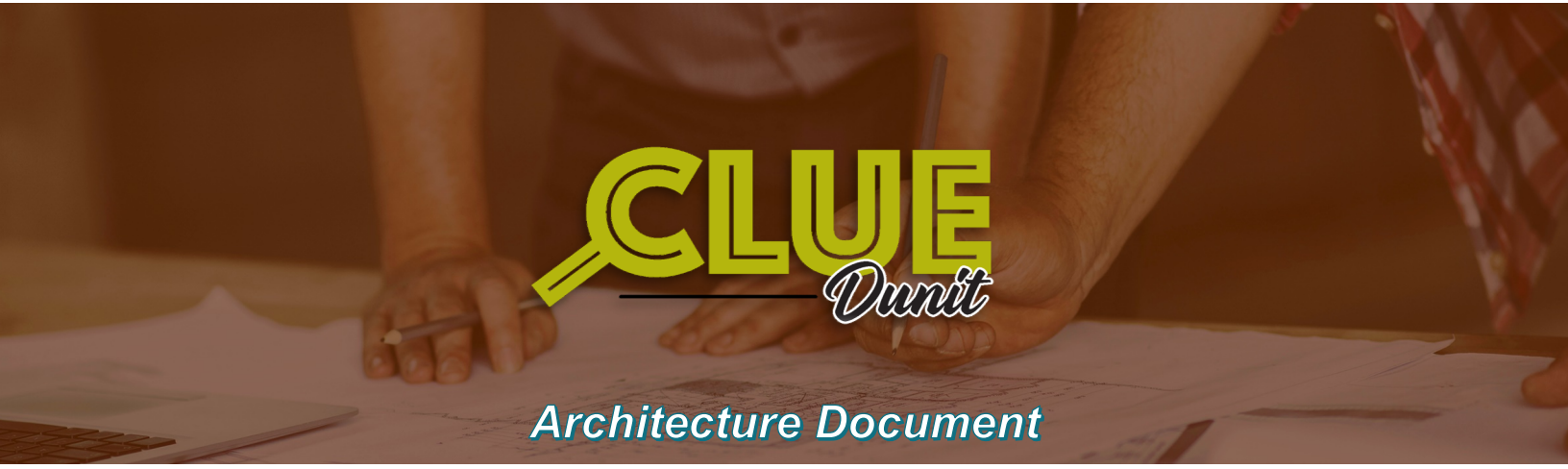

Authors: **George, Linh, Troy, Kerim, Rasa**

Revision History

Date	Version	Description	Author
3/23/2020	Version 1.0	Initial release	Clue-Dunit Team

Supplementary Specification

1. Related Documents:
    - Supplementary Specification for Functional Requirements

From requirement G.1 Clue-less shall be implemented as a client server game, the Clue-less game system will consist of both a client and a server, as shown in Figure 1. As can be seen from Figure 1, the player will interact with the client, and the client will manage all communications to and from the server.

1. Server Architecture

The server requirements are detailed in Supplementary Specification for Functional Requirements. Due to the requirements S.2, S.3, S.4, and S.5 the server will need to have three modules. The first module, the **listModule**

will be responsible for listening to incoming connects, accepting the and then enqueueing the connection.

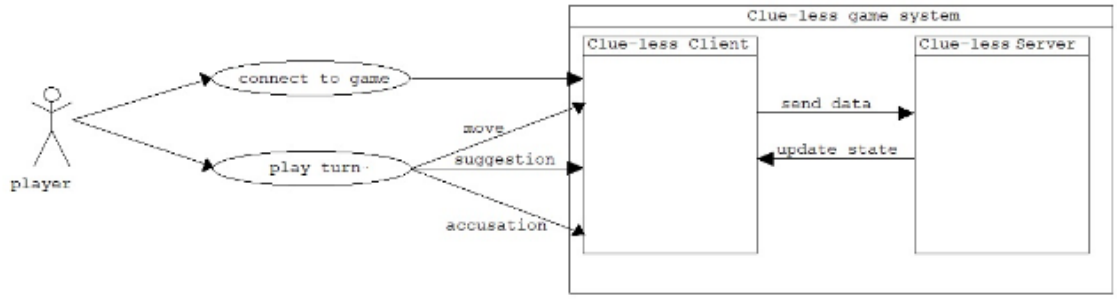

Figure 1: Top level use-case diagram for the Clue-less game system

The second module, **manModule** is responsible for managing the queue and spawning a game when at least the appropriate number players are in the queue. Finally, the **gameModule** is responsible for implementing the game logic expressed in Section 1.1. The sequence of interactions in the server are shown in Figure 2.

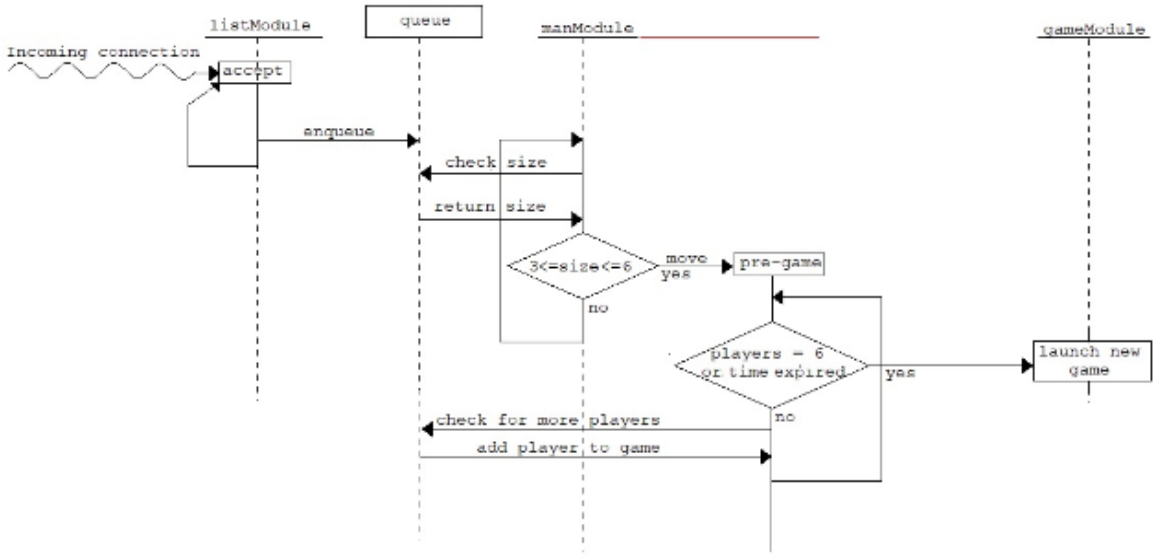

Figure 2: Sequence diagram for the Clue-less server

In the listening module, the main server socket is created, and it is marked as passive and then accept is called. This function will block till a connect attempt is made. The server accepts the connect and enqueues it and then return back to the accept function. Assuming that this module is implemented in its own thread it will be ensured that the server will remain responsive, thus meeting requirement S.2.

The next module **manModule** will be responsible for managing the player queue, and will periodically check the size of the queue, and if the is at least three players in the queue, move at most six players into the pre-game holding area and start a countdown timer as described in Requirement S.5. The manager module will the monitor if any new connections have been made, and add those connections to the pre-game staging area, ensuring that the number of players does not exceed 6, as per Requirement G.2. Once there are the maximum number of players, or the time expires the manager will have those connects of to the game module and a new game will start. At this time communications between the server and the clients will be managed by the game module.

The final module, the **gameModule** will implement the rule of the game expressed in Section 1.1. The first thing this module will do is prepare the case file as described in Requirements G.8 through G.17c. Then the remaining cards will be distributed as described. As expressed in Requirement G.3, play will start with the player playing Miss Scarlet.

1.1. Implementation Details

The server will be implemented in C++, using the standard Berkeley Socket API. We shall provide logging capability to a default or user-configured log file. It is expected that the server will execute without significant care and feeding. To support testing, we will provide a minimal command line interface.

2. Client Architecture

The client will have two components, the GUI component which will display information to the player, as well as accepting input from the player. The other, module is a communication module that manages all communications with the server.

The GUI component will display the user interface shown in Figure 3. The interface will support the game map (shown in the center), the investigators notebook (left side), the players cards (right side), and the game message (at the bottom). Also, note that the interface will have s status bar to display status messages. In this case we can see that the client is connected to the game server located at 192.168.1.19. The card display will have the able to display a maximum of six cards, which is the most that a player could receive if only three people were playing in the game.

The communication module will manage all communications between the client and the server. Because we expect that the communication module will reside in separate thread, information will need to be passed between the main, GUI thread and the communication thread. See section 2.1 for information on how we intend to do this.

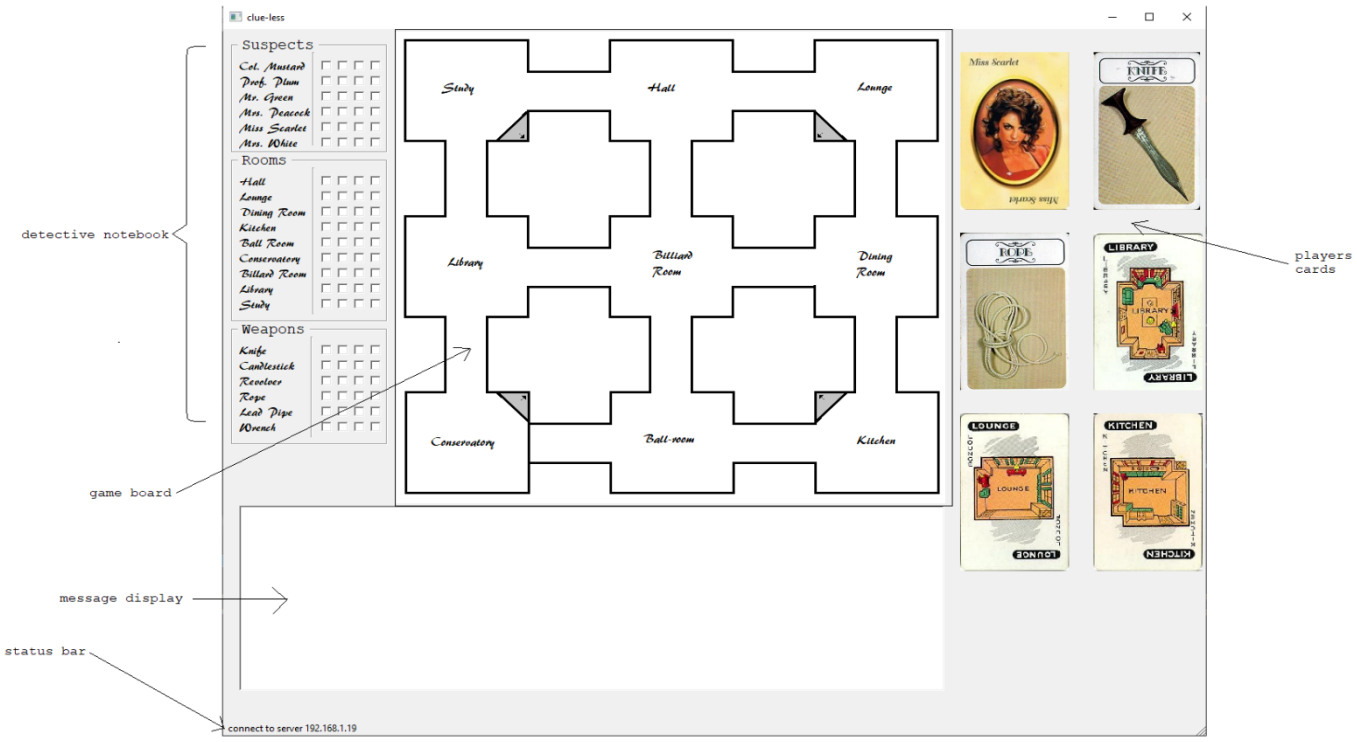

Figure 3: Clue-less GUI interface

2.1. Implementation Details

The client will be implemented in C++, using the QT library for the graphical presentation. By using QT as the application framework, we will be locked into their *signal/slot* mechanism. However, is actually an advantage in that QT treats threads as first-class citizens meaning that threads (if derived from QThread) will have their own event loop, being that they can sources of, or sink of *signals*. Thus, communication between these two modules will be handled by QT's *signal/slot* mechanism.