CODE REVIEW DOCUMENT

for

Virtual Museum

Theme : World War II

Version: 1.0
3th May 2017

Prepared by
Mayank Yadav (34), Harshit Bansal (26)
Nikunj Mittal (43)

IIT Guwahati

supervised by
Prof. Samit Bhattacharya

Contents

Revision History

Sno.	Date	Reason For Changes	Version
1	4/4/17	Original	1.0
2	5/4/17	Remarks from Prof. Bhattacharya	1.1

1 Introduction

1.1 Purpose

The purpose of code review for the Virtual Museum software is to discover bugs, scrutinize coding conventions, and look for potential bottlenecks and resource leakage. This document is primarily intended to reduce coding errors and help in producing high quality code.

1.2 Document Conventions

Term	Definition
User	Person who shall be using the software to experience the Virtual Museum
Object	Exhibit for display
Player	User's avatar/character in the Virtual Museum
Device	The electronic Device used to run the software

1.3 Project Scope

This software will help users to visit the museums at their ease directly from their homes and see the artifacts of historical significance in a virtual 3-D interactive environment. The product will be very useful for enthusiasts who are interested about history but are unable to find time to visit the museums and also helpful for Archaeologists and professionals to have access to information from museums all the time. For people with no knowledge ,the easy to use app will act as an interesting way to learn about the museums and help them learn about the past.

2 Code Review

2.1 Introduction

The intent of a code review is to catch bugs/issues/defects before the offending code is deployed to a production environment and to transfer knowledge of implementation details to the rest of the team. Code review involves a slow code inspection phase to check errors.

2.2 Goals of Code Review

The main reasons for performing code review is to :

- Finding bugs, since bug finding in code review are easier to find and fix than later in testing.
- Adherence to coding conventions
- Improving code quality
- Increasing efficiency, by finding trivial programming errors like data resource wastage or use of uninitialized variables.

Code Review primarily consists of Code Inspection.

2.3 Code Inspection

The aim of code inspection is to discover some common types of errors caused due to oversight and improper programming. In this phase, team members are selected and asked to perform the inspection, which involves first checking of the coding standards, then, the presence of certain kinds of errors such as modifying a formal parameter while the routine calls a constant parameter. Considering the statistics some common programming errors which we checked are :

- Incompatible assignments
- Non terminating loops
- Jumps into loops
- Improper storage allocation and deallocation

- Array indices out of bounds
- Use of incorrect logical operators or incorrect precedence among operators
- Mismatches between actual and formal parameter in procedure calls
- Use of uninitialized variables
- Comparison of equality of floating point variables
- Improper modification of loop variables

The report is then given to the author for appropriate changes to be done.

2.4 Team Details

Sno.	Name	Details
1	Chandan Satti	CSE
2	Mohit Kumar	CSE
3	Anup Aggarval	CSE

2.5 Coding Standard

The general coding standard is followed in group consensus some representative coding standards are as follows :

- *Rules for limiting the use of global*: These rules list what types of data can be declared global and what cannot.
- *Naming conventions*
  - Variable names should be lowercase without punctuation, e.g. rowName, columnCount.
  - Class and interface names should be capitalized (using CamelCase), begin with a letter, and contain only letters and numbers.
- Do not leave commented out code in the source files.
- *Excpetion Handling* : Catch specific exception rather than high level exceptions that can mask errors. Also the way errors are reported in functions is noted.
- Do not use a coding style that is too clever or too difficult to understand
- Do not use an identifier for multiple purposes : Sometimes several temporary entities are stored in the same variables citing memory efficiency. But problems associated with this are :
  - Variables cannot be identified since their naming would be the same and it makes it difficult to understand code

  - Error correction in such cases or further enhancements are then difficult

- The code should be well-documented.
- The length of any function should not exceed 20 source lines.
- Do not use goto statements.

2.6 Modules Tested

Module Id	Module Name
ACC	Acceleration Input
SEL	Select-Object Input
JOY	Joystick Input
ZOO	Pinch Zoom
PC	Player Controller
CC	Camera Controller
UIL	GUI List
OBV	Object Viewer
OBR	Object Rotate
OBT	Object Translate
WLM	Wish list Manager
SCR	Screen Resolution
MMP	Mini-Map
SM	Scene Manager
PM	Pause Module

2.7 Team member reports

2.7.1 Member 1 : Chandan Satti

General Impression

1. The class declarations are capitalized using CamelCase and start with letters.
2. Variable Declarations are grouped together resulting in neater comments.

3. The variables are understood on first look like startedOverJoyStick in Module JOY is a bool which stores if the touch started over joystick area.
4. The Function names are very descriptive , for e.g. GetChildObjectWithTag() in Module OBV.
5. Comment is present for every class instantiation.
6. Non-Class objects accessed in class are defined separately
7. The functions do not span large lines of code and are exactly doing what the function name denotes

Defects

- Use of Libraries imported is not commented properly.
- Module ACC
  1. Variables 'zoomScript' and 'currentAcc' not initialized anywhere in the program.
  2. Purpose of library function 'Lerp' not very clear through comments.
- Module SEL
  1. Variables 'textObjectSelected' , 'fpsCam' and more not initialized.
  2. 'SelectObject' function exceeds the standard lines limit.
  3. Header Documentation does not include definitions of all public functions
- Module PC
  1. Inappropriate public declaration of variables such as 'speed','jumpSpeed','gravity' ,'speedH' as they are not modified anywhere in the entire module.
  2. Redundant Set function for public variable jumpInput.
- Module CC
  1. Function 'LateUpdate' does not have sufficient documentation.
- Module MMP
  1. Module does not follow the standard notation for variables.
- Module OBV
  1. Variable 'resetPos' is used without initialization
  2. Function 'GetObjectToBeViewed' spans more than 20 lines.
  3. Use of 'Resources' class is not clear in function 'GetDescription'.

2.7.2 Member 2 : Mohit Kumar

General Impression

1. Modules UIL,ZOO,OBR,OBT,WLM are very well commented.
2. Standard rules for variable and class naming are followed.
3. Variable Declarations are descriptive and tell their uses in the Module.
4. Naming of variables is nicely done and is very easy to understand.
5. Comments are present for every class instantiation.
6. Public objects from outside the class accessed in class are defined separately.

Defects

- Module PC
  1. Variable jumpInput is declared as public, so JumpInput() which assigns value to the aforementioned variable is useless.
- Module SEL
  1. Header Documentation does not include definitions of all public functions
  2. SelectObject() is larger than the optimal size.
- Module CC
  1. LateUpdate() does not have sufficient documentation.
- Module JOY
  1. Working of MoveJoystick() is not explained properly
- Module ACC
  1. Purpose of library class Lerp not very clear through comments.
- Module OBV
  1. Variable resetPos is used without initialization
  2. GetObjectToBeViewed() is larger than the optimal size.
  3. Use of class Resources is not clear in GetDescription().
- Module MMP
  1. Variables are not in Standard notation

2.7.3 Member 3 : Anup Aggarwal

General Impression

1. All Modules follow required naming standards.
2. Class and methods start with Capital and use the Camel notation as in SelectObject() in Module SEL
3. Comments are comprehensive for most modules.
4. Naming of variables is nicely done and is very easy to understand.
5. Comments are present for every class instantiation.
6. Array indices do not go out of bounds
7. All Assignments are compatible
8. Nicely written Code

Defects

- Module SEL
    1. Function SelectObject() is larger than the optimal size.
    2. Header Documentation does not include definitions of all public functions
- Module PC
    1. Variable jumpInput should be declared Private as JumpInput() can be used to input value from outside the class.
- Module CC
    1. LateUpdate() does not have sufficient documentation.
- Module ACC
    1. Purpose of library class Lerp not very clear through comments.
- Module OBV
    1. Variable resetPos is used without initialization
    2. Use of class Resources is not clear in GetDescription().

3 Conclusion

- Imported libraries are not commented properly.
- Except Module MMP, all Modules follow proper naming conventions.
- Some Functions are larger than optimal size.
- Better Documentation is required for some functions.
- Note : Some Public Variables are uninitialized , but are initialized in the Unity Editor.

References

[1] Software Requirement Specification for Virtual Museum *Version 1.2, Dated: 15/2/2017*

[2] Unity User Manual `https://docs.unity3d.com/Manual/index.html`

[3] Unity Scripting Reference `https://docs.unity3d.com/ScriptReference/index.html`

[4] Share Latex Learn Documentation `https://www.sharelatex.com/learn`

[5] Stack Overflow Forum `https://stackoverflow.com/`

[6] Unity Forums `https://forum.unity3d.com/`

[7] Unity Tutorials `https://unity3d.com/learn/tutorials`