

# Software Design Specification

## Bread-Eating Game

*Revision 1.0*



## Table of Contents

<b>1.</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	PURPOSE	1
1.2	SYSTEM OVERVIEW	1
1.3	DESIGN MAP	1
1.4	DEFINITIONS AND ACRONYMS	1
<b>2.</b>	<b>DESIGN CONSIDERATIONS</b>	<b>1</b>
2.1	ASSUMPTIONS	1
2.2	CONSTRAINTS	2
2.3	SYSTEM ENVIRONMENT	2
2.4	DESIGN METHODOLOGY	2
2.5	RISKS AND VOLATILE AREAS	2
<b>3.</b>	<b>ARCHITECTURE</b>	<b>2</b>
3.1	OVERVIEW	2
3.1.1	SEQUENCE DIAGRAM	2
3.1.2	CLASS DIAGRAM	3
3.2	DETAILED DESCRIPTION OF COMPONENTS	3
3.2.1	COMPONENT TEMPLATE DESCRIPTION	3
3.2.2	SCENE CONTROLLER	4
3.2.3	FPSCAMERA	5
3.2.4	PHYSICS ENGINE	5
3.2.5	MODEL	6
3.2.6	PARTICLE SYSTEM	7
3.2.7	SHADER	7
3.3	STRATEGY	8
<b>4.</b>	<b>USER INTERFACE DESIGN</b>	<b>8</b>

---

# Revision History

Version	Name	Reason For Changes	Date
1.0	Wang Yinghao	Initial Revision	23/06/2017

# Approved By

*Approvals should be obtained for project manager, and all developers working on the project.*

Name	Signature	Department	Date
Mo Haoran		Development Team	23/06/2017

# 1. Introduction

## 1.1 Purpose

This design specification will detail the implementation of the requirements as defined in the Software Requirements Specification – Bread-Eating Game.

## 1.2 System Overview

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.

## 1.3 Design Map

*Summarize the information contained within this document or the family of design artifacts. Define all major design artifacts and/or major sections of this document and if appropriate, provide a brief summary of each. Discuss any significant relationships between design artifacts and other project artifacts.*

## 1.4 Definitions and Acronyms

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.

# 2. Design Considerations

## 2.1 Assumptions

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.

## **2.2 Constraints**

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.

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. Mainly photosensitivity seizures. In our design, we have to restrict the frame rate so that the user will not be exposed to inconsistent frame rate which may introduce such unwanted results.

## **2.3 System Environment**

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.

## **2.4 Design Methodology**

Structural programming with native opengl and sequential render.

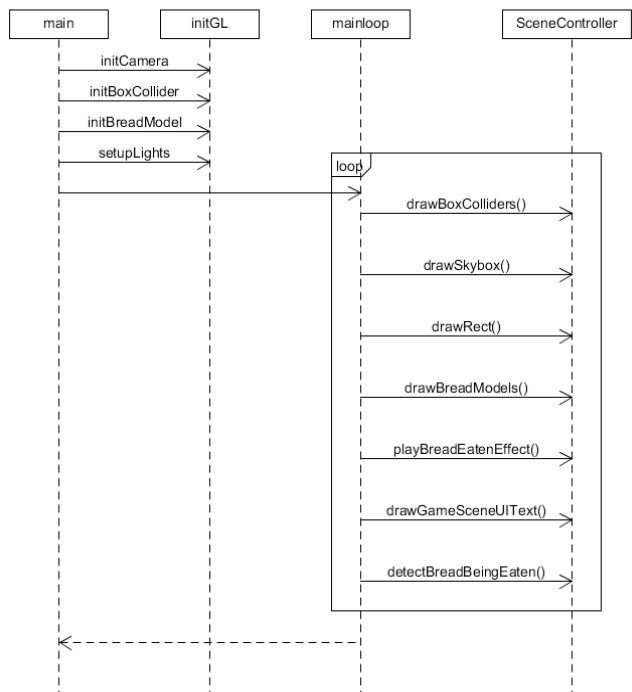
## **2.5 Risks and Volatile Areas**

None have been identified.

# **3. Architecture**

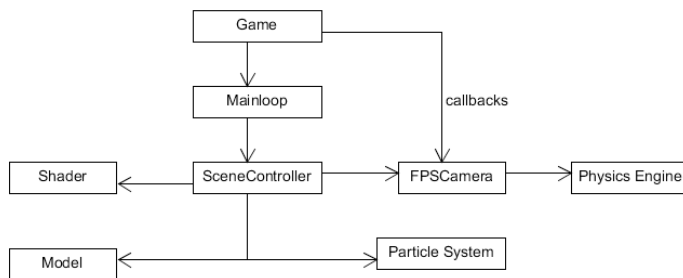
## **3.1 Overview**

### **3.1.1 Sequence Diagram**



The main render process is controlled by one main loop. The program ends with the termination of the loop. Besides, we used object oriented method to maintain modular principle and local data of certain functionalities.

### 3.1.2 Class Diagram



## 3.2 Detailed description of components

### 3.2.1 Component template description

<b>Identification</b>	The unique name for the component and the location of the component in the system.
<b>Type</b>	A module, a subprogram, a data file, a control procedure, a class, etc
<b>Purpose</b>	Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS, but are implied or adjunct to formally stated SDS requirements.
<b>Function</b>	What the component does, the transformation process, the specific inputs that are processed, the algorithms that are used, the outputs that are produced, where the data items are stored, and which data items are modified.
<b>Subordinates</b>	The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part.
<b>Dependencies</b>	How the component's function and performance relate to other components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components.
<b>Interfaces</b>	Detailed descriptions of all external and internal interfaces as well as of any mechanisms for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here.
<b>Resources</b>	A complete description of all resources (hardware or software) external to the component but required to carry out its functions. Some examples are CPU execution time, memory (primary, secondary, or archival), buffers, I/O channels, plotters, printers, math libraries, hardware registers, interrupt structures, and system services.
<b>Processing</b>	The full description of the functions presented in the <i>Function</i> subsection. Pseudocode can be used to document algorithms, equations, and logic.
<b>Data</b>	For the data internal to the component, describes the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary.

### 3.2.2 Scene Controller

<b>Identification</b>	Scene Controller
<b>Type</b>	Global Function Set

---

<b>Purpose</b>	Organize scene related codes
<b>Function</b>	Provides all basic entry points that helps render the scene, setup game rules, play particle effects, draw UI text, detect scene changes. Separate menu class instances for each menu screen
<b>Subordinates</b>	None, scene controller is the top level game manager.
<b>Dependencies</b>	Model, FPSCamera, Shader, ParticleSystem
<b>Interfaces</b>	Init*, Draw*.
<b>Resources</b>	glew, freeglut, glm functions
<b>Processing</b>	None aware of. Scene controller should focus on draw scene and game flow control, heavy processing should be separated with threading.
<b>Data</b>	Game object positions, attributes, game status counters, enumerations
<b>Interaction</b>	Scene Controller does not implement every function to its most detailed level, some of the functions are performed in separate modules and later called from this controller.


### 3.2.3 FPSCamera

<b>Identification</b>	FPSCamera
<b>Type</b>	Class
<b>Purpose</b>	Abstract interfaces related to viewing
<b>Function</b>	Provides functions of moving camera around, as well as implementation of input handling – input that changes view.
<b>Subordinates</b>	None
<b>Dependencies</b>	None
<b>Interfaces</b>	keyPressed(), keyUp(), getForward()
<b>Resources</b>	glew, freeglut, glm functions
<b>Processing</b>	Update current camera viewing angle and position according to user input. Compute and update current view matrix.
<b>Data</b>	Camera Position, View Angle, View Matrix, Projection Matrix.
<b>Interaction</b>	Control callback function will interact with this function to provide basic user input handling.

### 3.2.4 Physics Engine

---



<b>Identification</b>	Physics Engine
<b>Type</b>	Class
<b>Purpose</b>	Abstract interfaces related to gravity and collision detection
<b>Function</b>	Physics engine defines the set of rule of collision and gravity system. It prevents the view from intercepting with the COs of the world space.
<b>Subordinates</b>	None
<b>Dependencies</b>	None
<b>Interfaces</b>	keyPressed(), keyUp(), getForward()
<b>Resources</b>	glew, freeglut, glm functions
<b>Processing</b>	Compute line intersection and determine if collision occurs.
<b>Data</b>	Gravity acceleration
<b>Interaction</b>	<p>FPSCamera holds an instance of physics engine, as it needs it to update the position of the view with gravity and detect the collision.</p>  <pre> classDiagram     class FPSCamera     class PhysicsEngine     FPSCamera --&gt; PhysicsEngine </pre>

### 3.2.5 Model

<b>Identification</b>	Model
<b>Type</b>	Class
<b>Purpose</b>	Abstract interfaces related to model and texture loading.
<b>Function</b>	Model class wraps all utilities needed to import external models with external library <i>Assimp</i> . This module provides functions to load model and textures.
<b>Subordinates</b>	None
<b>Dependencies</b>	None
<b>Interfaces</b>	loadTextures, render, useTexture.
<b>Resources</b>	glew, freeglut, glm functions
<b>Processing</b>	File loading, loading process should stay out of main loop as it is heavy.
<b>Data</b>	aiScene data, textureID map.
<b>Interaction</b>	Scene controller will need to interact with this module to load model and

	place them in proper location.
--	--------------------------------

### 3.2.6 Particle system

<b>Identification</b>	Particle system
<b>Type</b>	Class
<b>Purpose</b>	Abstract interfaces takes care of the lifecycle, simulation and rendering of all particles.
<b>Function</b>	Particle system will initialize, simulate all particle lifecycle and perform render job.
<b>Subordinates</b>	None
<b>Dependencies</b>	None
<b>Interfaces</b>	Init*, simulate, render
<b>Resources</b>	glew, freeglut, glm functions
<b>Processing</b>	It contains a list of particles under its management, and need to update their status on every frame.
<b>Data</b>	A linear list of particles
<b>Interaction</b>	Scene controller will need to interact with this module so that particle effect can be rendered.

### 3.2.7 Shader

<b>Identification</b>	Shader
<b>Type</b>	Class
<b>Purpose</b>	Abstracting shader loading, use interfaces.
<b>Function</b>	Shader module provides basic management to vertex and fragment shader. It is essential in Opengl render pipeline.
<b>Subordinates</b>	None
<b>Dependencies</b>	None
<b>Interfaces</b>	loadShaders, UseShaders
<b>Resources</b>	glew, freeglut, glm functions
<b>Processing</b>	Load, compile shaders, output compile error when there are errors.
<b>Data</b>	Shader ID.

---

<b>Interaction</b>	Scene controller need to interact with this module to manage shader lifecycle, opengl fixed pipeline and custom pipeline switching in rendering.
--------------------	--

### 3.3 Strategy

One key strategy used in native opengl programming is called “clean-up” policy. Getting into rendering pipeline we need to keep in mind about our global context and local context. Before drawing, transforms needs to be made. It should be keep in mind that whenever one finishes drawing, the function should restore global context thus it will not affect other parts of rendering.

A similar scenario includes use of VBO style rendering. When bind with the shader, it should be reminded that the shader needs to be unbind to restore global context.

## 4. User Interface Design

A simple user interface stating the rules of the game and guide the user to enter the game scene should be provided. A similar interface may look as follow:



It should be noted that artistic design is not specified. Essential elements only include play guides and game entry interaction.

The main scene of the game design should fully imitate modern FPS style gaming. A similar reference is counter strike:



Note that gun and hand is not specified as this is not a shooting game.

# Appendix A: Project Timeline