Disclaimer: I have divided my work strangely – Solution 1 located inside the folder Project 1 contains my work that is relevant to the entire assignment (sections 1-4). Within the solution, there are two projects – one for the server, and another for the client.

Solutions 2 and 3 are located inside folders Project 2 and Project 3 respectively. These two solutions are directly related to Section 3 of the assignment.

Section 1: I have written my code to handle the platforms and player objects to be handled through separate threads from the main one. I first created an Abstract Thread as an interface for my two threads to inherit from. Within the abstract thread, I kept track of the mutex and condition variable. The PlayerThread and PlatformThread each held a pointer to the list of all Platforms and Players in order to allow the Player and Platform objects check for collision easier. In order to make it so that multithreading had any use for my program, I shuffled a few of the calculations within my Platform class so that I avoid running everything under a mutex.

Section 2: For the time portion, I did something similar to my process for creating the thread classes. I created an AbstractTimeline for my GameTime and RealTime to inherit from. Within the abstract class, I put a function for toggling pause, speeding up, slowing down. (NOTE: At the final stage of the game, I made it so that the server is the only one that can pause platforms – the clients all have their own GameTime objects that they're linked to – all clients can speed up or slow down their relative speeds by pressing the < or > symbol. – the space bar will toggle pause for the client/server triggering it). An unfortunate thing I have come to realize after the completion of this portion is that my cycle calculations isn't very good for my moving platforms (if I increase the speed of time significantly, the platform will likely disappear). I will need to

rethink my cycle calculations for objects that will fit my Time objects better. In order to make sure my time calculations are consistent, even when speeds (tic size) have changed, I gave my Time objects a getDeltaTime() function. This function will readjust calculations whenever tic size as changed. This is done by having my Time objects keep track of when the last time getDeltaTime() was called. Originally, when I was making my Time objects, I was using unsigned long variables to keep track of all my data. This proved to be a big issue for me as it was constantly losing data. To fix this, I replaced them with long doubles.

Section 3: For the server-client portion, I separated my work into two projects for easier debugging. It took me a while, but I managed to figure out a way to handle client server iterations with a combination of the PUB/SUB model and the REQ/REP model. The server creates a ClientJoinThread that will handle all new clients joining via REQ/REP, while the main thread will continuously pump out information on iterations through the PUB/SUB model. The clients, once through registering via REQ/REP, will continuously receive information on iterations. One last second interesting thing I noticed about my code is that the clients will always miss the first iteration for some reason. Another strange thing is that the pause button is unresponsive when either of the combination of keys are pressed: top+left or bottom+right. The pause button is responsive in every other scenario.

Section 4: Integration of my main code with server-client code took a good deal of time. I first separated my project into one for clients and one for servers. I intend to have the server act as a host (client and server) – although I have not fully implemented that.

The character for the host is reflected on all other clients, but none of the other characters are reflected on the host yet. Similar to Section 3, I had my server set up a thread to handle joining clients. This thread, however, also handles regular updates from clients about their respective positions. All the information being sent to clients and server is through JSON – Each JSON will carry information pertaining to position of an object. The positions of the Platforms are sent to the clients, all of which in turn calculate collisions for their respective Character object and return the updated position of the character.

Section 2:

Snippet of code calculating delta time for game time.

```
/**
* Calculates the deltatime based on last call to function in terms of gametime.
* @return the time since the last call to getDeltaTime function
*/
long double GameTime::getDeltaTime()
{
	long double now = getTime();
	long double deltaTime = now - lastDeltaTimeCall;
	lastDeltaTime = deltaTime;
	this->lastDeltaTimeCall = now;
	return deltaTime;
}
```

Snippet of code recalculating times after tic size change.

```
/**
* Sets the tic to a new value
* @param new_tic new tic value being set
*/
void GameTime::setTic(long double new_tic)
{
	std::cout << lastDeltaTime << std::endl;
	if (new_tic > 0) {
```

```
			this->lastDeltaTimeCall *= tic / new_tic;
			this->tic = new_tic;
		}
}
```

Section 3-4:

Snippet of code thread that handles incoming clients.

```
		try {
			zmq::socket_t socket(*context, ZMQ_REP);
			zmq::socket_t socketMovement(*context, ZMQ_REP);
			socket.bind("tcp://*:5555");
			socketMovement.bind("tcp://*:5554");

			while (*running) {
				zmq::message_t request;
				//  Keep checking for new client
				if (socket.recv(&request, ZMQ_DONTWAIT)) {
					std::cout << "Received new client." << std::endl;
					// Set up info transfer to clients
					json jInfo;
					{
						std::unique_lock<std::mutex> lock(*this->getMutex());
						//this->getConditionVar()->wait(lock);
						// Provide information on platforms to client
						jInfo.push_back(entityList->size());
						for (sf::Shape* s : *this->entityList) {
							Platform* p = dynamic_cast<Platform*> (s);
							sf::Vector2f tempPosition = p->getPosition();
							float tempPositionArray[4] = { tempPosition.x,
tempPosition.y, p->getOrientation(), p->getSpeed() };
							jInfo.push_back(tempPositionArray);
						}
```