Multiplayer Game

301CR – Assignment 1

Contents

[Introduction 3](#_Toc534634359)

[Playable Game 3](#_Toc534634360)

[Networking Multiplayer Functionality 3](#_Toc534634361)

[Middleware Used 3](#_Toc534634362)

[Game Systems 3](#_Toc534634363)

[Windowing 3](#_Toc534634364)

[Networking 3](#_Toc534634365)

[Audio Subsystem 4](#_Toc534634366)

[Graphics Subsystem 4](#_Toc534634367)

[User Input 4](#_Toc534634368)

[File Loading 4](#_Toc534634369)

# Introduction

This assignment is separated in two tasks. One is creating playable game and implementing different subsystems to it and the other one is adding multiplayer functionality to it.

## Playable Game

For the playable game I must implement the following subsystems: Graphics, Physics, Resource Management, Human Interface, Initialise/Shutdown, Networking and Profiling.

## Networking Multiplayer Functionality

The playable game should be extended to support multiplayer functionality, using a middleware library (SFML)

# Middleware Used

For this project I’ve used SFML as middleware for all my subsystems.

# Game Systems

For this project I managed to implement the following systems: Windowing, Graphics, Networking, Audio, User Input and File Loading. Below are all the systems explained.

## Windowing

The task of this subsystem is to create the window in which other subsystem will execute (Figure 1.1). This subsystem uses observer pattern, which notifies it when to close the window (Figure 1.2). The window makes use of the event system and waits for a close command. The window also uses the singleton pattern as the clients will never need more than one window running.

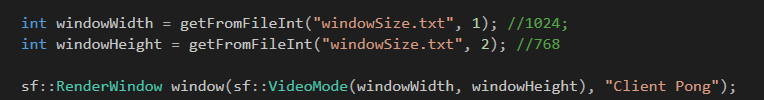


Figure .1 Window creation

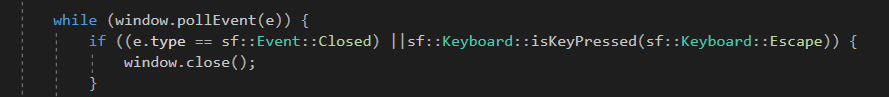


Figure 1.2 Observer Pattern

The windowing subsystem also checks if the user has selected the current window and only then proceeds to take user input. This is an extra protection to guarantee that each client can control only its own bat. The reason I take this step is SFML doesn’t separate user input from the active window.

## Networking

The networking subsystem is tasked with sending and receiving data between the server and all of the clients. For the making of this subsystem I used packets and SFML/Networking. The architecture of this system is the following:

1. The server starts and wait for connections.
2. Client tries to connect to the server.
3. Server checks if the client is still connected. If it is, it proceeds to the next step. Otherwise goes back to one
4. The server receives the movement state from the client.
5. The server does all of the calculations for every movement in the scene.
6. The server sends the updated locations of the players and the ball to the clients.
7. Clients update their own positions and draw.

Both the server and the clients use fixed time step for sending packets and frame limit. This guarantees that the game will run with the same speed on every client and every machine. The data send between the clients and the server makes use of the SFML types, which guarantees that the types sent take the same amount of bytes on both 64 bit and 32 bit systems. Dead reckoning is not implemented which results in latency issues in the application

## Audio Subsystem

For the audio subsystem I use SFML/Audio. This subsystem is applied as header to the client class and it has a couple different functions – loadAudioFile, setAudioState and runAudio. The sounds are declared in the client, outside of the game loop. The application looks for a background music with a given name, loads it and runs it before the game loops start.

## Graphics Subsystem

The graphics subsystem uses SFML/Graphics. It’s used both in the client and in the server. Although the server never draws anything, I use graphics types (e.g. RectShape) to make the ball Calculations. This is by far the most important subsystem as without it, nothing would be displayed on screen. The Graphics subsystem also makes use of Object Oriented Programming – every client has an object for the ball and for the bat and on received packets it updates their positions.

## User Input

User input uses the observer pattern. The subsystem is waiting to detect a pressed key. When this happens, it changes the state of that client. As long as the state goes through a few conditions, it gets sent to the server in a packet.

## File Loading

This subsystem is a header file in the client. It allows the client to read from text file and load the window size, the IP the clients is looking for and the Port it takes. It is a very simple class, that can read up to two variables and assign them to the desired variables.

# Reflection

I am extremely unsatisfied with this coursework. The most important lesson I learnt is doing more extensive research on the project I want to create. This Pong game was the 4th project I attempted. Before that I scraped a 3d Isometric game, Card Trading Game and an action Chess game. For each one of those I came across issues that eventually I realized will take too much time to be done. On a more positive node, I studied SFML in depth and I have way better understanding of how networking works. I also spent quite a lot of time studying dead reckoning algorithm, but I didn’t have the time to finish it. I feel like I created a good implementation of the networking architecture, which would’ve been working perfectly with Dead Reckoning Implementation. I also understood that networking programming is not as harsh as I expected and is a field that I will probably explore more in the future.

# Bibliography

Sourcemaking.com. (2019). *Design Patterns and Refactoring*. [online] Available at: https://sourcemaking.com/design\_patterns [Accessed 7 Jan. 2019].

Freemusicarchive.org. (2019). *Free Music Archive*. [online] Available at: http://freemusicarchive.org/ [Accessed 7 Jan. 2019].