Section 1: I have added a basic demonstration of script capabilities to my code. Currently I have game loop load scripts each loop to update the player object speed. This was a relatively simple addition that required a little bit of tweaking to the player object to allow the function that changes speed to be sent to the script. I was originally considering adding the function to change the color of the player into the script manager but gave up on that to save time.

I provided my game a basic demo of event raising and handling. Whenever the button "c" is pressed, the script will provide a boolean to the engine to generate an event. This event will eventually make its way to an event handler that will call upon the script to execute code. I kept things simple and only had it demonstrate printing a string. This event can be recorded if I modified my event manager slightly and add it to the "recordable" events. I can think of a lot of a lot of interesting things I could do with scripts for a game but due in part to a lack of experience earlier on in my project, reworking enough of my project to be able to flexibly utilize scripts would take too much time. The basic versions of scripting I've settled with didn't take too long, but heavier modifications could potentially make my program unstable (if I don't spend a long time diagnosing issues).

Section 2: I have completed a version of the space invaders game. There is currently only one level with two rows of enemies. These enemies have a travel pattern they follow as they move closer and closer to the planet. The player is locked into having movement left and right by the blue barriers on both sides of the screen (and the fact that player controls have been reduced to right and left only). Pressing the button "f" will generate a projectile from the player object that shoots upwards. An interesting unintended effect from my engine is that when projectiles are shot, they will also gain a velocity in the direction the player is moving. They will also bounce off the blue walls due to the collision components I added to the projectiles. Unfortunately, I got rather lazy towards this part of the project and the way I programmed these final parts are pretty inflexible and would require heavy modifications to create a different game.

Unfortunately due to the way I handle server-client communications, messages can be lost – which leads to a situation where objects I have put in queue to be deleted can be missed, leading to a misalignment of the indices of the objects in the list. This wouldn't have been an issue if I had provided a better unique identifier to each object. This way I wouldn't have to provide instructions on what to delete through the server (I could instead check to see if there was an update from the server for the object). Since I planned the space invader game to be single player in the first place – to save time I chose to reduce the role of the client to just demonstrating message sending. I had originally hoped to be able to replicate the display from the server to the client, but after a long while diagnosing and time being exhausted, I am resorting to this method to turn in the project on time.

The following is the difference between my two servers (platformer and space invader):

```
:: $ git diff --shortstat Game\ Project/Project\ 1/ Game\ Project\ \(Original\ Version\
-\ Part\ 1\ of\ Assignment\ 5\)/Game\ Project/Project\ 1/
 54 files changed, 147 insertions(+), 346 deletions(-)::
```

The additions weren't very significant – although they were pretty scrappy partially due to a rushed effort for this final project (more so due to previous design choices though). Frustratingly, when I was first creating the game objects for my engine, I made their constructors incredibly convoluted and frustrating to customize. At this point, I believe it would've been better for me to create another object that stores important engine information and feed the object to the game objects so that I don't have to tediously modify every game object's constructor each time I make a slight change. I believe around 1/3 of the deletions came from the previous game logic I had for the platformer.

Section 1:

Script code:

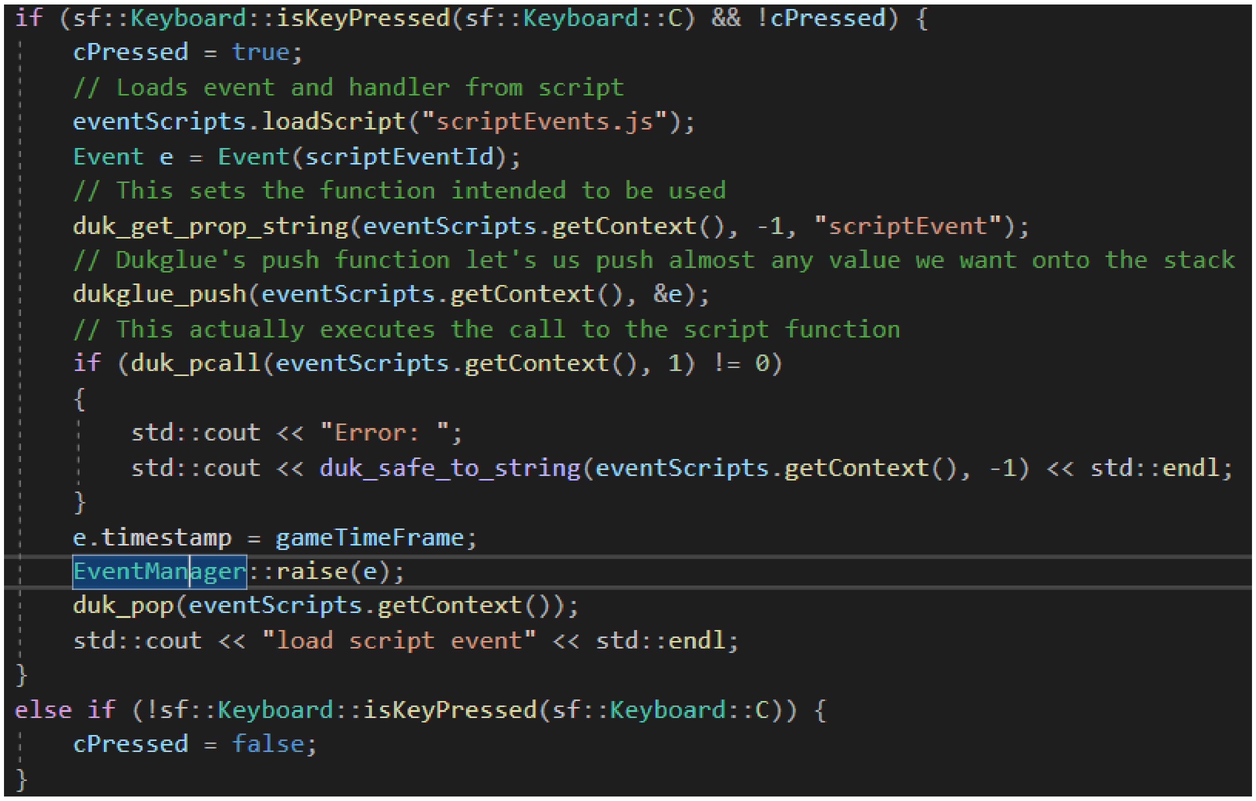
Adjusting speed:

```
function playerSpeedAdjust(player)
{
    player.adjustSpeed(1.5);

    return "Script Success 1";
}
```

Script event handling:

```
var eventId = internString("script_event");
var scriptBool = internString("script_bool");

function scriptEvent(event)
{
    event.addBool(scriptBool, true);

    return "Script Success 2";
}

function scriptEventId()
{
    return eventId;
}

function handleEvent(event)
{
    if (event.getBool(scriptBool) == true) {
        printString("The events being generated currently are true.");
    } else {
        printString("Script is now generating false events.");
    }
    return "Script Success 3";
}
```

Script event raising:


```cpp
if (sf::Keyboard::isKeyPressed(sf::Keyboard::C) && !cPressed) {
    cPressed = true;
    // Loads event and handler from script
    eventScripts.loadScript("scriptEvents.js");
    Event e = Event(scriptEventId);
    // This sets the function intended to be used
    duk_get_prop_string(eventScripts.getContext(), -1, "scriptEvent");
    // Dukglue's push function let's us push almost any value we want onto the stack
    dukglue_push(eventScripts.getContext(), &e);
    // This actually executes the call to the script function
    if (duk_pcall(eventScripts.getContext(), 1) != 0)
    {
        std::cout << "Error: ";
        std::cout << duk_safe_to_string(eventScripts.getContext(), -1) << std::endl;
    }
    e.timestamp = gameTimeFrame;
    EventManager::raise(e);
    duk_pop(eventScripts.getContext());
    std::cout << "load script event" << std::endl;
}
else if (!sf::Keyboard::isKeyPressed(sf::Keyboard::C)) {
    cPressed = false;
}
```