Matthew Warren

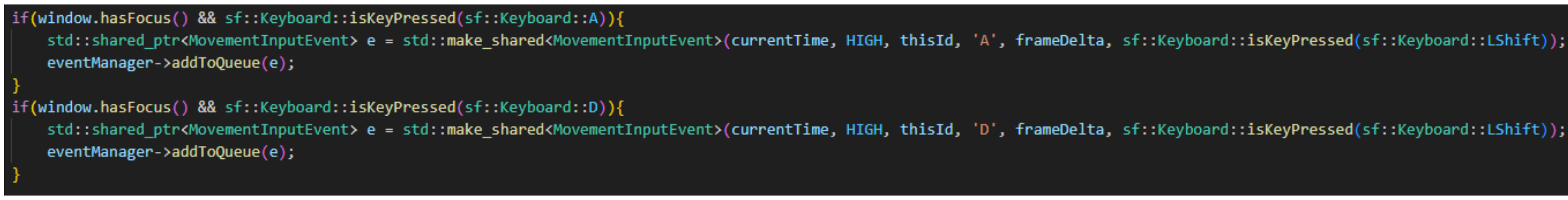
CSC 481 HW5 Part 1 Writeup

The first thing I implemented is the HID implementation. I decided to add a sprint button, where pressing shift and A/D would sprint in that direction. To do this, I updated my movement input event class to also take a new boolean parameter: if the shift is being held when this event is created [1]. This was a simple implementation that only checked if shift was being pressed if a or d were also being pressed, so just pressing shift did not do anything. Pressing a or d without shift also worked as normal, so it was only the combination of them that would change the behavior.

Scripting was a very new topic for me, so it took a lot of trial and error to get working correctly. To keep things simple, I only wanted my scripts to change my player character in some way. This meant with my current design, I could keep all scripting code in my client, not worrying about getting it also working on the server. A lot of the setup code was just copied from the example repository. I also kept the script manager and v8 helper classes to keep things simple.I read through and understood what they did, and did not feel I could make them any better by writing my own version. I wanted to keep as few contexts as possible to keep things simple, so I started with one being created before the client's loop starts. In my player, I wanted to expose different visual parts of the player. The first idea I had for a script was to constantly change the outline color of the player. I felt like this would be a good thing to add to modify game objects during regular update cycles, as it was something that could keep being changed without being too crazy. I wrote a getter and setter in my player class, that would take a value and only replace the red value of the color, with the others staying at 255. [2] The script here was simple, each time it was called it would increase the outline color by 1, bringing it back to 0 if it hits 255. [3] I wanted this to be run each time the game loop iterates, which is why the script changes it only by 1. Once I got the script done, I had to decide how to manage the scripts in the game. SInce the scripts are so small, I decided to compile them as the client starts and keep them in memory. I created a single player context, since all my scripts would just be modifying my player, exposing the clients player to this context. Then, all I had to do was run the script every frame. After getting this simple script done, I went on to make them work with events. My idea was to have a script that would prevent a player from moving back to the left. This would involve checking if the player's position is at the left boundary, and raising a death event if so. I wrote a function in my client that would create a death event and add it to my event manager, exposing it to v8 so it could be called in a script. I then had to expose the player's x position to the script as well. The script was simple, just checking the position and calling the function if the player was at the left boundary, and called every frame along with the first one. One thing I noticed is that there was a noticeable delay when hitting the bouldry before the death event happens. If I had to guess this is a product of scripting being slower than native code, as we discussed in class. But since the color changing scripts seem to work a lot faster, I'm thinking there could be other factors with my implementation that are affecting it as well. I found that this creates an interesting twist on my game, as a death event will completely reset my level. This script forces players to keep up with each other and makes it harder. Lastly, I needed a script to handle events. To accomplish this, I added a new event handler, one to manage scripting. This script handler would be registered to the events the script needs to handle, and run them in the onEvent function. I did have to pass the script manager into this handler, but this did not cause me any problems. All of these code snippets and examples were from my platformer game, but I followed a similar process in my Space Invaders game as well. My first script there changes the color of the player based on how many lives they have, which I thought was a cool idea that could easily be done in regular update cycles. For my events script, I wrote one that is registered to run whenever an enemy death event occurs. The script will check how many enemies are left, and give the player an extra life if they are halfway through.

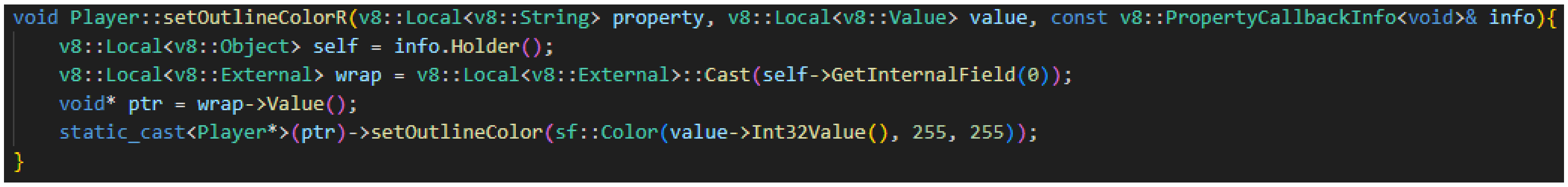
Appendix

[1]



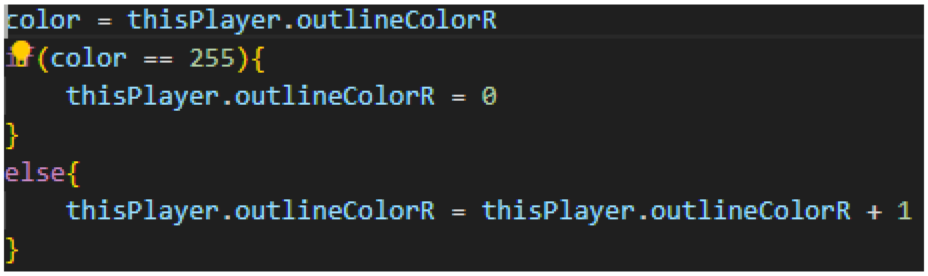
```
if(window.hasFocus() && sf::Keyboard::isKeyPressed(sf::Keyboard::A)){
    std::shared_ptr<MovementInputEvent> e = std::make_shared<MovementInputEvent>(currentTime, HIGH, thisId, 'A', frameDelta, sf::Keyboard::isKeyPressed(sf::Keyboard::LShift));
    eventManager->addToQueue(e);
}
if(window.hasFocus() && sf::Keyboard::isKeyPressed(sf::Keyboard::D)){
    std::shared_ptr<MovementInputEvent> e = std::make_shared<MovementInputEvent>(currentTime, HIGH, thisId, 'D', frameDelta, sf::Keyboard::isKeyPressed(sf::Keyboard::LShift));
    eventManager->addToQueue(e);
}
```

[2]



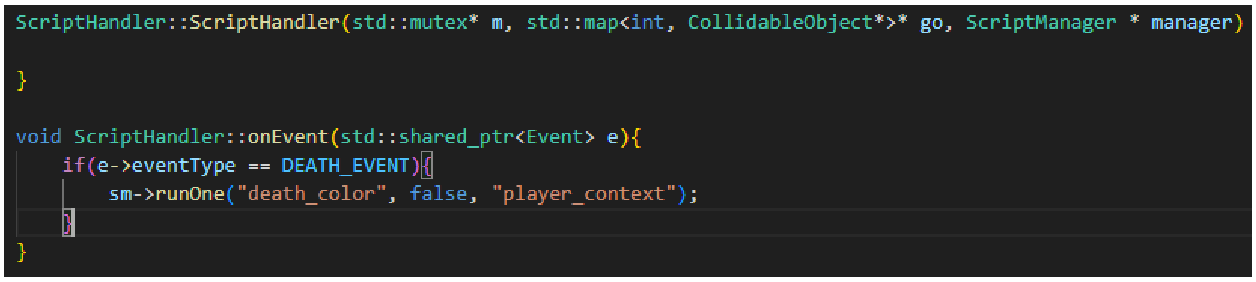
```
void Player::setOutlineColorR(v8::Local<v8::String> property, v8::Local<v8::Value> value, const v8::PropertyCallbackInfo<void>& info){
    v8::Local<v8::Object> self = info.Holder();
    v8::Local<v8::External> wrap = v8::Local<v8::External>::Cast(self->GetInternalField(0));
    void* ptr = wrap->Value();
    static_cast<Player*>(ptr)->setOutlineColor(sf::Color(value->Int32Value(), 255, 255));
}
```

[3]



```
color = thisPlayer.outlineColorR
if(color == 255){
    thisPlayer.outlineColorR = 0
}
else{
    thisPlayer.outlineColorR = thisPlayer.outlineColorR + 1
}
```

[4]



```
ScriptHandler::ScriptHandler(std::mutex* m, std::map<int, CollidableObject*>* go, ScriptManager * manager)

}

void ScriptHandler::onEvent(std::shared_ptr<Event> e){
    if(e->eventType == DEATH_EVENT){
        sm->runOne("death_color", false, "player_context");
    }
}
```