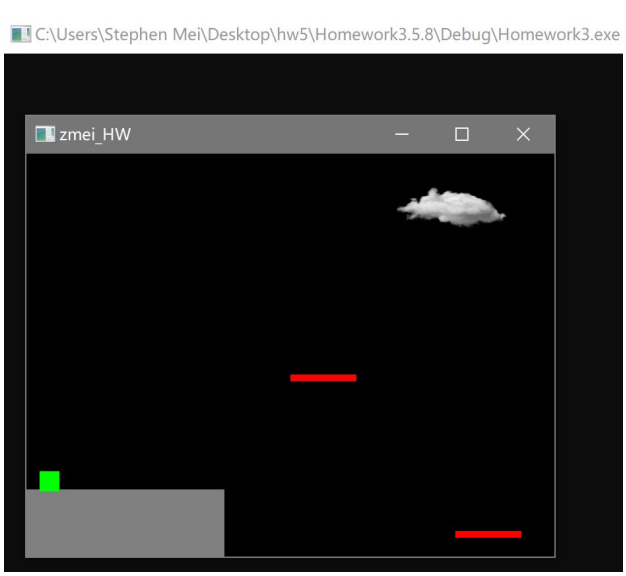
Zhongxiao Mei

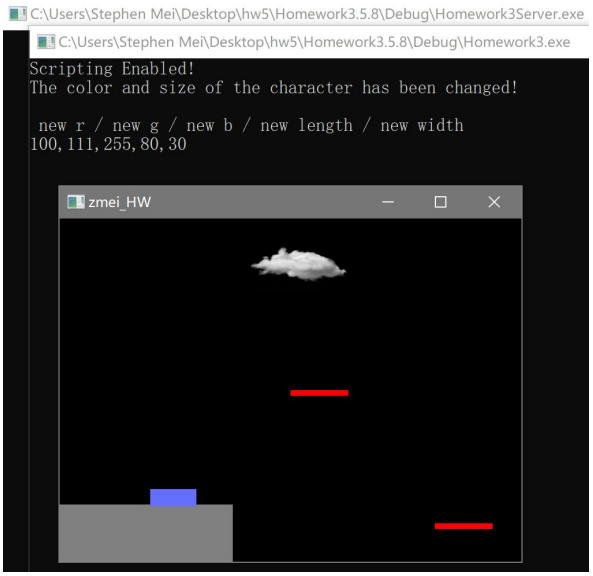
Part1: Scripting

I created a ScriptManager to enable scripting the behavior of game objects, a scriptEvent class to represent scripting events, and a scriptObject which works with the ScriptManager. The ScriptManager uses the Duktape and Dukglue library to load and run arbitrary scripts. The scriptObject class has getters and setters of color, size and position. The script manager can work with the scriptObject class to modify game objects' color, size, or position using scripts, and has the ability to handle events. Inside the ScriptManager, I have a helper function to load script from the JavaScrip file using duk_push_lstring() and a function called "runScripting()" to load, register and pass objects and member functions.

Inside the runScripting() function, I firstly use "duk_context* ctx = duk_create_heap_default();" to create and initialize a context, and then use "dukglue_register_constructor<scriptObject>(ctx, "scriptObject"); dukglue_register_method(ctx, &scriptObject::GetR, "get_r"); dukglue_register_method(ctx, &scriptObject::SetR, "set_r"); etc." to register the constructor and methods from the scriptObject class. After registering the script object, using the helper function "load_script_from_file(ctx, "scripting.js");" to load the script into the context. After loading script into the context, using "duk_push_global_object(ctx); duk_get_prop_string(ctx, -1, "enableScript");" to get the "enableScript" function in the "scripting.js" file. Finally, using "dukglue_push(ctx, &newScriptObject);" to pass objects declared in C++ to the script. In the "scripting.js" file, it calls setR(), setG() and setB() to set a game object's color (R,G,B) and calls setLength() and setWidth() to set a game object's size, etc.



before enable scripting

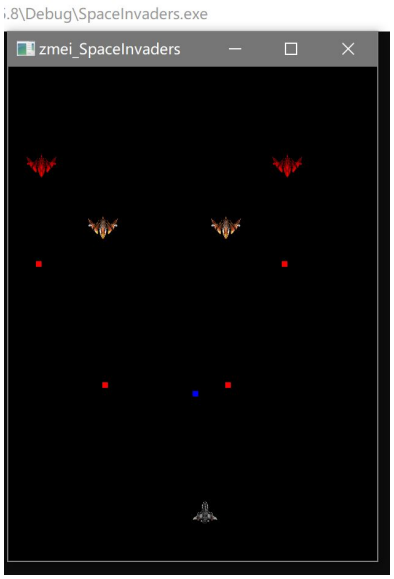


after enable scripting

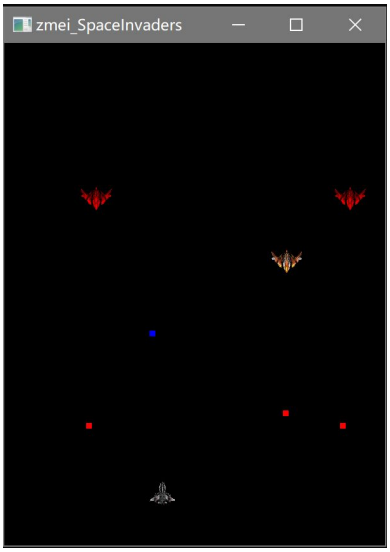
Part2: A Second Game

I decided to do a "Space Invaders" as my second game. I reused the game engine from my first 2D platform game, so the runtime game object model, client-server architecture and event management system are mostly the same. The runtime game object model is the Object-Centric Model. The GameObject is composed of different components, such as position, size, and color components. All

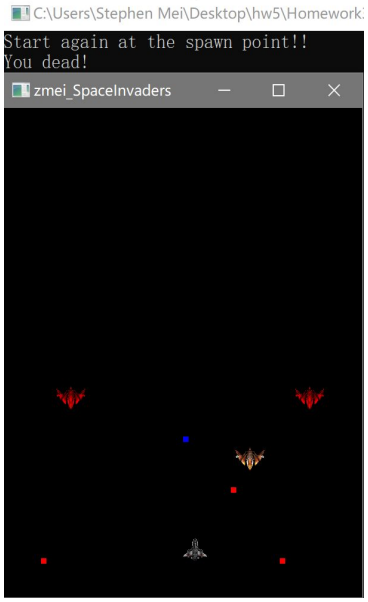
components (size, position, color) can be prepared at the beginning and can be used to create different game objects. The player, enemies, spawn points, dead zones and boundaries are instances of GameObject. Spawn points, dead zones, and boundaries are game objects that will not be drawn on the scene. The event management system also consists of Event (collisionEvent, deathEvent, spawnEvent, userInputEvent, scriptEvent), eventManager, and eventHandler. The eventManager keeps track of which event object is registered. After an event is raised, the “onEvent(Event*)” function in the eventHandler will be called to handle the particular event. The “onEvent(Event*)” function passes an Event as the parameter. It will check the event type and then decide which specific event is going to be handled based on the event type. The second game “Space Invaders” used the same client-server architecture as the first 2D platform game. The server should be opened and listen to clients, and then open the client to establish a connection with the server. The server handles user inputs and movement information sent from the client and re-sent to each client and draw on their scene.



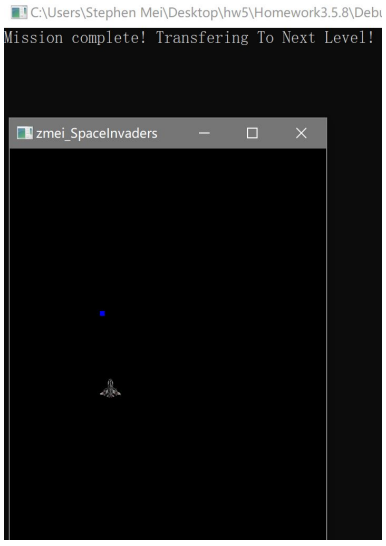
Start Space Invaders



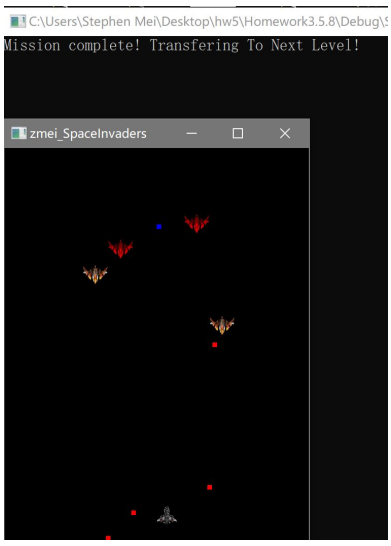
destroy one enemy



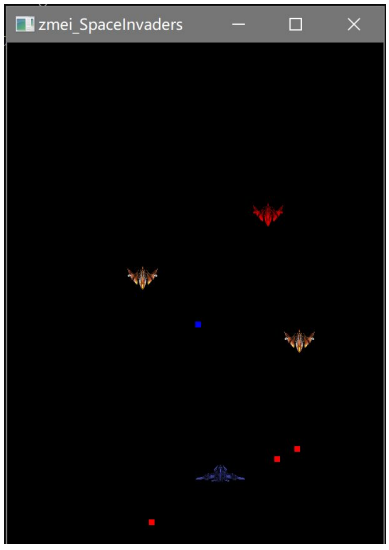
hitted by enemy, transfer to spawn point



destroy all enemy, transferring to next level



the next level



after enabled scripting