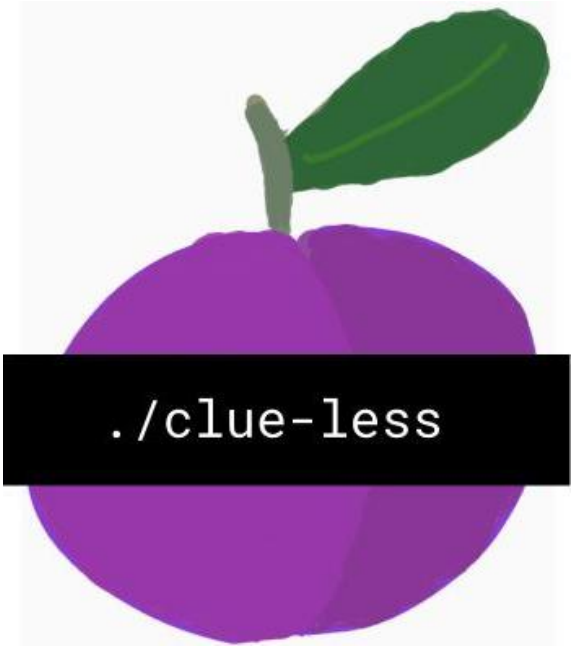
Johns Hopkins University



Software Design Document

Professor Plum's Programmers

Foundations of Software Engineering
Section 81
Spring 2021

v1.0

System Architecture

The Clue-less Game architecture will consist of two main components: one server and multiple connected client applications. In our implementation, the server will be responsible for maintaining all ongoing games, while the client will be responsible for displaying a Graphical User Interface (GUI) on the local machine that each player is using. Communications between the server and each client use Hyptertext Transfer Protocol (HTTP), using an Application Public Interface (API) developed for this project. For complete documentation of the Clue-less API, please see the API Readme.

API

General Game Updates

- Ongoing Games Update
  - HTTP Request: GET
  - Location: *serverHostLocation/*games
  - Request Parameters: None

- New Game
  - HTTP Request: Post
  - Location: *serverHostLocation/*games
  - Request Parameters
    - Game name
    - password (optional)

- Join Game
  - HTTP Request: POST
  - Location: *serverHostLocation/*games/join
  - Request Parameters
    - gameId
    - username
    - password (if set by Start)

- Start Game
  - HTTP Request: PUT
  - Location: *serverHostLocation/*games/join
  - Request Parameters
    - gameId

Player Actions

- Select Player Suspect
    - HTTP Request: PUT
    - Location: *serverHostLocation/*players/selectSuspect
    - Request Parameters
        - gameId
        - username
        - suspect

- Move
    - HTTP Request: PUT
    - Location: *serverHostLocation/*gameplay/move
    - Request Parameters
        - gameId
        - username
        - location

- Suggestion
    - HTTP Request: PUT
    - Location: *serverHostLocation/*gameplay/suggestion
    - Request Parameters
        - gameId
        - username
        - suggestion
            - suspect
            - weapon
            - room

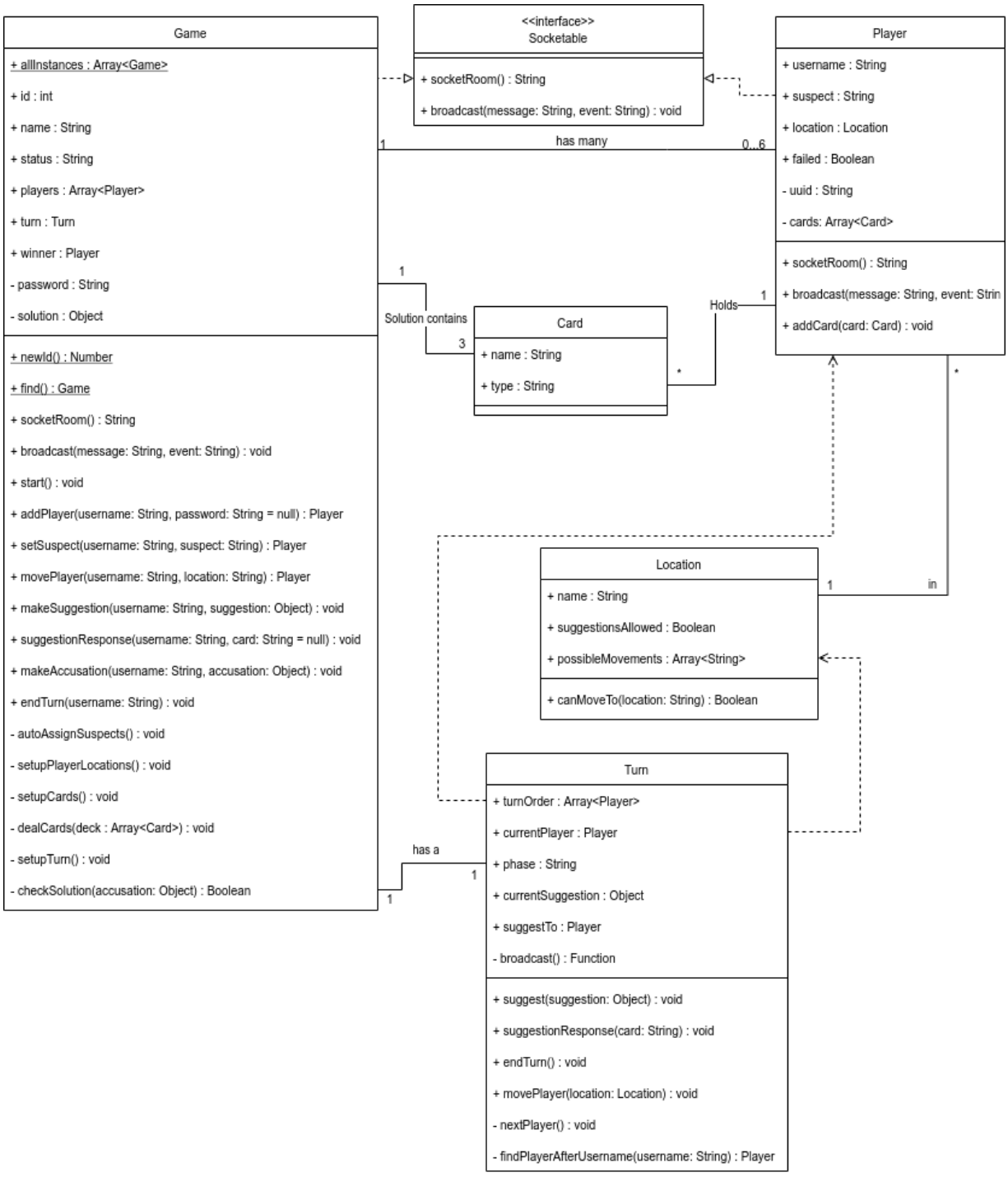
- Suggestion Response
    - HTTP Request: PUT
    - Location: *serverHostLocation/*gameplay/suggestion
    - Request Parameters
        - gameId
        - username
        - card (if player has card matching suggestion)

- Accusation
    - HTTP Request: PUT
    - Location: *serverHostLocation/*gameplay/accusation
    - Request Parameters
        - gameId
        - username
        - suggestion
            - suspect
            - weapon
            - room

Class Diagrams

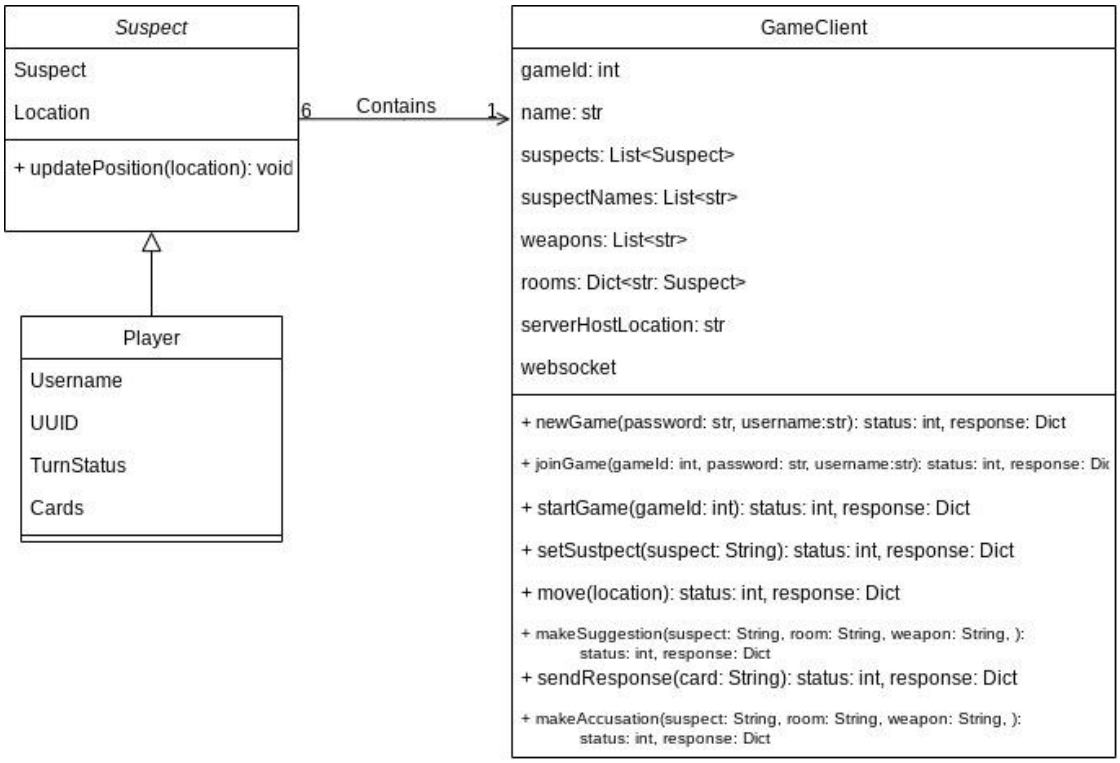
The following section shows UML Diagram for Clue-less entity classes as well as connections between these classes.

Server Class Diagram



In this implementation, the server is responsible for most of the game implementation, tracking each game instance, as well as all player locations, cards, and actions. All rule verification is done in the server classes, to remove the ability for players to tamper with game actions, i.e. allowing for movement to inaccessible rooms.

Client Class Diagram

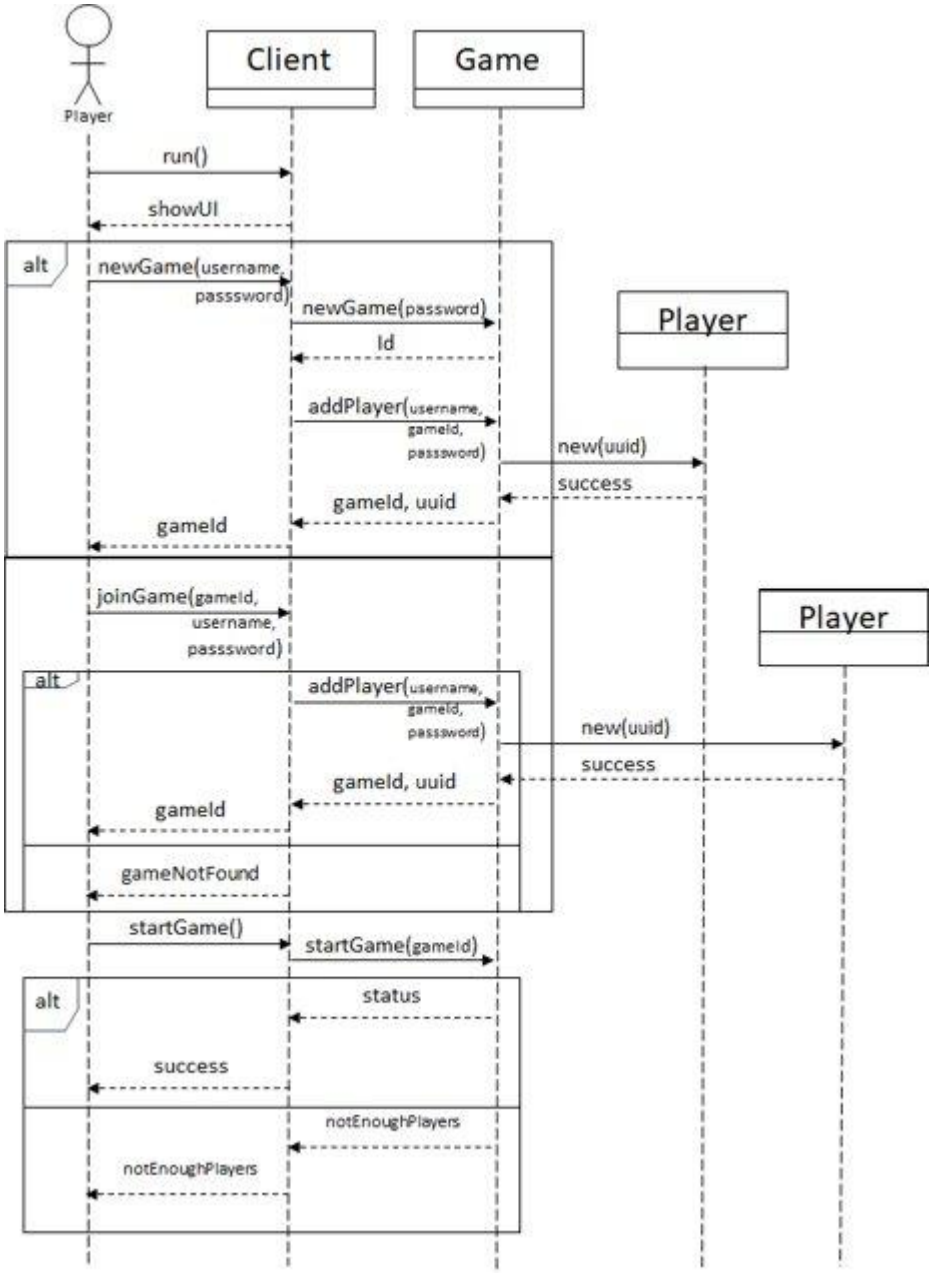


The client classes in this case are much simpler, since each client must only maintain information about the current user, and functions related to sending user information to the server. Each function in the GameClient class contains information from the API, for sending player actions and receiving information back including request status codes and pertinent game information after action is received. The game client also maintains a websocket for receiving information back from the server, which will be used to provide GUI alerts to players containing information about other player actions, including each move in the game history, as well as alerts when the local player's turn has arrived, or when a suggestion response is required.

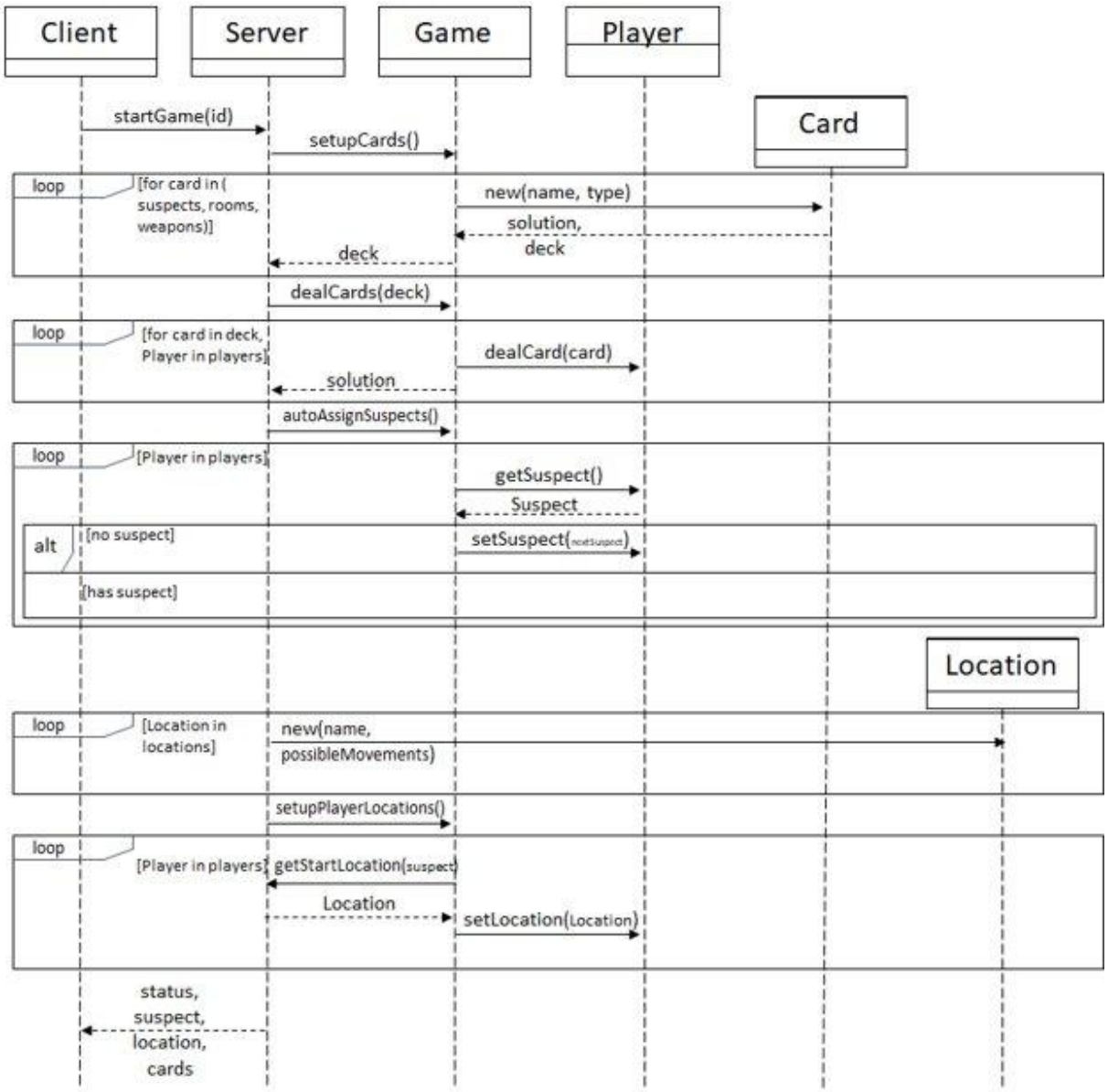
Scenarios

The following diagrams show interactions between the objects and classes shown in the entity Class Diagrams.

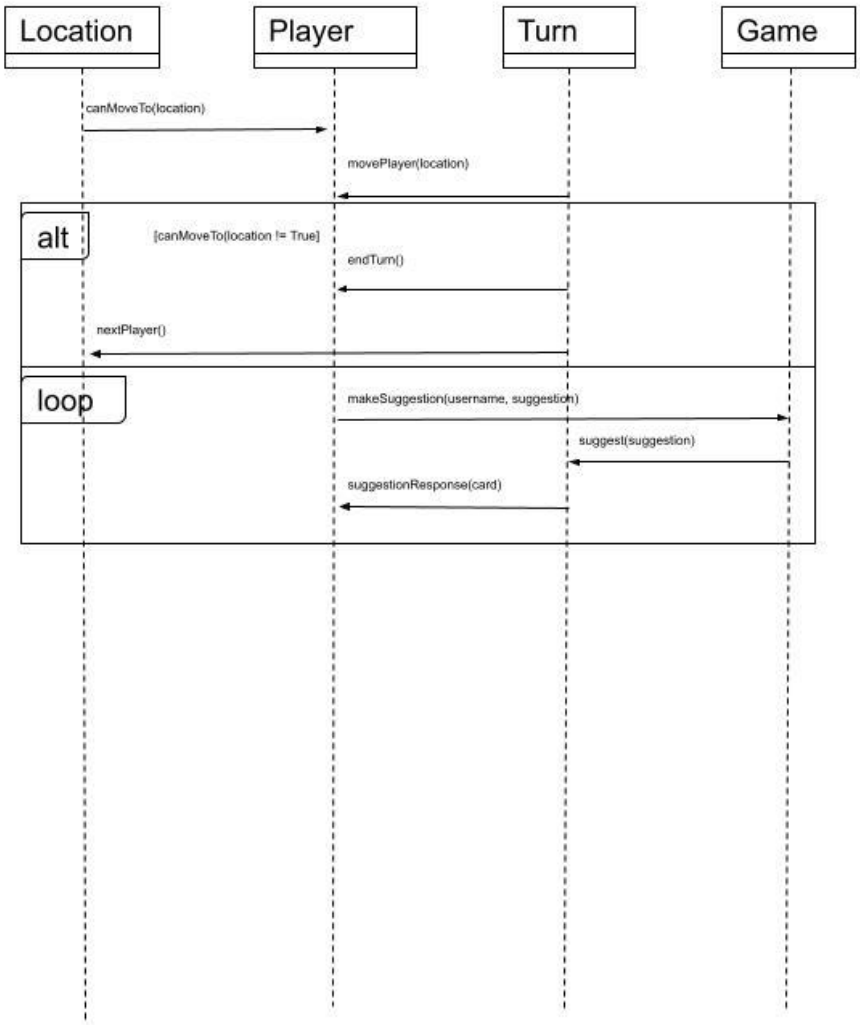
Game Instantiation



Game Setup


Client
Server
Game
Player
Card
Location
startGame(id)
setupCards()
loop
[for card in (suspects, rooms, weapons)]
new(name, type)
solution,
deck
deck
dealCards(deck)
loop
[for card in deck, Player in players]
dealCard(card)
solution
autoAssignSuspects()
loop
[Player in players]
getSuspect()
Suspect
alt
[no suspect]
setSuspect(
[has suspect]
loop
[Location in locations]
new(name, possibleMovements)
setupPlayerLocations()
loop
[Player in players]
getStartLocation(suspect)
Location
setLocation(Location)
status,
suspect,
location,
cards

Player Turns



Accusations