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

