**Class Descriptions**

Figure 4.1.1 depicts a representation of how our classes will be structured when we implement our design. This is a mock design and can undergo changes based on requirements or limitations encountered in the process of our implementation.

The major classes included in our code would resemble the following descriptions:-

1. Client
   1. Each player in a game is imagined to be a separate client and interacts with the server independently.
   2. Contains data teamNo (which team the player belongs to) and clientType (what role does the player play in the team)
   3. Can send commands to the server for a change in the position or system state of the spaceship.
   4. Can send messages to the server to send it to his other team members.
   5. Can get messages of fellow team members from the server.
   6. Is accessed by the player through the GUI
2. GUI
   1. Is the visual representation for the Client class
   2. Consists of four sub-classes: Captain, Engineer, First Officer, Radio Officer
   3. Can display Login window on the start of the game
   4. Can ask to start game
   5. Can ask to select role
   6. Can display recursively updated interface for the selected role
3. Server
   1. Main controller where all the actions would take place
   2. Can get commands from the clients
   3. Can update current Maps object based on the command
   4. Can update current Spaceships object based on the command
   5. Can get messages from a player and send it to all his teammates
   6. Can access and update database of user accounts
4. Spaceships
   1. Represents unique spaceships of each team
   2. Contains data shipPosition (where is the ship on the map), shipPath (string of letters based on the path the ship has taken), shipHealth (what is the health of the ship, i.e. from 1 to 4), shipSystems (object of class ShipSystems)
   3. Can initialize object when the game is started
   4. Can get the next direction from the player based on the command
   5. Can update the path of the ship based on recent movement
   6. Can return all the data of the ship to the client
5. Maps
   1. Represents all the types of maps the players can play on
   2. Contains data asteroidPosition (array of positions where all the asteroids are present) and minesPosition (where are the mines been placed)
   3. Can calculate crash based on the movement
   4. Can calculate all the positions mine blast would affect
   5. Can calculate all the positions missile blast would affect
6. ShipSystems
   1. Represents all the systems in a spaceship
   2. Contains data missile (missile system), mine (mine system), sonar (sonar system), drone (drone system), silence (silence system)
   3. Can return the status of each system
   4. Can update the status of each system
7. Database
   1. Used to store all the user account data
   2. Contains data userId (which player is playing), password (password to login), numGames (no of games played by the user), wins (no. of games won by the user) and avatar (picture chosen by user)
   3. Can initialize a new user
   4. Can check the password entered for login
   5. Can update password after login
   6. Can see statistics of player
   7. Can update new avatar from a list of choices

**Class Relations**

From the above Class Diagram, we can say that the classes are related to each other in the following ways:-

1. Client is related to GUI in a one to one cardinality. This means that each client will have only one GUI representing it and it would be based on the clientType. GUI displays the data present in client in a graphical manner
2. GUI is inherited by four classes – CaptainGUI, EngineerGUI, FirstOfficerGUI, RadioOfficerGUI. These four classes contain components of GUI class but also have some of their own implementations due to which they have different displays.
3. Client is related to Server in a 2..\* to 1 cardinality. This means that there is only one sever but the server can have clients from 2 to n. Client interacts with Server by sending commands, messages, getting ship object etc.
4. Client is related to Spaceships in a one to one cardinality. This means that each client can have only one ship as its object. The client gets the ship object from Server and can only get the ship statistics but cannot manipulate the ship data directly.
5. Server is related to Spaceships in a 1 to 2..\* cardinality. This means that there is only one server but the server can have spaceships from to 2 to n based on the number of teams playing. Server can manipulate all the data in the Spaceships class and sends the Spaceship object back to the client after every update.
6. Spaceships is related to ShipSystems in a one to one cardinality. This means that each spaceship will have only one set of systems. Spaceships can manipulate all the data in ShipSystems based on commands received.
7. Server is related to Maps in a 1 to 2..\* cardinality. This means that there can be multiple maps that the user can choose to play on and based on this choice the map will be initialized. Server can manipulate the data in Maps class and send the updated map to players.
8. Server is related to Database in a one to one cardinality. This means that there is only one server and one database which we will be dealing with in our game. Server can access and update the data in Database based on the user’s choice.

**Activity Descriptions**

Abc

**State Descriptions**

Abc