So for this homework write up, I will go over what I did, the reasons behind my design decisions as well as the refractor I had to do for each section of this homework, which includes my implementations of four types of events, character collision event, character death event, character spawn event and user input event. As well as how I adapt the implementation of character spawn event to work with Networked Event Management.

Firstly, I started with user input events, more specifically, the implementation of the key w, a, s and d, I started with this as I believed it to be the most fundamental part of the game, as collision and death can only be achieved with movement. I decided to do the character spawn event last as I decided to tie it to the networking element of my server and client. I decided that my event manager will have a prio queue of raised events, with events being handled if the time they were raised were <= the time “screen shot” taken by the manager , a map of handlers, mapping different types of events to a handler (I decided to go with the approach of making only 1 handler with different things it can do for each type of events, I will elaborate more on this when I talk about how I implemented my events), and a mutex for security. Now, for the events, I have an event parent class, that has a private time field, which takes in the time when the event is raised, and a string private type field, this is how I will be able to tell different types of events moving forward when I inherit from this parent class. From this point, I made the EventPlayerMovement class, which inherits from the aforementioned event class, being defined in the header file as

```
class EventPlayerMovement : public Event {
    public:
        void dummy() override;
        EventPlayerMovement(sf::Keyboard::Key KeyPressed);
        sf::Keyboard::Key getChar();
        void setChar(sf::Keyboard::Key NewInput);

    private:
        sf::Keyboard::Key currInput;
};
```

The EventPlayerMovement class constructor takes in an sd::Keyboard::Key KeyPressed object, as I planned to raise a new EventPlayerMovement event every time the player presses W, A, S or D and that respective key is passed into the function, which is then set ass currInput, the private field shown in the class. Before I continue explaining my implementation of this event, I want to explains my EventHandler class, this is the definition in the header file :

```
class EventHandler {
    public:
        EventHandler(playableRec *newCharacter, landPlatform *platformNew, int deathZoneNew);
        EventHandler();
        void onEventInput(Event *e);
    private:
        //pointer to playablleChar
        playableRec *playerChar;
        //pointer to a platform
        landPlatform *platform;
        int deathZone;

};
```

I decided that since for the current plan of development, the only things that will be effected by events are the playable characters, how they collide with platforms as well as how they die, I decided that these will be the information provided to the constructor of the eventHandler (as well as setting them to their respective private fields), the empty constructor seen in the image above is meant for future implementation. The function on event input with parameter pointer to an event, is where each type of event will be handled, the handler will check the type of event and handle it accordingly. Back to how I implemented the user input event, the handler will check the event being passed to the onEventInput function for their type (this type is determined when each event is raised by the event themselves in their constructor, in the case of EventPlayerMovement, it is set as “EventPlayerMovement” like this

```
EventPlayerMovement :: EventPlayerMovement(sf::Keyboard::Key keyPressed) : Event(){
    setChar(keyPressed);
    setType( "EventPlayerMovement");
}
```

.) then the handler will cast that event to it’s respective class and do what the event is supposed to do, in this case, updating the character position. Previously, this updating was done directly in the playableRec class, however, I refractor my code and moved this implementation to the EventHandler, one problem I ran into is that the movement of the character depended on the time line of the game (in my game, this dependency is represented by a float value called movementfactor), so I had to make a new function to get this movement factor to use it in the new EventHandler. Like this

```
//update that character position
void EventHandler :: onEventInput(Event *e){

    //movement input event handler
    if (e->getType() == "EventPlayerMovement") {
        EventPlayerMovement* movementEvent = dynamic_cast<EventPlayerMovement*>(e);
        if (movementEvent) {
            sf::Keyboard::Key keyPressed = movementEvent->getChar();
            // Implement the logic for handling the player movement event
            if(keyPressed == sf::Keyboard::A ){
                float newXPos = playerChar->getPosition().x - playerChar->getXVel() * playerChar->getmovementFactor();
                playerChar->setPosition(newXPos, playerChar->getPosition().y);
            }
            if(keyPressed == sf::Keyboard::W ){
                float newYPos = playerChar->getPosition().y - playerChar->getYVel() * playerChar->getmovementFactor();
                playerChar->setPosition( playerChar->getPosition().x, newYPos);
            }
            if(keyPressed == sf::Keyboard::S ){
                float newYPos = playerChar->getPosition().y + playerChar->getYVel() * playerChar->getmovementFactor();
                playerChar->setPosition( playerChar->getPosition().x, newYPos);
            }
            if(keyPressed == sf::Keyboard::D ){
                float newXPos = playerChar->getPosition().x + playerChar->getXVel() * playerChar->getmovementFactor();
                playerChar->setPosition(newXPos, playerChar->getPosition().y);
            }
        }
    }
```

As for changes in my client code to deal with the new addition of events, I made it so that after an event of player movement is raised, I set a bool movedClient variable to true, then if this value is true, this new position will be send to the server.

Secondly, I moved on to the implementation of character collision event, I won't go over how my eventManager and handler interacts with events anymore as I have talked about them earlier. For this event, I made a new class called EventCollision, this event, similar to the last event, inherited from the event class, this time, setting it's type as "EventCollision ". Now, for this part, I have decided refractor my previous code for stationary platforms to include a "deadPlatform" bool field, to indicate if a platform is dangerous for a player to collide with, if it is true, the player will be reset to their spawn position (this doesn't count as a death), if the bool is false, the collision will not do anything (I still have not implemented this part as I will implement jumping in the future, this is meant for that.) So, in my eventHandler, if it detects an event as the EventCollision type,it will cast and then find if the platform being passed in the constructor is a deadPlatform, if so, it will send the player back to spawn point.

Thirdly, is the Dead Event, I decided to implement this as any area below the y value of 600, so in game, it is like if the player falls of the map, they die. Similar to before, I also had a deadevent, with the type "DeathMapLimit", the if the player's position exceeds 600 (in the y direction), they will be resetted to the spawn point, however this is different than the previous collide event in that the playableRec class now have a death counter, or rather "hearts", the each players have 4 hearts and everytime they fall off the map, they lose 1, once it reaches 0, they character can't no longer be moved.

Lastly, is the spawn event, I decided to refractor my code from the previous implementation. Previously, when a new client connects to the server, it will req rep a "Hello" message and then send over newly created character with that Clients'ID, then the server will check if this character is new, update it's own character list and send pub sub that character to the rest of the clients to update their own list of characters. For this Homework, I decided to

make a new event called EventNewSpawn, it takes in the position, size, id of a character as well as a pointer to a characters list, so, when the server send over a new character, the clients will take the information sent, raise the new spawn event type with the newly received information and let the handler making the new character, setting it’s id and pushing that character to the list of characters the provided pointer is pointing to.