Received code:

- How well designed was the code for extensions, what particular elements aided or hindered extensibility?
    1. The code was really well designed for extensions. It minimises the coupling between objects. For instance, 'Table' and 'Ball' classes do not have any connection between (their relationship is controlled by 'SceneManager').
    2. Therefore it is very easy to drive classes from 'Table' and 'Ball' without considering any extra coupling. Some classes aided extensibility such as 'SceneBuilder', 'SceneManager' and 'MainWindow'. However, there are some elements hindering extensibility. For example, the base code uses type 'unique_ptr', which is out of our knowledge, so that it does take some time to figure out how to cast unique_ptr (In assignment 2, the cast is failed and extra functions are added to help). A basic attribute, ball colour, is missed in 'Ball' class, and in order not to modify a lot of base code, some extra 'non-necessary' code is added.
- How well documented was the code with respect to both external documentation and comments?
    1. Extremely good. Almost all functions including those in drieved classes have doxygen comments to help me understand what should a function do. Some classes such as 'Ball' have specific class documentations helping next coder understands the classes.
- Was the coding well done? What would you have done differently? What was good/bad about the implementation?
    1. Yes, code is clean and easy to understand.
    2. I would not use another class to manipulate 'Table' and 'Ball', instead, I would just simply add some attributes in both classes, which were a bad design because these classes are not supposed to be aware of each other.
    3. Implementation:
        1. Good:
            1. Decoupling the object relationships, and using another class to manipulate them.
            2. Hide implementation users do not need to know from definition.
        2. Bad:
            1. For the constructors, it does set up default values, but there is only one constructor. When a value B is provided while value A is not, the constructor does not work, and the program has to declare value A outside the constructor.
- Comment on the style of the code. Were names, layout, code clichés consistent?
    1. Yes. From most names of classes, the usage can be deduced as well as variable names.
    2. The layout is clear. It performs tasks with a logical procedure.

3. Yes

My code

- Explain the application of the design patterns for your code.
    - Two design patterns were used. One is Composite Design Pattern, and another is Decorator Design Pattern.
        - Composite Design Pattern (CDP):
            - It was used to design the balls used in stage-two. Because the balls can contain other balls, and meanwhile users are supposed to manipulate such balls as those not containing balls, the CDP would be the best design pattern.
        - Decorator Design Pattern (DDP):
            - It was used to design the table used in stage-two. As the stage-two table has pockets on it, and at the same time, its other functionalities are the same as the stage-one table, using DDP decorating the table using pockets could be a better choice.
- Explain advantage and disadvantages of the design patterns used with respect to your code.
    - Advantages:
        1. The 'leaf' in this CDP is a stage-one ball. . The CDP lets users consider the composed balls as the simple balls without acknowledging how they are composed, which means users can treat stage-one balls and stage-two balls uniformly.
        2. The DDP introduces a new feature (pockets) to the table without creating a new type of table. It reduces some coding effort as these two tables are exactly the same except for the pockets.
        3. The DDP allows adding features at run time rather than going back into existing table class and making changes.
    - Disadvantages:
        1. It is possible that the composed balls could not only be composed by balls but other objects. In other words, the composed balls are restricted to only one particular type, which is not good for extensions.
        2. The DDP could potentially result in many small objects in the design. In stage two, only one type of decoration (pockets) is added, but it is possible that there were a bunch of decorations, and for each decoration, a new class needs to be defined. Therefore, it could result in many small objects.
        3. The DDP could complicate the process of instantiating the decorated table, because it not only has to instantiate the base table, but also the decorations.