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. That Player game is stopped, System brings him back to the lobby screen
Use Cases :	Save Game

Operation :	Client :: upgradeUnit
Scope :	Unit
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 :	

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 :	

Operation :	Client :: combineVillagers //different overloaded methods but all the same
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 Player 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 :	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 :	

Operation :	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 map
Use Cases :	Setup

Operation :	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 :	Setup

Operation :	upgradeVillage //different overloaded methods but all the same operation
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 :	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 :	buildWatchTower
Scope :	WatchTower, Hex
Messages :	
New :	Watch Tower
Pre :	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 :	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 :	

Operation :	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 :	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 :	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 :	