Received code:

How well designed was the code for extensions, what particular elements aided or hindered extensibility? **(10%)**

There code was designed half well for extensions. The use of a default constructor for the ball class made it easier to serialize ball objects. The classes that lacked a default constructor made it difficult to perform serialization. For example, the Table, Pocket and Game class as well as its concrete implementations lacked a default constructor in addition to setters and getters hindering the ability to perform deserialization.

How well documented was the code with respect to both external documentation and comments? **(10%)**

The code had very inconsistent documentation. Some methods had documentation while there other methods that didn't. The documentation didn't just exclude the getters and setters (which is alright), but also some non-trivial methods where it was unclear what the parameters were supposed to represent.

Was the coding well done? What would you have done differently? What was good/bad about the implementation? **(10%)**

The coding was not done very well. The code did not compile on my machine without doing some massive changes. It was clear that whoever wrote the code did not have portability in mind. Portability issues aside, the implementation of most objects except the ball class seemed to follow a telescoping anti pattern. There was no default construct for any class except for the ball class. The default constructor for the Ball interface allowed it to be easily extended. On the other hand, the Initializer class (the director) appears to have changed its method to take dialog as an argument. Since dialog did not have any methods marked virtual this made it an unsuitable widget to extend.

Comment on the style of the code. Were names, layout, code clichés consistent? **(10%)**

The code style wasn't fully consistent. The naming of member variables sometimes started with the "m_" prefix and sometimes it didn't. There was also a lack of space between variables when using binary and language operators. The coding style of pointers was consistent. Consistently bad. Pointers should be next to the variable they represent not between the type and variable name.

Your code:

Explain the application of the design patterns for your code. **(20%)**

<u>Memento</u>

The memento design patter was used to save the state of the game and allow it to be restored by pressing "r" key on the keyboard. The state stored by the memento was a vector of balls. The memento "stores" by having the caretaker deep copy the balls before it passes the state onto the originator. The originator never returns the state of state of balls stored by the memento. Instead it returns a deep copy which the caretaker is free to modify.

Prototype

The prototype design pattern was applied to the ball objects. The clone operation implemented in the balls were deep copies. This deep copying was achieved through serialization of the object that was to be copied and deserialization into the object which was the copy. The prototype pattern was used to allow the memento pattern to clone the balls and save the ball states.

Explain advantage and disadvantages of the design patterns used with respect to your code. **(20%)**

Memento

Advantages

- Enables the game to perform a series of undoes. The caretaker can also be easily modified to perform a redo.
- The memento state stored by the memento is created by a deep copy which means that the original copy can continue to be modified.

Disadvantage

- The memento implementation in this code is only able to store a vector of ball. To make the memento code store the state of other objects such as pockets and table, a template method could've been used along with memento.
- The originator and memento participants require the caretaker to set a state in the originator which it will not modify.

Prototype

Advantages:

- Since deep copying was performed, objects with pointers will not have a memory problem with pointers.
- The prototype reuses code, the serialization functions and methods.
- Allows the memento pattern to be implemented with ease.

Disadvantages:

- Clients expecting a shallow copy may make unnecessary mistakes to “fix” a non-existent problem.
- A modification to the member variables of the Protoype classes could exponentially increase the complexity of the clone method to maintain deep copying.

All code:

Graphical visualisation in UML of the code. **(20%)**

See Attached Files