SOFTWARE REQUIREMENTS SPECIFICATION

for

GET
2d Game Engine Toolbox

Version 1.0 approved

Prepared by Kier Lindsay and Léo Abiguime

CMPT-376W

April 7, 2020

Contents

Revision History

Name	Date	Reason For Changes	Version
Kier Lindsay	April 7, 2020	Inital Copy	1

1 Introduction

1.1 Purpose

This 2d game engine is designed to bridge a gap between existing graphics library and large complete 3d game engines. It will extend the capabilities of graphics library and provide common systems and feature needed to develop a game without forcing the user into a large complex engine.

1.2 Document Conventions

This document follows the IEEE SRS template. It also makes use of the KISS mentality in order to make things easy to parse.

1.3 Intended Audience and Reading Suggestions

This document is intended for the developers who will be implementing the game engine. We would also like to consider users of the engine as this document may offer a high level overview of what the product offers and its design philosophy.

1.4 Project Scope

This project is meant to complement an existing graphics library providing the core functionalists all games require and tools for developers to use. It is meant to provide a more advanced and highly customization starting point for game developers who want to have full control over there game. It is also meant to be modulate so that developers are not forced into using our designs and have the freedom to incorporate as many or as few of the features we provide.

1.5 References

SFML IMGUI SDL2 RayLib

Minimal programing guides.

others as we write them or references to documentation for implementation

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information

so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

2 Overall Description

2.1 Product Perspective

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

2.2 Product Functions

Engine

Timing - Setups at maintian a core game loop Allows users to use real dt or fix it so each tick is a consistent time. Allows users to set tickrate and framerate independently Allows users to hook into update and render calls

States - Manage switching between game states Allow users to Bind a number of game states to an engine each with its own update and render functions Allow users to toggle states.

Assets - Manages the loading and management of game assets such as fonts and textures. Supports Fonts Supports Sprites Supports Textures Supports Animations

Physics Module - This will provide Management and Physics functions that apply to bodys. Camera - This will provide cameras that can be applied to A game state. or accessed as a transformation. UI - Adds the ability to make and control UI's Util - extra utility functions that extend GL or offer useful functions Extend Math for GL's vectors. Add Extra bodys and objects such as partical systems

<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary (such as a bullet list) is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, is often effective.>

2.3 User Classes and Characteristics

Full users - these users will use this software as their starting point and build there game around our components and tools.

Partial users - These users will user the core graphics library as their starting point and may incorporate some components of our software into there project.

2.4 Operating Environment

Our Target environment for this project would be to extend SFML and we should expect the same environment that sfml does.

OS: Linux, OSX, Windows
Depandancys: SFML, OpenGL, Imgui
Language Target: C++

2.5 Design and Implementation Constraints

SFML does not support 3d inherently so it should be expected that the protect also does not support 3d.

2.6 User Documentation

API Reference
Setup Guide
Basic examples todo: expand on each of these maybe add more

2.7 Assumptions and Dependencies

3 External Interface Requirements

3.1 User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

3.2 Hardware Interfaces

joysticks? <Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

3.3 Software Interfaces

Grapics Library <Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

3.4 Communications Interfaces

Networking multiplayer <Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or

HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

4 System Features

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

4.1 Engine Module

4.1.1 Description and Priority

This feature proves the core game loop and state functionality it is an essential feature and is high priority. The timing component we allow users to manage the main loop speed. and when it calls the current states update and render functions. These can be different so games can run at a constant physics rate and scale that as necessary but only render when needed or at a fixed frame rate. It will have game states that can be switched between from events within the update function. This is meant to be extended by the users game and the update and render functions overridden with games specific code.

4.1.2 Stimulus/Response Sequences

- Stimulus: Time adjusted
  Response: Internal time variables set so the new timing take effect the next tick

- Stimulus: State changed
  Response: the corresponding states update and draw function well be used on the next tick.

- Stimulus: New Tick
  Response: Current states update function is called and if it is time to draw a frame the render function is also called

- Stimulus: Camera Set
  Response: Current states Camera is set and the cameras transformation from world coordinates to Camera coordinates is applied while rendering

- Stimulus:
  Response:

4.1.3 Functional Requirements

REC-ET1: Tick rate and Frame rate must be controlled independently

REC-ET2: When calling Update in a tick the Tick time must be sent as a parameter

REC-ET3: If ticks cannot be computed in time render frames may be dropped.

REC-ES1: The update and render function should be called for all active states.

REC-ES2: States should be able toggle states from within the update function.

REC-ES3: States Should have an Identifier, Update function, and Draw function

REC-ES4: States should accept a camera transformation which applies to the render function.

REC-ES5: States should be able to be given indexes for rendering order. This should default to the order they were added

4.2 Physics Module

4.2.1 Description and Priority

The Physics Module will be a high priority. This will have three components the *Physics Manager* which can be used to hold *Physics Bodies* and manage the relationships between them as well as the *Physics Functions* which will provided many useful physics calculations and can be used on bodies independently or be used by the physics manager to update the bodies it manages..

4.2.2 Stimulus/Response Sequences

- Stimulus: New Body Created
  Response: Body Initialized with provided parameters and is returned to user for use

- Stimulus: New Manager Created
  Response: Physics Manager Initialized with universal constants provided and environment for bodies is setup

- Stimulus: Body Added to manager
  Response: Body is added to the manager and will be updated and rendered all world relationships will be applied to it

- Stimulus: Relationship Added to manager
  Response: Bodies referenced by the relationship will have it applied to them

- Stimulus: Collision Detected between Bodies
  Response: Bodies are separated and appropriate forces are applied.

- Stimulus: Body Updated delta time
  Response: All applied relationships are computed and bodies parameters are updated

- Stimulus: Body Rendered
  Response: Body is drawn to screen at corresponding position

- Stimulus: Body Deleted
  Response: All relationships are removed and corresponding managers and bodies are notified then body is deleted.

4.2.3 Functional Requirements

REC-PB1: The Physics Body class should store all physical properties required by the Functions this includes: Shape, Position, Velocity, Acceleration, Rotation,Rotational Velocity, Friction, Elasticity, Shape

REC-PB2: The Physics Body class should have an *Update* function which updates the bodies property according to classical mechanics when called, this function will take in a delta time parameter

REC-PB3: The Physics Body class should have an *Render* function which Draws the bodies shape. This should be designed to be overridden by users.

REC-PB4: The Physics Body class should keep an list of relationships it is a part of so that is can apply them all when updated.

REC-PF1: The Physics Function should accept universals constants as parameters or allow global defaults be set.

REC-PF2: The Physics Function should include *Collision detection* between two bodies with *Minimum Translation Vectors* (MTV). It should include fast approximate bounding box collision detection as well.

REC-PF3: The Physics Function should include *collision solving* functions which will apply the necessary forces to handle a collision between two bodies.

REC-PF4: The Physics Function should include *gravity* functions to apply gravity between two bodies.

REC-PF5: The Physics Function should include *constant force* functions to apply constant forces to a body these could be used to create world gravity or wind forces.

REC-PM1: The Physics Manager should create an environment to hold Body' s as well as set world constants for the bodies it manages.

REC-PM2: The Physics Manager should allow relationships to be made between bodies which define additional iterations. This includes forces and whether or not bodies should interact with each other.

REC-PM3: The Physics Manager should have an *Update* function which applies all relations and updates all bodies

REC-PM4: The Physics Manager should have an *Render* function which renders all bodies. This should allow for bodies to be drawn in a defined order

REC-PM5: The Physics Manager should allow bodies to be grouped and relations or physics functions can be defined for the whole group or for each par of bodies in the group.

REC-PM6: The Physics Manager should allow world Relationships to be set which apply to all bodies. These include global forces and n-body gravity as well as other optional relationships as appropriate.

REC-PM7: The Physics Manager should allow preform collision detection between select objects this should be efficiently implemented so that full collision detection is only used when necessary not at all times.

5 Other Nonfunctional Requirements

5.1 Performance Requirements

Physics management must fast and able to support thousands of objects.

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

5.5 Business Rules

<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>

6 Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

6.1 Appendix A: Glossary

computer is a programmable machine that receives input, stores and manipulates data, and provides output in a useful format. 12

graphics library is a library the provides a window and functions to draw to the window. Examples of graphics libraries are SFML and SDL2. 5, 12

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

6.2 Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship a single project in each SRS.>

6.3 Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

6.4 Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>