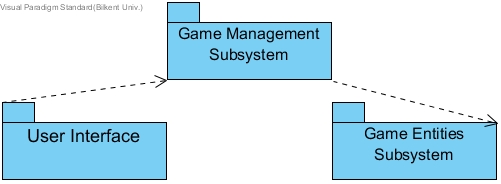
# 2.3 Architectural Styles

## 2.3.1 Layers

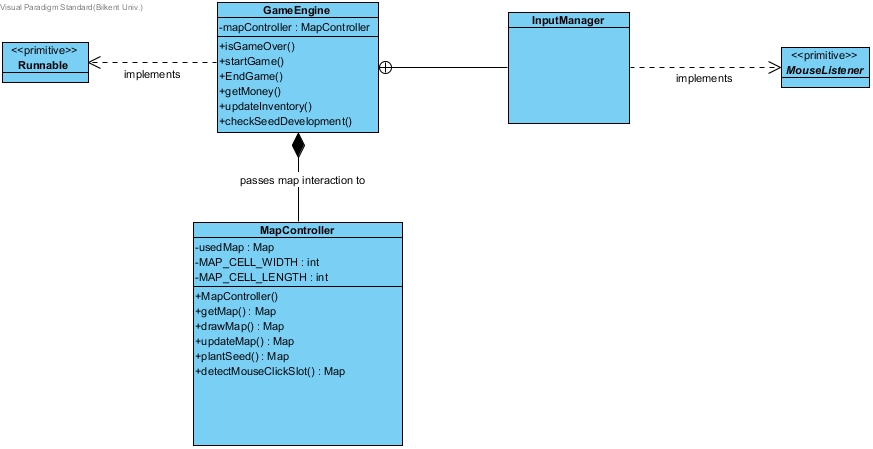
System of game will be separated to 3 different layers including UI (User Interface), Game-Map management, and game entities. It can be implied that UI system decomposition does not depend from any other subsystem unlike Game Management and Game Entities system decompositions. On the other hand, last layer Game Management subsystem depends from UI and also controls Game Entities and Game Logic. Additionally, Game Entities subsystem are composed of all entity objects which are brought together.



## 2.3.2 Model View Controller (MVC)

Aim of this architectural style is to separate your subsystems into 3 parts including controller, model, and view. As it is implied in the layers section we have 3 different layers, which these layers will be classified for MVC. In MVC architectural style, controller responds to user’s input by updating model objects. Besides, view is the demonstration of the models in desired format. And model is to management of the objects’ data. In relation with our system, Game Management Subsystem establishes Controller. User Interface which is interaction between user and system establishes View. At last, Game Entities Subsystem makes up model of the MVC.

# 3.3 Game Management Subsystem



### GameEngine class

### C:\Users\Fuad\AppData\Local\Microsoft\Windows\INetCache\Content.Word\GameEngine.jpg

* GameEngine class is the one of the main game management class that is also can be evaluated as Façade class since it hides complexities of other classes and collects all the operations here for the ease of use.

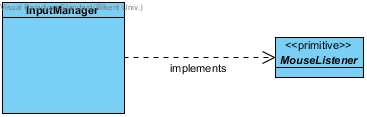
#### Attributes:

* Private MapController mapController: this attribute is an object og MapController class, which is for to use it is methods and associations.

#### Operations:

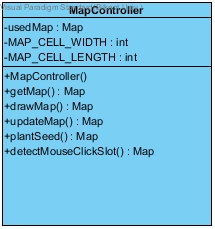
* public boolean isGameOver(): it checks if the game is over or not. This method is related to MapManager class.
* public void startGame(): starts new game, resets all the data about player and inventory.
* Public void endgame(): ends game by the information got from isGameOver() operation.
* Public int getMoney(): notifies the GameEngine class with how much farmer’s money left, which is also used in isGameOver() operation.
* updateInventory(): updates inventory after every crop or buy from store.
* public void checkSeedDevelopment() : this operation checks how much seeds grows larger, and it invokes MapContoller to change map accordingly.

### InputManager class



* This class is essential for detecting user actions with mouse. And it implements MouseListner interface.

### MapManager class



* MapManager class controls, updates, and modifies the map accordingly to relations with other classes.

#### Attributes:

* Private Map usedMap: this is the object of the Map class from the Model layer which this instance keeps all the places of the other objects.
* Int MAP\_CELL\_WIDTH & MAP\_CELL\_LENGTH: this is integer variable for dividing map equally and detecting places of the objects.

#### Operations:

* mapController(): is the constructor of the MapController class which initializes instance of map.
* public Map getMap() : is the operation to return instance of the map for the relations with other classes.
* public void drawMap() : operation for to draw initialized map for View.
* public void updateMap(): operation to update instance of a Map according to any changes especially for planting seeds, and changes in inventory.
* public void plantSeed() : operation to modify map according to states of the seeds or foods. This function also calls updateMap()
* public void detectMouseSelectedSlot() : with this operation MapController will be notified that which slot is selected and call updateMap() operation map according to this method.