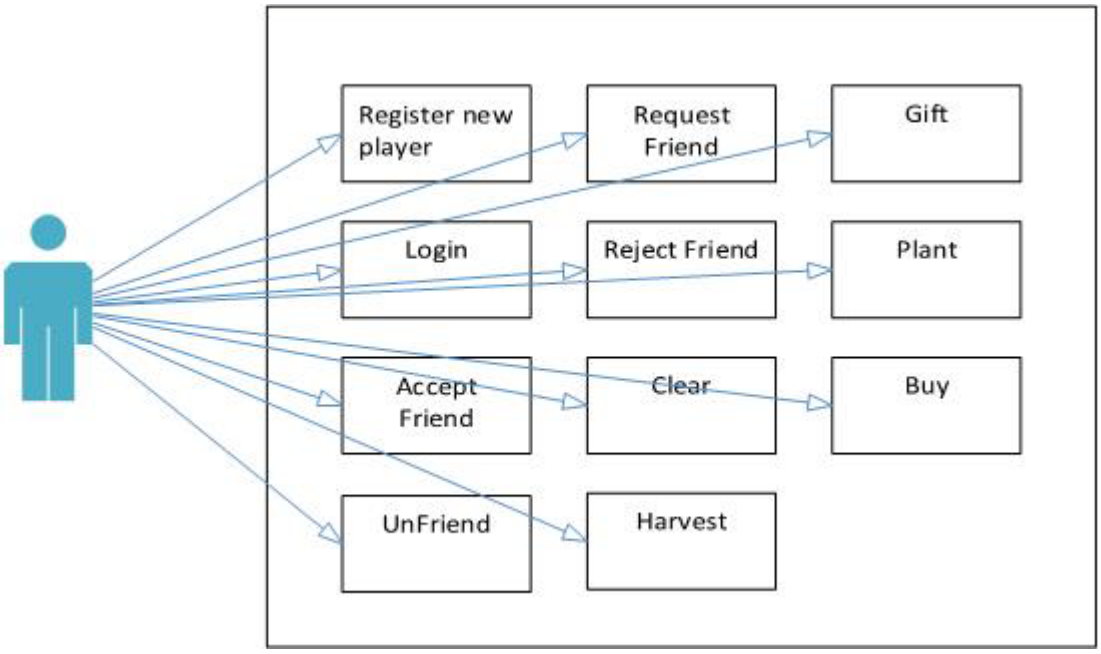
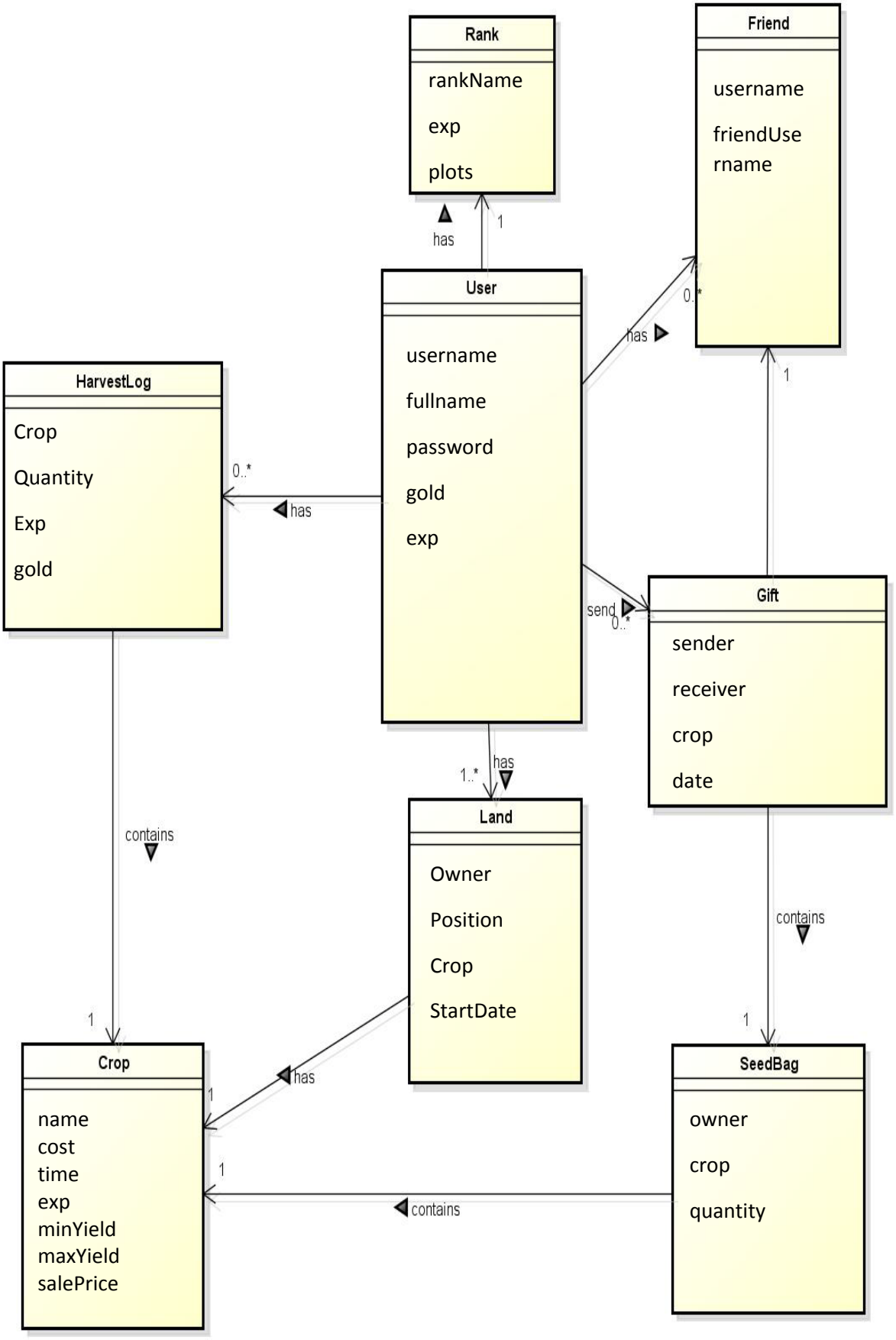
Contents

1. Use case diagram



Use Case	Description
Register new player	This use case allows public to register as a new farmer.
Login	This use case allows player to login.
Accept friend	This use case allows a player to accept another player's friend request.
Unfriend	This use case allows player to unfriend another player.
Request friend	This use case allows player to request to be a friend of another player.
Reject friend	This use case allows a player to remove another player from his friend list.
Plant	This use case allows player to plant crops
Clear	This use case allows player to clear away wilted crop
Harvest	This use case allows player to harvest crops
Buy	This use case allows player to buy seeds
Gift	This use case allows player to send seeds to other player

2. Domain diagram



3. Use case scenarios

3.1 Register new player use case

Actor: Player

Precondition: None

Main flow of events:

1. The use case begins when the player choose to register as a new farmer.

2. The system prompts player for username.

3. The player enters the username.

4. The system prompts player for full name.

5. The player enters the full name.

6. The system prompts the player for password.

7. The player enters the password.

8. The system prompts user to confirm password.

9. The player re-enters the password.

10. The system verifies the two passwords match, creates a new farmer profile and inform player that account is created successfully.

11. The use case ends when the player is brought back to the login page.

Alternate flow 9a: Password does not match first input.

1. System informs player that password does not match.

2. System goes back to main flow 6.

Alternate flow 9b: Username is taken.

1. System informs player that username is in use.

2. System goes back to the main flow 2.

3.2 Login use case

Actor: Player

Precondition: None

Main flow of events:

1. This use case begins when player choose to login.

2. The system prompts player for the username.

3. Player enters the username.

4. The system prompts player for the password.

5. Player enters the password.

6. System verifies the username and password as valid and direct player to the main menu.

7. The use case ends.

Alternate flow 5a: Username or Password is invalid

1. System informs player that username or password is invalid.

2. Return to main flow 2.

3.3 Accept friend use case

Actor: Player

Precondition: Player is logged in and in My Friend menu

Main flow of events:

1. This use case begins when the player choose to accept another player.

2. System will add friend into player's friend list and displays the accepted friend. After which, it will display "My Friends" menu.

3. The use can ends when player choose to return to main menu.

Alternate flow 1a: Input is invalid

1. System informs player that input is invalid.

2. Return to main flow 1.

3.4 Unfriend use case

Actor: Player

Precondition: Player is logged in and in My Friend menu

Main flow of events:

1. This use case begins when the player choose to unfriend another player.

2. System will remove friend from player's friend list and displays the removed friend. After which, it will display "My Friends" menu.

3. The use can ends when player choose to return to main menu.

Alternate flow 1a: Input is invalid

1. System informs player that input is invalid.

2. Return to main flow 1.

3.5 Request friend use case

Actor: Player

Precondition: Player is logged in and in My Friend menu

Main flow of events:

1. This use case begins when the player choose to request to be a friend of another player.

2. System will send a request to player's request list and displays the requested friend. After which, it will display "My Friends" menu.

3. The use can ends when player choose to return to main menu.

Alternate flow 1a: Input is invalid

1. System informs player that input is invalid.

2. Return to main flow 1.

3.6 Reject friend use case

Actor: Player

Precondition: Player is logged in and in My Friend menu

Main flow of events:

1. This use case begins when the player choose to reject a friend request from another player.

2. System will remove requested player's name from player's request list and displays the rejected friend. After which, it will display "My Friends" menu.

3. The use can ends when player choose to return to main menu.

Alternate flow 1a: Input is invalid

1. System informs player that input is invalid.

2. Return to main flow 1.

3.7Plant use case

Actor: Player

Precondition: Player is logged in and in My Farm menu

Main flow of events:

1. This use case begins when the player choose to plant a seed on a chosen piece of land.

2. System displays the player's list of seeds and prompt player to enter the choice of seed to plant.

3. Player enters choice of seed to plant.

4. System displays the seed that was planted on preferred land.

5. The use case ends when user choose to return to main menu.

3.8 Clear use case

Actor: Player

Precondition: Player is logged in and in My Farm menu

Main flow of events:

1. This use case begins when the player choose to clear a land.

2. System displays the land that was being cleared and deduct 5 gold coins from player's account.

3. System repeats step 1 and 2 till user choose to return to main menu.

3.10 Harvest use case

Actor: Player

Precondition: Player is logged in and in My Farm menu

Main flow of events:

1. This use case begins when the player choose to harvest a land.

2. System will add XP and gold coins to player's account and display the units of crop harvested, the XP and gold coins gained.

3. System repeats step 1 and 2 till user choose to return to main menu.

3.11Buy use case

Actor: Player

Precondition: Player is logged in and in My inventory menu

Main flow of events:

1. This use case begins when the player choose to buy a bag of seed.

2. System will display list of all the seeds available and prompt user to select a bag of seed for purchase.

3. Player enters choice of seed to purchase.

4. System prompts player to input amount of bag of seeds to be purchased

5. Player enters a choice on amount of bag of seed to be purchased.

6. System will add the bag(s) of seeds into player's inventory and deduct the gold from his account.

7. Use case ends when user choose to return to main menu.

Alternate flow 5a: Input is invalid

1. System informs player that input is invalid.

2. Return to main flow 1.

3.12 Gift use case

Actor: Player

Precondition: Player is logged in and in My inventory menu

Main flow of events:

1. This use case begins when the player choose to gift a bag of seed.

2. System will display list of all the seeds available and prompt user to select a seed for gifting.

3. Player enters a choice of seed to gift.

4. System prompts player to input username of friend to be gifted

5. System will add the bag of seed into friend's inventory.

6. Use case ends when user choose to return to main menu.

Alternate flow 4a: Username not found

1. System informs player that username is invalid.

2. Return to main flow 4.

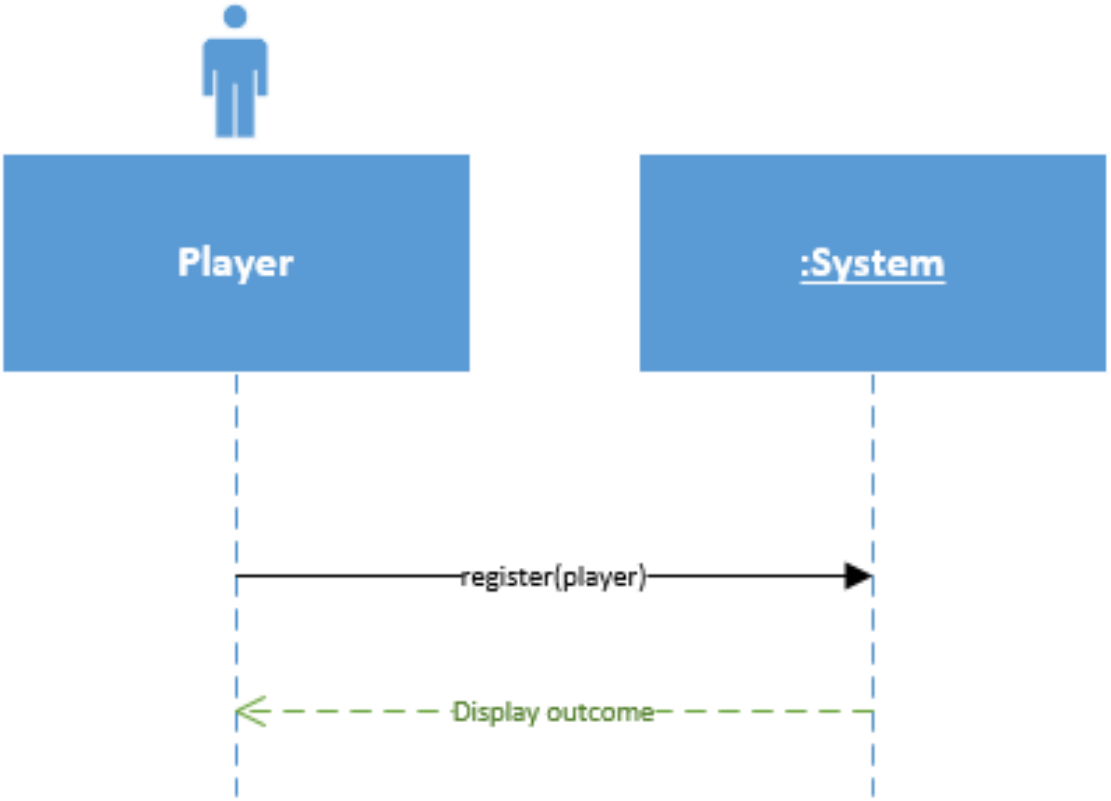
Alternate flow 4b: Username is not a friend of player

1. System informs player that input is invalid.

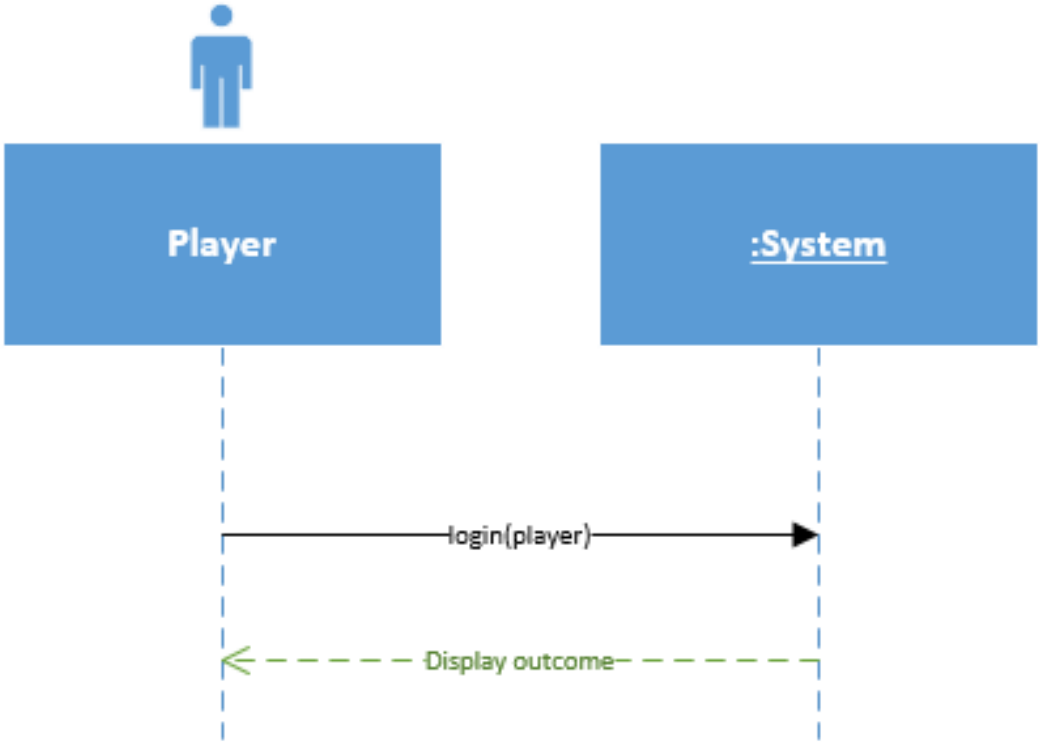
2. Return to main flow 1.

4. System sequence diagram

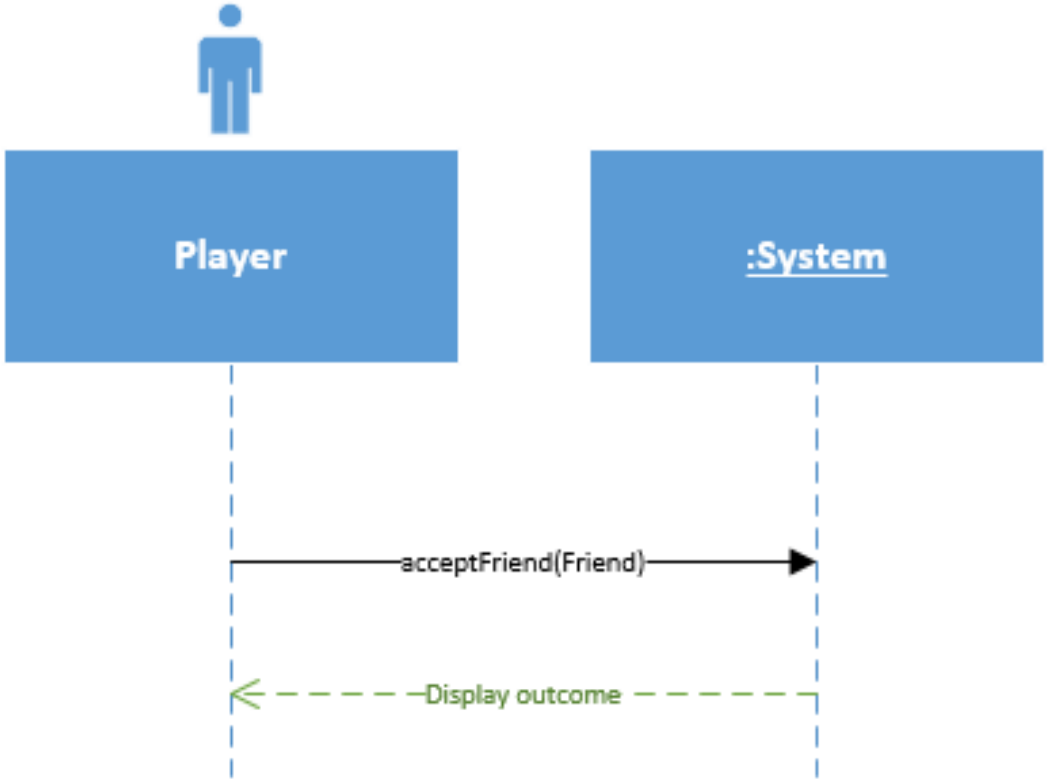
4.1 Register new player



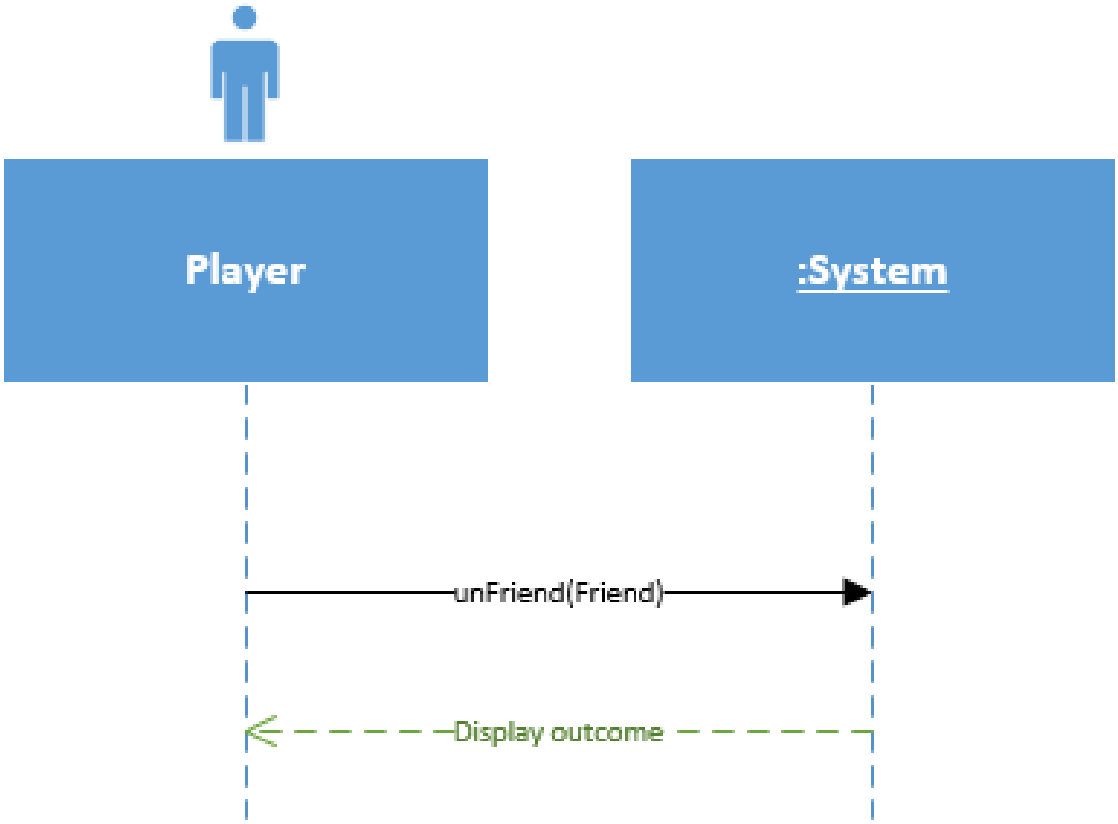
4.2 Login



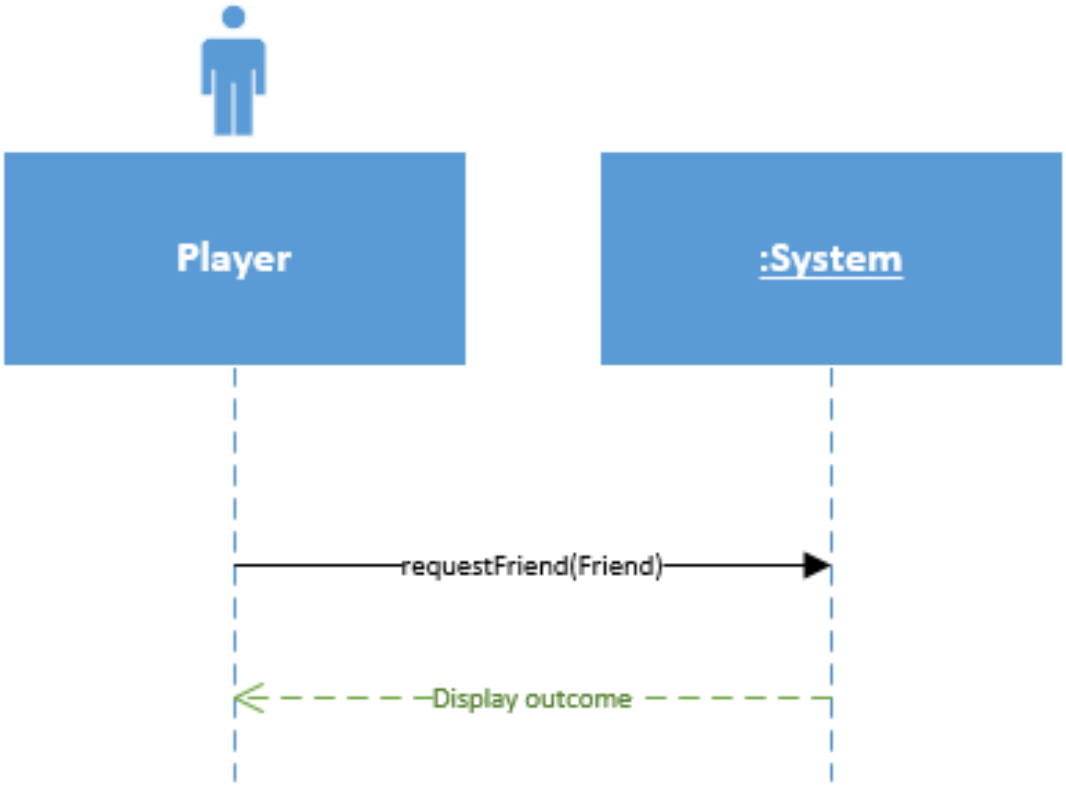
4.3Accept friend



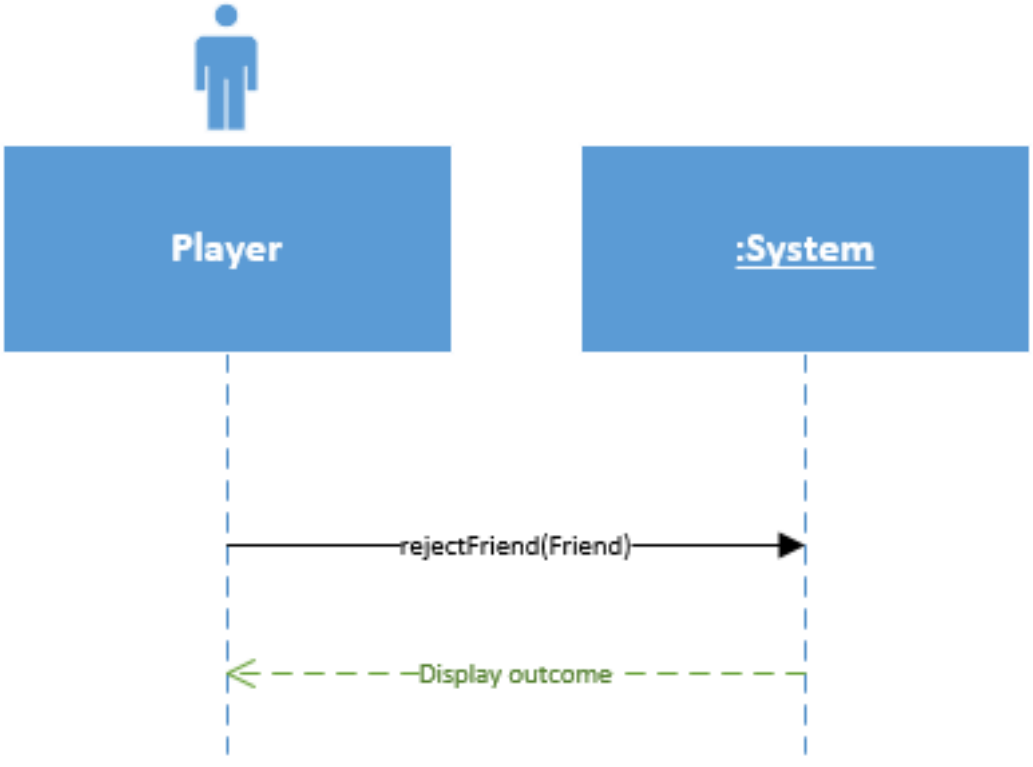
4.4 Unfriend


Player
:System
unFriend(Friend)
Display outcome

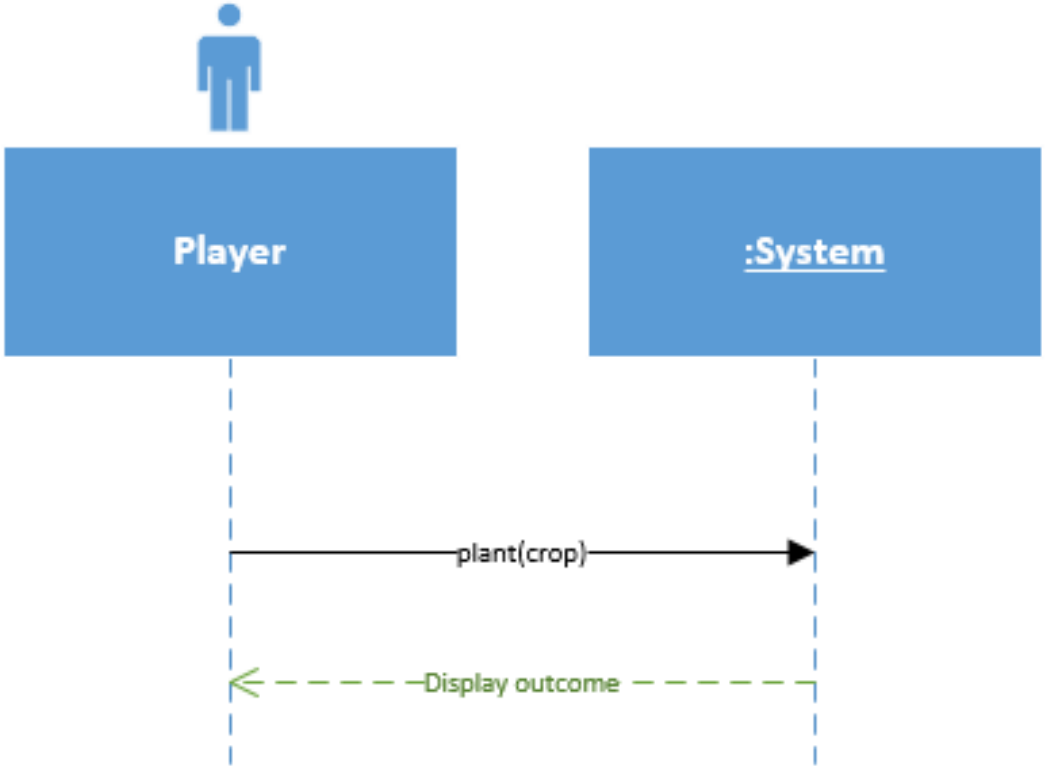
4.5 Request friend



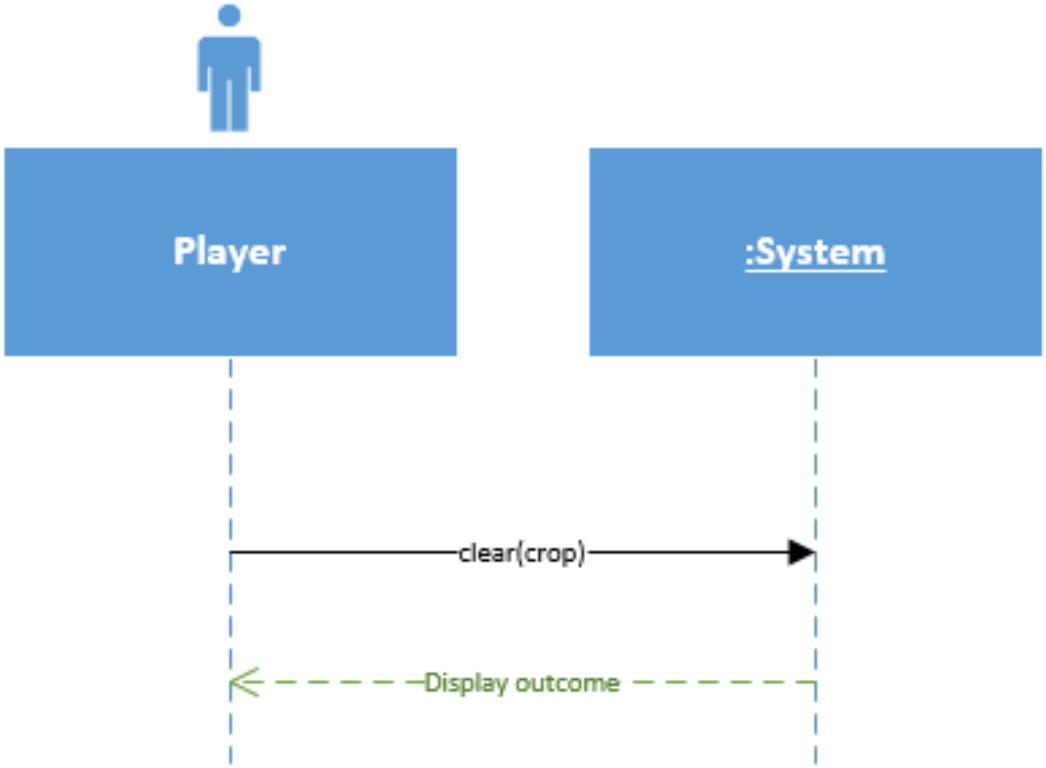
4.6 Reject friend



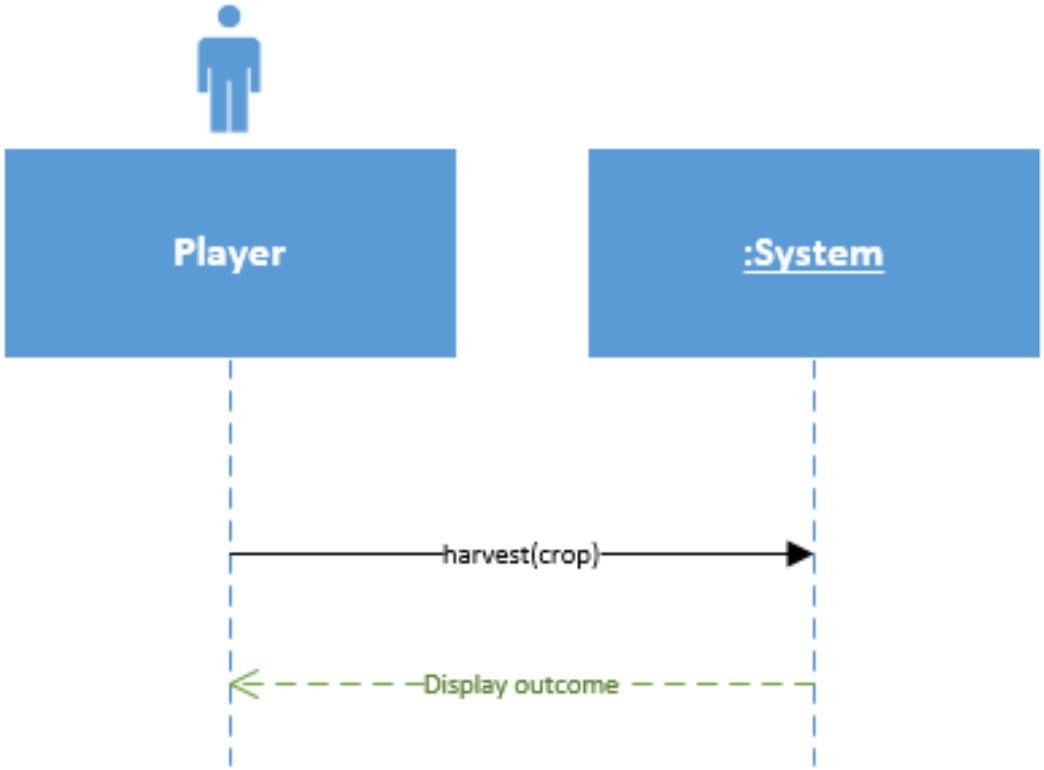
4.7 Plant



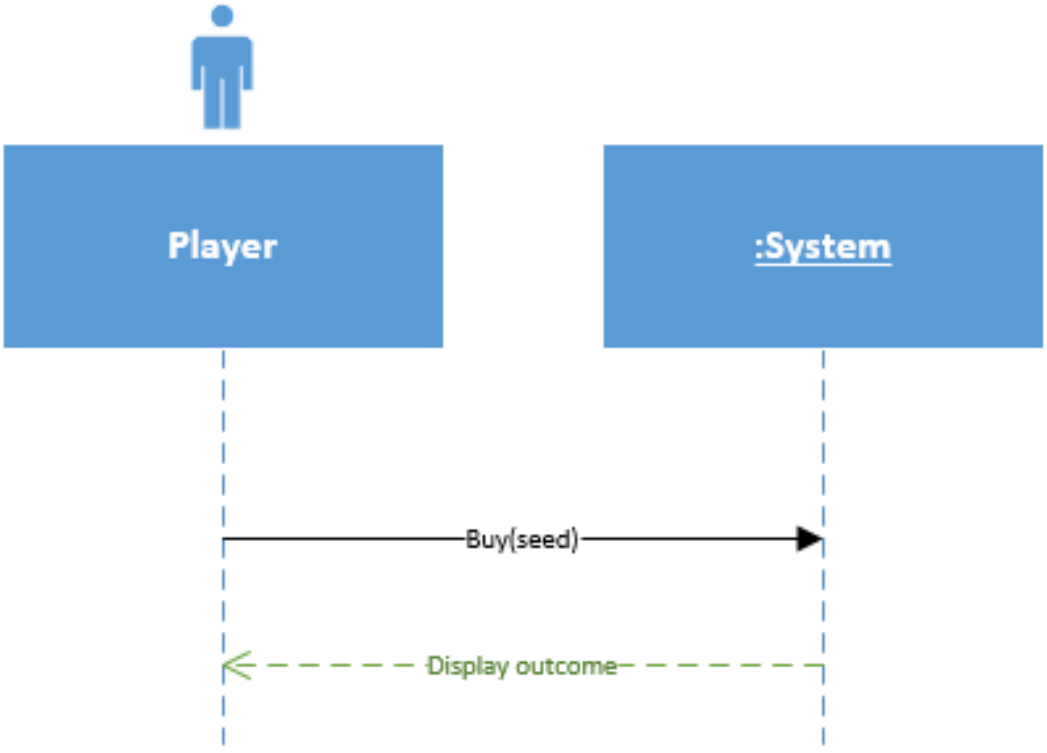
4.8 Clear



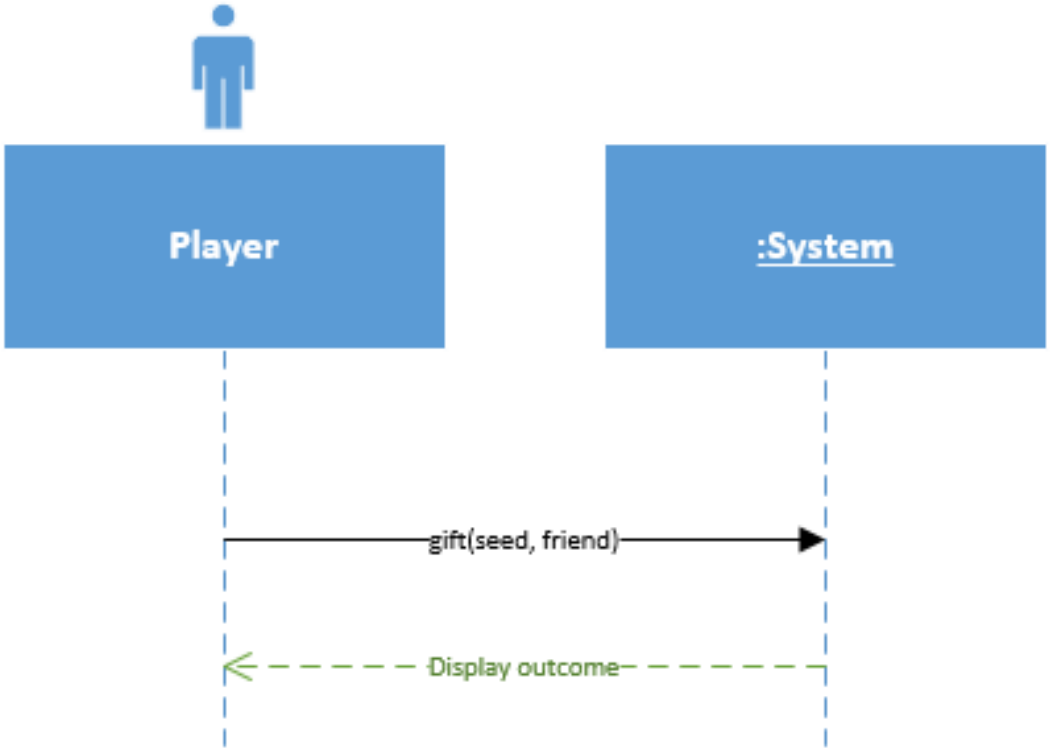
4.9 Harvest



4.10Buy

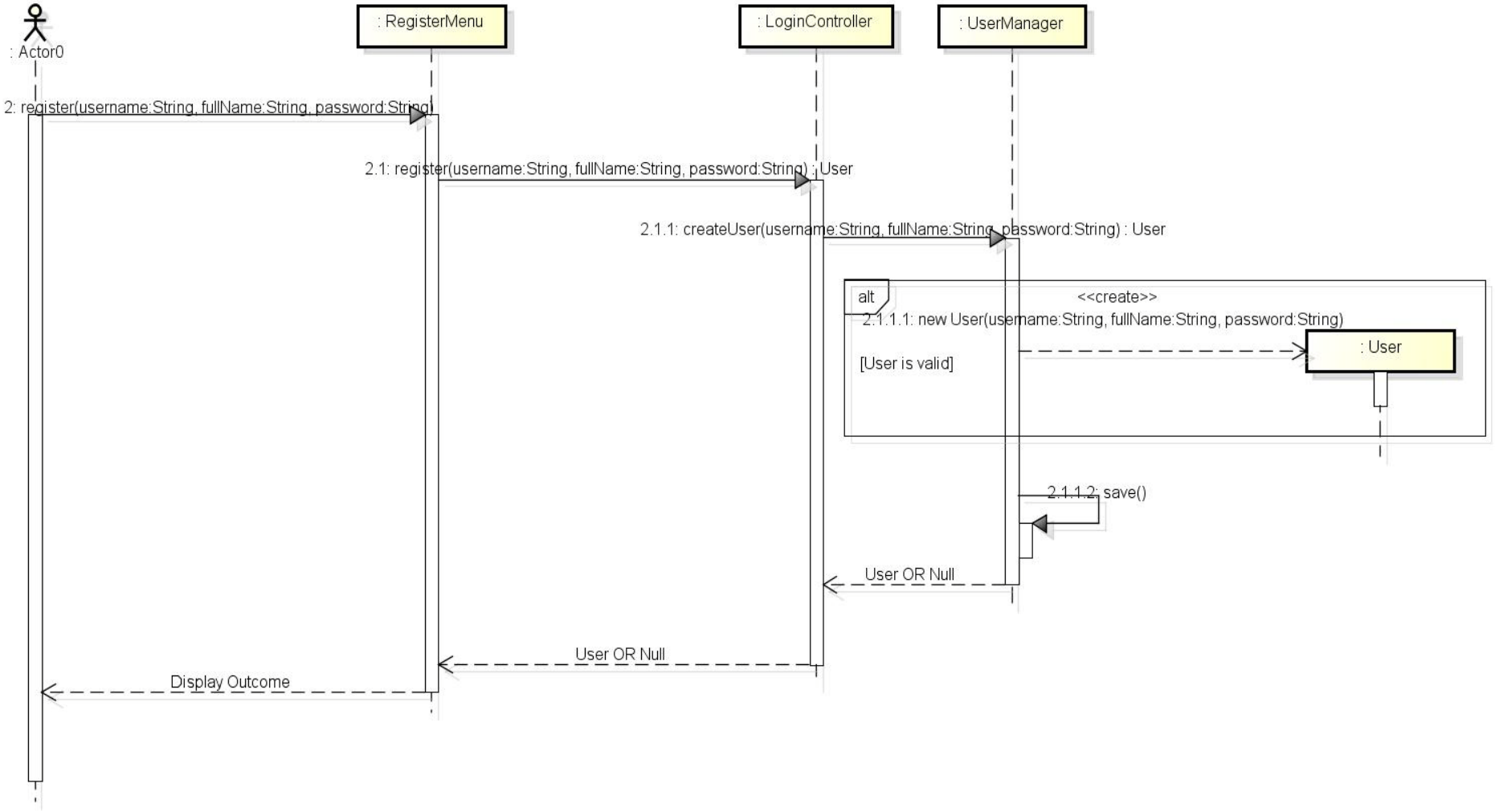


4.11 Gift

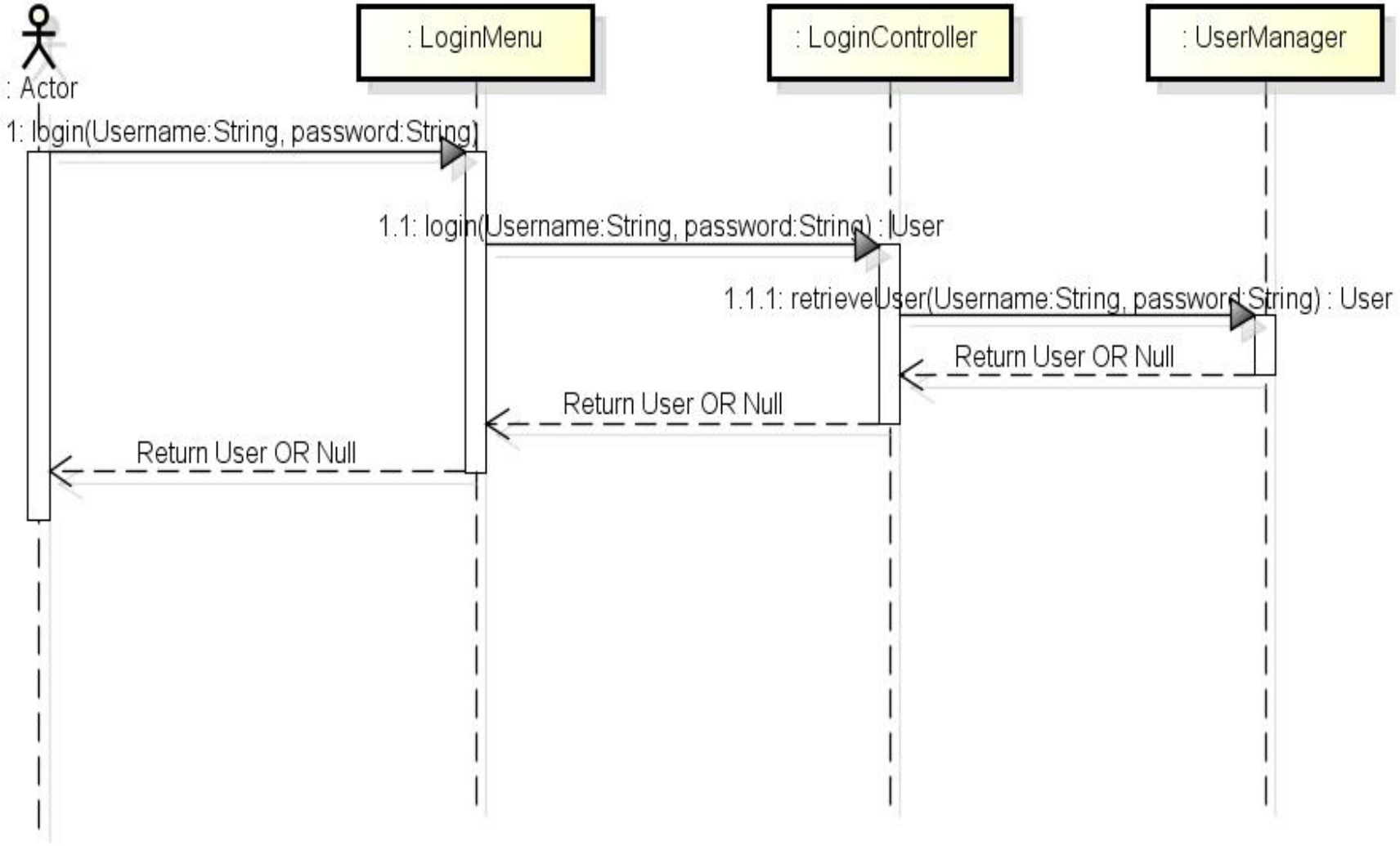


5. Sequence diagram

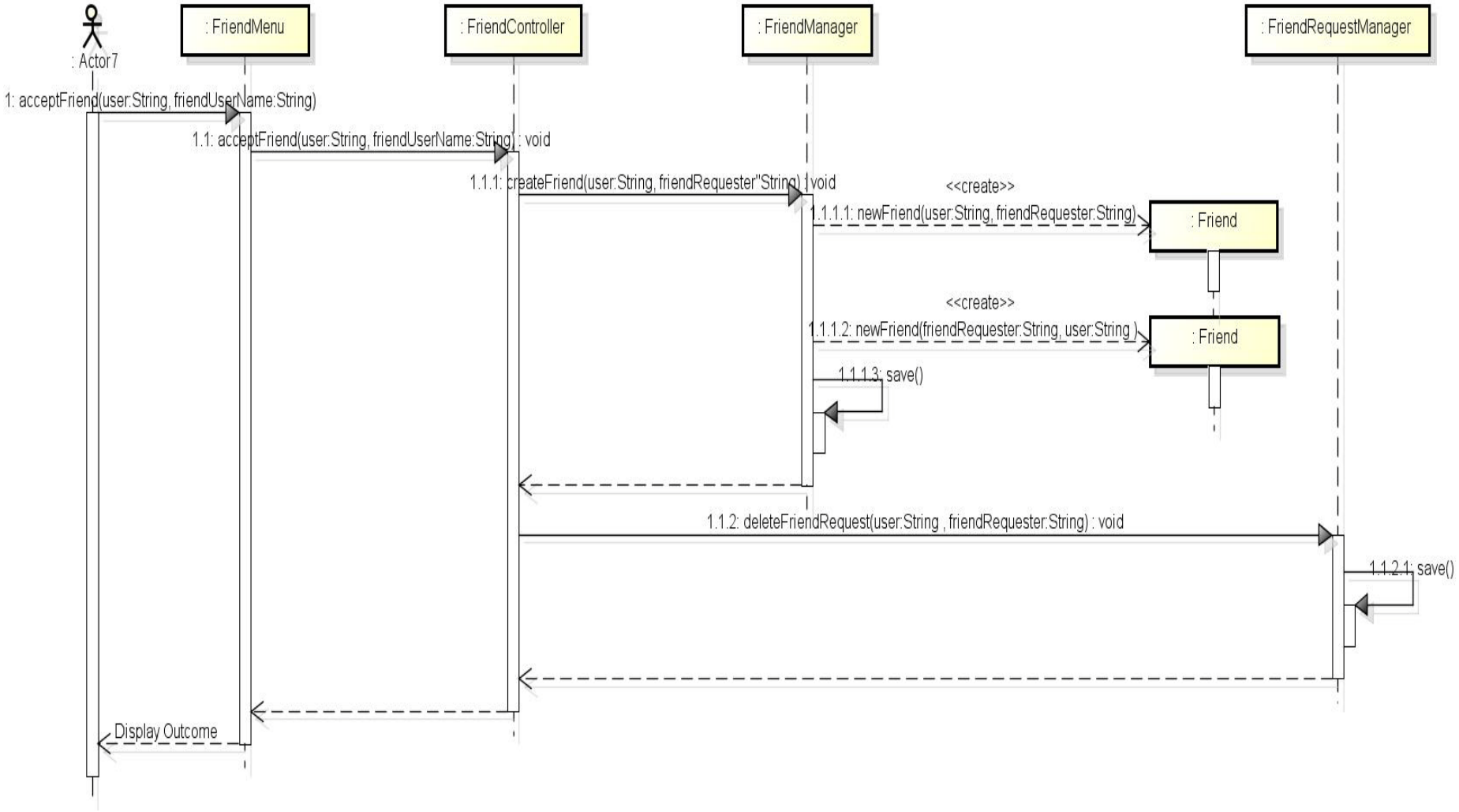
5.1 Register new player



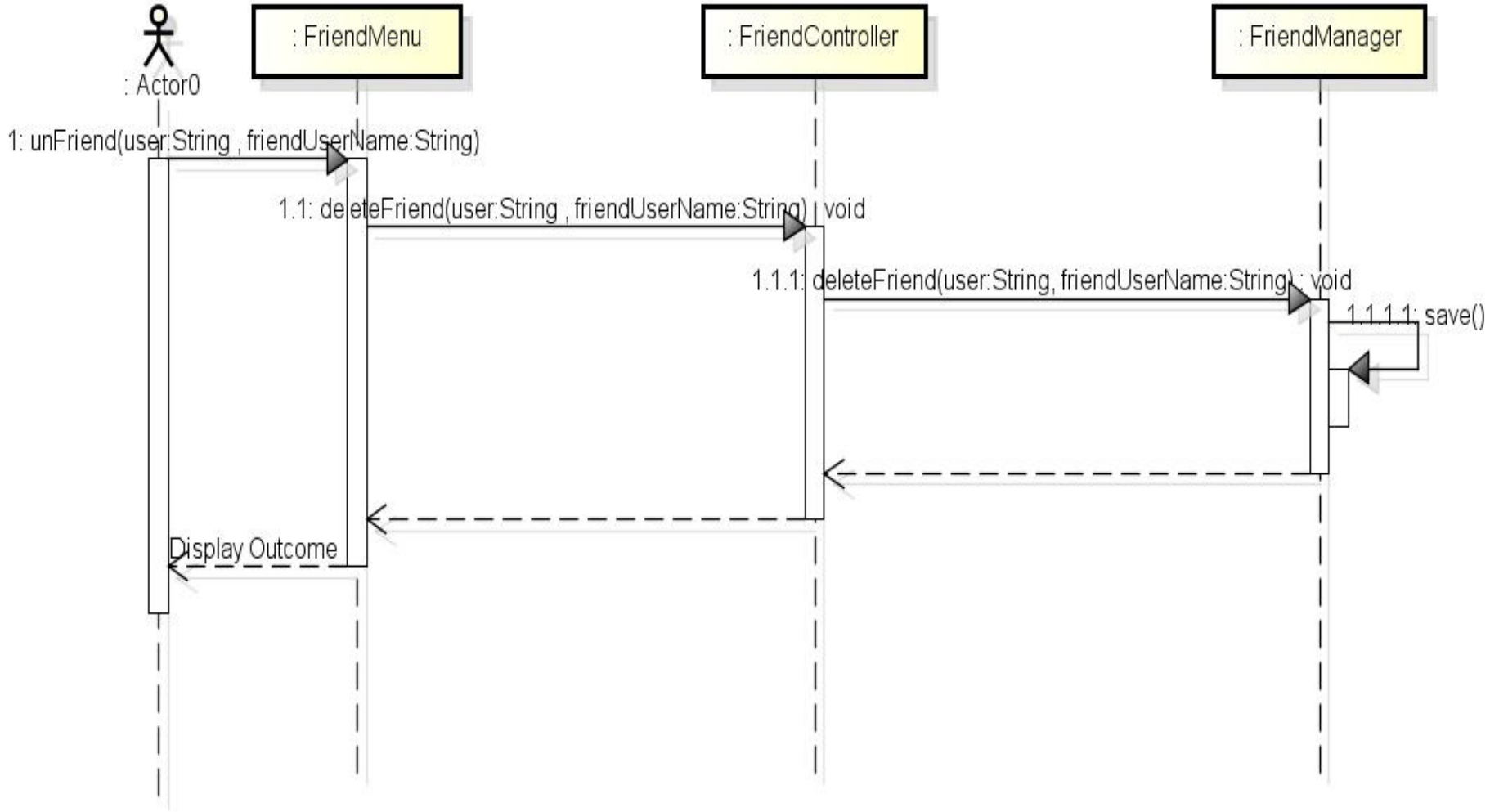
5.2 Login


: Actor
: LoginMenu
: LoginController
: UserManager
1: login(Username:String, password:String)
1.1: login(Username:String, password:String) : User
1.1.1: retrieveUser(Username:String, password:String) : User
Return User OR Null
Return User OR Null
Return User OR Null

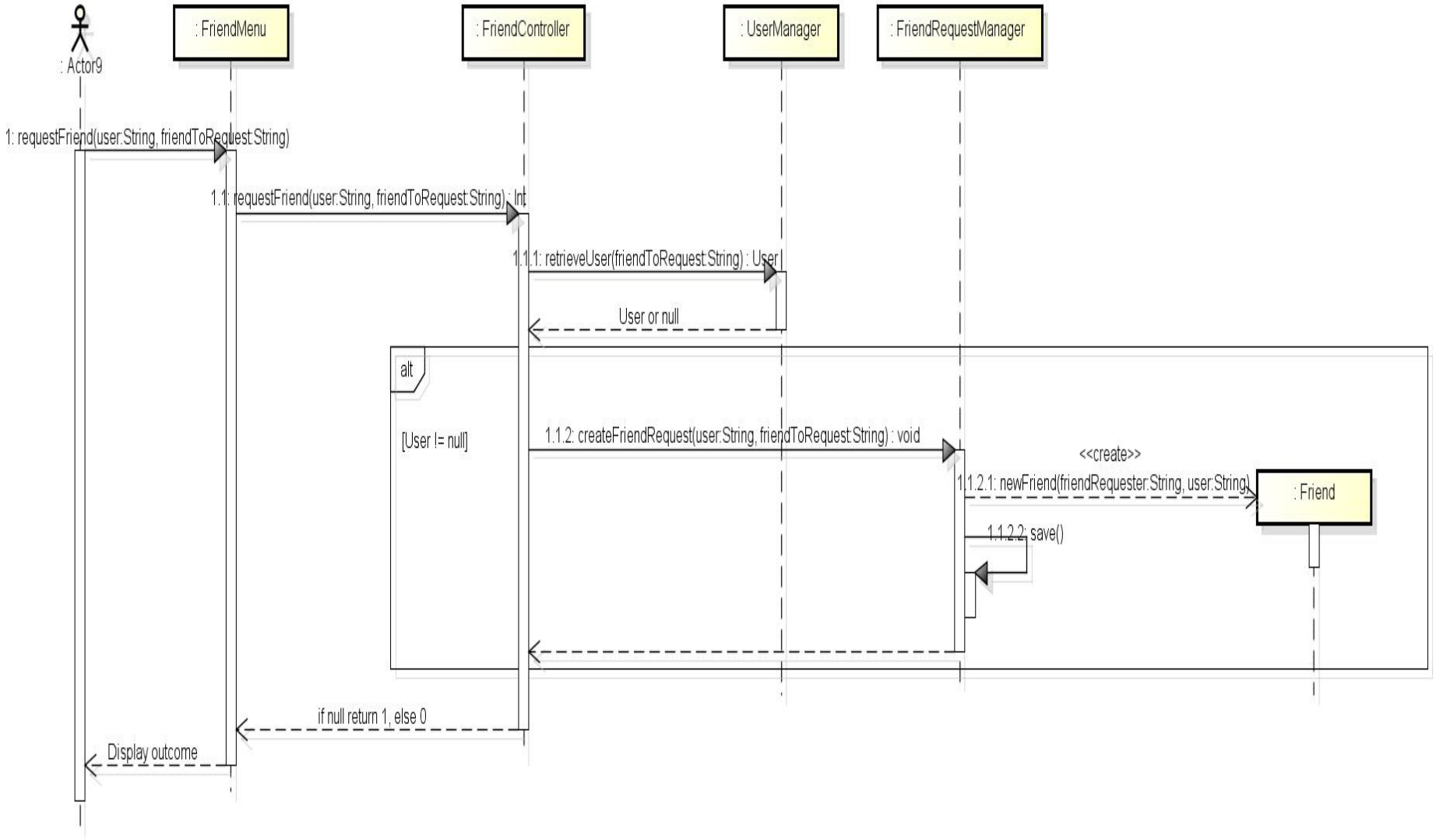
5.3 Accept Friend



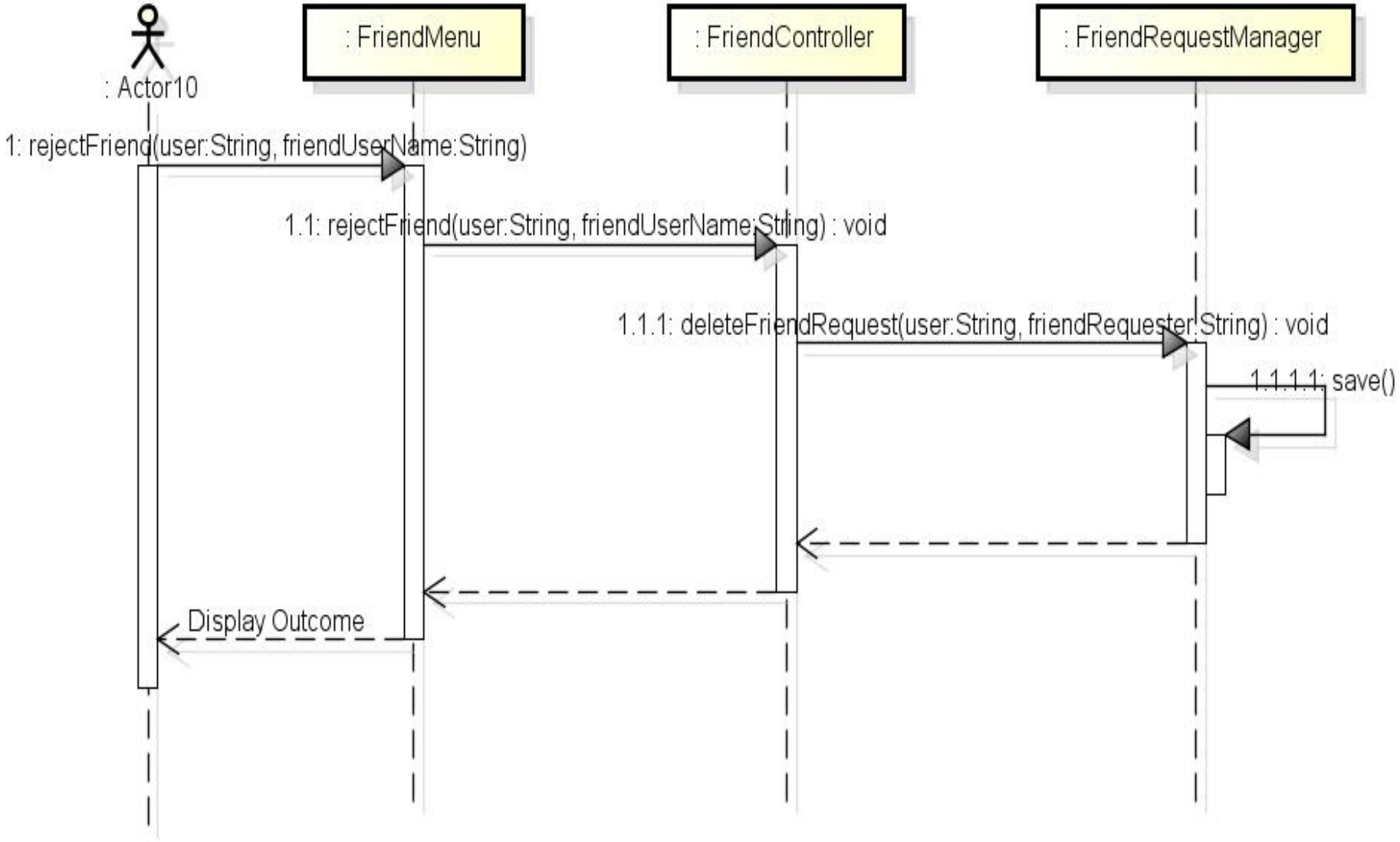
5.4 Unfriend


: Actor0
: FriendMenu
: FriendController
: FriendManager
1: unFriend(user:String , friendUserName:String)
1.1: deleteFriend(user:String , friendUserName:String) : void
1.1.1: deleteFriend(user:String, friendUserName:String) : void
1.1.1.1: save()
Display Outcome

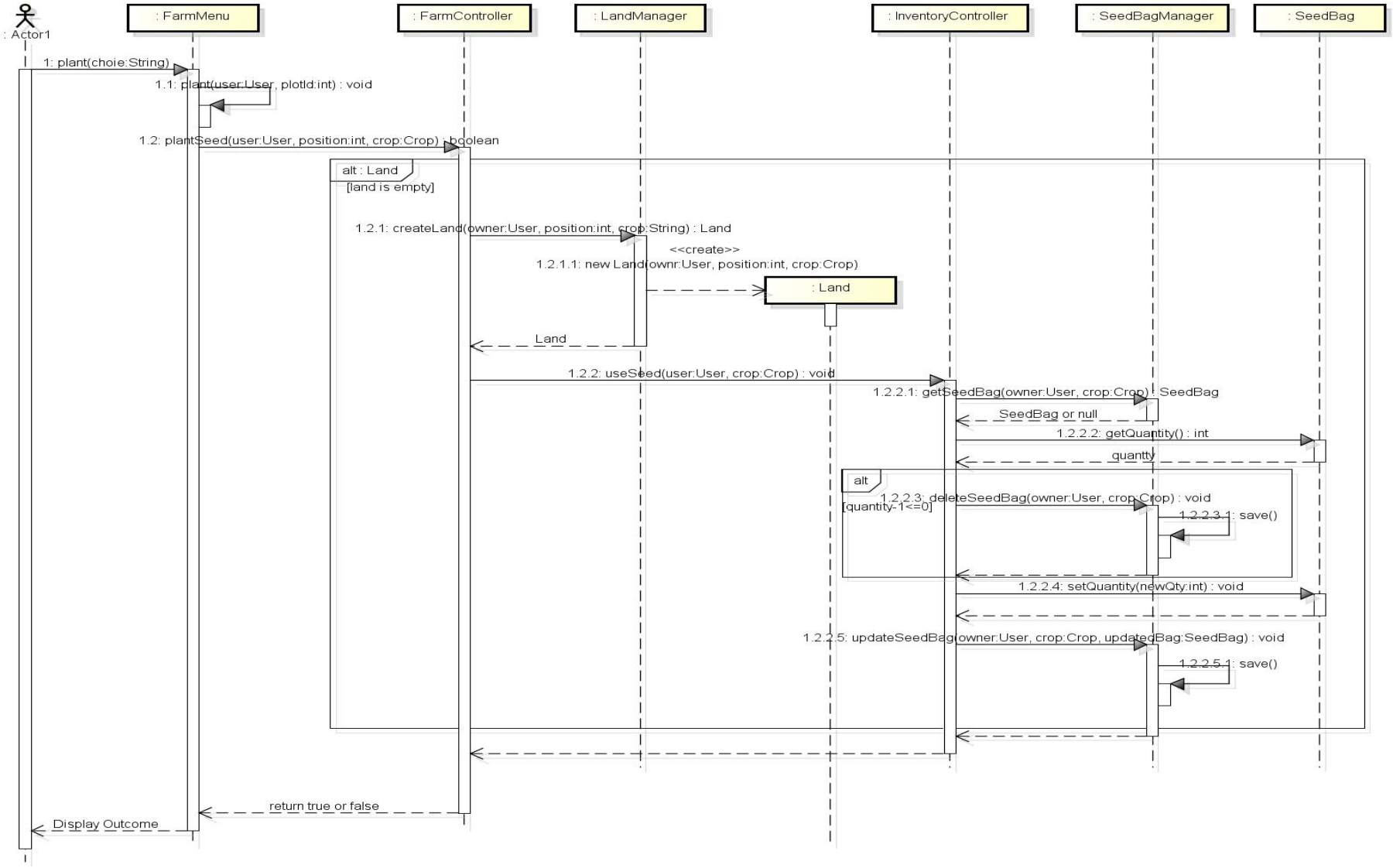
5.5 Request Friend



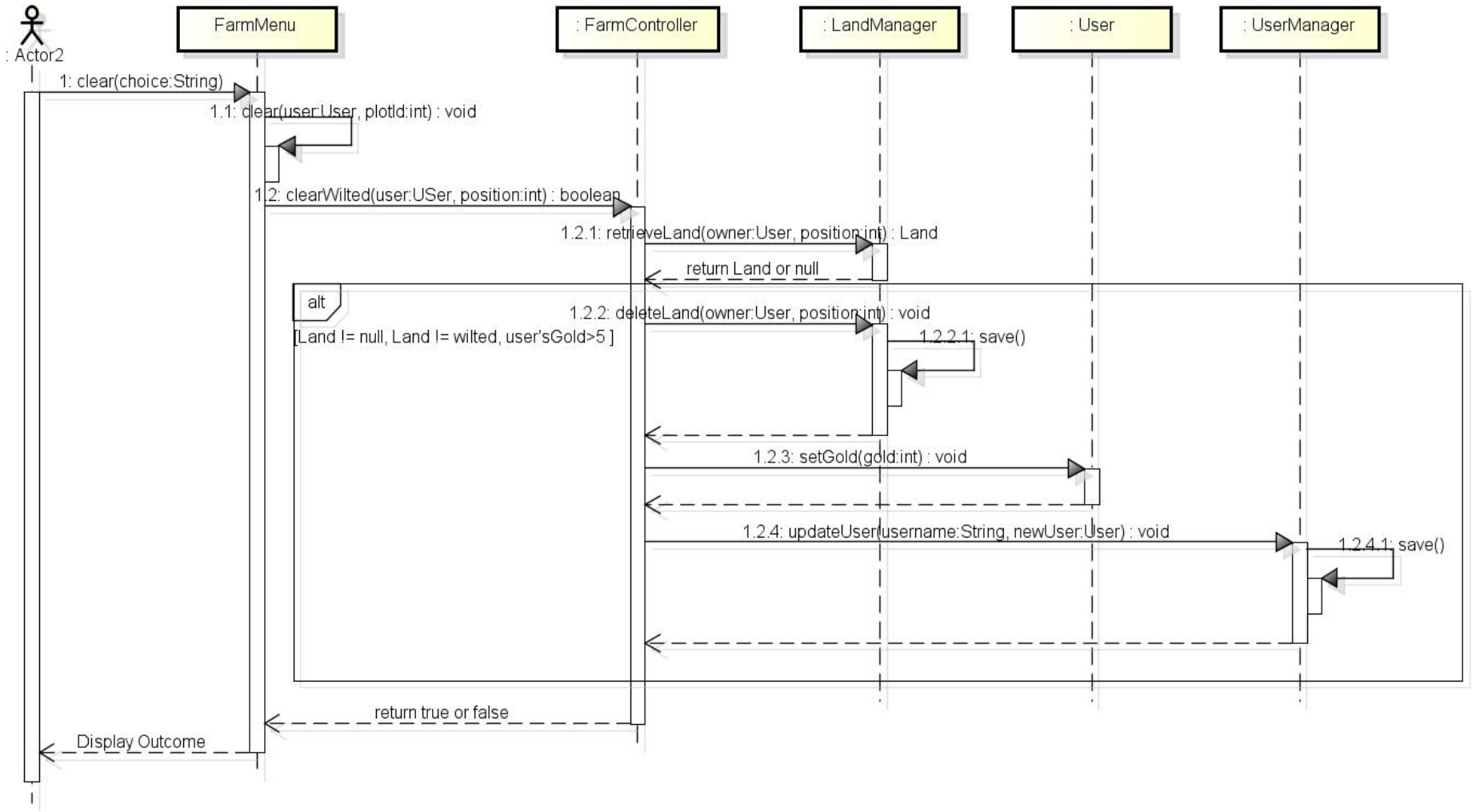
5.6 Reject Friend



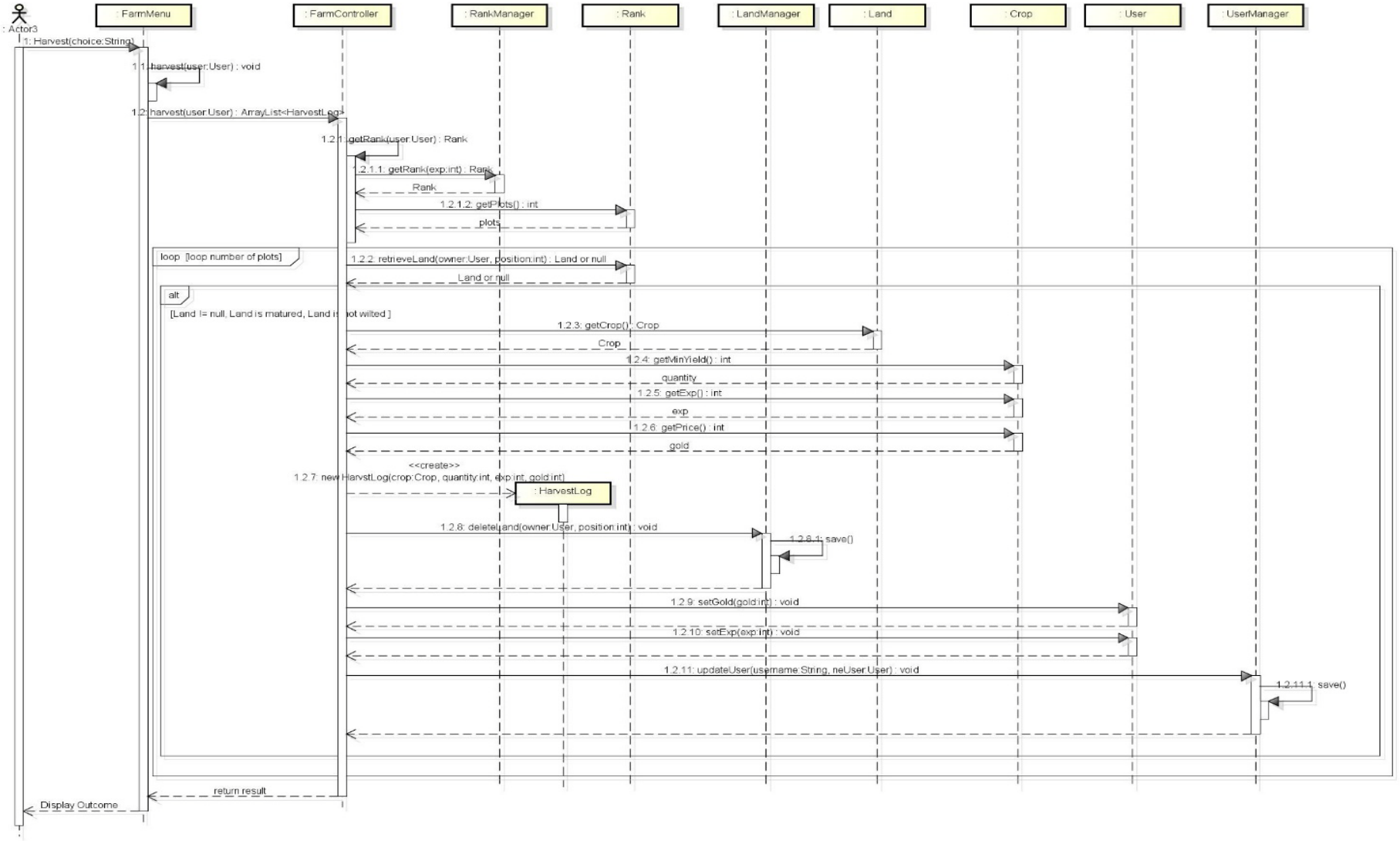
5.7 Plant


: Actor1
: FarmMenu
: FarmController
: LandManager
: InventoryController
: SeedBagManager
: SeedBag
1: plant(choie:String)
1.1: plant(user:User, plotId:int) : void
1.2: plantSeed(user:User, position:int, crop:Crop) : boolean
alt : Land
[land is empty]
1.2.1: createLand(owner:User, position:int, crop:String) : Land
<<create>>
1.2.1.1: new Land(ownr:User, position:int, crop:Crop)
: Land
Land
1.2.2: useSeed(user:User, crop:Crop) : void
1.2.2.1: getSeedBag(owner:User, crop:Crop) : SeedBag
SeedBag or null
1.2.2.2: getQuantity() : int
quantty
alt
[quantity-1<=0]
1.2.2.3: deleteSeedBag(owner:User, crop:Crop) : void
1.2.2.3.1: save()
1.2.2.4: setQuantity(newQty:int) : void
1.2.2.5: updateSeedBag(owner:User, crop:Crop, updatedBag:SeedBag) : void
1.2.2.5.1: save()
return true or false
Display Outcome

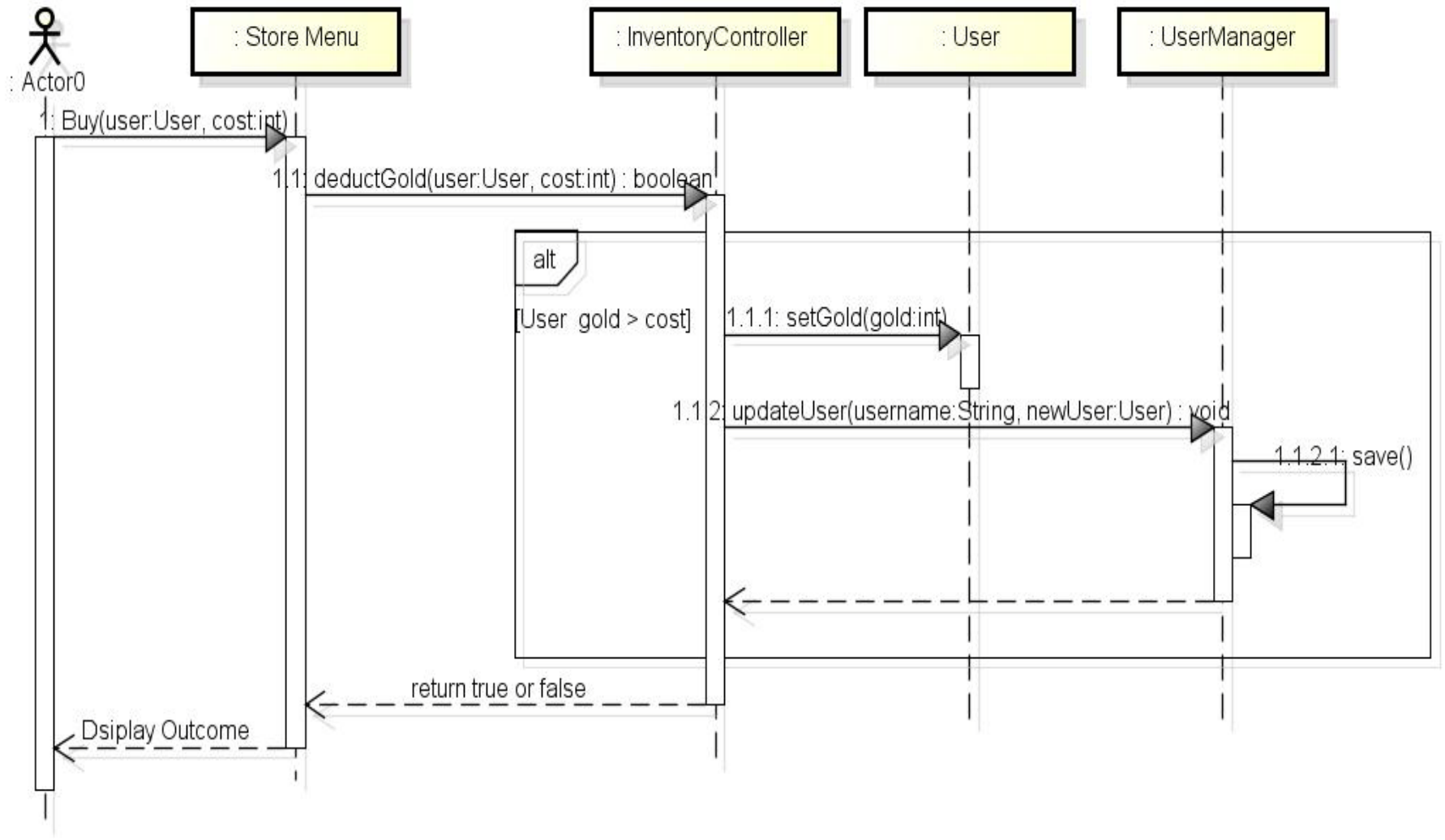
5.8 Clear


: Actor2
FarmMenu
: FarmController
: LandManager
: User
: UserManager
1: clear(choice:String)
1.1: clear(user:User, plotId:int) : void
1.2: clearWilted(user:USer, position:int) : boolean
1.2.1: retrieveLand(owner:User, position:int) : Land
return Land or null
alt
[Land != null, Land != wilted, user'sGold>5]
1.2.2: deleteLand(owner:User, position:int) : void
1.2.2.1: save()
1.2.3: setGold(gold:int) : void
1.2.4: updateUser(username:String, newUser:User) : void
1.2.4.1: save()
return true or false
Display Outcome

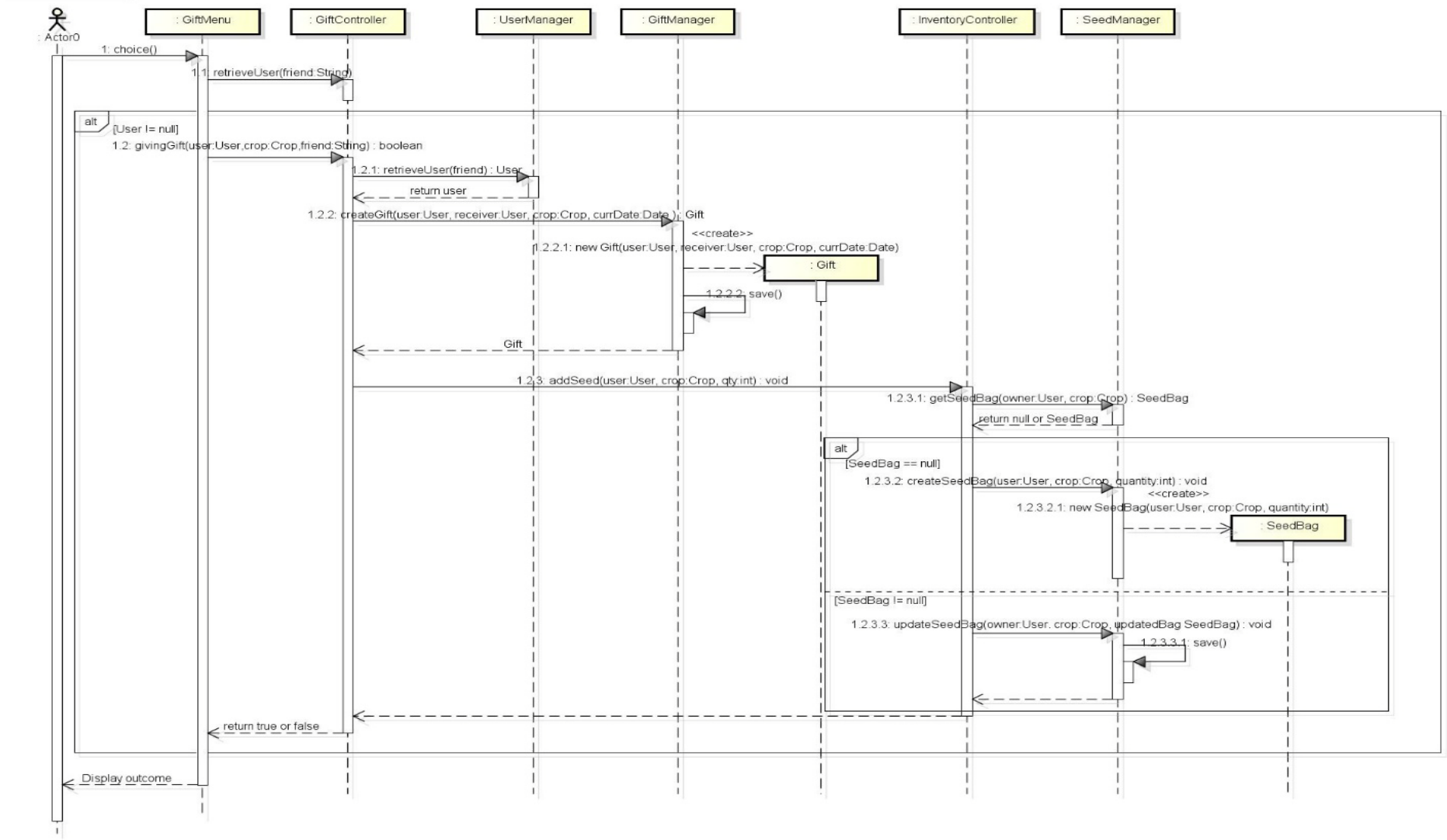
5.9 Harvest


: Actor3
: FarmMenu
: FarmController
: RankManager
: Rank
: LandManager
: Land
: Crop
: User
: UserManager
1: Harvest(choice:String)
1.1: harvest(user:User) : void
1.2: harvest(user:User) : ArrayList<HarvestLog>
1.2.1: getRank(user:User) : Rank
1.2.1.1: getRank(exp:int) : Rank
Rank
1.2.1.2: getPlots() : int
plots
loop [loop number of plots]
1.2.2: retrieveLand(owner:User, position:int) : Land or null
Land or null
alt
[Land != null, Land is matured, Land is not wilted]
1.2.3: getCrop() : Crop
Crop
1.2.4: getMinYield() : int
quantity
1.2.5: getExp() : int
exp
1.2.6: getPrice() : int
gold
<<create>>
1.2.7: new HarvstLog(crop:Crop, quantity:int, exp:int, gold:int)
: HarvestLog
1.2.8: deleteLand(owner:User, position:int) : void
1.2.8.1: save()
1.2.9: setGold(gold:int) : void
1.2.10: setExp(exp:int) : void
1.2.11: updateUser(username:String, neUser:User) : void
1.2.11.1: save()
return result
Display Outcome

5.10Buy

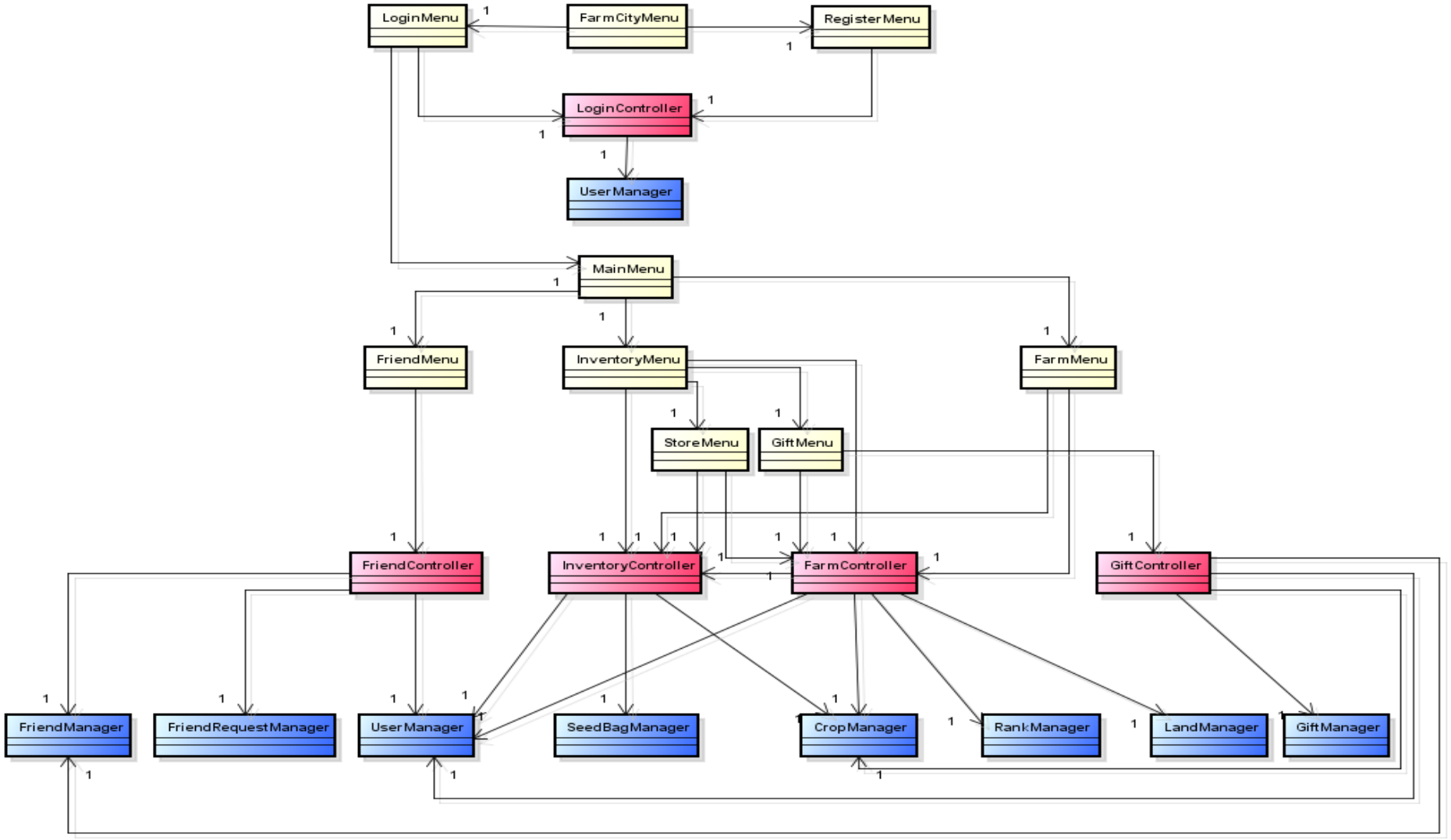

: Actor0
: Store Menu
: InventoryController
: User
: UserManager
1: Buy(user:User, cost:int)
1.1: deductGold(user:User, cost:int) : boolean
alt
[User gold > cost]
1.1.1: setGold(gold:int)
1.1.2: updateUser(username:String, newUser:User) : void
1.1.2.1: save()
return true or false
Dsiplay Outcome

5.11 Gift

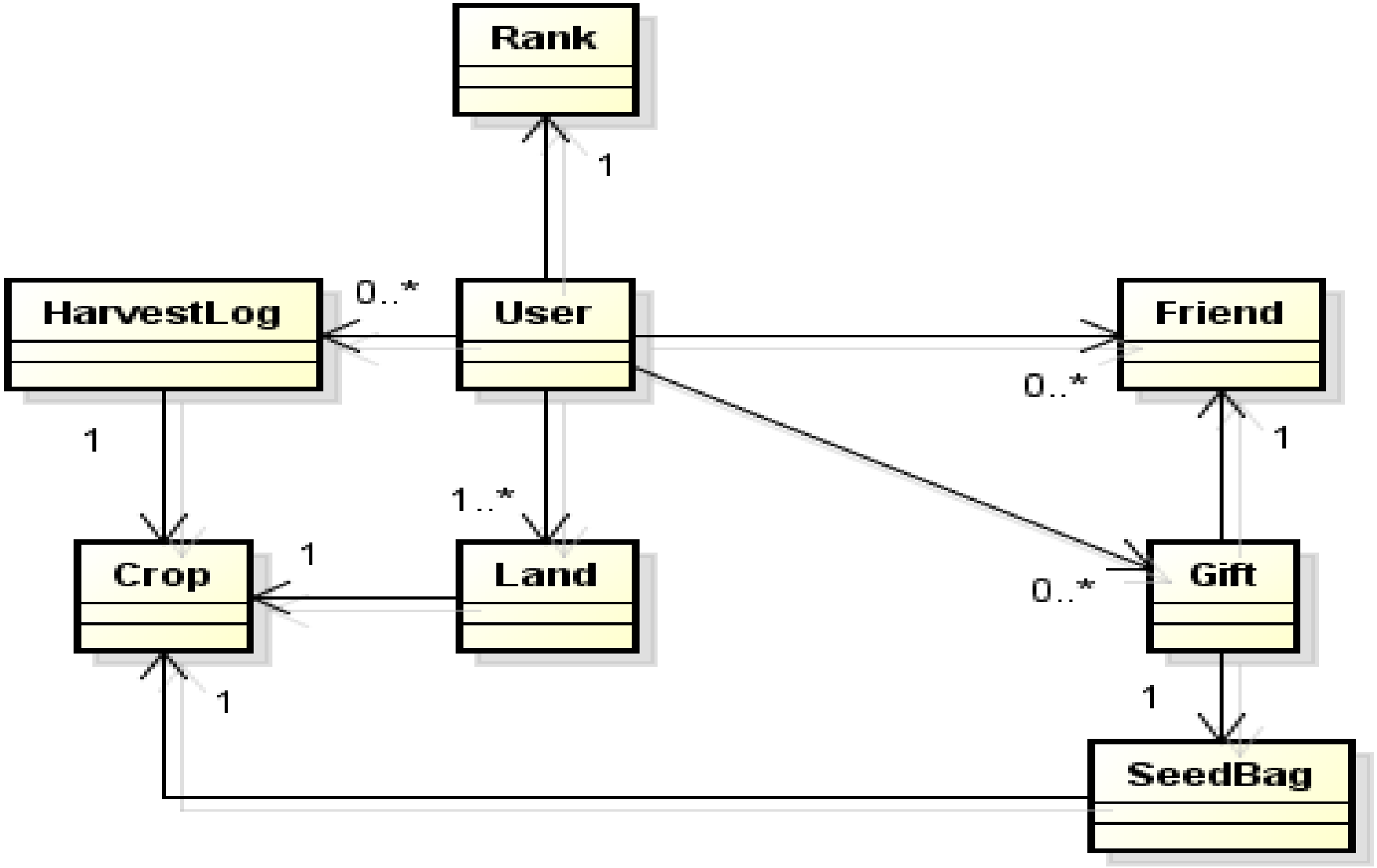

: Actor0
: GiftMenu
: GiftController
: UserManager
: GiftManager
: InventoryController
: SeedManager
1: choice()
1.1: retrieveUser(friend:String)
alt
[User != null]
1.2: givingGift(user:User,crop:Crop,friend:String) : boolean
1.2.1: retrieveUser(friend) : User
return user
1.2.2: createGift(user:User, receiver:User, crop:Crop, currDate:Date) : Gift
<<create>>
1.2.2.1: new Gift(user:User, receiver:User, crop:Crop, currDate:Date)
: Gift
1.2.2.2: save()
Gift
1.2.3: addSeed(user:User, crop:Crop, qty:int) : void
1.2.3.1: getSeedBag(owner:User, crop:Crop) : SeedBag
return null or SeedBag
alt
[SeedBag == null]
1.2.3.2: createSeedBag(user:User, crop:Crop, quantity:int) : void
<<create>>
1.2.3.2.1: new SeedBag(user:User, crop:Crop, quantity:int)
: SeedBag
[SeedBag != null]
1.2.3.3: updateSeedBag(owner:User, crop:Crop, updatedBag SeedBag) : void
1.2.3.3.1: save()
return true or false
Display outcome

6. Class diagram

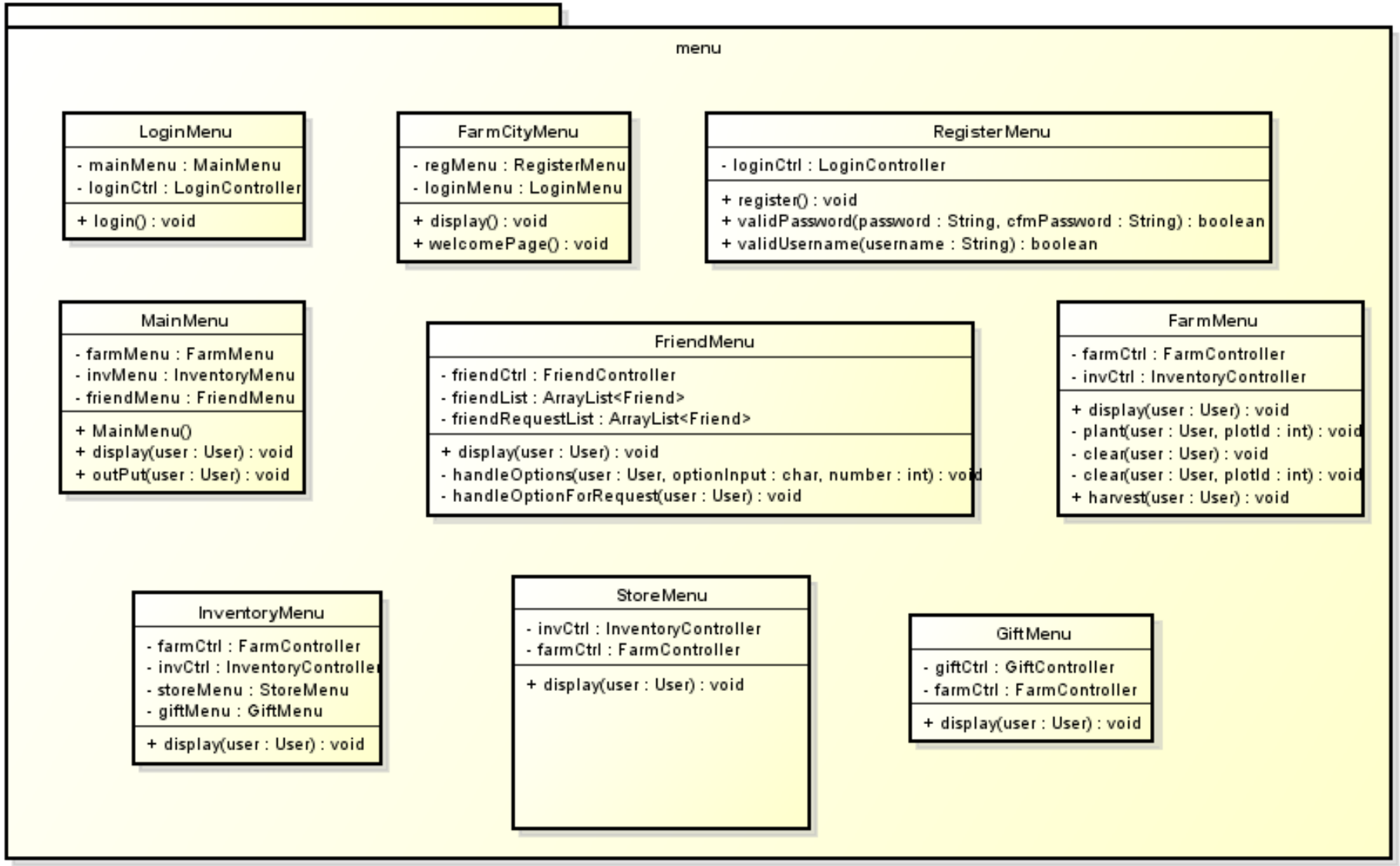
6.1 Menu, Controller, Manager Class diagram



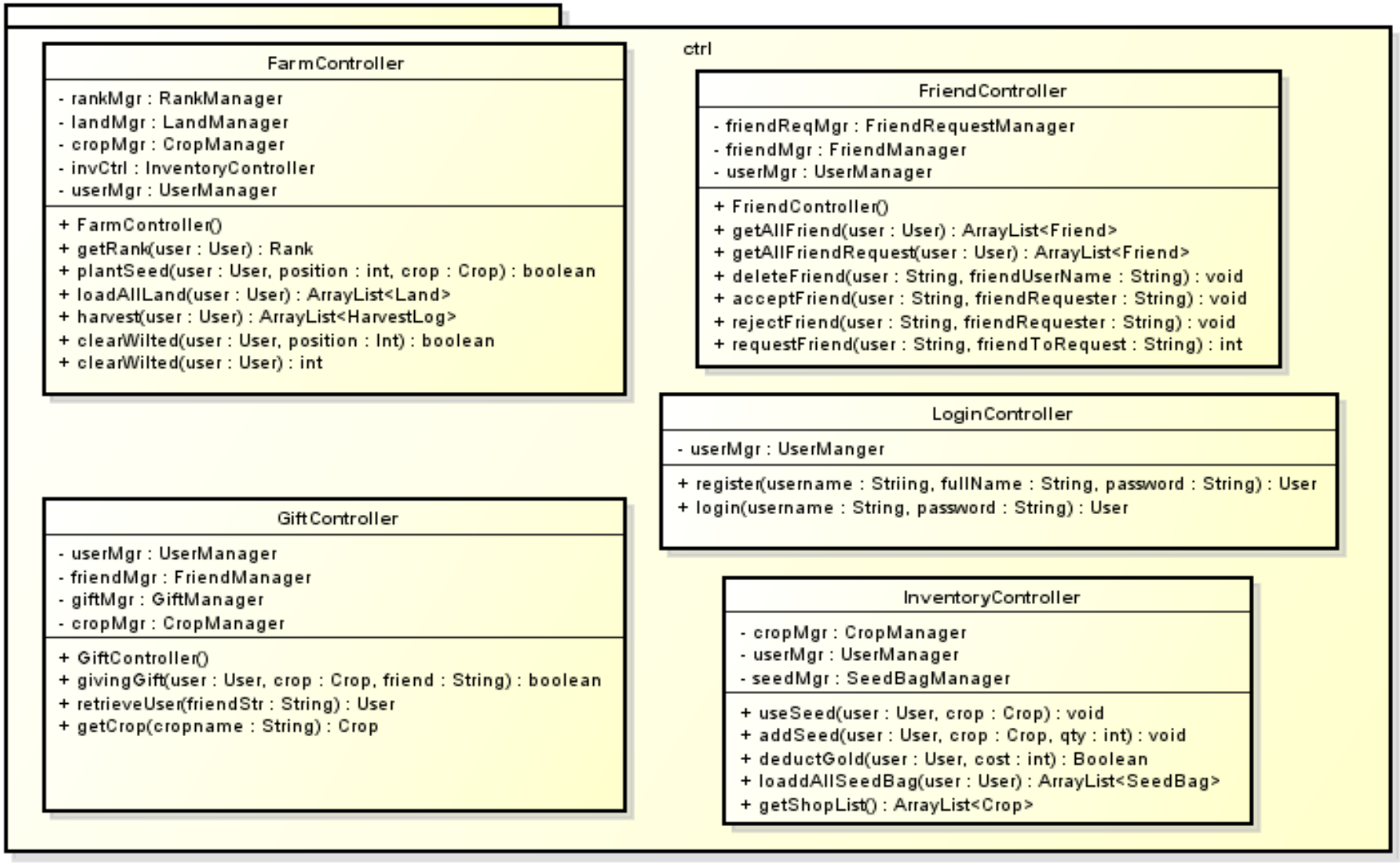
6.2 Entity class diagram



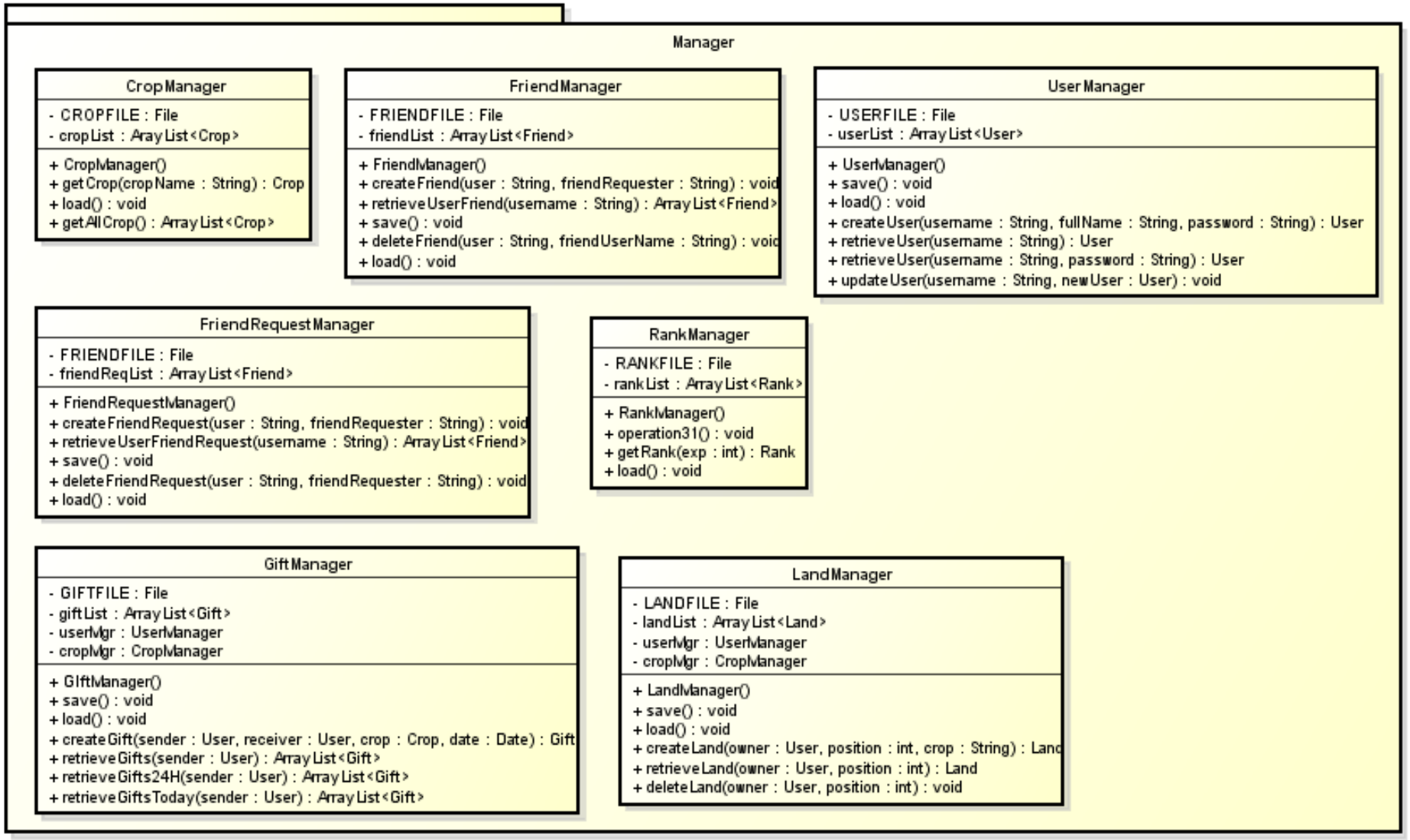
6.3 Menu classes details



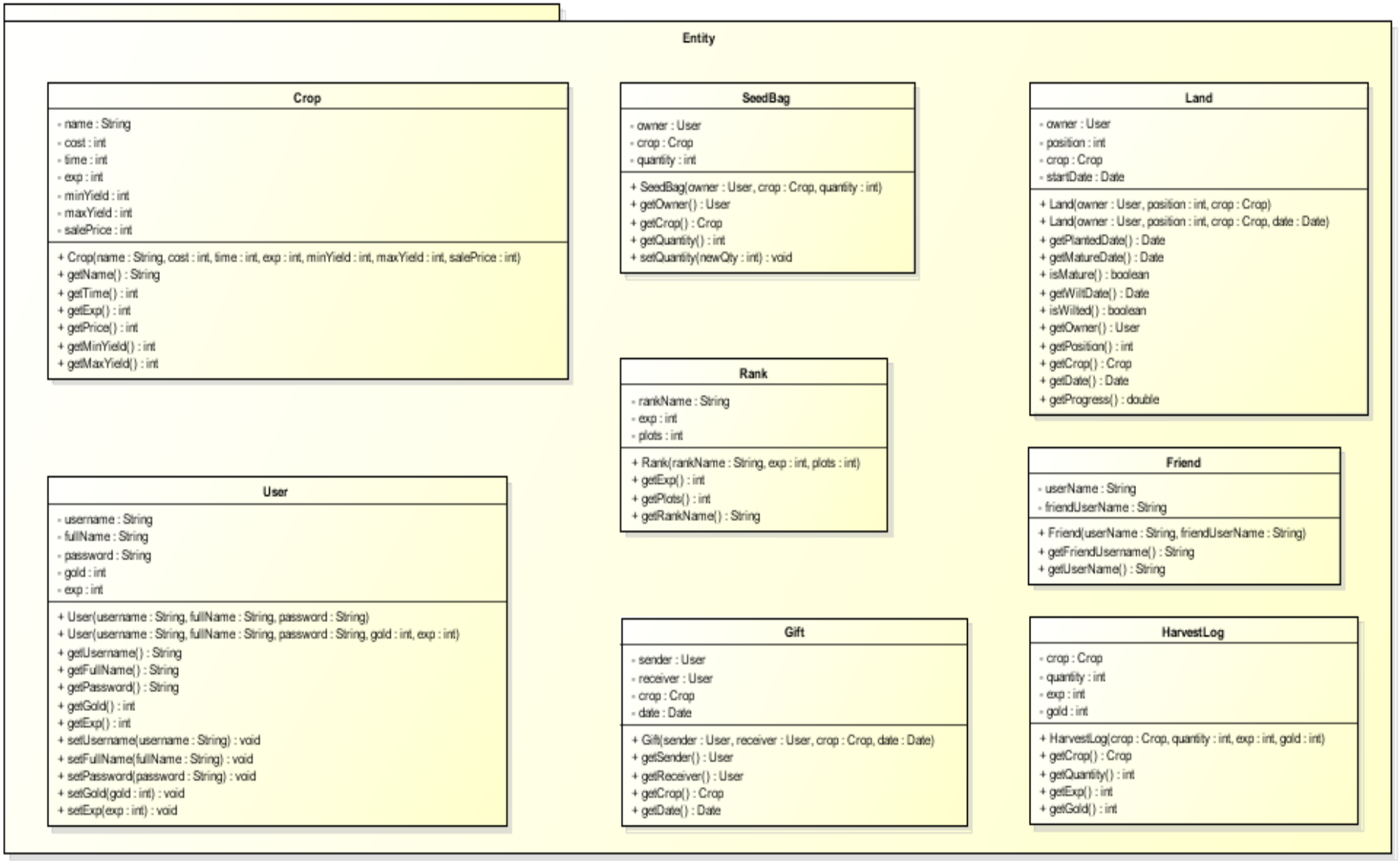
6.4 Controller classes details



6.5 Manager classes details



6.6 Entity classes details



7. Screenshots / User guide

7.1 Register

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

1. Enter 1 to register

```
== Farm City :: Welcome ==
Hi,
1. Register
2. Login
3. Exit
Enter your choice > 1
== Farm City :: Registration ==
Enter your username > kevin
Enter your full name > kevin steppe
Enter your password > 123
Confirm your password > 123
```

2. Fill in the details for registration

```
Hi, kevin steppe! Your account is successfully created!

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

3. Your account is successfully created!

7.2 Login

```
== Farm City :: Welcome ==
Hi,
1. Register
2. Login
3. Exit
Enter your choice > 2
Enter your username > kevin
Enter your password > 123
```

1. Enter valid username and password.

```
== Farm City :: Main Menu ==
Welcome, kevin steppe!

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

2. You will see main menu when you successfully logged in!

7.3 My Friends

Enter 1 from main menu to view friends menu

```
== Farm City :: Main Menu ==
Welcome, kevin steppe!

1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice > 1
== Farm City :: My friends ==
Welcome, kevin steppe!

My Friends:

My Requests

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

7.3.1 Request Friends

```
[M]ain | [U]nfriend | re[Q]uest | [A]ccept | [R]eject > Q
Enter the username > kim
A friend request is sent out to kim
```

Chose Q, and enter the username of user that you wish to request to be friends with.

7.3.2 Accept/Reject Friends

Under “My Request”, you will be able to see the friend’s requests you have received. To accept the request, type “A” followed by the number that indicates the friend that you wish to accept. For instance, if you wish to accept Kevin, you should enter “A” follow by “2” which indicates kevin. Same rule goes to “Reject”, type “R” followed by the number that indicated the friend that you wish to reject. For instance, if you wish to reject the request from kevin, enter “R2”.

7.3.3 Unfriend

Under “My Friends”, you will be able to see your list of friends. To unfriend a friend, type “U” followed by the number that indicates the friend that you wish to unfriend. For instance, if you wish to unfriend Kevin, you should enter “U2”.

Now kevin is no longer your friend!

7.4 My Inventory

Enter 3 from main menu to view Inventory Menu

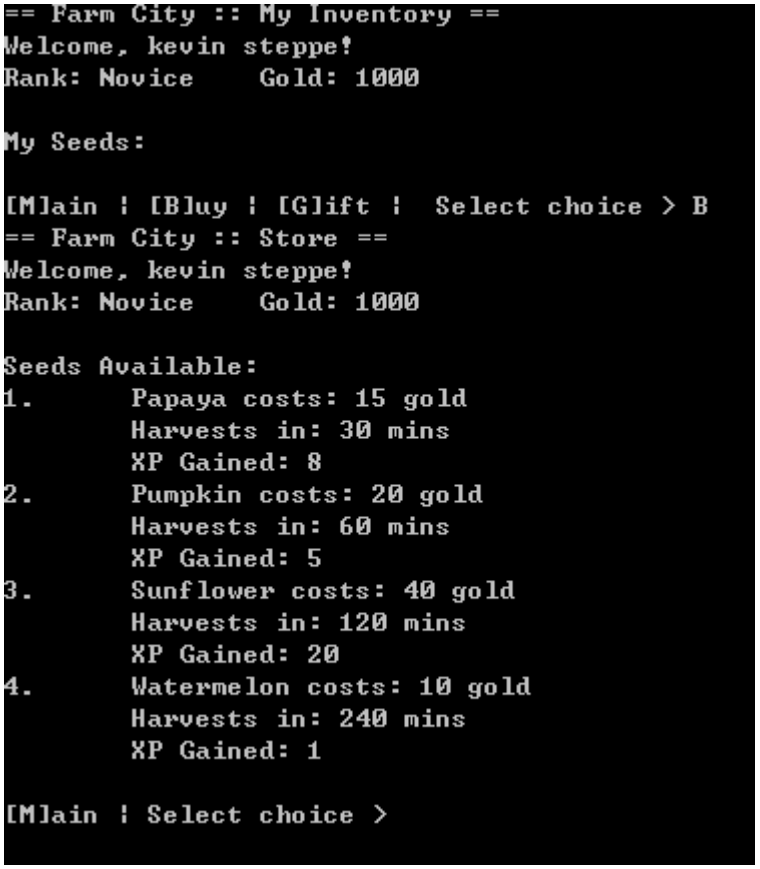
```
== Farm City :: Main Menu ==
Welcome, zz!

1. My Friends
2. My Farm
3. My Inventory
4. Logout
Enter your choice > 3
== Farm City :: My Inventory ==
Welcome, zz!
Rank: Novice    Gold: 1000

My Seeds:

You have no seeds


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

If you don't have any seeds, you will see message "you have no seeds"

Otherwise, you will see the list of seeds you have.

```
My Seeds:
1. 49 Bags of Papaya
2. 6 Bags of Pumpkin
3. 2 Bags of Sunflower
4. 4 Bags of Watermelon
```

7.4.1 Buy seeds



To purchase a new seed, enter B.From the list of seeds available, choose one to purchase a bag of seeds. The price of the seed will be deducted from your gold.

```
[M]ain ¦ Select choice > 2
Enter quantity > 2
```

Select type of seed and quantity.

```
== Farm City :: My Inventory ==
Welcome, kevin steppe!
Rank: Novice    Gold: 960

My Seeds:
1. 2 Bags of Pumpkin

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

2 bags of pumpkin has been successfully bought! Notice that your gold has been deducted to 960 (1000-(20*2))

7.4.2 Gift Seeds

```
My Seeds:
1. 49 Bags of Papaya
2. 6 Bags of Pumpkin
3. 2 Bags of Sunflower
4. 4 Bags of Watermelon

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

Select g to send gift.

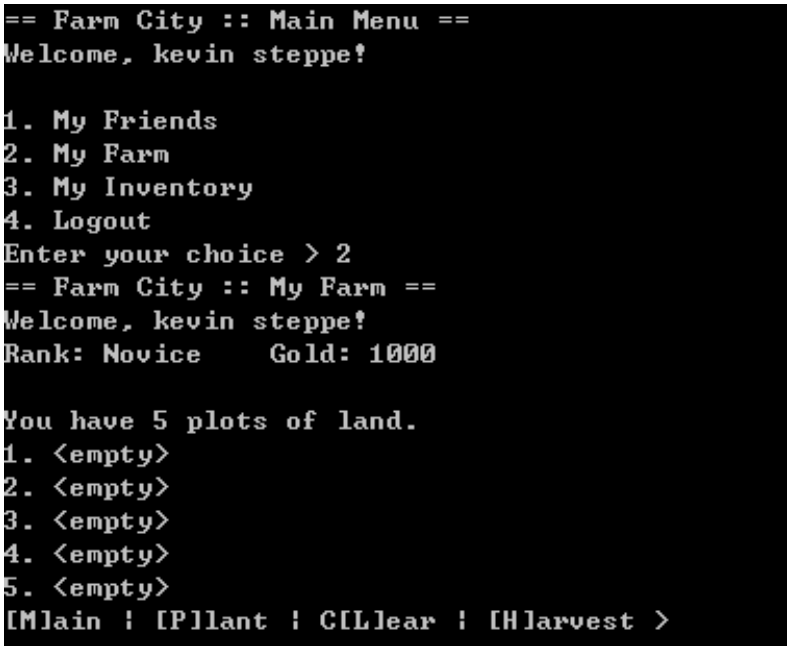
```
Gifts Available:
1. 1 Bag of Papaya Seeds
2. 1 Bag of Pumpkin Seeds
3. 1 Bag of Sunflower Seeds
4. 1 Bag of Watermelon Seeds
[M]ain | Select choice > 2
Send to> jerots
Gift sent to jerots

== Farm City :: Send a Gift ==
Welcome, kim!
Rank: Novice    Gold: 1000
[M]ain | Select choice > 2
Send to> jerots,kevin
```

Enter the seed that you wish to send to your friends and user id of the friend. You can send to multiple number (up to 5) at once.

```
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 > 2
Send to> jerots,kevin
Gifting to jerots failed!
```

You can only send one gift to up to 5 friends a day (within 24 hours). If not, error message will be sent like the above.

7.5 My Farm



Select 2 on Main menu to see farm of yours. Now, you haven't harvested any yet.

7.5.1 Plant

```
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 > H2
```

To plant, type “P” followed by the number that indicates the plot of land. For instance, if you wish to plant in plot2, type “P2”.

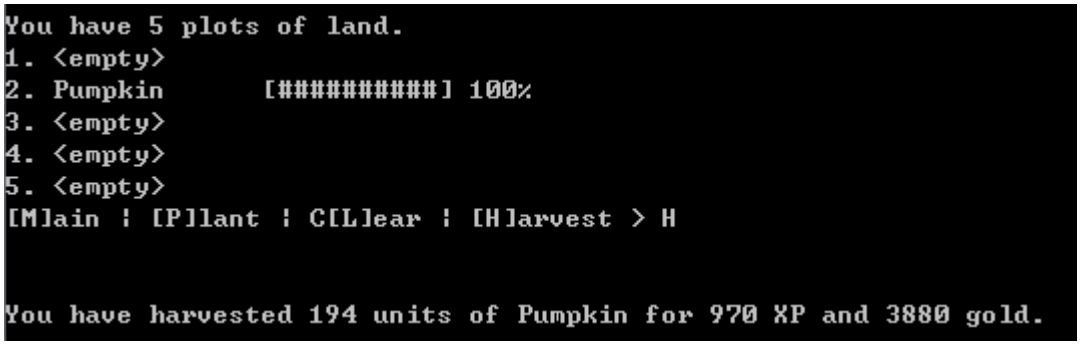
```
[M]ain | [P]lant | C[L]ear | [H]arvest > P2
Select the crop:
1. Pumpkin
[M]ain | Select Choice > 1
```

Select the type of seed you wish to plant.

```
You have 5 plots of land.
1. <empty>
2. Pumpkin      [----------] 0%
3. <empty>
4. <empty>
5. <empty>
[M]ain | [P]lant | C[L]ear | [H]arvest >
```

Pumpkin has been planted in plot2.

7.5.2 Harvest



When crop's maturity reaches 100%, you can finally harvest

7.5.3 Clear

```
== Farm City :: My Farm ==
Welcome, kevin steppe!
Rank: Novice    Gold: 4690

You have 5 plots of land.
1. Pumpkin      [  wilted  ] 100%
2. <empty>
3. <empty>
4. <empty>
5. <empty>
[M]ain | [P]lant | C[L]ear | [H]arvest > L1

Plot 1 is cleared. 5 gold deducted.

== Farm City :: My Farm ==
Welcome, kevin steppe!
Rank: Novice    Gold: 4685

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 >
```

To clear, type “L” followed by the number that indicates the plot of land. For instance, if you wish to clearplot1, type “L1”.

8. Object Oriented Design Considerations

8.1 Single Responsibility Principle (SRP)

In the course of building the application, we applied the Single Responsibility Principle (SRP). SRP is a principle of coding whereby we code in a way that each class has only one responsibility. By specializing one class with one responsibility, we reduce the dependencies on each classes to minimize the cascading effects of small changes in coding. In this way, it will be easier to understand coding and make improvement in each classes when necessary. For instance, Farm controller is only responsible for managing farm object and do not interfere other irrelevant classes.

8.2 Design approaches

We adopted an iterative approach in the course of completing the project. We have been through a number of iterations of checking requirements, designing, coding, integrating and finally testing and getting feedbacks from other group mates. To ensure that the entire project is integrated and not affected by overlapped parts, we completed functionalities step by step. This allowed us to plan our work better and also to minimize the mistakes, as it is easier for us to make changes when it is done small part at a time and to spot the overlapping functionality.

.

8.3 Package Structure

In the course of coding, we made sure that each class is packaged into different sets of classes, which helps categorize the responsibilities of the class. The categories include boundary, controller, manager (DAO) and entity classes. Boundary classes are used for direct interaction with the user. Controller classes are used to perform business logic to achieve required functions. Manager (DAO) classes are responsible for managing the data objects (Create, Retrieve, Update, Delete) found in the entity package. Packaging the application made it easier for user/programmers to identify the responsibilities of each class.

9. Challenges and lesson learnt

We learnt that effective cooperation within a team to accomplish the job is imperative. Effective teamwork plays a vital role in solving problems efficiently and creatively. Throughout the course of the project, we have met a number of challenges, such as solving the complicated gifting functions in the project, coming out with the most effective way to present the code, and as well as managing time and resources. Our team worked as a team, rather than dividing the parts to complete individually to ensure integrity of our design and code, and also to make sure that each team members understood the course completely. When any of the teammate faced problems, the entire teammates gathered together to solve the problem together, despite busy schedule. This way, the team was able to complete the project efficiently on time with great team-bonding.

We also learnt that the planning and design of the system before start coding is important to ensure the integrity of our code and help prevent possible problems. Through the System Sequence Diagram (SSD) and Sequence Diagram (SD), we were able to agree on design of our program beforehand. This ensures that the processes go on as planned. This design phase will also be useful when we delegate the coding task to other people in the future.

Finally, the project enabled us to understand object-oriented design concepts with regard to building an application better as we build the application first-hand, rather than learning concepts by relying on theories and class exercises. Through the process, we learnt when or when not to use specific functions. For example, initially, our group's code used many static declarations. However we learnt that this may not be the best way, considering the risk that we may face if this program was used by more people (lack of multithreading support). It is these considerations that we have to include, which taught us that programming requires active and critical thinking, based on the program's context and requirements.