ECE419S: Distributed Systems

Winter 2016

Assignment 2

Submission deadline: Tuesday, February 2nd 11:59 PM

1 Introduction

In the first assignment, you were introduced to basic techniques for building distributed systems. You implemented a simple Broker system providing stock quotes to users. In the next two related assignments, you are to design and implement distributed versions for a multiplayer game, Mazewar. The goals are twofold. First, you will need to understand a given codebase and extend it by adding network communication support and making it distributed – Assignment 2. Second, now that you have more experience in the tools necessary for building distributed systems, as well as the knowledge of the concepts from the lectures, you will design and implement a distributed system on your own. That is, you will have the freedom to reason about the design of a distributed system, to pick the design choices that you think are best and to produce a specification and implementation showing your creativity – Assignment 3.

1.1 Assignment Requirements

In this assignment, you are to implement a client-server version of the multiplayer game, Mazewar, that is described below. You have been provided with a version of Mazewar where you can play against simple computer controlled opponents. This way you can concentrate on designing and implementing the client-server protocol rather than the user interface and other framework issues.

You are required to use the *ug* computers in the GB 243 lab to implement and run your game in a client-server fashion – i.e., playing against your team member or other classmates and relying on a central server to aid with communication and managing game events/state. You need basic knowledge of Java to implement the assignment requirements.

The code and starter files are placed on the *ug* machines, at `/cad2/ece419s/` referred as `${ECE419_HOME}`. The source code for `MAZEWAR` is under `${ECE419_HOME}/labs/lab2/mazewar/`. We provide a basic `Makefile` that compiles the single-player version of `MAZEWAR`. A partial solution is provided in the subdirectory partial. Please start with this partial solution code and update this solution as necessary, but without modifying its socket code.

NOTE: If you plan to run the code on your home machine, you will have to update the path to Java (i.e. `${ECE419_HOME}` and `${JAVA_HOME}`).

1.2 How to Work on the Lab

Both Assignment 2 and Assignment 3 consist of two parts: design and implementation. For this assignment alone, you are required to implement a client-server solution following the design given below. For Assignment 3, you will fully design and implement a decentralized version of the distributed game.

Students should form both *design teams* and *programming teams*. A *programming team* should have *two people*, similar to Assignment 1 – there will be no exceptions. A *design team* should consist of up to 3 programming teams i.e., up to 6 people. In the worst case, you can have the *design team* be the same as the *programming team*.

The *design teams* are for the purpose of discussing the given codebase, designing and specifying a protocol for the game, not for jointly writing code. You may not share code with other *programming teams*. For this assignment, *design team* members can discuss the codebase and the given design. For the next assignment, all members of a *design team* can devise and implement a common design.

I suggest that you stick to the rules provided in the Mazewar game description, but you are free to make any additions that you think improve the game, as long as you document them clearly.

Your game should not allow inconsistent states to be seen by different players. Specifically, all player moves should be displayed in a consistent order on all player displays. Assuming there are only two players, e.g., the players of your team, you should not allow that the display of one player shows that the local player has made a move before its opponent, while the other display shows the reverse.

2 MAZEWAR Game Description

You are given a template for a simple multi-player game called MAZEWAR. MAZEWAR is a distributed, multiplayer game that allows each player to control a client in a maze. A player receives points for vaporizing other clients with a projectile and loses points for being vaporized. The game is based on the X Window System version of Mazewar, which is in turn based on the classic game of Mazewar that ran on the Xerox Alto in the 1970s.

When you first start Mazewar, you will be presented with a dialog asking you your name. After that, a game window will be displayed.

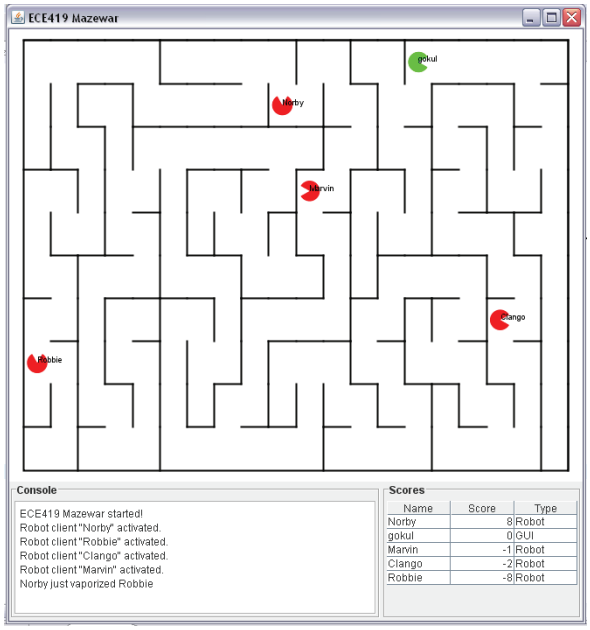


Figure 1: Mazewar Screenshot

There are three parts to the window (shown in Figure 1):

- A view of the maze showing the position and orientation of the clients from above.
- A scoreboard, listing the score of all the clients connected.
- A console, where debugging and other status information may be printed.

The **green** client is the local GUI controlled client. Robot clients are **red**, and remote clients are **magenta**. The user interface operates as follows:

Key	Action
Left-arrow	Turn 90 degrees counter-clockwise
Right-arrow	Turn 90 degrees-clockwise
Up-arrow	Move forward
Down-arrow	Move backward
Spacebar	Fire
q	Quit

Table 1: Player actions

2.1 Rules of the Game

The rules of the game are as follows.

- The maze consists of a 20x10 array of cells. Each cell is either occupied by a client, projectile, or is empty. You may adjust the size and layout the maze if you so desire, but all clients playing together during a given session must use the same maze for consistency purposes.
- Projectiles have a finite speed, advancing one cell every 200 milliseconds. You may adjust this if you desire, however, again for consistency purposes all clients must behave the same in a given session.
- A client needs some amount of time to reload after launching a projectile. During this time it can do anything except for launch another projectile.
- A client hit by an opponent's projectile is vaporized instantly. Projectiles/missiles are the full width of the corridor, so that if a projectile passes through any cell, it certainly hits the client in that cell.
- A client that is hit loses 5 points. A client that successfully hits another gains 11 points. Firing a projectile costs 1 point.
- When a client is hit, it reappears at a random position (in an unoccupied cell) with a random orientation (though not facing a wall) in the maze.

2.2 Game Design

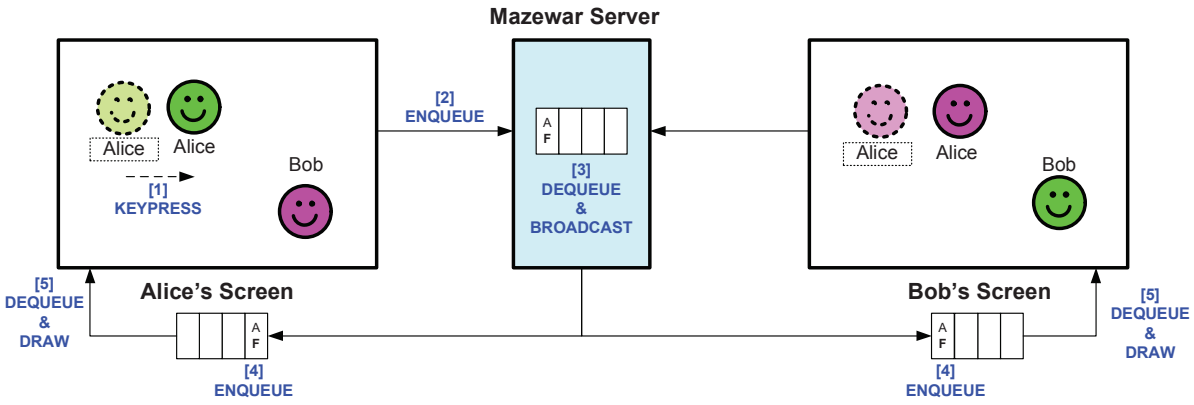


Figure 2: A Walkthrough

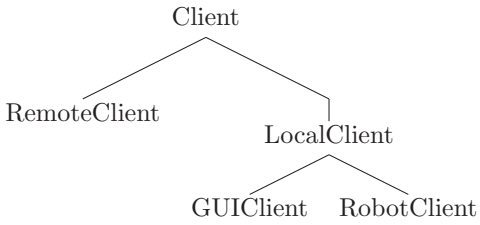
Figure 2 goes through an example of what types of messages are exchanged during the game in the client-server architecture. It shows a game played between two players, *Alice* and *Bob*. Following Mazewar's specifications, on Alice's screen, Alice is represented in **green** since she is the local player and Bob is represented in **magenta**. Similarly, on Bob's screen, Bob is represented in **green** and Alice is represented in **magenta**. The design includes the central server **Mazewar Server** (`MazewarServer`) which sequences the actions taken by Alice and Bob.

Figure 2 shows an example of what actions are taken to notify the other player of the move and how to coordinate the messages so both players see a consistent game window. To clarify, if two windows are consistent, then they should *look the same*. Suppose, during the game, Alice moves *forward*.

1. A notification is generated when Alice presses the up-arrow (↑).
2. At this point, a message is sent to the `MazewarServer` indicating Alice moved forward. The server enqueues Alice's request in a local queue. This establishes a global order of player actions since the enqueue operation occuring at the server is atomic (even if there are multiple threads at the server to handle connections). The server tags each action with a sequence number.
3. `MazewarServer` then dequeues an operation and broadcasts to all clients.
4. Upon receiving a message from `MazewarServer`, all clients enqueue the action in their local queue based on the sequence number, such that lower sequence number messages are at the front of the queue.
5. Each client dequeues the action message at the front of the queue if it is the next one in sequence i.e., there are no gaps in the order of sequence numbers for messages, and displays the action by moving the client icon forward accordingly.

NOTE: In this assignment, you should implement the design described above.

2.3 Description of the Class Structure in Mazewar



Now that you have seen a high level description of the actions taken by the game, we also guide you through the class structure to aid you in your implementation. We present the class structure of `MAZEWAR`. The players in the maze are instances of the `Client` class. The `Client` class has methods to implement all of the actions defined in Table 1. The methods have meaningful names and it is easy to understand what action they implement. For example, the method `forward()` implements the code to move the client forward in the maze. Other methods are similar.

The `Client` class is extended by both `RemoteClient` and `Localclient`. There is one instance of `LocalClient` (specifically `GUIClient`) representing the local player. The different keypresses are handled by `GUIClient` which in turn invokes methods in `LocalClient`. So, In Figure 2, when Alice pressed the up-arrow key (↑), `GUIClient` generates an event saying "Alice pressed the Up-Arrow key". This subsequently calls the method `forward()` in `LocalClient`. The `LocalClient` then generates a message indicating "Alice moved forward" and sends it to the `MazewarServer` which is then broadcast to all clients.

Similarly, each instance of `RemoteClient` represents a player playing from a remote machine. Naturally, this means that any action taken by the remote player will invoke a method in this class. In Figure 2, on Bob's screen Alice is represented as a `RemoteClient` object. Thus, when the message from `MazewarServer` arrives to Bob saying "Alice moved forward", the `forward()` method is invoked in the `RemoteClient` object representing Alice.

2.4 Files

In the ECE419 directory (`${ECE419_HOME}`), we have provided some relevant files to you. These include a sample Makefile, the single player code and the partial solution for `MAZEWAR (inside subdirectory partial)`. You need to start with and use the Java socket code provided with the partial solution (you are not allowed to modify the socket code), but may create new files and modify the given files in order to submit your complete solution. Also make sure that you update the Makefile such that everything compiles properly.

3 Grading

This lab assignment represents 6% of your final grade for the course and is due by Tuesday February 2nd 2016 at 11:59 PM.

You must implement your solution to:

1. Be a *consistent* game
2. Be playable, i.e. the game must respond quickly to player moves
3. Handle up to 4 players

The lab will be graded according to the following scheme:

- Full credit for an implementation having all components working according to the objectives above.
- 80% if your game is playable, but has a minor functional or logical mistake, e.g., inconsistencies or visible delays.
- `20-`40% if your program is partly working but has several or major functional or logical mistakes.

4 Submission

Only one team member needs to submit the assignment. Include the names and student numbers for both team members in a README file. Your submission should consist of the source files, a `Makefile`, and the `README`. Feel free to use/update the sample Makefile provided to you, or write your own Makefile.

Your submission must be in the form of one compressed archive named `Lastname1.Lastname2.tar.gz`. The directory structure of the archive should be as follows:

```
Lastname1.Lastname2.Lab2/
   README (short description of how to run your code and other aspects you feel worth mentioning)
   Makefile
   <Java source files>
```

If you need to provide additional explanations about your solution, please do so in the general README file. Once you have a compressed archive in the `.tar.gz` format, you may submit your solution by the deadline using the submitece419s command, located under /local/bin on the *ug* machines:

```
/local/bin/submitece419s 2 Lastname1.Lastname2.tar.gz
```