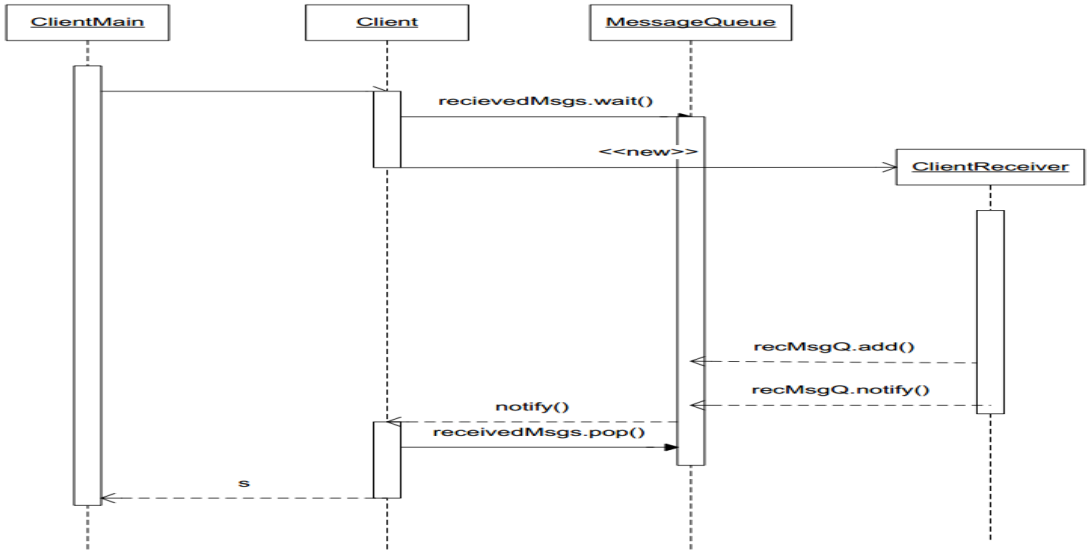
Documentation of Milestone 1: Bomberman

Introduction: Multilayer Bomberman is a small-scale MVC project that consists of a communication protocol, game state, and a gui. UDP is used for the communication protocol between the server and client. The gui is on the client side and the game state/model is on the server side. Refer to the class diagram PDF for the full details on classes used.

Design: To read more about the message specification that was used refer to the ReadMe.txt file attached to the project. The reasoning behind choosing UDP as the communication protocol is that there is less overhead rather than TCP. Also in addition there is no need for all the services of TCP and order of the packets arriving doesn't matter. To run multiple clients on the bomberman game threads are needed. With multiple threads changing a global variable the possibility of data corruption increases, therefore synchronization between client threads and locks to ensure mutual exclusion.

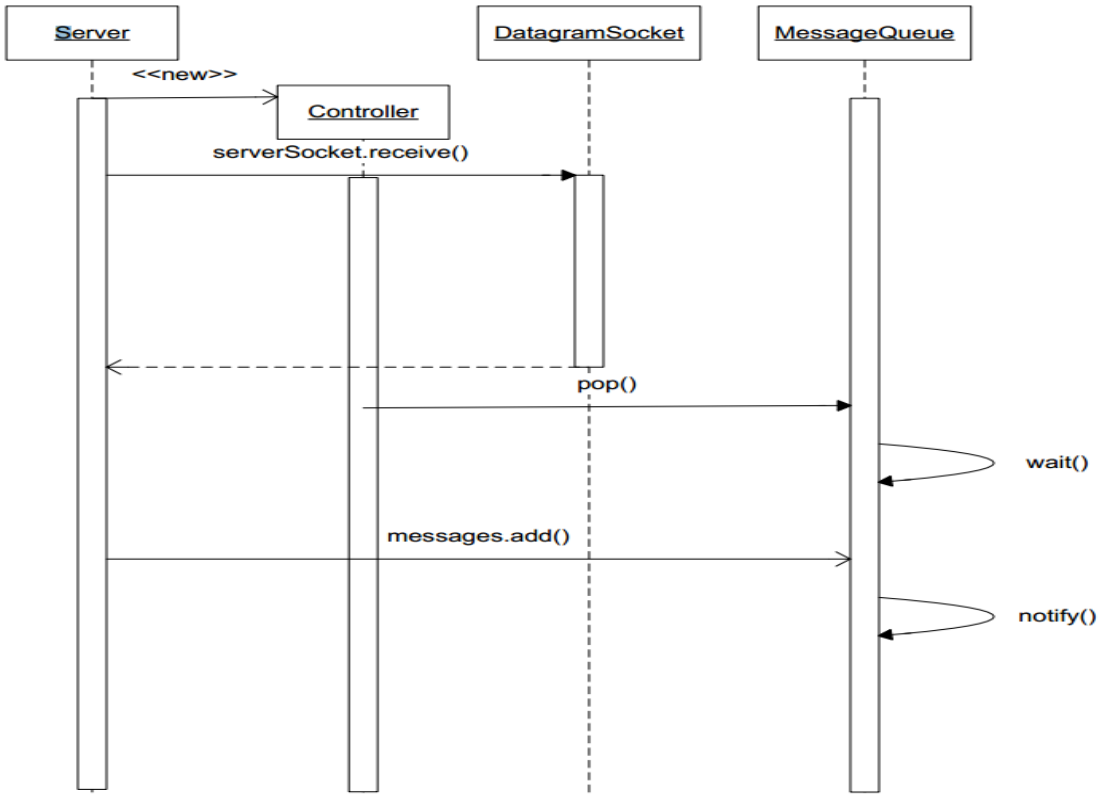
Synchronization client-side: There are four synchronization methods in the MessageQueue class: add, pop, isEmpty, and clear. The reason why these methods are synchronized is to ensure mutual exclusion on the shared reference variable of type MessageQueue. The run method inside the ClientSender class is locked on the variable sendMsgQ because it will need to pop off an item from the message queue. To prevent multiple clients popping off items from an empty queue synchronization between the clients is handled by the while loop checking to see if the queue is empty and waiting (blocking the program) if it is. The client class shares this message queue variable and calls notify() when a message has been added to the queue. The notify() lets the thread that is waiting on the locked object to continue to execute.



(Figure 1: Client-side sequence diagram)

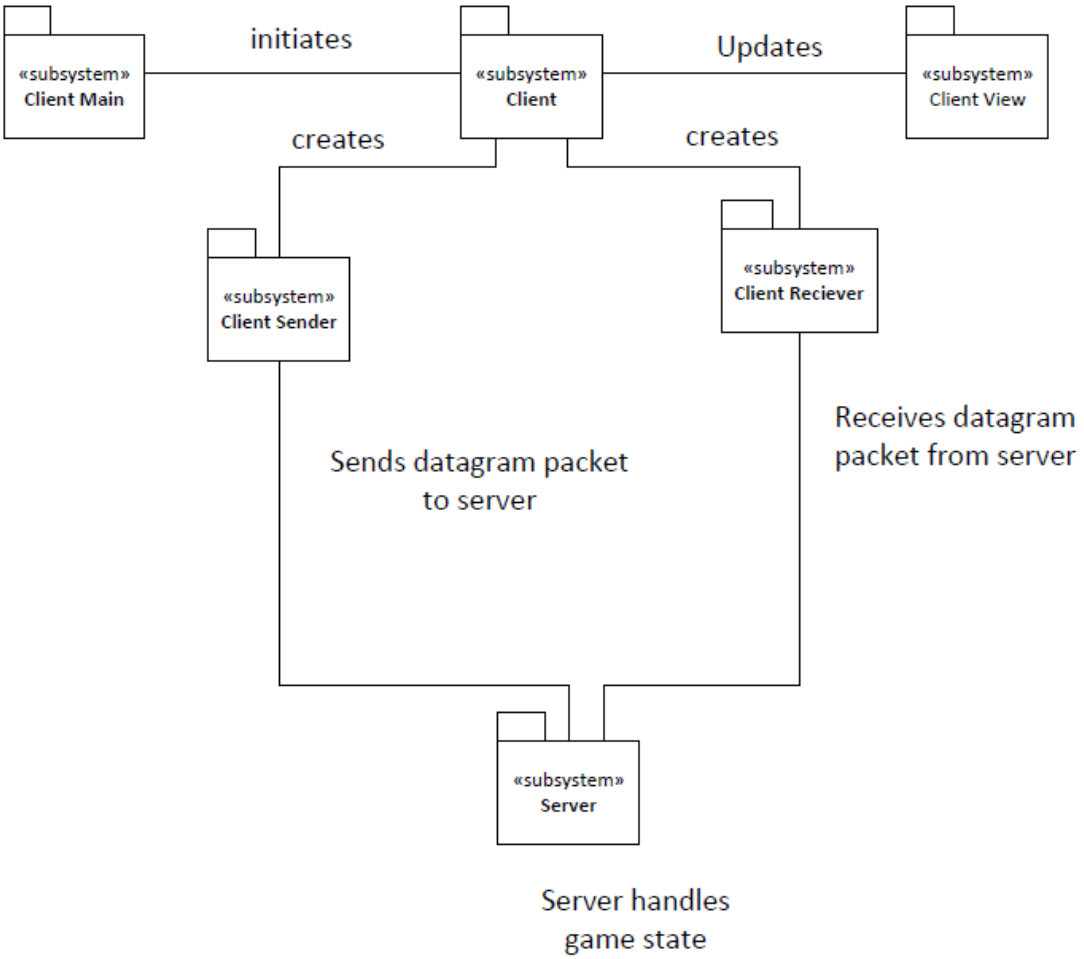
Overall the program is executing multiple threads client receiver client sender and the main thread. The main thread runs client receiver which receives datagram packets from the server-side and adds it to the message queue. The client then reads from the message queue and updates the ClientView(GUI) accordingly. When the user interacts with the GUI, the command is put on a message queue and then is sent by a clientSender to the server-side.

Synchronization server-side: There are synchronization methods inside the MessageQueue class that surround adding a datagram packet to the array of datagrams and removing a datagram from the array. The purpose of these synchronization methods is to ensure that when the server adds a datagram (that it has received from the client-side) it will be added in mutual exclusion.



(Figure 2: Server-side sequence diagram)

The overall program runs the server which takes datagrams from the client-side adds it to the MessageQueue and then the client will send the data to the game model the game model will change and the state of the game then a broadcaster will periodically send the datagrams of the changed state to the client-side.



(Figure 3: Subsystem diagram)