**Subsystem Decomposition**

In this part, we will decompose our system into three parts; User Interface, Game Management and Game Objects. With this approach, we can reduce the complexity of the game and make the system eligible to changes. We will achieve high coherency and reduce coupling.

MVC(Model-View-Controller) architectural pattern will be chosen for this system. The system will be decomposed considering this pattern. User Interface subsystem will be our view and it will provide interface to the user. It will contain Menu, Settings etc. Our Game Management subsystem will be controller for the system. It will control the game and it will maintain communication between Model and View. For example, map controller will control the map and game engine will be responsible for controlling the game. Game objects and map will be the Model part of the system.

Subsystems will be separate from each other but they will work closely. User Interface will get the user input and transmit this information to the Game Management system. Then, Game Management will update Game Objects according to the information. Façade design pattern is

----------------------------------------------------------------------------------------------------

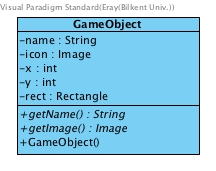
**Class Interfaces**

**Game Entities Subsystem**

Before getting into details of the classes of this subsystem, note that some information is not shown for easier understanding:

* For the sake of clarity, getters and setters are neither mentioned in the descriptions nor shown in the UML diagrams (except the ones whose absence may cause some confusion). However, there will be, of course, getters and setters for all instance variables.
* Unless overridden, inherited attributes are not always mentioned in child classes’ properties (again, unless these members are crucial in understanding the rationale).
* Finally, default constructors are not specified in the descriptions.

**GameObjects Subsystem**



**GameObject Class**

This is an abstract class to generalize the concept of game objects.

Attribute(s):

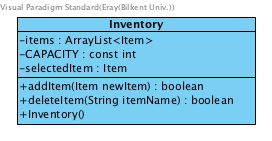
|  |  |
| --- | --- |
| private int x; | is the x-coordinate of a *GameObject* (on the screen). |
| private int y; | is the y-coordinate of a *GameObject*. |
| private Rectangle rect | is used to define the boundary areas of the objects (for the game logic). |
| private String name | is the *name* of the *GameObject*. Specifically, it is the type (such as “*CornSeed*”). |
| public GameObject(int x, int y) | constructs a *GameObject*, assigns *x* and *y* values to the ones provided. It also creates a *Rectangle* object (which is assigned to *rect*). |

* Operation(s):

|  |  |
| --- | --- |
| public abstract String getName(); | is the abstract method to return the *name*, to be implemented accordingly in child classes |
| public abstract Image getImage(); | is, similarly, another abstract method that returns the *icon* of a *GameObject* instance. This will, also, be implemented differently in child classes. |
|  |  |

----------------------------------------------------------------------------------------------------

**Inventory Class**

****

This class represents the inventory where the purchased seeds and collected food are stored. *Inventory* class is one of the children of *GameObject*.

Attribute(s):

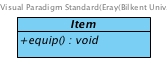
|  |  |
| --- | --- |
| private ArrayList<Item> items; | is an array list to keep *Item* objects (will be used to store *Seed* and / or *Food* instances). |
| private const int CAPACITY; | defines the maximum number of *Item* instances that can be stored in the inventory. |
| private Item selectedItem; | is the *Item* instance that is currently chosen / equipped. |

Operation(s):

|  |  |
| --- | --- |
| public boolean addItem(Item newItem); | receives an *Item* instance and adds it to the array list. If the *CAPACITY* is already full, returns false to indicate this unsuccessful attempt (returns true for the successful case). |
| public boolean deleteItem(String itemName); | given the *name* of an *Item* object as a string, for instance let “corn” be received through the parameter, decreases the *count* attribute of that *Item* by 1. More specifically, if the *Item* instance in the list whose type is received as a string has a quantity (*count*) more than 1, then *count* gets decremented after calling this function. Otherwise, when the *Item* instance has *count* equal to 1, the instance of that *Item* is deleted from the array list. Returns true when an *Item* instance gets deleted or its *count* is decremented (returns false when such an *Item* does not exist). |

----------------------------------------------------------------------------------------------------

**Item Class**

****

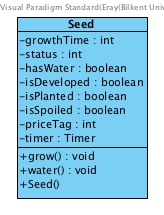
Being another child class of *GameObject*, this abstract class is the parent of *Seed*, *Food* and *WateringCan* classes.

Operation(s):

|  |  |
| --- | --- |
| public abstract void equip(); | will be used to equip an *Item* instance (*selectedItem* in the *Inventory* will be adjusted). |

----------------------------------------------------------------------------------------------------

**Seed Class (and its children)**

****

The *Seed* class is the superclass of its variations (*StrawberrySeed*, *CornSeed*, *SunflowerSeed, TomatoSeed* and *PotatoSeed*) and is a child of the *Item* class. Since there are many shared properties among all kinds of seeds, this generalization of a *Seed* superclass proves useful.

Attribute(s):

|  |  |
| --- | --- |
| private int growthTime; | is the time required for a Seed instance to grow fully. |
| private int status; | demonstrates the status of a *Seed* in terms of growth. |
| private boolean hasWater | keeps whether a *Seed* instance has been watered. |
| private boolean isDeveloped | indicates if a *Seed* object is developed. |
| private boolean isPlanted | is used to understand if a *Seed* object is planted or not. |
| private boolean isSpoiled | is true when a *Seed* gets spoiled, false otherwise. |
| private boolean isFertilized; | keeps whether this *Seed* object is fertilized. |
| private int priceTag; | is the amount of money required to buy a *Seed* from the *Store*. |
| private Timer timer | is kept to measure the time after a *Seed* is being planted. |

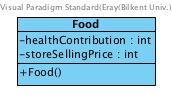
Note that all *Seed* types (i.e. the child classes) have the same attributes as those above. However, *growthTime* and *priceTag* will be overridden in each subclass.

Operation(s):

|  |  |
| --- | --- |
| public void grow(); | will be called to let a *Seed* instance grow, adjusting the status. |
| public void water(); | helps to water a *Seed* instance. |

----------------------------------------------------------------------------------------------------

**Food Class (and its children)**

****

Similar to the *Seed* class, *Food* is a child of *Item*. Moreover, its kinds (*Strawberry*, *Corn*, *Sunflower, Potato, Tomato, Cherry, Raspberry* and *Apple*) are represented as the children of this class.

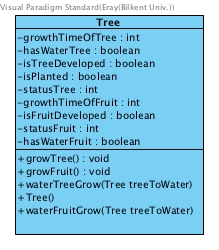
Attributes:

|  |  |
| --- | --- |
| private int healthContribution; | is how much a *Food* instance, when eaten, will add up to the *health* of the *Farmer*. |
| private int storeSellingPrice; | is, similarly, how much *money* will be received by the *Farmer* when a *Food* is sold. |

Again, the child classes (classes representing different kinds of *Food*) have the same attributes. However, each of them will be overriding both the *healthContribution* and *storeSellingPrice* (not shown for easier understanding).

----------------------------------------------------------------------------------------------------

**Tree Class (and its children)**

****

*Tree* class is the superclass of *AppleTree*, *RaspberryTree* and *CherryTree*.

Attribute(s):

|  |  |
| --- | --- |
| private int growthTimeOfTree; | is the growth time of a tree. |
| private boolean hasWaterTree; | keeps if a *Tree* instance is watered. |
| private boolean isTreeDeveloped; | keeps whether a *Tree* instance is developed. |
| private boolean isPlanted; | is used to understand if a *Tree* object is planted somewhere. |
| private int statusTree; | defines the status of the tree in terms of growth. |
| private int growthTimeOfFruit | keeps the growth time of the fruit which is generated from a tree. |
| private boolean isFruitDeveloped; | holds whether a *Tree* instance has formed any fruits (*Food*). |
| private int statusFruit; | keeps the status of the fruit on the tree. |
| private boolean hasWaterFruit; | tracks the water condition of the fruit. |

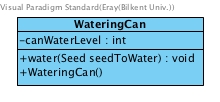
Operation(s):

|  |  |
| --- | --- |
| public void growTree(); | is called to let a *Tree* instance grow. |
| public void growFruit(); | depending on the tree’s development, this function will help produce a fruit. |
| public void waterTreeGrow(Tree treeToWater); | is used to water a tree which has not produced any fruits yet. |
| public void waterFruitGrow(Tree treeToWater); | is used to water a tree that already has produced fruits. |

The subclasses of the *Tree* class have exactly the same members as their parent. However, again, they will be overloading particular members depending on the type of the *Tree* object.

----------------------------------------------------------------------------------------------------

**WateringCan Class**

****

As the name suggests, this is the class representing a watering can. Again, this class is a subclass of *Item*.

Attribute(s):

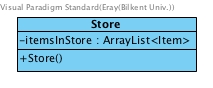
|  |  |
| --- | --- |
| private int canWaterLevel; | is the amount of the available water. |

Operation(s):

|  |  |
| --- | --- |
| public void water(Seed seedToWater); | calls the *water()* function of the received *Seed* instance and adjusts the *canWaterLevel*. |

----------------------------------------------------------------------------------------------------

**Store Class**

****

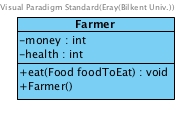
This is another child of the *GameObject* class and it represents the store in the game.

Attribute(s):

|  |  |
| --- | --- |
| private ArrayList<Item> itemsInStore; | keeps the *Item* objects that are available in the store. |

----------------------------------------------------------------------------------------------------

**Farmer Class**

****

This class represents the *Farmer*.

Attribute(s):

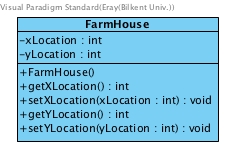
|  |  |
| --- | --- |
| private int money; | is the amount of *money* that the *Farmer* has. |
| private int health | represents the *health* level of the *Farmer*. |

Operation(s);

|  |  |
| --- | --- |
| public void eat(Food foodToEat); | given a *Food* instance via the parameter,  the Farmer’s *health* gets incremented according to that *Food*’s *healthContribution*. |

----------------------------------------------------------------------------------------------------

**FarmHouse Class**

****

This is the class illustrating the *FarmHouse*. Being a child of *GameObject*, the inherited members are to be used.

----------------------------------------------------------------------------------------------------

**Land Class**

As a child of the *GameObject* class, this class is the parent of *Grass* and *Pit*. This abstract class does not have any members except those inherited from the *GameObject* class.

----------------------------------------------------------------------------------------------------

**Grass Class**

This class illustrates a type of Land that is not suitable for planting.

----------------------------------------------------------------------------------------------------

**Pit Class**

In contrast with *Grass*, this class demonstrates the kind of *Land* available for planting.

Attribute(s):

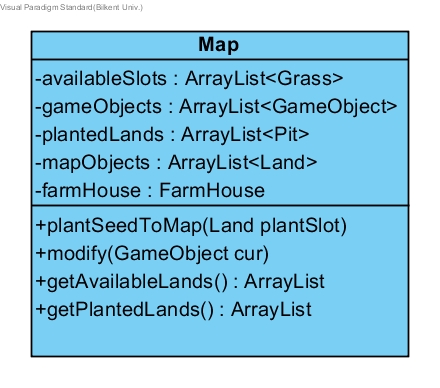
|  |  |
| --- | --- |
| private Seed sownPlant | keeps the *Seed* that is planted. |

Operation(s):

|  |  |
| --- | --- |
| public boolean plantSeed(Seed toPlant) | receives a *Seed* object to assign it to *sownPlant*. When planting is successful, true is returned. Otherwise, if the *Pit* is already planted for instance, this operation returns false. |
| public Food harvest() | when called on a planted *Pit* object, returns the *Food* of that *Seed*. Note that the *Seed* planted in that *Pit* object should have been already grown in order to be harvested. |
| public void waterSeeds() | is used to water the *Seed* instance on a *Pit* object. |

----------------------------------------------------------------------------------------------------

**Map Class**

****

This class is to keep objects in a grouped manner for better usage in *MapController*.

Attributes:

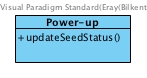
|  |  |
| --- | --- |
| private ArrayList< Grass> availableSlots | this ArrayList is to keep available *Grass* slots to change them into plantable *Pit* objects to plant *Seed* instances. |
| private ArrayList< GameObject> gameObjects | this list is to keep track of all objects by storing them. |
| private ArrayList< Pit> plantedLands | to keep track of where *Seed* objects have been planted. |
| private ArrayList< Land> mapObjects | to keep track of all Grass, *Pit*, in other words *Land* objects. |
| private FarmHouse farmHouse | this is an instance of the *FarmHouse* class which will be kept in the *Map* class. |

Operations:

|  |  |
| --- | --- |
| public void plantSeedToMap(Land plantSlot) | this operation is to plant *Seed* to specific *Land* slots, and modify ArrayLists. |
| public void modify(GameObject cur) | this operation is to change selected *GameObject* to *GameObject* *cur* by the commands from game logic. |
| public ArrayList<Grass> getAvailableLands() | is to return *availableSlots* to *MapController* class for using it in Game Logic. Especially, for separating planted lands from plantable lands. |
| public ArrayList<Pit> getPlantedLands() | this operation is to return *plantedLands* to *MapController* for a better understanding of the separation between plantable and planted lands like *getAvailableLands*(). |

----------------------------------------------------------------------------------------------------

**Power-up Class**

****

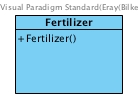
This is the parent of power-ups which are *Fertilizer* and *GMC* (genetically modified crop).

Operation(s):

|  |  |
| --- | --- |
| public void updateSeedStatus() | to update the *Seed* objects’ *status* after a power-up is applied. |

----------------------------------------------------------------------------------------------------

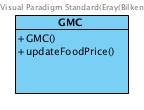
**Fertilizer Class**

****

This class, as a child of *Power-up*, operates on *Seed* objects. After being purchased from the store and applied on a planted slot, then that specific farm slot becomes fertile (meaning that the seeds would grow faster on that slot).

----------------------------------------------------------------------------------------------------

**GMC Class**

****

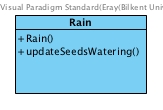
Standing for “Genetically Modified Crop,” this is another child of the *Power-up* class. After it is bought from the store, it is applied on *Seed* instances.

Operation(s):

|  |  |
| --- | --- |
| public void updateFoodPrice() | decrements the *storeSellingPrice* of *Food* instances that are generated from genetically modified *Seed* instances (as the *Food* becomes less healthy, it worths less). |

----------------------------------------------------------------------------------------------------

**Rain Class**

****

Being a child of the *GameObject* class, this class acts on *Seed* objects to change their water condition. More specifically, when this power-up is purchased once, the planted seeds may become watered at an unknown random time.

Operation(s):

|  |  |
| --- | --- |
| public void updateSeedsWatering() | is called at a random time to water the planted seeds. |

**Conclusion**

In this design report, we decided to add new features, classes after analysis report. Firstly, we have added new Entity objects including Potato, Tomato, Cherry, Raspberry, and Apple class as a subclass of Food class. Accordingly, TomatoSeed and PotatoSeed classes have been added. Besides, Tree class added with AppleTree, RaspberryTree, and CherryTree classes, which Tree class is superclass of these classes. In other words, we added ability to plant new trees to our farmland and new type of plants. Also, we added power-ups like GMO, Fertilizer, and Rain. These new features, will increase the entertainment level of game.