DigiPen Institute of Technology

CSD1130

Game Implementation Techniques

Assignment 1

Game Flow – Game State Manager

Due Date

Friday, Jan 20th, 2023, at 3:00pm (Section **A**)
Friday, Jan 20th, 2023, at 5:00pm (Section **B**)

Topics

The assignment will cover the following topics:

- Implementing the game state manager
- Implementing the game flow
- Adding two levels

Goal

The goal of this assignment is to have a **game engine** which can switch between two levels and restart the same level, while displaying the correct order of state function calls (Load, Initialize, Update, Draw, Free, Unload).

Submission Guidelines – Visual Studio [Using Microsoft C++ Compiler]

- Submit at Moodle entry: "**csd1130_Assignment_1_Submission_VisualStudio**"
- To submit your programming assignment, organize a **folder** consisting of the following:
  - The solution "**.sln**" file.
  - The Project folder named "**csd1130_gsm**".
  - Any other folder that your project depends on.
- In other words, *your submission must be ready for compiling and linking by the grader*.
- Your **folder** must not contain *Debug*, *Release*, *x64* and *x86* folders, object files or executable files.
- Your **folder** must not contain a **.db*, **.opendb*, **.sdf* or **.opensdf* files.
- Do delete the "**.vs**" read-only file. (*make sure windows explorer can show hidden files. ".vs" folders are usually set to hidden*). This will reduce the size of your submission file by a lot.
- All the data text files (.txt) used, must be located near the project file "**.vcxproj**".
- Do not submit the "***Output_To_Match.txt***" file.
- When running the project, the output must be printed onto to console.
- Re-Name this **folder** using the following convention:
  - **<class>_<section>_<student login name>_<assignment#>_<part#>**. For example, if your login is ***foo.boo*** and you are from section A, and you are submitting assignment 1 part 2, your **folder** must be re-named, all small letters, as: **csd1130_a_foo.boo_1_2**
- Zip this **folder** and name the resulting file using the following convention:
  - **<class>_<section>_<student login name>_<assignment#>_<part#>.zip.** For example, if your login is ***foo.boo*** and you are from section A, and you are submitting assignment 1 part 2, your zipped file must be named as: **csd1130_a_foo.boo_1_2.zip**, all small letters.
- Next, upload your zipped file into your course Moodle website using the link: https://distance3.sg.digipen.edu.
- Finally, perform a sanity check to determine if your programming submission follows the guidelines by downloading the previously uploaded zip file, unzipping it, then compiling, linking, and executing it.

Submission Guidelines – VPL [Using g++ Compiler under A1 – VPL submission]

- Submit under this VPL entry, after you finish and submit your work under the Visual Studio entry!
- Submit at Moodle entry: "**csd1130_Assignment_1_Submission_VPL**"
- To submit your programming assignment:
    - Upload all your '.h' and '.cpp' code files, done in the visual studio project
    - Run
        - Correct run output:
            - Console output will match "***Output_To_Match.txt***" file.
    - Evaluate
        - Correct evaluation will give the following result:
            - **Summary of tests**
            - [1 test run/ 1 test passed]

<u>Using the right Compiler and Microsoft Visual Studio setup</u>

- The project must be tested in **RELEASE/DEBUG** modes with **warning level 4**, under **x64** platform. Under project settings the **SDK Version**: must be set to *<u>10.0 (latest installed version)</u>* and the **Platform Toolset** set to *<u>Visual Studio 2019 (v142)</u>*. It must generate 0 warnings and 0 errors. This can be verified on a PC located at **<u>Pascal lab</u>**.
- Please validate the previous statement before your submission!

Description

I. This project must be implemented as a Win32 **Console** Application.
Make sure to start from the template project provided.
This project is compatible with Microsoft Visual Studio **2019**.

II. Language: **C**/C++

III. Some functions will have to print a specific string to the console.

IV. The engine should have a header file which includes an enumeration containing all the possible states (GS_LEVEL1 and GS_LEVEL2) and the other special states (GS_RESTART and GS_QUIT). This enumeration is added to:

- **GameStateList.h** file.

V. You need to implement a game state manager as follow:

- The game state manager is responsible for switching states and calling the correspondent state functionalities.
- Remember that the game state manager has a **function pointer** to each state function.
- These function pointers should be set to the appropriate functions each time we switch to a new state (level). This is done inside the *update* function of the game state manager.
- The game state manager uses three variables to determine the *previous*, *current,* and *next* states.
- Two functions are needed for the game state manager:
  - **GSM_Initialize** function: Sets the previous, current and next indicators to the same value (The initial state, which is input as an argument to this function) The game state manager initialization function should print "GSM:Initialize" string to the console.
  - **GSM_Update** function: Sets the six functions pointers to the currently selected state. The game state manager update function should print "GSM:Update" string to the console.
- The game state manager should be implemented in two files:
  - **GameStateManager.cpp**
  - **GameStateManager.h**

VI. Two levels should be implemented in the assignment as follow:

- Each level (game state) will have its own state functions to load, initialize, update, draw, free, and unload its data.
- For now, each of these functions will print its functionality to the console as follow:
  - Loading level 1 should print "Level1:Load"
  - Initializing level 1 should print "Level1:Initialize"
  - Updating level 1 should print "Level1:Update"
  - Drawing level 1 should print "Level1:Draw"
  - Freeing level 1 should print "Level1:Free"
  - Unloading level 1 should print "Level1:Unload"

  - Loading level 2 should print "Level2:Load"
  - Initializing level 2 should print "Level2:Initialize"
  - Updating level 2 should print "Level2:Update"
  - Drawing level 2 should print "Level2:Draw"
  - Freeing level 2 should print "Level2:Free"
  - Unloading level 2 should print "Level2:Unload"

  Note that the double quotes "" should not be printed on the file.

  Remember that these six functions will not be called directly: They will be called through the function pointers of the game state manager. The two levels should be implemented in these four files:

  - **Level1.cpp**
  - **Level1.h**
  - **Level2.cpp**
  - **Level2.h**

VII. A system handler should be implemented (*In later projects, when we will use Alpha_Engine library, the system will handle initializing the input manager, the graphic manager etc…*) as follow:

- Two functions are needed for the system handler:
  - **System_Initialize** function: Should print "System:Initialize" to the console.
  - **System_Exit** function: The system termination function should print "System:Exit" to the console.
- The system handler should be implemented in two files:
  - **System.cpp**
  - **System.h**

VIII. An input handler should be implemented (*In future projects, when we will use Alpha_Engine library, the input handler will handle reading input from the keyboard and processing it*) as follow:

- One function is needed for the input handler:
    - **Input_Handle** function: This function should print "Input:Handle" to the console.
- The input handler should be implemented in two files:
    - **Input.cpp**
    - **Input.h**

Reminder: the double quotes "" should not be printed on the file.

IX. The game flow should be entirely implemented.

- The game flow should be implemented in the "main()" function of the application.
- In the "main()" function you will translate the pseudo-code given in the uploaded "***Lecture 3 - Game State Manager - Function Pointers - DOC.pdf***" file.
- The game flow should contain a game loop which will be able to handle all the functionalities described in the game flow chart found in the course.
- You do **not** need to include the *FrameRateController* functions.
- The game flow should be implemented in one file:
    - **csd1130_gsm.cpp**

<u>State Description</u>

I. Each level has one or two variables. These variables should be loaded/initialized from values found in text files. These text files will be supplied.

II. Level 1:
   - The 1st level has a variable called “Level1_Counter” of type integer which should be initialized in **level 1's load** function.
   - The initial value of “Level1_Counter” should be fetched from a file named “Level1_Counter.txt” (Supplied)
   - “Level1_Counter” should be decremented by 1 in **level 1's update** function.
   - When “Level1_Counter” reaches 0, the game should switch to Level 2.

III. Level 2:
   - The 2nd level has 2 variables: “Level2_Counter” and “Level2_Lives”, both of type integer.
   - “Level2_Counter” should be initialized in **level 2's initialize** function. Its value should be fetched from a file called “Level2_Counter.txt” (Supplied)
   - “Level2_Lives” should be initialized in **level 2's load** function. Its value should be fetched from a file called “Level2_Lives.txt” (Supplied)
   - “Level2_Counter” should be decremented by 1 in **level 2's update** function.
   - When “Level2_Counter” reaches 0, “Level2_Lives” should be decremented by 1. Here we have two options:
     - If “Level2_Lives” is equal to 0, the game should exit.
     - If “Level2_Lives” is still greater than 0, level 2 should be restarted.

Testing

Here is the full console output, that you can also find it in the "**Output_To_Match.txt**" file, after running the application, assuming:

- Level1_Counter's initial value is 3.
- Level2_Counter's initial value is 2.
- Level2_Lives' initial value is 2.

```
System:Initialize
GSM:Initialize
GSM:Update
Level1:Load
Level1:Initialize
Input:Handle
Level1:Update
Level1:Draw
Input:Handle
Level1:Update
Level1:Draw
Input:Handle
Level1:Update
Level1:Draw
Level1:Free
Level1:Unload
GSM:Update
Level2:Load
Level2:Initialize
Input:Handle
Level2:Update
Level2:Draw
Input:Handle
Level2:Update
Level2:Draw
Level2:Free
Level2:Initialize
Input:Handle
Level2:Update
Level2:Draw
Input:Handle
Level2:Update
Level2:Draw
Level2:Free
Level2:Unload
System:Exit
```

Documentation

Each ".cpp" and ".h" file, **that you modify or add**, in your homework should include the following header at the top:

```
/* Start Header *****************************************************************/
/*!
\file           <put file name here> (i.e. csd1130_gsm.cpp)
\author         <provide your name, student login, and student id>
                (i.e. Elie Hosry, ehosry, 390666621)
\par            <provide your email address> (i.e. email: ehosry\@digipen.edu)
\date           <date on which you created this file> (i.e. Jan 22, 20xx)
\brief          <give a brief description of this file>

Copyright (C) 20xx DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents
without the prior written consent of DigiPen Institute of
Technology is prohibited.
*/
/* End Header *******************************************************************/
```

Comments

You need to comment your code. The comments help in explaining what a variable or a function or any other module in the code does?

Example1: Code comments (when calling/using a function):

```
Input_Handle(); // To handle the input from mouse and keyboard devices
```

Example2: Functions comments (before the definition or declaration of a function):

```
// ---------------------------------------------------------------------------
// This function loads all necessary assets in Level1
// It should be called once before the start of the level
// It loads assets like textures, meshes and music files etc…
// ---------------------------------------------------------------------------
void Level1_Load()
{
        …
        …
}
```

NOTE1: For functions comments, they can be added either on the .h files or on the .cpp files only.
NOTE2: You can follow the Doxygen documentation format if you like so.

Evaluation

Here are the most common reasons assignments are marked down:

- Project does not build.
- Project does not build without warnings.
- One or more items in the “Submission Guidelines” section was not satisfied.
- A fundamental concept was not understood.
- Code is sloppy and hard to read (i.e. indentation is not consistent, no comments, etc…).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, etc…
- Project assignment was turned in late.

Grading

This project will be graded according to the following rules. Any rule that is missing, letter grades will be deducted:

- Not following the submission guidelines (from above section). This includes wrong files naming, wrong file's locations, missing essential files…
- Missing header comment, at the top of the modified/added only ".h" and ".cpp" files
    - For any number of files
    - Penalty applies, only, on the files that the students update or add
- Header comment does not match the given one, or missing essential information as described under "Documentation"
    - For any number of files
    - Penalty applies, only, on the files that the students update or add
- Does not compile for some reason!
    - Must verify that it builds/compiles before submission. You must verify on a Pascal lab's machine
- Compile Warnings (Unique Instances)
- Output not matching the given one (or runtime stuck in endless loops, or crashing at runtime)
    - Make sure you match your console output with "**Output_To_Match.txt**" file
    - Capital letters and small letters must be matched!
- Checking the source code:
    - No comments
    - Poor comments
    - No consistent indentation
    - No appropriate identifiers
    - Console Window contains unnecessary debug information, and not cleaned

Notes for Moodle submissions:

- Visual Studio submission is counted as the full score for A1
- VPL submission is not counted for A1 grading!
    - It is a first test for us, for future graded VPL assignments

Notes for late submissions:

- Within two days (48 hours) after the due date/time (-10% per day)
- Assignments turned in more than two days late will not receive credit! It will receive an "F" grade