The objective of the section is to reflect on how efficient my game engine design was while implementing two games and the improvements that can be made.

I have designed a skeleton of basic game engine with the following components namely:- EventManager, Timeline, NetworkHandler, GameObject model, and ScriptManager functionality. Each of these components has evolved over the entire course and has let me understand the design principles of implementing a game engine. Each component in my game engine can be described as shown below:-

1) **EventManager**: It is a class that handles raised events in the game engine. It is encapsulated with basic functions that includes registering function callbacks with particular event type, storing the raised events occurring across the game engines based on their timestamps and executing the handler functions for the scheduled events.
2) **Timeline**: It is a class that handles the timeline in the game. It comprises of methods that fetch the timestamp associated with the initialized timeline as well as the system time.
3) **NetworkHandler**: It handles the network related tasks such as sending and receiving messages from the engine using ZeroMQ library.
4) **GameObject**: It is a class that handles the game object related functionality in the engine such as modifying its position, size. It's based on the *Generic Component* paradigm to make each object functionality independent of each other.
5) **ScriptManager**: It handles the script related functionality of the game engine that includes creating new scripts, compiling and bind with C++ functions.

In order to utilize the functionalities of these components, an object has to be instantiated in the main game logic for each game that is created.

To test the versatility of the game engine design, I implemented two games to test the re-usability of these components and the ease of integrating with game logic. The first single-player game named *Space Shooter*, it contains three objects namely Shooter, Bullets and Asteroid. Here, the Shooter has to destroy the Asteroid using the Bullet object. The score increases by 1 when the Bullet collides with the Asteroid. The second single-player game is a *Ping Pong* game, where there are two game objects namely Bat and Ball. Ball starts at a point and falls below and rebounds when collides with window boundaries and Bat and score increases when the Ball touches the window top boundary.

While I was implementing these two games, I had integrated the game engine components with ease apart from few minor changes in order to adjust with respect to the game's logic. In the *Space Shooter* game, I did my modifications to the **EventManager**. Because, the number of events that needs to be handled in this game is less when compared with the original game. Hence, I had to remove the unused Event type switch-case statements from the $EventManager :: handleGameEvent()$. It is similar in case of *Ping Pong* game, I had to remove the unused event types from the switch-case statement in $EventManager :: handleGameEvent()$.

I can justify these observations based on the code comparison with the respective source code files of these games with respect to the original game. To measure these changes, I made use of a library namely *cloc* that provides the code comparisons between two source code files. The following results are shown in the table below for both games:-

I also measured the number of code lines used in each of these files that are as follows:-

Parameters	Code Lines
Same	356
Modified	78
Added	76
Removed	324

TABLE 1. Code Comparison between Original & *Space Shooter* games

Parameters	Code Lines
Same	348
Modified	64
Added	71
Removed	346

TABLE 2. Code Comparison between Original & *Ping Pong* games

- Original game has 758 lines of code.
- Space Shooter game has 510 lines of code.
- Ping Pong game has 483 lines of code.

Before computing the percentage of similarity between these files, there is a minor flaw while counting these lines that is some components like EventManager, NetworkHandler and Timeline classes are included in the same file as the main game logic code. We have to assume the lines that are similar includes the code lines of these engine components as well. Based on the tables, we can infer that the *Space Shooter* game has a code similarity around 356 i.e. 70% with respect to original game. For the *Ping Pong* game, we infer that it has a code similarity around 348 i.e. 72% with respect to the original game.

On an overall view, I have implemented the functionality of each engine components keeping in mind the basic design constraints required to implement a proper working game. To raise and handle all types of events, I had implemented functions that registering function callbacks, store raised events and schedule handling of events using maps, priority queues and timestamps in EventManager component. In NetworkHandler, there are two generic functions that send and receive messages including json objects to and from server. It also includes GameObject class in a header file that can be inherited to each class that can become a game object used in a game. So, I believe due to the design principles and adhered to maintaining basic working and abstraction, therefore, it made me successful in creating these games by reusing these engine components.

Apart from the changes to the engine components, I had to additionally make new game objects and integrate them with new game logic and functionality. For instance, in the *Space Shooter* game, I had to implement three new game objects namely Shooter, Bullet and Asteroid, which inherits features from base class GameObject model. Along with this, I have to implement functions that handles spawning of Bullet objects and raise event upon collision between Bullet and Asteroid, to make the game work. Moving onto *Ping Pong* game, I had to implement two new game objects namely Bat and Ball that inherits features from base class GameObject model. I had to additionally add methods in derived game object class Ball, to change directions upon rebound, to make this work.

If I were to start over again to design my game engine, there are a few aspects that I would like to make changes. One thing that lacks in the current engine design is code maintainability. That is, if I want to add additional features to my game engine, it might give raise to bugs. I failed to grasp the importance of modularity in code. Therefore, I want to implement each component as modular as possible. Apart from this, I will use the existing component functionalities and design principles if I design a game engine again based on the above inferences.