

# Writeup for Homework 5

200333470

yhuang64

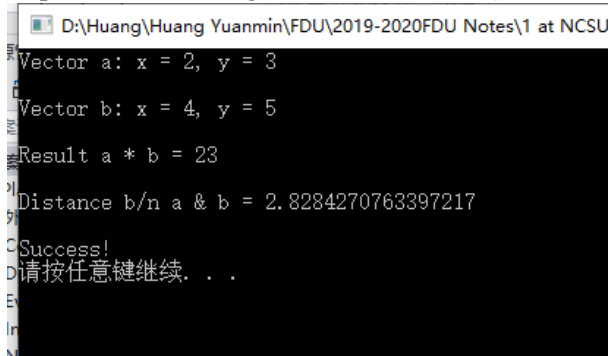
Yuanmin Huang

## Section 1

For section 1, I tried really hard to integrate the scripting module into my project but failed. In the following part of the writeup of this section, I will try to show you the furthest step I have been to and hope that I can at least get some credit for the things I have done.

Firstly, I referred to the demo published online by Noah Benveniste on moodle. For test use, I set up a new project “Scripting” to run the code below.

After setting up the environment for duktape and dukglue, I the accomplished effect of the test script as the following screenshot shows. (run demo2.cpp)

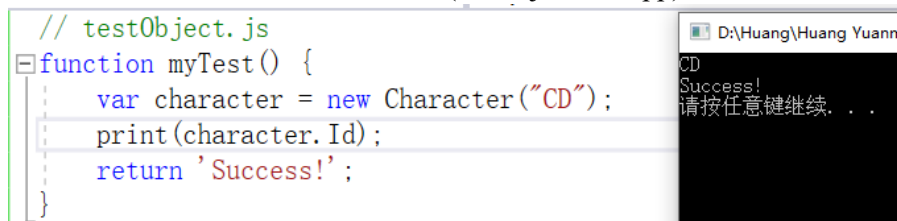


```
D:\Huang\Huang Yuanmin\FDU\2019-2020FDU Notes\1 at NCSU
Vector a: x = 2, y = 3
Vector b: x = 4, y = 5
Result a * b = 23
Distance b/n a & b = 2.8284270763397217
Success!
请按任意键继续. . .
```

Then, I think that to modify game objects, I should try how to gain access to my game object in scripts. Thus, I implemented the following test cpp, which registered GameObject and Character class to the context and specified there inheritance relationship.

```
dukglue_register_constructor<GameObject>(ctx, "GameObject");
dukglue_register_property(ctx, &GameObject::getId, nullptr, "Id");
dukglue_register_constructor<Character, string>(ctx, "Character");
dukglue_set_base_class<GameObject, Character>(ctx);
```

At the same time, in the script, I tried to new a character object and referred to its id. The screenshot below showed me the correct effect. (run objectTest.cpp)



```
// testObject.js
function myTest() {
    var character = new Character("CD");
    print(character.Id);
    return 'Success!';
}
```

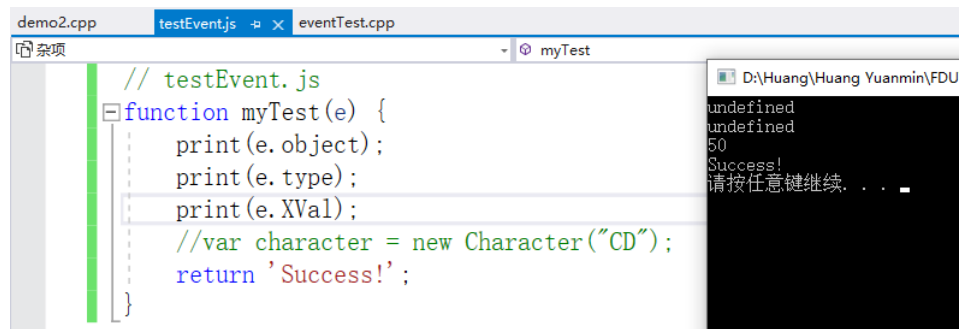
```
D:\Huang\Huang Yuanmin
CD
Success!
请按任意键继续. . .
```

Next, for the use of handling events, I wanted to try if dukglue can work with my event objects.

I registered the EObjMovement class to context and pushed an event object of this type to the stack. I want to try if dukglue can return the object pointer (object), double value (XVal) stored in the variant class and enum class value (event type) for me.

```
dukglue_register_constructor<EObjMovement>(ctx, "EObjMovement");
dukglue_register_property(ctx, &EObjMovement::getObject, nullptr, "Object");
dukglue_register_property(ctx, &EObjMovement::getXVal, nullptr, "XVal");
```

After running the script for this test, I achieved the following result. (run eventTest.cpp)



```
// testEvent.js
function myTest(e) {
    print(e.object);
    print(e.type);
    print(e.XVal);
    //var character = new Character("CD");
    return 'Success!';
}
```

```
undefined
undefined
50
Success!
请按任意键继续. . .
```

It turned out that I could get the game object pointer and the event type value using dukglue.

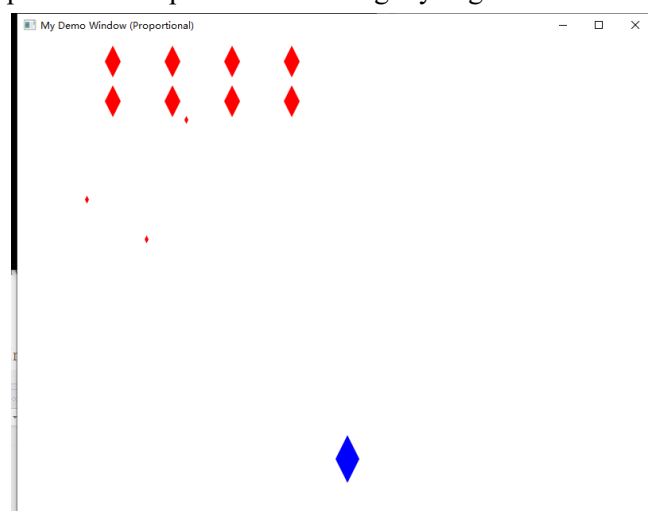
I think that this makes it nearly impossible for me to use this to handle the object movement event because the generic component embedded in my game object, which is the part I must have access to in order to handle events, can only be referred to using enum class value. Not to mention the inaccessible game object through the event object.

I think that the main reason for the failure may be the deep call stack when accessing the actual attribute which I need to modify. Therefore, not to waste time on this, I decided to implement the second part first.

To test the functionality I have tested above, you can build and run the Scripting project in the solution. What you need to do is to switch between the three mentioned test files (demo2.cpp, testObject.cpp, testEvent.cpp), which asks you to comment two of them and uncomment the other one before running the project.

## Section 2

For section 2, I implemented a space invader using my engine.



As is shown above, the 2 \* 4 red diamonds are invaders and the blue diamond is the

character (defender). The tiny red diamonds flying are the bullets fired by invaders.

The invaders will move from left to right and step down once and then move from right to left, which is the same as the original game. The characters can fire bullets back, but at most two at a time and at most one every one second.

The character will die if it is hit by a bullet, which is also the same for invaders. However, the bullets in my game will not disappear after hitting someone. They will only disappear after they fly out of the window. Also, due to the latency of networking, there maybe some weird death.

Once all the invaders are dead, the character will receive a message of wining in console and the game ends.

```
D:\Huang\Huang Yuanmin\FDU\2019-2020FDU Notes\1 at
Connecting to server on port 5555...
Subscribing to server on port 5556...
Connected to server
Congratulations! You win!
请按任意键继续. . .
```

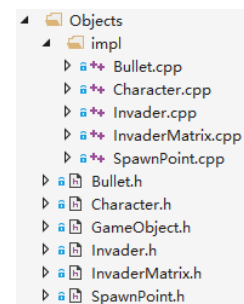
After the overview, I will show you how I designed and implemented the game. I will divide them into five main parts.

## 1. Objects

To me, besides the Character and SpawnPoint, Invader and Bullet are the other important objects in this game. I made them be renderable, movable and collidable.

Furthermore, I implemented a InvaderMatrix to manage my invaders. When constructing it, how many rows and columns of invaders are needed can be flexibly chosen. Then, a `vector<vector<Invader*>>` would be initialized accordingly.

```
InvaderMatrix(EventManager* manager, int row, int column,
              Vector2f topleft, Vector2f range, float moveRange,
              Vector2f velocity, Timeline& timeline);
```



Also, it takes care of the movement and firing logic of invaders. That is, the whole invader matrix should move together for some distance at a time. And, randomly selected invaders would fire a bullet towards the character.

At the same time, the character is restricted to move only horizontally, and can only have two bullets in the window at a time. The gun of the character is limited to fire one bullet every one second. This is implemented by maintaining a list of current bullets and recording the last time of firing.

## 2. Components

To define the movement of invaders and bullets, I implemented `cMove` and `bMove` functions in Movable component.

```
14 class Movable : public GenericComponent // in charge of moving related things
15 {
16     private:
17         Renderable* renderable;
18         Vector2f velocity;
19         Timeline& timeline;
20         ::Move move;
21
22         void cMove(double elapsed);
23
24         void bMove(double elapsed);
25
26         void iMove(double elapsed);
```

The cMove function moves the cluster of invaders towards the direction computed by the invader matrix, while the bMove function simply moves the bullet towards up or down.

In addition, the work function of Collidable implements the collision detection between invaders and character's bullets and between character and invaders and their bullets. In either cases, corresponding death events are raised.

### 3. Events

This leads us to the events design. The new event here is the EInvaDeath which represents the death of a invader. The others are basically the same as the 2D platformer game.

```

> ECharDeath.h
> ECharSpawn.h
> EInvaDeath.h
> EObjMovement.h
> EUserInput.h
> Event.h

```

### 4. Event Handlers

What has changed are the event handling functions so that they can provide functionalities needed by the space invader game.

For example, when an invader is killed, the handler calls the kill function of the invader matrix, which will move the invader pointer away from the matrix to an killed invader list. Later, the killed invaders will be deleted and cleared.

```

21 void EventHandler::onInvaDeath(EInvaDeath e)
22 {
23     Invader* invader = e.getInvader();
24     invader->getMatrix()->kill(invader);
25 }

180 void InvaderMatrix::kill(Invader* invader)
181 {
182     // locate invader
183     string id = invader->getId();
184     int row = atoi(id.substr(1, 1).c_str());
185     int column = atoi(id.substr(2, 1).c_str());
186
187     // remove from matrix
188     invaders[row].erase(find(invaders[row].begin(), invaders[row].end(), invader));
189
190     // put into killed
191     killed.push_back(invader);
192 }

```

### 5. Networking

My design for networking of this game is having all invaders and their bullets on server and character and its bullets on client.

The client would send the movement information of character and its bullets to the server so that the server can create corresponding objects. Then, those objects are used for collision detection after the update of invaders and their bullets. Finally, the movement of invaders, bullets and character (if killed, respawn will generate a character movement) is sent back to the client so that the client can interact accordingly. Specifically, when server finds out that invaders are killed, an additional message to indicate this would be sent to client. At the same time, expired bullets of both invaders and character would also cause additional messages to be sent.

This is implemented by utilizing the expired bullets list and killed invaders list.

```

148 // generate expired bullets message, BulletID = B + invaderID + roundnum
149 list<Bullet*>* expired = invaders->getExpiredBullets();
150 for (Bullet* bullet : *expired)
151 {
152     message += bullet->getId() + "\n";
153     //delete bullet;
154     this->expired.push_back(bullet);
155 }
156 expired->clear();
157
158 // generate killed invader message, InvaderID = I + row + column
159 list<Invader*>* killed = invaders->getKilled();
160 for (Invader* invader : *killed)
161 {
162     message += invader->getId() + "\n";
163     //delete invader;
164     this->expired.push_back(invader);
165 }
166 killed->clear();

```

Another thing needs to be mentioned in this part is the latency. Maybe due to my architecture, there are some situations in which the latency would affect the game experience. For instance, it seems on the client end that your bullet has hit an invader, but it hasn't on the server end.

Then, I will show you the difference in code between my new game and the old one.

In the screenshots below, the left side would be the code of the new game, and the right side would be the old one. Due to the limitation of the tool I used, I can only observe the number of blocks of different codes between comparable files (files with the same name) instead of line-wise counts of different codes between two directories. I will try my best to explain where the difference lies in my code.

\*Add-on: I used the diff tool after I finished the whole part of writeup and found the percentage of different lines is  $(706\text{-client-diff} + 584\text{-server-diff}) / (1782\text{-client} + 1609\text{-server}) = 38.04\%$

The screenshot shows a Windows command prompt window with the following content:

```

C:\WINDOWS\system32\cmd.exe

D:\Huang\Huang Yuanmin\FDU\2019-2020FDU Notes\1 at NCSU\Game Engine Foundation\Code\src_code>cloc -l 1.84.exe ./2DPlatformerClient
49 text files.
49 unique files.
12 files ignored.

github.com/AlDanial/cloc v 1.84 T=0.50 s (98.0 files/s, 4512.0 lines/s)
-----
Language           files  blank  comment  code
-----
C++                 23      161      97      990
C/C++ Header       26      208       8      792
SUM:                49      369     105     1782
-----

D:\Huang\Huang Yuanmin\FDU\2019-2020FDU Notes\1 at NCSU\Game Engine Foundation\Code\src_code>cloc -l 1.84.exe ./2DPlatformerServer
43 text files.
43 unique files.
12 files ignored.

github.com/AlDanial/cloc v 1.84 T=1.00 s (43.0 files/s, 2046.0 lines/s)
-----
Language           files  blank  comment  code
-----
C++                 20      149      97      923
C/C++ Header       23      185       6      686
SUM:                43      334     103     1609
-----

D:\Huang\Huang Yuanmin\FDU\2019-2020FDU Notes\1 at NCSU\Game Engine Foundation\Code\src_code>diff -yr --suppress-common-lines ./2DPlatformerClient ./S
paceInvadersClient | wc -l
706

D:\Huang\Huang Yuanmin\FDU\2019-2020FDU Notes\1 at NCSU\Game Engine Foundation\Code\src_code>diff -yr --suppress-common-lines ./2DPlatformerServer ./S
paceInvadersServer | wc -l
584

D:\Huang\Huang Yuanmin\FDU\2019-2020FDU Notes\1 at NCSU\Game Engine Foundation\Code\src_code>

```

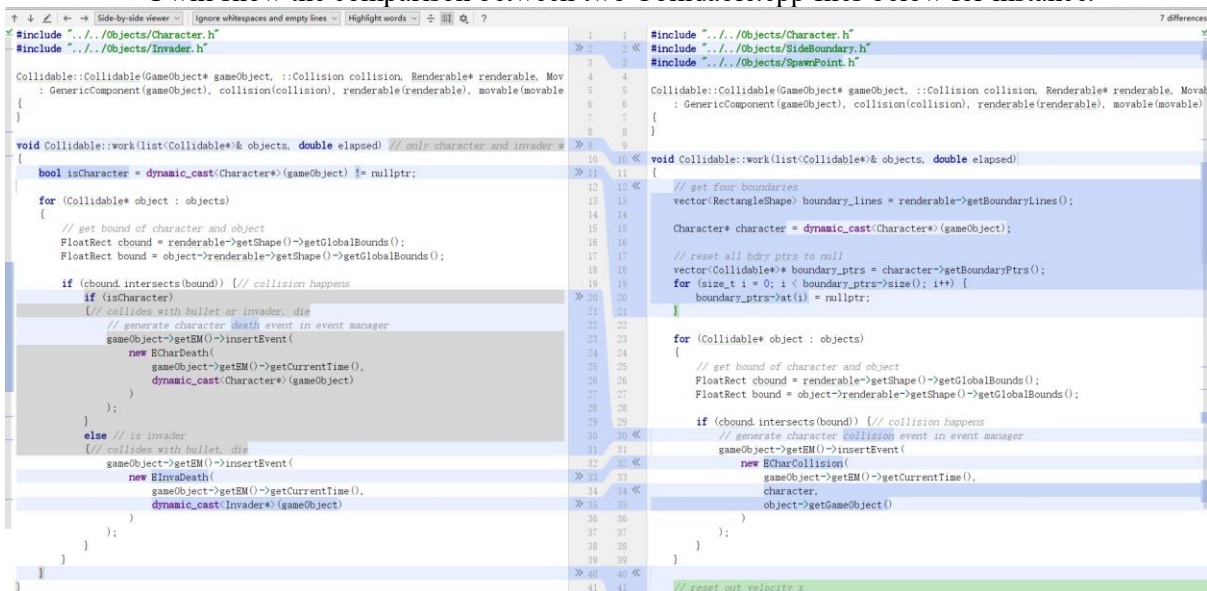
## 1. Components, total difference: 28

Name	Size	Date	*	Date	Size	Name
 Collidable.h	699	2019/12/2 2:...	≠	2019/11/19 ...	717	 Collidable.h
 GenericComponent.h	546	2019/11/19 ...	=	2019/11/19 ...	546	 GenericComponent.h
 Movable.h	801	2019/12/2 1:...	≠	2019/11/19 ...	813	 Movable.h
 Renderable.h	654	2019/11/19 ...	=	2019/11/19 ...	654	 Renderable.h
/impl/						
 Collidable.cpp	1297	2019/12/4 1:...	≠	2019/11/19 ...	1356	 Collidable.cpp
 Movable.cpp	2335	2019/12/3 1:...	≠	2019/11/19 ...	3712	 Movable.cpp
 Renderable.cpp	1679	2019/11/19 ...	=	2019/11/19 ...	1679	 Renderable.cpp

As you can see, the original structure of component model hasn't been changed. There is no new component, and the GenericComponent abstract class and the Renderable component remain the same.

What's new here mainly occurs in Movable and Collidable, which is because of the change of behaviors that the game objects would have in the new game. Therefore, I mainly modified the move and collision detection function implementations and kept the original structure.

I will show the comparison between two Collidable.cpp files below for instance.



```

1 1 #include ".../Objects/Character.h"
2 2 #include ".../Objects/SideBoundary.h"
3 3 #include ".../Objects/SpawnPoint.h"
4 4
5 5 Collidable::Collidable(GameObject* gameObject, Collision collision, Renderable* renderable, Movable* movable) : GenericComponent(gameObject), collision(collision), renderable(renderable), movable(movable) {}
6 6
7 7
8 8
9 9
10 10 void Collidable::work(list(Collidable*)& objects, double elapsed) // only character and invader
11 11 {
12 12     bool isCharacter = dynamic_cast<Character*>(gameObject) != nullptr;
13 13     for (Collidable* object : objects)
14 14     {
15 15         // get bound of character and object
16 16         FloatRect cbound = renderable->getShape()->getGlobalBounds();
17 17         FloatRect bound = object->renderable->getShape()->getGlobalBounds();
18 18
19 19         if (cbound.intersects(bound)) // collision happens
20 20         {
21 21             // collides with bullet or invader, die
22 22             // generate character death event in event manager
23 23             gameObject->getEM()->insertEvent(
24 24                 new BCharDeath(
25 25                     gameObject->getEM()->getCurrentTime(),
26 26                     dynamic_cast<Character*>(gameObject)
27 27                 ));
28 28         }
29 29         else // is invader
30 30         {
31 31             // collides with bullet, die
32 32             gameObject->getEM()->insertEvent(
33 33                 new BInvaderDeath(
34 34                     gameObject->getEM()->getCurrentTime(),
35 35                     dynamic_cast<Invader*>(gameObject)
36 36                 ));
37 37         }
38 38     }
39 39 }
40 40
41 41 // reset out velocity x

```

## 2. Objects, total difference: 40

Name	Size	Date	*	Date	Size	Name
Bullet.h	276	2019/12/2 8:...	→			
Character.h	1618	2019/12/3 2:...	≠	2019/12/1 2:...	1468	Character.h
			←	2019/11/19 ...	228	DeathZone.h
GameObject.h	793	2019/11/19 ...	=	2019/11/19 ...	793	GameObject.h
Invader.h	397	2019/12/4 3:...	→			
InvaderMatrix.h	2093	2019/12/4 3:...	→			
			←	2019/11/19 ...	855	MovingPlatform.h
			←	2019/11/19 ...	839	SideBoundary.h
SpawnPoint.h	161	2019/11/19 ...	=	2019/11/19 ...	161	SpawnPoint.h
/impl/						
Bullet.cpp	750	2019/12/2 2:...	→			
Character.cpp	3264	2019/12/3 1:...	≠	2019/11/19 ...	4092	Character.cpp
			←	2019/11/19 ...	473	DeathZone.cpp
Invader.cpp	1313	2019/12/4 3:...	→			
InvaderMatrix.cpp	4961	2019/12/4 3:...	→			
			←	2019/11/19 ...	891	MovingPlatform.cpp
			←	2019/11/19 ...	1945	SideBoundary.cpp
SpawnPoint.cpp	276	2019/11/19 ...	=	2019/11/19 ...	276	SpawnPoint.cpp

The change in this part is severe, though the game object structure hasn't been changed. As you can see, except GameObject, only SpawnPoint remains unchanged. Character has been changed a lot. The rest of them are totally different files (thus are not counted into the difference).

#include "../Invader.h"	1	#include "../MovingPlatform.h"	1
#include "../InvaderMatrix.h"	2		2
Invader::Invader(string id, EventManager* manager,	3	MovingPlatform::MovingPlatform(string id, EventManager* manager,	3
Vector2f pos, Vector2f velocity, Timeline& timeline, InvaderMatrix* matrix)	4	::Shape shape, ::Color color, Vector2f size,	4
GameObject(id, manager), matrix(matrix) // alive(true)	5	Vector2f pos, Vector2f velocity, Timeline& timeline, ::Move move, float negBound, float range)	5
{	6	GameObject(id, manager), headingPositive(false), negBound(negBound), posBound(negBound + range)	6
this->addGC(	7	{	7
ComponentType::RENDERABLE,	8	this->addGC(	8
new Renderable(this, ::Shape::DIAMOND, ::Color::RED, Vector2f(20.f, 40.f), pos)	9	ComponentType::RENDERABLE,	9
);	10	new Renderable(this, shape, color, size, pos)	10
this->addGC(	11	);	11
ComponentType::MOVABLE,	12	this->addGC(	12
new Movable(	13	ComponentType::MOVABLE,	13
this,	14	new Movable(	14
dynamic_cast<Renderable*>(this->getGC(ComponentType::RENDERABLE)),	15	this,	15
velocity, timeline, ::Move::CLUSTER	16	dynamic_cast<Renderable*>(this->getGC(ComponentType::RENDERABLE)),	16
);	17	velocity, timeline, move, negBound, range	17
);	18	);	18
this->addGC(	19	this->addGC(	19
ComponentType::COLLIDABLE,	20	ComponentType::COLLIDABLE,	20
new Collidable(	21	new Collidable(	21
this,	22	this,	22
Collision::INVADER,	23	Collision::PLATFORM	23
dynamic_cast<Renderable*>(this->getGC(ComponentType::RENDERABLE)),	24	dynamic_cast<Renderable*>(this->getGC(ComponentType::RENDERABLE)),	24
dynamic_cast<Movable*>(this->getGC(ComponentType::MOVABLE))	25	dynamic_cast<Movable*>(this->getGC(ComponentType::MOVABLE))	25
);	26	);	26
}	27	}	27
	28		28
	29		29

However, this is because these two games are totally different games which need different functionalities. It's fair that they have different set of game objects and design of them.

The main purpose for the game engine is that the mode we design our objects should be reusable, which is preserved in my design. As you can see above, different objects in both games still have similar ways of construction. (I'm showing the difference between Invader and MovingPlatform).

### 3. Times, total difference: 0

Name	Size	Date	*	Date	Size	Name
GameTime.h	331	2019/11/19 ...	=	2019/11/19 ...	331	GameTime.h
LocalTime.h	239	2019/11/19 ...	=	2019/11/19 ...	239	LocalTime.h
Timeline.h	683	2019/11/19 ...	=	2019/11/19 ...	683	Timeline.h
/impl/						
GameTime.cpp	1704	2019/11/19 ...	=	2019/11/19 ...	1704	GameTime.cpp
LocalTime.cpp	558	2019/11/19 ...	=	2019/11/19 ...	558	LocalTime.cpp

The time module between two games are completely identical. The time part is portable enough for my engine to utilize in different games.



#### 4. Event Management, total difference: 40

Name	Size	Date	*	Date	Size	Name
 ECharDeath.h	307	2019/11/19 ...	←	2019/11/19 ...	465	 ECharCollision.h
 ECharSpawn.h	451	2019/11/19 ...	=	2019/11/19 ...	307	 ECharDeath.h
			=	2019/11/19 ...	451	 ECharSpawn.h
			←	2019/11/19 ...	286	 EEndPlaying.h
			←	2019/11/19 ...	278	 EEndREC.h
 EInvaDeath.h	295	2019/12/2 2...	→			
 EObjMovement.h	645	2019/12/2 1...	≠	2019/11/19 ...	901	 EObjMovement.h
			←	2019/11/19 ...	282	 EStartREC.h
 EUserInput.h	449	2019/11/19 ...	=	2019/11/19 ...	449	 EUserInput.h
 Event.h	901	2019/12/2 2...	≠	2019/11/19 ...	957	 Event.h
 EventHandler.h	620	2019/12/2 2...	≠	2019/12/1 2...	970	 EventHandler.h
 EventManager.h	2892	2019/12/4 2...	≠	2019/11/19 ...	3381	 EventManager.h
/impl/						
			←	2019/11/19 ...	323	 ECharCollision.cpp
 ECharDeath.cpp	214	2019/11/19 ...	=	2019/11/19 ...	214	 ECharDeath.cpp
 ECharSpawn.cpp	301	2019/11/19 ...	=	2019/11/19 ...	301	 ECharSpawn.cpp
			←	2019/11/19 ...	204	 EEndPlaying.cpp
			←	2019/11/19 ...	191	 EEndREC.cpp
 EInvaDeath.cpp	209	2019/12/2 2...	→			
 EObjMovement.cpp	733	2019/12/2 1...	≠	2019/11/19 ...	889	 EObjMovement.cpp
			←	2019/11/19 ...	199	 EStartREC.cpp
 EUserInput.cpp	292	2019/11/19 ...	=	2019/11/19 ...	292	 EUserInput.cpp
 Event.cpp	118	2019/11/19 ...	=	2019/11/19 ...	118	 Event.cpp
 EventHandler.cpp	2854	2019/12/2 2...	≠	2019/12/2 0:...	6302	 EventHandler.cpp
 EventManager.cpp	1614	2019/12/2 2...	≠	2019/11/19 ...	2236	 EventManager.cpp
/util/						
 Variant.cpp	521	2019/12/2 2...	≠	2019/11/19 ...	555	 Variant.cpp
 Variant.h	778	2019/11/19 ...	=	2019/11/19 ...	778	 Variant.h

Like objects, the event types and handling have been changed a lot during the shifting between two games. In addition, I removed the replay system and its related files, which may also cause some difference.

Whereas, the main mechanism of event representation, raising, queuing and handling remain unchanged. What has been changed are the event types enumerations, new events and there handling functions.

The Event class is shown below to illustrate that only the enumerations are different between the new one and the old one.



```

#pragma once
#include <map>
#include "util/Variant.h"
using namespace std;

enum class Event_t {
    CHAR_DEATH, INVADER_DEATH, CHAR_SPAWN, USER_INPUT, OBJ_MOVEMENT
};

enum class Content_t {
    X_VAL, Y_VAL, CHARACTER_PTR, INVADER_PTR, OBJ_PTR, USER_INPUT, POSITIVE
};

class Event
{
protected:
    Event_t type;
    double executeTime;
    map<Content_t, Variant> args;

    void addArg(Content_t content_t, Variant_t variant_t, void* arg)
    {
        args.insert({ content_t, Variant(variant_t, arg) });
    }

    Variant getArg(Content_t content_t)
    {
        return args.find(content_t)->second;
    }
public:
    Event(Event_t type, double executeTime);
};
  
```

```

#pragma once
#include <map>
#include "util/Variant.h"
using namespace std;

enum class Event_t {
    CHAR_COLLISION, CHAR_DEATH, CHAR_SPAWN, USER_INPUT, OBJ_MOVEMENT,
    START_REC, END_REC, END_PLAY
};

enum class Content_t {
    X_VAL, Y_VAL, CHARACTER_PTR, OBJ_PTR, USER_INPUT, POSITIVE,
    REPLAY_PTR, OFFSET_X, OFFSET_Y
};

class Event
{
protected:
    Event_t type;
    double executeTime;
    map<Content_t, Variant> args;

    void addArg(Content_t content_t, Variant_t variant_t, void* arg)
    {
        args.insert({ content_t, Variant(variant_t, arg) });
    }

    Variant getArg(Content_t content_t)
    {
        return args.find(content_t)->second;
    }
};
  
```

The two event manager implementations are also identical except for the removed replay related design.

```

#include "../EventManager.h"

EventManager::EventManager(Timeline* gameTime, mutex* mtxObjMov)
: gameTime(gameTime), handler(gameTime, this), GVT(gameTime->getTime()),
  mtxObjMov(mtxObjMov), connected(true)
{
    addQueue(SELF_NAME);
}

EventManager::~EventManager()
{
    // delete all stored event pointers
    for (auto& pair : queues)
    {
        auto& queue = pair.second;
        while (!queue.empty())
        {
            delete queue.top();
            queue.pop();
        }
    }

    void EventManager::executeEvents()
    {
        updateGVT();

        for (auto& pair : queues)
        {
            // go through each queue
            auto& queue = pair.second;

            // handle events on top of queue if execution time <= GVT
            while (!queue.empty() && queue.top()->getExecuteTime() <= GVT)
            {
                const ::Event* e = queue.top();
            }
        }
    }
  
```

```

#include "../EventManager.h"

EventManager::EventManager(Timeline* gameTime, mutex* mtxObjMov, Replay* replay)
: gameTime(gameTime), handler(gameTime, this), GVT(gameTime->getTime()),
  mtxObjMov(mtxObjMov), connected(true), replaying(false), replay(replay)
{
    addQueue(SELF_NAME);
}

EventManager::~EventManager()
{
    // delete all stored event pointers
    for (auto& pair : queues)
    {
        auto& queue = pair.second;
        while (!queue.empty())
        {
            delete queue.top();
            queue.pop();
        }
    }

    void EventManager::executeEvents()
    {
        updateGVT();

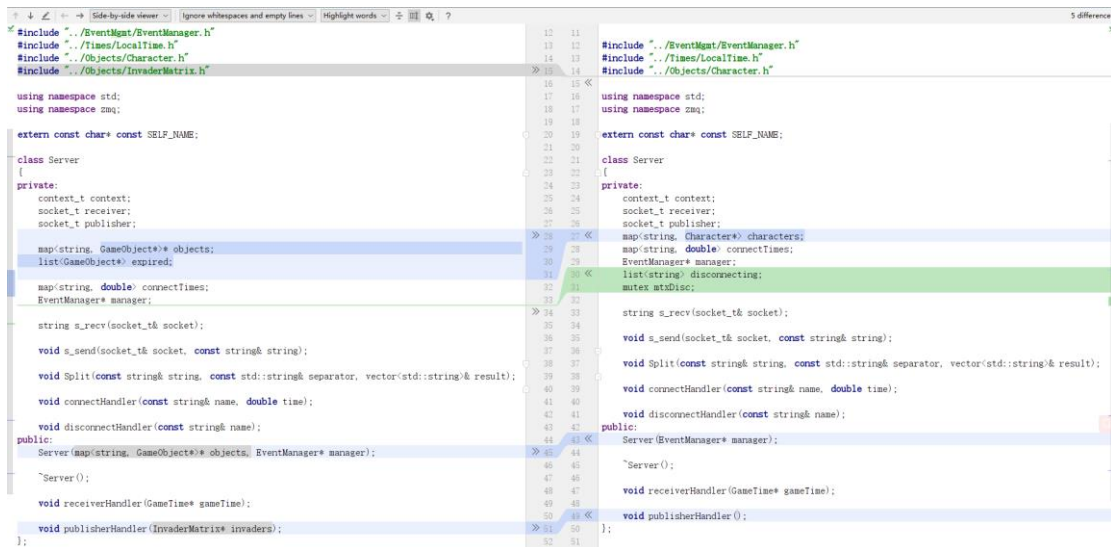
        for (auto& pair : queues)
        {
            // go through each queue
            auto& queue = pair.second;

            bool isEndRec = false;
            bool isEndPlay = false;
            // handle events on top of queue if execution time <= GVT
            while (!queue.empty() && queue.top()->getExecuteTime() <= GVT)
            {
                const ::Event* e = queue.top();
            }
        }
    }
  
```

## 5. Networking, total difference: 30 (client) + 25 (server)

Name	Size	Date	*	Date	Size	Name
Client.h	1042	2019/12/3 2...	≠	2019/11/19 ...	938	Client.h
/impl/						
Client.cpp	5802	2019/12/4 2...	≠	2019/11/19 1...	5564	Client.cpp
Server.h	1101	2019/12/4 0...	≠	2019/11/19 ...	1019	Server.h
/impl/						
Server.cpp	6933	2019/12/4 3...	≠	2019/11/19 ...	5770	Server.cpp

To be honest, the modification of this part is the most painful part among all. The connection code that can remain unchanged are really few. Because of the difference between two games, the content of message that are teleported between client and server must be redesigned and thus the way of processing and interpreting messages. The only good news is that the design for most interfaces remain unchanged or only has been changed a little, while the implementations beneath have huge differences. The picture below is the difference between two Server header files. As you can see, most function signatures are similar to each other.



## 6. Main, total difference: 34 (client) + 12 (server)

Again, different games, different main loops. That's fair.

We are surly going to have different object initializations and game-related implementation in the main loop.

Despite those differences, the order of everything happens in the main loop should remain the same. For example, we shall answer key input, move objects, do collision detections and render the objects on the window. Beyond that, not much code is reusable.

## 7. Conclusion

All in all, the most design part of the game engine of the two games are identical to each other. But some additions and deletions in code still have to occur because of the difference between the games we want to implement.

I believe that the design of my engine isn't too bad, so that I can utilize the generic design pattern I defined to create new classes to provide new functionalities, like components, game objects and events. In my opinion, this is the key point of reusing a game engine.

Also, I think that there will be better practice of the design of the engine such that we can achieve new functionalities without too much painful work in modifying the code. Instead, we may only need to modify some configuration files, which will be the reward of a more flexible and better-designed game engine.