# ****Designing and Implementing A Peer-to-Peer Distributed Maze Game******

## 1.1 **Overview**

You are to implement a distributed maze game. The maze is an N-by-N grid, consisting of N*N "cells". Each cell contains either no "treasure" or exactly one "treasure". At the start of the game, there are K treasures placed randomly at the grid. The number of players is arbitrary. Each player can be viewed as running on a separate node in the distributed system. Each player aims to collect as many treasures as possible. A player collects one or more treasures when the player's location is the same as the treasure's location. A player cannot occupy a cell that another player occupies. Each player has a current score which equals to the number of treasures the player has collected. The number of remaining treasures in the game will always be K — this means that when a treasure is taken by a player, the game should automatically create a new treasure at some random location. We want the system to be fault-tolerant. Namely, we want the system to be robust against player crashes.

## 1.2 **The Tracker Program**

For implementing this distributed game, you should first design and implement a program called the Tracker. The tracker will have a well-known IP address and port number (i.e., known to all people who want to play the game). Furthermore, the tracker knows the values of N and K. 1Whenever a new player wants to join the game, the new player should contact the tracker first (at the well-known IP address and port number). This allows the new player to obtain information about the existing players (e.g., their IP addresses and port numbers) in the game, as well as N and K. If you are familiar with BitTorrent, our tracker here provides a similar functionality to a BitTorrent tracker. It is up to you what information a new player gets from the tracker. For example, the tracker can return the entire list of current players in the game. Or the tracker can return a random current player in the game. After the new player gets such information from the tracker, the new player will contact the existing player(s) (based on such information) to actually join the game. Note that since we allow player crashes, it is possible that the information returned from the tracker becomes stale by the time the new player tries to use it. So you should design your program carefully to deal with such cases. It is important to note that a player is allowed to contact the tracker only when i) it first joins the game and ii) some player crashes and the system is trying to regenerate the game servers (see later). A player is not allowed to contact the tracker during its normal gameplay (e.g., when doing moves). The tracker should not maintain any information about the current game state (such as the location of the players in the maze and which player is the primary/backup server), other than the list of current players, N, and K. The reason is that the tracker is a single node, and hence a good design should always minimize the load on the tracker.

Your tracker program should be invoked in the following way:

• If you are using Java, you should have a "Tracker.class" in the working directory after compilation. Then

you should run:
java Tracker [port-number] [N] [K]

• If you are using C, please name your binary file as "Tracker.exe" after compilation. Then you should run:
Tracker.exe [port-number] [N] [K]
Here port-number is the port over which the Tracker is listening, while N and K are the parameters for the maze.

The (implicit) IP address will be the local machine's IP address on which the tracker runs.

1.3 **The Game Program**

You should design and implement a program called Game, which will be the main program in this assignment. A user should be able to execute Game (from any node/computer) to join the maze game as a new player. As explained earlier, this Game program should contact the tracker as a bootstrapping point. Each player has a name, which has exactly two characters. The player should join the maze game in the following way:

• If you are using Java, you should have a "Game.class" in the working directory after compilation. Then you
should run:
java Game [IP-address] [port-number] [player-id]

• If you are using C, please name your binary file as "Game.exe" after compilation. Then you should run:
Game.exe [IP-address] [port-number] [player-id]
Here IP-address is the well-known IP address of the Tracker and port-number is the port over which the Tracker is
listening. The player-id is the two-character name of the player.
The Game program should have a GUI, showing:

• The name of the local player (should be shown at the top bar of the window, i.e., the title of the window).

• The current score of all the players (should be shown at the left part of the window).

2• The maze as a grid (should be shown at the right part of the window).

• Each treasure in the current maze should be shown as a "*" on the grid.

• The current positions of all players (including both the local player and all other players) in the grid, by indicating in the respective cells the players' two-character names.

• The name of the player currently acting as the primary server and the name of the player currently acting as the backup server. (See the definitions of the primary server and the backup server later.)
These should be indicated in the left part of the window (together with the scores of all the players).
Additional eye candies in your GUI are allowed but will not earn you extra credits, since the focus of this assignment is on distributed systems.

The Game program should read text lines from standard input:

• If the line equals "0", then the player does not move but refreshes its local state (i.e., so that its local state
contains the most up-to-date game state information, including the scores of all the players, the positions of
all the treasures, as well as the positions of all the players).

• If the line equals "1", then the player moves West by one step and refreshes its local state.

• If the line equals “2”, then the player moves South by one step and refreshes its local state.

• If the line equals “3”, then the player moves East by one step and refreshes its local state.

• If the line equals “4”, then the player moves North by one step and refreshes its local state.

• If the line equals “9”, then the player exits, and all other players should delete this player from the game (so
that this player is no longer shown in their GUIs).

• For all other cases, the program does nothing.
The grid boundaries are solid, e.g. a player cannot move south if he/she is already at the south-most row. Players should be able to move in an asynchronous fashion. For example, it should be possible for one player to move five steps when another player only makes one move. How fast a player moves should only be constrained by how fast the user inputs the directions.

1.4 **The Game State**
The game state includes the current positions of all the treasures and the current positions/scores of all the players. Since we are designing a peer-to-peer distributed game, there will be no servers storing the game state. Rather, the players themselves have to do the job and maintain the game state. There are compelling commercial reasons to get rid of the server – if your company can support distributed games without maintaining servers, it means that your company can collect revenue while incurring minimum operational cost. To do a peer-to-peer game, among all the players, one player should act as the primary server (in addition to being a player itself), and another player should act as the backup server (in addition to being a player itself). The notions of primary server and backup server are logical here, since there are actually no dedicated game servers. When the game initially starts, whoever joins first should be the primary server, and whoever joins second should be the backup server. Both the primary server and the backup server should maintain up-to-date game state. When a player moves, it is allowed to communication with the primary server and/or the backup server to update the game state as well 3as retrieve the latest game state. The primary server and the backup server may communicate with each other as well if you would like them to. However, the primary server and the backup server should not communicate with other players (i.e., other than the player making the move request). In particular, they are not allowed to immediately broadcast the updated state to the other players. The updated game state should be sent to the other players only when they make their moves. In other words, the other players will often see a game state that is slightly stale. This requirement serves to make sure that your design is realistic – in the real world for large-scale games, immediately updating all players incurs prohibitive overhead. Your primary server and backup server are also responsible for creating new treasures when existing treasures are collected by the players. Recall that the total number of remaining treasures in the maze game should always be K.