Medusa's Coders

Requirements Model

Comp 361: Software Engineering Model

Yanis Hattab

Yang Yan

Julie Roy-Prévost

Enting Zhou

Aaron Kay

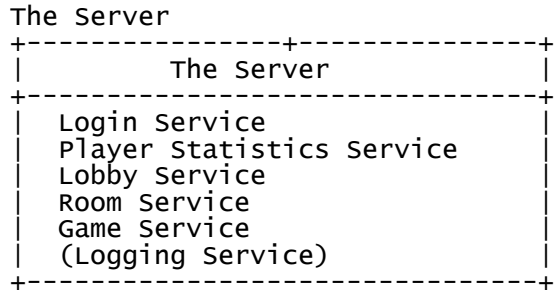
Novembre 24, 2014

Mcgill University

architecture

We will use the client-server model.

The Server



Login Service:
keeping track of online players. Log player in. Create new player. Log player out.

Player Statistics Service:
keeps track of a record for each client. This record includes player name, client ID, number of game played, and wins. The service will include updating client's record and retriving client's record. This service is private and can only be used by the server itself.

Lobby Service:
will be in charge of keeping a record of all the rooms, listing the players online, listing the rooms, displaying players' statistics

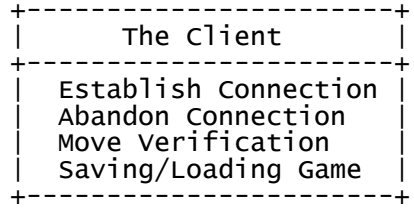
Game Lobby/Room Service:
creating a room, listing players in the game room and preparing game and starting the game.
In the case the client choose to load a game from his/her hard drive, this local map will be sent to the server and the server will send the map to all the other clients.

Game Service:
serve as a message broker for the all the clients in a game. The server will keep a map that is in sync with the latest map of the clients. It will communicate with client using Remote Procedure Call.
If a client had made a game move, it would send a message to the server, the server would then update itself's map and then push the update to all the other clients. The update will only succeed if the sender received a confirmation from the recipient, if not succeed, it will block and wait until timeout, and the client will be considered disconnected from the server.
The Game Service will also be in charge of notifying the client of the turn change, the game win and ending the game.
Only the server can generate trees and gold at the end of each turn.

Logging service:
keeping a log of the current game. This service is also private.

Storage:
player stats should be permanent and will be stored in file or a database. rooms and games should be in RAM, and are destroyed when nobody remains in the room/game.

The Client



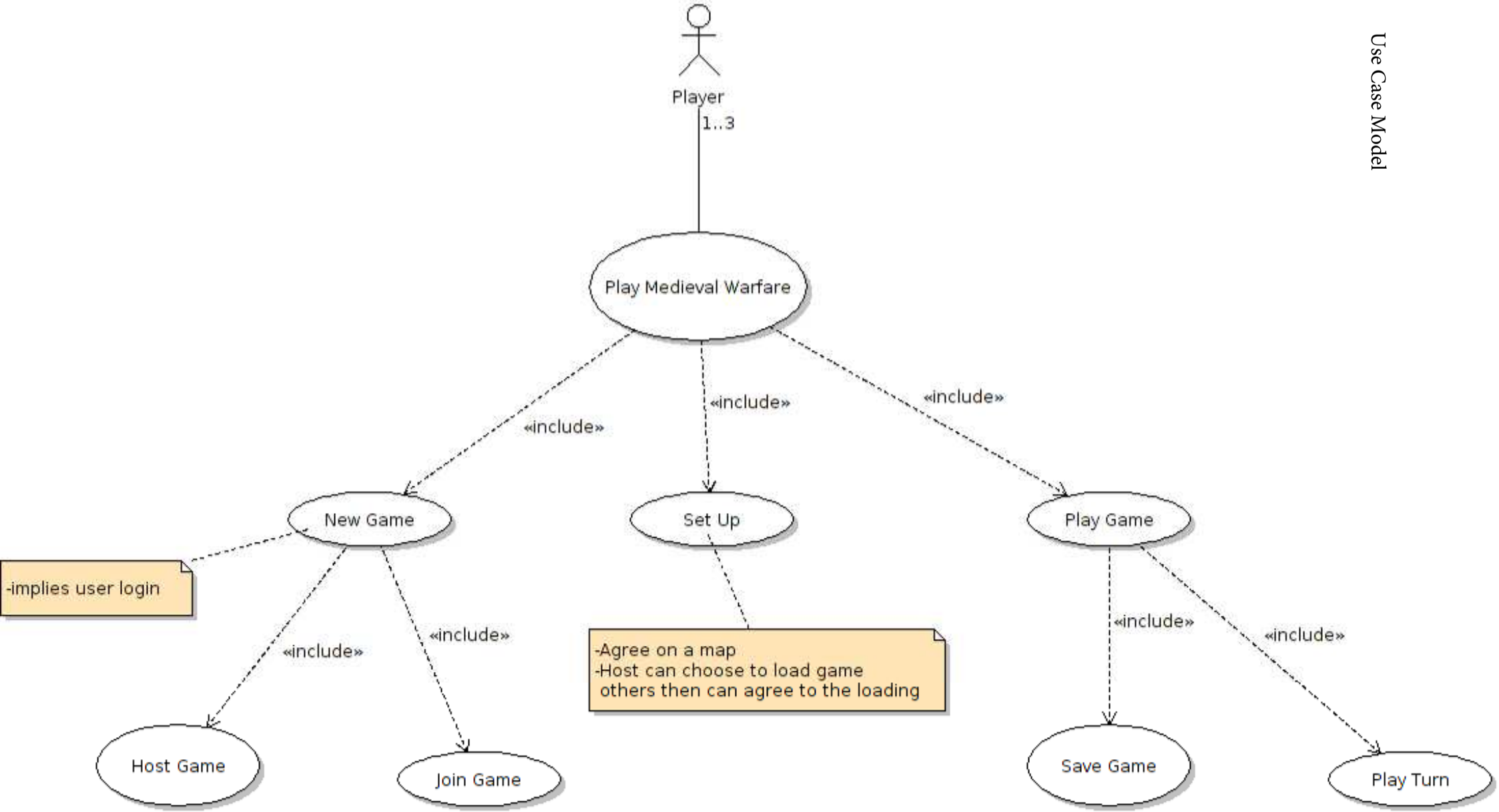
Establish Connection:
when a client is started, it will automatically establish a connection with the server.

Abandon Connection:
	when the client exits, it will disconnect from the server.

Move Verification:
	verify each game move is valid before making it.

Saving/Loading Game:
	each client can save game, game will be saved in local drive. And when in the game room, The host(whoever created the room) will be able to load the game from his/her hard drive. Before the game started, the loaded game will be sent to the server, and the server will send the game to all other clients in the room.

Storage:
	currently playing game is stored in RAM. And gameSave will be in hard disk.



Use Case Model

Use Cases

Use Case	Play Medieval Warfare
Scope	System
Level	Summary
Intention in Context	The intention of the Player is to be able to achieve all different requirements in order to finally play the game. This doesn't include only playing itself but it includes all setup and hosting stuff
Multiplicity	4
Primary Actor	Player
Main Success Scenario	1. Player starts a new game, completes logging in New Game
	2. Play either Join Game or Host Game
	3. Player setup all that is needed (agree on a map, possibly load a game) in Set Up
	4. Player starts playing the game in Play Game
	4. Player finishes or saves the game
Extensions	2a. Fails if Player tries to load a game but chosed to join an existing game, then must select new game back at step 1

Use Case :	Set Up
Scope :	Engine, Player,Server
Level :	User Goal
Intention in Context :	The Player intention is to set up the remaining requirements in order to be ready to play. (Agree on a map, load a previously saved game, etc.)
Multiplicity :	4

Primary Actor :	Player
Main Success Scenario :	1. Player loads a map (new or existing one)
	2. All players agree on the map and sends agreed decision to the System
	3. System requires all 4 players to be ready then moves to Play Game
Extensions :	3a. If not enough players, then wait for players or try another map which might have enough player waiting

Use Case :	Save Game
Scope :	Engine, Map
Level :	Subfunction
Intention in Context :	The intention of the Player is the save all the data associated with the current game in order to play able to play at a later time
Multiplicity :	4
Primary Actor :	Player
Main Success Scenario :	1. Player selects Save Game in the in game menu
	2. Game data gets saved locally
Extensions :	

Use Case :	New Game
Scope :	Server, UI, Player
Level :	User Goal
Intention in Context :	The intention of the Player is to be able to fill in the logging info (enter usernmae), and decide to Host Game or Join Game

Multiplicity :	1
Primary Actor :	Player
Main Success Scenario :	1. Player logs onto System providing username
	2. Player decide to Host Game or Join Game
Extensions :	

Use Case :	Play Game
Scope :	Engine, UI
Level :	User Goal
Intention in Context :	The Player intention is to start playing the game which means Play Turn, or Save Game
Multiplicity :	4
Primary Actor :	Player
Main Success Scenario :	1. Player Play Turn
	2. System allow Player to see other players turns(moves)
	3. When needed, Player can Save Game
Extensions :	2a. If other players stop playing, then game ends

Use Case :	Play Turn
Scope :	Engine, UI, Map, Unit
Level :	Subfunction
Intention in Context :	The intention of the Player is to do some moves in order to play the game and then control is given over to other players when he clicks *endTurn* until its his turn again
Multiplicity :	4
Primary Actor :	Player
	1. Player performs a legal move

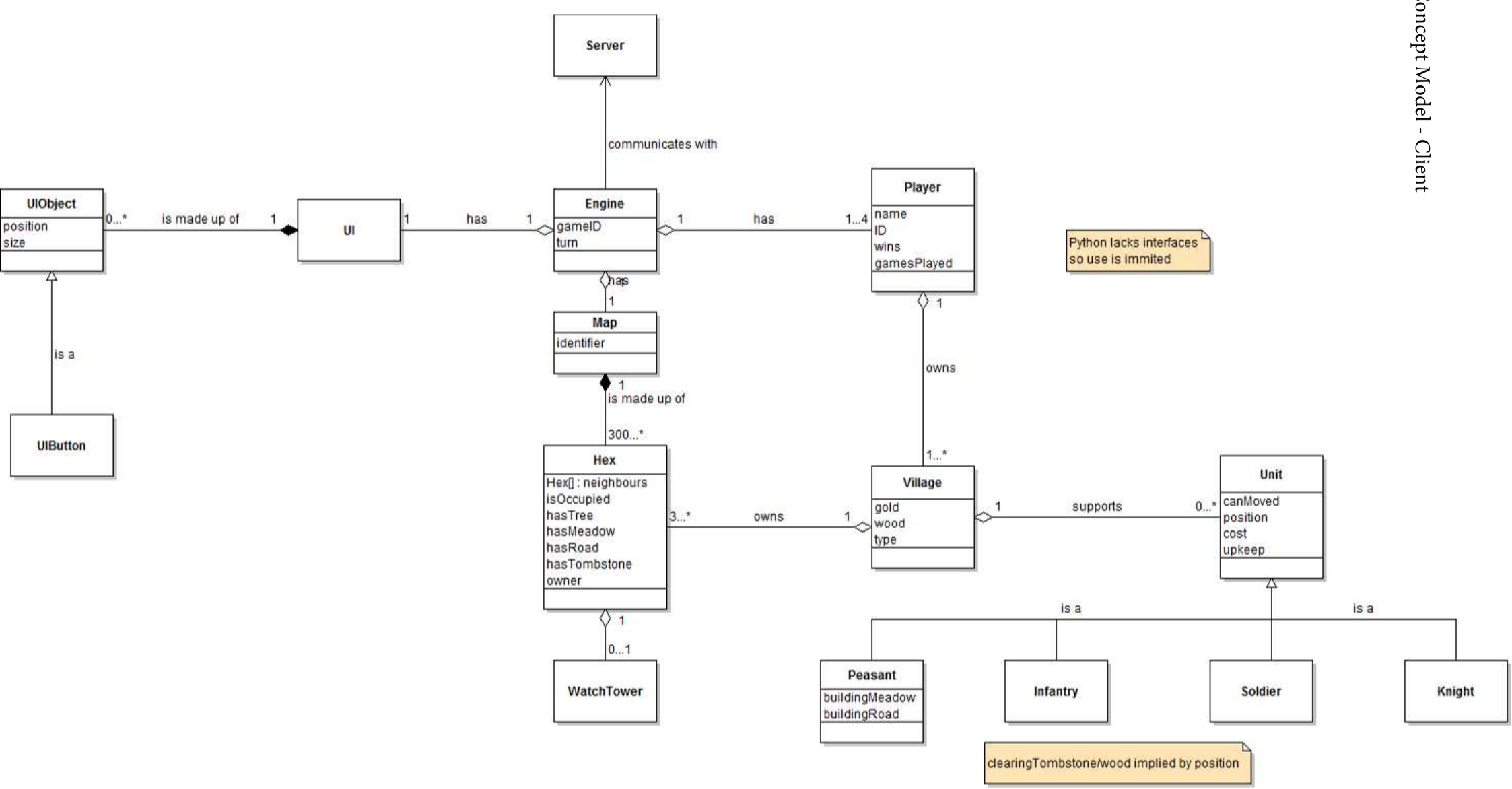
Main Success Scenario :	2. Player end his turn and tells System to hand control over to other players
	3. System tells Player to regain control when its his turn
Extensions :	1a. Player performs illegal move he must input another move

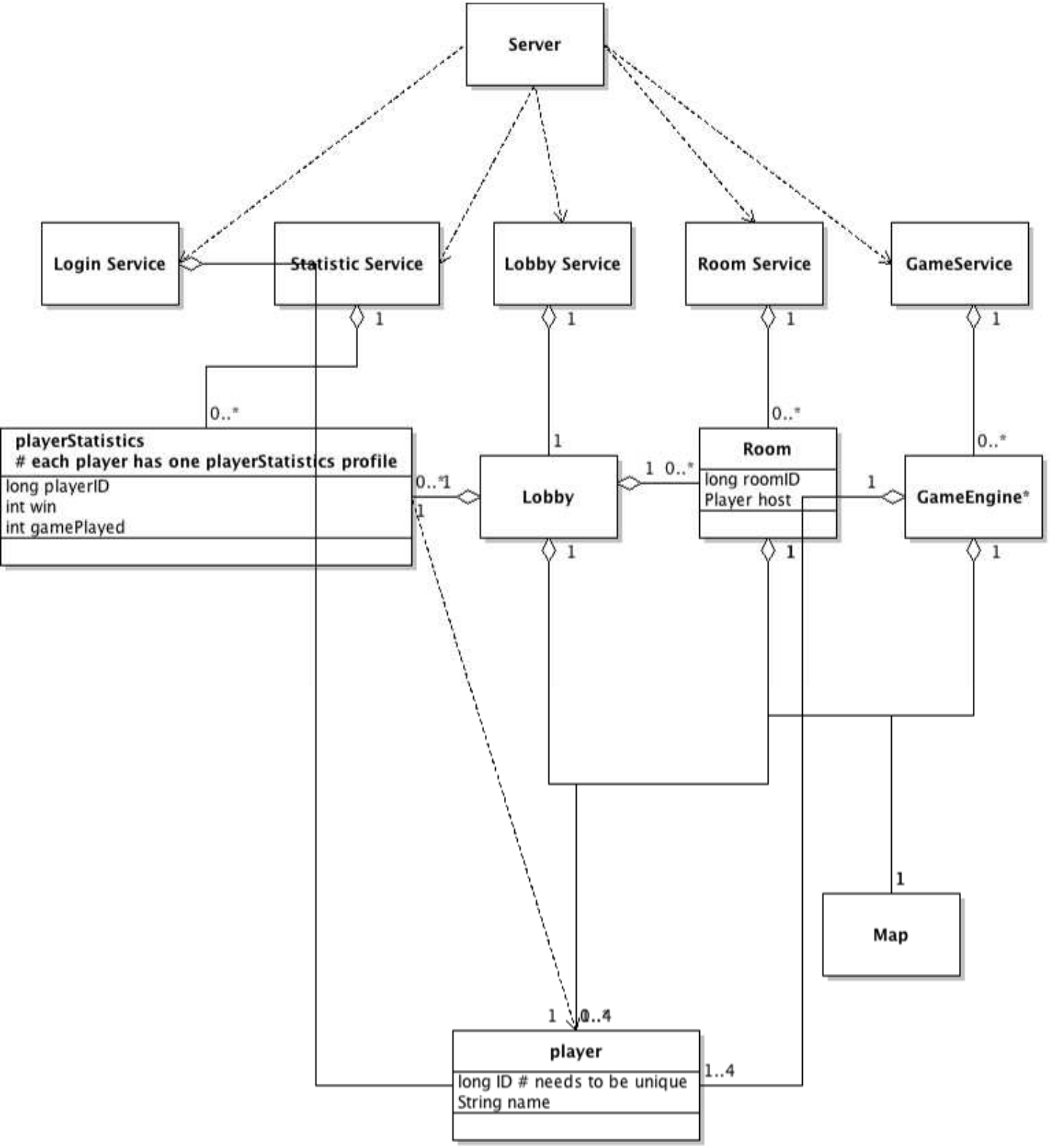
Use Case :	Host Game
Scope :	Server, Player
Level :	Subfunction
Intention in Context :	The intention of the Player is to start a completely new game. It then becomes the host of this game, he must select a map.
Multiplicity :	1
Primary Actor :	Player
Main Success Scenario :	1. Player selects New Game
	2. Player selects Host Game
	2. Player selects a map and moves to Set Up
Extensions	

Use Case :	Join Game
Scope :	Server, UI
Level :	Subfunction
Intention in Context :	The Player intention is to join a lobby that another host created and a host decide on game and play with them
Multiplicity :	3
Primary Actor :	Player
	1. System displays existing games

Main Success Scenario :	2. Player decides to join its prefered existing game and moves to Set Up
Extensions :	1a. If no existing game present in lobby, then Player must create a new game

Concept Model - Client





Services handles the input from clints and call the cooresponding method in the stateful object

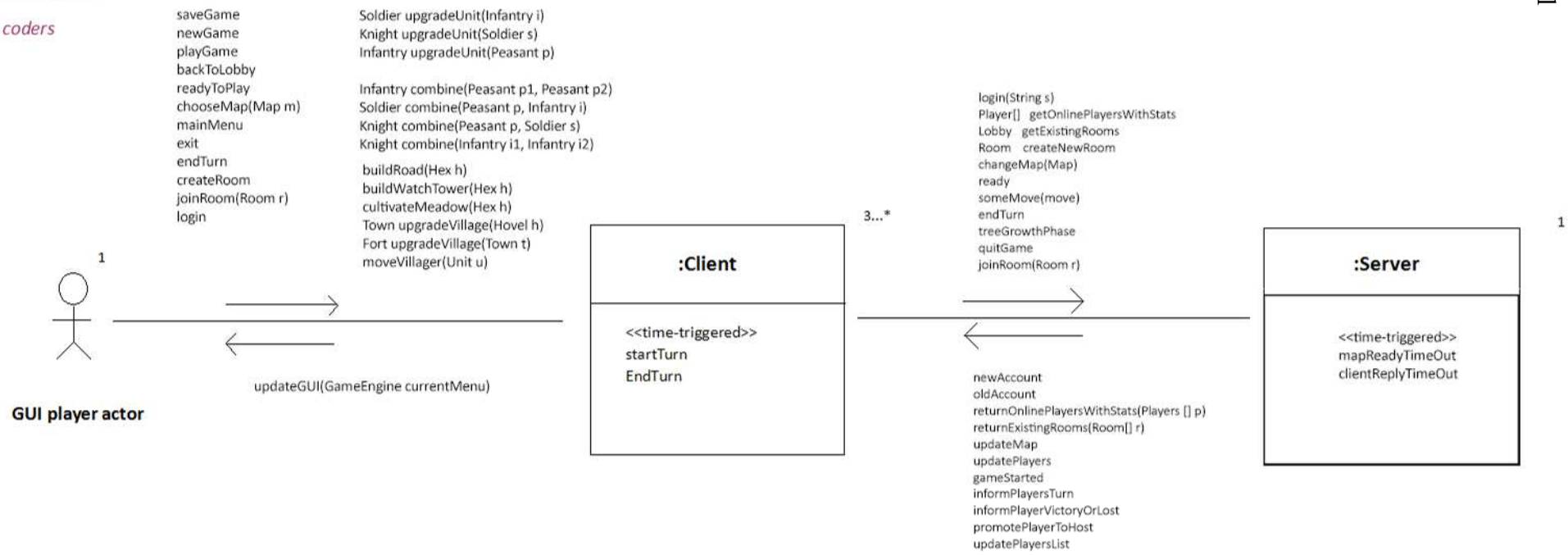
It is pretty much the same as client GameEngine Model
This is the GameEgine on Server, it contains methods that client does not have, e.g. change turn, decide game win

map is stored in RAM in server

Concept Model - Server

Environment Diagram

By Medusa's coders



Operation model	
Operation :	Client :: saveGame
Scope :	Engine
Messages :	
New :	GeneratedGameData file
Pre :	The player who wants to saveGame must be in game. This player must click the save button of the menu.
Post :	The System writes to a file the current game data and it is saved locally on that Player's drive.
Use Cases :	Save Game
Operation :	Client :: upgradeUnit
Scope :	Unit, Village
Messages :	Unit : {upgrade}
New :	upgradedUnit
Pre :	There must be enough resources in the selected village. The selected unit must not already be a knight
Post :	The System destroys the current unit and creates an upgraded unit. The System decreases the amount of gold of the selected village
Use Cases :	Play Turn
Operation :	Client :: buildRoad
Scope :	Hex, Peasant, Map
Messages :	Hex :: {Hex has road}
New :	
Pre :	The selected hex must have no forest on it. That hex cannot have a road on it already. The unit must be a peasant, and it must be on that selected hex already
Post :	The System builds a road on the hex occupied by the peasant. (One turn required to complete)
Use Cases :	Play Turn

Operation :	Client :: exit
Scope :	Engine, UI
Messages :	Engine :: {Player ID X removed from game}
New :	
Pre :	The Player can be at any point in the game or at setup stage. The Player wants to exit and kill everything related to Medieval Warfare
Post :	The System closes the whole game and brings the Player(User) back to desktop. Player is removed from list of current Players
Use Cases :	Play Turn

Operation :	Client :: mainMenu
Scope :	UI, Player
Messages :	
New :	
Pre :	The Player must be in game(playing).
Post :	The System clears the game data and brings the Player back to lobby
Use Cases :	Play Turn

Operation :	Client :: combineVillagers //different overloaded methods but all the same operation schema
Scope :	Unit
Messages :	unit : {combined units}
New :	upgradedUnit
Pre :	The Player must provide to the System the required combinations of units (two peasants, one peasant one infantry, one peasant one soldier, two infantries). i.e. The Player must have those different combinations in its game
Post :	Both provided units are deleted, one stronger unit is created(Infantry, Soldier, or Knight)
Use Cases :	Play Turn

Operation :	Client :: buildMeadow
Scope :	Hex, Unit, Map

Messages :	Unit :: {HexBeingBuilt}
New :	meadowedHex
Pre :	The current Hex must not have a forest on it
Post :	The Player tells the System that he initiates the building of a meadow (two turns required to complete).
Use Cases :	Play Turn

Operation :	Client :: endTurn
Scope :	Engine
Messages :	Player : TakeControlAway from that Player
New :	ActionEventsBundle
Pre :	The Player must be the one currently playing in order to end its turn. The Player must not and cannot be in the middle of a non completed action in order to end its turn.
Post :	The System shifts the control over to another player
Use Cases :	Play Turn

Operation :	CLIENT :: startGame
Scope :	Engine
Messages :	
New :	
Pre :	Every Player in the room must have confirmed to that System that they are ready to play. The room must contain enough Players to start a game.
Post :	The System starts the game and hands control over to the first Player
Use Cases :	Play Game

Operation :	CLIENT :: joinRoom
Scope :	Engine
Messages :	
New :	
Pre :	There need to exist at least one room. The room must not be full.
Post :	The Player(user) enters this room
Use Cases :	Join Game

Operation :	CLIENT ::backToLobby

Scope :	UI
Messages :	
New :	
Pre :	The Player needs to be in a room
Post :	The System brings the Player back to the lobby
Use Cases :	Set Up

Operation :	CLIENT ::readyToPlay
Scope :	Engine
Messages :	
New :	
Pre :	The Player must be in a room. The Player cannot already be ready to play
Post :	The System is aware the that Player agreed on current map.
Use Cases :	Set Up

Operation :	CLIENT ::ChooseMap
Scope :	Map, Engine
Messages :	
New :	
Pre :	The Player(user) has to be the host and he has to be in a room.
Post :	The Player selected the map for which other players entering that room must agree on. System aware of which map is selected.
Use Cases :	Set Up

Operation :	CLIENT ::upgradeVillage //different overloaded methods but all the same operation schema
Scope :	Map, Village
Messages :	Village : {Type changed to X}
New :	Setup
Pre :	The selected village cannot be a fort. The selected village must have at least 8 wood.
Post :	The selected village is upgraded (It becomes either a town or a fort)
Use Cases :	Play Turn

Operation :	CLIENT ::createRoom
Scope :	
Messages :	

New :	gameID
Pre :	The Player(user) needs to be in the lobby.
Post :	The Player is now in the created room. The Player becomes the host of this created room. The System adds a new room to its list of current rooms
Use Cases :	Host Game
Operation :	CLIENT ::buildWatchTower
Scope :	WatchTower, Hex
Messages :	
New :	Watch Tower
Pre :	The selected village must have at least 5 wood. The selected village cannot be an hovel. The tile that you select must be on one of the selected villages controlled hexes.
Post :	A watch tower is build on the hex you selected to build on
Use Cases :	Play Turn
Operation :	CLIENT ::login
Scope :	Player, Engine
Messages :	
New :	Player :: {Player ID added}
Pre :	The Player(user) must be at the login screen.
Post :	The Player is now logged into the System and he is in the lobby
Use Cases :	New Game
Operation :	CLIENT ::moveUnit
Scope :	Unit, Hex
Messages :	Hex : {Hex X is occupied}
New :	
Pre :	The selected unit can still move in this turn. The Player needs to designated a valid hex where to move to. The unit can be a peasant infantry, soldier or knight
Post :	The unit is now in the designated hex.
Use Cases :	Play Turn
Operation :	CLIENT ::cultivateMeadow
Scope :	Hex, Unit

Messages :	
New :	
Pre :	Unit must be a peasant. There cannot be a tree nor a meadow present on the hex the peasant occupies.
Post :	Peasant will be unable to move for remaining turn and next turn. Meadow will be placed on the hex the peasant occupies on the third turn.
Use Cases :	Play Turn

Operation :	GUI::updateGUI
Scope :	UI, Engine
Messages :	
New :	
Pre :	
Post :	Observer callback to tell GUI to regenerate data. The GUI is updated with the new game state. New needed windows and frames are displayed
Use Cases :	all
///////////////	**Client-Server Operations**

Operation :	Client::updateMap
Scope :	Server.RoomService
Messages :	client : {new-map-id; new-bitmap}
New :	if new-bitmap, create a new map object from the given bitmap
Pre :	host changed or loaded a new map
Post :	Client update its map and notify GUI to display the new map. When the host of the room change or load a map, the server will send a updateMap message to the clients in this room.
Use Cases :	Set Up

Operation :	Client::updatePlayers
Scope :	Server.RoomService

Messages :	client : {players[]}
New :	players list
Pre :	A client left or joined the room, while other clients are in the lobby.
Post :	Client updates its player list and notify GUI to display the difference. When someone left or join the room, lead to the room having a different player list in it. The server will notify the clients in this room of the new player list.
Use Cases :	Set Up

Operation :	Client::gameStarted
Scope :	Server.RoomService
Messages :	client : {gameStartedMsg}
New :	
Pre :	at least 2 players in a room and all of them are ready
Post :	Client will enter game state and update the GUI to display the game. When 2 or more players in the room and all are ready, the game will automatically start by the server. The server send gameStarted Message to indicate this.
Use Cases :	Set Up

Operation :	Client::promotePlayerToHost
Scope :	Server.RoomService
Messages :	client : {new-host}
New :	
Pre :	when the host of the room left the room
Post :	Clients will update who is the host from the player list and notify GUI of the difference.When the host left the room, a new host is promoted by the server.
Use Cases :	Play Turn

Operation :	Client::informPlayerTurn
Scope :	Server.GameService
Messages :	client : {your-turn}
New :	
Pre :	when it is this client's turn
Post :	Client will notify GUI that it is it's turn. When last client had end its turn. The server will send informPlayerTurn to the next client in the player list.

Use Cases :	Play Turn
Operation :	Client::informPlayerVictoryOrLost
Scope :	Server.GameService
Messages :	client : {victory; lost}
New :	
Pre :	one client won
Post :	When a client wins, the server will notify its victory and notify other players' their lost. The server will return to the roomService state and clients can do rematch. And the server will update the player's statistic according to wins and lost. The Client is returned to the Room and Player stats should be updated on the server.
Use Cases :	Play Turn
Operation :	updatePlayerList
Scope :	Server.GameService
Messages :	client : {players[]}
New :	
Pre :	When a player left from the game. Other players in the game will be notified.
Post :	Other clients will update their map to reflect the player's absence, and will notify GUI for the changes. When a player left a game. Other players will be informed.
Use Cases :	Play Turn
Operation :	treeGrowthPhase
Scope :	Server.GameService
Messages :	client : {seed(int)}
New :	random trees
Pre :	At the end of the round, i.e. the last player in the list has finished his/her turn
Post :	All clients will generate trees randomly using the seed provided. The random generation of trees is done by the server providing a seed at the end of the round and clients generate trees using this seed
Use Cases :	Play Turn

Operation :	Server::login
Scope :	LoginService, player
Messages :	String:{username}
New :	
Pre :	The client asked the player to input his username
Post :	The server checks if the username already exists, if so it returns a welcome back
Use Cases :	New Game

Operation :	Server::getOnlinePlayersWithStats
Scope :	StatisticService, playerStatistics, player
Messages :	
New :	
Pre :	After the player legin was identified
Post :	The server returns currently logged in players and their statistics
Use Cases :	New Game

Operation :	Server::getExistingRooms
Scope :	LobbyService, Lobby, Room
Messages :	
New :	
Pre :	The client asks for the rooms available after it has logged in.
Post :	The Server returns the currently open rooms so that client can choose to join one.
Use Cases :	Join Game

Operation :	Server::createNewRoom
Scope :	LobbyService, Room, Player
Messages :	Client::{Map}
New :	Room
Pre :	The client chose to host a game and provided a map.
Post :	The Server creates and returns a new room in the global lobby with the requesting
Use Cases :	Host Game

Operation :	Server::changeMap
Scope :	LobbyService, Room, Player
Messages :	Client:{Map}
New :	
Pre :	The Client must be the host to be able to change the map

Post :	The Server changes the map of the room that is hosted by the requesting client to the provided one
Use Cases :	Set Up

Operation :	Server::ready
Scope :	LobbyService, Room, Player
Messages :	
New :	upgradedUnit
Pre :	The client must have joined a room and decided he agrees on the proposed map.
Post :	The Server records that that client agrees with the map choice, if all players agree,
Use Cases :	Set Up

Operation :	Server::someMove
Scope :	GameService, Player, GameEngine
Messages :	Client :: {Move}
New :	
Pre :	The game has started and it's the client's turn.
Post :	The server records the move that has been validated by the client, it adds the
Use Cases :	Play Turn

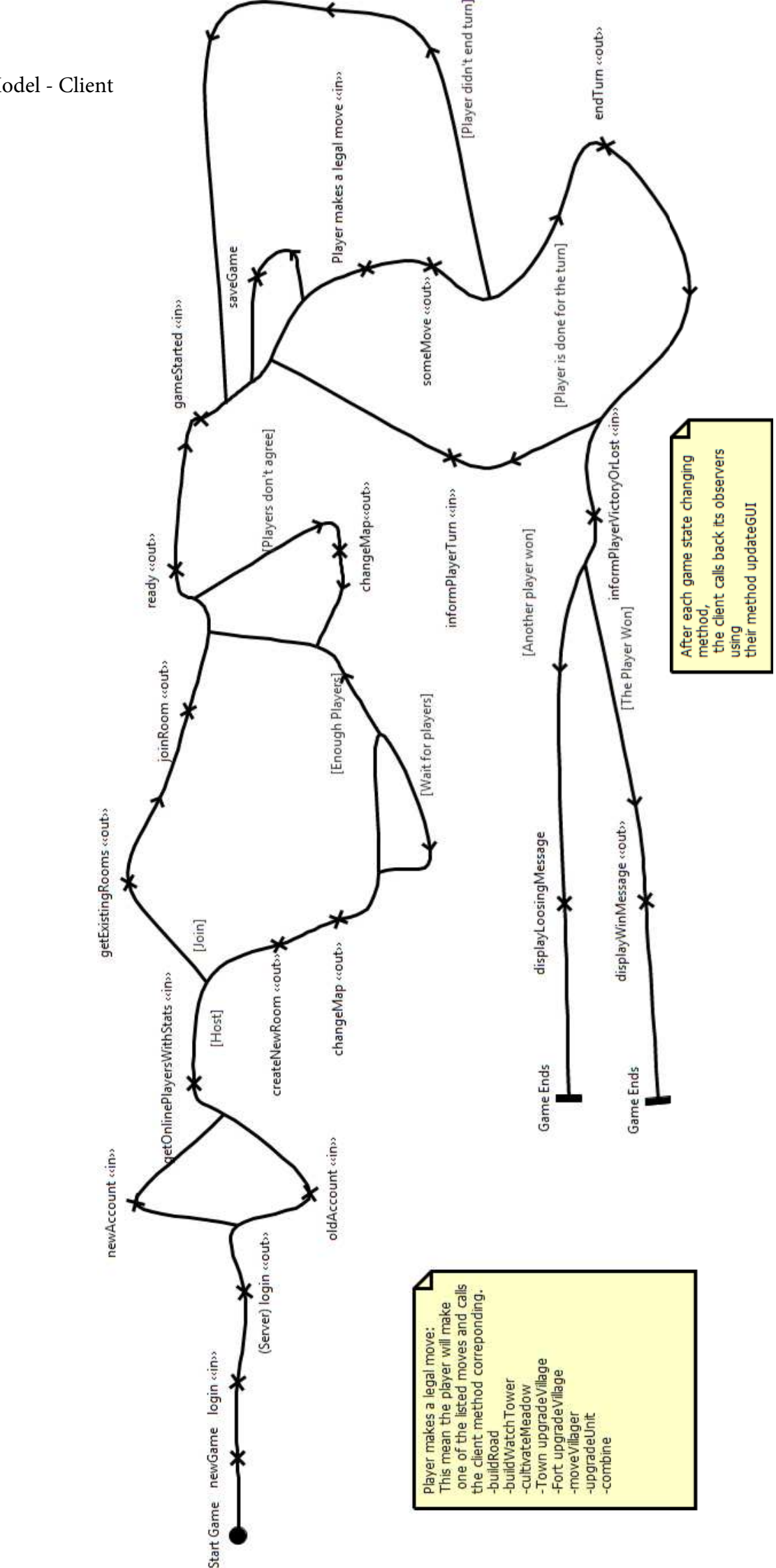
Operation :	Server:: endTurn
Scope :	GameService, Player
Messages :	
New :	
Pre :	It must be the client's turn to end its turn.
Post :	The server bundles all the actions of the client in a collection and sends them to
Use Cases :	Play Turn

Operation :	Server :: quitGame
Scope :	GameService, Player
Messages :	
New :	
Pre :	A player must currently be playing in order to quit the game.
Post :	The server removes the player from the room and deletes all his units, his territory
Use Cases :	Play Turn

Operation :	Server::joinRoom

Scope :	LobbyService, Lobby, Room
Messages :	Client:{Room}
New :	
Pre :	The client decided to join a room after it has logged in and selected one from the
Post :	The Server puts that player in the required room and sends him the room data
Use Cases :	Join Game

Protocol Model - Client



Protocol Model - Server