Fall 2011
CS 529 | Fundamentals of Game Development
Project 1 | Game Flow – Game State Manager – Win32

Part 1 Files (submit folder) due
- Monday, September 19, 2011
- 11:55pm

Part 2 Files (submit folder) due
- Wednesday, September 21, 2011
- 11:55pm

Topics
The assignment will cover the following topics

I. Implementing the game state manager
II. Implementing the game flow
III. Adding 2 levels
IV. Incorporating the game engine flow into a Win32 application

Goal
- The goal of this assignment is to have a game engine which can switch between 2 levels and restart the same level, while displaying the correct order of state function calls (Load, Initialize, Update, Draw, Free, Unload).
- The game engine flow will also handle windows events.

Assignment Submission
- Compress (.zip) the solution folder (Delete the debug/release folders and the .ncb file first), and submit it on distance.digipen.edu.

- Your submitted assignment should use the following naming convention:
  <student login name>_<class>_<assignment#>_<Part#>.zip

- Example: John Smith should submit: john.smith_CS529_Project1_Part1.zip

Description

I. Language: C/C++

II. The project is divided into 2 parts
   a) Part 1: Implement the game state manager and the game flow
      - This part should be implemented as a Win32 Console Application
      - Make sure to select "Empty Project" when creating the project in Visual Studio
   b) Part 2: Incorporate the game flow into a Win32 application (Provided)

Part 1

I. Some functions will have to append a string to a file called "Output.txt". This "Output.txt" file should be created and opened in the system initialization function, and it should be closed in the system exit function.

II. The engine should have a file which includes an enumeration containing all the possible states (Level_1 and Level_2) and the other special states (Restart and Quit).
   a) GameStateList.h

III. You need to implement a game state manager.
   a) The game state manager is responsible for switching states and calling the correspondent state functionalities.
   b) Remember that the game state manager has a function pointer for each state function.
   c) 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.
   d) The game state manager uses 3 variables in order to determine the previous, current and next states.
   e) 2 functions are needed for the game state manager
      - "GSM_Initialize": Sets the previous, current and next indicators to the same value (The intial state, which is sent as an argument to this function)
      - The game state manager initialization function should append "GSM:Initialize" to "Output.txt"
      - "GSM_Update": Sets the 6 functions pointers to the currently selected state. The game state manager initialization function should append "GSM:Update" to "Output.txt"
   f) The game state manager should be implemented in 2 files:
      - GameStateManager.cpp
      - GameStateManager.h

IV. Two levels should be implemented in the assignment.
   a) Each level will have its own state functions to load, initialize, update, draw, free and unload its data.
   b) For now, each of these functions will append its functionality on the "Output.txt" file.
      - Loading level 1 should append "Level1:Load"
      - Initializing level 1 should append "Level1:Initialize"
      - Updating level 1 should append "Level1:Update"
      - Drawing level 1 should append "Level1:Draw"
      - Freeing level 1 should append "Level1:Free"
      - Unloading level 1 should append "Level1:Unload"
      - Loading level 2 should append "Level2:Load"
      - Initializing level 2 should append "Level2:Initialize"
      - Updating level 2 should append "Level2:Update"
      - Drawing level 2 should append "Level2:Draw"
      - Freeing level 2 should append "Level2:Free"
      - Unloading level 2 should append "Level2:Unload"

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

      Remember that these 6 functions will not be called directly: They will be called through the function pointers of the game state manager. The 2 levels should be implemented in these 4 files:
      - Level1.cpp
      - Level1.h
      - Level2.cpp
      - Level2.h

V. A system handler should be implemented. (In later projects, the system will handle initializing the input manager, the graphic manager...)
   a) 2 functions are needed for the system handler.
      - "Systeme_Intialize": Besides actually creating the "Output.txt" file, should append "System:Initialize" to "Output.txt"
      - "System_Exit": The system termination function should append "System:Exit" to "Output.txt"
   b) The system handler should be implemented in 2 files
      - System.cpp
      - System.h

VI. An input handler should be implemented (In later projects, the input handler will handle reading input from the keyboard and processing it)
   a) 1 function is needed for the input handler.
      - "Input_Handle": The input update functions should append "Input:Handle" to "Output.txt"
   b) The input handler should be implemented in 2 files:
      - Input.cpp
      - Input.h

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

VII. The game flow should be entirely implemented.
   a) The game flow should be implemented in the "main()" function of the application.
   b) 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.
   c) The game flow should be implemented in 1 file:
      - Main.cpp

State description

I. Each level has 1 or 2 variables. These variables should be initialized from values found in text files. These text files will be supplied.

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

III. Level 2
   a) The 2nd level has 2 variables: "Level2_Counter" and "Level2_Lives", both of type integer.
   b) "Level2_Counter" should be initialized in level 2's "initialize" function. Its value should be fetched from a file called "Level2_Counter.txt" (Supplied)
   c) "Level2_Lives" should be initialized in level 2's "load" function. Its value should be fetched from a file called "Level2_Lives.txt" (Supplied)
   d) "Level2_Counter" should be decremented by 1 in level 2's "update" function.
   e) 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 Part 1

Here's the output that you should find in your "Output.txt" file after running the first part of 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
```

Part 2

I. Once you have the game engine flow working properly, merge it into the provided Win32 application.

II. Windows events have to be handled during **every state loop**
   a) **Make sure to review the difference between "GetMessage" and "PeekMessage"**

III. In the window procedure function of the provided Win32 application, make sure to quit the application when the "Escape" key is pressed

```
case WM_KEYDOWN:
                if(wParam == VK_ESCAPE)
                        ;       // Quit the application by setting the next game state
                                // indicator to the QUIT "state"
                        break;
```

   a) This should exit the project when the escape key is pressed

Testing Part 2

a) Level1's counter and level2's counter/lives were set to very small values in the previous part in order to make sure that the game flow is working properly.
b) Change one of them to 10000 in order to allow the application to run for few seconds, giving you enough time to check whether pressing the "Escape" key is closing the application.
c) Once the application closes after pressing the "Escape" key, open "Output.txt":
   - Do not check the entire content of the file!
   - Just scroll down to the end and check if the application was closed properly, by making sure that the running level's free and unload functions were called, followed by the system exit function.

Finally, each ".cpp" and ".h" file in your homework should include the following header:

```
/* Start Header -------------------------------------------------------

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.

File Name:     <put file name here>
Purpose:       <explain the contents of this file>
Language:      <specify language and compiler>
Platform:      <specify compiler version, hardware requirements, operating systems>
Project:       <specify student login, class, and assignment. For example: if foo.boo is
               in class CS 529 and this file is a part of assignment 3, then write:
               CS529_fooboo_3>
Author:        <provide your name, student login, and student id>
Creation date: <date on which you created this file>

- End Header --------------------------------------------------------*/
```