CSC 581 Homework 5 Reflection

Jae Jimmy Wong

1 Introduction

This document is the reflection on the overall design of the game engine. The engine has evolved over the course of the semester and has a lot of changes since the start. Plenty of functions has been added to the engine also some of them is rewritten to fit the requirements of new games. The section 2 focused on the reusability of the code in the engine. In section 3, I discussed some of the improvements that can be done if starting over again. The section 4 uses diff tools to calculate the actual number of lines of code is changed between each game.

2 Code Reuse

There are five main components in this game engine: client-server communication, timeline, event management system, world object model, and scripting system. The first three components are fully reused in the second game and the third game. No modification is required to reuse their code while making new game.

However, the world object model and the scripting system requires some changes to fit the need of new games. In the design of this world object model, each type of object has their own C++ class. The design increases the memory space efficiency of world objects, but it also makes game objects harder to be reused by other games. For example, the "flappy bird" game has vertical pipes which do not exist in the first platformer game, so a new C++ class to need to be added to implement the object.

Since the scripting system has to export the world object into the V8 environment, any changes in the world object model result in some modifications in the code of the scripting system. For example, the "ns-shaft" game has a score property on world object that does not exist in the first platformer game, hence a new pair of V8 accessors needs to be added to the code script manager. In case of complex objects with plenty of attributes, it would take hundreds lines of code to make it accessible to the JavaScript scripting environment.

To summarize, there are two places in the game engine code that I changed while creating new game. The first one is the definition of the world object. I added the newly defined game object types to the structure of the world object. The second one is the implementation of the script manager. I added a getter callback and a setter callback for each property on the new game object types.

3 Possible Improvement

If I am going to start over to design the game engine again, there are many design decisions that I would like to change. Here I listed the most important ones:

- Modularization of server-client communications

  In the current design, the code of server-client communication is scattered inside the source files of the server and the client. Though it can be reused in other games without modification, the design makes the server source file more lengthy and less readable.

  A better design would be wrapping all the code related to network communication into C++ classes. This way not only increases the readability of the source files, but also makes it easier to create more communication channels between the server and the client.

- Decoupling event management and world object model

  The current event management system is tightly related to the world object model. When event system handles events, it takes a world object and makes changes to it. The design was mean to facilitate the replaying function, because it requires multiple different versions of world to coexist in the server. Though it did achieve the replaying function, there are two downsides that is created by this design.

  First, it makes the code more complex and more confusing. For example, the code of the server contains some scenarios that needs to use the global world object and others that needs to use the local world object from the function arguments.

  Second, it increases the difficulty to add new features to the event management system. For instance, a rewrite of the event system is necessary to let event handlers to access objects outside the world object.

- Using key-value mapping to store object attributes

  Although using C++ classes to define object type results in better performance and smaller memory footprint, it makes functions related to the object model more difficult to be reused. The best example is the binding between V8 scripting environment and the world object. By switching to storing attributes in key-value mapping, all accessors of the world object can be created automatically without writing any additional code.

4 Code Difference

There are a total of 855 line of source codes in the first platformer game. While comparing with the second “ns-shaft” game with a diff tool, I found 141 lines of deletions and 110 lines of insertions. Therefore, the code in the first game that was changed is around 16.5 percent. The number shows that most part of the game engine is reused in the second game.

As for the third “flappy bird” game, the deletion is 173 lines and the insertions is 143 lines. This leads to a higher modification percentage of 20.1 which is still representing a high ratio of reusing code.