Tausif Islam Soil Maintenance:

**Class: Soil Matcher**

optimalSeed(desiredSoil)

FOR i=0 to numSeeds

LET wantedSeed = desiredSoil.preference(i);

IF (wantedSeed IS IN farm)   
 RETURN wantedSeed;

ENDIF

ENDFOR

RETURN NULL;// No seeds

optimalSoil(desiredSeed)

FOR i=0 to numSoils

LET wantedSeed = desiredSeed.preference(i);

IF(wantedSoil IS IN farm)   
 return wantedSoil;

ENDIF

RETURN NULL; //No seeds

**Class: Crops:** Only Getters

**Class:Fields:** Only Getters

**Class: AutoPlanter**

plantSeeds(Fields, Crops)

DO Planter moveTo Fields

DO Planter TILL Fields

DO Planter PLANT Crops

LET seedsHeld- -;

Rest Getters

**Class: SoilChecker**

**//Assume Checker is already in field**

checkCrop()

LET health = CHECK\_HEALTH

LET soilType = CHECK\_TYPE

LET needWater? = WATER\_COTNENT

LET needFertilizer? = FERTILIZER\_CONTENT

RETURN (health, soilType, needWater?, needFertilizer?)

checkSoil()

LET health = CHECK\_HEALTH

LET seedType = WHAT\_SEED

RETURN (health, seedType)

reportStats(WeeklyStats) //Do at end of day

PRINT (checkCrop(), checkSoil())

tellFertilize(AutoFertilizer)

NOTIFY(AutoFertilizer)

tellWater(AutoWater) {

NOTIFY(AutoWater)

}

Rest Getters

**Class AutoWater**

give(Fields,Crops)

IF !EMPTY

DO goto fields

DO water crops

ENDIF

LET amountWatered++;

LET needWater = isEmpty();

notify()

WHILE (ONLINE)

WAIT signal from checker

DO give water;

ENDWHILE

reportStats()

PRINT amountWatered, needWater

Rest Getters

**Class AutoFertilizer**

give(Fields,Crops)

IF !EMPTY

DO goto fields

DO fertilize crops

ENDIF

LET amountFertilized++;

LET needFertilizer = isEmpty();

notify()

WHILE (ONLINE)

WAIT signal from checker

DO give fertilizer;

ENDWHILE

reportStats()

PRINT amountFertilizer, needFertilizer

Rest Getters

**Class: WeeklyStats**

getState(**SoilChecker,AutoWater,AutoFertilizer** )

LET waterStat = reportStats(AutoWater)

LET fertilizeStat = reportStats(AutoFertilizer)

LET plantedStat = reportStats(SoilChecker)

Alex La - External Factors Pseudocode

**Class: Sensor**

detectChange()

IF (sensor detects change)

changeDetected = true

LET pest = null

LET disease = null

SWITCH (the detected change)

LET solution = solution grabbed from database

CASE (Pest)

` pest = new Pest(name of pest, crops where the sensors were

placed, the indicators that triggered the alarm, solution)

CASE (Disease)

Disease = new Disease(name of disease, crops where sensors

were Placed, Livestock where sensors were placed,

indicators that triggered the alarm,solution)

LET detection = new Detection(this, pest, disease)

notify(detection)

RETURN detection

notify(Detection detection)

IF (Detection.getPest() != null)

Detection.getPest().sendMessage()

IF (Detection.getDisease() != null)

Detection.getDisease().sendMessage()

**Class: User**

pollSensors()

LET timer = a new timer that activates every 5 seconds and calls the lines

FOR every sensor in getSensors()

detectChange()

checkDisaster()

IF (external API call finds that there is a natural disaster coming)

RETURN new NaturalDisaster(name) //The other attributes will be determined

// by a database

RETURN null

**Class: Solution**

doSolution()

FOR every id in getIds

If (device with id’s method == getMethod)

Activate device associated with id

IF (there was an error in one of the devices)

RETURN false

RETURN true

**Class NaturalDisaster**

sendMessage()

RETURN “A”+ getName() + “is coming at” + getDate().toString()

**Class Pest**

sendMessage()

RETURN getName() + “were found at the location of +” getCrops().toString().

“The Indicators were” + getIndicators

**Class Disease**

sendMessage()

RETURN getName() + “were found at the location of +” getCrops().toString().

“and/or” + getLivestock().toString(). “The Indicators were” + getIndicators

## 

## Kojo Poku - Tools & Equipment

### Class Inventory

**Method updateInventory(newQuantity):**

**If** newQuantity is a non-negative integer:

**LET** quantity = newQuantity to to reflect the updated count

**If** newQuantity == 0:

Update status to "Out of Stock"

**Else If** newQuantity is < 5 Or specified threshold:

Update status to "Low Stock"

**Else:**

Update status to "In Stock"

**Return** a confirmation message “Inventory successfully updated”

**Else:**

**Return** an error message “invalid quantity update attempt”

**Method checkAvailability():**

**If** quantity is greater than 0:

**Return**  “item is available”

**Else**:

**Return** “Item is not available”

**Method relocateEquipment(newStatus):**

Change the current status to newStatus to reflect the new state of the equipment

**If** newStatus == "In Transit”: which indicates a location change

**Print** "Initiating relocation process for equipment."

**Print** "Equipment ID [equipmentID] is now in transit."

Perform other necessary steps to update internal records use print statements

**Else**:

**Print** "Updating equipment status to [newStatus]."

**Return** "Equipment status successfully updated to [newStatus]."

**END**

### 

### 

### Class Inventory

**Method updateInventory(newQuantity)**

**If** newQuantity >= 0 **Then**

**SET** quantity = newQuantity

**If** newQuantity == 0 **Then**

**SET** status = 'Out of Stock'

**Else If** newQuantity < 5 **Then**  Assuming 5 is the threshold for low stock

**SET** status = 'Low Stock'

**Else**

**SET** status = 'In Stock'

End If

**Return** 'Inventory successfully updated.'

**Else**

**Return** 'Invalid quantity update attempt.'

**Method checkAvailability()**

**If** quantity > 0 **Then**

**Return** 'Item is available.'

**Else**

**Return** 'Item is not available.'

**Method relocateEquipment(newStatus)**

**LET** status = newStatus

**If** newStatus == 'In Transit' **Then**

**Print** 'Initiating relocation process for equipment.'

**Print** 'Equipment ID ' + equipmentID + ' is now in transit.'

**Else**

**Print** 'Updating equipment status to ' + newStatus + '.'

End If

**Return** 'Equipment status successfully updated to ' + newStatus + '.'

**END**

### Class RepairHistory

Class of printing information from getter methods

// Methods

**Method recordRepair(newHistoryID, associatedEquipmentID, repairDetails, repairDate)**

**LET**  historyID = newHistoryID

**LET** equipmentID = associatedEquipmentID

**LET**  details = repairDetails

**LET** dateOfRepair = repairDate

**Print** 'Repair history for equipment ID ' + equipmentID + ' recorded on ' + dateOfRepair + '.'

**Method getRepairHistory(equipmentID) >> getter method**

**Method getLastRepairDate(equipmentID) >> getter method**

**END**

### Class MaintenanceRequest

**Method createRequest(requestID, equipmentID, dataRequested, urgency):**

**Set** maintenanceRequestID = requestID

**Set** this.request = equipmentID

**Set** requestedData = dataRequested

**Set** urgencyLevel = urgency

**Set** status ="Pending"

**Return** "Maintenance request created successfully with ID: " + requestID

**Method updateRequestStatus(requestID, newStatus):**

**If** newStatus is one of ["Pending", "InProgress", "Completed", "Cancelled"]:

**Set** status of request with requestID = newStatus

**Print** "Maintenance request " + requestID + " status updated to " + newStatus

**Return** "Request status updated successfully."

**Else**:

**Return** "Invalid status update attempt for request " + requestID

**Method escalateRequest(requestID):**

Retrieve request with requestID

**If** urgencyLevel is not "High": **Then**

**SET** urgencyLevel = "High"

**Print** "Maintenance request " + requestID + " has been escalated to high priority."

**Return** "Request escalated successfully."

**Else**:

**Return** "Request is already at the highest priority level."

**END**

### Class AuditLog

**Method recordChange(logID, affectedID, timestamp, changeDescription, changedByUserID):**

**Set** auditLogID = logID

Record the ID of the affected item as affectedID

**Set** the logTimestamp = timestamp

Record changeDetails as changeDescription

Record the ID of the user who made the change as changedByUserID

**Print** "Change for " + affectedID + " recorded by user " + changedByUserID + " at " + timestamp + "."

**Return** "Change recorded successfully."

**Method retrieveLogs(filterCriteria):**

Retrieve all logs that match filterCriteria from the log storage

**If** logs are found: **Then**

**Return** the retrieved logs

**Else**:

**Return** "No logs found matching the criteria."

**Method filterLogsByClass(classType):**

Retrieve all logs where the affected item is of type classType

**If** logs are found: **Then**

**Return** the filtered logs

**Else**:

**Return** "No logs found for items of type " + classType + "."

**END**

### Class SafetyCheck

**Method scheduleInspection(safetyCheckID, equipmentID, inspectionDate):**

**Set** checkID = safetyCheckID

Associate this check with equipmentID

**Set** the scheduledDate = inspectionDate

**Set** status = "Scheduled"

**Return** "Safety inspection scheduled for " + inspectionDate + " on equipment ID " + equipmentID + "."

**Method recordInspectionOutcome(safetyCheckID, isSafe):**

Retrieve safety check with safetyCheckID

**Set** safetyStatus = isSafe

**If** isSafe: **Then**

**Print** "Safety check " + safetyCheckID + " for equipment ID " + equipmentID + " passed."

**Return** "Safety status recorded as safe."

**Else**:

**Print** "Safety check " + safetyCheckID + " for equipment ID " + equipmentID + " failed."

**Return** "Safety status recorded as not safe."

**END**

Chris Lam - Livestock/Crop Management Pseudocode

**HealthTracker class**

**monitorTemperature() -> getter**

**checkSleepHabits() ->getter**

**checkHeartRate() ->getter**

**checkEatingHabits()->getter**

**checkSickness()**

IF(livestockTemp > 104)

bool sick = true;

**YieldPredictor class**

**howManySick()**

count = 0

FOR from 0 to all livestock in system

IF(livestock.sick == true)

count++

return count

**howManyMature()**

count = 0

FOR from 0 to all livestock in system

IF(checkMaturity(livestock) == true)

count++

return count

**AutoFeeder class**

**setFeedInterval() ->setter**

**setFeedAmount() ->setter**

**checkFeedLevel() ->getter**

**sendRefillNotification(notification)**

new Notification(“Your feeder is running low, please refill ASAP”,time,date);

**MaturityTracker class**

**checkMaturity() ->getter**

**checkEggs() ->getter**

**sendMaturityNotif(notification)**

new Notification(“There are x livestock mature out of y as of right now”,time,date);

**HealthSolution class**

**checkSickness(livestock) ->getter**

**findSolution(sickness)**

IF(checkSickness(livestock)==true)

new Notification(“Your livestock is sick, here are some potential health solutions: etc…”,time,date);

Zain Mirza - Sensors and Data Analysis Tools

**ReportGenerator class:**

graphOverTime(type: String, style: String, duration: int)

IF type == “tempandhumid”

LET tempReadings = list of temperature readings stored as a global variable

LET humidReadings = list of humidity readings stored as a global variable

IF style == “Line”

FOR int i = 0 to duration

plot i, tempReadings[i]

Plot i, humidReadings[i]

END

ENDIF

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

textSummary()

LET summary = “Temperature exceeded thresholds during the following time periods: ”

LET tempReadings = list of temperature readings stored as a global variable

LET threshHigh = High temperature threshold set in constructor

LET threshLow = Low temperature threshold set in constructor

FOR int i = 0 to length of tempReadings

IF tempReadings[i] > threshHigh || tempReadings < threshLow

Summary = Summary + i + “ “

ENDIF

END

return summary

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

calculateEfficiency()

LET actualTotal = 0

Let theoreticalTotal = 0

FOR int day = 0 to length of numDays, a count for the number of days in the season so far

actualTotal += yieldScorer(“actual”) //yieldScorer is an external library that scores yield //based on a complex algorithm that takes into account numerous farm factors

theoreticalTotal += yieldScorer(“total”)

END

return actualTotal/theoreticalTotal

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

calculateEffectiveness()

return calculateEfficiency() //For now this method is the same as efficiency, but it was //provisioned in design I and may have more usefulness as the app develops

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

gernerateMaintenanceLog(toolList: String List of Lists)

LET log = “”

FOR int i = 0 to toolList.length

log = log + “Tool name: “ + toolList[i][0] + “\n”

log = log + “Last repaired: “ + toolList[i][1] + “\n”

log = log + “Condition score: “ + toolList[i][2] + “\n”

END

return log

**LocalStore class:**

checkAndSync()

LET connection = a boolean keeping track of whether the connection between the measurement device and the wider network is available

IF connection == true

sync() //External API call to sync local data with remote repository

else

saveLocal() //local method call to record data in local database

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

saveLocal()

statement.executeUpdate("INSERT INTO \_\_\_\_\_\_ " + "VALUES (\_\_, \_\_, \_\_\_\_, \_\_\_\_, \_\_\_\_)");

**Thermometer, Barometer, Location, and Tool have only setter and getter methods.**

## Gerald Akalugwu - Harvest Management

### Class: Crop

**get\_average\_market\_price**() {

**if** length(market\_prices) == 0:

return "No market prices available"

total\_price = 0

**for** price **in** market\_prices:

total\_price = total\_price + price

average\_price = total\_price / length(market\_prices)

**return** average\_price

}

### Class: Crops

**get\_yield\_percentage**() {

**return** (current\_yield / max\_yield) \* 100

}

**get\_estimated\_market \_value**() {

**return** this.crop.get\_average\_market\_price() \* this.weight\_harvested()

}

### Class: Auto Harvester

**harvest**(SoilPatch soil\_patch){

**if** (this.getAvailability() == false) {

return -1; // returns -1 meaning the function wasn’t carried out

}

**if** (soil\_patch.get\_current\_crop() == null) {

return -1; // returns -1 meaning the function wasn’t carried out

}

return 0; // returns 0 meaning the function was carried out

}

**assign\_task**(Task task ){

**if** (this.getAvailability() == false) {

return -1; // returns -1 meaning the function wasn’t carried out

}

**for** (int i =0; i < soil\_patch.size(); i++) {

soil\_patch.atLocation(i).harvest();

}

return 0; // returns 0 meaning the function was carried out

}

### Class: Task

**trigger\_auto\_harvest**( SoilPatch soil\_patch){

**while** (soil\_patch.hasNext() && !(soil.\_patch.next.isOccupied())) {

harvest(soil\_patch.next);

}

**return** 0 // returns 0 meaning the function was carried out

}

**notify\_farmer**(farmer farmers[], SoilPatch soil\_patch){

Farmer currentAvailableFarmer = new Farmer();

**for** farmer **in** farmers {

**if** (farmer.getTask() == **null**) {

farmer.addNotification(this) // notifies all available farmers found

}

}

return 0 // returns 0 meaning the function was carried out

}

### Class: Soil Patch

**get\_location**(int x\_cord, int y\_cord){

**return** “Soil Patch is located at” + **x\_cord** + “, “ + **y\_cord**

}

### Class: Crop Rotation

**Class task addToTask**(int x\_cord, int y\_cord) { // returns task

Task task = new Task();

` task.crop\_rotation(this); // inserts crop rotation into task

task.is\_crop\_rotation() // returns true;

task.description = “A crop rotation replacing + $this.previous\_crop with

$this.current\_crop would require ”; // builds description string

**for each** nutrient **in** this**.**nutrient\_required:

task.description += $nutrient “, ”

task.description += $this.amount\_of\_water, and + $this.amount\_of\_fertilizer.;

**return** task

}