Jayden Sansom, jksanso2

Homework 2: Time and Networking Foundations

Introduction

This assignment allowed for an introduction to networking within the SFML and ZeroMQ libraries. The following sections explore the design decisions made in each of the assignment parts.

Multithreaded Loop Architecture

To create a multithreaded architecture I began with altering the given ThreadExample class to contain a function to run in the thread as well as setting up the mutex to lock in necessary class functions. During the initialization of a Thread object, the user will now also have to include a function that will be executed on that thread when run. This allowed for a more dynamic implementation of the threads as they would not have to rely on a specific extended class. Additionally, I locked the mutex in functions such as run() and isBusy() to ensure that my game loop would be thread-safe.

When choosing the tasks to run on separate threads, I immediately identified my two update functions, one for updating the player and one for updating the moving platform. By separating these into two separate threads, I would be able to speed up the calculations of each loop, bringing them back together before the next frame. The one issue I came across was the error of X11 libraries being requested when running. While this was due to what tasks I chose to thread, it was solved by separating the keyboard input checks from inside the Player class and extracting them to the main function where they would then be passed to the Player update function.

Measuring and Representing Time

The next portion of the assignment was built upon the previous task of multithreading, meaning that in future multithreading requirements, I can be ensured that it will work with my created Timeline. The Timeline class followed the structure laid out in the lecture, including a tic speed, start time, elapsed time, last paused time, and whether it is currently paused. Omitted from the base structure is the anchor, which I found to not work with my eventual implementation.

The main challenge I faced was logically understanding the conversion and flow from the seconds I was using in the main loop and the milliseconds I needed to use in the Timeline class. This led to much trial and error, but eventually, I found the right combination of dividing and subtracting to allow my getTime() function to deliver the proper time. Additionally, I was having issues when trying to get the window to pause correctly but eventually found that putting an isPaused() check in the main loop would allow me to set the elapsed time to zero instead of risking a miscalculation. Finally, changing the tic speed from key input proved to work immediately, but led to some confusion as I misunderstood that it would change the literal speed of the game rather than its frame change rate.

Networking Basics

Thankfully, during this portion of the assignment, the tutorials and examples provided by the ZeroMQ documentation were incredibly helpful. I decided to take from both the REQ-REP and PUB-SUB models rather than just one of the two due to some advice I had received. By using both I will be able to handle Client connections on the REQ-REP sockets and handle game data on the PUB-SUB sockets. Figures A.1 and A.2 show the results of the Clients connecting to the Server and receiving messages with data for themselves and other connected Clients.

I eventually learned that the Server, containing the publisher and replier, would be able to run the replier in the main loop to constantly check for new Clients while running the publisher on a separate thread, the class created in part one, so that it would calculate the information of each Client and send it to their connect subscribers. The Client worked differently, running the requester and waiting for the Server to confirm their connection before starting the data collection from the subscriber, running in a loop. While the system of running the requester first would not change, I realized that in consideration of future architecture, the subscriber would likely have to be placed in a separate thread so that it can constantly look for new messages from the Server containing game data while also updating the player in the main loop.

Putting It All Together

With all of the previous parts of the assignment completed, I was ready to begin piecing them together. I started by migrating my work from part two over into a Server folder. I removed most of the code within the main game loop, deleting the player sprite and input checks, leaving the platforms and their updates remaining. This allowed me to begin my Server class where I made my replier wait for clients on a separate thread and I had my publisher run in the main loop, sending the updated platform information to the client through strings.

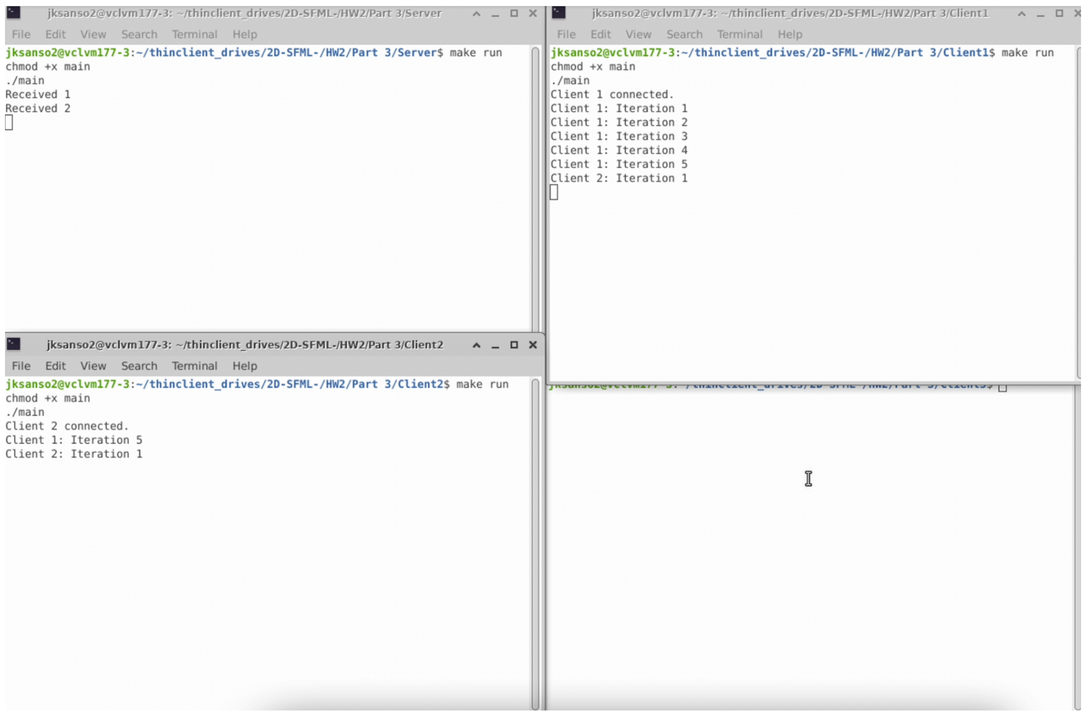
With the base implementation of the Server completed, I moved to work on the client by duplicating all class files over to a separate folder. In the main file, I kept the creation of the platforms, as well as a character to be used by the Client. In the structure, I had to run the subscriber on another thread to check for all game object updates, while the requester ran to initialize the player but also within the game loop to send character updates to the Server. In this design, I ran into many issues, mainly surrounding pointers to objects and characters and how to send that information. After resolving these issues with PlayerClient and Client structs, I was able to send information between the Client and Server through a pseudo-pipe.

Ultimately I achieved my original design goal of having the server contain all shared objects, write shared objects and player information to a message that is sent to a Client, and receive player information from all Clients. Simultaneously, the Client also contains individual information characters and instances of shared objects, writes character updates to a message and sends it to the Server, and receives a list of all drawable objects from the Server, including other Client characters. This was achieved using both the REQ-REP and PUB-SUB models, running in parallel via threads to achieve a quality connection of sending updated information and connecting clients. The results of the design can be seen in Figures B.1 - B.4 where it shows the Client connections and data relation to the Server. Ultimately, I believe that my final design will be able to be slightly tweaked instead of entirely restructured to fit the upcoming requirements of assignments as it was designed with the goal in mind of allowing for easy restructuring and flexibility. By implementing functionality into separate classes and having the ability to build onto them, I believe that my design will be able to withstand the future and grow to be a versatile engine.

Jayden Sansom, jksanso2

Appendix A

Networking Basics

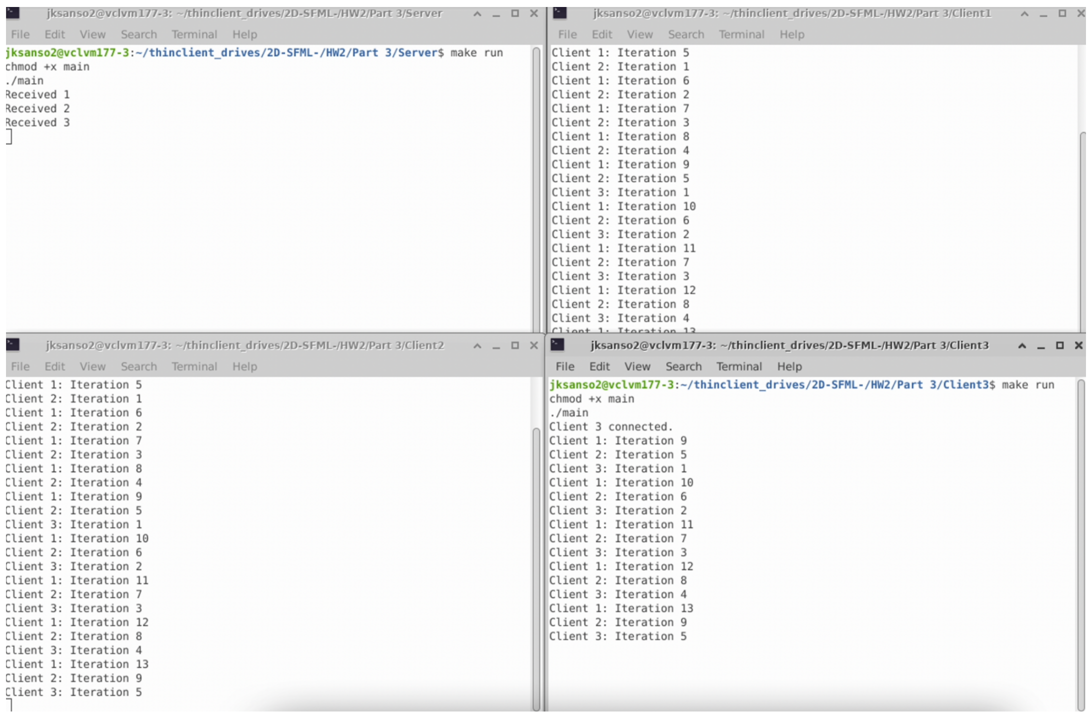


```
jksanso2@vclvm177-3:~/thinclient_drives/2D-SFML-/HW2/Part 3/Server$ make run
chmod +x main
./main
Received 1
Received 2
```

```
jksanso2@vclvm177-3:~/thinclient_drives/2D-SFML-/HW2/Part 3/Client1$ make run
chmod +x main
./main
Client 1 connected.
Client 1: Iteration 1
Client 1: Iteration 2
Client 1: Iteration 3
Client 1: Iteration 4
Client 1: Iteration 5
Client 2: Iteration 1
```

```
jksanso2@vclvm177-3:~/thinclient_drives/2D-SFML-/HW2/Part 3/Client2$ make run
chmod +x main
./main
Client 2 connected.
Client 1: Iteration 5
Client 2: Iteration 1
```

Figure A.1: Client 1 connected to the Server, Client 2 connecting at a later time and receiving data from the Server.



```
jksanso2@vclvm177-3:~/thinclient_drives/2D-SFML-/HW2/Part 3/Server$ make run
chmod +x main
./main
Received 1
Received 2
Received 3
```

```
Client 1: Iteration 5
Client 2: Iteration 1
Client 1: Iteration 6
Client 2: Iteration 2
Client 1: Iteration 7
Client 2: Iteration 3
Client 1: Iteration 8
Client 2: Iteration 4
Client 1: Iteration 9
Client 2: Iteration 5
Client 3: Iteration 1
Client 1: Iteration 10
Client 2: Iteration 6
Client 3: Iteration 2
Client 1: Iteration 11
Client 2: Iteration 7
Client 3: Iteration 3
Client 1: Iteration 12
Client 2: Iteration 8
Client 3: Iteration 4
Client 1: Iteration 13
```

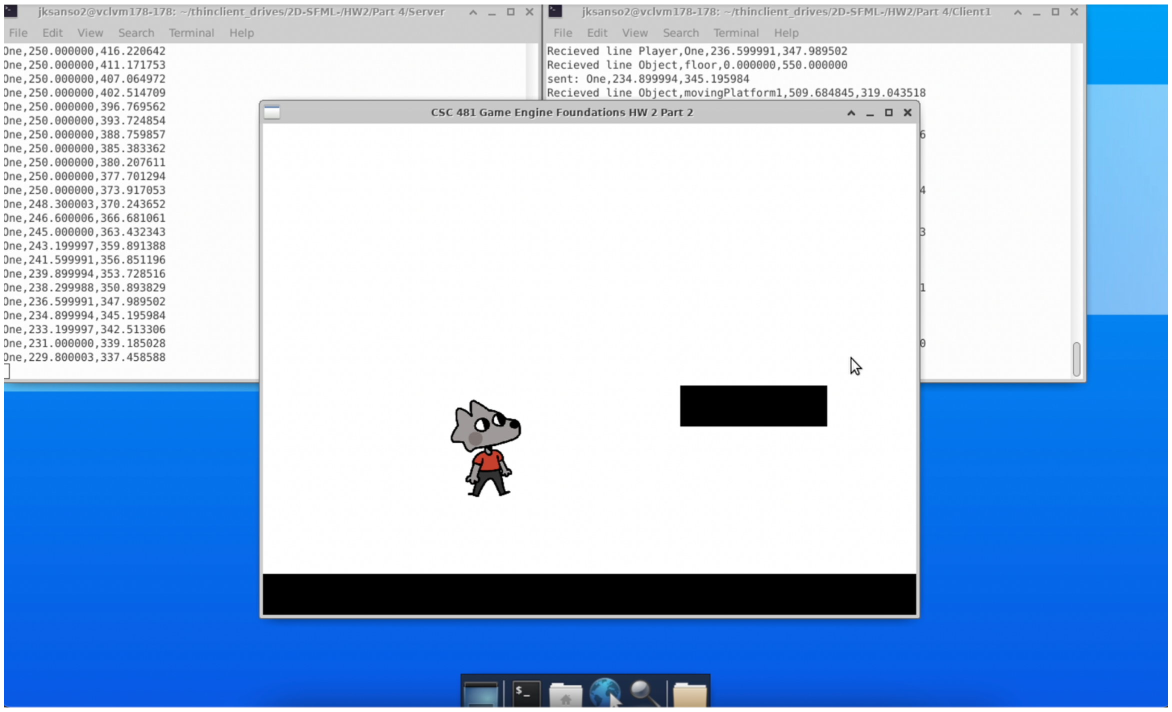
```
Client 1: Iteration 5
Client 2: Iteration 1
Client 1: Iteration 6
Client 2: Iteration 2
Client 1: Iteration 7
Client 2: Iteration 3
Client 1: Iteration 8
Client 2: Iteration 4
Client 1: Iteration 9
Client 2: Iteration 5
Client 3: Iteration 1
Client 1: Iteration 10
Client 2: Iteration 6
Client 3: Iteration 2
Client 1: Iteration 11
Client 2: Iteration 7
Client 3: Iteration 3
Client 1: Iteration 12
Client 2: Iteration 8
Client 3: Iteration 4
Client 1: Iteration 13
Client 2: Iteration 9
Client 3: Iteration 5
```

```
jksanso2@vclvm177-3:~/thinclient_drives/2D-SFML-/HW2/Part 3/Client3$ make run
chmod +x main
./main
Client 3 connected.
Client 1: Iteration 9
Client 2: Iteration 5
Client 3: Iteration 1
Client 1: Iteration 10
Client 2: Iteration 6
Client 3: Iteration 2
Client 1: Iteration 11
Client 2: Iteration 7
Client 3: Iteration 3
Client 1: Iteration 12
Client 2: Iteration 8
Client 3: Iteration 4
Client 1: Iteration 13
Client 2: Iteration 9
Client 3: Iteration 5
```

Figure A.2: All three Clients are connected to the Server and receiving messages from the Server about separate Clients.

Appendix B

Putting It All Together

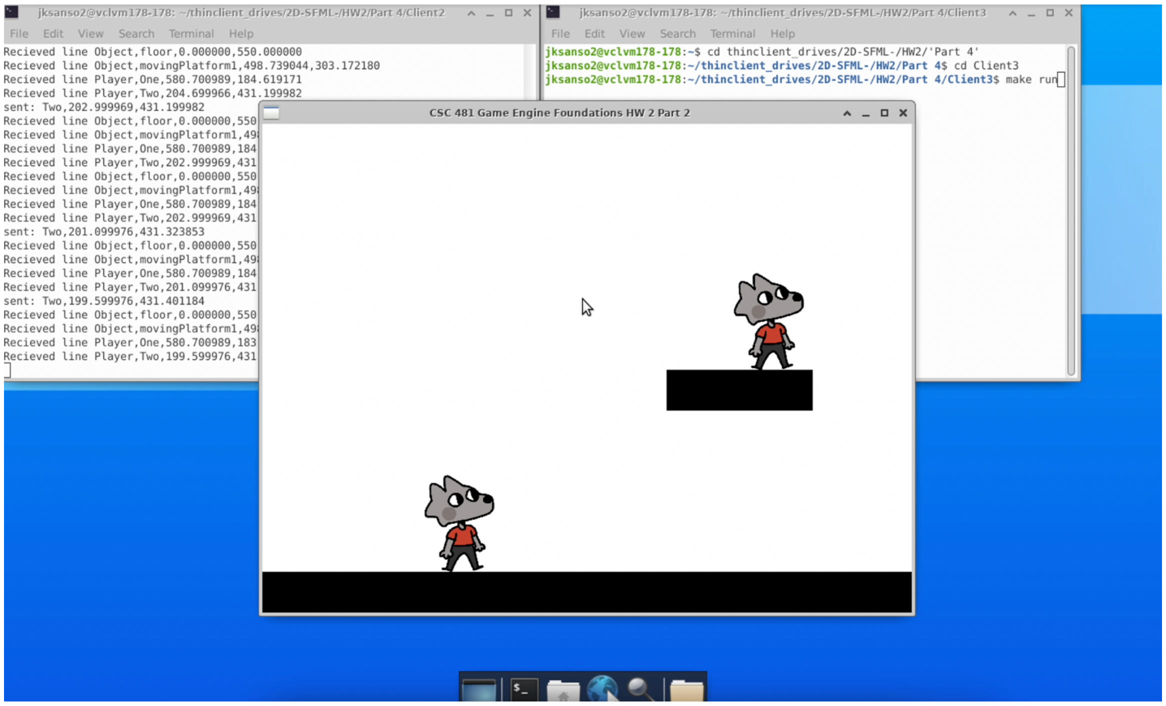


```
jksanso2@vclvm178-178: ~/thinclient_drives/2D-SFML-/HW2/Part 4/Server
One,250.000000,416.220642
One,250.000000,411.171753
One,250.000000,407.064972
One,250.000000,402.514709
One,250.000000,396.769562
One,250.000000,393.724854
One,250.000000,388.759857
One,250.000000,385.383362
One,250.000000,380.207611
One,250.000000,377.701294
One,250.000000,373.917053
One,248.300003,370.243652
One,246.600006,366.681061
One,245.000000,363.432343
One,243.199997,359.891388
One,241.599991,356.851196
One,239.899994,353.728516
One,238.299988,350.893829
One,236.599991,347.989502
One,234.899994,345.195984
One,233.199997,342.513306
One,231.000000,339.185028
One,229.800003,337.458588
```

```
jksanso2@vclvm178-178: ~/thinclient_drives/2D-SFML-/HW2/Part 4/Client1
Recieved line Player,One,236.599991,347.989502
Recieved line Object,floor,0.000000,550.000000
sent: One,234.899994,345.195984
Recieved line Object,movingPlatform1,509.684845,319.043518
```

File Edit View Search Terminal Help
CSC 481 Game Engine Foundations HW 2 Part 2

Figure B.1: Client 1 connected, receiving platform data and sending player data to and from the Server.

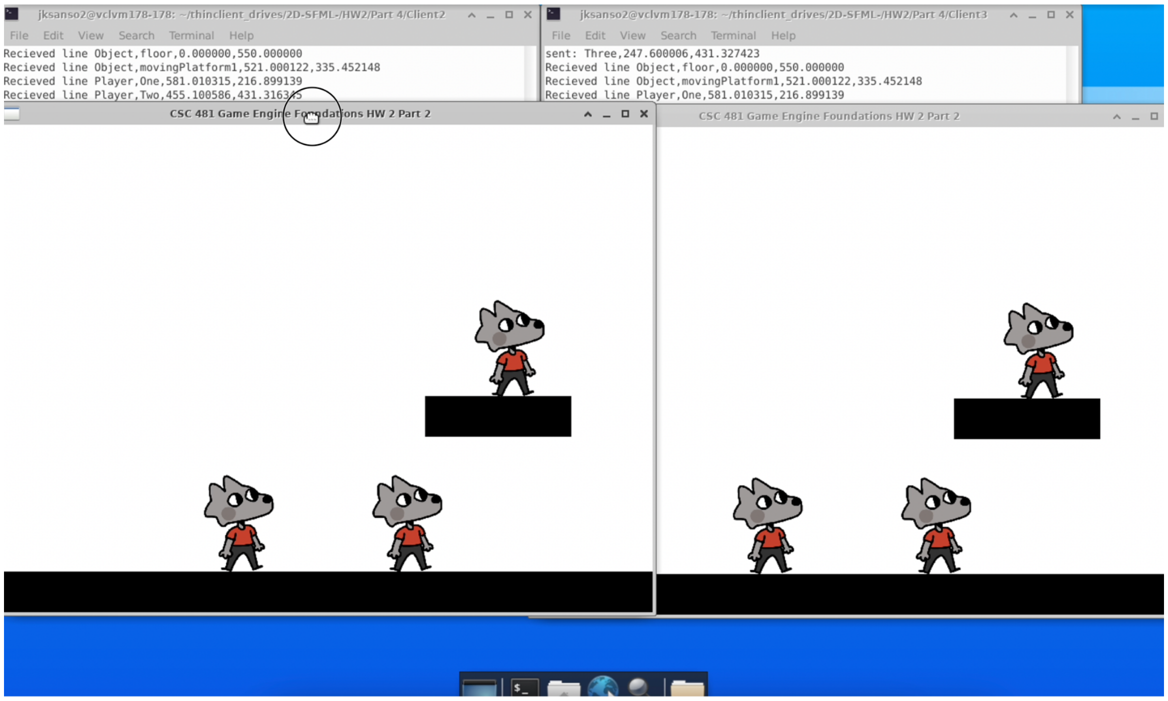


```
jksanso2@vclvm178-178: ~/thinclient_drives/2D-SFML-/HW2/Part 4/Client2
Recieved line Object,floor,0.000000,550.000000
Recieved line Object,movingPlatform1,498.739044,303.172180
Recieved line Player,One,580.700989,184.619171
Recieved line Player,Two,204.699966,431.199982
sent: Two,202.999969,431.199982
Recieved line Object,floor,0.000000,550
Recieved line Object,movingPlatform1,49
Recieved line Player,One,580.700989,184
Recieved line Player,Two,202.999969,431
Recieved line Object,floor,0.000000,550
Recieved line Object,movingPlatform1,49
Recieved line Player,One,580.700989,184
Recieved line Player,Two,202.999969,431
sent: Two,201.099976,431.323853
Recieved line Object,floor,0.000000,550
Recieved line Object,movingPlatform1,49
Recieved line Player,One,580.700989,184
Recieved line Player,Two,201.099976,431
sent: Two,199.599976,431.401184
Recieved line Object,floor,0.000000,550
Recieved line Object,movingPlatform1,49
Recieved line Player,One,580.700989,183
Recieved line Player,Two,199.599976,431
```

```
jksanso2@vclvm178-178: ~/thinclient_drives/2D-SFML-/HW2/Part 4/Client3
jksanso2@vclvm178-178:~$ cd thinclient_drives/2D-SFML-/HW2/'Part 4'
jksanso2@vclvm178-178:~/thinclient_drives/2D-SFML-/HW2/Part 4$ cd Client3
jksanso2@vclvm178-178:~/thinclient_drives/2D-SFML-/HW2/Part 4/Client3$ make run
```

File Edit View Search Terminal Help
CSC 481 Game Engine Foundations HW 2 Part 2

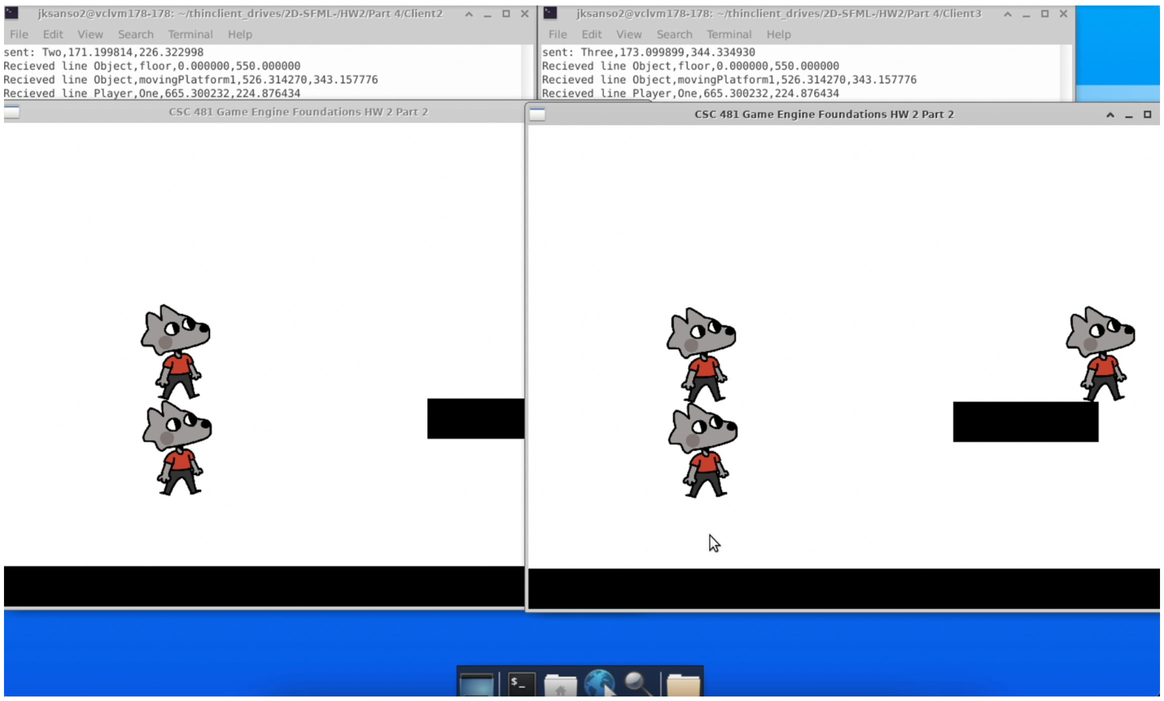
Figure B.2: Client 2 connected, showing Client 1 moving on the moving platform.


jksanso2@vclvm178-178: ~/thinclient_drives/2D-SFML-/HW2/Part 4/Client2
File Edit View Search Terminal Help
CSC 481 Game Engine Foundations HW 2 Part 2
jksanso2@vclvm178-178: ~/thinclient_drives/2D-SFML-/HW2/Part 4/Client3
File Edit View Search Terminal Help
CSC 481 Game Engine Foundations HW 2 Part 2

```
Recieved line Object,floor,0.000000,550.000000
Recieved line Object,movingPlatform1,521.000122,335.452148
Recieved line Player,One,581.010315,216.899139
Recieved line Player,Two,455.100586,431.316345
```

```
sent: Three,247.600006,431.327423
Recieved line Object,floor,0.000000,550.000000
Recieved line Object,movingPlatform1,521.000122,335.452148
Recieved line Player,One,581.010315,216.899139
```

Figure B.3: All three Clients connected, Client 2 and Client 3 windows are shown.


jksanso2@vclvm178-178: ~/thinclient_drives/2D-SFML-/HW2/Part 4/Client2
File Edit View Search Terminal Help
CSC 481 Game Engine Foundations HW 2 Part 2
jksanso2@vclvm178-178: ~/thinclient_drives/2D-SFML-/HW2/Part 4/Client3
File Edit View Search Terminal Help
CSC 481 Game Engine Foundations HW 2 Part 2

```
sent: Two,171.199814,226.322998
Recieved line Object,floor,0.000000,550.000000
Recieved line Object,movingPlatform1,526.314270,343.157776
Recieved line Player,One,665.300232,224.876434
```

```
sent: Three,173.099899,344.334930
Recieved line Object,floor,0.000000,550.000000
Recieved line Object,movingPlatform1,526.314270,343.157776
Recieved line Player,One,665.300232,224.876434
```

Figure B.4: Only the Client whose window is focused on will receive keyboard input. Client 3 jumps while under Client 2, colliding and moving together while Client 1 remains on the other side of the window.