ECE419S: Distributed Systems

Winter 2016

Assignment 3

Submission deadline: **Thursday** March 3rd 2016 11:59 PM

1 Introduction

In the second assignment, you implemented a client-server version of the multiplayer game, Mazewar. You were provided with the single player version of Mazewar where you could play against simple computer controlled opponents, and you were asked to extend it by adding network communication support and making it distributed according to the specified client-server design.

Now that you have a better understanding of the game codebase, 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 decentralized distributed game system on your own. That is, you will have the freedom to reason about the design of a decentralized distributed game solution, to pick the design choices that you think are best and to produce a specification and implementation showing your creativity.

2 Assignment Requirements

In this assignment, you are to *design and implement* a decentralized distributed version of the multiplayer game, Mazewar. Unlike Assignment 2, all processes in this distributed version will be more or less identical. That is, there will be no central server that receives events from any client/player process and broadcasts them back to all processes in the system.

You are required to use the *ug* computers in the GB 243 lab to implement and run your game in a distributed fashion, e.g., by playing against your team member or other classmates.

The source code for the `MAZEWAR` single player version that was provided to you for Assignment 2 is placed on the *ug* machines, under `${ECE419_HOME}/labs/lab2/mazewar/`. `${ECE419_HOME}` is `/cad2/ece419s/`.

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}`).

2.1 How to Work on the Lab

This assignment consist of two parts: design and implementation. You are required to fully design and implement a decentralized version of the distributed game.

At this point, you should be part of a *programming team*. The *programming team* should have *two people*. In addition, your *programming team* should be part of a *design team*. A *design team* should consist of up to 3 programming teams i.e., up to 6 people. In the worst case, the *design team* can be the same as the *programming team*.

The *design teams* are for the purpose of discussing the given code base, 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, all members of a *design team* should interact, discuss design choices, tradeoffs, and devise a common design to be implemented by each *programming team* separately.

Each design team should produce a final design document of about 2, 3 pages, where you describe your design choices and answer the questions in the Design section below. All members of a design team can discuss a common design, have the same design choices and produce the same design document. A preliminary version of this design document needs to be handed in on February 23rd in class ! This is just to make sure that you are done with the design by then. You need to optimize your design for scalability to 1000's of players in one of the following two hypothetical game environments (choose one of the following and design for that platform):

1. high network bandwidth, high parallelism in network (many routers), high network latency, high-end nodes

   or

2. high network bandwidth, very low parallelism in network, very low network latency, low-end nodes.

Each *programming team* should implement the distributed game independently based on the design. The final version of the design document, `design.pdf`, should be submitted together with the code by each *programming* *team,* by the assignment deadline. In the design document, include the names and student IDs of the submitting programming team members (as well as the names and student IDs of the larger design team members).

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.

You can choose any algorithm introduced in class to implement a total ordering of events across all processes in your distributed system or a new algorithm of your own design. Furthermore, you can use any distributed system structure including, but not limited to, organizations of the distributed system in a ring, where each node communicates only with a predefined neighbour or a symmetric structure where a node communicates with all other nodes. You can also assign special roles to processes in your distributed system, as long as the role functionality is minimal and lightweight.

2.2 Design

Your design team is to jointly design the specification of the protocol used by the game components as well as the overall distributed algorithm. The design description should be thorough enough so that all programming units of the design team can independently implement a Mazewar program to the specification and, if needed, have it interoperate with each of the other programs.

The description should be submitted in the `design.pdf` document along with the code. The description should have the information in an easy to understand format such that someone else can duplicate your design.

For instance, the design description should include:

- Overall distributed algorithm functionality, components, interactions
- Communication protocol – packet types and exchanges
- How a player locates, joins and leaves a game of Mazewar
- If you handle *player node* failures: how is this handled ?
- Timings of protocol events, and how they provide sufficient consistency for the game state
- Pseudo-code or an abstract description of process behaviour
- Anything else you think of that's relevant to your system

Questions to be Answered in the Design Document

Answer each of the following questions in the final `design.pdf` document. Point form answers are encouraged.

- Evaluate the portion of your design that deals with starting, maintaining, and exiting a game – what are its strengths and weaknesses?
- Evaluate your design with respect to its performance on the current platform (i.e. *ug* machines in a small LAN). If applicable, you can use the robot clients in Mazewar to measure the number of packets and packet sizes sent for various time intervals and number of players. Analyze your results.
- How does your current design scale for an increased number of players for the type of environment you chose? You can give examples for environments that might fit the hypothetical scenario you chose e.g., game is played on desktops or mobile devices, wired/wireless. Use Big Oh notation in your performance analysis.
- Evaluate your design for consistency. What inconsistencies can occur? How are they dealt with?

3 Grading

This lab assignment represents 12% of your final grade for the course and is due by Thursday March 3rd 2016 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, and handle up to 4 players
3. Handle packet drops in the network in addition to delays and reorderings
4. Optional: Handle `RobotClients` (5% bonus)
5. Optional: Dynamic behaviour – dynamic joins&departures of players (5% bonus)
6. Optional: Fault tolerance for node failures (5% bonus)

You also need to provide a *live demonstration* of your distributed game during a Lab session (date to be determined). The lab will be graded according to the following scheme:

- 20% for the design document based on the criteria described below (include final design doc in e-submission)
- Full credit (80% of total assignment grade) for an implementation having all components working according to the objectives – game is decentralized, consistent, fast/playable.
- 50% if your game is playable, but has a minor functional or logical mistake, e.g., inconsistencies or visible delays.
- 20% if your program is partly working but has several or major functional or logical mistakes.
- Otherwise, 10% if the code compiles successfully. To receive this credit, your code has to show "reasonable effort" towards the objective of the lab part and any compilable code will not award you this credit :)
- If you do not hand in a preliminary version of your Design document by Tuesday, February 23rd, you lose 10%.

The design document will be graded for the design team as a whole (i.e., a single grade will be given to all members of the design team), based on the following criteria:

- **Correctness.** Does the design support a distributed game of Mazewar that conforms to the specification in the game description?
- **Efficiency.** Does the protocol avoid unnecessary overhead and optimize for the network bottleneck and/or any limits on the processing load on each node? How? Does your design introduce any potential bottlenecks, such as a single coordinator that handles complex tasks? A 15% penalty will be incurred for the use of any centralized component, except for the use of a naming server, which is allowed just for group membership maintenance & connection set-up. Generic optimization is less important than fit to environment chosen.
- **Analysis.** Argue that your solution is a good fit for scaling to 1000's of players in the chosen environment. Include message complexity (Big Oh) and/or actual statistics in your discussion, as appropriate.

4 Submission

Only one team member needs to submit the assignment. Include the names and student numbers for your programming team members in a `README` file. Your submission should consist of the `design.pdf` document, source files, `Makefile`, and the `README`.

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.Lab3/
   design.pdf (don't forget to include names, student IDs for all your design team members)
   README (short description of how to run your code and other aspects you feel worth mentioning)
   Makefile
   <Java source files>
   scripts
```

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 3 Lastname1.Lastname2.tar.gz
```