Reflection
Qiuyu Chen

By completing this overall project, I realized that a good game engine helps a lot when building a new game. When I was building the game object model and the event management system, I had a hard time understanding the purpose of having such a model in the game engine. It was complex to do the refactoring due to some implementation of my code, and I thought it did not improve the structure of the game engine. I changed my mind after implementing the second game.

When I implemented the Space invaders, I reuse most of my code. Since Space invaders is not a 2D platform game, I excluded the platform and modified the original move component. The move component for the first game evolved the gravity and jump. What I needed to do was remove the gravity and jump components and made some changes to the keyboard control. Using the game object model made it possible to add arbitrary new components or delete components without affecting other components. I reuse the movingPlatform game object from the first game to implement the space ship, they both move in a regular component. I created a new class Fire that extends the GameObject, which represents each fire the gun turret shoots. By extendings the GameObject class, I could easily use the function that has already been designed in the GameObject class and this reduced a lot of work. Shooting was created as a new component that a character(gun turret) has. In this case, a gun turret is able to shoot out fires, which are game objects that have a rectangle shape. I created a list for holding all fires that the gun turret shoots so for each game loop, the movement of fires can be updated.

I might have to modify my event management system. I found it was not easy to integrate the new requirements to evolve the events. So it is apparent that how I implemented the event system might not provide enough information efficiency. I was not able to easily define and handle new types of events. I need to change the structure of the Event class, so it will be able to handle the necessary information for raising and handling an event.

For the client and server, I found that how I designed the way using socket to send information was not general enough. It was hard to reuse the server code I wrote for the first game. In the first game, I send the coordination of the moving platform through the server, which only supports a limited number of moving platforms. When there are a great number of space ships, it is not efficient to send each coordination. In order to deal with this, I have to change how I designed the server to make it general enough for supporting different game requirements.Multithreaded loop architecture did not modify a lot in the second game, multi threads were used to make safe updates of the character's movement and the space ship movement. However, I did not evolve the multi threads in the server side, and it should support different uses than the limited uses I implemented.

Overall, the game engine has three major components. The Game Object model that supports different components which helps modify new functionality that an object has. The event management system that can raise events with different priority and handle the event with the order of the priority. By having the event management system, we are able to control the order that an event happens instead of letting it execute immediately, this helps avoiding some conflict when modifying the same game object. The multithreaded loop architecture supports the similar use, it avoids modification of the same variable at the same time, which can easily create mess. After designing, implementing and reusing those game engine components. I realized that the design is the most important part when building a game engine. Implementing a new game without considering those game engine components will make the code complex and loose structure. There will be many repeat functions. For example, if the game engine does not have a game object model, I have to implement every similar functionality in each game object, this is time consuming and not space efficient. Besides, sometimes it was hard to do the refactor when adding new components to the game engine, this let me realize that I should fully design what I plan to do instead of just starting. A well planned design saves time to modify and add new functionality, and has the high possibility for reuse of each component. The components of my game engine that were successfully reused were the game object model and the multithreaded loop architecture. The client-server and event management needed a great number of modifications.

If I am going to start over again to design the game engine again. I will definitely spend more time designing the components of the game engine. Such as how they structure, what are the potential components that most of the game have, and how they can be reused. I will keep the similar game object model, and add more useful components. Components such as move and collide are almost suitable for all kinds of games. A well implemented game object and components helps adding new features to the game. Besides, in order to organize the components, I will update the component model to a generic component model which supports more flexible component implementation. The way that multithreaded loop architecture I implemented was easy to reuse. However, I always lost the idea of how to use multithreading. I was not very clear about the implementation that might affect sharing variables and how to split them to use the multithreaded. To solve this problem, I need to take time to comb the relationship between each action I take. A well implemented and used multithreaded loop architecture helps reduce bugs and make the function that runs in the game loop in a certain order.Since I was new to the zmq, I realized that I did not well utilize it to build a client-server system. What I designed was complex and not efficient. It only supports at most three clients. I have to study through the zmq library and design a new server which supports arbitrary connections. In addition, I need to explore a way that can send the information efficiently. The event management system is another big deal. I will spend time on it to make another design. I will change how I implement the Event class and the EventHandler. Each game object might be able to extend the EventHandler and have a different onEvent method. This helps make specific implementations of different onEvent methods.