CSCI499 Homework 3

< A Multiplayer Plane Fighter Game >

Viney Khera,
Florian Eddy Robert Ilg

Table of Contents

Overview 1
Usage 1
Example test scenario 2
Comments on Code 3
Architecture 4
Protocol 5

Overview

Given the homework 3 requirements a peer to peer online game was created. The game simulates fighter planes that attack target in enemy territory. Each peer is a friendly. The friendlies communicate about discovered targets, team formations and attack sequences.

The Java implementation contains an additional GUI that displays the battlefield graphically. Targets are displayed red, friendlies green and team captains yellow. A red line indicates a friendly that is focused on a target and yellow lines indicate team member relationships.

The following special data types are used during the game:

- Locations: Coordinates in the form x, y in the range from 0,0 to 800,920.
- Identifiers: IP and port of a peer in the form w.x.y.z:port.

Targets have types and sizes, which are both arbitrary integers. Each peer has an inventory of 5 different types of ammunition. If the ammunition index used is equal to the target type, the target is destroyed, otherwise it will survive.

Usage

The programs all have interactive console UIs and the Java implementation has an additional GUI. They can be compiled and executed as follows:

```
make              # Compiles C++ and Java code
./PPMain          # Starts C++ peer
java PPMain       # Starts Jave peer
```

The peers may optionally be passed 3 parameters. The first is the multicast port, the second the peer port (unicast) and the third the location of the peer in the game.

Example test scenario

As the game has quite many functions, here is a sample test scenario that includes C++ and Java peers and all the 12 message sequences:

```
start peer 1: PPMain 20000 30001 "60,140"
start peer 2: PPMain 20000 30002 "100,100"
start peer 3: java PPMain 20000 30003 "90,30"
start peer 4: java PPMain 20000 30004 "740,50"

peer1: 1                     (register)
peer2: 1                     (register)
peer3: 1                     (register)
peer4: 1                     (register)
peer4: 0                     (display data)
peer4: <Enter>
peer3: 99                    (start GUI)
peer4: 4                     (identify new target)
peer4: 725,350
peer1: 4                     (identify new target)
peer1: 325,230
peer2: 5                     (tell target size)
peer2: 325,230
peer2: 3
peer2: 6                     (tell target type)
peer2: 325,230
peer2: 1

peer4: 5                     (tell target size)
peer4: 725,350
peer4: 1
peer4: 6                     (tell target type)
peer4: 725,350
peer4: 3
peer4: 0                     (display data)
peer4: <Enter>
peer1: 7                     (request help
                            NOTE: all idle peers will offer help)
peer1: 2
peer1: 8                     (make a team)
peer1: 55
peer4: 11                    (update location)
peer4: 770,390
```

```
peer1: 11                  (update location)
peer1: 270,280
peer2: 11                  (update location)
peer1: 270,190
peer3: 11                  (update location)
peer1: 370,220
peer3: 3                   (inform that another was observed dead)
peer3: 127.0.0.1:30004
peer1: 9                   (attack)
peer1: 4
peer1: 10                  (is the target destroyed?
                           team captain asks all members
                           about their observation)
peer2: 12                  (leave team)
peer1: 9                   (attack)
peer1: 3
peer1: 10                  (is target destroyed?)
```

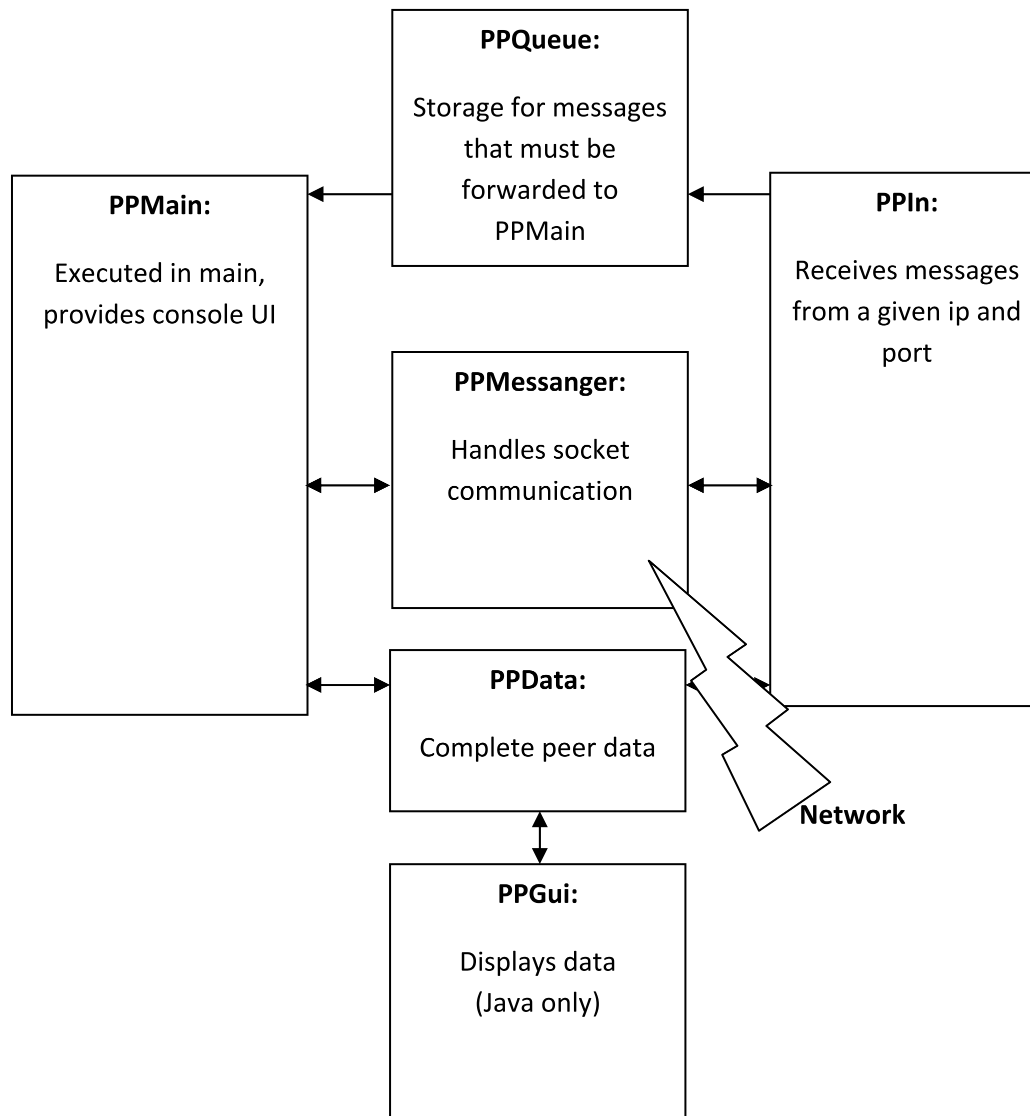
Comments on Code

The application was designed with an object oriented approach and implemented in Java and then a 1:1 port to C++ was performed. For synchronization in C++, a base class called “PPSynchronizer” was implemented. This class “emulates“ the intrinsic object lock of Java, meaning every instance of the derived class has a lock and the following methods:

- lock()
- unlock()
- wait()
- notify()
- notifyall()

These functions correspond to the Java equivalents. A synchronized method must first call lock(), and in EVERY exit branch it must call unlock(), i.e. all before all returns and if an exception is thrown. The C++ code was split into headers and source, whereby not all function bodies were moved to the source files – small functions were left inline (for faster code and programming simplicity).

Architecture



PPMain starts PPIn threads for every interface, it can receive messages on, e.g. 127.0.0.1:30000 (unicast) and 226.2.3.1:20000 (multicast). The PPIn threads then receive messages from the network. If the message can be processed directly and independently of PPMain, it will be processed in PPIn. If the message is a response to another message in a message sequence, it will be added to a PPQueue object, from which it can then be obtained by PPMain. Hence, if PPMain gets a command from the user, writes a message and waits for a response, it actually waits for the response to appear in the queue and the response will be received by PPIn.

PPMain and PPIn are connected to a PPData structure, which contains the peer state. It can be displayed by pressing “0” in the UI. In the java implementation this data may optionally be displayed graphically.

Protocol

Messages must start with the "VEPP" prefix. This is to ensure that the message actually belongs to our application. After that the message will contain at least two items. An item is a 4-byte ASCII length followed by the actual item data (messages can be displayed in the UI by pressing "98"). The first item is the sender ID of the message and the second item is the type of the message. After that message-dependant items may follow.

Overview of message sequences:

Seq. Number	*Type*	*Name*	*Parameters*	*Description*
1	M	Register		Register into group
	U	Hello		Replies from all peers
2	M	Unregister		Unregister from group
3	M	Died	ID	Unregister other peer from group
4	M	Target Identified	Location	Indentify target
5	M	Target Size	Location, Size	Identify target size
6	M	Target Type	Location, Type	Identify target type
7	M	Request Help		Request help for attack
	U	Offer Help		Offer help
	U	Accept		Accept the help
	U	RequestHelpOk		Agree to the acception
8	M	Team	GMID 3 A;B,C	Broadcast team formation; form a subgroup with multicast ID GMID
	U	OK		Only team members confirm
	GM	Ranking	A r1 B r2 C r3	Send team ranking
	U	RankingOK		Agreements for ranking
9	GM	Ready		Ready to attack
	U	Ammo	Ammo-List	Peer's available ammo
	GM	Attack	Location, Ammo-Type	Attack a location with Ammo-Type
	U	AttackOK		Attack confirmation
10	GM	Destroyed	Location	Ask if target is destroyed
	U	Oberservation	Yes / No	Tell observation
	GM	Done		If destroyed, say done
	M	Destroyed	Location	If destroyed, tell everyone
11	M	Update Location	Location	Update peer location
12	U	Leave	GMID	Leave a subgroup

M=Multicast, U=Unicast, GM=Subgroup multicast.

In detail the message sequences are as follows:

1) **Register into group**
   This is a multicast message. It allows a fighter plane peer to register into the group. When a peer registers into the group, all the other peers add the ID (i.e. IP:port) of the new peer in their friendly data table. Also, the new peer gets the IDs of all the existing peers as they reply to this message with hello.

2) **Unregister from group**
   It allows a fighter plane peer to unregister from the group. In this case every peer deletes the ID of this peer from their friendly table so that they won't send any further messages to the unregistered peer. This is a multicast message.

3) **Inform that a peer died**
   A peer can inform that a particular peer was observed dead (whose ID was IP:port). So, all other peers will remove this peer from their friendly data table. This is a multicast message.

4) **Inform about an identified target**
   A peer can inform other peers through a multicast message that it has detected a new target. It tells the location of new target to every other peer in the group. Note that if a peer registers later than the target is identified, it will not know about the target.

5) **Inform about target size**
   A peer can inform other peers through a multicast message about the observed target size.

6) **Inform about target type**
   A peer can inform other peers through a multicast message about the observed type of the target.

7) **Request help**
   This is the first four message sequence. A peer can request other peers to help attack a target as a team. It can take help from 1 to 3 peers depending on the size of the target. It asks for help to all other peers through a multicast message.

   - Other peers offer help through a offer-help message, unicast to the sender of the request help message, i.e. the captain of the new team.
   - The captain accepts help from some of the peers who responded first by sending a unicast message to them.
   - Peers which receive accept help messages will respond to the sender by request-hep-ok messages. This is a unicast message.

   Here, the captain or requester has formed a team.

8) **Making new team**
   - The captain multicasts to everyone that it has formed a team.
   - Team members reply by sending the unicast message teamOk, to confirm their availability.
   - Captain decides and sends the ranking of the team to the team members.

- Team members reply by the unicast rankingOk message.

9) Deciding ammunition to be used

- The captain sends a multicast message to the team members that he is ready to attack.
- Team members reply with their ammunition information to captain.
- The captain decides on the ammunition to be used for the attack and informs about the same to all the team members.
- The team members respond with an attackOk message.

10) Is target destroyed?

- Captain asks the team members if the target is destroyed by sending destroyed message multicasted to the team.
- The team members reply with their observation to the captain.
- The captain checks general agreement. If the target is destroyed, the captain sends done message to the team and a targetDestroyed message to the whole group.

11) Update location

It is a multicast message to inform about a location change.

12) Leave the team

It is a unicast message to the team captain, if a peer decides to leave the team.

On the User Menu

0) Display current peer data

Displays peer data

99) Show GUI

Displays graphics for peer fighter plane and targets.

100) Exit

Exits the program