Software Requirements Specification

for

Bread-Eating Game

Version 1.0

Prepared by Team glBread

Team Members: Yinghao Wang, Haoran Mo, Yongning Fu, Xuefen Chen

June 19, 2017

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Yinghao Wang	19/06/17	Initial Draft	1.0

1. Introduction

Team glBread is an SYSU CG course team dedicated for creating opengl based software product that's fun, easy to play, flexible and scalable. We employ the latest OpenGL functions, providing both best performance and visual experience for our users.

1.1 Purpose

This SRS is a proposal to the requirements for our Computer Graphics final project, a bread-eating game. Bread eating game is a natively developed game based on raw opengl and c++, we implemented most of the functions of the game engine by ourselves. Besides gaming purposes, this project also serves as an important foundation of a full scale game engine.

1.2 Scope

The requirements specified in this document will be used for designing all the aspects and components of the game. The document will be updated as the requirements grow and change over the design and development process.

1.3 Document Conventions

SYSU = Sun-Yat-Sen University
CG = Computer Graphics
OpenGL = Open Graphics Library
GLUT = OpenGL Utility Toolkit: provides window support and useful utilities
freeGLUT = A successor for GLUT library
GLEW = OpenGL Extention Wrangler
MS = Microsoft Corporation
VS = Visual Studio
3D = Three dimensional
DevIL = DevIL, a full featured cross-platform image library
Assimp = Asset Importer, a model loading library
CO = Concrete Objects, objects which are defined as impenetrable in game space.
UI = User Interface
HUD = Head-Up Display, handy display that shows game related info while in game.
VBO = Vertex Buffer Objects, critical object to control drawing in OpenGL.

1.4 Intended Audience and Reading Suggestions

This document should be read by the developers and faculty advisors. The developers should read every section to ensure that there is an understanding for the project. The main sections for the customers and faculty advisor to review are section 1.4 Project Scope, 2.7 Assumptions, and section 3. Features.

1.5 Project Scope

The purpose of this game consists of two parts. The first is to experiment cutting edge CG techniques, providing real-time stunning visual experiences, while applying the basic knowledge from CG courses to use. The second is to create a foundation for future game development. A gaming engine that's scalable and completely modifiable by ourselves. It provides maximum flexibility to game developer and best chances of optimization.

1.6 References

None

2. Overall Description

2.1 Product Perspective

The Bread-Eating Game is a standalone game developed on Windows platform with MS visual studio IDE. It runs as a normal Windows application, and depends on several opengl support libraries as well as VS runtime environment.

2.2 Product Features

1. User may roam freely around as wish in the 3D world we constructed.
2. User will experience fall like gravity if he/she falls from high above.
3. Modern Opengl techniques are employed to enhance the visual experience for gamers.
4. We provided particle simulation to display stunning explosion effects

2.3 User Classes and Characteristics

The User who will most frequently use our software are teenagers who are intrigued into fantasy worlds and good stories. These people will find our game immersive and fun to explore with.

2.4 Operating Environment

This game is playable with a regular modern PC or laptop. See more in Section 4.

2.5 Design and Implementation Constraints

The main design constraint is that the game must be playable by people of all ages, therefore the proper considerations must be made for the users. There are certain safety requirements that we must follow. See section 5.2.

The mini-game will be written in C++ using the OpenGL libraries and functions. Therefore, the mini-game will comply to the programming practices of C++ and OpenGL. The delivered software will be open-source and maintained by Team glBread.

2.6 User Documentation

There will be a basic tutorial document along with an in game tutorial to aid users.

2.7 Assumptions and Dependencies

1. The clients' computer's graphics card driver supports at least OpenGL 3.0 and above operations.
2. The C++ redistributable package for Visual Studio is pre-installed on clients' computer.

3. System Features

3.1 Free Roaming

3.1.1 Description and Priority

Free Roaming allows user to move freely in the defined 3D space in our game. There are certain areas in the game where user cannot walk through. These are defined as concrete objects (CO) in the gaming space. This is of high priority of the game development as it will provide basic function for user to move around.

3.1.2 Stimulus/Response Sequences

When user enters the game scene, pressing W,S,A,D will move the view forward, backward, left and right side-ways respectively.

3.1.3 Functional Requirements

F-1:	The camera position, pitching and yawing angles must be recorded.
F-2:	On each frame, the position and angles must be applied correctly onto the viewing matrix.
F-3:	Required by modular programming and scalability, a camera class is required to manage viewing information.
U-1:	User moving key must map to the correct callback functions.

3.2 Physics system and collision detection

3.2.1 Description and Priority

After free roaming, a physics system is required so the the user will feel as if he is walking on earth's surface. One major aspect of the system is the gravity implementation. A collision detection system is also necessary as it ensures a ground surface for user to walk on. The priority is high.

3.2.2 Stimulus/Response Sequences

When user enters game scene, pressing space will make the view "jump" into mid air and then fall back down as if gravity is pulling. The user falls back onto the "ground" and remain still.
When User walks onto other COs, he/she will stand on top of it and remain still.
When user attempts to walk across any COs, he/she will remain

3.2.3 Functional Requirements

F-4: All collision boundaries must be pre programed into the physics engine.
F-5: Segment intersection algorithm, gravity applied camera update must be implemented.
U-2: User jumping key must map to the correct call back function.

3.3 Vivid bread model display

3.3.1 Description and Priority

We use a externally imported model for the eating target – bread. It is the essential part of our game objectives, thus it is of high priority.

3.3.2 Stimulus/Response Sequences

When user enters game scene, he/she sees the bread model, the closer he moves towards, the clearer the bread model renders. Before being eaten, the bread is rotating to itself.

3.3.3 Functional Requirements

F-6: Bread model must be placed at the correct location. Vertex information and texture correctly loaded and drawn.
F-7: Bread bounding box for eating detection must be correctly placed.
F-8: For modular programming requirement, a separate Model class should be implemented to manage model information.

3.4 Skybox

3.4.1 Description and Priority

Skybox gives user a immersive view as if he's in the virtual world, it is an essential part of gaming experience. The priority is not as high as the above two, it is medium.

3.4.2 Stimulus/Response Sequences

When user enters game scene, the world reveals as he can see skybox surrounding him.
User moves around and changes he's perspective, the skybox change as well.

3.4.3 Functional Requirements

S-1: Skybox texture mapping must be accurate.

3.5 Multi Mapping

3.5.1 Description and Priority

Multi mapping gives a real-life viewing experience of COs we placed in our gaming space. All COs we use comes with simple textures. Applying multi-mapping on COs is a meaningful feature. It it, however, of low priority.

3.5.2 Stimulus/Response Sequences

When user gets close up to the CO and look closely, the object texture appears with bumps and reflects accordingly to the position of light position.

3.5.3 Functional Requirements

S-2: Make sure the CO that needs multi-mapping uses VBOs to render. Thus providing full shader control.
S-3: Make sure bump normal are calculated correctly even in transformed space.

3.6 Intuitive UI, in game HUD display

3.6.1 Description and Priority

User will receive an intuitive user interface to instruct him/her how to play this game and he/she will be guided into the game scene. There will also be an intuitive in-game HUD display showing critical gaming info while user is in game. It is of medium priority.

3.6.2 Stimulus/Response Sequences

When user enters game, an UI with gaming instructions will show up. When user is in-game, he/she will see the in-game HUD display at handy positions.

3.6.3 Functional Requirements

U-3: Make sure UI scene is displayed correctly.

3.7 Explosion effect and particle system

3.6.1 Description and Priority

When user eats the bread, he/she needs some responsive hints that the bread is eaten. A particle system aided explosion effect will give user that hint. The priority is medium.

3.6.2 Stimulus/Response Sequences

When user walk pass a bread, a explosion effect will display at user, implying the bread is eaten.

3.6.3 Functional Requirements

U-4: A particle system must be implemented correctly to manage the particle and where to spawn them.
F-9: The particle system needs to be managed by a separate class.
P-1: The particle system must be implemented efficiently so that it will not affect other part of the system.

4. External Interface Requirements

4.1 User Interfaces

A laptop or desktop PC equipped with monitor, mouse and keyboard is essential. A black and white monitor is also playable, but may reduce gaming experience.

4.2 Hardware Interfaces

The laptop or desktop PC must equip with intel based microprocessor which supports X86 instruction sets.

4.3 Software Interfaces

The game currently only supports Windows 10 64bit Professional/Home edition. Other runtime library includes: MSVS C++ Redistributable Packages, OpenGL 3.0 or newer, IL, Assimp, freeglut, glew. It should be noted that besides MSVS C++ Redistributable Packages all other libraries are included in our release.

4.4 Communications Interfaces

There aren't any communication interfaces as currently the game is single player and does not send out the data it collects unless run in the test environment.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The game must be able to run at 30 frames per second (or above) on Windows platform OS. At this frame rate, the game will remain constant and playable. It will not be technically demanding and able to run on lower end computers.

5.2 Safety Requirements

A very small percentage of individuals may experience epileptic seizures or blackouts when exposed to certain light patterns or flashing lights. Exposure to certain patterns or backgrounds on a computer screen or when playing video games may trigger epileptic seizures or blackouts in these individuals. These conditions may trigger previously undetected epileptic symptoms or seizures in persons who have no history of prior seizures or epilepsy. We have paid attention to not create drastic changing scene and avoid inconsistent refreshing frame rates which may introduce unwanted photosensitivity seizures.

5.3 Security Requirements

Security will not be a concern for this project since there is no sensitive data being stored.

5.4 Software Quality Attributes

The two main quality attributes that the Bread Eating Game will be focusing on is correctness and usability. The game must be able to provide correct rendering of every object and textures in gaming space. The collision system and physic engine must also interact properly with viewing object.

Usability is also a priority because this is a game. If the user were to be confused or annoyed by the User interface, then the game would be less enjoyable.

6. Other Requirements

None