

Hyunsirk Choi

CSC 481

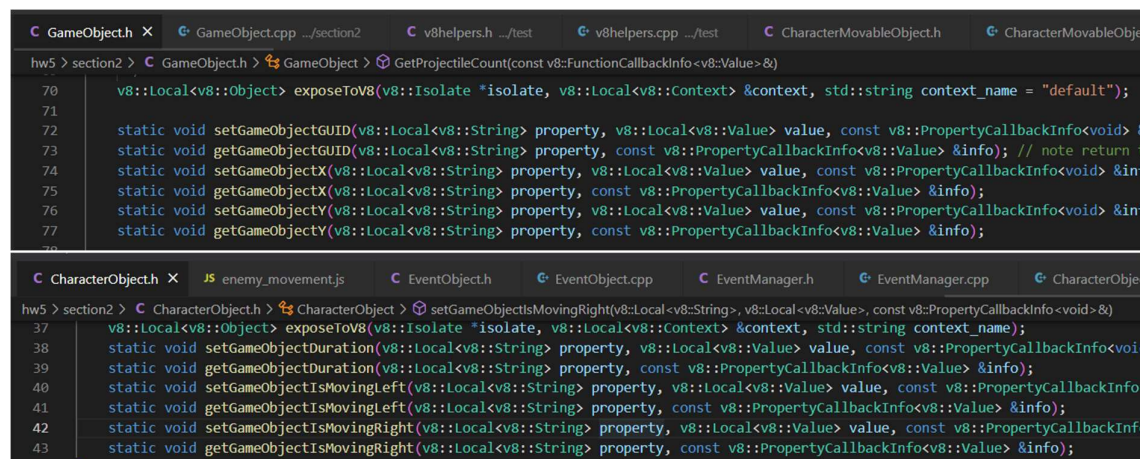
Dr. Robert

06 December 2022

## Homework 5 Write-Up

### Section 1: HIDS and Scripting

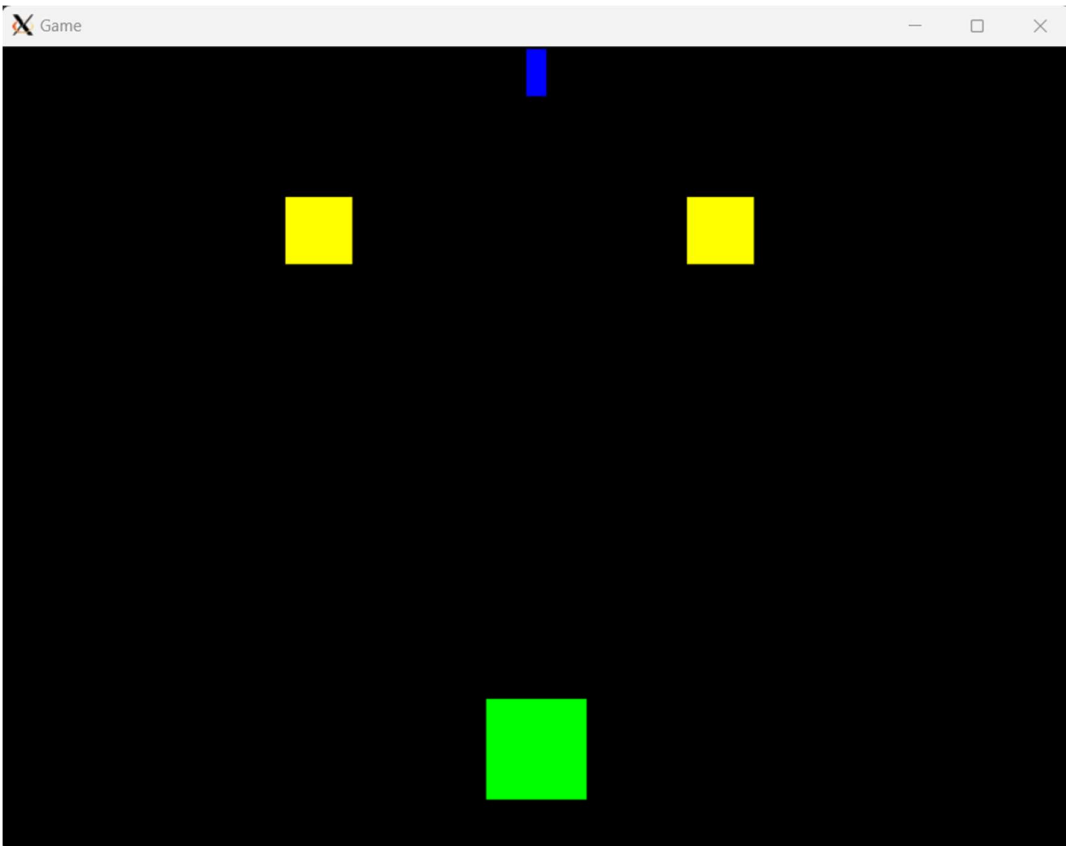
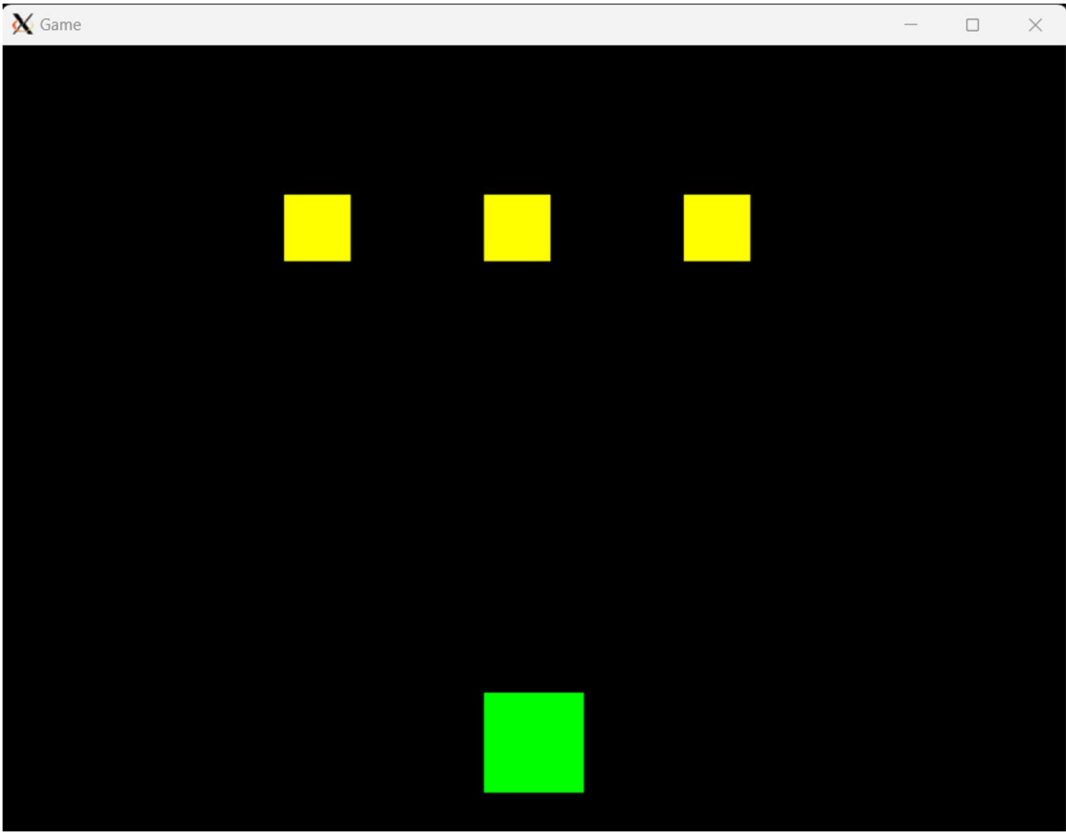
To allow scripting for the existing `GameObject` class in my engine, I had to implement `exposeToV8()` and getter and setters for the accessible fields. This `exposeToV8()` will be able to be overridden by any `GameObject` inheriting classes that may want to add additional fields. So, the `GameObject` inheriting `CharacterObject` class has an overridden `exposeToV8()` function to include an additional field to be accessible from the script. This way, the `character_movement.js` controls the movement of the object per input. As the C++ program detects input and registers it as a new input event, the `character_movement.js` script will run per detected input event during the input event polling.



```
C GameObject.h X  GameObject.cpp .../section2  v8helpers.h .../test  v8helpers.cpp .../test  CharacterMovableObject.h  CharacterMovableObject.cpp
hw5 > section2 > C GameObject.h > GameObject > GetProjectileCount(const v8::FunctionCallbackInfo<v8::Value>&)
70      v8::Local<v8::Object> exposeToV8(v8::Isolate *isolate, v8::Local<v8::Context> &context, std::string context_name = "default");
71
72      static void setGameObjectGUID(v8::Local<v8::String> property, v8::Local<v8::Value> value, const v8::PropertyCallbackInfo<void> &info);
73      static void getGameObjectGUID(v8::Local<v8::String> property, const v8::PropertyCallbackInfo<v8::Value> &info); // note return t
74      static void setGameObjectX(v8::Local<v8::String> property, v8::Local<v8::Value> value, const v8::PropertyCallbackInfo<void> &info);
75      static void getGameObjectX(v8::Local<v8::String> property, const v8::PropertyCallbackInfo<v8::Value> &info);
76      static void setGameObjectY(v8::Local<v8::String> property, v8::Local<v8::Value> value, const v8::PropertyCallbackInfo<void> &info);
77      static void getGameObjectY(v8::Local<v8::String> property, const v8::PropertyCallbackInfo<v8::Value> &info);
78
C CharacterObject.h X  JS enemy_movement.js  EventObject.h  EventObject.cpp  EventManager.h  EventManager.cpp  CharacterObject.cpp
hw5 > section2 > C CharacterObject.h > CharacterObject > setGameObjectsMovingRight(v8::Local<v8::String>, v8::Local<v8::Value>, const v8::PropertyCallbackInfo<void>&)
37      v8::Local<v8::Object> exposeToV8(v8::Isolate *isolate, v8::Local<v8::Context> &context, std::string context_name);
38      static void setGameObjectDuration(v8::Local<v8::String> property, v8::Local<v8::Value> value, const v8::PropertyCallbackInfo<void> &info);
39      static void getGameObjectDuration(v8::Local<v8::String> property, const v8::PropertyCallbackInfo<v8::Value> &info);
40      static void setGameObjectIsMovingLeft(v8::Local<v8::String> property, v8::Local<v8::Value> value, const v8::PropertyCallbackInfo<void> &info);
41      static void getGameObjectIsMovingLeft(v8::Local<v8::String> property, const v8::PropertyCallbackInfo<v8::Value> &info);
42      static void setGameObjectIsMovingRight(v8::Local<v8::String> property, v8::Local<v8::Value> value, const v8::PropertyCallbackInfo<void> &info);
43      static void getGameObjectIsMovingRight(v8::Local<v8::String> property, const v8::PropertyCallbackInfo<v8::Value> &info);
```

### Section 2: A Second Game

Due to a lack of time, this section of the project is incomplete to implement "Space Invaders". As 2 types of `CharacterObject` instances can be spawned as either character or enemy, the player will control the character and will be able to shoot `ProjectileObject` projectile instances. When the player presses ENTER, a projectile will spawn in front of the character, and the projectile will continue to move until it hits the end of the screen or a `CharacterObject` enemy. When the projectile hits an enemy, the projectile will kill and clear the hit enemy. As the projectile clears an enemy or clears itself by escaping the window, the game will completely freeze which disables the character input control.



In terms of design, it was fairly simple to implement as there are only two core classes to implement: `CharacterObject` and `ProjectileObject`. The `CharacterObject` is a representation class for both the player character and the enemy in the game. The `ProjectileObject` class has a field to remember its source shooter to disable friendly fires and create an enemy death event on contact. The plan was using 3 different scripts to enable controls on the player character, enemy, and projectile movement.