CSC581 Homework 4 Writeup

Jae Jimmy Wong

1 Introduction

The main task of this homework is to design an event management system for a game engine. The functionalities from past homework, including multiplayer and object models, also need to be implemented in this homework. Especially, networking is closely related to event management.

2 Event Management System

Event Representation

The core objective of an event representation is being flexible enough to store any type of event. To achieve this goal, I utilized `std::variant` which is available in the standard library since C++17. The class template `std::variant` represents a type-safe data structure that holds a value of one of its alternative types. While combining with `std::map`, it becomes a key-value store that can hold multiple types of values at the same time. Therefore, it is used to maintain the parameter fields in my event presentation.

Event Registration

To support multiple handlers for a single type of event, the event manager associates each event type with an array of handler functions. The dynamic array type `std::vector` is used for this purpose. Instead of a C function pointer, I choose to use `std::function` polymorphic function wrapper. `std::function` can hold not only standard function pointers but also callable objects, so complex event handlers may be written as C++ classes.

Event Raising

When an event is raised, the time of raising is recorded inside the event data structure. The timestamp is later used to determine the order of handling and the replay system. I use an append-only data structure for storing raised events. This provides two benefits for event handling. First, events are naturally ordered according to the time-based priority so the insertion is done in constant time. Second, the list of all events after a specific timestamp

can be searched in an execution time of logarithm complexity. As a result, event raising and handling is faster than a straightforward priority-queue design.

Event Handling

To prepare for the replay system, an abstraction of the game world is included in the event handling system. A "world" is the collection of all game objects that are present at a single time point. When an event is applied to a world, the world transforms into a new world that has its timestamp advanced and some of its game objects updated. This can be represented as the following equation:

$$world_{i+1} = apply(world_i, event_{i+1}) \tag{1}$$

When the abstraction is paired with the data structure storing raised events, the system can advance the time of the game world to a precise timestamp.

3 Networked Event Management

Following the technology stack from previous homework, ZeroMQ is used as the network sockets and JSON is used as the serialization format. To be accurate, the actual library that is used to implement the network event management is `cppzmq` and `nlohmann-json`. A server-centric design is used: on the server side, a dedicated ZMQ reply socket is created in a thread to receive client events and raise them; on the client side, the corresponding ZMQ request socket is created to send all events to the server. Since the events need to be sent through sockets, the event representations have to be serialized into JSON strings. This is tricky because the data types in the representation are flexible and the JSON type system is different from C++. I managed to handle it by mapping the parameter data types to JSON data types one by one.

4 Replays

The replay system involves many components of the event system, which we discussed in section 2 and section 3. In addition to the involvements, the system itself needs dedicated recording storage because a client is viewing a world different than others' while replaying. The recording storage maps a user to a recording. Each recording has a copy of the world of recording start and may have a `record_end` timestamp and a `replay_end` timestamp. When a recording ends and a replaying starts, the system starts to apply events to the stored world according to elapsed time. The stored world is then sent to the corresponding client for display.

5 Conclusion

In this homework, I designed a networked event management system that features high performance and replaying capabilities. By using raised time as the priority, the insertion and search of events is done in lower time complexity compared to the traditional method. By using the abstraction of the world and events, the system allows each player to view their replay world.