**How well designed was the code for extensions, what particular elements aided or hindered extensibility?**

In the code base A, the clear structure of the code can be recognized by the correct use of design pattern in the code base. That really makes it easy and convenient to understand how each method works and the relation between different classes. Also, it uses many advanced tools and methods provided either by QT or C++, those tools include unique\_pointer , vector and different QT types. By using those, the code becomes more understandable and easy to make modification and improvement as well. It means that you can access or manage those type objects in a pleasant way without defining methods myself.

But there are some parts of the code base can prevent the other people from working further on it. The name of some classes is confusing, for example, the concrete class is called ‘simple’ class. Also, in builder pattern, the abstract builder class and the concrete builder class are named creator and simple creator respectively. All those will decrease the efficiency of developing.

**How well document was the code with respect to both external documentation and comments?**

There are only a few comments beside each method in the code base, it could be hard to figure out some method’s function. But due to the good implementation of smart pointers and STL container, having a good understanding of the code doesn’t require too many comments. On the other hand, the external documentation by Doxygen is really helpful, actually, I looked into the htlm file generated by Doxygen when I tried to figure out the structure of the code, and it turns out that it does help. Each method in every class has its own brief description, parameter and return units showed on the web page.

**Was the coding well done? What would you have done differently? What was good/bad about the implementation?**

Considering that this code is programmed by my peer at the same degree, it is of good quality and much better than my stage 1 code. However, if some issues of the code can be resolved, it will be more impressive. In this code base, its lack of encapsulation of some class is the first problem I would try to improve. By this I mean some attributes of the class are public rather than private or protected damaging the encapsulation. Other than this, the implementation is quite good. Especially, except the good use of smart pointer and STL container in this code, it tries to define the frame rate and the tick rate in different speed in the util.h file. What it tries to fix is that sometimes balls could overlap a little before the collision or even just pass by if the frame rate is not optimized enough. Besides, compared with my implementation, it reads the configuration file and builds object together inside the builder (creator here). Also, the dialog (mainwindow here) just uses the pointer of a scencemanager object to manage the game rather than seeing the whole content of the game directly.

**Comment on the style of the code. Were names, layout, code clichés consistent?**

The code style is slightly different from mine, it uses more space and between every two words which is a pretty good way of coding and good for viewers. For the names, layout, code clichés, the confusion about the naming of some classes I have mentioned above. For code cliché, I don’t like the way the designer of code base deals with parenthesis, because sometimes the too many brackets on the same line without spaces would prevent efficient reading and understanding. In this case, I would prefer putting each of those brackets on one line.

**Explain the application of the design patterns for your code.**

In my code, on the base of the code base A, I use the decorator pattern and the composite pattern to help build my stage 2 pool game.

For the decorator pattern, I use it for adding the pockets on the table so that balls can fall into those pockets and disappear then. In theory, The decorator pattern is to add additional responsibilities to an object dynamically. As the UML diagram shows, in my code, I create a new class named PocketTable that has two member variables (pockets and table). The table variable is a reference to a simpleTable object which is from the base code, so I can add additional functions on the object without inheritance or changing the whole data structure of it. And the pockets variable and related methods to manage the pockets are the additional responsibilities for the SimpleTable class. The pockets member variable is a vector of pointer to pocket objects which is added by the decorator for removing balls from the table in the pool game.

For the composite pattern, according to the composite design pattern structure, the leaf and composite should both inherit from the same base class, but if we visualize the dimensional structure as a tree, the composite class should be the node that has children. In my code, I build a new class named CompositeBall, it inherits from the Ball class the same as SimpleBall. And it has a vector of Balls that can contain a couple of balls either that is SimpleBall or CompositeBall. So the SimpleBall object has to be the leaf that without any child, while the CompositeBall object can have several children. By this implementation, this code can realize a recursive way of contained structure for the ball. For example, every ball could be the CompositeBall and its component ball could also be the CompositeBall.

**Explain advantages and disadvantages of the design patterns used with respect to your code.**

The advantages of using those design patterns are from different perspectives. Using decorator to add pockets to the table can make it easier to construct the table object during the construction stage and save effort for writing unnecessary code. At the same time, the structure could be clearer because two different concrete tables are both derived from the same base class. Using composite pattern is the most efficient way to make contained ball function utilizing the base code as much as possible. Without the composite pattern, it could be really hard to construct balls, when the hierarchy structure for balls are extremely complicated. Also, using composite pattern help manage the balls together like drawing the ball on the table.

The disadvantages mainly come from the complexity of the ball, using the composite pattern especially for complex structure could be easy but hard to restrict. In my code, once one particular ball goes wrong, the effort to trace the issue and resolve it is costing. When I use decorator pattern, I do not implement the abstract decorator in my code. So that could be a hindrance for further development in stage 3. Also, the type of reference in my decorator is SimpleBall rather Ball, when adding a new class of table into the code, it will be hard to modify the code.

**UML**

