” with c++. |
| Call the function with an event that does not exist yet. | Output the invalid function name | Silently ignores the call. |
| Change the object type from world to view | Display different perspective | Can update the display to show the different perspective. |
| Game State | | |
| Close program without saving it | Game should attempt to intercept the call, then attempt to close normally. | Game attempts to intercept the call, attempts to close normally. |
| Change camera location | View different perspective | Views different perspective. Rotations are slightly “annoying” due to being Z-Y-X rotation and hence double rotations causes a ghost rotation in third axis. |
| Game Interrupt | Save the latest update | Closes game with notification. |
| System | | |
| Call all of the modules | All of the components should respond | All modules respond. |

## Software Documentation

The end user documentation will be split into multiple types depending on the end user.

For end users who wish to utilize all or some of the components available a basic interaction guide and tutorial will be available.

* [User Guide For Engine](https://docs.google.com/document/d/1-DKFmecn8fAYTDddtEsXq87vIHSOzJVQIl15kWUTk1Y/edit)

For end users who wish to install the prototype games created an installation and interaction guide for ‘games’ will be available.

* [Demo Guide](https://docs.google.com/document/d/1YBW0K1uobM-jCOjjdg-o60lzdgbmaSe_dAnYDWog8oE/edit)

## Developer Logs

03/12 **BY** tested on Initialized Linux system

03/11 **ZB LW CC** Fixed MacOSX bug and tested

03/10 **ZB** Discovered compatibility issue with MacOSX

03/09 **ZB CC** Finished integration of all components

03/08 **CC** View movement in 3D space added

03/07 **CC** Movement in 3D space added

03/01 **ZB** Change vertex handling to read from file or user inputted files

03/01 **LH** Created object file reader

03/01 **CC** Basic event handler created and tested

* Added multithreading
* Added gamemanager

02/24 **ZB** - Updated the comments and reorganized the repo slightly.

* Added indexing optimization

02/21 **ZB** - Updated the display and opengl to support z/depth buffering

* Updated the project with a simple obj file reader, does not read all files, or files in an unspecified format

02/17 **ZB** - Created the majority of basic components including

* Display : Creating a window of variable size with a custom title
* Texture : Parsing a user defined image and mapping the image into an OpenGL compatible coordinate system
* Mesh : Successfully created 3 vertices
* Vertex: Creating vertex with position and texture mapping
* Shader: Successfully parsed and compiled simple shaders
* Camera: Successfully created a camera in 3D space
* Transform: Successfully added real time movement
* Code based off <https://www.khronos.org/opengl/wiki/Tutorials>