VERSION [1.0 - FINAL]

SMU
SINGAPORE MANAGEMENT UNIVERSITY

IS201 OBJECT ORIENTED APPLICATION DESIGN – GROUP PROJECT: FARM CITY

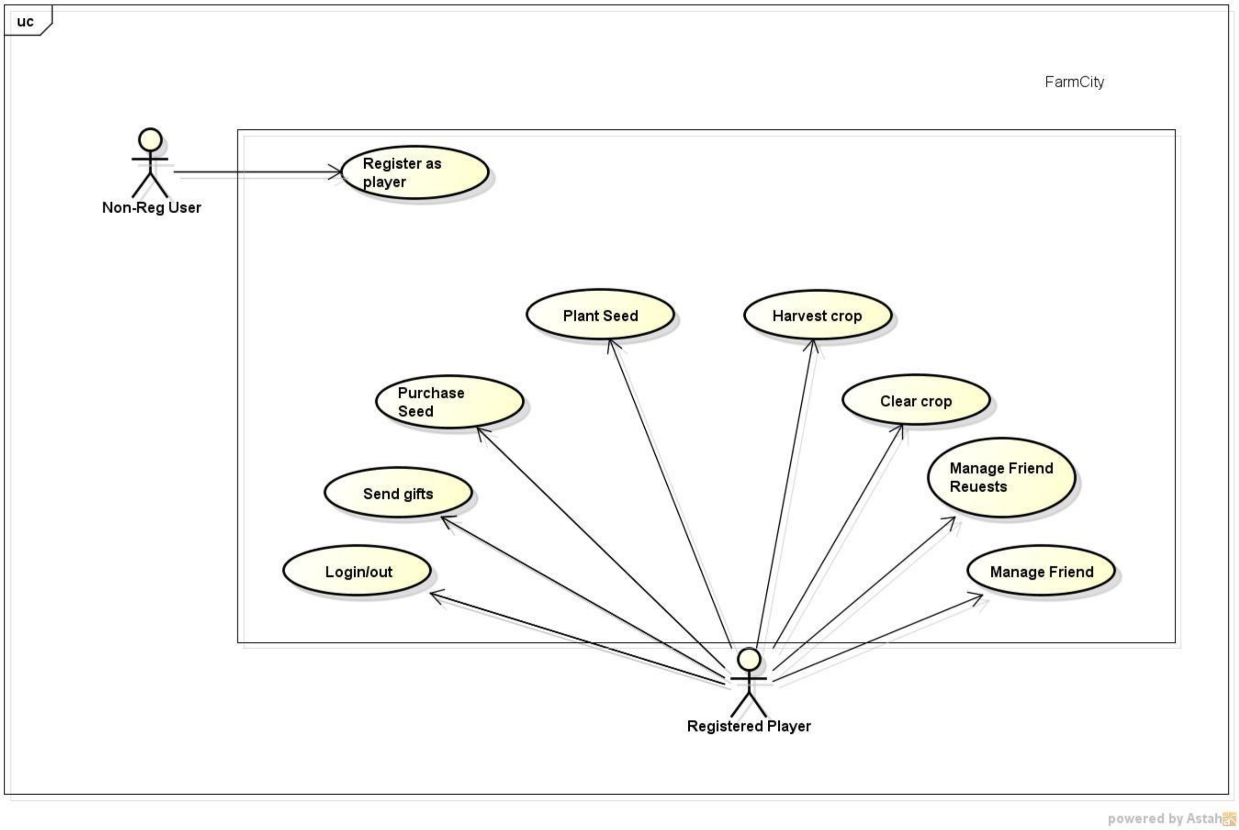
PRESENTED BY:
XU JIA WEN
SITI NORDIYANA BTE FUAD
LAM WAN TING

Table of Content

2. MISCELLANEOUS

3. SCREENSHOTS

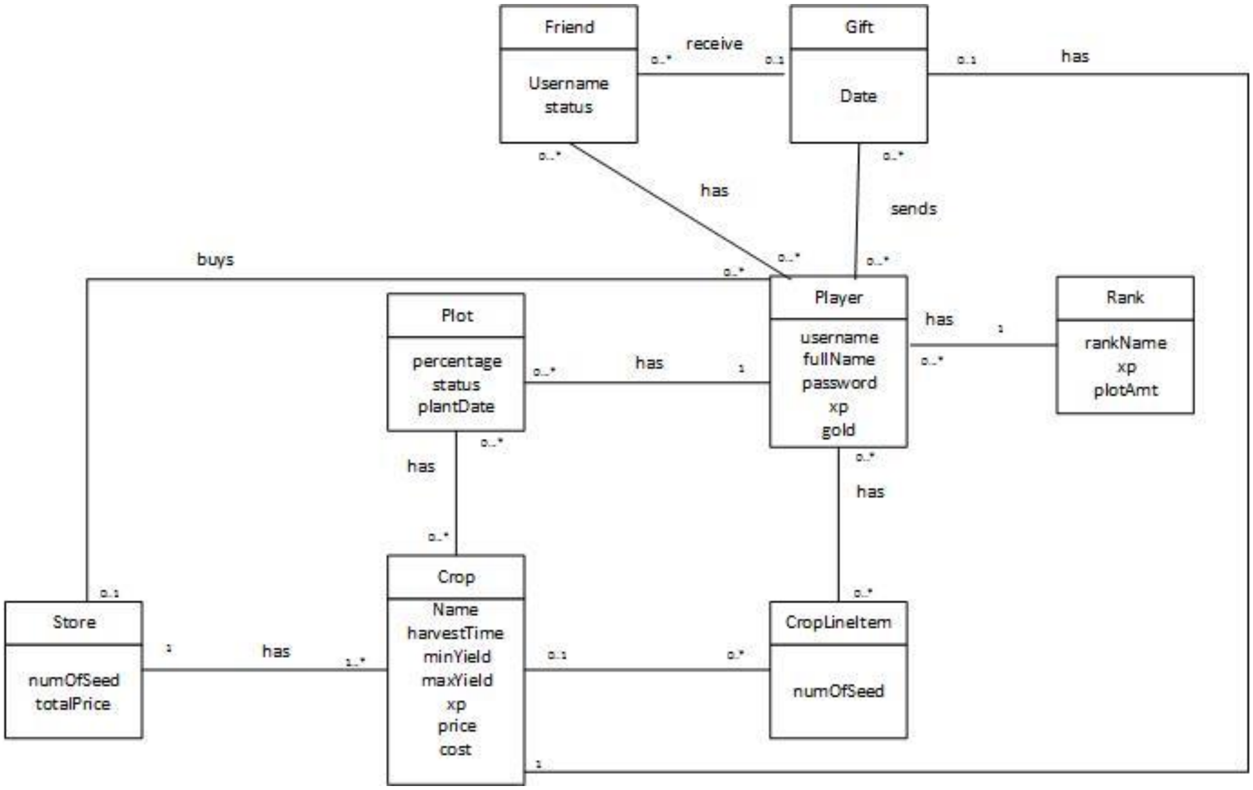
USE CASE DIAGRAM



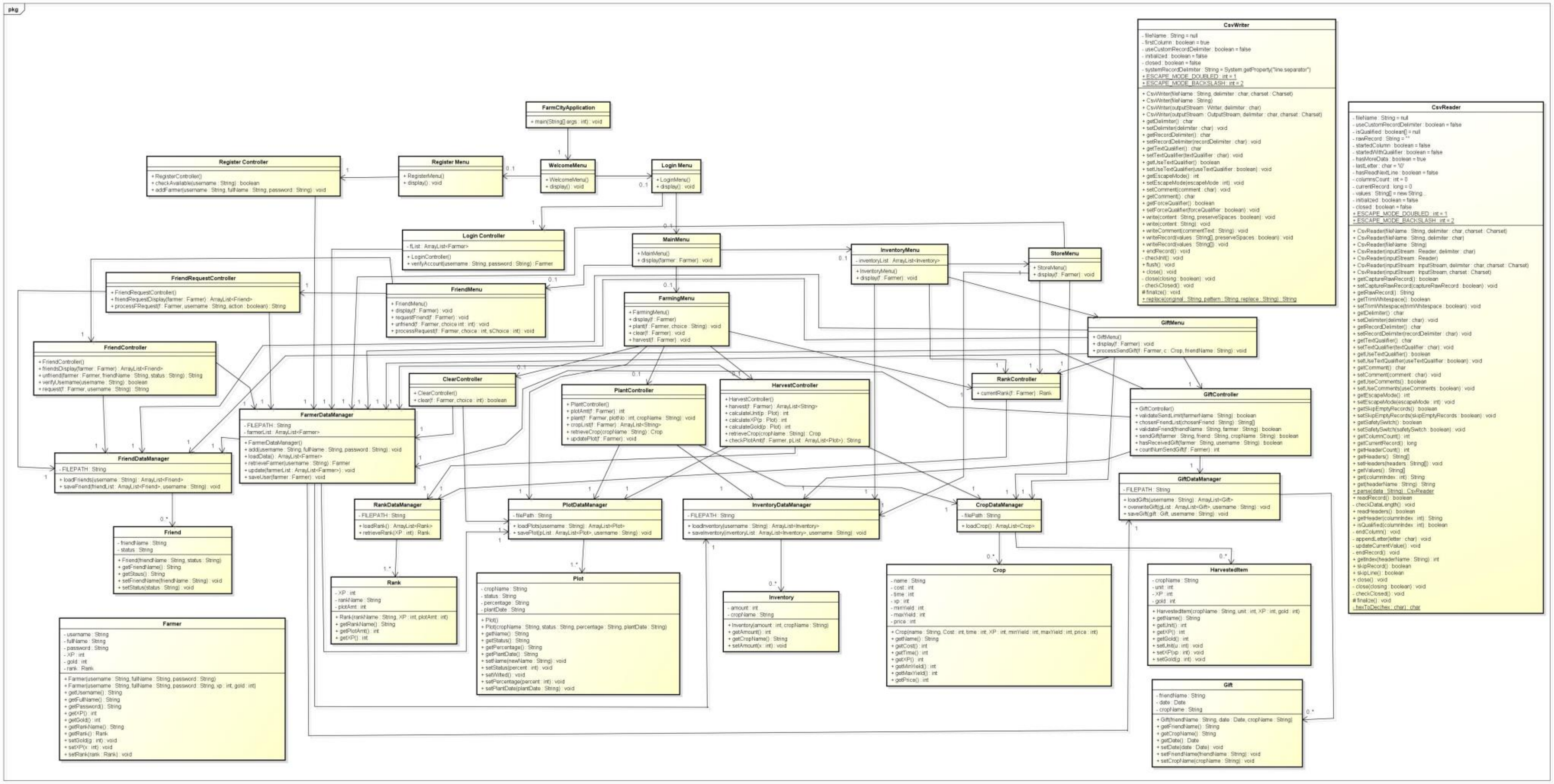
USE CASE DESCRIPTION

1. [Register as Player] A non-registered user may register to become a registered player/farmer in the game.
2. [Log In/Out] A registered player is allowed to log in or out of the game.
3. [Send Gifts] A registered player may send gifts to his/her friends if he has not met the sending limit.
4. [Purchase Seed] A registered player may purchase seed at the store if he has sufficient gold.
5. [Plant Seed] A registered player can decide the crop to plant at a specific plot if he has an empty land and his inventory is not empty.
6. [Harvest Crop] A registered user can harvest his crops when the seeds have fully grown to 100%.
7. [Clear Crop] A registered user may clear plots of land if it contains a wilted crop.
8. [Manage Friend Request] A registered user may accept or reject a friend request that he received from other users.
9. [Manage friends] A registered user may unfriend a friend or send a request to other users.

DOMAIN DIAGRAM



CLASS DIAGRAM


pkg
FarmCityApplication
+ main(String[] args : int) : void
WelcomeMenu
+ WelcomeMenu()
+ display() : void
Register Menu
+ RegisterMenu()
+ display() : void
Register Controller
+ RegisterController()
+ checkAvailable(username : String) : boolean
+ addFarmer(username : String, fullName : String, password : String) : void
Login Menu
+ LoginMenu()
+ display() : void
Login Controller
- fList : ArrayList<Farmer>
+ LoginController()
+ verifyAccount(username : String, password : String) : Farmer
MainMenu
+ MainMenu()
+ display(farmer : Farmer) : void
FriendRequestController
+ FriendRequestController()
+ friendRequestDisplay(farmer : Farmer) : ArrayList<Friend>
+ processFRequest(f : Farmer, username : String, action : boolean) : String
FriendMenu
+ FriendMenu()
+ display(f : Farmer) : void
+ requestFriend(f : Farmer) : void
+ unfriend(f : Farmer, choice int : int) : void
+ processRequest(f : Farmer, choice : int, sChoice : int) : void
FarmingMenu
+ FarmingMenu()
+ display(f : Farmer)
+ plant(f : Farmer, choice : String) : void
+ clean(f : Farmer) : void
+ harvest(f : Farmer) : void
FriendController
+ FriendController()
+ friendsDisplay(farmer : Farmer) : ArrayList<Friend>
+ unfriend(farmer : Farmer, friendName : String, status : String) : String
+ verifyUsername(username : String) : boolean
+ request(f : Farmer, username : String) : String
ClearController
+ ClearController()
+ clear(f : Farmer, choice : int) : boolean
PlantController
+ PlantController()
+ plotAmt(f : Farmer) : int
+ plant(f : Farmer, plotNo : int, cropName : String) : void
+ cropList(f : Farmer) : ArrayList<String>
+ retrieveCrop(cropName : String) : Crop
+ updatePlot(f : Farmer) : void
HarvestController
+ HarvestController()
+ harvest(f : Farmer) : ArrayList<String>
+ calculateUnit(p : Plot) : int
+ calculateXP(p : Plot) : int
+ calculateGold(p : Plot) : int
+ retrieveCrop(cropName : String) : Crop
+ checkPlotAmt(f : Farmer, pList : ArrayList<Plot>) : String
InventoryMenu
- inventoryList : ArrayList<Inventory>
+ InventoryMenu()
+ display(f : Farmer) : void
StoreMenu
+ StoreMenu()
+ display(f : Farmer) : void
GiftMenu
+ GiftMenu()
+ display(f : Farmer) : void
+ processSendGift(f : Farmer, c : Crop, friendName : String) : void
RankController
+ currentRank(f : Farmer) : Rank
GiftController
+ GiftController()
+ validateSendLimit(farmerName : String) : boolean
+ chosenFriendList(chosenFriend : String) : String[]
+ validateFriend(friendName : String, farmer : String) : boolean
+ sendGift(farmer : String, friend : String, cropName : String) : boolean
+ hasReceivedGift(farmer : String, username : String) : boolean
+ countNumSendGift(f : Farmer) : int
FarmerDataManager
- FILEPATH : String
- farmerList : ArrayList<Farmer>
+ FarmerDataManager()
+ add(username : String, fullName : String, password : String) : void
+ loadData() : ArrayList<Farmer>
+ retrieveFarmer(username : String) : Farmer
+ update(farmerList : ArrayList<Farmer>) : void
+ saveUser(farmer : Farmer) : void
FriendDataManager
- FILEPATH : String
+ loadFriends(username : String) : ArrayList<Friend>
+ saveFriend(friendList : ArrayList<Friend>, username : String) : void
Friend
- friendName : String
- status : String
+ Friend(friendName : String, status : String)
+ getFriendName() : String
+ getStaus() : String
+ setFriendName(friendName : String) : void
+ setStatus(status : String) : void
Farmer
- username : String
- fullName : String
- password : String
- XP : int
- gold : int
- rank : Rank
+ Farmer(username : String, fullName : String, password : String)
+ Farmer(username : String, fullName : String, password : String, xp : int, gold : int)
+ getUsername() : String
+ getFullName() : String
+ getPassword() : String
+ getXP() : int
+ getGold() : int
+ getRankName() : String
+ getRank() : Rank
+ setGold(g : int) : void
+ setXP(x : int) : void
+ setRank(rank : Rank) : void
RankDataManager
- FILEPATH : String
+ loadRank() : ArrayList<Rank>
+ retrieveRank(XP : int) : Rank
Rank
- XP : int
- rankName : String
- plotAmt : int
+ Rank(rankName : String, XP : int, plotAmt : int)
+ getRankName() : String
+ getPlotAmt() : int
+ getXP() : int
PlotDataManager
- filePath : String
+ loadPlots(username : String) : ArrayList<Plot>
+ savePlot(pList : ArrayList<Plot>, username : String) : void
Plot
- cropName : String
- status : String
- percentage : String
- plantDate : String
+ Plot()
+ Plot(cropName : String, status : String, percentage : String, plantDate : String)
+ getName() : String
+ getStatus() : String
+ getPercentage() : String
+ getPlantDate() : String
+ setName(newName : String) : void
+ setStatus(percent : int) : void
+ setWilted() : void
+ setPercentage(percent : int) : void
+ setPlantDate(plantDate : String) : void
InventoryDataManager
- FILEPATH : String
+ loadInventory(username : String) : ArrayList<Inventory>
+ saveInventory(inventoryList : ArrayList<Inventory>, username : String) : void
Inventory
- amount : int
- cropName : String
+ Inventory(amount : int, cropName : String)
+ getAmount() : int
+ getCropName() : String
+ setAmount(x : int) : void
CropDataManager
- filePath : String
+ loadCrop() : ArrayList<Crop>
Crop
- name : String
- cost : int
- time : int
- xp : int
- minYield : int
- maxYield : int
- price : int
+ Crop(name : String, Cost : int, time : int, XP : int, minYield : int, maxYield : int, price : int)
+ getName() : String
+ getCost() : int
+ getTime() : int
+ getXP() : int
+ getMinYield() : int
+ getMaxYield() : int
+ getPrice() : int
GiftDataManager
- FILEPATH : String
+ loadGifts(username : String) : ArrayList<Gift>
+ overwriteGift(gList : ArrayList<Gift>, username : String) : void
+ saveGift(gift : Gift, username : String) : void
HarvestedItem
- cropName : String
- unit : int
- XP : int
- gold : int
+ HarvestedItem(cropName : String, unit : int, XP : int, gold : int)
+ getName() : String
+ getUnit() : int
+ getXP() : int
+ getGold() : int
+ setUnit(u : int) : void
+ setXP(xp : int) : void
+ setGold(g : int) : void
Gift
- friendName : String
- date : Date
- cropName : String
+ Gift(friendName : String, date : Date, cropName : String)
+ getFriendName() : String
+ getCropName() : String
+ getDate() : Date
+ setDate(date : Date) : void
+ setFriendName(friendName : String) : void
+ setCropName(cropName : String) : void
CsvWriter
- fileName : String = null
- firstColumn : boolean = true
- useCustomRecordDelimiter : boolean = false
- initialized : boolean = false
- closed : boolean = false
- systemRecordDelimiter : String = System.getProperty("line.separator")
+ ESCAPE_MODE_DOUBLED : int = 1
+ ESCAPE_MODE_BACKSLASH : int = 2
+ CsvWriter(fileName : String, delimiter : char, charset : Charset)
+ CsvWriter(fileName : String)
+ CsvWriter(outputStream : Writer, delimiter : char)
+ CsvWriter(outputStream : OutputStream, delimiter : char, charset : Charset)
+ getDelimiter() : char
+ setDelimiter(delimiter : char) : void
+ getRecordDelimiter() : char
+ setRecordDelimiter(recordDelimiter : char) : void
+ getTextQualifier() : char
+ setTextQualifier(textQualifier : char) : void
+ getUseTextQualifier() : boolean
+ setUseTextQualifier(useTextQualifier : boolean) : void
+ getEscapeMode() : int
+ setEscapeMode(escapeMode : int) : void
+ setComment(comment : char) : void
+ getComment() : char
+ getForceQualifier() : boolean
+ setForceQualifier(forceQualifier : boolean) : void
+ write(content : String, preserveSpaces : boolean) : void
+ write(content : String) : void
+ writeComment(commentText : String) : void
+ writeRecord(values : String[], preserveSpaces : boolean) : void
+ writeRecord(values : String[]) : void
+ endRecord() : void
- checkInit() : void
+ flush() : void
+ close() : void
- close(closing : boolean) : void
- checkClosed() : void
# finalize() : void
+ replace(original : String, pattern : String, replace : String) : String
CsvReader
- fileName : String = null
- useCustomRecordDelimiter : boolean = false
- isQualified : boolean[] = null
- rawRecord : String = ""
- startedColumn : boolean = false
- startedWithQualifier : boolean = false
- hasMoreData : boolean = true
- lastLetter : char = '\0'
- hasReadNextLine : boolean = false
- columnsCount : int = 0
- currentRecord : long = 0
- values : String[] = new String...
- initialized : boolean = false
- closed : boolean = false
+ ESCAPE_MODE_DOUBLED : int = 1
+ ESCAPE_MODE_BACKSLASH : int = 2
+ CsvReader(fileName : String, delimiter : char, charset : Charset)
+ CsvReader(fileName : String, delimiter : char)
+ CsvReader(fileName : String)
+ CsvReader(inputStream : Reader, delimiter : char)
+ CsvReader(inputStream : Reader)
+ CsvReader(inputStream : InputStream, delimiter : char, charset : Charset)
+ CsvReader(inputStream : InputStream, charset : Charset)
+ getCaptureRawRecord() : boolean
+ setCaptureRawRecord(captureRawRecord : boolean) : void
+ getRawRecord() : String
+ getTrimWhitespace() : boolean
+ setTrimWhitespace(trimWhitespace : boolean) : void
+ getDelimiter() : char
+ setDelimiter(delimiter : char) : void
+ getRecordDelimiter() : char
+ setRecordDelimiter(recordDelimiter : char) : void
+ getTextQualifier() : char
+ setTextQualifier(textQualifier : char) : void
+ getUseTextQualifier() : boolean
+ setUseTextQualifier(useTextQualifier : boolean) : void
+ getComment() : char
+ setComment(comment : char) : void
+ getUseComments() : boolean
+ setUseComments(useComments : boolean) : void
+ getEscapeMode() : int
+ setEscapeMode(escapeMode : int) : void
+ getSkipEmptyRecords() : boolean
+ setSkipEmptyRecords(skipEmptyRecords : boolean) : void
+ getSafetySwitch() : boolean
+ setSafetySwitch(safetySwitch : boolean) : void
+ getColumnCount() : int
+ getCurrentRecord() : long
+ getHeaderCount() : int
+ getHeaders() : String[]
+ setHeaders(headers : String[]) : void
+ getValues() : String[]
+ get(columnIndex : int) : String
+ get(headerName : String) : String
+ parse(data : String) : CsvReader
+ readRecord() : boolean
- checkDataLength() : void
+ readHeaders() : boolean
+ getHeader(columnIndex : int) : String
+ isQualified(columnIndex : int) : boolean
- endColumn() : void
- appendLetter(letter : char) : void
- updateCurrentValue() : void
- endRecord() : void
+ getIndex(headerName : String) : int
+ skipRecord() : boolean
+ skipLine() : boolean
+ close() : void
- close(closing : boolean) : void
- checkClosed() : void
# finalize() : void
- hexToDec(hex : char) : char

USE CASE SCENARIOS – REGISTER AS PLAYER

ACTOR

Unregistered User

PRE-CONDITIONS

- User does not have an existing account

MAIN FLOW OF EVENTS

1. This use case begins when an unregistered user chooses the register option at the start-up menu
2. The system will prompt the user for username.
3. User will submit a username.
4. The system will verify the username.
5. The system will prompt for user’s full name.
6. User will submit his full name.
7. The system will verify the full name submitted.
8. The system will prompt user for a password.
9. The user will submit a password.
10. The system will prompt user to confirm password.
11. The user will submit the same password as the previous password.
12. The system will verify that the two passwords submitted match.
13. The system will create a new farmer object.
14. The use case will end

ALTERNATE FLOW OF EVENTS

4a. The username submitted contains characters that are not alphanumeric

i. The system will display an error message
ii. The user will be prompted to submit a new username
iii. If username contains alphanumeric characters only, continue to step 5

4b. The username submitted exist

i. The system will display an error message
ii. The user will be prompted to submit a new username
iii. If username does not exists, continue to step 5

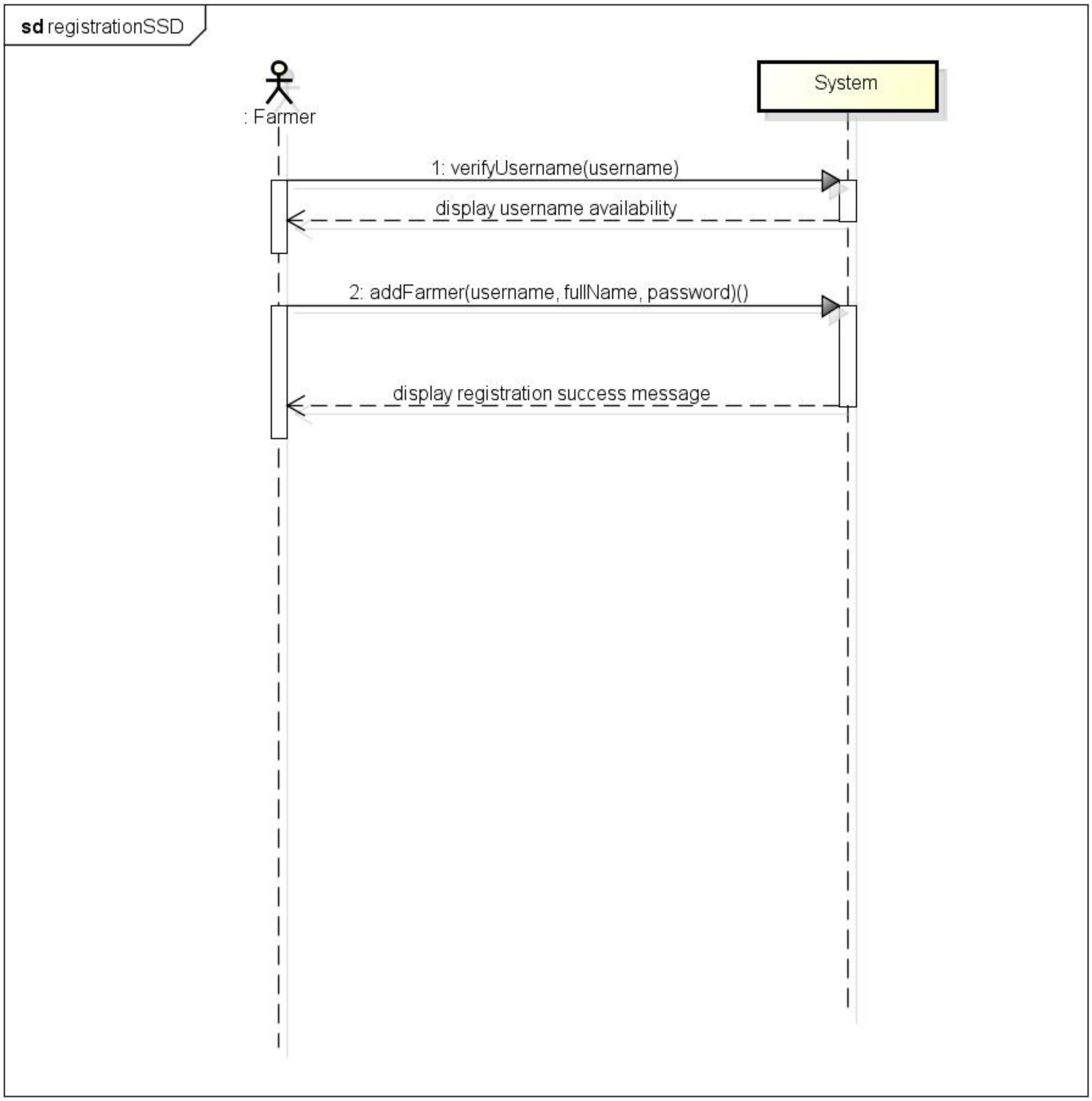
7a. The full name submitted contains characters that are not alphanumeric

i. The system will display an error message
ii. The user will be prompted to submit a new full name
iii. If the full name does not contain alphanumeric characters, continue to step 8

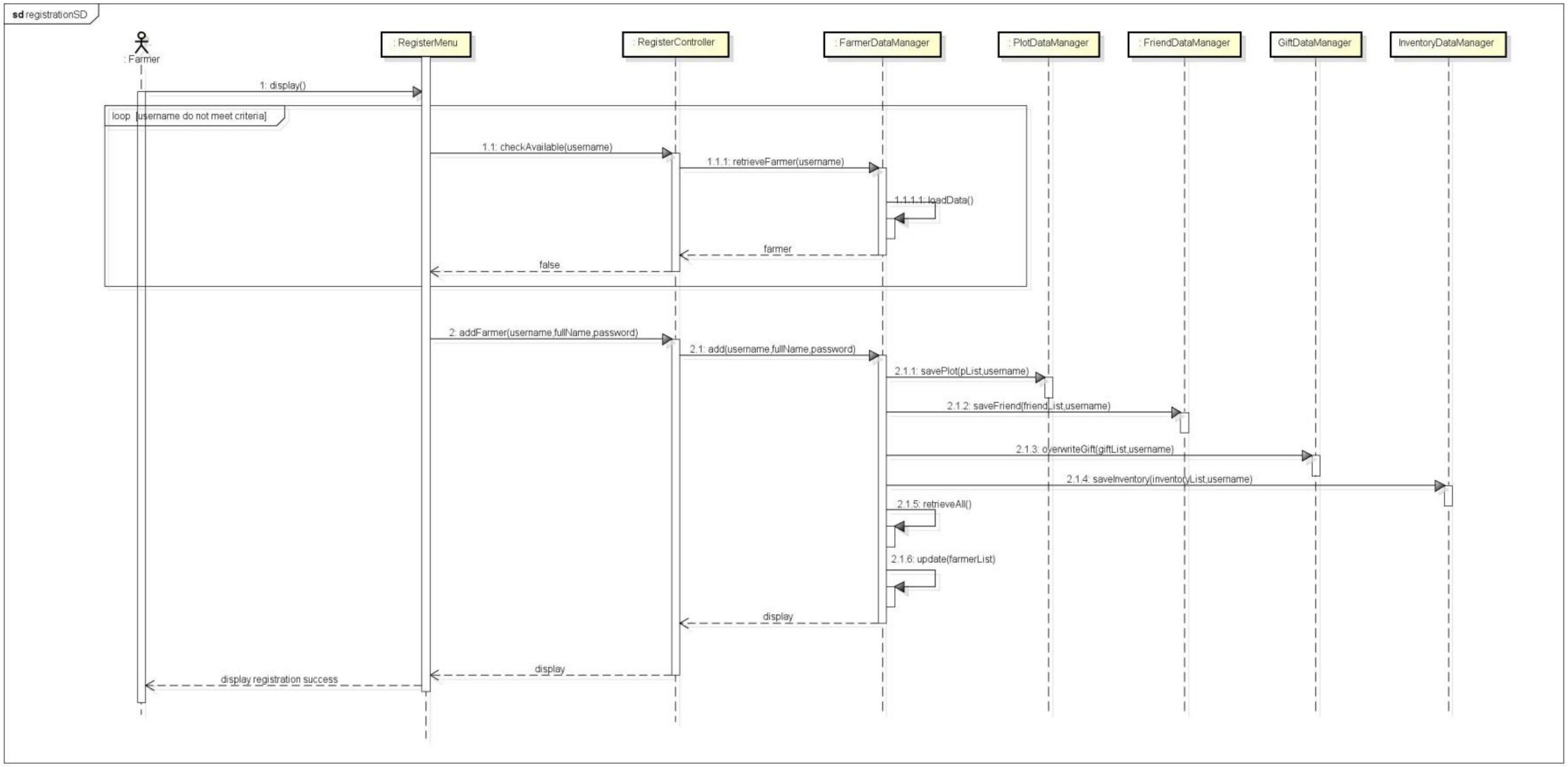
12a. The password submitted at step 9 and 11 does not match

i. The system will display an error message
ii. The user will be prompted to submit a new password and use case will continue at step 10

SYSTEM SEQUENCE DIAGRAM – REGISTER AS A PLAYER


sd registrationSSD
: Farmer
System
1: verifyUsername(username)
display username availability
2: addFarmer(username, fullName, password)()
display registration success message

SEQUENCE DIAGRAM – REGISTER AS PLAYER


sd registrationSD
: Farmer
: RegisterMenu
: RegisterController
: FarmerDataManager
: PlotDataManager
: FriendDataManager
GiftDataManager
InventoryDataManager
1: display()
loop [username do not meet criteria]
1.1: checkAvailable(username)
1.1.1: retrieveFarmer(username)
1.1.1.1: loadData()
farmer
false
2: addFarmer(username,fullName,password)
2.1: add(username,fullName,password)
2.1.1: savePlot(pList,username)
2.1.2: saveFriend(friendList,username)
2.1.3: overwriteGift(giftList,username)
2.1.4: saveInventory(inventoryList,username)
2.1.5: retrieveAll()
2.1.6: update(farmerList)
display
display
display registration success

USE CASE SCENARIOS LOG IN/OUT

ACTOR

Farmer

PRE-CONDITIONS

- User has an existing account

MAIN FLOW OF EVENTS

1. The use case will begin when the user selects the login option at the start-up menu
2. The system will display the login menu
3. The system will prompt the user for his username
4. The user will submit a username
5. The system will prompt the user for his password
6. The user will submit a password
7. The system will verify the password and username
8. The user will be given access to the app
9. The use case will end

ALTERNATE FLOW OF EVENTS

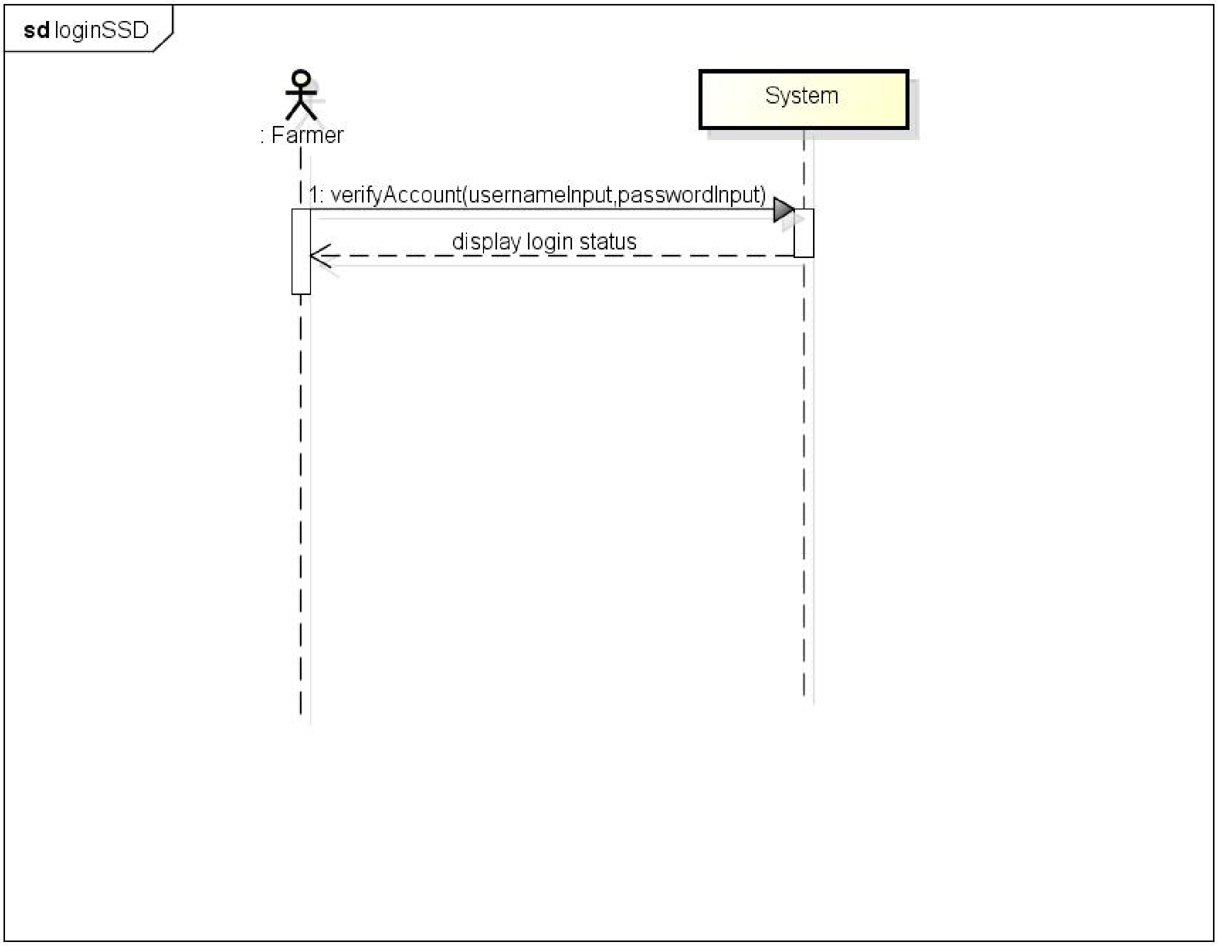
6a. The user submits an incorrect password or username

i. The system will display an error message
ii. The use case will end

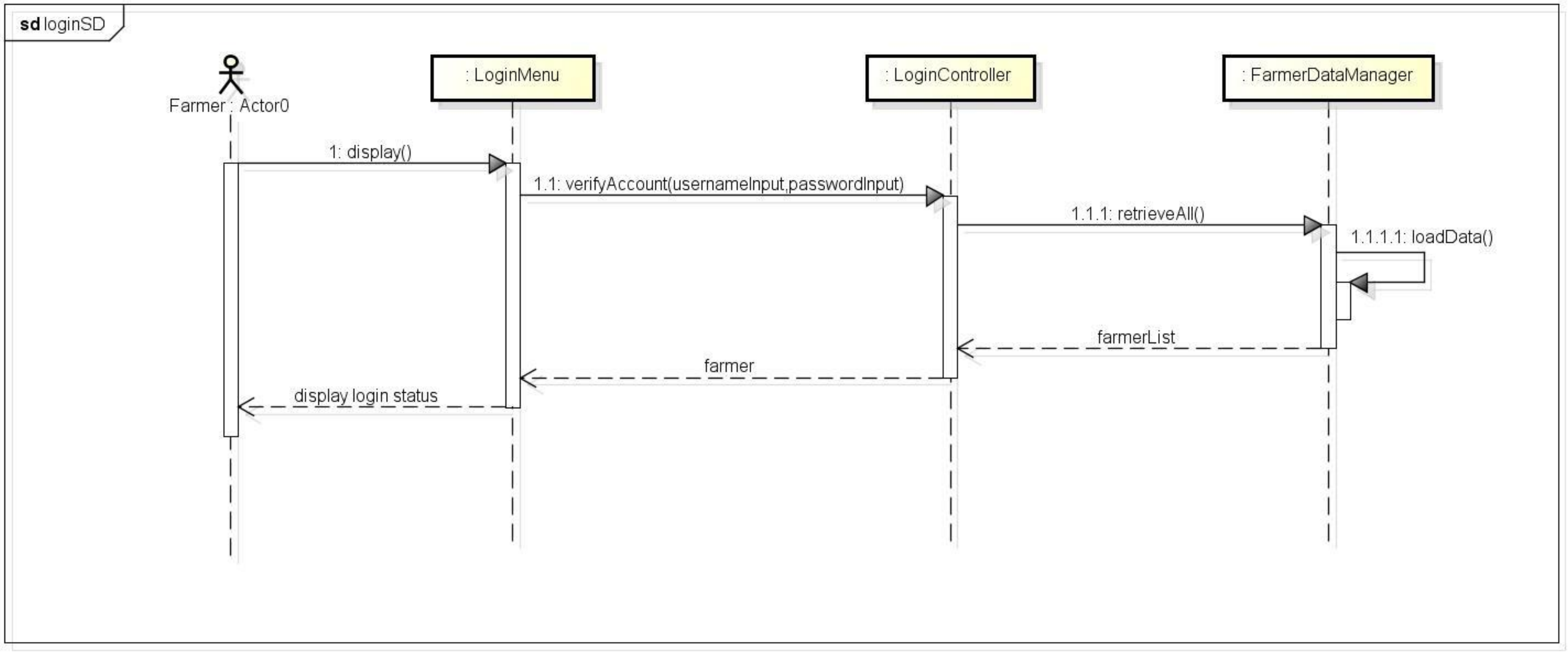
1a. The user has already logged in and selects the option to logout

i. The user exits the app
ii. The use case will end

SYSTEM SEQUENCE DIAGRAM – LOG IN/ OUT



SEQUENCE DIAGRAM – LOG IN/OUT



USE CASE SCENARIOS – SEND GIFT

ACTOR

Farmer

PRE-CONDITIONS

- The user is logged in
- The user has at least one confirmed friend

MAIN FLOW OF EVENTS

1. The use case begins when farmer selects the gift option at the Inventory Menu
2. The system will display a list of gifts that can be sent to a friend
3. The system will prompt farmer to select the gift that will be sent
4. Farmer submits the gift choice
5. The system will prompt farmer to list the friends that will receive the gift
6. Farmer enters the username of his friend or friends
7. The system will verify the list of friends
8. The system will verify the sending limit has not been met
9. The system will update the farmer's gift list with the friend's username, the gift selected and the date sent
10. The system will update friend's inventory with the gift received.
11. The system will display a successful message to farmer.
12. The system will display the Inventory Menu.
13. The use case will end.

ALTERNATE FLOW OF EVENTS

4a. Farmer submits an invalid option for gift

i. System will display an error message
ii. The use case will end

7a. Farmer submits an invalid username

i. System will display an error message
ii. The use-case will end

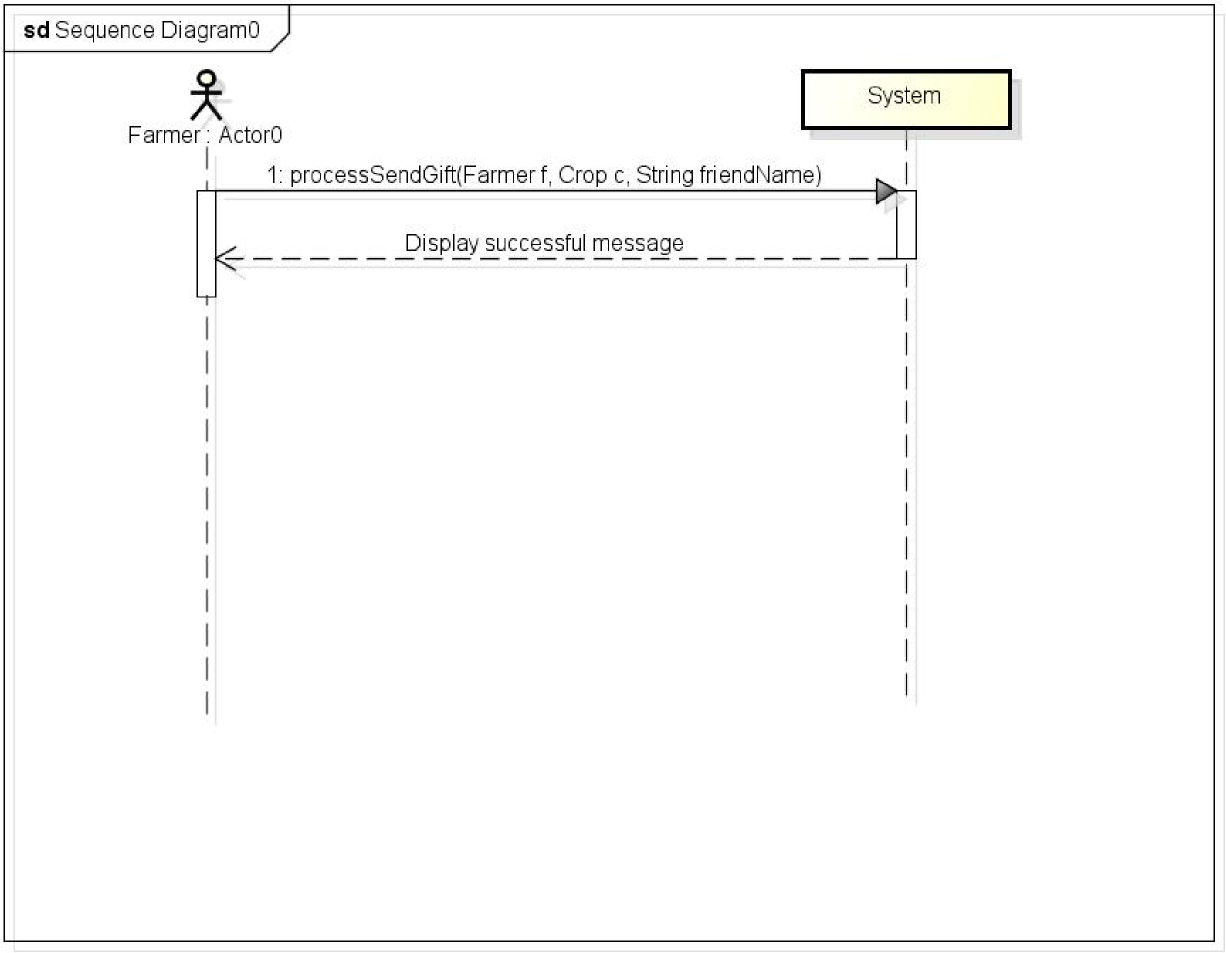
5a. Farmer has sent a gift to the specified friend on that particular day

i. System will display an error message
ii. The use-case will end

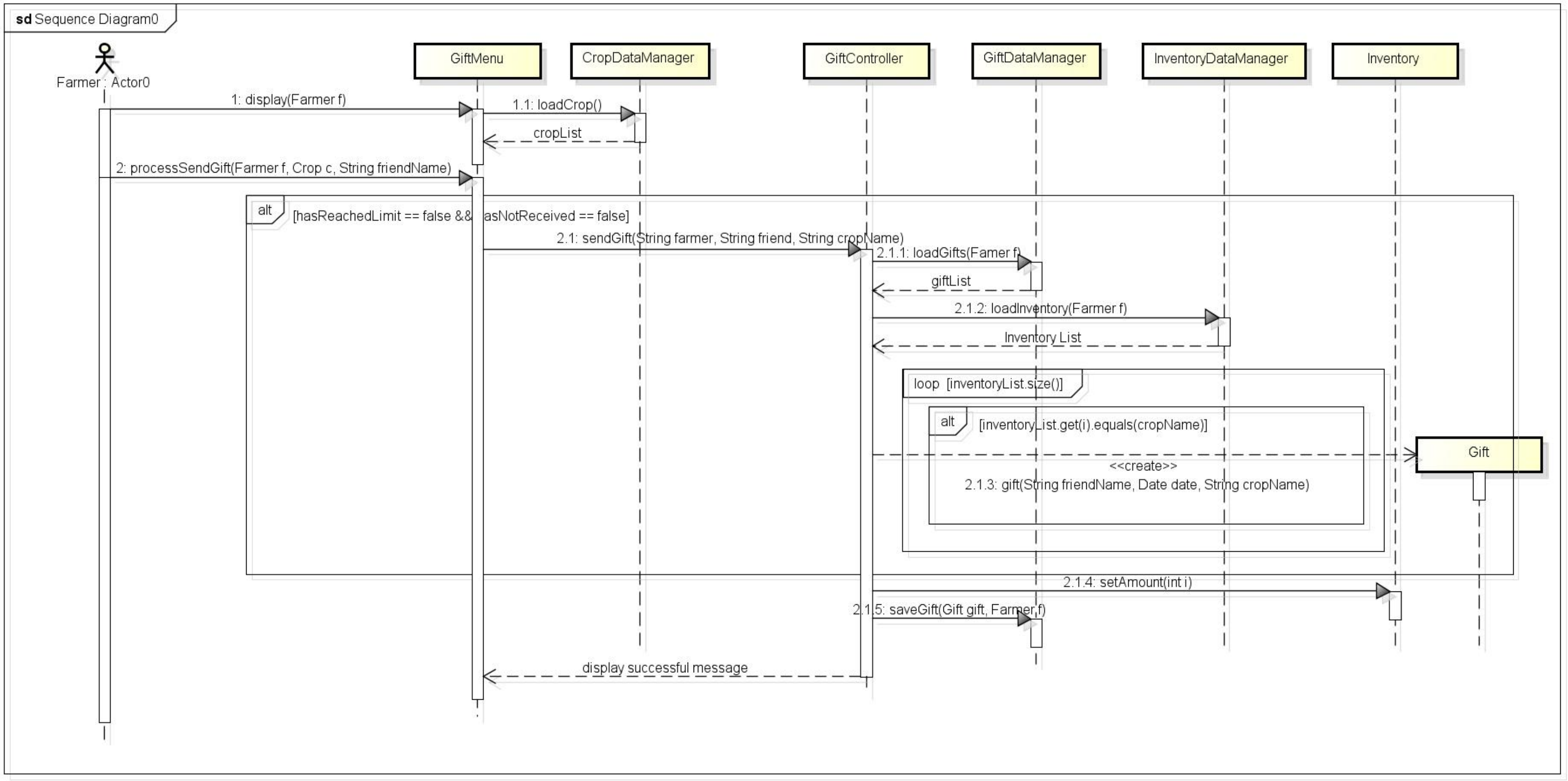
5b. Farmer has sent the maximum number of gifts for the day

i. System will display an error message
ii. The use-case will end

SYSTEM SEQUENCE DIAGRAM – SEND GIFT


sd Sequence Diagram0
Farmer : Actor0
System
1: processSendGift(Farmer f, Crop c, String friendName)
Display successful message

SEQUENCE DIAGRAM – SEND GIFT



sd Sequence Diagram0
Farmer : Actor0
GiftMenu
CropDataManager
GiftController
GiftDataManager
InventoryDataManager
Inventory
1: display(Farmer f)
1.1: loadCrop()
cropList
2: processSendGift(Farmer f, Crop c, String friendName)
alt
[hasReachedLimit == false && hasNotReceived == false]
2.1: sendGift(String farmer, String friend, String cropName)
2.1.1: loadGifts(Famer f)
giftList
2.1.2: loadInventory(Farmer f)
Inventory List
loop [inventoryList.size()]
alt
[inventoryList.get(i).equals(cropName)]
Gift
<<create>>
2.1.3: gift(String friendName, Date date, String cropName)
2.1.4: setAmount(int i)
2.1.5: saveGift(Gift gift, Farmer f)
display successful message

USE CASE SCENARIOS – PURCHASE SEED

ACTOR

Farmer

PRE-CONDITIONS

- The farmer is logged in
- The farmer has gold

MAIN FLOW OF EVENTS

1. The use case begins when the farmer selects the buy option at the Inventory Menu
2. The system will display all crops that are available for sale
3. The system will prompt user to select seed
4. The farmer selects the seed to purchase
5. The system will verify the farmer's input
6. The system will prompt the quantity the farmer would like to purchase
7. The farmer inputs the quantity to purchase
8. The system will verify the input and calculate the total cost of purchase
9. The system will verify the farmer has sufficient gold to purchase
10. The system will deduct the farmer's gold
11. The system will update farmer's inventory
12. The system will display successful message
13. The system will display the updated inventory list
14. The use case will end

ALTERNATE FLOW OF EVENTS

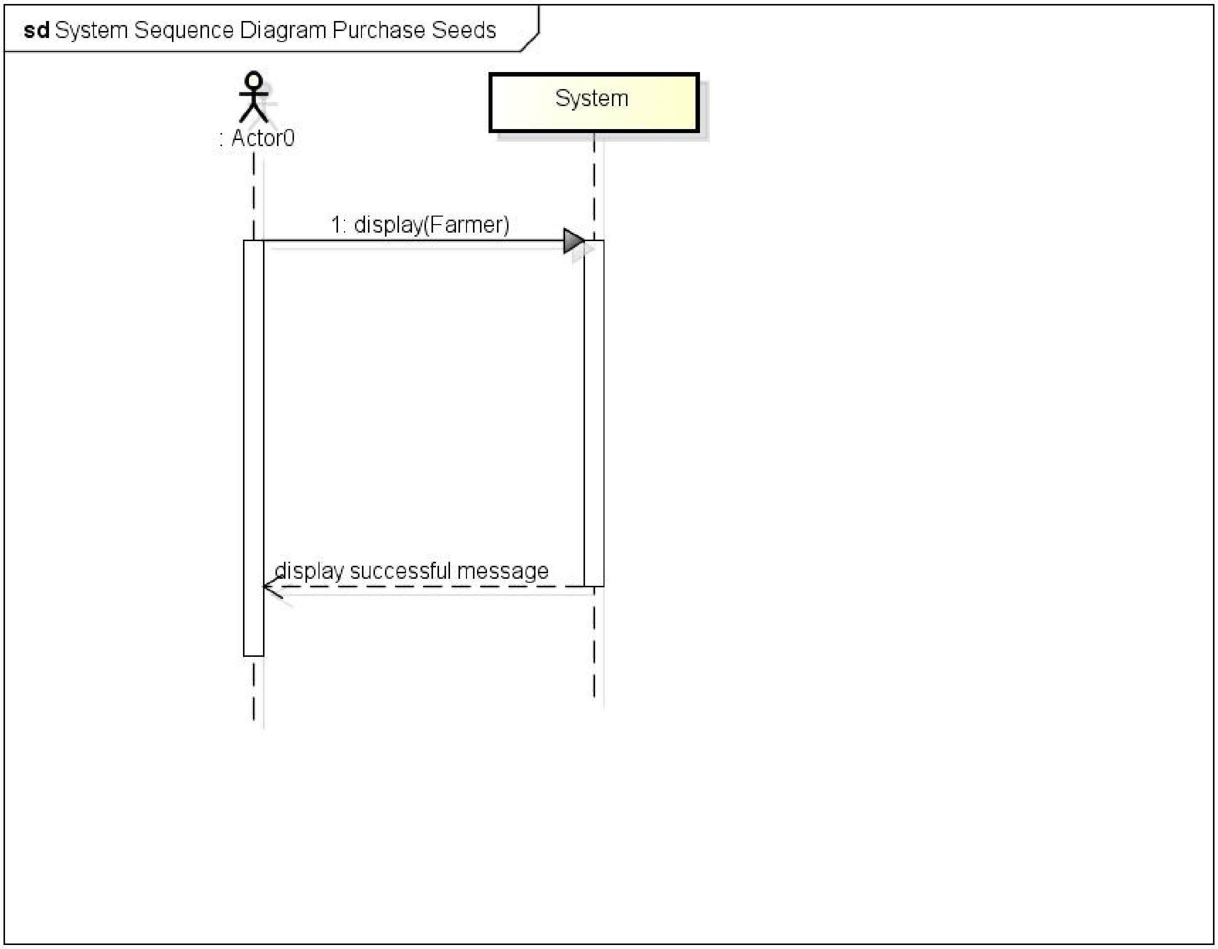
3a. The farmer submits an invalid input

i. The system will display an error message
ii. The use case will end

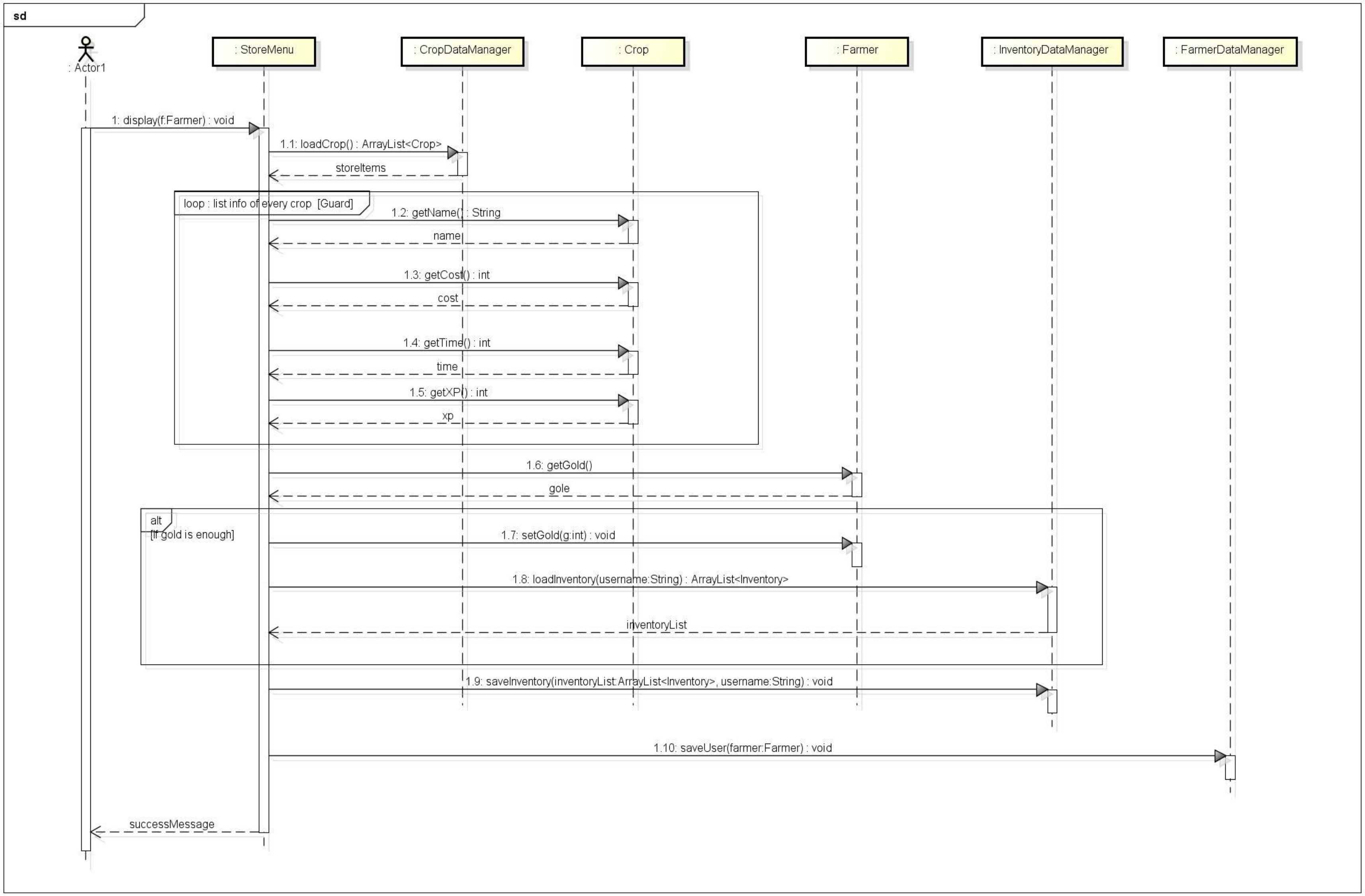
9a. The farmer has insufficient gold to purchase crop

i. The system will show error message
ii. The use case will end

SYSTEM SEQUENCE DIAGRAM – PURCHASE SEED


sd System Sequence Diagram Purchase Seeds
: Actor0
System
1: display(Farmer)
display successful message

SEQUENCE DIAGRAM – PURCHASE SEED


sd
: Actor1
: StoreMenu
: CropDataManager
: Crop
: Farmer
: InventoryDataManager
: FarmerDataManager
1: display(f:Farmer) : void
1.1: loadCrop() : ArrayList<Crop>
storeItems
loop : list info of every crop [Guard]
1.2: getName() : String
name
1.3: getCost() : int
cost
1.4: getTime() : int
time
1.5: getXP() : int
xp
1.6: getGold()
gole
alt
[if gold is enough]
1.7: setGold(g:int) : void
1.8: loadInventory(username:String) : ArrayList<Inventory>
inventoryList
1.9: saveInventory(inventoryList:ArrayList<Inventory>, username:String) : void
1.10: saveUser(farmer:Farmer) : void
successMessage

USE CASE SCENARIOS – PLANT SEEDS

ACTOR

Farmer

PRE-CONDITIONS

- The farmer is logged in
- The farmer has an empty plot of land to plant a seed
- The farmer has seeds in his inventory

MAIN FLOW OF EVENTS

15. This use case begins when the farmer decides to plant a crop.
16. The system display all plots activities.
17. The farmer selects an empty plot of land to grow his crop
18. The system displays the seeds available in the inventory
19. The farmer selects a seed to grow from his inventory
20. The system will update the plot’s status and farmer’s inventory.
21. The farmer ends the process.

ALTERNATE FLOW OF EVENTS

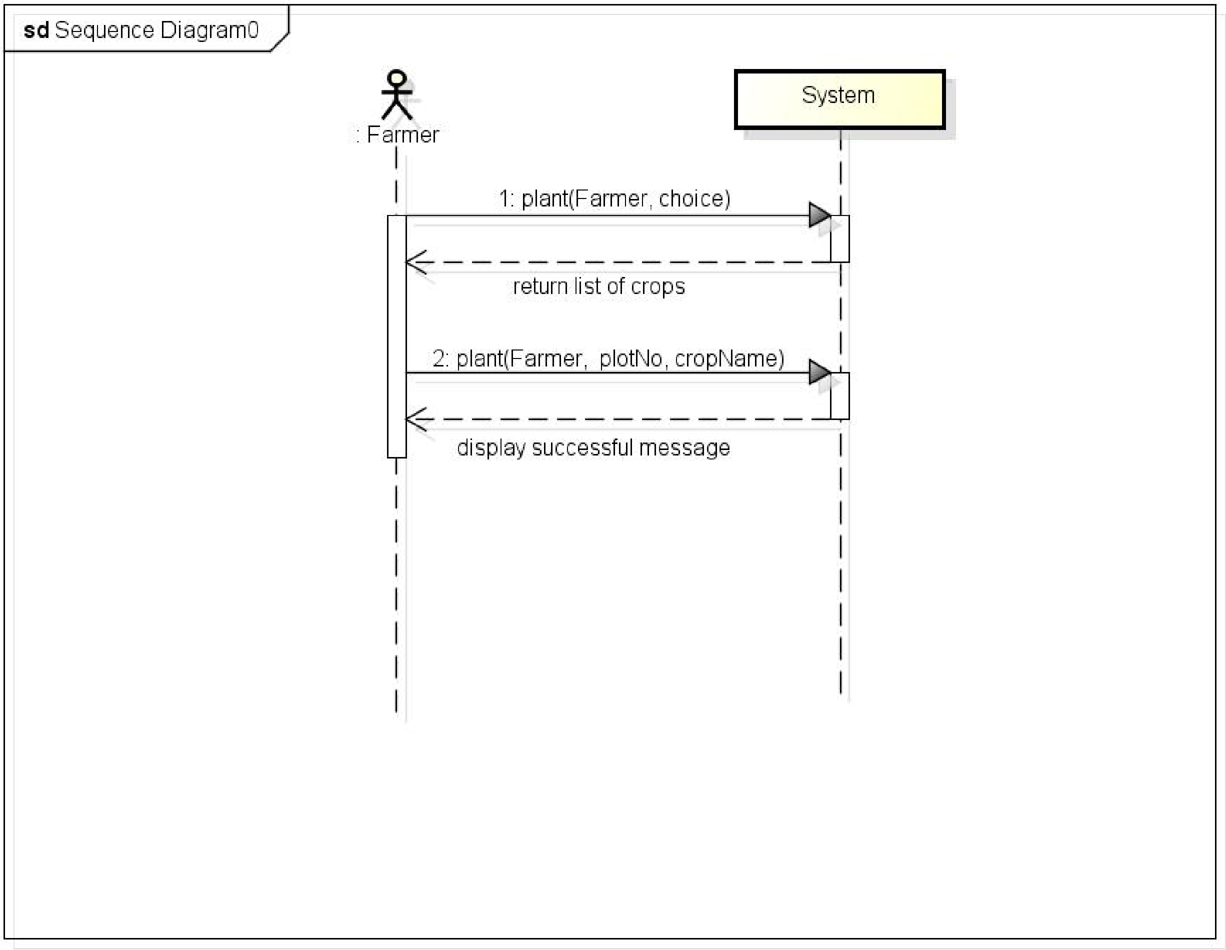
3a. Plot chosen is not empty

i. Display error message

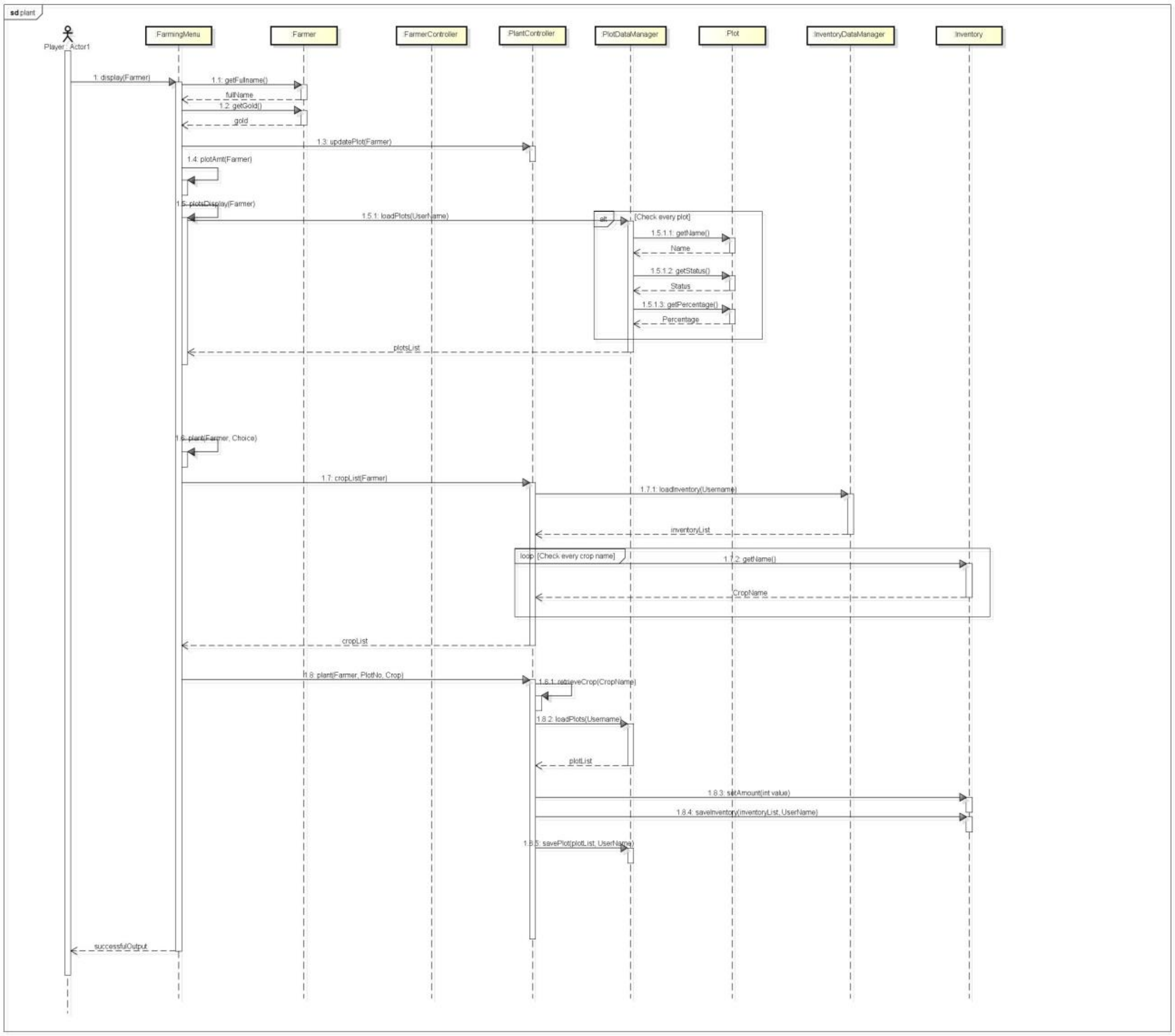
4a. The farmer’s inventory is empty

i. Display error message
ii. Use case will end

SYSTEM SEQUENCE DIAGRAM – PLANT SEEDS



SEQUENCE DIAGRAM – PLANT SEEDS



USE CASE SCENARIOS – HARVEST CROP

ACTOR

Farmer

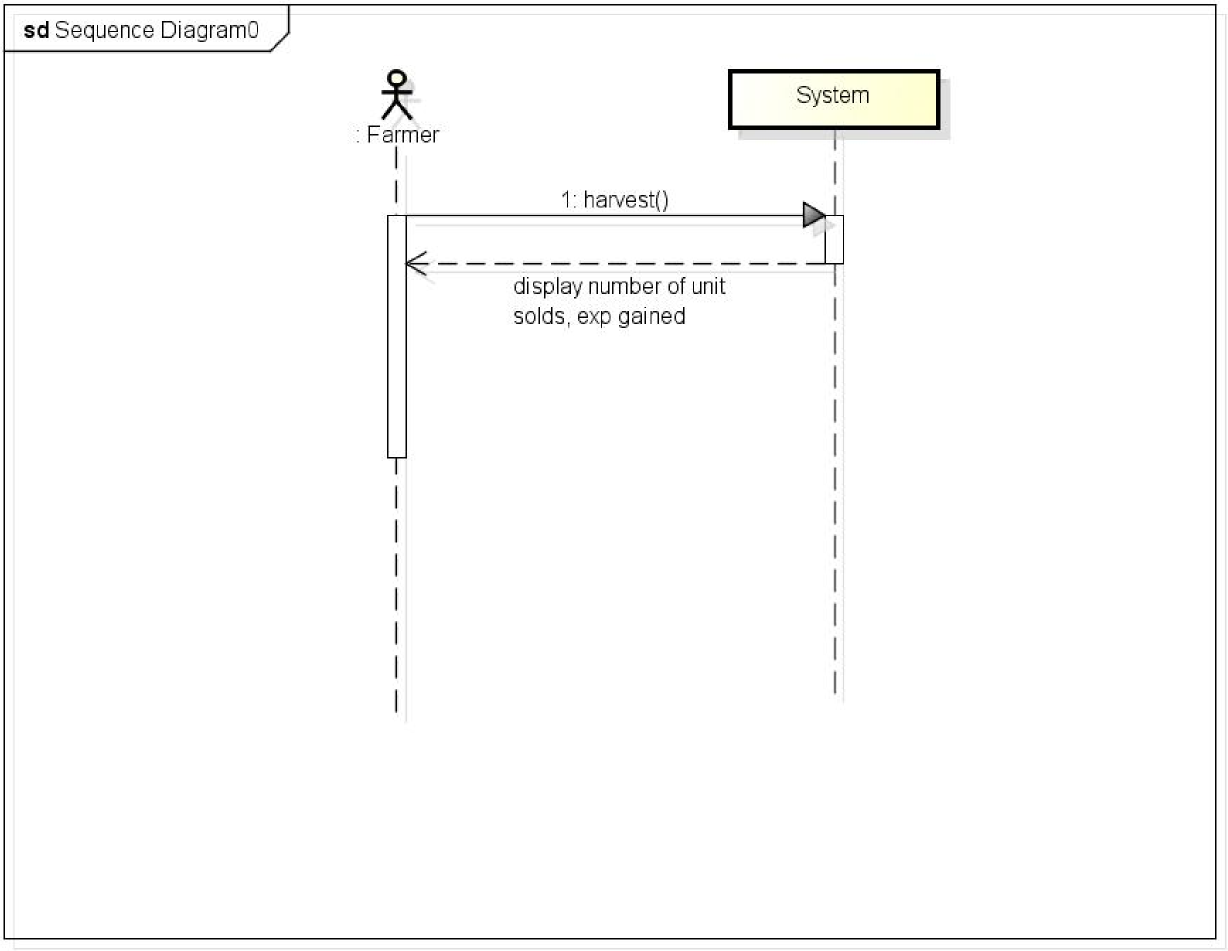
PRE-CONDITIONS

- The farmer is logged in
- The farmer has a fully-grown crop

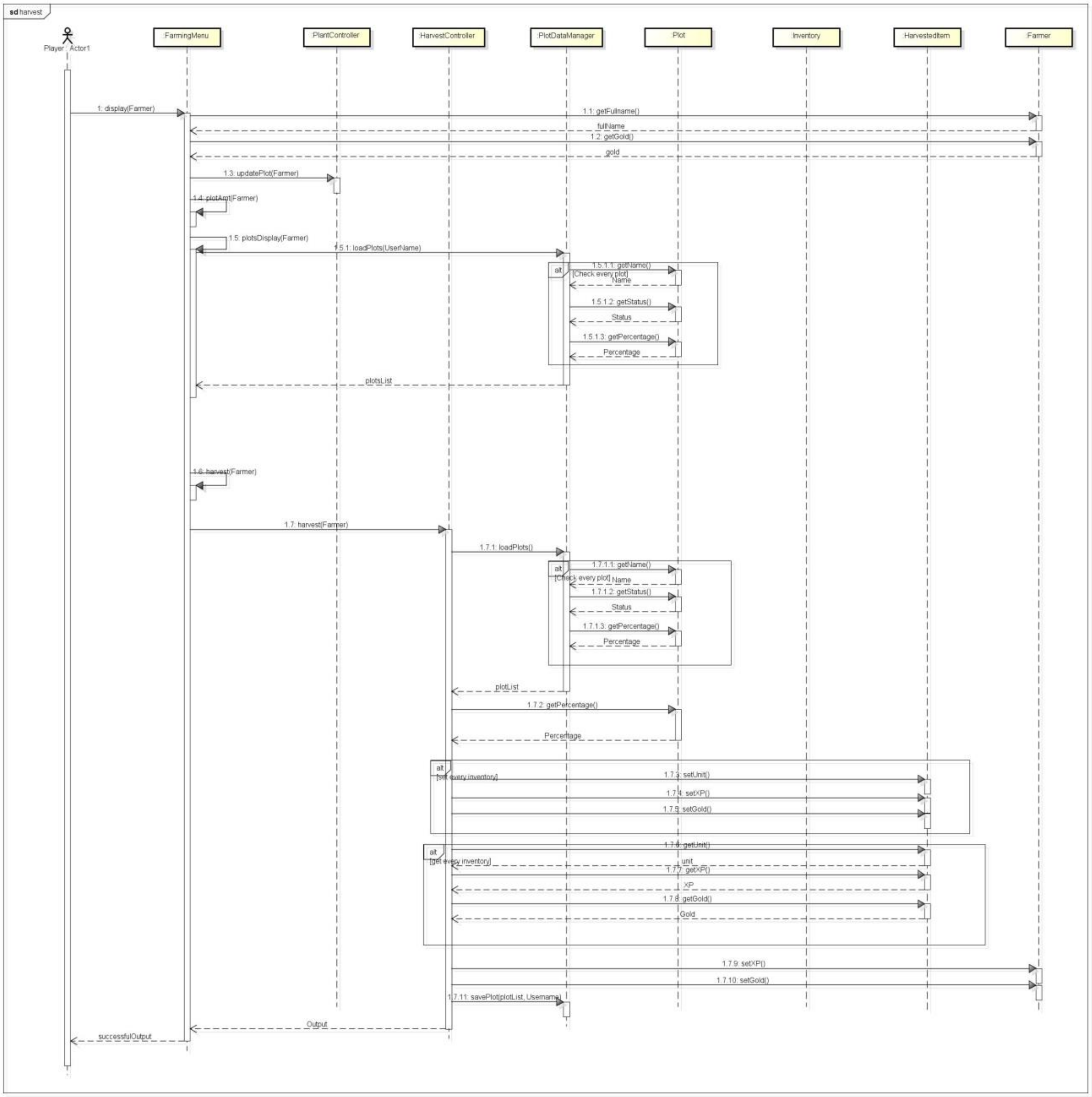
MAIN FLOW OF EVENTS

1. This use case begins when the farmer has a fully-grown crop
2. The farmer harvests all plants that are fully grown and not wilted
3. The system will automatically sell the harvested plants and display the golds gained from the transaction
4. The system will update the farmer’s gold and XP
5. The system will remove the crops that were harvested from the plot and update the status to empty.

SYSTEM SEQUENCE DIAGRAM – HARVEST CROP



SEQUENCE DIAGRAM – HARVEST CROP



USE CASE SCENARIOS – CLEAR CROP

ACTOR

Farmer

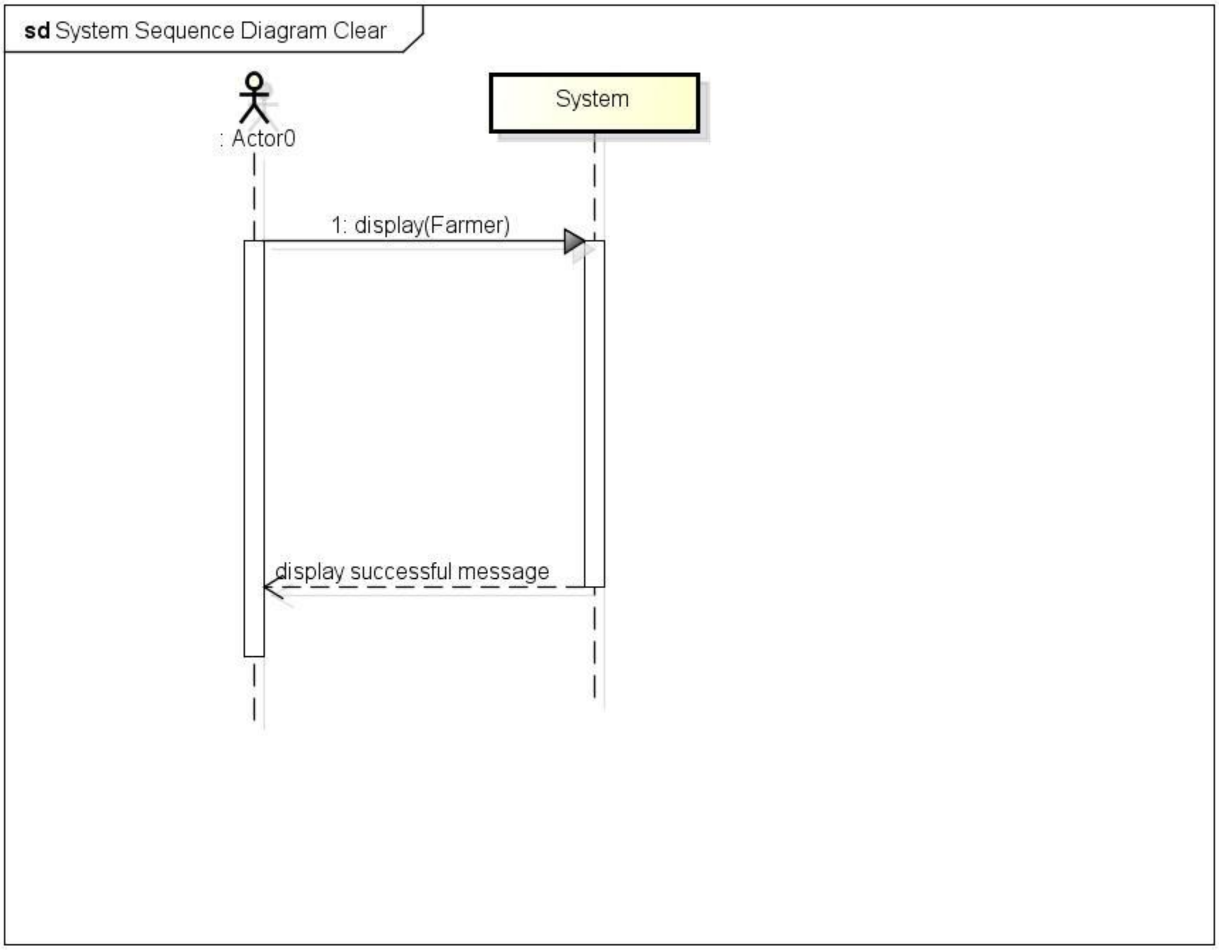
PRE-CONDITIONS

- The farmer is logged in
- The farmer has wilted crops

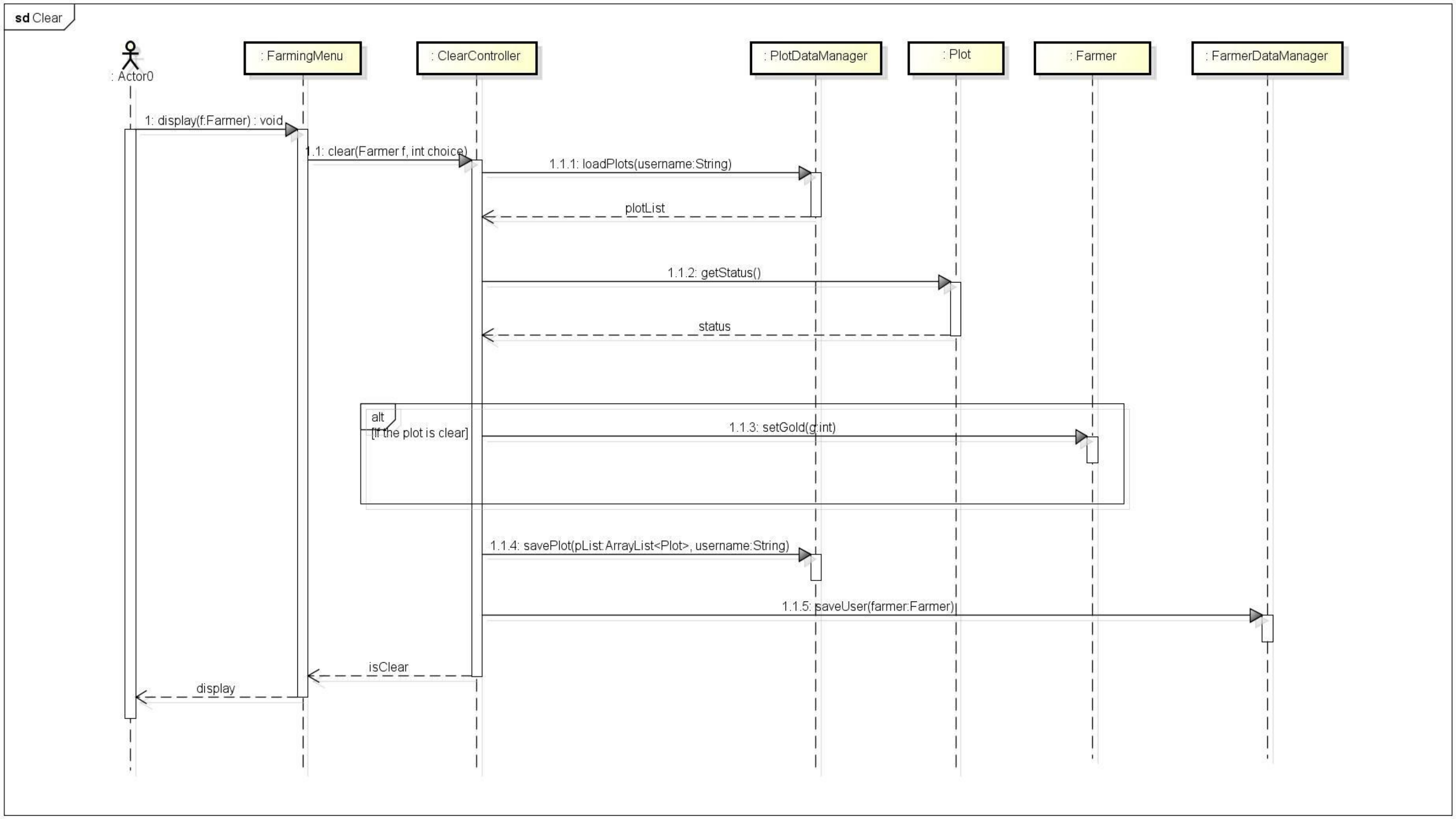
MAIN FLOW OF EVENTS

1. The use case begins when the farmer selects the clear option at the Farming Menu
2. The system will clear all wilted crops
3. The system will update the plot as empty
4. The system will verify that the farmer has sufficient gold
5. The system will deduct 5 gold coins from farmer
6. The system will display successful message
7. The use case will end

SYSTEM SEQUENCE DIAGRAM – CLEAR CROP



SEQUENCE DIAGRAM – CLEAR CROP



USE CASE SCENARIOS – MANAGE FRIEND

ACTOR

Farmer

PRE-CONDITIONS

- The farmer is logged in
- The farmer has at least one confirmed friend

MAIN FLOW OF EVENTS

1. The use case begins when farmer selects the Unfriend option in Friends Menu
2. The system will display list of friends
3. The system will prompt farmer to select a friend
4. Farmer will select a friend to unfriend
5. The system will verify the input
6. The system will remove friend from farmer's list of friends
7. The system will remove farmer from friend's list of friends
8. The system will display a successful message
9. The system will display updated list of friends
10. The use case will end

ALTERNATE FLOW OF EVENTS

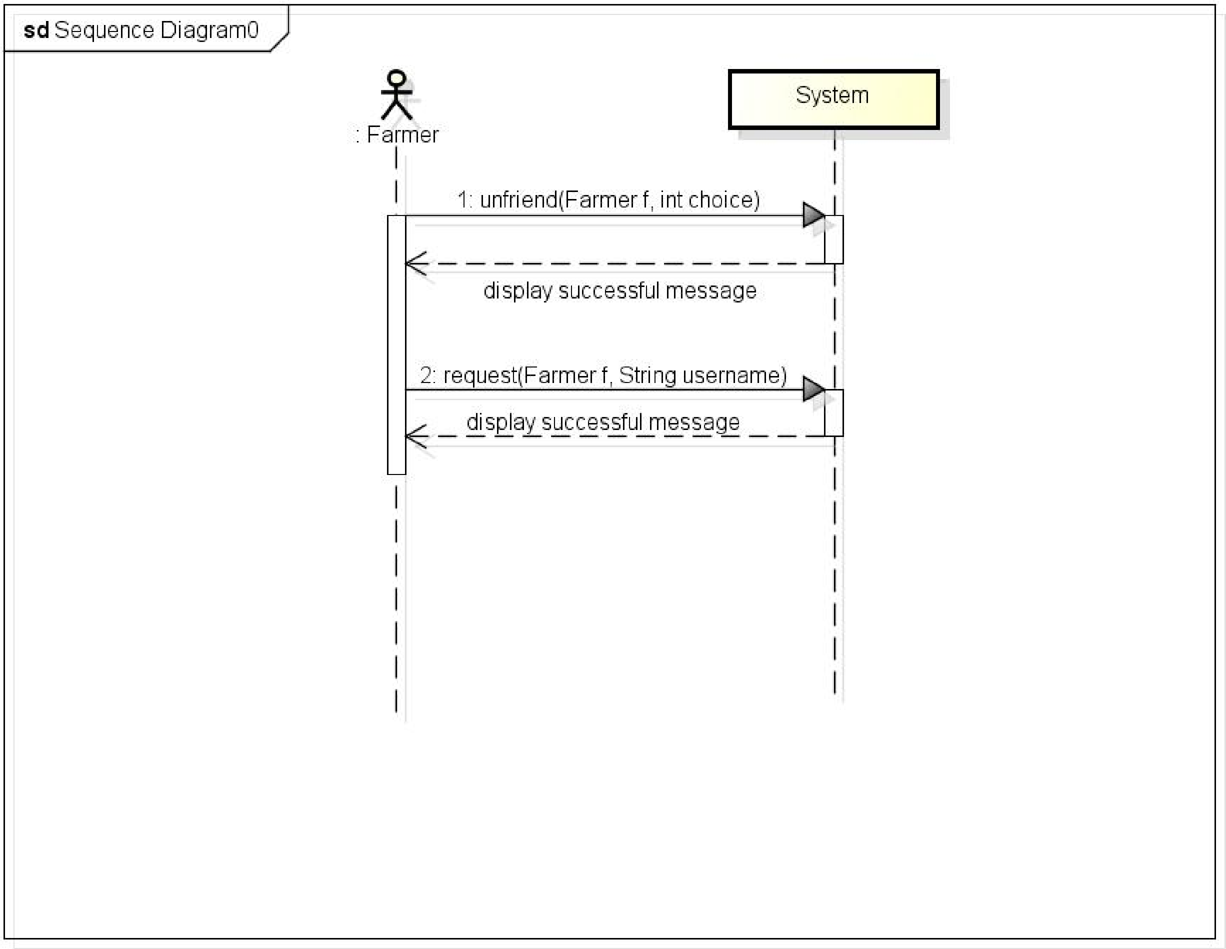
1a. The farmer selects the Request option in Friends Menu

i. The system will display list of friends
ii. The system will prompt user to submit a username
iii. Farmer will enter the username of friend that he wishes to add
iv. The system will verify the username
v. The system will update farmer's list with friend request
vi. The system will update friend's list of friends with request from farmer
vii. The system will display a successful message
viii. The use case will end

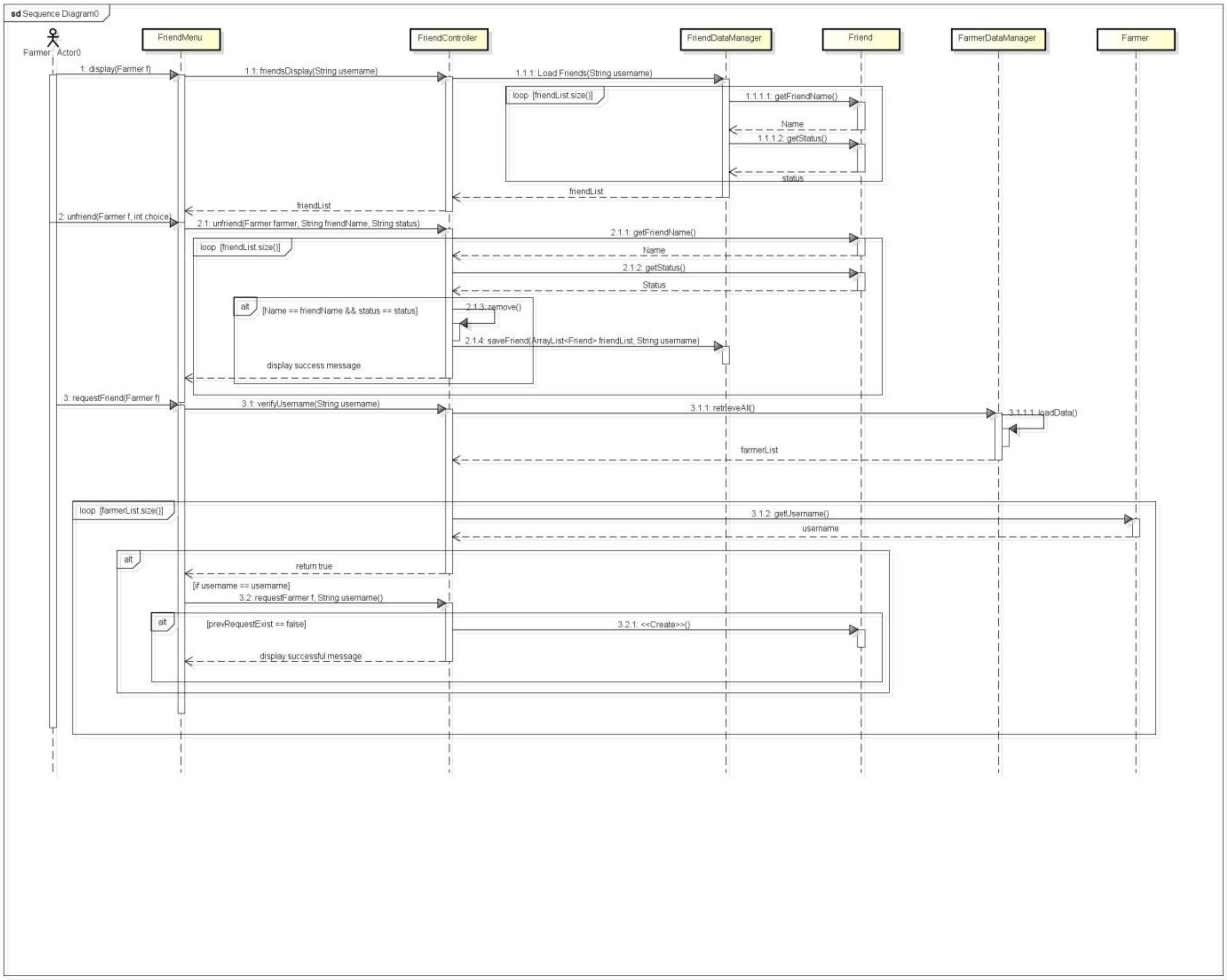
4a. The user submits an invalid input

i. The system will display an error message
ii. The use case will end

SYSTEM SEQUENCE DIAGRAM – MANAGE FRIEND


sd Sequence Diagram0
: Farmer
System
1: unfriend(Farmer f, int choice)
display successful message
2: request(Farmer f, String username)
display successful message

SEQUENCE DIAGRAM – MANAGE FRIEND



USE CASE SCENARIOS – MANAGE FRIEND REQUEST

ACTOR

Farmer

PRE-CONDITIONS

- The farmer is logged in
- The farmer has at least one friend request

MAIN FLOW OF EVENTS

1. The use case begins when farmer decides to respond to a friend request
2. The system will display list of friend requests
3. Farmer selects the friend request that he will accept
4. The system will update friend’s status in Farmer list of friends
5. The system will update farmer’s status in friend’s list of friends
6. The system will display successful message
7. The system will display updated list of friends
8. The use case will end.

ALTERNATE FLOW OF EVENTS

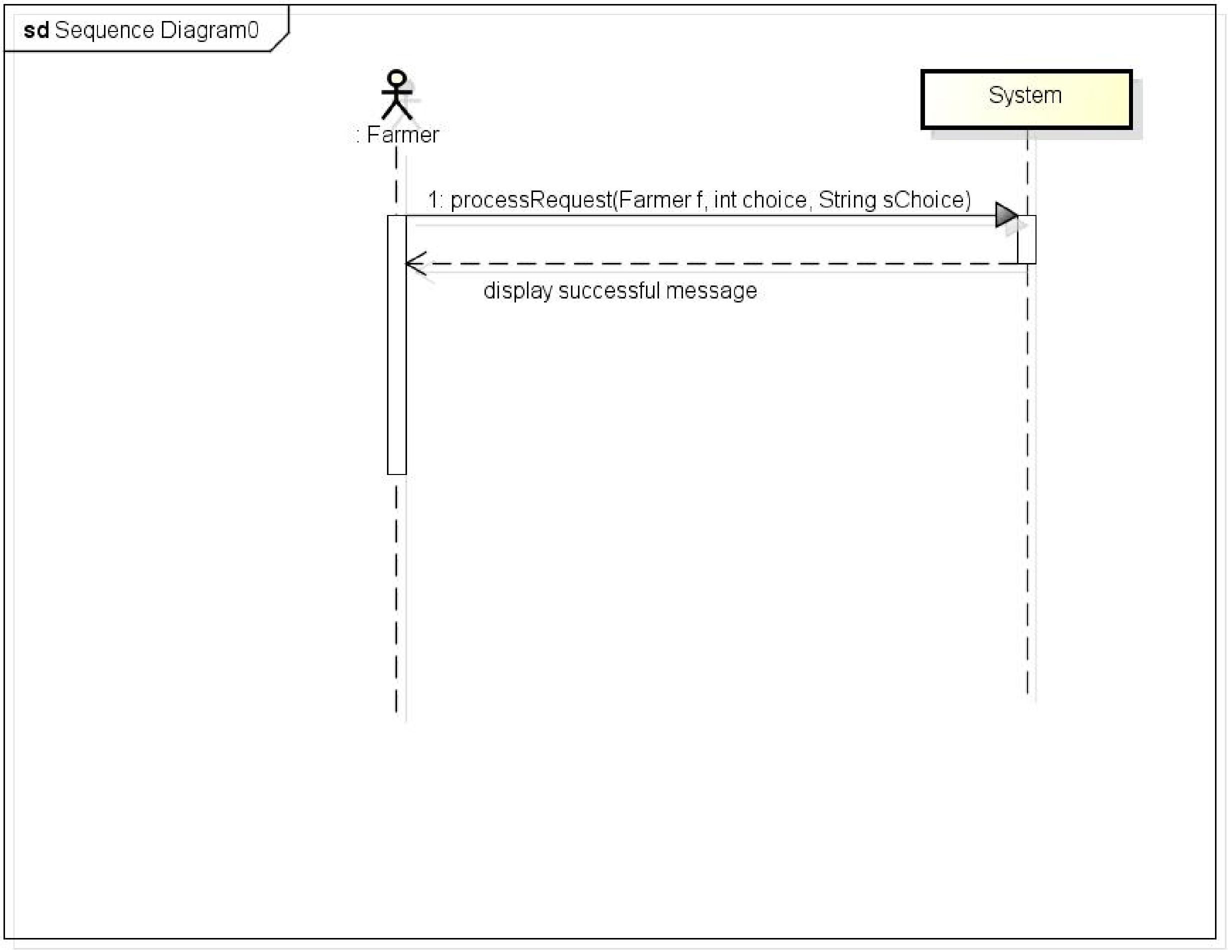
3a. Farmer selects the friend request that he will reject

i. The system will remove friend from farmer’s list
ii. The system will remove farmer from friend’s list
iii. The system will display successful message
iv. The system will display updated list of friends
v. The use case will end

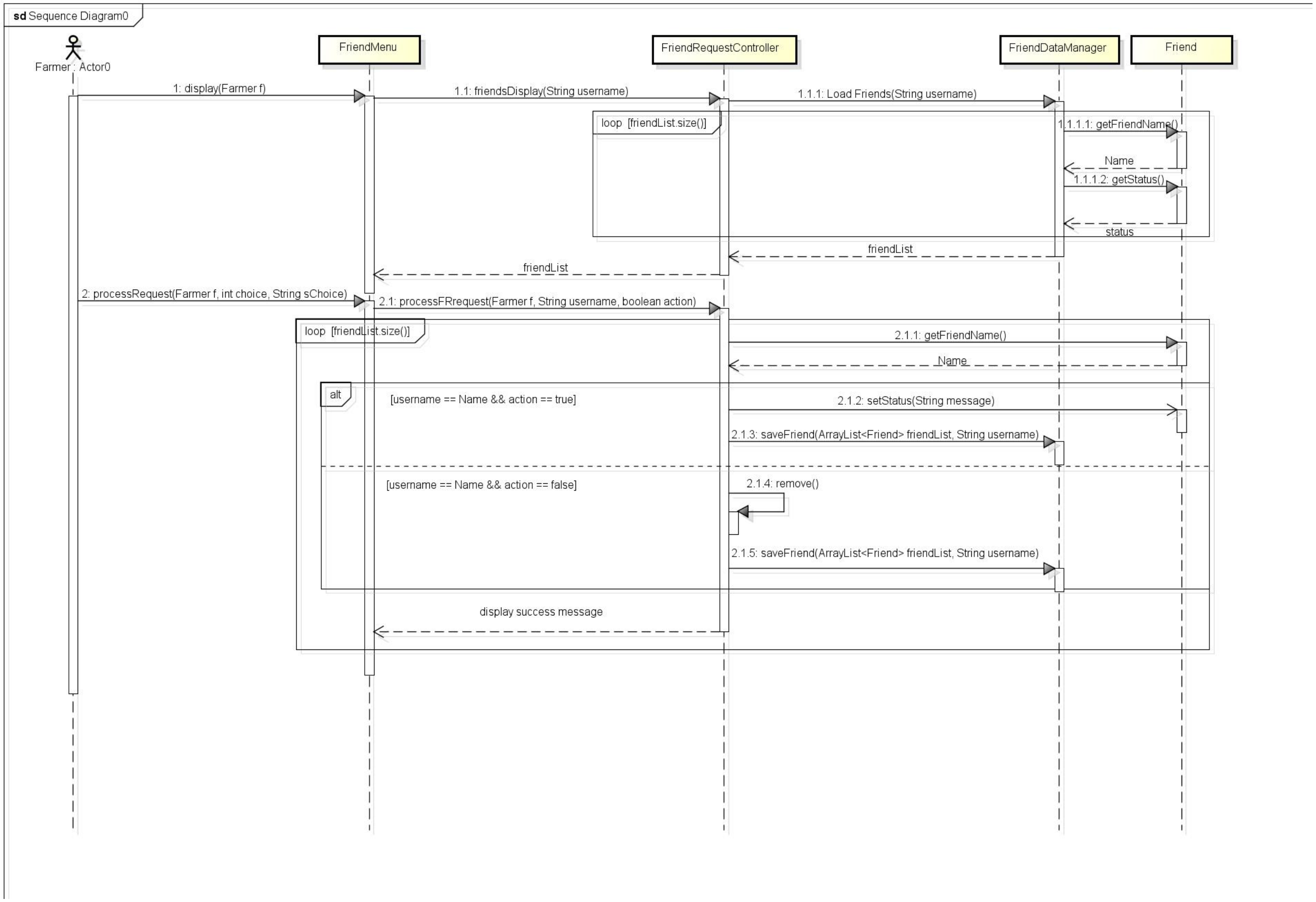
3b. Farmer enters an invalid input

i. The system will display error message
ii. The use case will end

SYSTEM SEQUENCE DIAGRAM – MANAGE FRIEND REQUEST


sd Sequence Diagram0
: Farmer
System
1: processRequest(Farmer f, int choice, String sChoice)
display successful message

SEQUENCE DIAGRAM – MANAGE FRIEND REQUEST


sd Sequence Diagram0
Farmer : Actor0
FriendMenu
FriendRequestController
FriendDataManager
Friend
1: display(Farmer f)
1.1: friendsDisplay(String username)
1.1.1: Load Friends(String username)
loop [friendList.size()]
1.1.1.1: getFriendName()
Name
1.1.1.2: getStatus()
status
friendList
friendList
2: processRequest(Farmer f, int choice, String sChoice)
2.1: processFRrequest(Farmer f, String username, boolean action)
loop [friendList.size()]
2.1.1: getFriendName()
Name
alt
[username == Name && action == true]
2.1.2: setStatus(String message)
2.1.3: saveFriend(ArrayList<Friend> friendList, String username)
[username == Name && action == false]
2.1.4: remove()
2.1.5: saveFriend(ArrayList<Friend> friendList, String username)
display success message

OBJECT ORIENTED DESIGN PRINCIPLES

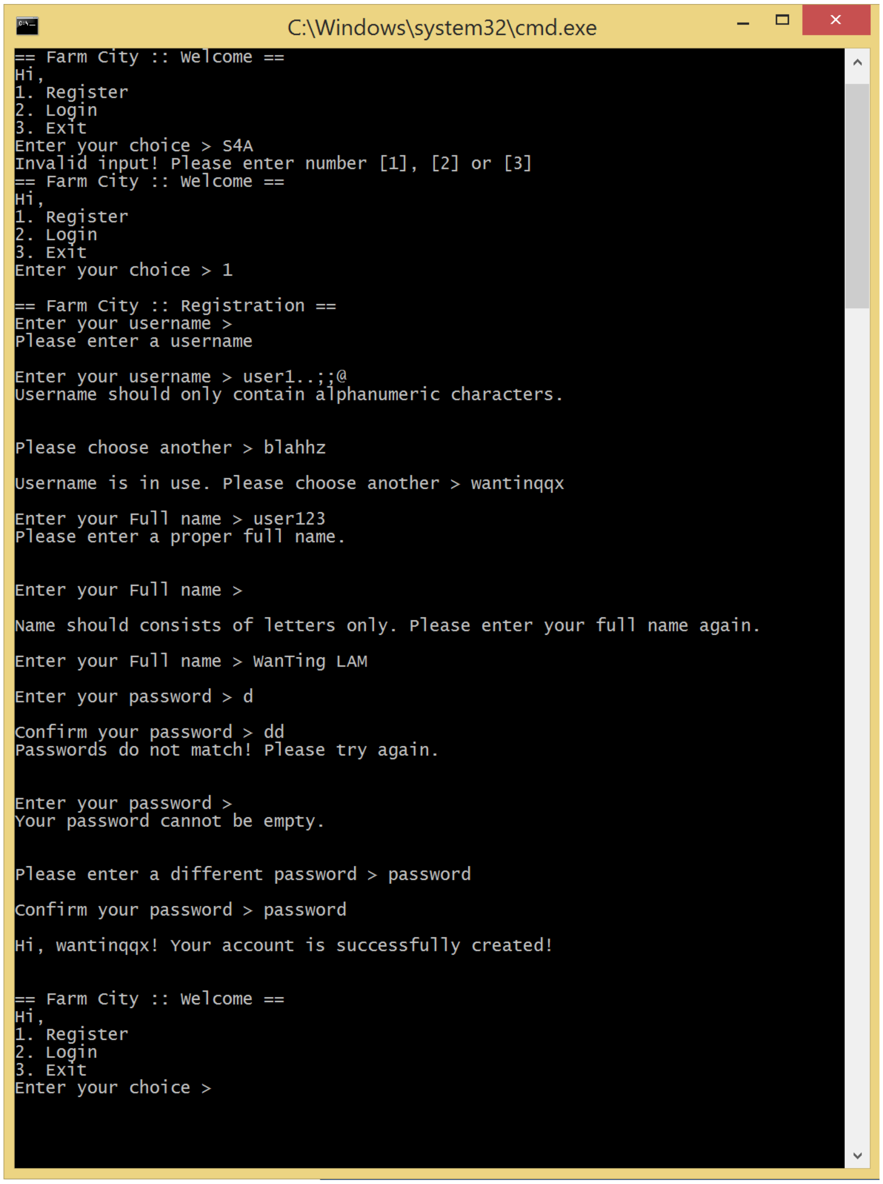
When we design our program, we organized our program based on the functionalities required. We packaged our codes into 4 individual packages namely Entities, Data Manager, Controllers and Menus. The entities classes are primarily used to create a new object and stores the setters and getters for that specific entity. The function of Data-Manager classes are mainly to update the files and load the data into an ArrayList. On the other hand, the Controller classes will interact with the Data Managers to extract the information required for the menus. It will also interact with the Menu classes to process the functionalities required in the menu. Finally, the Menu class features the user interface for the users. The start-up menu is placed outside of the 4 packages. This approach was used as it enabled us to identify our errors with little problems. For instance, when a user wants to harvest crops, we will interact with the Harvest Controller and Harvest Data Manager to update the plots and at the same time, the Farmer Controller will update the Farmer's XP and gold. This ensures that every function will only interact with one central system.

In addition, we applied the Single Responsibility-Principle. Each package created has specific functions. This ensure that when we make changes in one file, in will not affect other classes. In addition, this will make our codes easier to comprehend and designate separate responsibilities to discrete classes. To identify the responsibilities of an object we are required to have the knowledge about the private attributes how the objects interact with each other, as shown in the domain diagram. In addition, it will reduces the possibility of the program crashing when we integrate the whole program as the changes will be localized and will not affect other portions of the program. Also, it has fine granularity of highly related functionality support increased reuse potential of codes. If we implemented multiple responsibilities instead, the codes might be difficult to understand, maintain and reuse. For instance, similar codes can be used to harvest and clear crops. By using single responsibility principle, it was simpler for us to reuse the codes and minimize the possibility of the program crashing. Thus, our program is not rigid as it is not complex for a user to manipulate the program to make simple changes without causing the program to crash as the codes is easy to comprehend.

MISCELLANEOUS

	Learning Outcome	Challenges
Knowledge Application	Through the group project, we were able to test our understanding on the concepts that were taught in class. Also, it helped to improve our knowledge on object-oriented and Java language.	We found it difficult to identify which concept was best suited for the given scenario. We had a few misconceptions about the project requirement that were cleared during the interim submission.
Collaboration	The project allowed us to acquire collaborative skills. Each member contributed to the project and were always helping each other whenever someone faces an issue.	Due to different class schedules and work-load, it was difficult to set a meeting date hence, we usually split the work-load before the meeting. This is to ensure that we maximize the time when we meet and focus on the issues that we face and making sure that we are all on the right track.
Integrating requirement and technology	Through this project, we were able to analyses the scenario given and identify the functionalities that we were required to develop in our program. We believe the concepts taught in this module was sufficient to complete this project. However, as we gain more IT skills, we will be able to develop a more efficient and advance program.	

REGISTRATION



```
== Farm City :: Welcome ==
Hi,
1. Register
2. Login
3. Exit
Enter your choice > S4A
Invalid input! Please enter number [1], [2] or [3]
== Farm City :: Welcome ==
Hi,
1. Register
2. Login
3. Exit
Enter your choice > 1

== Farm City :: Registration ==
Enter your username >
Please enter a username

Enter your username > user1..;;@
Username should only contain alphanumeric characters.


Please choose another > blahhz

Username is in use. Please choose another > wantinqqx

Enter your Full name > user123
Please enter a proper full name.


Enter your Full name >

Name should consists of letters only. Please enter your full name again.

Enter your Full name > WanTing LAM

Enter your password > d

Confirm your password > dd
Passwords do not match! Please try again.


Enter your password >
Your password cannot be empty.


Please enter a different password > password

Confirm your password > password

Hi, wantinqqx! Your account is successfully created!


== Farm City :: Welcome ==
Hi,
1. Register
2. Login
3. Exit
Enter your choice >
```

[VALIDATION]

Welcome Menu

- Only 1,2 or 3 accepted

Username

- Not Empty
- Alphanumeric only
- Username availability

Full Name

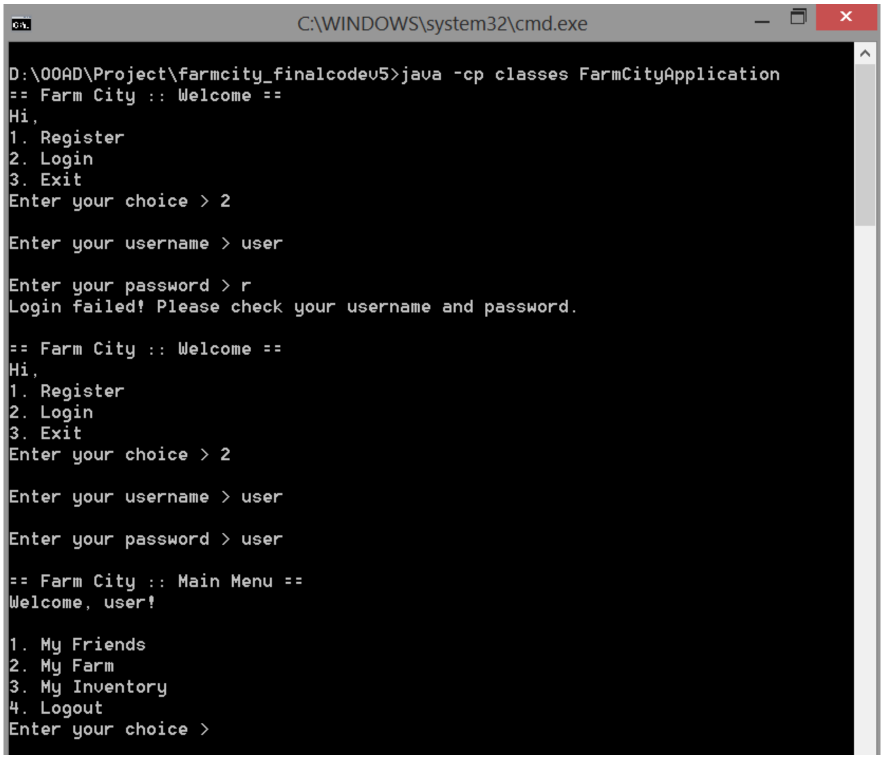
- Not Empty
- Alphabets only

Password

- Not Empty
- Must match with Confirm Password

If all criteria are fulfilled, the user account will be created and it displays a success confirmation.

LOGIN



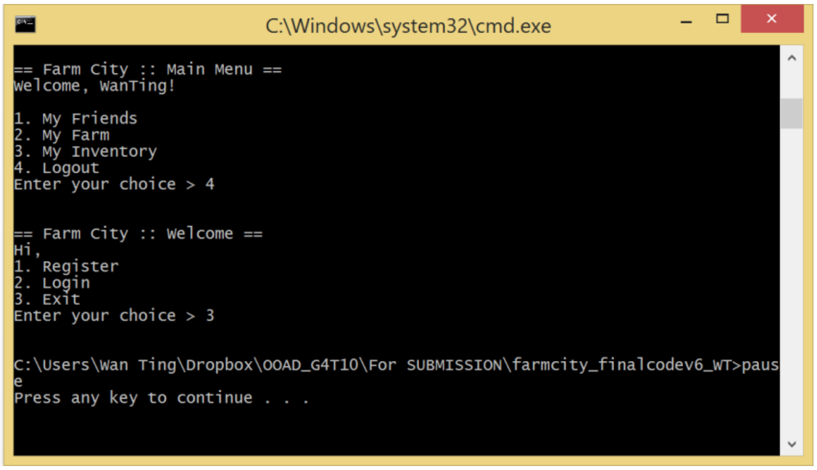
```
D:\OOAD\Project\farmcity_finalcodev5>java -cp classes FarmCityApplication
== Farm City :: Welcome ==
Hi,
1. Register
2. Login
3. Exit
Enter your choice > 2

Enter your username > user

Enter your password > r
Login failed! Please check your username and password.

== Farm City :: Welcome ==
Hi,
1. Register
2. Login
3. Exit
Enter your choice > 2

Enter your username > user

Enter your password > user

== Farm City :: Main Menu ==
Welcome, user!

1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice >
```

[VALIDATION]

Login Menu

- Verifies if the account and password exists in our database

LOGOUT



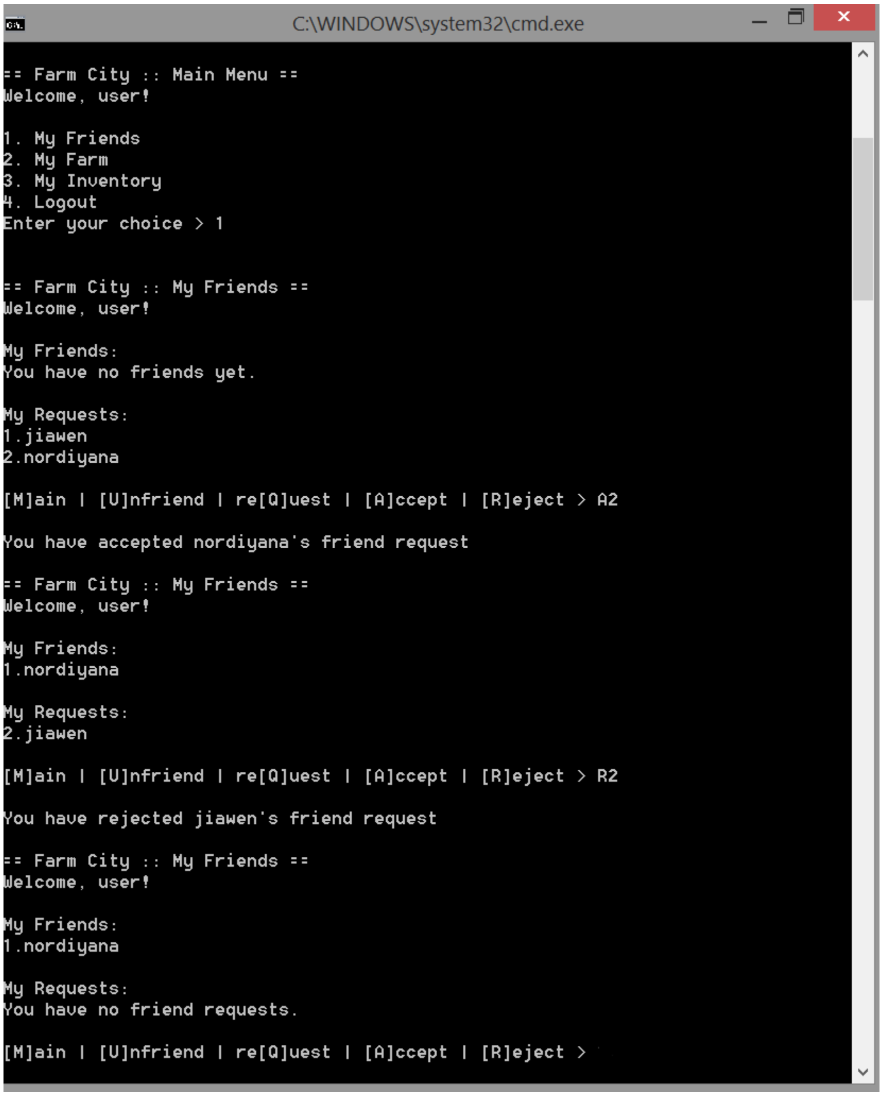
```
== Farm City :: Main Menu ==
Welcome, WanTing!

1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice > 4


== Farm City :: Welcome ==
Hi,
1. Register
2. Login
3. Exit
Enter your choice > 3


C:\Users\Wan Ting\Dropbox\OOAD_G4T10\For SUBMISSION\farmcity_finalcodev6_WT>paus
e
Press any key to continue . . .
```

[VALIDATION]

Main Menu

- Logs the user out and brings them back to the Welcome Menu

MY FRIENDS [ACCEPT/REJECT]



```
== Farm City :: Main Menu ==
Welcome, user!

1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice > 1


== Farm City :: My Friends ==
Welcome, user!

My Friends:
You have no friends yet.

My Requests:
1.jiawen
2.nordiyana

[M]ain | [U]nfriend | re[Q]uest | [A]ccept | [R]eject > A2

You have accepted nordiyana's friend request

== Farm City :: My Friends ==
Welcome, user!

My Friends:
1.nordiyana

My Requests:
2.jiawen

[M]ain | [U]nfriend | re[Q]uest | [A]ccept | [R]eject > R2

You have rejected jiawen's friend request

== Farm City :: My Friends ==
Welcome, user!

My Friends:
1.nordiyana

My Requests:
You have no friend requests.

[M]ain | [U]nfriend | re[Q]uest | [A]ccept | [R]eject >
```

[VALIDATION]

Main Menu

- User selects [1] to “My Friends”

Friends Menu

- Displays current friend lists & requests

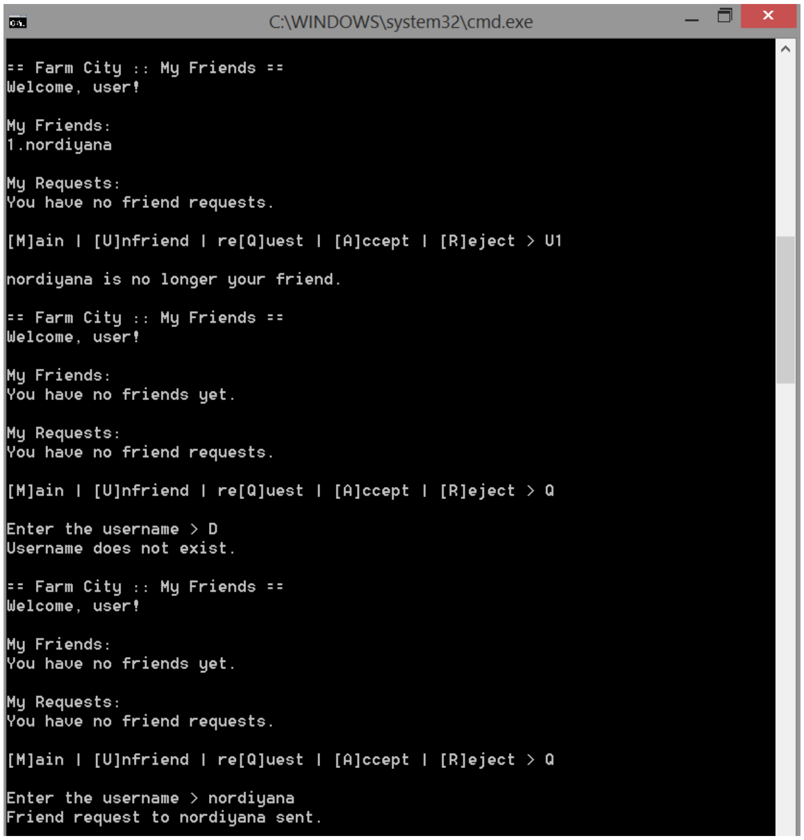
[ACCEPT REQUESTS]

- User selects A<ID> to accept.
- Changes the friend status from request to “My Friends”

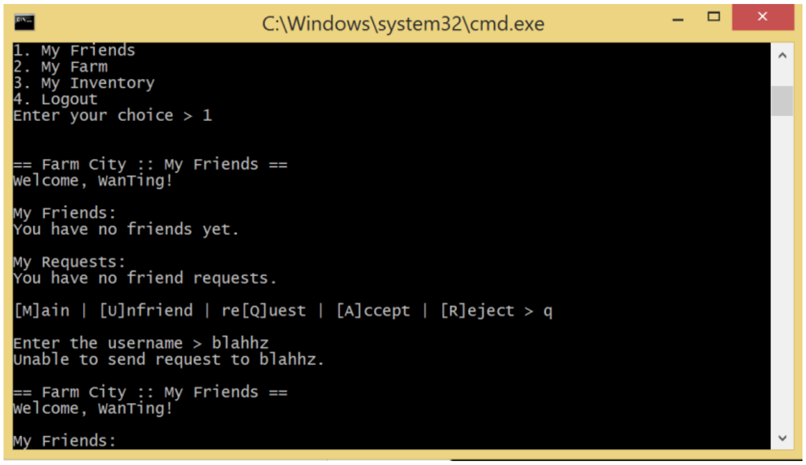
[REJECT REQUESTS]

- User selects R<ID> to reject.
- Removes the player from the user’s request list

MY FRIENDS [UNFRIEND/SENDING REQUESTS]



```
== Farm City :: My Friends ==
Welcome, user!

My Friends:
1.nordiyana

My Requests:
You have no friend requests.

[M]ain | [U]nfriend | re[Q]uest | [A]ccept | [R]eject > U1

nordiyana is no longer your friend.

== Farm City :: My Friends ==
Welcome, user!

My Friends:
You have no friends yet.

My Requests:
You have no friend requests.

[M]ain | [U]nfriend | re[Q]uest | [A]ccept | [R]eject > Q

Enter the username > D
Username does not exist.

== Farm City :: My Friends ==
Welcome, user!

My Friends:
You have no friends yet.

My Requests:
You have no friend requests.

[M]ain | [U]nfriend | re[Q]uest | [A]ccept | [R]eject > Q

Enter the username > nordiyana
Friend request to nordiyana sent.
```



```
1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice > 1


== Farm City :: My Friends ==
Welcome, WanTing!

My Friends:
You have no friends yet.

My Requests:
You have no friend requests.

[M]ain | [U]nfriend | re[Q]uest | [A]ccept | [R]eject > q

Enter the username > blahhz
Unable to send request to blahhz.

== Farm City :: My Friends ==
Welcome, WanTing!

My Friends:
```

[VALIDATION]

Friends Menu

- Displays current friend lists & requests

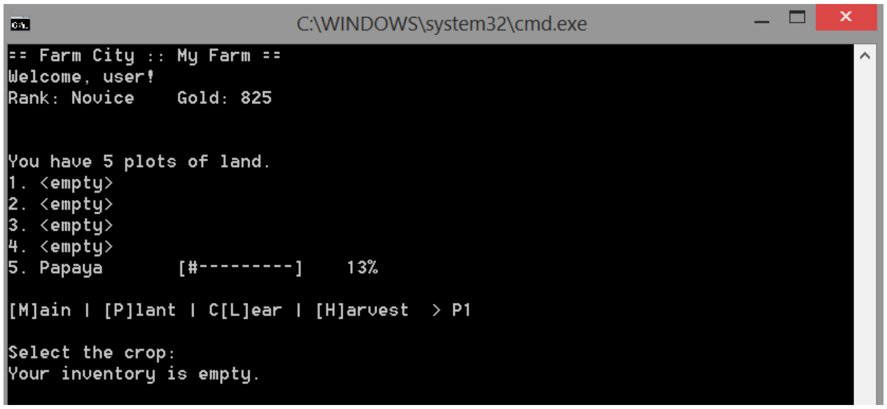
[UNFRIEND]

- User selects U<ID> to unfriend.
- Removes the player from the user's friend list

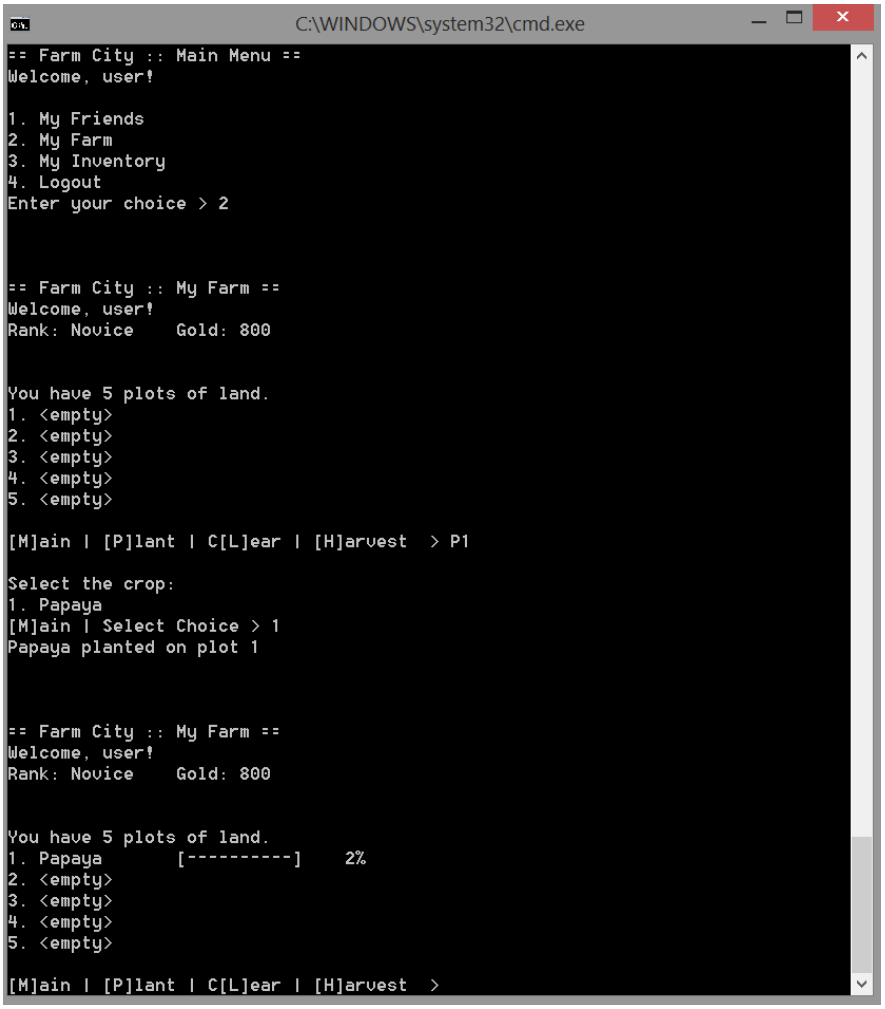
[SENDING REQUESTS]

- User selects Q to send requests.
- User enters username of the user he/she wishes to make friends with.
- System checks if username exists in the system, and if the username is not their own username.
- Displays friend request sent.

MY FARM (**PLANT**, HARVEST, CLEAR)



```
== Farm City :: My Farm ==
Welcome, user!
Rank: Novice    Gold: 825


You have 5 plots of land.
1. <empty>
2. <empty>
3. <empty>
4. <empty>
5. Papaya       [#---------]    13%

[M]ain | [P]lant | C[L]ear | [H]arvest  > P1

Select the crop:
Your inventory is empty.
```



```
== Farm City :: Main Menu ==
Welcome, user!

1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice > 2



== Farm City :: My Farm ==
Welcome, user!
Rank: Novice    Gold: 800


You have 5 plots of land.
1. <empty>
2. <empty>
3. <empty>
4. <empty>
5. <empty>

[M]ain | [P]lant | C[L]ear | [H]arvest  > P1

Select the crop:
1. Papaya
[M]ain | Select Choice > 1
Papaya planted on plot 1



== Farm City :: My Farm ==
Welcome, user!
Rank: Novice    Gold: 800


You have 5 plots of land.
1. Papaya       [----------]    2%
2. <empty>
3. <empty>
4. <empty>
5. <empty>

[M]ain | [P]lant | C[L]ear | [H]arvest  >
```

[VALIDATION]

Farming Menu

- Displays user's rank, gold and plot statuses

[PLANTING SEEDS]

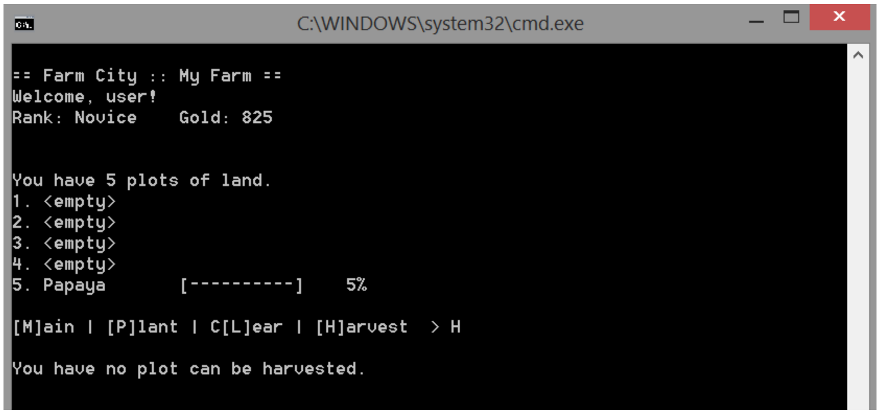
- User selects P<ID>/ p<ID> to plant seeds on the plot id selected.

- Displays inventory for users to choose the seed they wish to plant.

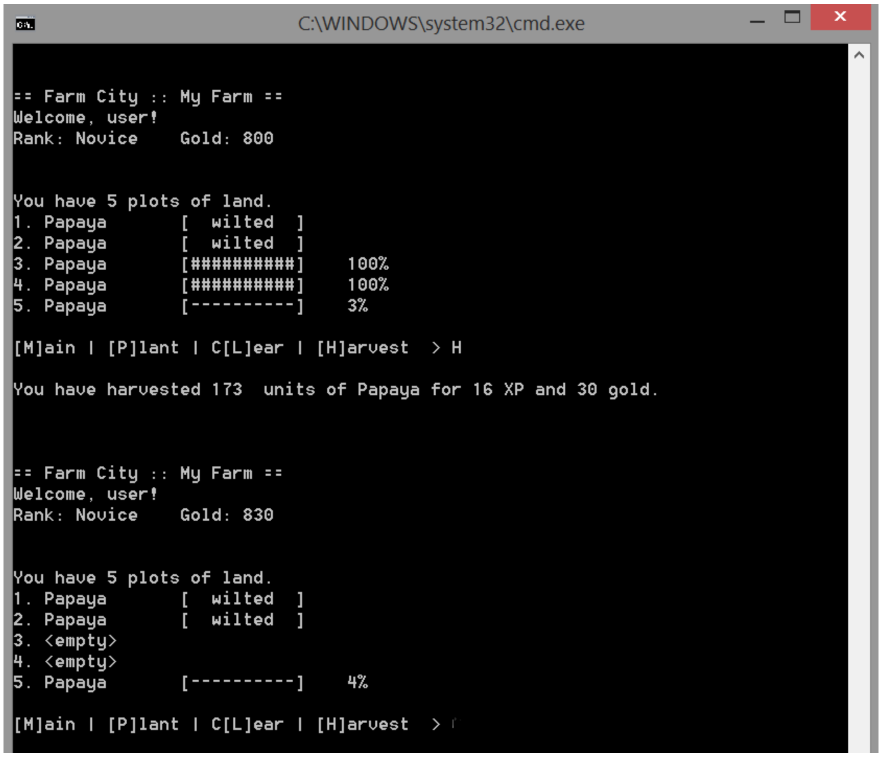
- Users will not be allowed to plant if inventory is empty

- If inventory has seeds, user selects the seed ID to plant on the chosen plot ID.
- Plot ID will display the crop status of the seed.

MY FARM (PLANT, **HARVEST**, CLEAR)



```
== Farm City :: My Farm ==
Welcome, user!
Rank: Novice    Gold: 825


You have 5 plots of land.
1. <empty>
2. <empty>
3. <empty>
4. <empty>
5. Papaya       [----------]    5%

[M]ain | [P]lant | C[L]ear | [H]arvest  > H

You have no plot can be harvested.
```



```
== Farm City :: My Farm ==
Welcome, user!
Rank: Novice    Gold: 800


You have 5 plots of land.
1. Papaya       [  wilted  ]
2. Papaya       [  wilted  ]
3. Papaya       [##########]    100%
4. Papaya       [##########]    100%
5. Papaya       [----------]    3%

[M]ain | [P]lant | C[L]ear | [H]arvest  > H

You have harvested 173  units of Papaya for 16 XP and 30 gold.



== Farm City :: My Farm ==
Welcome, user!
Rank: Novice    Gold: 830


You have 5 plots of land.
1. Papaya       [  wilted  ]
2. Papaya       [  wilted  ]
3. <empty>
4. <empty>
5. Papaya       [----------]    4%

[M]ain | [P]lant | C[L]ear | [H]arvest  >
```

[VALIDATION]

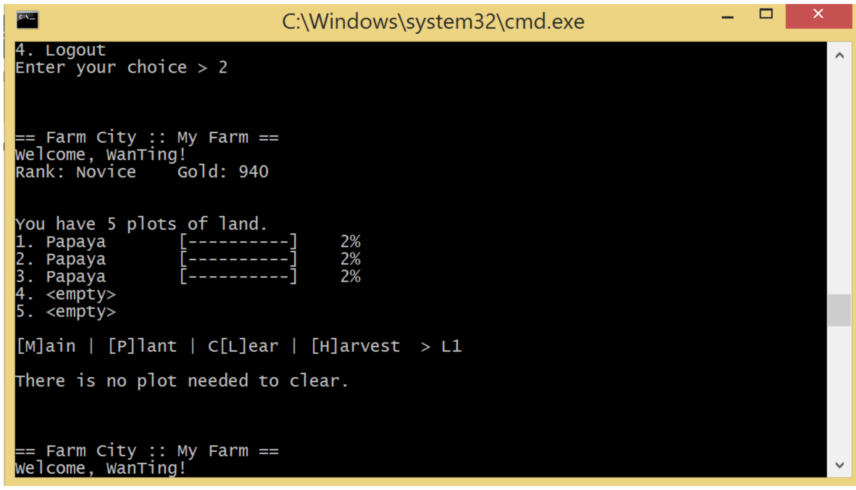
[HARVESTING CROPS]

- User selects H to harvest seeds.

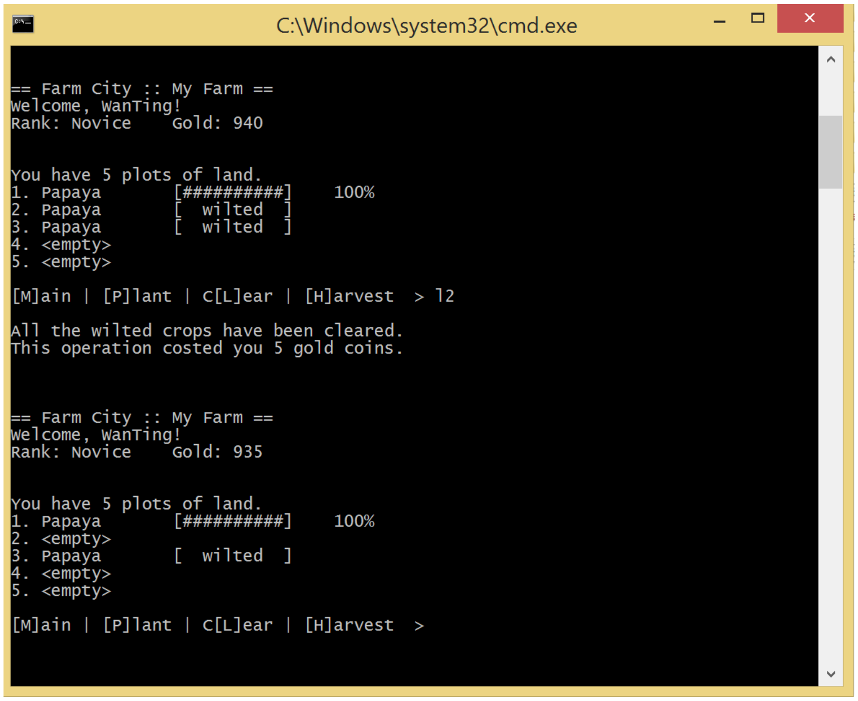
- If there are no seeds that are at 100%, the system will display an error message.

- If plot list has fully grown seeds at 100%, these seeds will be harvested and displays harvest information.
- The gold and XP of the user will be updated.
- Plot list will reset the harvested plot IDs back to <empty>.

MY FARM (PLANT, HARVEST, **CLEAR**)



```
4. Logout
Enter your choice > 2


== Farm City :: My Farm ==
Welcome, WanTing!
Rank: Novice    Gold: 940


You have 5 plots of land.
1. Papaya       [----------]    2%
2. Papaya       [----------]    2%
3. Papaya       [----------]    2%
4. <empty>
5. <empty>

[M]ain | [P]lant | C[L]ear | [H]arvest  > L1

There is no plot needed to clear.


== Farm City :: My Farm ==
Welcome, WanTing!
```



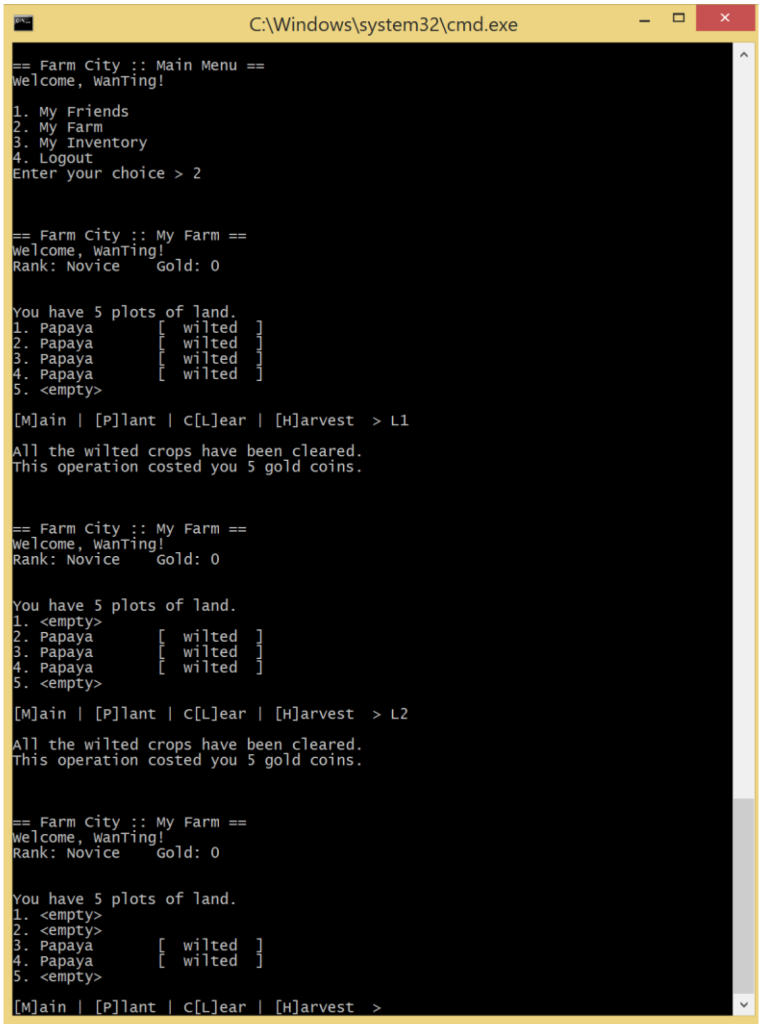
```

== Farm City :: My Farm ==
Welcome, WanTing!
Rank: Novice    Gold: 940


You have 5 plots of land.
1. Papaya       [##########]    100%
2. Papaya       [  wilted  ]
3. Papaya       [  wilted  ]
4. <empty>
5. <empty>

[M]ain | [P]lant | C[L]ear | [H]arvest  > l2

All the wilted crops have been cleared.
This operation costed you 5 gold coins.


== Farm City :: My Farm ==
Welcome, WanTing!
Rank: Novice    Gold: 935


You have 5 plots of land.
1. Papaya       [##########]    100%
2. <empty>
3. Papaya       [  wilted  ]
4. <empty>
5. <empty>

[M]ain | [P]lant | C[L]ear | [H]arvest  >
```

[VALIDATION]

[CLEARING CROPS]

- User selects L<ID> to clear wilted crops.
- If there are no seeds that are wilted, the system will display an error message.
- If plot list has wilted crops, these plots will cleared and be reset back to <empty>.
- The gold of the user will be deducted.

MY FARM (PLANT, HARVEST, **CLEAR**)



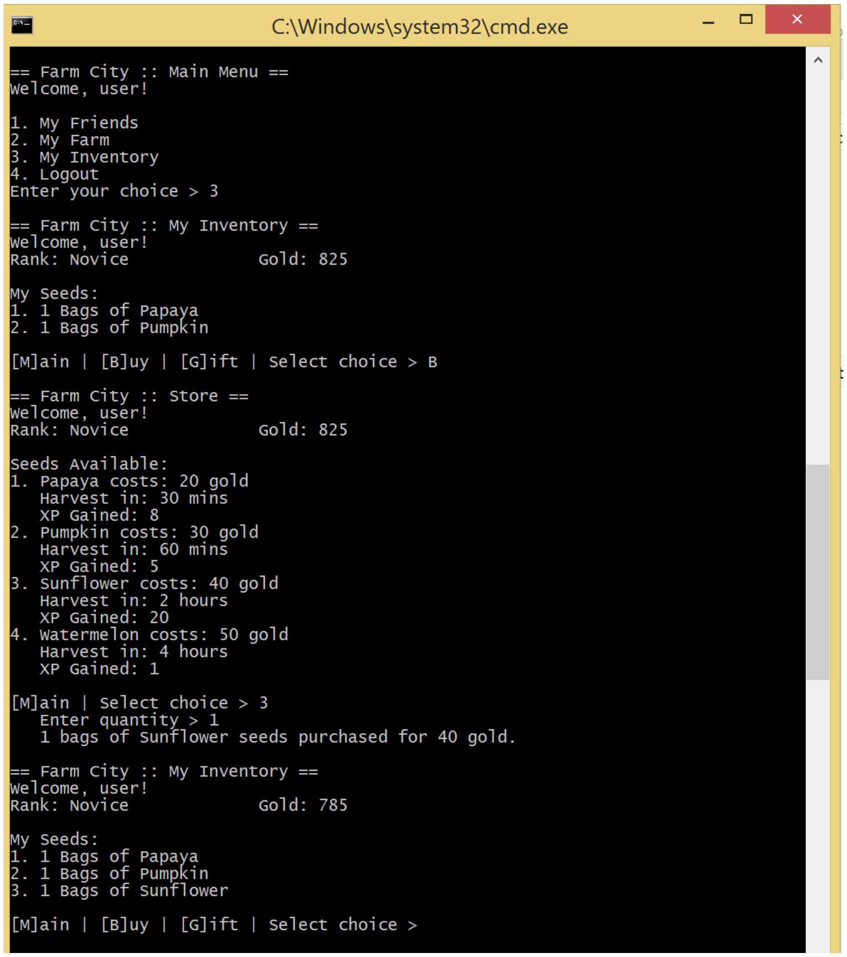
```
== Farm City :: Main Menu ==
Welcome, WanTing!

1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice > 2


== Farm City :: My Farm ==
Welcome, WanTing!
Rank: Novice    Gold: 0


You have 5 plots of land.
1. Papaya        [  wilted  ]
2. Papaya        [  wilted  ]
3. Papaya        [  wilted  ]
4. Papaya        [  wilted  ]
5. <empty>

[M]ain | [P]lant | C[L]ear | [H]arvest  > L1

All the wilted crops have been cleared.
This operation costed you 5 gold coins.


== Farm City :: My Farm ==
Welcome, WanTing!
Rank: Novice    Gold: 0


You have 5 plots of land.
1. <empty>
2. Papaya        [  wilted  ]
3. Papaya        [  wilted  ]
4. Papaya        [  wilted  ]
5. <empty>

[M]ain | [P]lant | C[L]ear | [H]arvest  > L2

All the wilted crops have been cleared.
This operation costed you 5 gold coins.


== Farm City :: My Farm ==
Welcome, WanTing!
Rank: Novice    Gold: 0


You have 5 plots of land.
1. <empty>
2. <empty>
3. Papaya        [  wilted  ]
4. Papaya        [  wilted  ]
5. <empty>

[M]ain | [P]lant | C[L]ear | [H]arvest  >
```

[VALIDATION]

[CLEARING CROPS]

- If the user has no gold and only wilted crops left, all plots can be cleared for free.

MY INVENTORY (**BUYING SEEDS**/GIFTING)



```
== Farm City :: Main Menu ==
Welcome, user!

1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice > 3

== Farm City :: My Inventory ==
Welcome, user!
Rank: Novice                Gold: 825

My Seeds:
1. 1 Bags of Papaya
2. 1 Bags of Pumpkin

[M]ain | [B]uy | [G]ift | Select choice > B

== Farm City :: Store ==
Welcome, user!
Rank: Novice                Gold: 825

Seeds Available:
1. Papaya costs: 20 gold
   Harvest in: 30 mins
   XP Gained: 8
2. Pumpkin costs: 30 gold
   Harvest in: 60 mins
   XP Gained: 5
3. Sunflower costs: 40 gold
   Harvest in: 2 hours
   XP Gained: 20
4. Watermelon costs: 50 gold
   Harvest in: 4 hours
   XP Gained: 1

[M]ain | Select choice > 3
   Enter quantity > 1
   1 bags of Sunflower seeds purchased for 40 gold.

== Farm City :: My Inventory ==
Welcome, user!
Rank: Novice                Gold: 785

My Seeds:
1. 1 Bags of Papaya
2. 1 Bags of Pumpkin
3. 1 Bags of Sunflower

[M]ain | [B]uy | [G]ift | Select choice >
```

[VALIDATION]

Inventory Menu

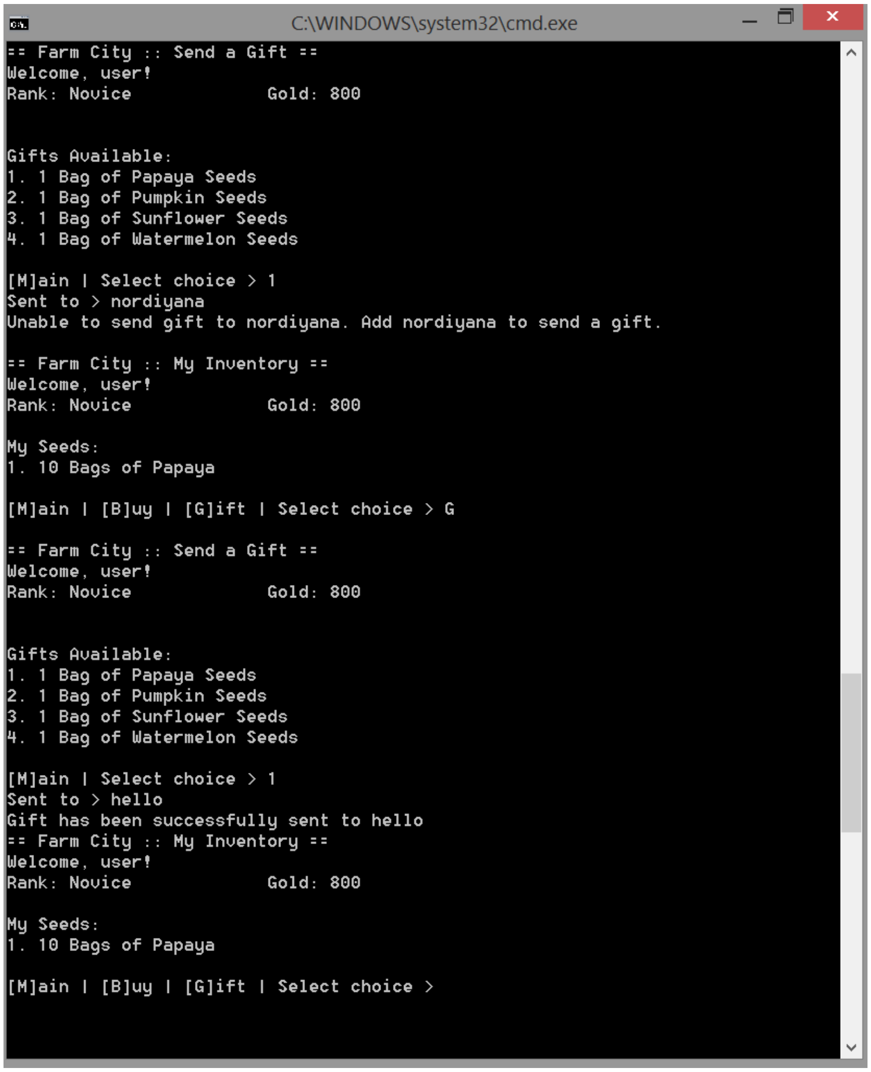
- Inventory menu lists the variety of seeds that the user has bought along with the quantity.

- To purchase seeds, users will select B to enter the store.

Store Menu

- Store sells a variety of seeds (in bags) to the player.
- Users will select the seed ID and the quantity they wish to buy.
- All purchased seeds will be added to the inventory list.

MY INVENTORY (BUYING SEEDS/**GIFTING**)



```
== Farm City :: Send a Gift ==
Welcome, user!
Rank: Novice              Gold: 800


Gifts Available:
1. 1 Bag of Papaya Seeds
2. 1 Bag of Pumpkin Seeds
3. 1 Bag of Sunflower Seeds
4. 1 Bag of Watermelon Seeds

[M]ain | Select choice > 1
Sent to > nordiyana
Unable to send gift to nordiyana. Add nordiyana to send a gift.

== Farm City :: My Inventory ==
Welcome, user!
Rank: Novice              Gold: 800

My Seeds:
1. 10 Bags of Papaya

[M]ain | [B]uy | [G]ift | Select choice > G

== Farm City :: Send a Gift ==
Welcome, user!
Rank: Novice              Gold: 800


Gifts Available:
1. 1 Bag of Papaya Seeds
2. 1 Bag of Pumpkin Seeds
3. 1 Bag of Sunflower Seeds
4. 1 Bag of Watermelon Seeds

[M]ain | Select choice > 1
Sent to > hello
Gift has been successfully sent to hello
== Farm City :: My Inventory ==
Welcome, user!
Rank: Novice              Gold: 800

My Seeds:
1. 10 Bags of Papaya

[M]ain | [B]uy | [G]ift | Select choice >
```

[VALIDATION]

Inventory Menu

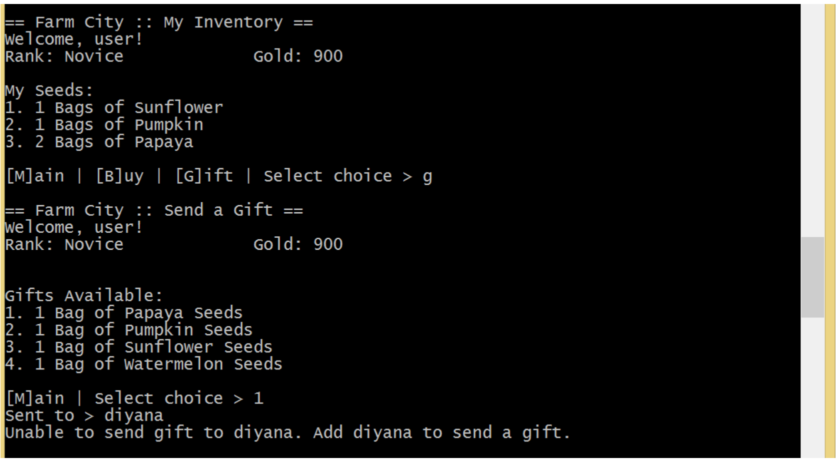
- Inventory menu lists the variety of seeds that the user has bought along with the quantity.

- To send gifts, users will select G to enter the gift menu.

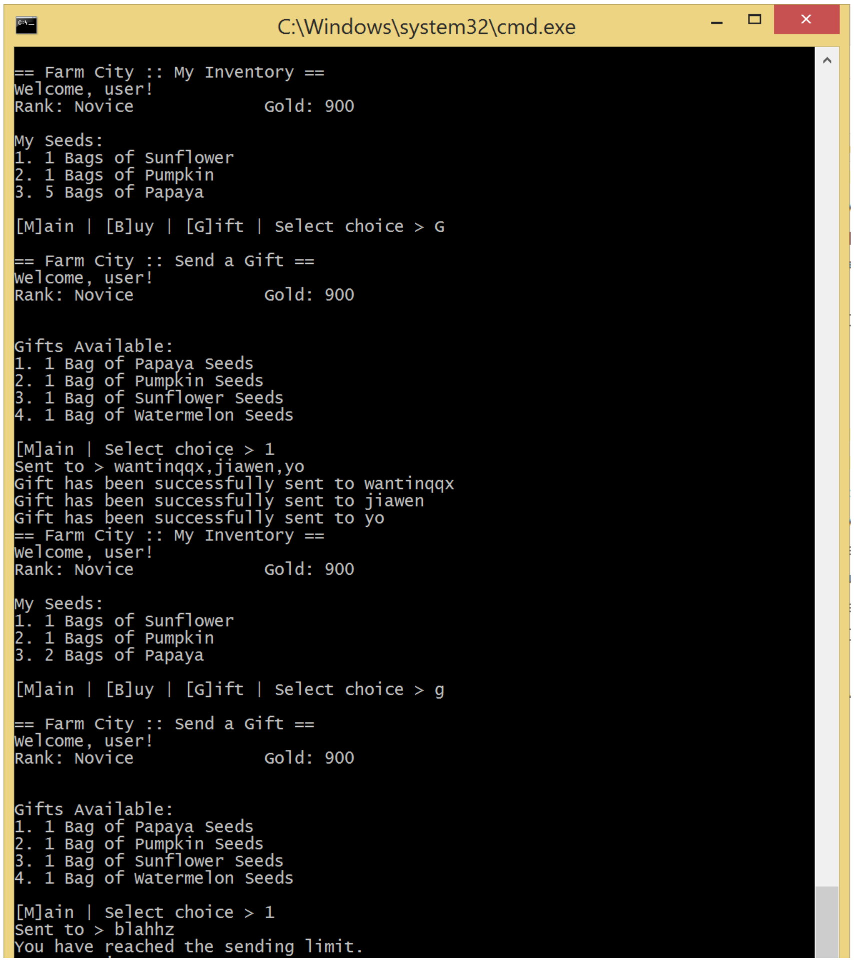
Gift Menu

- Users will select the type of seed they wish to send from their inventory, along with the username.
- If the user is not friends with the chosen username, an error message will be displayed
- If gift is sent successfully, the system will reduce the seed in the inventory by one and increase the recipient’s inventory by the same amount.

MY INVENTORY (BUYING SEEDS/**GIFTING**)



```
== Farm City :: My Inventory ==
Welcome, user!
Rank: Novice                Gold: 900

My Seeds:
1. 1 Bags of Sunflower
2. 1 Bags of Pumpkin
3. 2 Bags of Papaya

[M]ain | [B]uy | [G]ift | Select choice > g

== Farm City :: Send a Gift ==
Welcome, user!
Rank: Novice                Gold: 900


Gifts Available:
1. 1 Bag of Papaya Seeds
2. 1 Bag of Pumpkin Seeds
3. 1 Bag of Sunflower Seeds
4. 1 Bag of Watermelon Seeds

[M]ain | Select choice > 1
Sent to > diyana
Unable to send gift to diyana. Add diyana to send a gift.
```



```
== Farm City :: My Inventory ==
Welcome, user!
Rank: Novice                Gold: 900

My Seeds:
1. 1 Bags of Sunflower
2. 1 Bags of Pumpkin
3. 5 Bags of Papaya

[M]ain | [B]uy | [G]ift | Select choice > G

== Farm City :: Send a Gift ==
Welcome, user!
Rank: Novice                Gold: 900


Gifts Available:
1. 1 Bag of Papaya Seeds
2. 1 Bag of Pumpkin Seeds
3. 1 Bag of Sunflower Seeds
4. 1 Bag of Watermelon Seeds

[M]ain | Select choice > 1
Sent to > wantingqx,jiawen,yo
Gift has been successfully sent to wantingqx
Gift has been successfully sent to jiawen
Gift has been successfully sent to yo
== Farm City :: My Inventory ==
Welcome, user!
Rank: Novice                Gold: 900

My Seeds:
1. 1 Bags of Sunflower
2. 1 Bags of Pumpkin
3. 2 Bags of Papaya

[M]ain | [B]uy | [G]ift | Select choice > g

== Farm City :: Send a Gift ==
Welcome, user!
Rank: Novice                Gold: 900


Gifts Available:
1. 1 Bag of Papaya Seeds
2. 1 Bag of Pumpkin Seeds
3. 1 Bag of Sunflower Seeds
4. 1 Bag of Watermelon Seeds

[M]ain | Select choice > 1
Sent to > blahhz
You have reached the sending limit.
```

[VALIDATION]

Gift Menu

- Users can send up to 5 gifts per day, to multiple friends, but not to themselves.
- If the user is not friends with the chosen username, an error message will be displayed
- If gift is sent successfully, the system will reduce the seed in the user's inventory by one and increase the recipient's inventory
- If the user exceeds the sending limit, an error message will be displayed. Gift limits are reset at 00:00hours each day.