Section 1: I have refactored a chunk of my code to follow to be managed through events – collision, character input, gravity, and animation in particular. My events are stored as hashed strings within a class named Event to promote efficient comparison between event types. Due to some oversight in how to implement some of my code, I have somewhat a lengthy recompilation time whenever I want to introduce a new event type, since it modifies the header file of the EventManager class (where all the hashed event string types are generated) – although I'm certain with a little bit of time, I can refactor it so that this isn't the case. Another thing I had to do for my events was add one variable to my AbstractGameObjects to keep track of the time the game's frame was on. Honestly, I could've simplified my code a good amount by passing the game timeline to my objects, but I didn't have it in me at the time to do so…

I have created a representation of variants for my events to accept as arguments when raising events. These variants (arguments) can be game object pointers, component pointers, integers, floats, long doubles, two floats stored as a vector, or boolean. Variants for every event are stored in a unordered map to maintain efficiency. These variants allow easy addition of arguments to events – although same as my way of handling event string types, it can take some time to compile the creation of new arguments. Anything that is defined can be used in a quick manner however.

Each event keeps track of a timestamp assigned by the starting location and priority assigned to it by the EventManager. This is to maintain order within the priority queue when running. The order of importance currently maintained within the EventManager starts from: gravity, player object input, death/respawn/translate, collision, server update, animate, replay record/terminate, and then any unrecognized event.

I have a system to register interest between EventHandlers and the triggering of events. An unordered map of hashed strings paired with vectors of abstract event handlers allow for multiple event handlers to be notified of the occurrence of an event with ease – reusable and easily expanded upon. All of my handlers only have one function: onEvent(event). This is called by the EventManager when an Event registered with the

EventHandler is called. In order to execute events at a proper time, a provided the EventManager a function to store the game timeline that the game is running on.

Within the EventManager for raising and dispatching events, I have introduced a mutex to prevent heap corruption when raising new events before the old one can be deleted.

My server update event is generated on the server or client whenever they receive an update message. The server update event operates pretty much exactly the same as a animation event, just under a different name so that I can keep track of where the events are coming from (I’m essentially sending animation events through the network to be raised an handled on other clients/server).

In order to handle calculations properly, I have made it so that my Animation component runs all the previous events prior to running. If this wasn’t done, it’ll introduce a scenario in which old values are being used to calculate object movements and will make the player appear very “bouncy”. Interestingly enough migrating my system to handle events has fixed some previous bugs (when jumping, the player used to get “interrupted” when pausing – thus when unpausing they’ll start falling no matter if they just jumped) and introduced new ones (likely fixable through adjustments in collision calculations – currently when colliding with objects on the side towards the top or bottom, there is a high chance of getting launched upward or downwards depending on which side the object is on). (Another bug to tag on – if I hit the translation zone to the right but fail to trigger the spawn set in the center of the screen, and then fall to the character’s death – it’ll leave the screen there without returning to the previous).

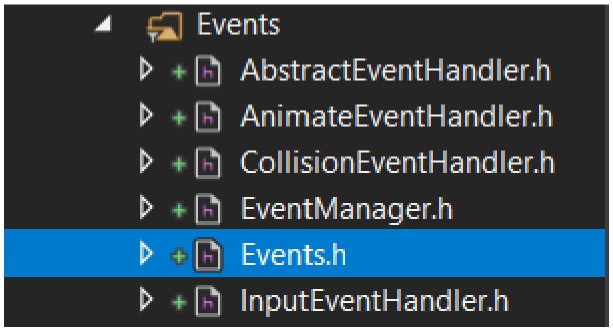
Section 2: For my replays, although I didn’t have to add too much code, the thoughts I had to put behind it were significant. The only thing I couldn’t fully implement was the change in speed while replaying a recording. The system I have in place on the client currently can technically playback a slow recording, albeit after a long pause. I have a good understanding of what is causing this bug but have not gotten enough time to figure out how to resolve this issue (because the way I determine when to run a event from a recording is based on the timestamp (kept in terms of real time), when I try and change the speed (thus tic size) the timing of the events are messed up). In order to demonstrate a proper recording – I have a slightly set up for the server recording mechanism (it can record both speed up and slow down of events, but is unable to change speeds at all during replay). (NOTE – the implementation difference for replay between server and client is only one line of code calculating time)

In order to prevent my priority queue from getting too clogged up, I have two priority queues storing events. When the event to start recording is triggered, I save a duplicate event of animation, server update, death, spawn set, and translation to my secondary priority queue that is there explicitly for replays. Once my recording has been terminated, the replay will immediately start playing, starting from when the recording began.

The recording and replaying can be initiated on both the server and the client – if replay is initiated by the server, all other clients are paused until the replay is over. If the replay is initiated by the client, they will remain in place on other clients until their replay is over. In order to maintain the order of events properly a time for the beginning of the replay is kept for calculations.

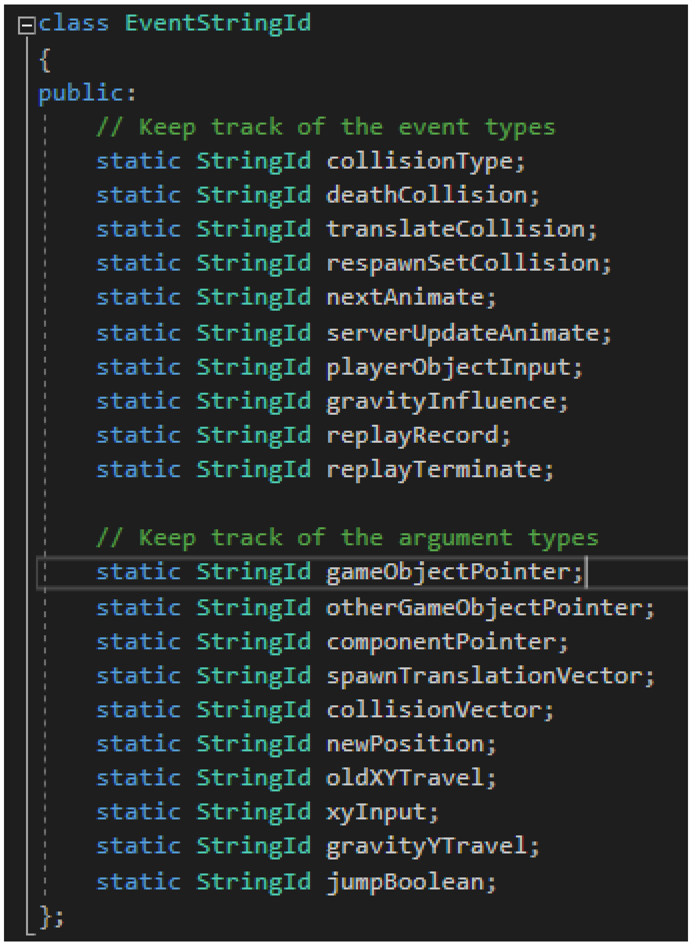
Section 1:

New classes:



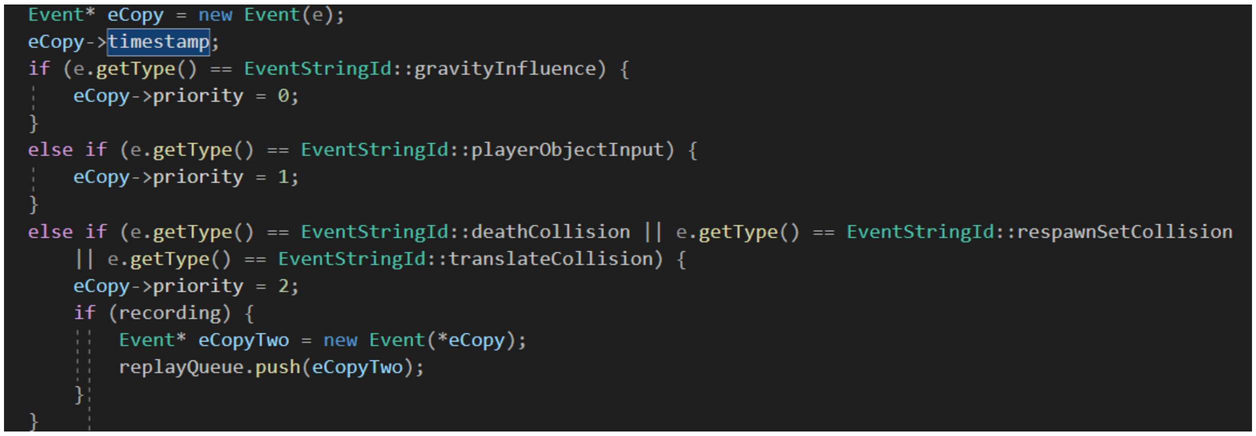
```
Events
    AbstractEventHandler.h
    AnimateEventHandler.h
    CollisionEventHandler.h
    EventManager.h
    Events.h
    InputEventHandler.h
```

The reason behind longer compiling times:



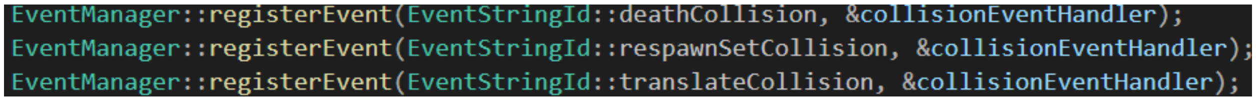
```cpp
class EventStringId
{
public:
    // Keep track of the event types
    static StringId collisionType;
    static StringId deathCollision;
    static StringId translateCollision;
    static StringId respawnSetCollision;
    static StringId nextAnimate;
    static StringId serverUpdateAnimate;
    static StringId playerObjectInput;
    static StringId gravityInfluence;
    static StringId replayRecord;
    static StringId replayTerminate;

    // Keep track of the argument types
    static StringId gameObjectPointer;
    static StringId otherGameObjectPointer;
    static StringId componentPointer;
    static StringId spawnTranslationVector;
    static StringId collisionVector;
    static StringId newPosition;
    static StringId oldXYTravel;
    static StringId xyInput;
    static StringId gravityYTravel;
    static StringId jumpBoolean;
};
```

Example of setting priority and duplicating for replayQueue:



```
Event* eCopy = new Event(e);
eCopy->timestamp;
if (e.getType() == EventStringId::gravityInfluence) {
    eCopy->priority = 0;
}
else if (e.getType() == EventStringId::playerObjectInput) {
    eCopy->priority = 1;
}
else if (e.getType() == EventStringId::deathCollision || e.getType() == EventStringId::respawnSetCollision
    || e.getType() == EventStringId::translateCollision) {
    eCopy->priority = 2;
    if (recording) {
        Event* eCopyTwo = new Event(*eCopy);
        replayQueue.push(eCopyTwo);
    }
}
```

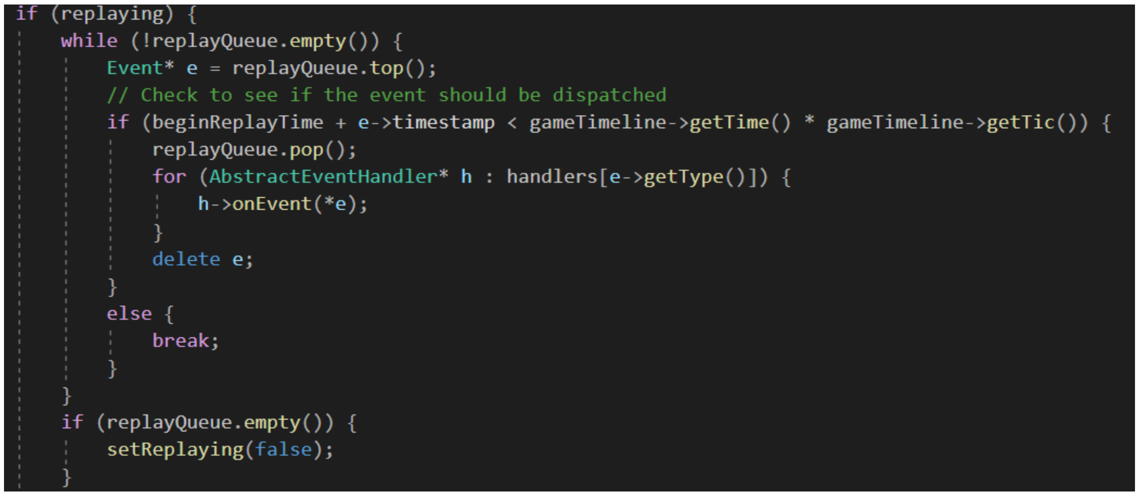
Example of registering events:



```
EventManager::registerEvent(EventStringId::deathCollision, &collisionEventHandler);
EventManager::registerEvent(EventStringId::respawnSetCollision, &collisionEventHandler);
EventManager::registerEvent(EventStringId::translateCollision, &collisionEventHandler);
```

Section 2:

Code handling replay (server-side ~ the client has a different set of calculations in the if statement just before popping from the queue to demonstrate my attempted implementation of speed up/slow down replays):



```
if (replaying) {
    while (!replayQueue.empty()) {
        Event* e = replayQueue.top();
        // Check to see if the event should be dispatched
        if (beginReplayTime + e->timestamp < gameTimeline->getTime() * gameTimeline->getTic()) {
            replayQueue.pop();
            for (AbstractEventHandler* h : handlers[e->getType()]) {
                h->onEvent(*e);
            }
            delete e;
        }
        else {
            break;
        }
    }
    if (replayQueue.empty()) {
        setReplaying(false);
    }
```