Extreme Multiprogramming 2013/14 Assignment 2 (programming part)

Implementing xmpMUD

The second assignment for XMP is to write a *very rudimentary* Multi-User Dungeon game. You can read up on this old fashioned game style here:

`http://en.wikipedia.org/wiki/MUD`

In a MUD game there can be multiple users and there are autonomous program agents. The purpose of this assignment is of course to test your ability to design and implement a concurrent design, not to test your skills as a game writer!

A MUD game consists of a number of rooms a player may move in between, through designated doors. In a room there may be objects that can be picked up, similarly a player can leave objects. A real MUD game will also have an action component that allows fighting and dying, this is not required in this assignment!

Thus your game must include:

- A few rooms
- Doors the player can pass through to move from one room to another
- A few objects that may be picked up and left again
- At least one autonomous agent that moves between the rooms as a programmed agent – it need not do anything but move around

A player should be able to issue the following commands

- 'look' – return a description of the current room, what is in the room, who is in the room and a list of items that the player is holding
- 'take' – should pick up an object in the room, i.e. take gold should pickup the gold in the room and the player now holds the gold (this is assuming there is any gold in the room)
- 'leave' – should allow any object the player is currently holding to be left it in the room (this is the inverse of the 'take' operation)
- 'exit' – should end the game and terminate all processes
- 'N' – should move the player through a door to the north if such a door exist
- 'E' – should move the player through a door to the east if such a door exist
- 'S' – should move the player through a door to the south if such a door exist
- 'W' – should move the player through a door to the west if such a door exist

A room in the game should:

- Be able to hold objects
- Be able to host more than one player or agent at the same time

- When a player or agent enters or leaves the room all other players and agents in the room should be informed to the fact
- A room may never be blocked for an extended time, i.e. the room may execute any of the supported player commands but not block while waiting for another user or agent to act

Your report must include

- A graphical CSP-network design of your solution
- Consideration on the potential concurrency in your system
- An explanation of how your network is shut-down when the user issues the exit command, address all processes!

In all the report on this portion may not exceed two pages, excluding source-code but including a graphical representation of your network.

Warning

Please remember that this is an exercise in concurrency design and programming, not game design or story telling. This means that you should not go over board and spend too much time on the game part, spend your time considering how the concurrency issues are best handled.

While this assignment looks very simple it does have two challenges; a room must always be reactive in final time, i.e. a player or agent cannot be blocked until another player or agent acts, secondly, termination of the overall game may be quite complex, depending on your design. So address the good design and consider the pitfalls before you spend time playing your game.

If you find that there are any ambiguities or missing information in the description of the assignment you have two options: either ask on the Absalon forum or make your own assumptions, which you then specify in the report. If you make your own assumptions you should not make any such assumptions that will make the assignment trivial or in any way hinder the primary objective of this assignment, which is to test your ability in concurrency design and implementations.