Aaron Kay
Yanis Hattab
Yang Yan
Julie Roy Prevost
Enting Zhou

DESIGN MODELS

GAME MANAGEMENT

We choose to set the controller of the newGame() operation as the Engine (game engine) as initially the other class haven't been instantiated yet and it's the engine's responsibility to do that.
On the other hand, most of the beginTurn() operation is assigned to the Player class because it has access to its villages and units and it is the most logical choice. We divided the operation into 4 sub-operations that the Engine calls on the Player p given as argument.

1) in newGame() we pass data about the map as an argument called mapData, the map will be encoded as a bit array that is passed to the constructor of the class Map so it can generate its Hexs (tiles) accordingly

2) Unit has a string to represent the unit type called unitType, it can be set to "peasant", "infantry" "soldier" or "knight"

3) Village also has its type represented as a string called VillageType that can be one of "hovel", "town" or "fort"

4) We use an enum of actions a unit can be doing during a turn, most building activities last only one turn except the cultivate one, for which we have cultivate1 for the first turn of the action that gets changed to cultivate2 when we review the unit at the beginning of the second turn

5) Each tile (hex) has a method getIncome() that gives back an int og gold that the hex is worth depending on its type

GAME MOVES

All the 5 operations are under the responsibility of the Engine. This choice was made to centralize them in one single class as to be accessible by the GUI which will communicate directly with the Engine. All the five operations are directly linked to a GUI action (clicking on build road button, moving a unit, etc...)

1) within takeOver() we call the method getUnconnectedHexs() on the enemy village that just lost a tile, this is a method that does a BFS through the villages hexs to identify unconnected Hexs and return them

2) still within takeOver(), we call the populateHexs() method defined on a map to handle the unconnected hexs which means that if possible (3 or more hexs connected) spawn villages and transfer units on their territory to them, if not possible turn them into neutral hexs (killing units present on them).

3) inside the upgradeUnit() operation, we use the helper function upgradePrice(String newLevel) that is defined on a unit to get the price to upgrade that unit to the new level newLevel

4) A hex has a method isProtected(Unit u) that takes the attacking unit and checkes neighbouring hexs to see if there is a unit of equal or higher level, if so it returns true, otherwise it returns false