2014

SINGAPORE MANAGEMENT UNIVERSITY

[DUKE HUNT APPLICATION]

G6-Team 11

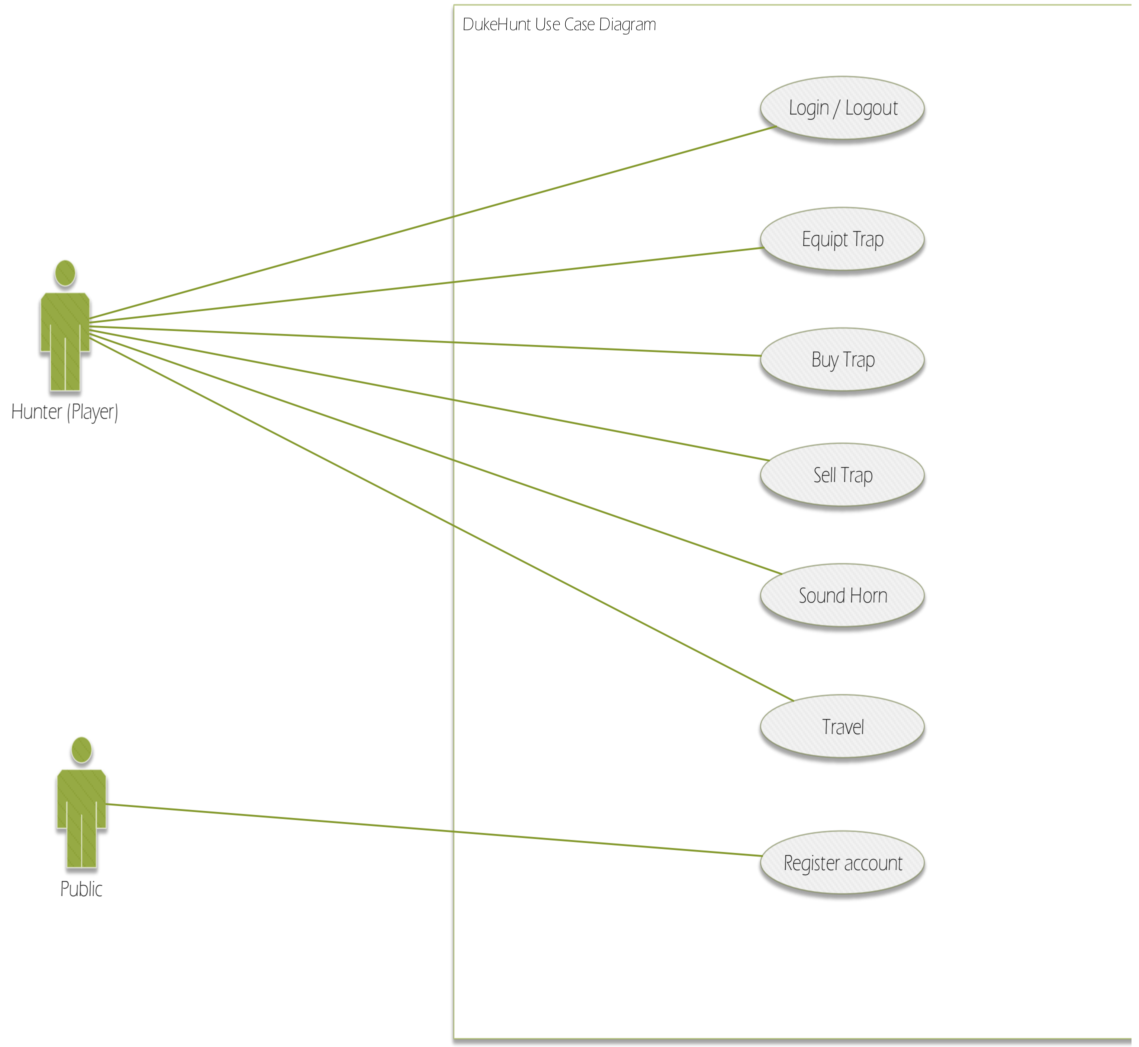
Charlie Chong Yick Soon

Toh Xin Ning

Zhao Yazhi

Contents

1. Use Case Diagram



1.1 Login / Logout
This allows the user to login and logout of the Duke Hunt application.

1.2 Equip Trap
This allows the hunter to view his current owned traps and also to equip or change his current trap.

1.3 Buy Trap
This allows the hunter to buy traps in the Trap Smith shop.

1.4 Sell Trap
This allows the hunter to sell his traps in the Trap Smith shop.

1.5 Sound Horn
This allows the hunter to view his hunting history and to sound horn.

1.6 Travel
This allows the hunter to travel to another region that he can go to according to his current rank.

1.7 Register Account
This allows the public to register an account with the Duke Hunt application.

2. Use Case Scenarios

2.1 Login/ Logout

Use Case Scenario	
Reference:	Login
Actor:	Hunter
Preconditions:	
Main Flow of Events:	1. The Hunter enters his/her username and password. 2. The system checks if the username has been registered previously and the password entered matches. 3. The hunter will then be brought to the main menu page where the system will load the following information of the Hunter's: 3.1. Name/username 3.2. Rank title 3.3. Last Visited Location 3.4. Current XP 3.5. Current gold 4. The end
Alternative Flow of Events:	2a. if the username entered cannot be found, the system displays an error message to the user; prompting him to register first. i. The hunter is then brought back to the welcome page. 2b. if the username exists but the hunter enters the wrong password, ii. The hunter is then brought back to the welcome page.

Use Case Scenario	
Reference:	Logout
Actor:	Hunter
Preconditions:	1. Hunter is logged in
Main Flow of Events:	1. The system will display a successful message and the hunter is logged out successfully. 2. The end.
Alternative Flow of Events:	

2.2 Equip Trap

Use Case Scenario	
Reference:	Equip Trap
Actor:	Hunter
Preconditions:	1. The hunter is logged in.
Main Flow of Events:	1. The system will get all the available traps that the Hunter owns and display to the hunter. 2. The hunter chooses the trap that he/she wishes to be equipped with. 3. Once the trap is equipped, the system updates the: i. Current equipped weapon ii. Updates the Hunter's Inventory (so that the current weapon is not displayed in the list) and display a successful message. 4. The end
Alternative Flow of Events:	3a. If the Hunter chooses to change his Trap again, Step 1 to 3 are repeated.

2.3 Buy Trap

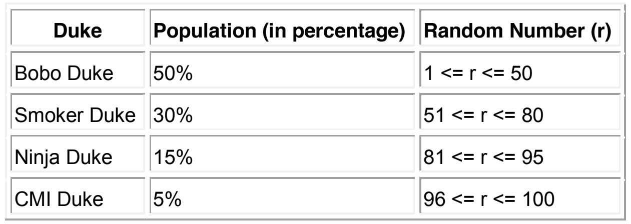
Use Case Scenario	
Reference:	Buy Trap
Actor:	Hunter
Preconditions:	1. Hunter is logged in 2. The trap is not found in the Hunter's inventory
Main Flow of Events:	1. The hunter enters the trap smith shop in the region that he/she is in. 2. The system loads all the traps in the region that the hunter is in. 3. The hunter chooses a trap that he would like to purchase in that region. 4. The system checks if the hunter has enough experience points and enough gold to purchase the trap. 5. For a successful purchase, the system will display a message to notify the hunter and updates the hunter's inventory list. 6. The End
Alternative Flow of Events:	2a. If the hunter does not have sufficient gold or do not have enough experience points, the purchase is unsuccessful. i. The hunter is notified of the unsuccessful purchase. ii. The End

2.4 Sell Trap

Use Case Scenario	
Reference:	Sell Trap
Actor:	Hunter
Preconditions:	1. Hunter is logged in 2. The trap exists in the Hunter's inventory
Main Flow of Events:	1. The hunter enters the trap smith shop in the region that he/she is in. 2. The system loads all the traps in the region that the hunter is in. 3. The hunter wishes to sell his inventory in that region. 4. The system checks if the hunter has more than one trap in his inventory after the sales. 5. The hunter will gain back 25% of gold of the sold trap. 6. The system removes the sold trap in the hunter's inventory and displays a successful message. 7. The end.
Alternative Flow of Events:	2a. If the hunter has less than one trap after sales, the sales transaction is unsuccessful. i. The system will display an unsuccessful message to the hunter. 4a. If the sold trap is currently equip by the hunter, i. The system will update the hunter's inventory and display the current equip trap as "–Nil-"

2.5 Sound Horn

Use Case Scenario	
Reference:	Sound Horn
Actor:	Hunter
Preconditions:	1. The Hunter is logged in.
Main Flow of Events:	1. The system will read and retrieve the first 10 recent records from hunting record.csv and display them to the hunter. 2. The system will also retrieve all the dukes from the duke.csv in the region that the hunter is at. 3. The system will populate the population of dukes in the region from the population given in the duke.csv file:



Duke	Population (in percentage)	Random Number (r)
Bobo Duke	50%	1 <= r <= 50
Smoker Duke	30%	51 <= r <= 80
Ninja Duke	15%	81 <= r <= 95
CMI Duke	5%	96 <= r <= 100

	4. The system will check if the Hunter has an equip trap. 5. If yes, the system will attempt to catch the duke. 6. The system checks if the hunter’s current trap has enough power to catch the chose Duke that is chose randomly.

7. The percentage difference in the below table determines if the duke will be caught or not:

Row	Percentage Difference (%)	Escape Range
1	D <= 0	0 – 94
2	0 < D <= 25	0 – 14
3	25 < D <= 75	0 – 8
4	75 < D <= 100	0 – 6
5	100 < D	0 – 4

8. If the duke is caught, the system will use the caught duke's power to determine the gold and XP points the hunter will gain.
   Experience Points = ROUNDUP (Power of Duke * r1 / 1000)
   Gold = ROUNDUP (Power of Duke * r2 / 1000)

9. The system then updates the Hunter's gold and XP points.
10. The system will display to the hunter the result of the sound horn.
11. The information is then written into .csv
12. The end.

Alternative Flow of Events:

1a. if the hunter does not have any equip traps
   i. The system will notify the hunter to equip trap before sounding the horn again.

5a. If the duke is not caught, the hunter's gold and XP is not updated.
   i. Go to main flow step 10.

2.6 Travel

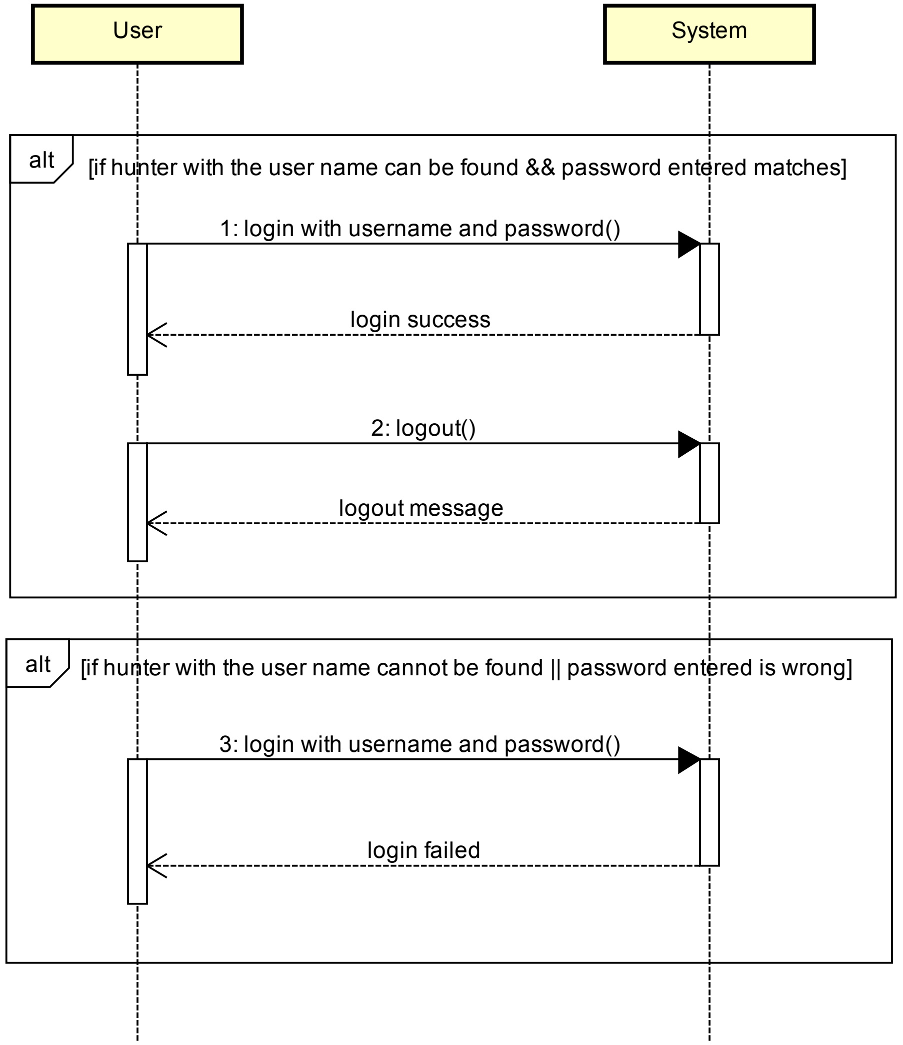
Use Case Scenario	
Reference:	Travel
Actor:	Hunter
Preconditions:	1. Hunter is logged in.
Main Flow of Events:	1. The system displays the regions that the hunter can travel to and the travelling cost to the selected regions. 2. If the hunter has enough money, the system will deduct the money from the hunter. 3. The system will also update the hunter's current region and the information is written into the .csv file. 4. The system then notifies the hunter of his current region. 5. The end.
Alternative Flow of Events:	1a. If the hunter's current experience point is between 0 and 1799, the hunter cannot travel to any other regions. i. The system will display a message notifying the hunter he cannot travel now. 3a. if the hunter does not have enough money, the hunter is unable to travel. i. The system displays an error message telling the hunter he does not have enough money.

2.7 Register Account

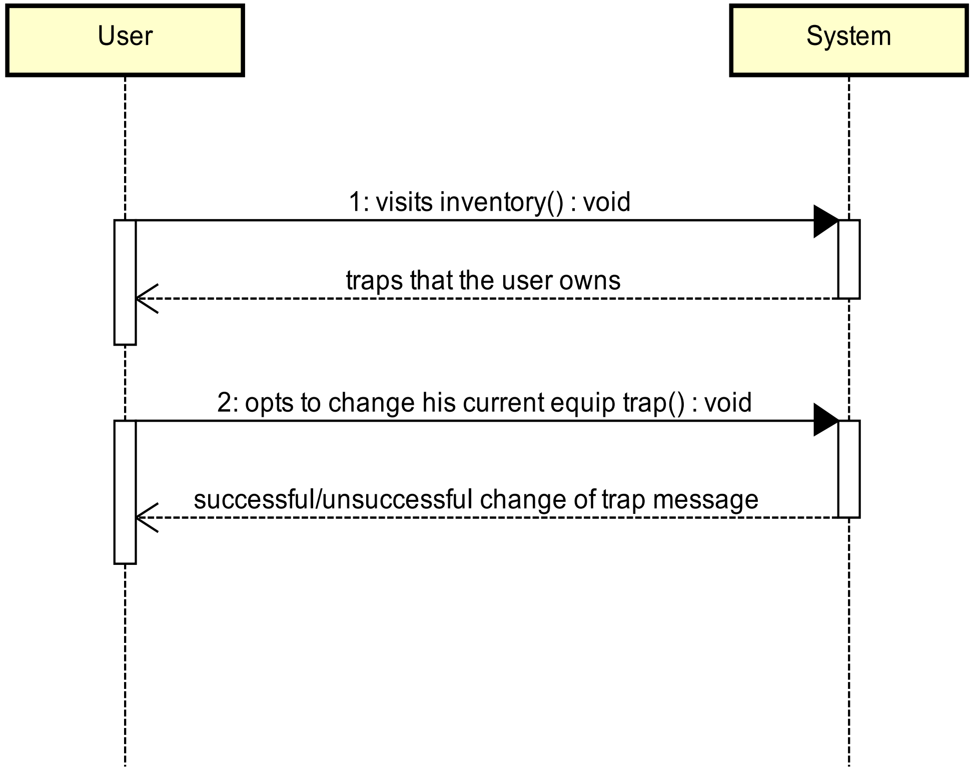
Use Case Scenario	
Reference:	Register account
Actor:	Public
Precondition(s):	
Main Flow of Events:	1. The public user enters his preference username and password. 2. The system will check if there is a previous user with the same username in the .csv file. 3. The system then checks if the entered passwords match each other. 4. Upon successful registration, the system displays a success message and the system will store the new username and password into the .csv file. 5. The public user will then be brought back to the Welcome Page where he/she can choose to log in or exit the application 6. The End
Alternative Flow of Events:	2a. If the username entered already belongs to another user; the system displays an error message stating that the username is taken. i. Public is prompted to re-enter their registration details again. 3a. If the passwords entered do not match, the system displays an error message stating that password entered does not match. i. Public is prompted to re-enter their registration details again.

3. System Sequence Diagrams

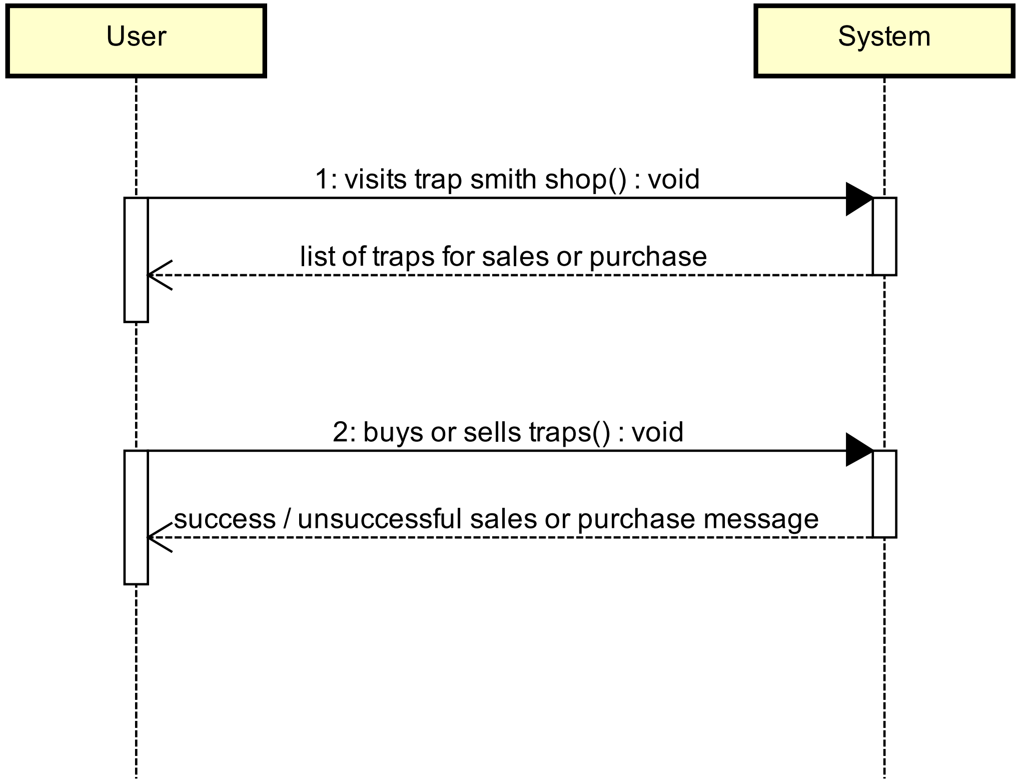
3.1 Login/ Logout


User
System
alt
[if hunter with the user name can be found && password entered matches]
1: login with username and password()
login success
2: logout()
logout message
alt
[if hunter with the user name cannot be found || password entered is wrong]
3: login with username and password()
login failed

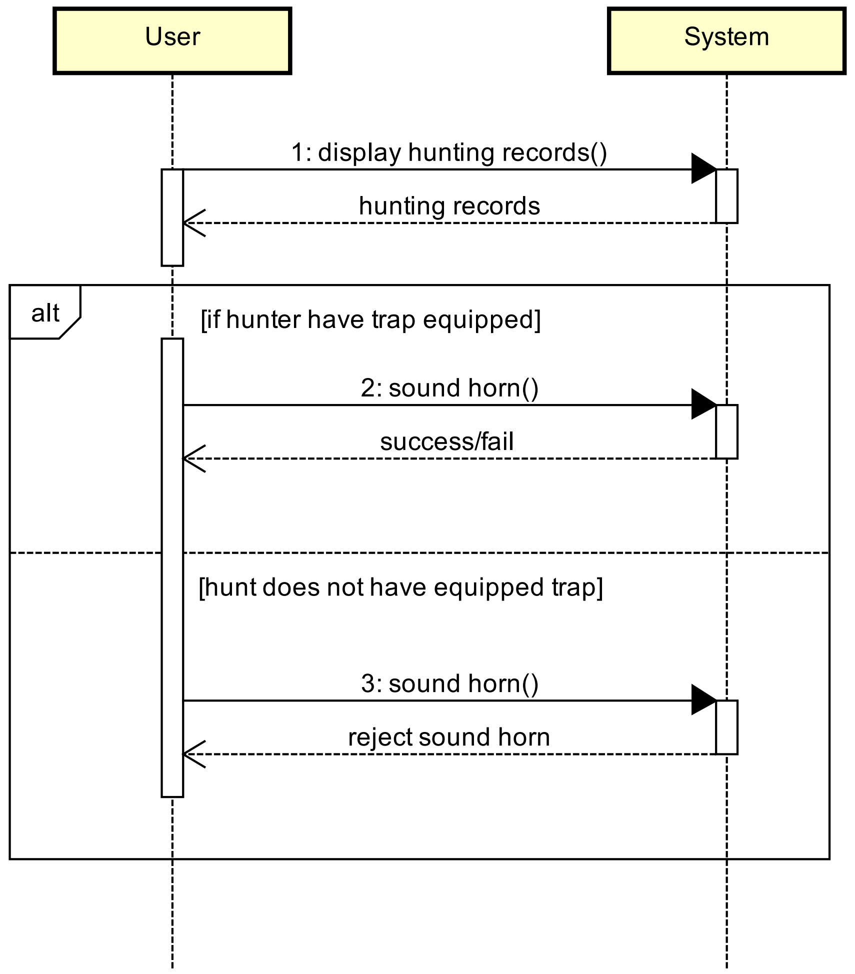
3.2 Equip Trap


User
System
1: visits inventory() : void
traps that the user owns
2: opts to change his current equip trap() : void
successful/unsuccessful change of trap message

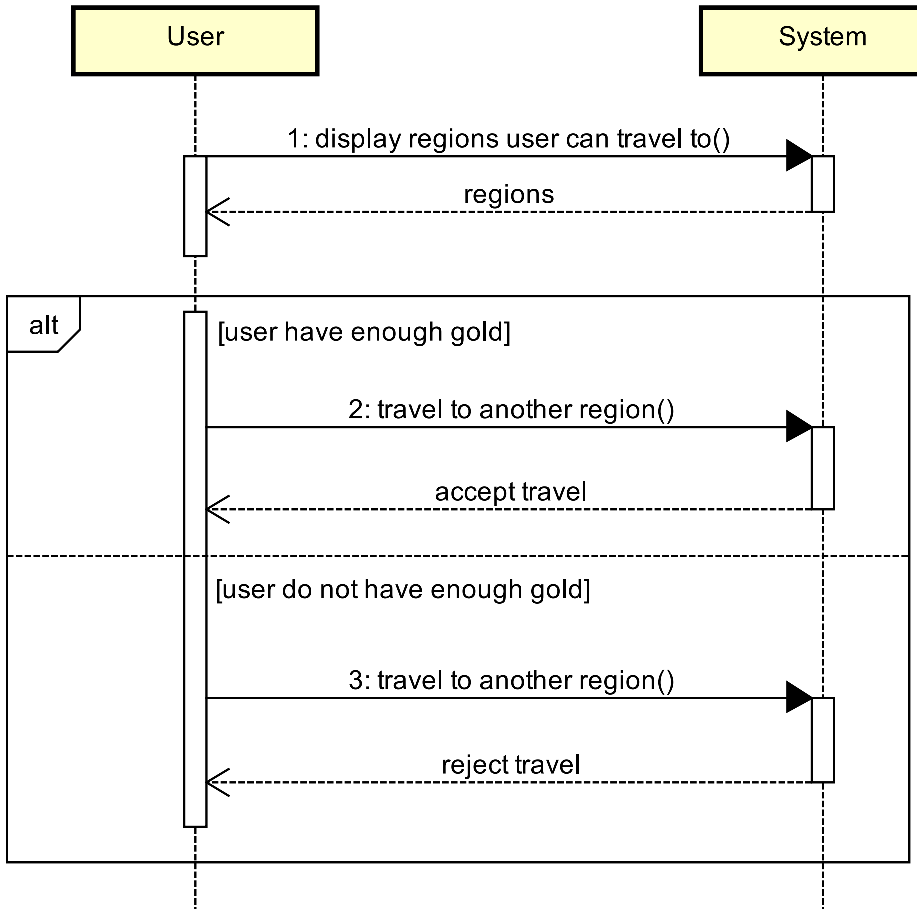
3.3 Buy / Sell Trap


User
System
1: visits trap smith shop() : void
list of traps for sales or purchase
2: buys or sells traps() : void
success / unsuccessful sales or purchase message

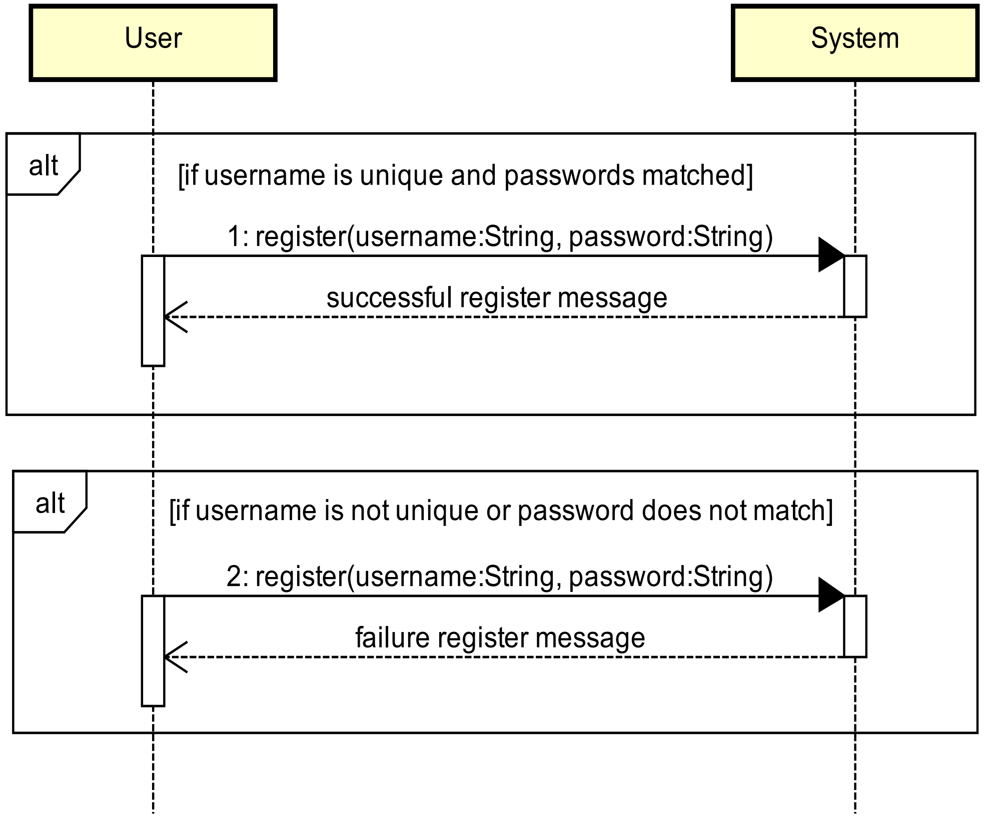
3.4 Sound Horn


User
System
1: display hunting records()
hunting records
alt
[if hunter have trap equipped]
2: sound horn()
success/fail
[hunt does not have equipped trap]
3: sound horn()
reject sound horn

3.5 Travel


User
System
1: display regions user can travel to()
regions
alt
[user have enough gold]
2: travel to another region()
accept travel
[user do not have enough gold]
3: travel to another region()
reject travel

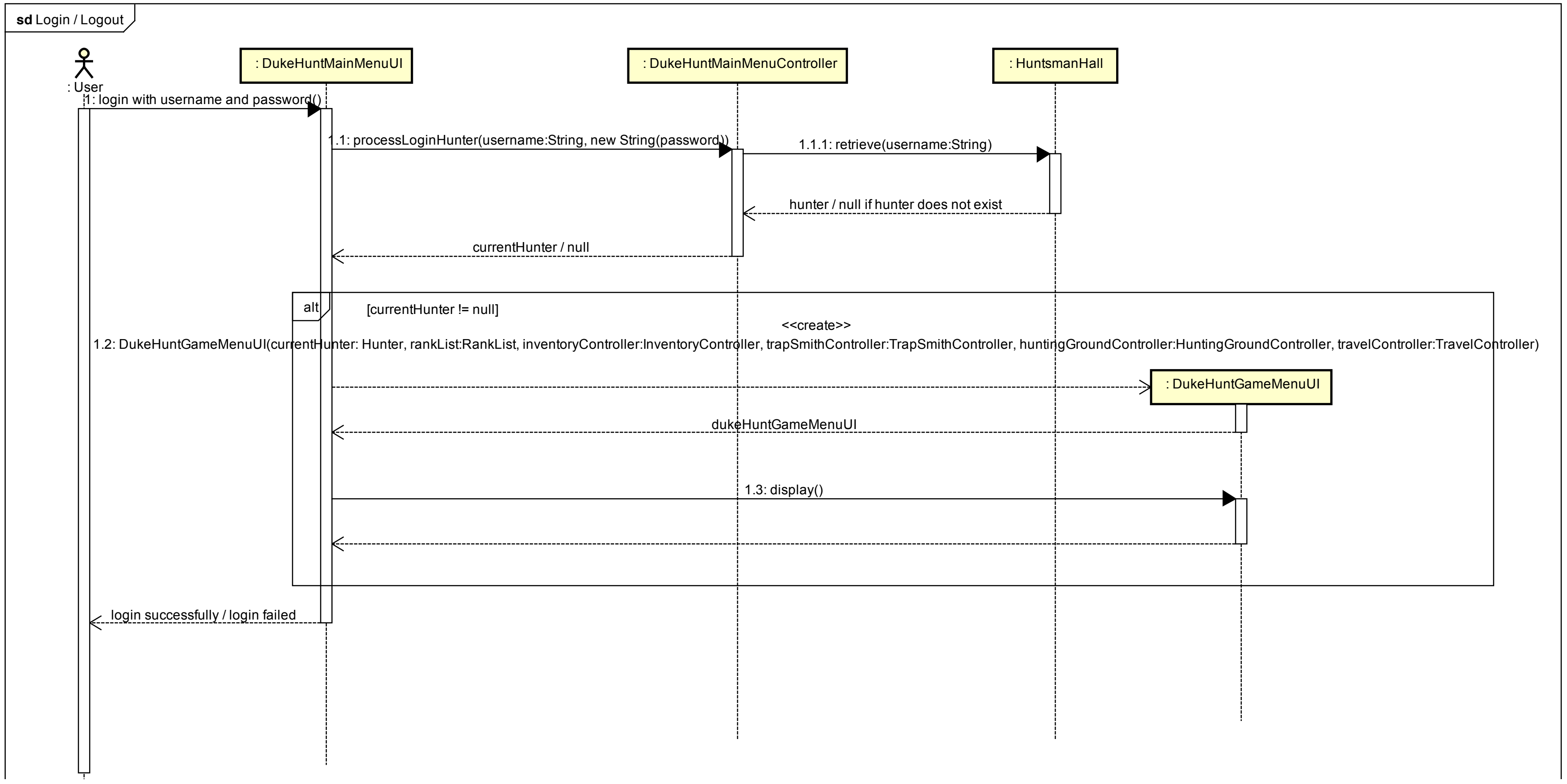
3.6 Register Account


User
System
alt
[if username is unique and passwords matched]
1: register(username:String, password:String)
successful register message
alt
[if username is not unique or password does not match]
2: register(username:String, password:String)
failure register message

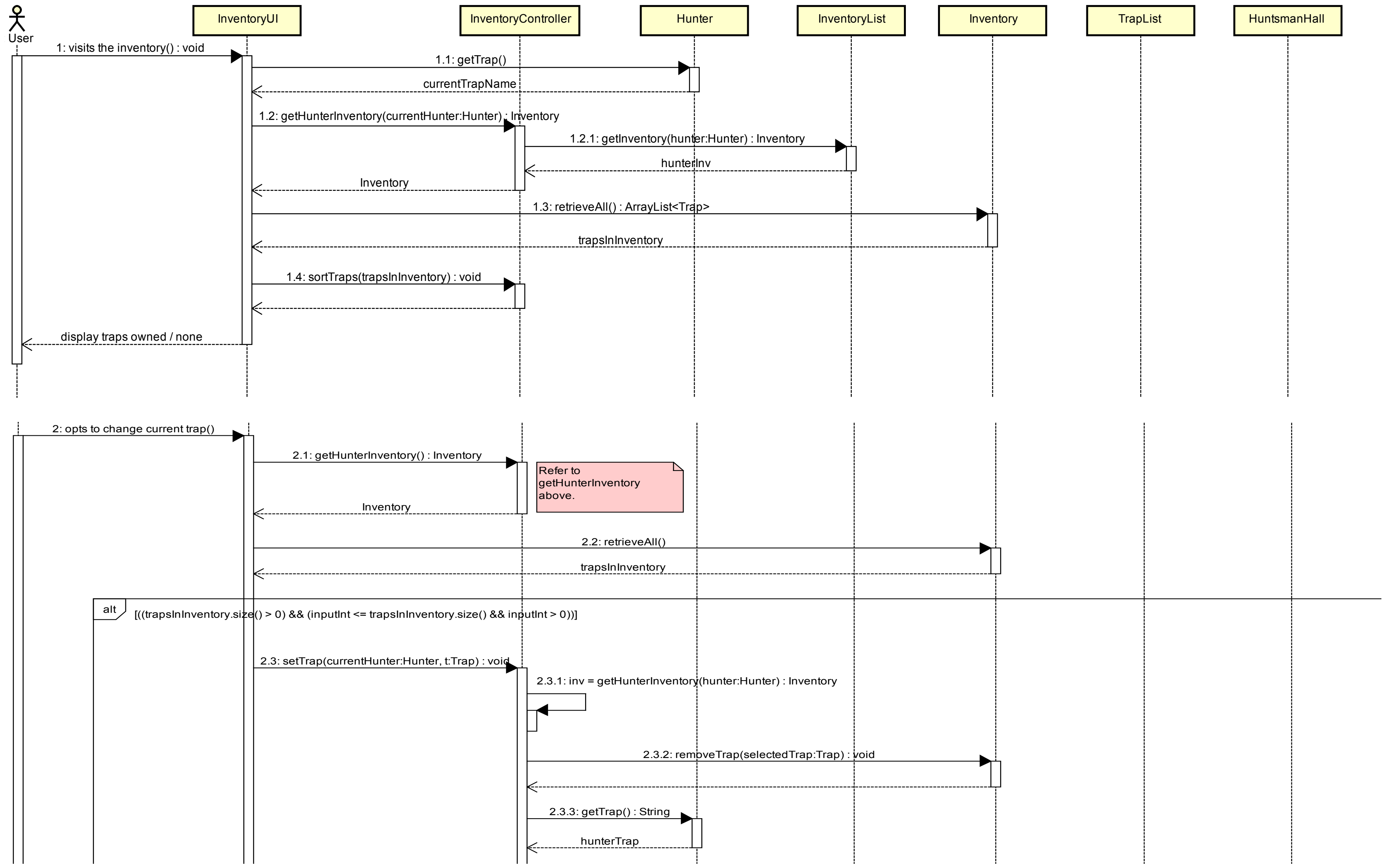
4. Sequence Diagrams*

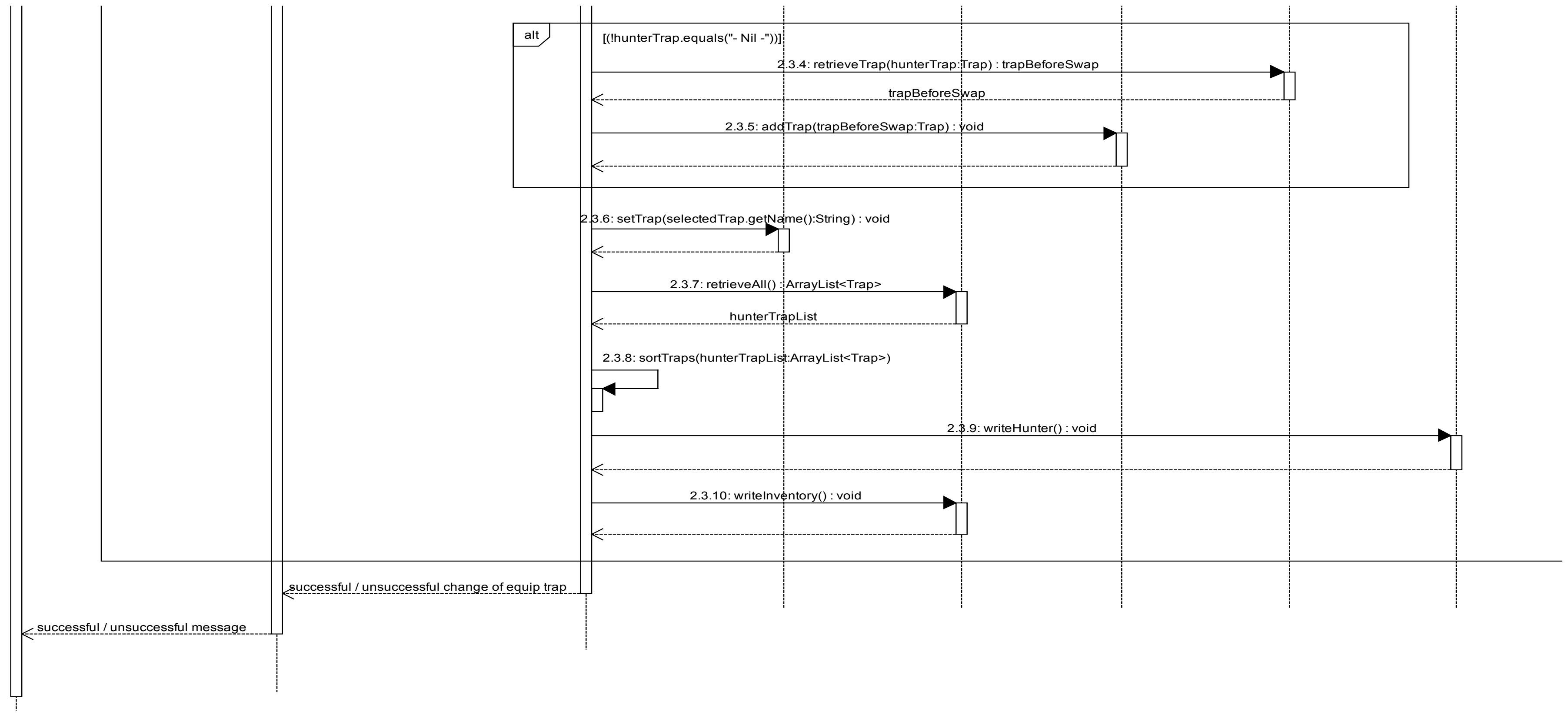
**Note that all return and lifelines' lines are dotted. They are visible upon zooming in.*

4.1 Login/ Logout

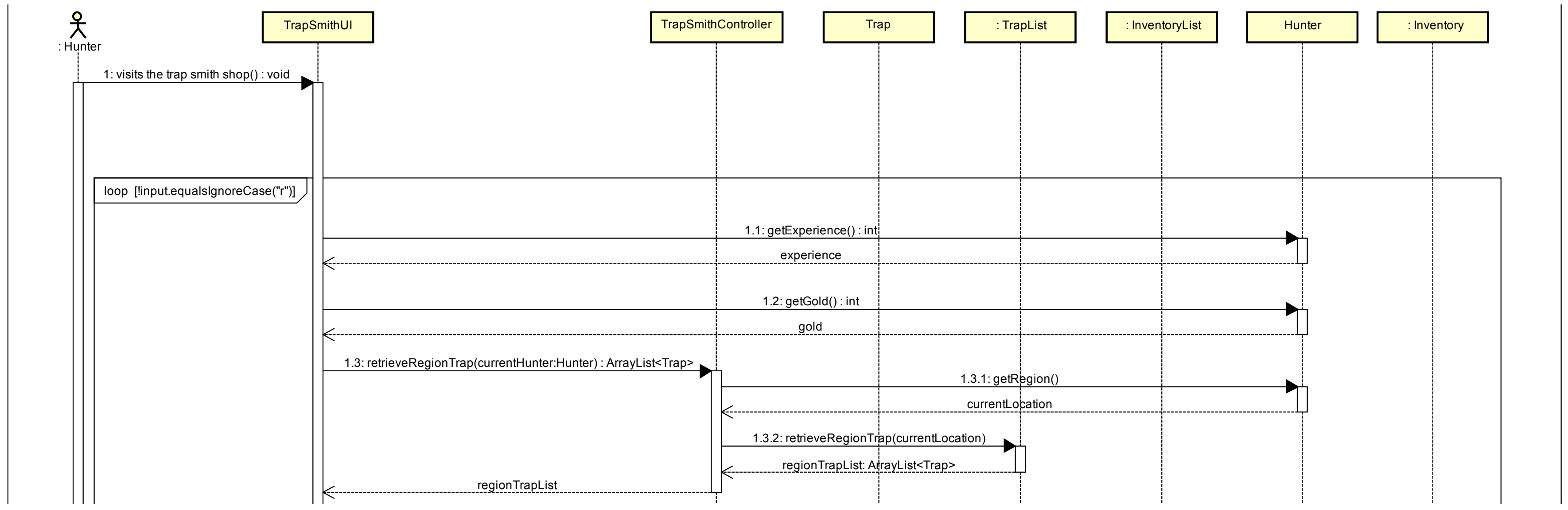

sd Login / Logout
: User
: DukeHuntMainMenuUI
: DukeHuntMainMenuController
: HuntsmanHall
1: login with username and password()
1.1: processLoginHunter(username:String, new String(password))
1.1.1: retrieve(username:String)
hunter / null if hunter does not exist
currentHunter / null
alt
[currentHunter != null]
<<create>>
1.2: DukeHuntGameMenuUI(currentHunter: Hunter, rankList:RankList, inventoryController:InventoryController, trapSmithController:TrapSmithController, huntingGroundController:HuntingGroundController, travelController:TravelController)
: DukeHuntGameMenuUI
dukeHuntGameMenuUI
1.3: display()
login successfully / login failed

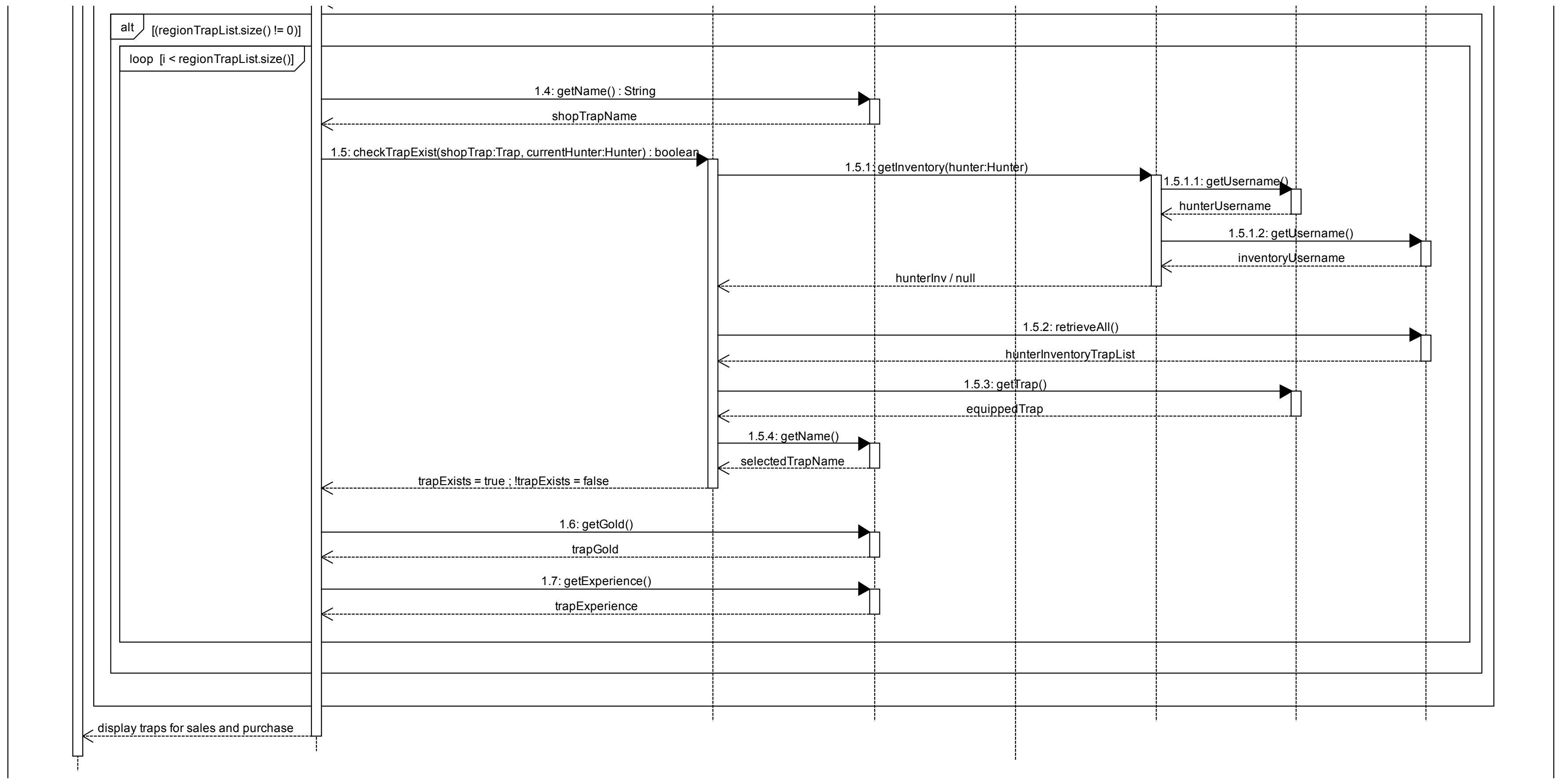
4.2 Equip Trap

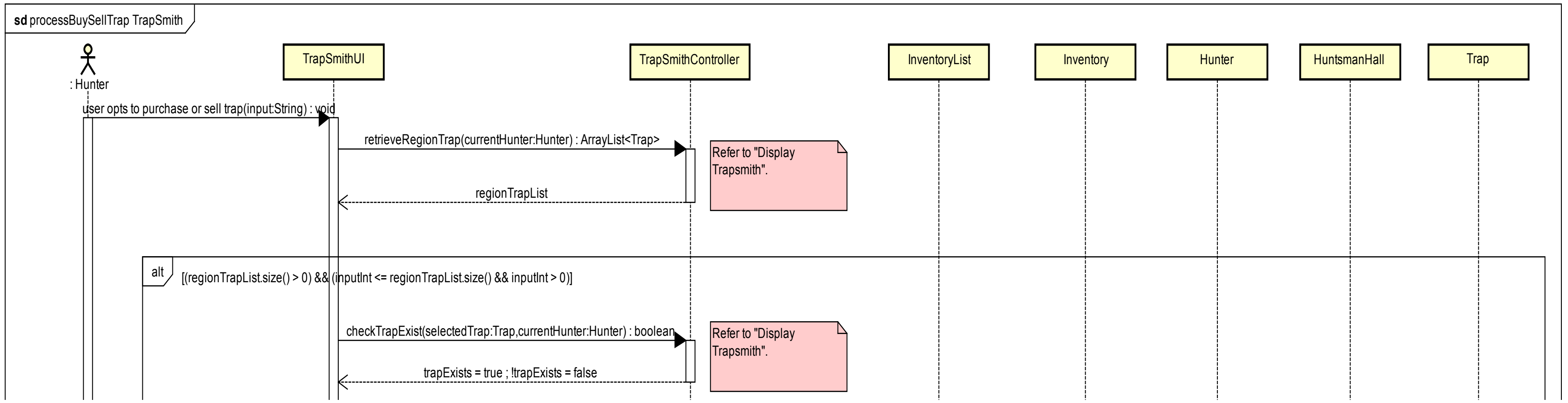

User
InventoryUI
InventoryController
Hunter
InventoryList
Inventory
TrapList
HuntsmanHall
1: visits the inventory() : void
1.1: getTrap()
currentTrapName
1.2: getHunterInventory(currentHunter:Hunter) : Inventory
1.2.1: getInventory(hunter:Hunter) : Inventory
hunterInv
Inventory
1.3: retrieveAll() : ArrayList<Trap>
trapsInInventory
1.4: sortTraps(trapsInInventory) : void
display traps owned / none
2: opts to change current trap()
2.1: getHunterInventory() : Inventory
Refer to getHunterInventory above.
Inventory
2.2: retrieveAll()
trapsInInventory
alt
[((trapsInInventory.size() > 0) && (inputInt <= trapsInInventory.size() && inputInt > 0))]
2.3: setTrap(currentHunter:Hunter, t:Trap) : void
2.3.1: inv = getHunterInventory(hunter:Hunter) : Inventory
2.3.2: removeTrap(selectedTrap:Trap) : void
2.3.3: getTrap() : String
hunterTrap

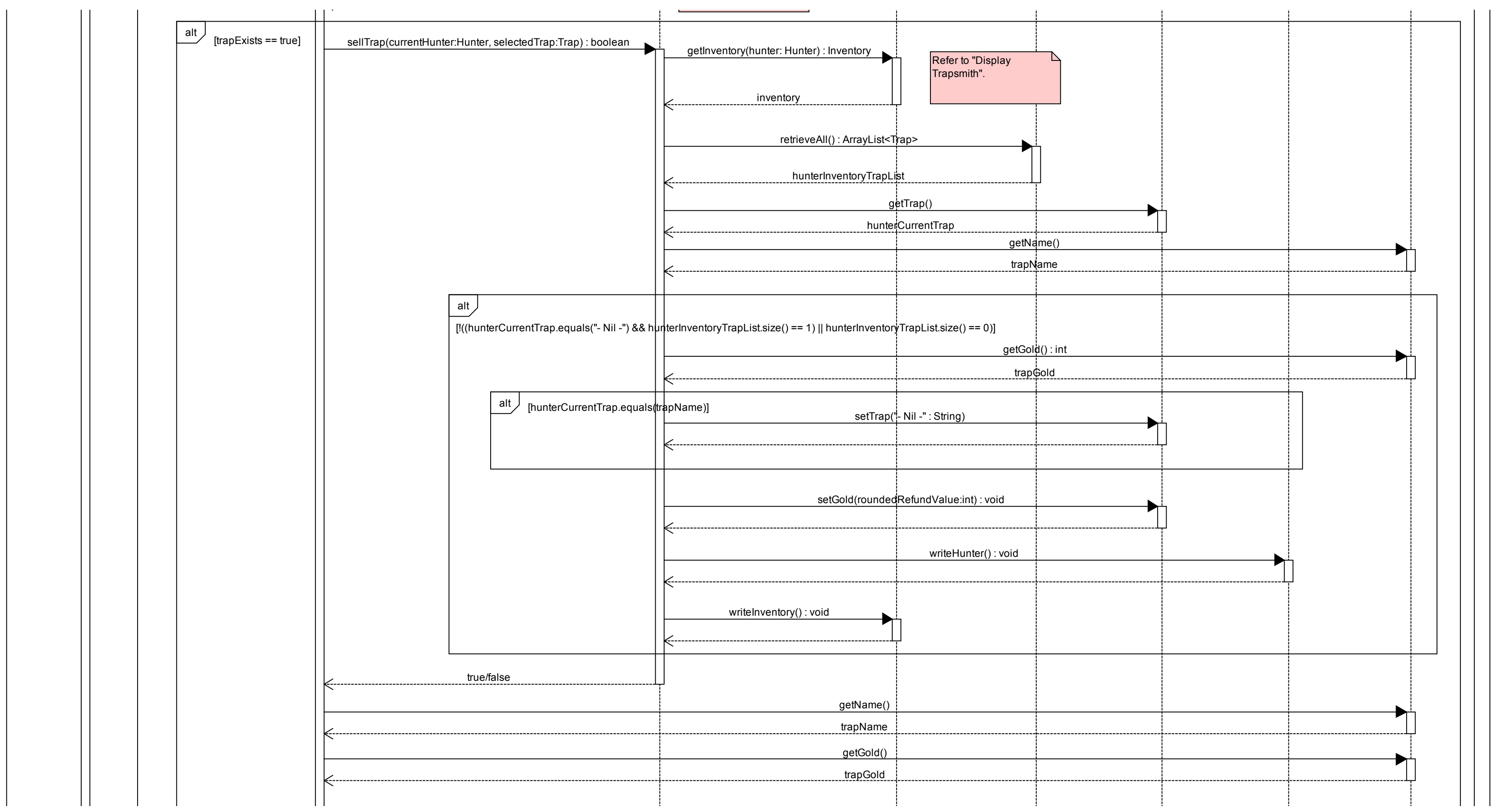

alt
[(!hunterTrap.equals("- Nil -"))]
2.3.4: retrieveTrap(hunterTrap:Trap) : trapBeforeSwap
trapBeforeSwap
2.3.5: addTrap(trapBeforeSwap:Trap) : void
2.3.6: setTrap(selectedTrap.getName():String) : void
2.3.7: retrieveAll() : ArrayList<Trap>
hunterTrapList
2.3.8: sortTraps(hunterTrapList:ArrayList<Trap>)
2.3.9: writeHunter() : void
2.3.10: writeInventory() : void
successful / unsuccessful change of equip trap
successful / unsuccessful message

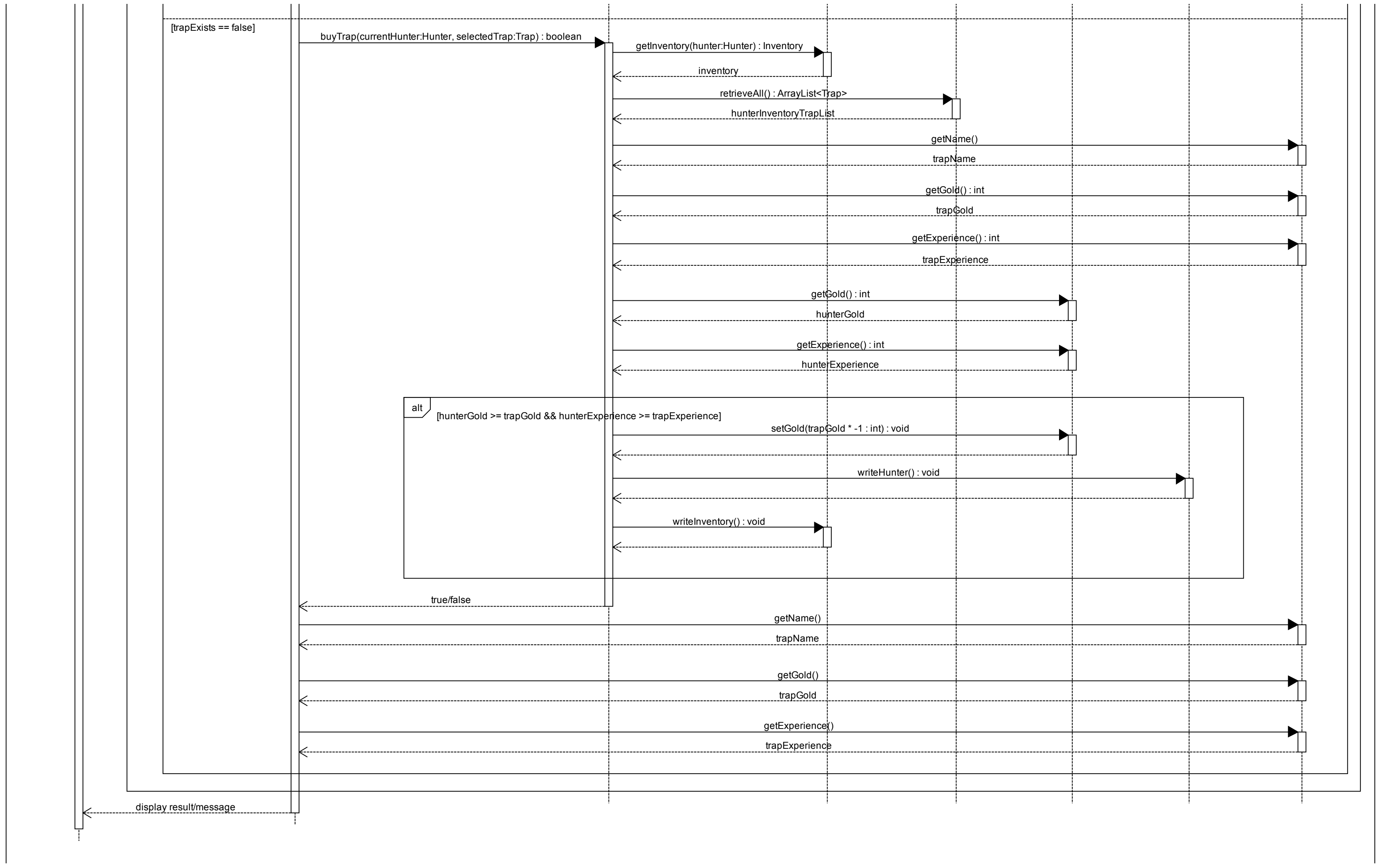
4.3 Buy / Sell Trap


: Hunter
TrapSmithUI
TrapSmithController
Trap
: TrapList
: InventoryList
Hunter
: Inventory
1: visits the trap smith shop() : void
loop [!input.equalsIgnoreCase("r")]
1.1: getExperience() : int
experience
1.2: getGold() : int
gold
1.3: retrieveRegionTrap(currentHunter:Hunter) : ArrayList<Trap>
1.3.1: getRegion()
currentLocation
1.3.2: retrieveRegionTrap(currentLocation)
regionTrapList: ArrayList<Trap>
regionTrapList

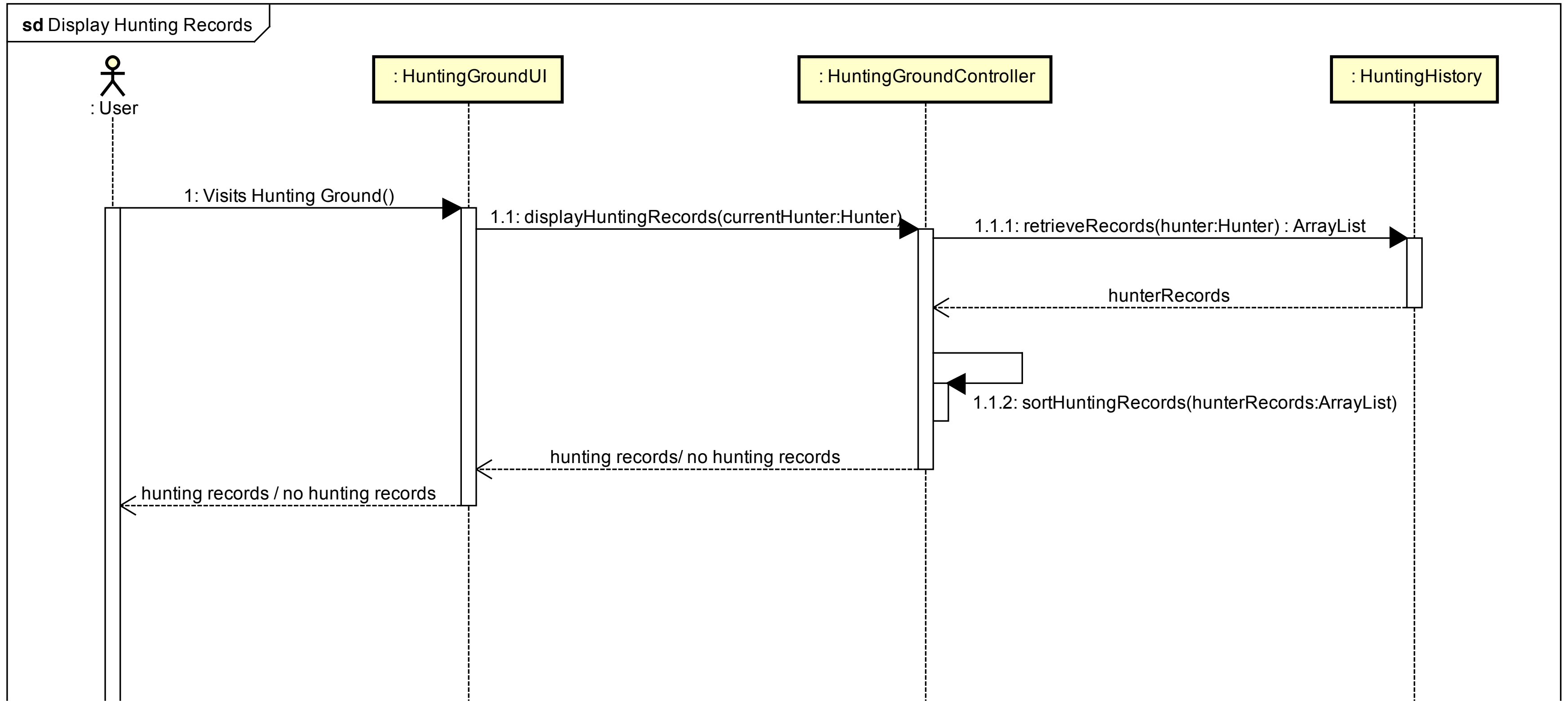

alt
[(regionTrapList.size() != 0)]
loop [i < regionTrapList.size()]
1.4: getName() : String
shopTrapName
1.5: checkTrapExist(shopTrap:Trap, currentHunter:Hunter) : boolean
1.5.1: getInventory(hunter:Hunter)
1.5.1.1: getUsername()
hunterUsername
1.5.1.2: getUsername()
inventoryUsername
hunterInv / null
1.5.2: retrieveAll()
hunterInventoryTrapList
1.5.3: getTrap()
equippedTrap
1.5.4: getName()
selectedTrapName
trapExists = true ; !trapExists = false
1.6: getGold()
trapGold
1.7: getExperience()
trapExperience
display traps for sales and purchase

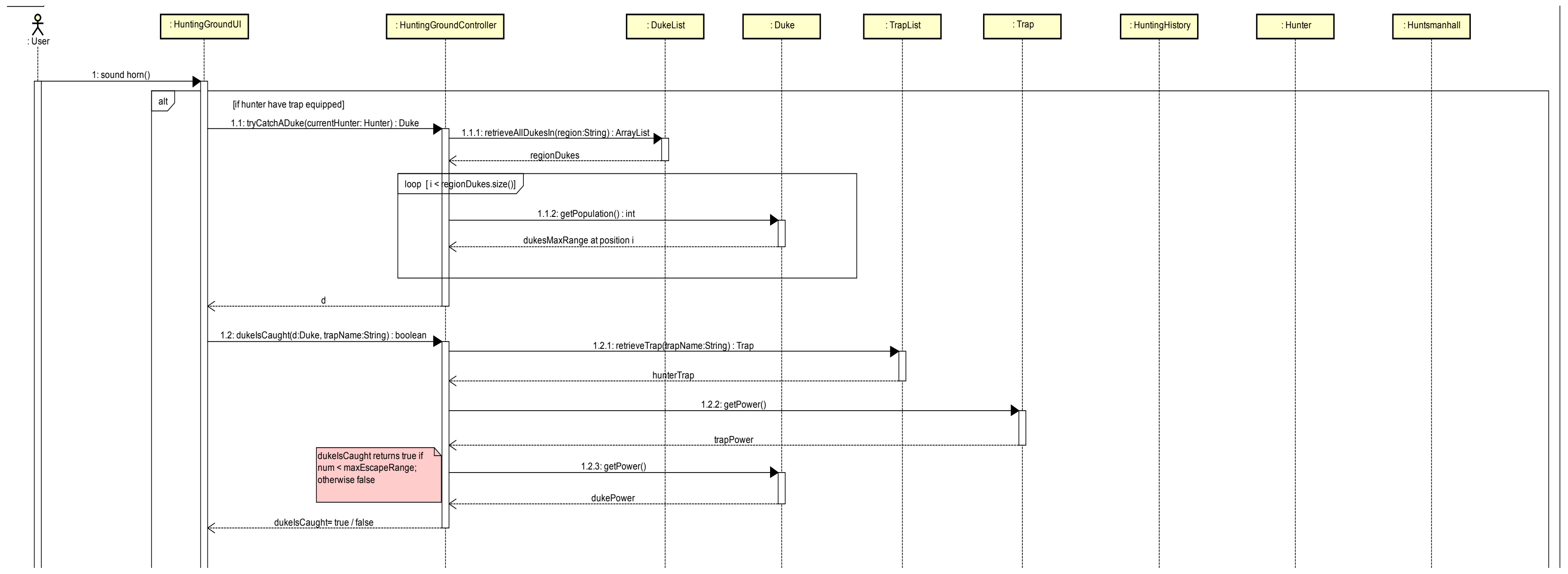

sd processBuySellTrap TrapSmith
: Hunter
TrapSmithUI
TrapSmithController
InventoryList
Inventory
Hunter
HuntsmanHall
Trap
user opts to purchase or sell trap(input:String) : void
retrieveRegionTrap(currentHunter:Hunter) : ArrayList<Trap>
regionTrapList
Refer to "Display Trapsmith".
alt
[(regionTrapList.size() > 0) && (inputInt <= regionTrapList.size() && inputInt > 0)]
checkTrapExist(selectedTrap:Trap,currentHunter:Hunter) : boolean
trapExists = true ; !trapExists = false
Refer to "Display Trapsmith".


alt
[trapExists == true]
sellTrap(currentHunter:Hunter, selectedTrap:Trap) : boolean
getInventory(hunter: Hunter) : Inventory
Refer to "Display Trapsmith".
inventory
retrieveAll() : ArrayList<Trap>
hunterInventoryTrapList
getTrap()
hunterCurrentTrap
getName()
trapName
alt
[!((hunterCurrentTrap.equals("- Nil -") && hunterInventoryTrapList.size() == 1) || hunterInventoryTrapList.size() == 0)]
getGold() : int
trapGold
alt
[hunterCurrentTrap.equals(trapName)]
setTrap("- Nil -" : String)
setGold(roundedRefundValue:int) : void
writeHunter() : void
writeInventory() : void
true/false
getName()
trapName
getGold()
trapGold

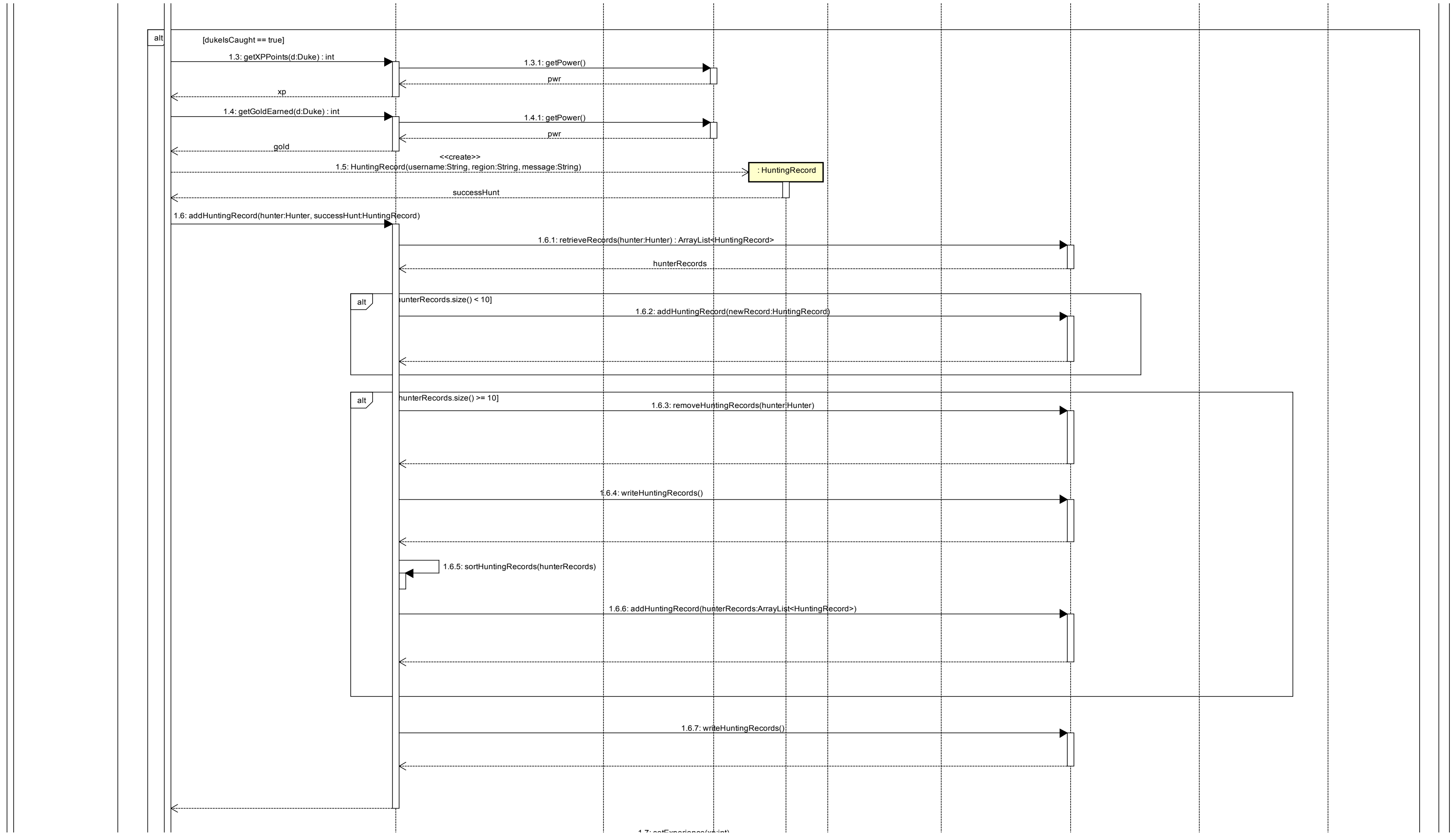

[trapExists == false]
buyTrap(currentHunter:Hunter, selectedTrap:Trap) : boolean
getInventory(hunter:Hunter) : Inventory
inventory
retrieveAll() : ArrayList<Trap>
hunterInventoryTrapList
getName()
trapName
getGold() : int
trapGold
getExperience() : int
trapExperience
getGold() : int
hunterGold
getExperience() : int
hunterExperience
alt
[hunterGold >= trapGold && hunterExperience >= trapExperience]
setGold(trapGold * -1 : int) : void
writeHunter() : void
writeInventory() : void
true/false
getName()
trapName
getGold()
trapGold
getExperience()
trapExperience
display result/message

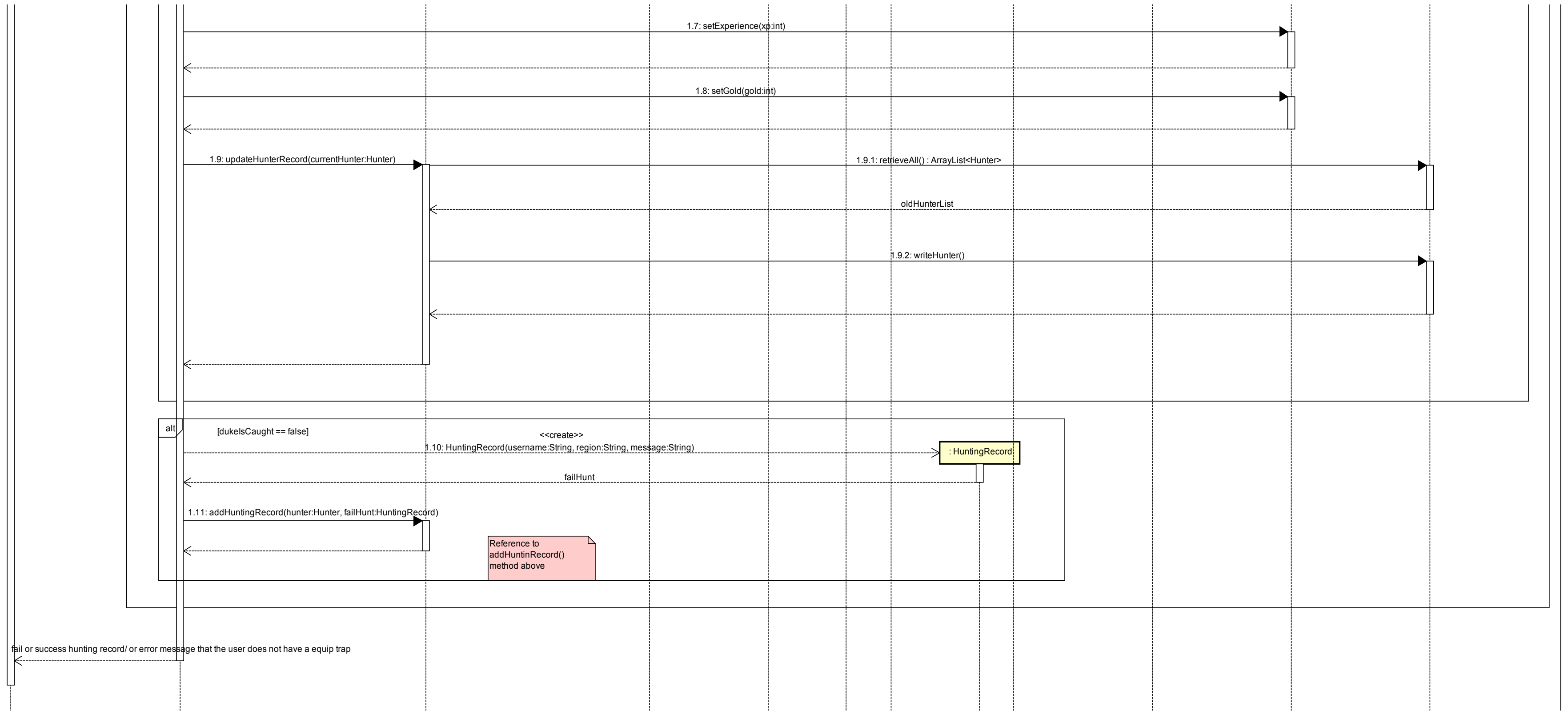
4.4 Sound Horn


sd Display Hunting Records
: User
: HuntingGroundUI
: HuntingGroundController
: HuntingHistory
1: Visits Hunting Ground()
1.1: displayHuntingRecords(currentHunter:Hunter)
1.1.1: retrieveRecords(hunter:Hunter) : ArrayList
hunterRecords
1.1.2: sortHuntingRecords(hunterRecords:ArrayList)
hunting records/ no hunting records
hunting records / no hunting records

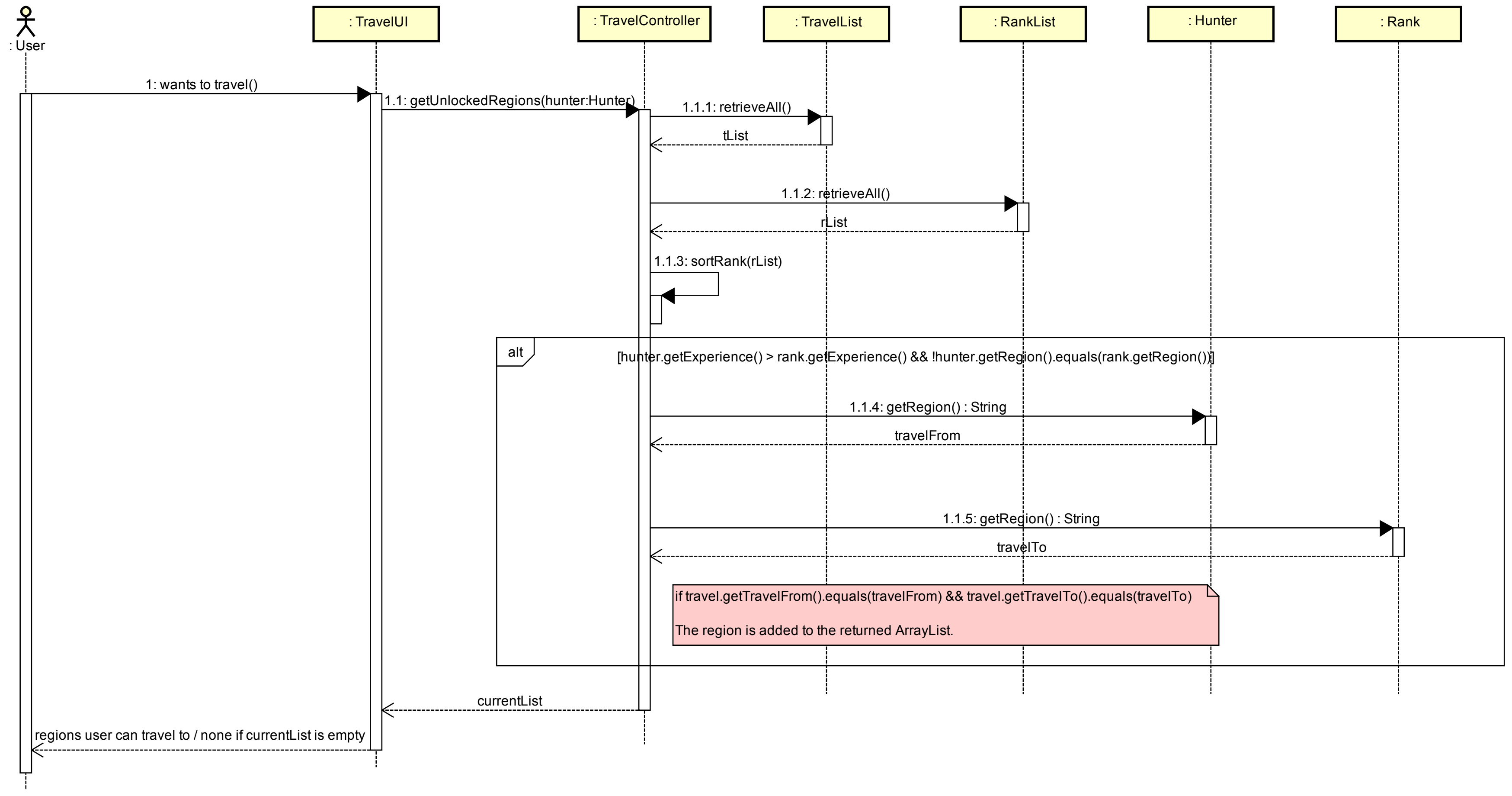


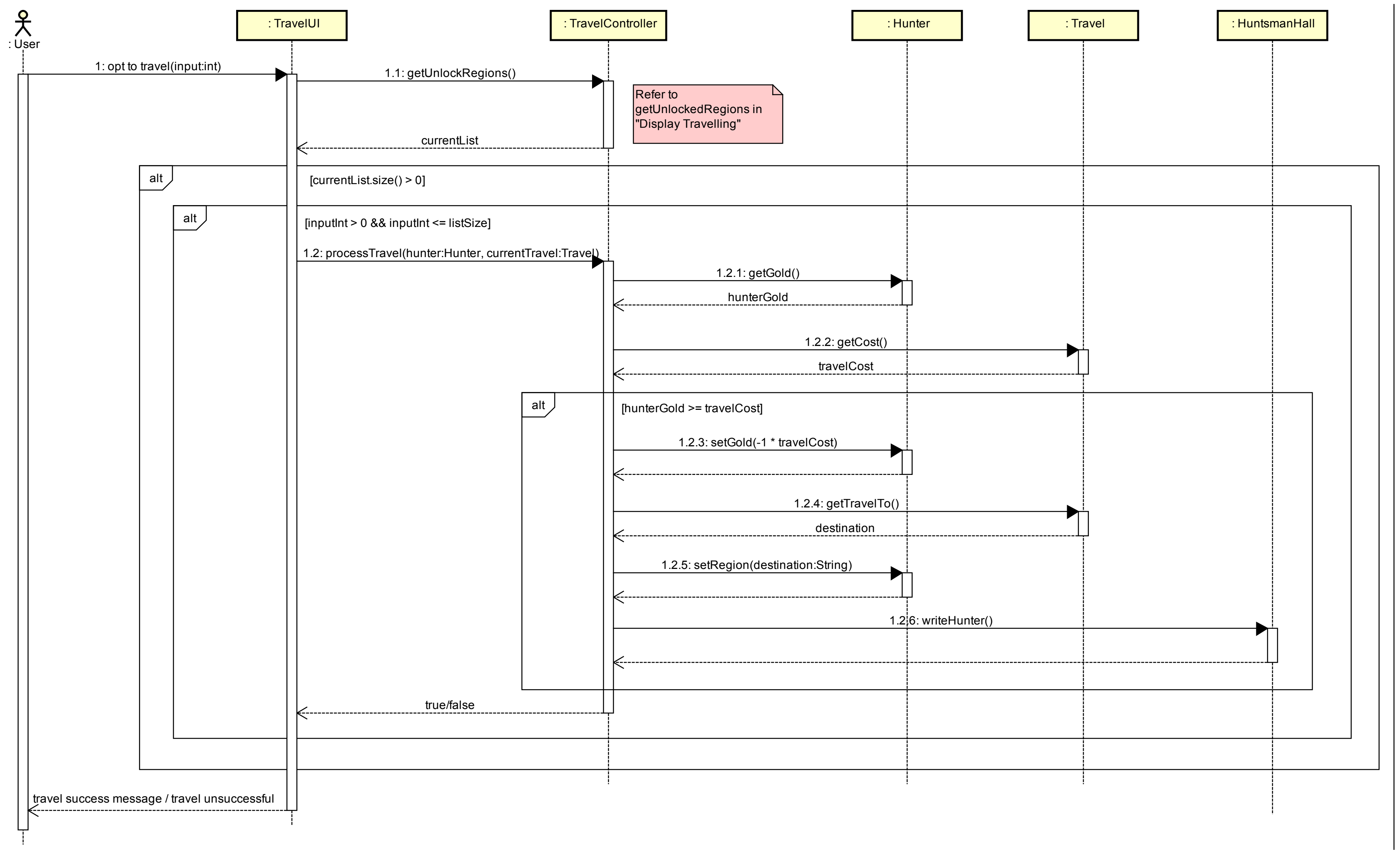
: User
: HuntingGroundUI
: HuntingGroundController
: DukeList
: Duke
: TrapList
: Trap
: HuntingHistory
: Hunter
: Huntsmanhall
1: sound horn()
alt
[if hunter have trap equipped]
1.1: tryCatchADuke(currentHunter: Hunter) : Duke
1.1.1: retrieveAllDukesIn(region:String) : ArrayList
regionDukes
loop [i < regionDukes.size()]
1.1.2: getPopulation() : int
dukesMaxRange at position i
d
1.2: dukeIsCaught(d:Duke, trapName:String) : boolean
1.2.1: retrieveTrap(trapName:String) : Trap
hunterTrap
1.2.2: getPower()
trapPower
dukeIsCaught returns true if num < maxEscapeRange; otherwise false
1.2.3: getPower()
dukePower
dukeIsCaught= true / false


alt
[dukeIsCaught == true]
1.3: getXPPoints(d:Duke) : int
1.3.1: getPower()
pwr
xp
1.4: getGoldEarned(d:Duke) : int
1.4.1: getPower()
pwr
gold
<<create>>
1.5: HuntingRecord(username:String, region:String, message:String)
: HuntingRecord
successHunt
1.6: addHuntingRecord(hunter:Hunter, successHunt:HuntingRecord)
1.6.1: retrieveRecords(hunter:Hunter) : ArrayList<HuntingRecord>
hunterRecords
alt
hunterRecords.size() < 10]
1.6.2: addHuntingRecord(newRecord:HuntingRecord)
alt
hunterRecords.size() >= 10]
1.6.3: removeHuntingRecords(hunter:Hunter)
1.6.4: writeHuntingRecords()
1.6.5: sortHuntingRecords(hunterRecords)
1.6.6: addHuntingRecord(hunterRecords:ArrayList<HuntingRecord>)
1.6.7: writeHuntingRecords()
1.7: setExperience(xp:int)


1.7: setExperience(xp:int)
1.8: setGold(gold:int)
1.9: updateHunterRecord(currentHunter:Hunter)
1.9.1: retrieveAll() : ArrayList<Hunter>
oldHunterList
1.9.2: writeHunter()
alt
[dukeIsCaught == false]
<<create>>
1.10: HuntingRecord(username:String, region:String, message:String)
: HuntingRecord
failHunt
1.11: addHuntingRecord(hunter:Hunter, failHunt:HuntingRecord)
Reference to addHuntinRecord() method above
fail or success hunting record/ or error message that the user does not have a equip trap

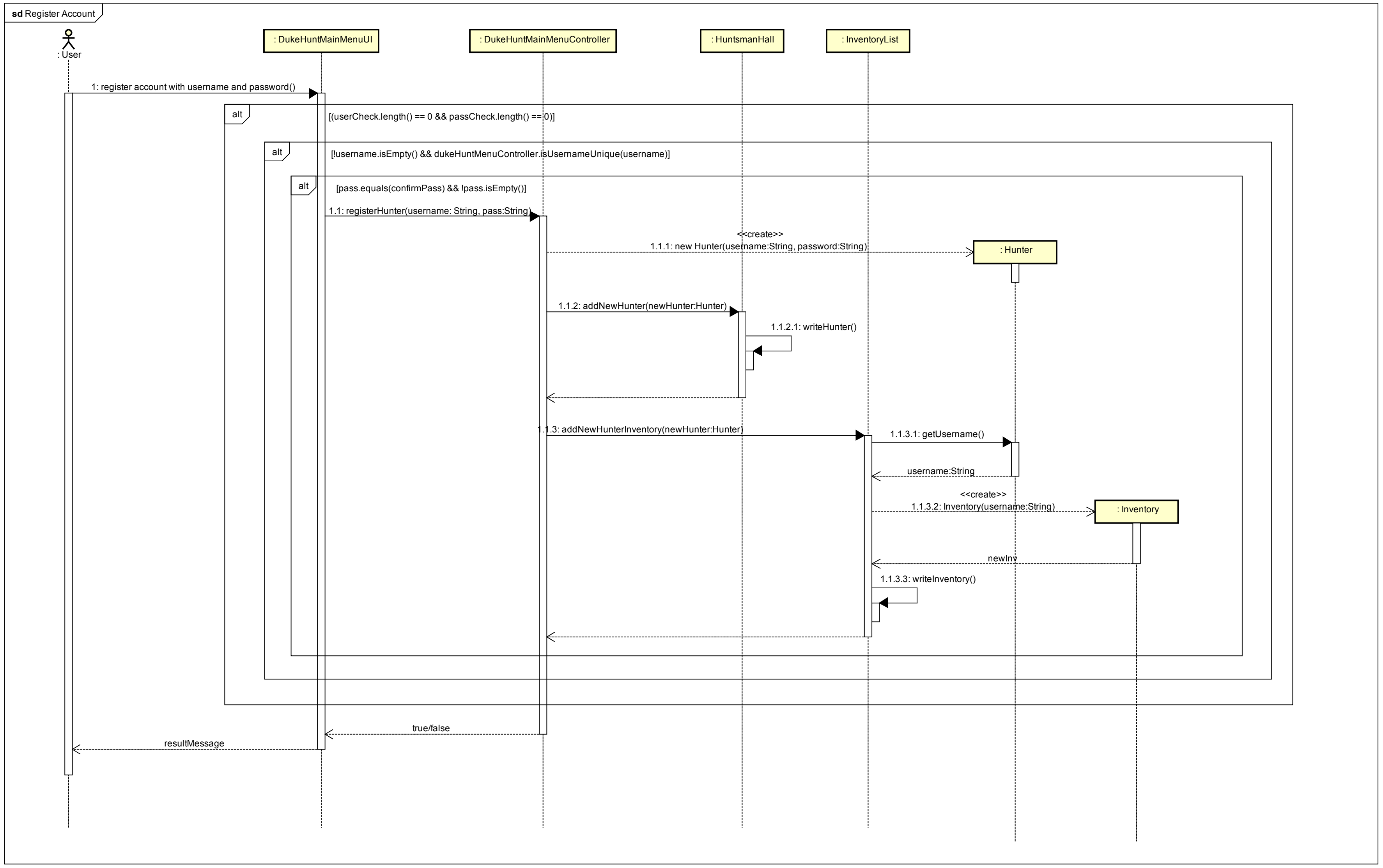
4.5 Travel


: User
: TravelUI
: TravelController
: TravelList
: RankList
: Hunter
: Rank
1: wants to travel()
1.1: getUnlockedRegions(hunter:Hunter)
1.1.1: retrieveAll()
tList
1.1.2: retrieveAll()
rList
1.1.3: sortRank(rList)
alt
[hunter.getExperience() > rank.getExperience() && !hunter.getRegion().equals(rank.getRegion())]
1.1.4: getRegion() : String
travelFrom
1.1.5: getRegion() : String
travelTo
if travel.getTravelFrom().equals(travelFrom) && travel.getTravelTo().equals(travelTo)
The region is added to the returned ArrayList.
currentList
regions user can travel to / none if currentList is empty

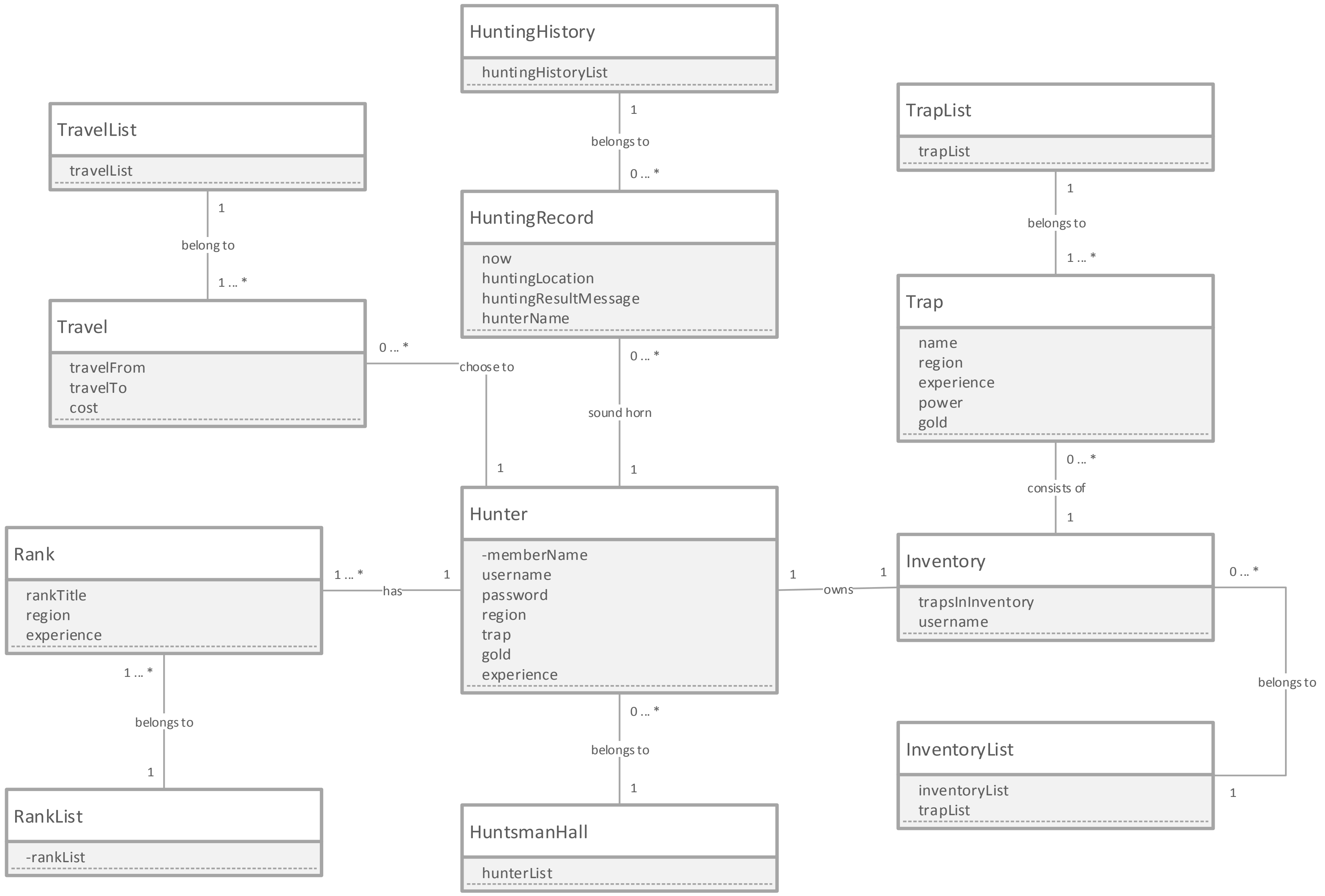


: User
: TravelUI
: TravelController
: Hunter
: Travel
: HuntsmanHall
1: opt to travel(input:int)
1.1: getUnlockRegions()
Refer to getUnlockedRegions in "Display Travelling"
currentList
alt
[currentList.size() > 0]
alt
[inputInt > 0 && inputInt <= listSize]
1.2: processTravel(hunter:Hunter, currentTravel:Travel)
1.2.1: getGold()
hunterGold
1.2.2: getCost()
travelCost
alt
[hunterGold >= travelCost]
1.2.3: setGold(-1 * travelCost)
1.2.4: getTravelTo()
destination
1.2.5: setRegion(destination:String)
1.2.6: writeHunter()
true/false
travel success message / travel unsuccessful

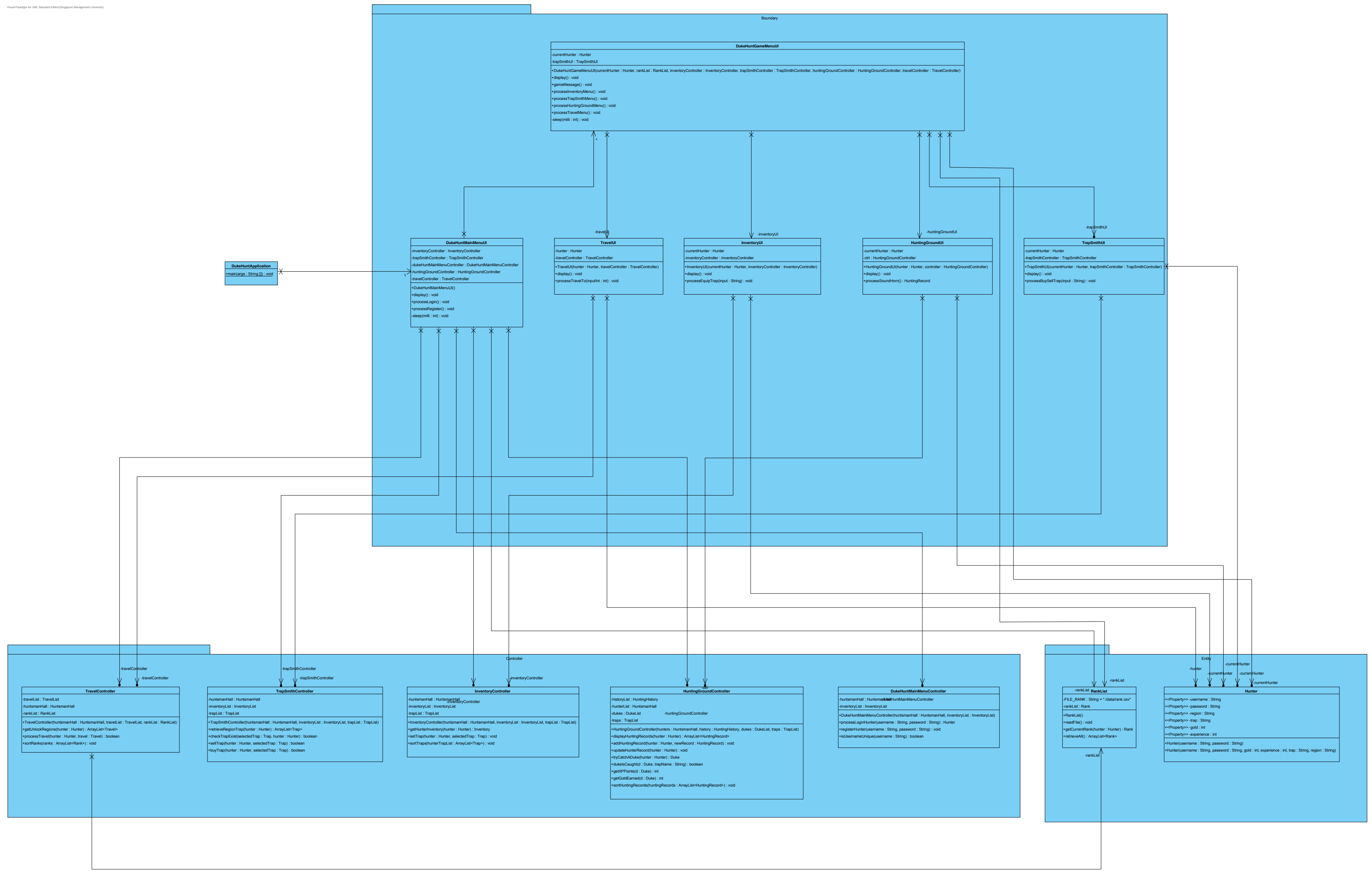
4.6 Register Account

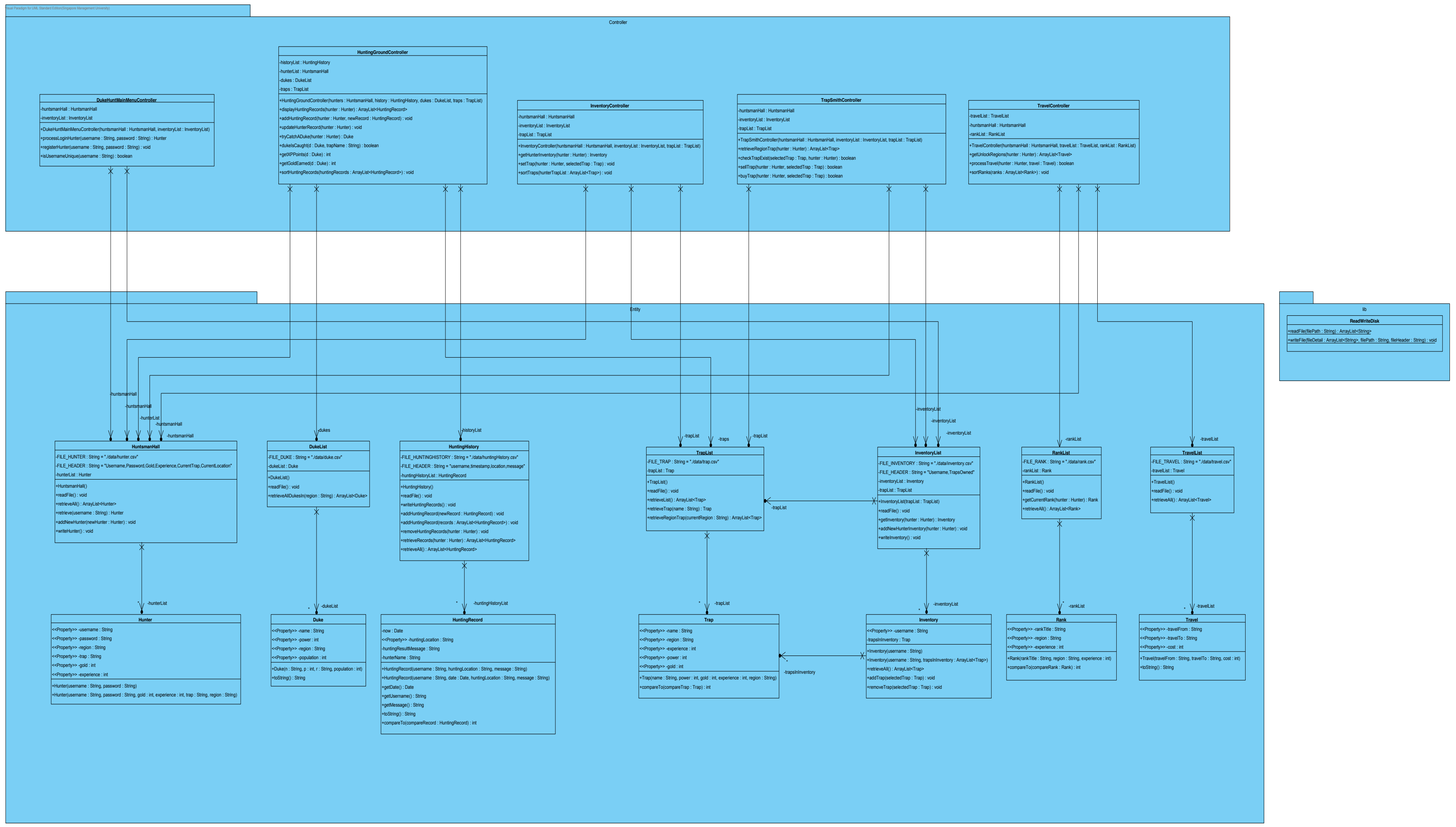

sd Register Account
: User
: DukeHuntMainMenuUI
: DukeHuntMainMenuController
: HuntsmanHall
: InventoryList
1: register account with username and password()
alt
[(userCheck.length() == 0 && passCheck.length() == 0)]
alt
[!username.isEmpty() && dukeHuntMenuController.isUsernameUnique(username)]
alt
[pass.equals(confirmPass) && !pass.isEmpty()]
1.1: registerHunter(username: String, pass:String)
<<create>>
1.1.1: new Hunter(username:String, password:String)
: Hunter
1.1.2: addNewHunter(newHunter:Hunter)
1.1.2.1: writeHunter()
1.1.3: addNewHunterInventory(newHunter:Hunter)
1.1.3.1: getUsername()
username:String
<<create>>
1.1.3.2: Inventory(username:String)
: Inventory
newInv
1.1.3.3: writeInventory()
true/false
resultMessage

5. Domain Diagram



6. Class Diagram


Visual Paradigm for UML Standard Edition(Singapore Management University)
Boundary
DukeHuntGameMenuUI
-currentHunter : Hunter
-trapSmithUI : TrapSmithUI
+DukeHuntGameMenuUI(currentHunter : Hunter, rankList : RankList, inventoryController : InventoryController, trapSmithController : TrapSmithController, huntingGroundController : HuntingGroundController, travelController : TravelController)
+display() : void
+gameMessage() : void
+processInventoryMenu() : void
+processTrapSmithMenu() : void
+processHuntingGroundMenu() : void
+processTravelMenu() : void
-sleep(milli : int) : void
1
DukeHuntApplication
+main(args : String []) : void
1
DukeHuntMainMenuUI
-inventoryController : InventoryController
-trapSmithController : TrapSmithController
-dukeHuntMainMenuController : DukeHuntMainMenuController
-huntingGroundController : HuntingGroundController
-travelController : TravelController
+DukeHuntMainMenuUI()
+display() : void
+processLogin() : void
+processRegister() : void
-sleep(milli : int) : void
-travelUI
TravelUI
-hunter : Hunter
-travelController : TravelController
+TravelUI(hunter : Hunter, travelController : TravelController)
+display() : void
+processTravelTo(inputInt : int) : void
-inventoryUI
InventoryUI
-currentHunter : Hunter
-inventoryController : InventoryController
+InventoryUI(currentHunter : Hunter, inventoryController : InventoryController)
+display() : void
+processEquipTrap(input : String) : void
-huntingGroundUI
HuntingGroundUI
-currentHunter : Hunter
-ctrl : HuntingGroundController
+HuntingGroundUI(hunter : Hunter, controller : HuntingGroundController)
+display() : void
+processSoundHorn() : HuntingRecord
-trapSmithUI
TrapSmithUI
-currentHunter : Hunter
-trapSmithController : TrapSmithController
+TrapSmithUI(currentHunter : Hunter, trapSmithController : TrapSmithController)
+display() : void
+processBuySellTrap(input : String) : void
Controller
-travelController
-travelController
TravelController
-travelList : TravelList
-huntsmanHall : HuntsmanHall
-rankList : RankList
+TravelController(huntsmanHall : HuntsmanHall, travelList : TravelList, rankList : RankList)
+getUnlockRegions(hunter : Hunter) : ArrayList<Travel>
+processTravel(hunter : Hunter, travel : Travel) : boolean
+sortRanks(ranks : ArrayList<Rank>) : void
-trapSmithController
-trapSmithController
TrapSmithController
-huntsmanHall : HuntsmanHall
-inventoryList : InventoryList
-trapList : TrapList
+TrapSmithController(huntsmanHall : HuntsmanHall, inventoryList : InventoryList, trapList : TrapList)
+retrieveRegionTrap(hunter : Hunter) : ArrayList<Trap>
+checkTrapExist(selectedTrap : Trap, hunter : Hunter) : boolean
+sellTrap(hunter : Hunter, selectedTrap : Trap) : boolean
+buyTrap(hunter : Hunter, selectedTrap : Trap) : boolean
-inventoryController
-inventoryController
InventoryController
-huntsmanHall : HuntsmanHall
-inventoryList : InventoryList
-trapList : TrapList
+InventoryController(huntsmanHall : HuntsmanHall, inventoryList : InventoryList, trapList : TrapList)
+getHunterInventory(hunter : Hunter) : Inventory
+setTrap(hunter : Hunter, selectedTrap : Trap) : void
+sortTraps(hunterTrapList : ArrayList<Trap>) : void
-ctrl
-huntingGroundController
HuntingGroundController
-historyList : HuntingHistory
-hunterList : HuntsmanHall
-dukes : DukeList
-traps : TrapList
+HuntingGroundController(hunters : HuntsmanHall, history : HuntingHistory, dukes : DukeList, traps : TrapList)
+displayHuntingRecords(hunter : Hunter) : ArrayList<HuntingRecord>
+addHuntingRecord(hunter : Hunter, newRecord : HuntingRecord) : void
+updateHunterRecord(hunter : Hunter) : void
+tryCatchADuke(hunter : Hunter) : Duke
+dukeIsCaught(d : Duke, trapName : String) : boolean
+getXPPoints(d : Duke) : int
+getGoldEarned(d : Duke) : int
+sortHuntingRecords(huntingRecords : ArrayList<HuntingRecord>) : void
-dukeHuntMainMenuController
DukeHuntMainMenuController
-huntsmanHall : HuntsmanHall
-inventoryList : InventoryList
+DukeHuntMainMenuController(huntsmanHall : HuntsmanHall, inventoryList : InventoryList)
+processLoginHunter(username : String, password : String) : Hunter
+registerHunter(username : String, password : String) : void
+isUsernameUnique(username : String) : boolean
Entity
-rankList
-rankList
RankList
-FILE_RANK : String = "./data/rank.csv"
-rankList : Rank
+RankList()
+readFile() : void
+getCurrentRank(hunter : Hunter) : Rank
+retrieveAll() : ArrayList<Rank>
-rankList
-hunter
-currentHunter
-currentHunter
-currentHunter
-currentHunter
Hunter
<<Property>> -username : String
<<Property>> -password : String
<<Property>> -region : String
<<Property>> -trap : String
<<Property>> -gold : int
<<Property>> -experience : int
+Hunter(username : String, password : String)
+Hunter(username : String, password : String, gold : int, experience : int, trap : String, region : String)


Visual Paradigm for UML Standard Edition(Singapore Management University)
Controller
DukeHuntMainMenuController
-huntsmanHall : HuntsmanHall
-inventoryList : InventoryList
+DukeHuntMainMenuController(huntsmanHall : HuntsmanHall, inventoryList : InventoryList)
+processLoginHunter(username : String, password : String) : Hunter
+registerHunter(username : String, password : String) : void
+isUsernameUnique(username : String) : boolean
HuntingGroundController
-historyList : HuntingHistory
-hunterList : HuntsmanHall
-dukes : DukeList
-traps : TrapList
+HuntingGroundController(hunters : HuntsmanHall, history : HuntingHistory, dukes : DukeList, traps : TrapList)
+displayHuntingRecords(hunter : Hunter) : ArrayList<HuntingRecord>
+addHuntingRecord(hunter : Hunter, newRecord : HuntingRecord) : void
+updateHunterRecord(hunter : Hunter) : void
+tryCatchADuke(hunter : Hunter) : Duke
+dukeIsCaught(d : Duke, trapName : String) : boolean
+getXPPoints(d : Duke) : int
+getGoldEarned(d : Duke) : int
+sortHuntingRecords(huntingRecords : ArrayList<HuntingRecord>) : void
InventoryController
-huntsmanHall : HuntsmanHall
-inventoryList : InventoryList
-trapList : TrapList
+InventoryController(huntsmanHall : HuntsmanHall, inventoryList : InventoryList, trapList : TrapList)
+getHunterInventory(hunter : Hunter) : Inventory
+setTrap(hunter : Hunter, selectedTrap : Trap) : void
+sortTraps(hunterTrapList : ArrayList<Trap>) : void
TrapSmithController
-huntsmanHall : HuntsmanHall
-inventoryList : InventoryList
-trapList : TrapList
+TrapSmithController(huntsmanHall : HuntsmanHall, inventoryList : InventoryList, trapList : TrapList)
+retrieveRegionTrap(hunter : Hunter) : ArrayList<Trap>
+checkTrapExist(selectedTrap : Trap, hunter : Hunter) : boolean
+sellTrap(hunter : Hunter, selectedTrap : Trap) : boolean
+buyTrap(hunter : Hunter, selectedTrap : Trap) : boolean
TravelController
-travelList : TravelList
-huntsmanHall : HuntsmanHall
-rankList : RankList
+TravelController(huntsmanHall : HuntsmanHall, travelList : TravelList, rankList : RankList)
+getUnlockRegions(hunter : Hunter) : ArrayList<Travel>
+processTravel(hunter : Hunter, travel : Travel) : boolean
+sortRanks(ranks : ArrayList<Rank>) : void
Entity
-huntsmanHall
-huntsmanHall
-hunterList
-huntsmanHall
-huntsmanHall
HuntsmanHall
-FILE_HUNTER : String = "./data/hunter.csv"
-FILE_HEADER : String = "Username,Password,Gold,Experience,CurrentTrap,CurrentLocation"
-hunterList : Hunter
+HuntsmanHall()
+readFile() : void
+retrieveAll() : ArrayList<Hunter>
+retrieve(username : String) : Hunter
+addNewHunter(newHunter : Hunter) : void
+writeHunter() : void
-dukes
DukeList
-FILE_DUKE : String = "./data/duke.csv"
-dukeList : Duke
+DukeList()
+readFile() : void
+retrieveAllDukesIn(region : String) : ArrayList<Duke>
-historyList
HuntingHistory
-FILE_HUNTINGHISTORY : String = "./data/huntingHistory.csv"
-FILE_HEADER : String = "username,timestamp,location,message"
-huntingHistoryList : HuntingRecord
+HuntingHistory()
+readFile() : void
+writeHuntingRecords() : void
+addHuntingRecord(newRecord : HuntingRecord) : void
+addHuntingRecord(records : ArrayList<HuntingRecord>) : void
+removeHuntingRecords(hunter : Hunter) : void
+retrieveRecords(hunter : Hunter) : ArrayList<HuntingRecord>
+retrieveAll() : ArrayList<HuntingRecord>
-trapList
-traps
-trapList
TrapList
-FILE_TRAP : String = "./data/trap.csv"
-trapList : Trap
+TrapList()
+readFile() : void
+retrieveList() : ArrayList<Trap>
+retrieveTrap(name : String) : Trap
+retrieveRegionTrap(currentRegion : String) : ArrayList<Trap>
-trapList
-inventoryList
-inventoryList
-inventoryList
InventoryList
-FILE_INVENTORY : String = "./data/inventory.csv"
-FILE_HEADER : String = "Username,TrapsOwned"
-inventoryList : Inventory
-trapList : TrapList
+InventoryList(trapList : TrapList)
+readFile() : void
+getInventory(hunter : Hunter) : Inventory
+addNewHunterInventory(hunter : Hunter) : void
+writeInventory() : void
-rankList
RankList
-FILE_RANK : String = "./data/rank.csv"
-rankList : Rank
+RankList()
+readFile() : void
+getCurrentRank(hunter : Hunter) : Rank
+retrieveAll() : ArrayList<Rank>
-travelList
TravelList
-FILE_TRAVEL : String = "./data/travel.csv"
-travelList : Travel
+TravelList()
+readFile() : void
+retrieveAll() : ArrayList<Travel>
*
-hunterList
Hunter
<<Property>> -username : String
<<Property>> -password : String
<<Property>> -region : String
<<Property>> -trap : String
<<Property>> -gold : int
<<Property>> -experience : int
+Hunter(username : String, password : String)
+Hunter(username : String, password : String, gold : int, experience : int, trap : String, region : String)
*
-dukeList
Duke
<<Property>> -name : String
<<Property>> -power : int
<<Property>> -region : String
<<Property>> -population : int
+Duke(n : String, p : int, r : String, population : int)
+toString() : String
*
-huntingHistoryList
HuntingRecord
-now : Date
<<Property>> -huntingLocation : String
-huntingResultMessage : String
-hunterName : String
+HuntingRecord(username : String, huntingLocation : String, message : String)
+HuntingRecord(username : String, date : Date, huntingLocation : String, message : String)
+getDate() : Date
+getUsername() : String
+getMessage() : String
+toString() : String
+compareTo(compareRecord : HuntingRecord) : int
*
-trapList
Trap
<<Property>> -name : String
<<Property>> -region : String
<<Property>> -experience : int
<<Property>> -power : int
<<Property>> -gold : int
+Trap(name : String, power : int, gold : int, experience : int, region : String)
+compareTo(compareTrap : Trap) : int
*
-trapsInInventory
*
-inventoryList
Inventory
<<Property>> -username : String
-trapsInInventory : Trap
+Inventory(username : String)
+Inventory(username : String, trapsInInventory : ArrayList<Trap>)
+retrieveAll() : ArrayList<Trap>
+addTrap(selectedTrap : Trap) : void
+removeTrap(selectedTrap : Trap) : void
*
-rankList
Rank
<<Property>> -rankTitle : String
<<Property>> -region : String
<<Property>> -experience : int
+Rank(rankTitle : String, region : String, experience : int)
+compareTo(compareRank : Rank) : int
*
-travelList
Travel
<<Property>> -travelFrom : String
<<Property>> -travelTo : String
<<Property>> -cost : int
+Travel(travelFrom : String, travelTo : String, cost : int)
+toString() : String
lib
ReadWriteDisk
+readFile(filePath : String) : ArrayList<String>
+writeFile(fileDetail : ArrayList<String>, filePath : String, fileHeader : String) : void

7. Object-oriented design considerations

1. <u>Interface Segregation Principle (ISP)</u>

When designing Duke Hunt application, our group has adopted the Interface Segregation Principle (ISP) considering the number of functionalities that the application has. As Martin, Robert C. says, "Many client specific interfaces are better than one general purpose interface". This has allowed our team:

1. More flexibility when distributing work; each of us can code our assigned tasks simultaneously as compared to coding in a single interface file.
2. To consume less time when debugging; since each interfaces are segregated, the amount of codes per class file is reduced thus make reading the codes easier while debugging.

With this principle, if the client wished to add a new functionality to the application, he would only need to import the new functionality and instantiate in the main and game file class.

2. <u>Use of Entity, Boundary and Controller class & package</u>

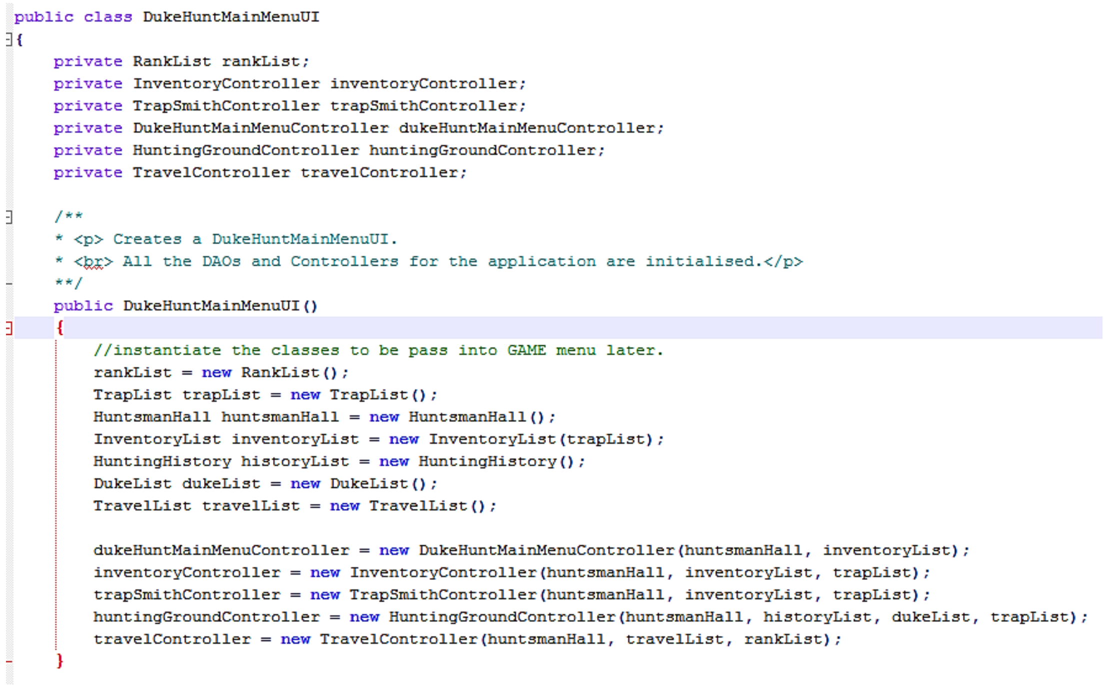
Our team have made use of the concept of entity, boundary and controller class to enforce SRP design principle. For each use case, it has its own controller and boundary class. Each controller and user interface has no relationship with other functionalities' user interface and controller. This is to ensure that the application would not be rigid and easy to integrate. All classes are also packaged into Entity, Boundary and Controller package.

3. <u>Single Responsibility Principle (SRP)</u>

In our Duke Hunt application, each class is assigned to only one responsibility. All data managers (E.g. TrapList, HuntsmanHall) are responsible for managing the assigned object (In this case, TrapList $\rightarrow$ Trap, Huntsmanhall $\rightarrow$ Hunter) as well as invoking the read and write methods in ReadWriteDisk to store data into the .csv files. This makes each class/ methods easy to understand and easy to maintain as changes are isolated from other methods.

4. Initiating all required user interfaces/controllers/entities for Duke Hunt in Main Menu interface class

When designing the application, our team has decided to initiate all required interfaces and controllers in the Main Menu user interface. When the user has logged in successfully, entities (the hunter, rankList) and all controllers will be passed into the Game Menu user interface.



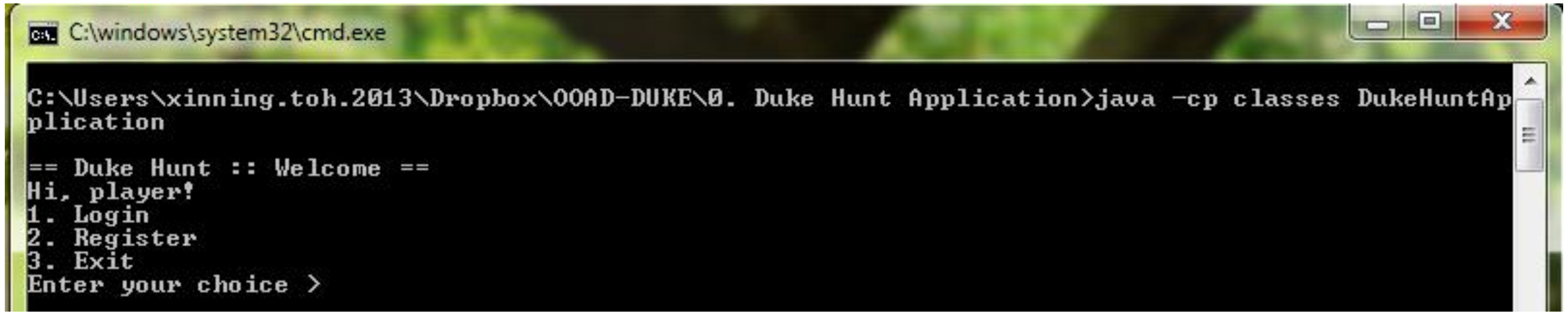
```
public class DukeHuntMainMenuUI
{
    private RankList rankList;
    private InventoryController inventoryController;
    private TrapSmithController trapSmithController;
    private DukeHuntMainMenuController dukeHuntMainMenuController;
    private HuntingGroundController huntingGroundController;
    private TravelController travelController;

    /**
    * <p> Creates a DukeHuntMainMenuUI.
    * <br> All the DAOs and Controllers for the application are initialised.</p>
    **/
    public DukeHuntMainMenuUI()
    {
        //instantiate the classes to be pass into GAME menu later.
        rankList = new RankList();
        TrapList trapList = new TrapList();
        HuntsmanHall huntsmanHall = new HuntsmanHall();
        InventoryList inventoryList = new InventoryList(trapList);
        HuntingHistory historyList = new HuntingHistory();
        DukeList dukeList = new DukeList();
        TravelList travelList = new TravelList();

        dukeHuntMainMenuController = new DukeHuntMainMenuController(huntsmanHall, inventoryList);
        inventoryController = new InventoryController(huntsmanHall, inventoryList, trapList);
        trapSmithController = new TrapSmithController(huntsmanHall, inventoryList, trapList);
        huntingGroundController = new HuntingGroundController(huntsmanHall, historyList, dukeList, trapList);
        travelController = new TravelController(huntsmanHall, travelList, rankList);
    }
```

Why do we choose to do it this way?

Considering that there are several Data Manager entities such as HuntsmanHall, InventoryList and TrapList being required in “My Inventory”, “My Inventory” and “Travel” functionalities, this will help to ensure consistency and make sure that each HuntsmanHall Data Manager entities for example; are pointing to the same object with the same object address. This will ensure consistency and makes it easier when updating the hunter’s attributes such as gold, experience, etc.

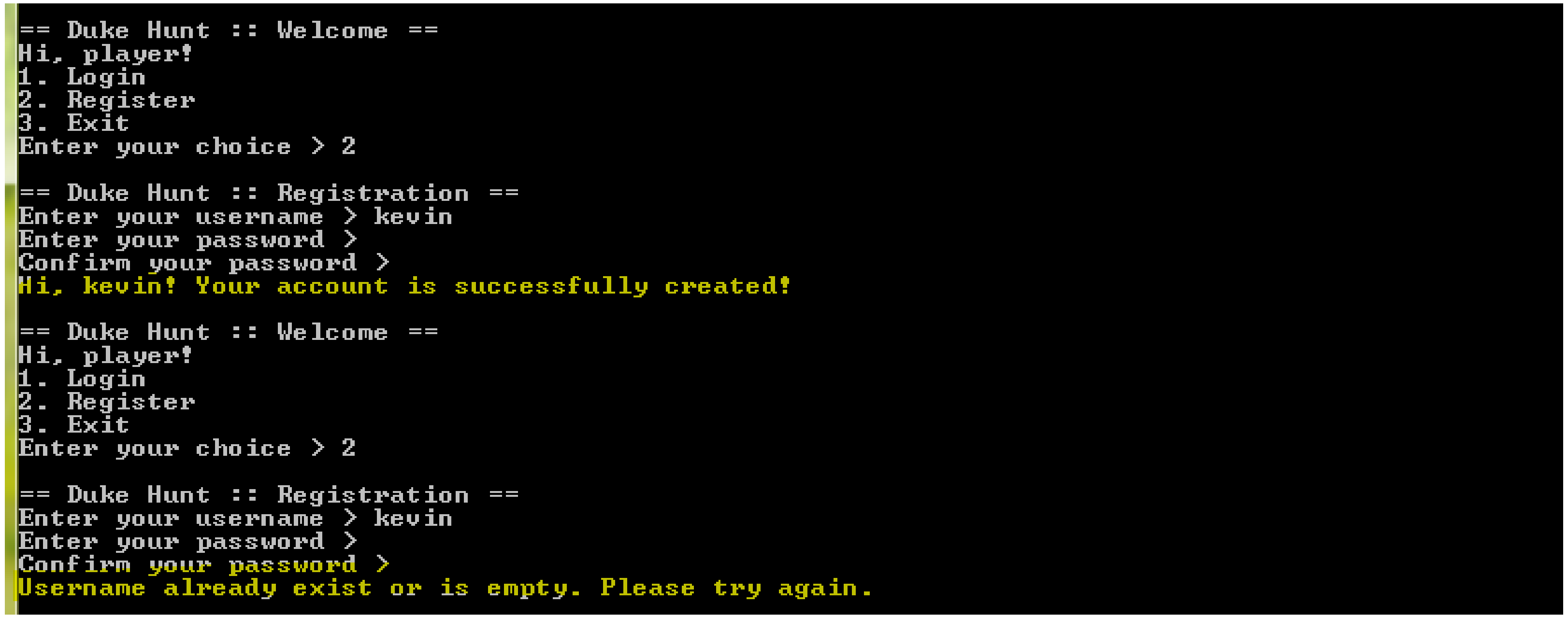
8. Deliverables

This is the main interface that the user will see when the application is run.



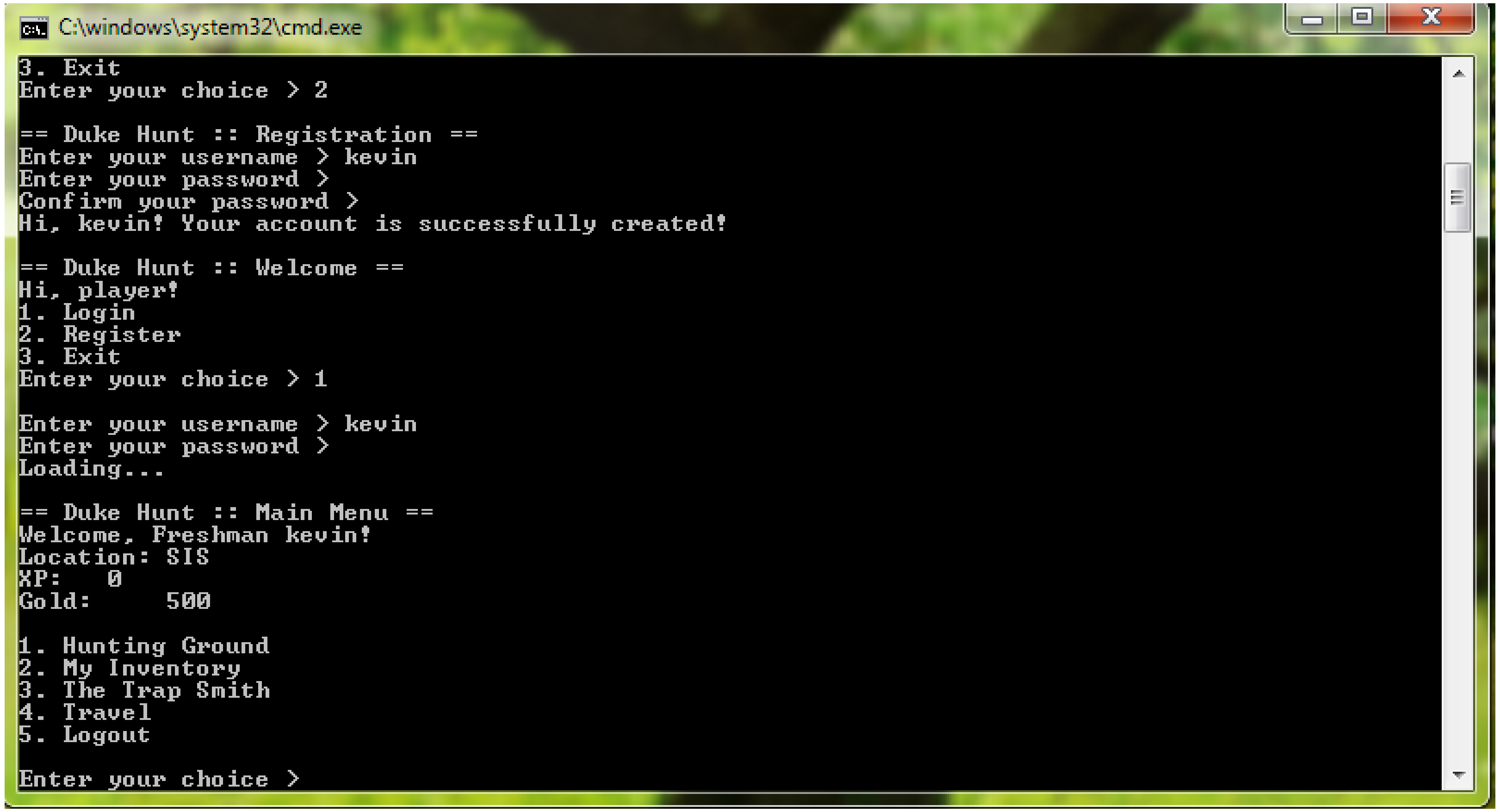
```
C:\Users\xinning.toh.2013\Dropbox\OOAD-DUKE\0. Duke Hunt Application>java -cp classes DukeHuntAp
plication

== Duke Hunt :: Welcome ==
Hi, player!
1. Login
2. Register
3. Exit
Enter your choice >
```

Passwords entered are masked. When user tries to create another account with same username, an error occurs.



```
== Duke Hunt :: Welcome ==
Hi, player!
1. Login
2. Register
3. Exit
Enter your choice > 2

== Duke Hunt :: Registration ==
Enter your username > kevin
Enter your password >
Confirm your password >
Hi, kevin! Your account is successfully created!

== Duke Hunt :: Welcome ==
Hi, player!
1. Login
2. Register
3. Exit
Enter your choice > 2

== Duke Hunt :: Registration ==
Enter your username > kevin
Enter your password >
Confirm your password >
Username already exist or is empty. Please try again.
```

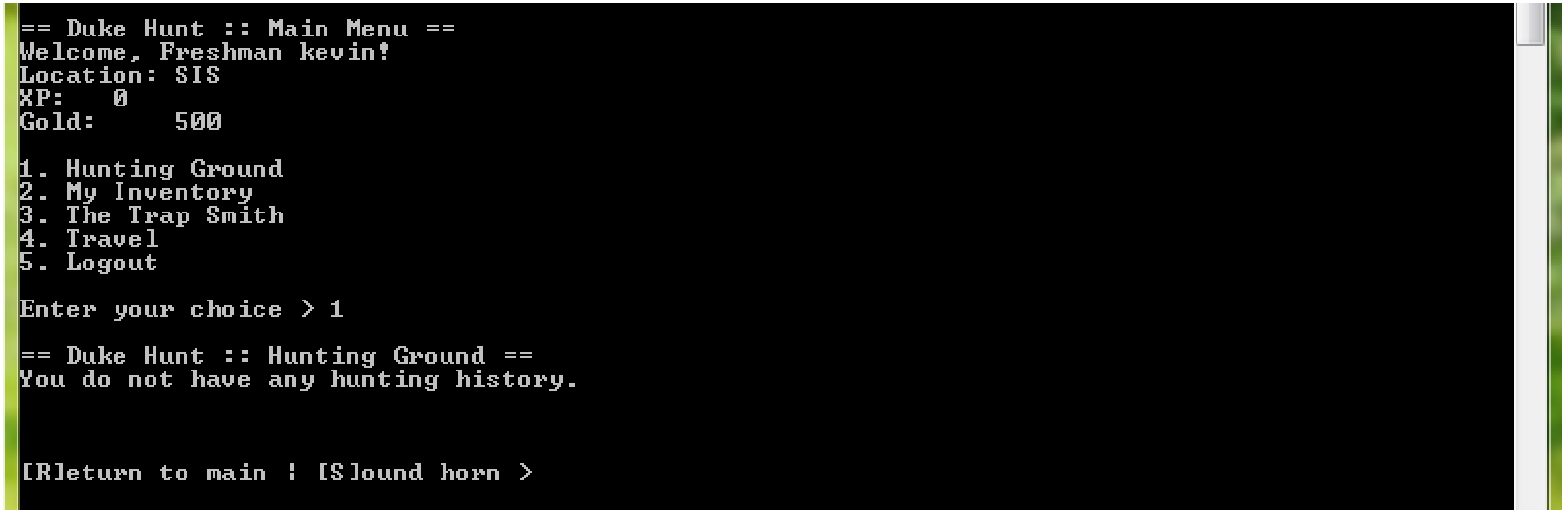
When user logs in, the user interface will show the user's current rank, location, XP and gold.



C:\windows\system32\cmd.exe

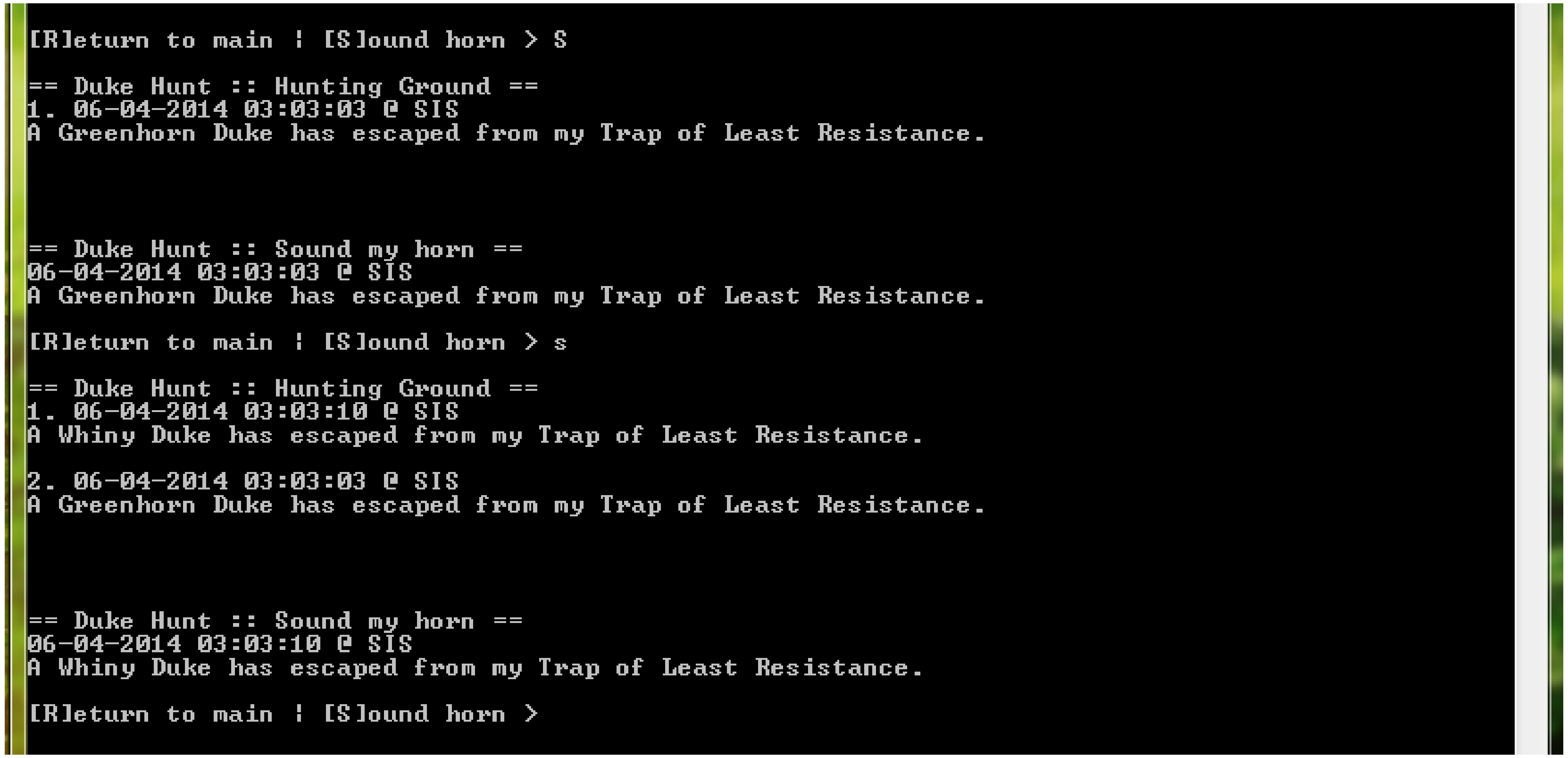
```
3. Exit
Enter your choice > 2

== Duke Hunt :: Registration ==
Enter your username > kevin
Enter your password >
Confirm your password >
Hi, kevin! Your account is successfully created!

== Duke Hunt :: Welcome ==
Hi, player!
1. Login
2. Register
3. Exit
Enter your choice > 1

Enter your username > kevin
Enter your password >
Loading...

== Duke Hunt :: Main Menu ==
Welcome, Freshman kevin!
Location: SIS
XP:    0
Gold:      500

1. Hunting Ground
2. My Inventory
3. The Trap Smith
4. Travel
5. Logout

Enter your choice >
```

When a new user logs in and enters the hunting ground, the application will tell the user he does not have any hunting history yet.



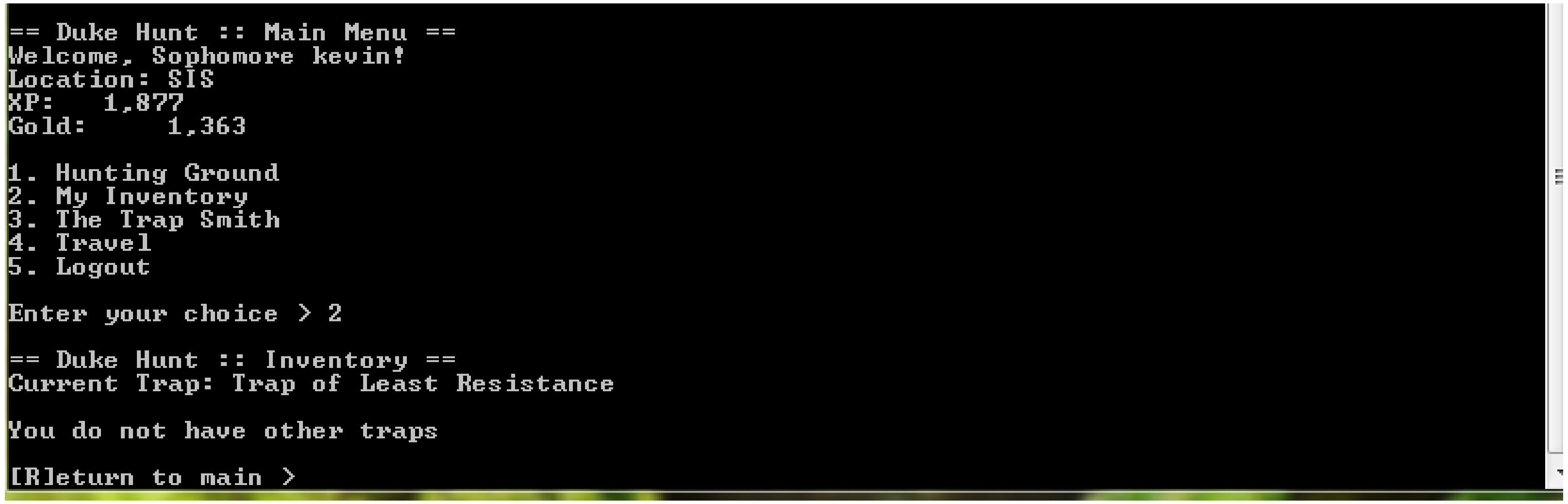
```
== Duke Hunt :: Main Menu ==
Welcome, Freshman kevin!
Location: SIS
XP:    0
Gold:      500

1. Hunting Ground
2. My Inventory
3. The Trap Smith
4. Travel
5. Logout

Enter your choice > 1

== Duke Hunt :: Hunting Ground ==
You do not have any hunting history.



[R]eturn to main | [S]ound horn >
```

When the user sounds horn, a hunting record is created. The records are sorted from the latest (top) to the oldest (bottom). Only the latest 10 records are retained.



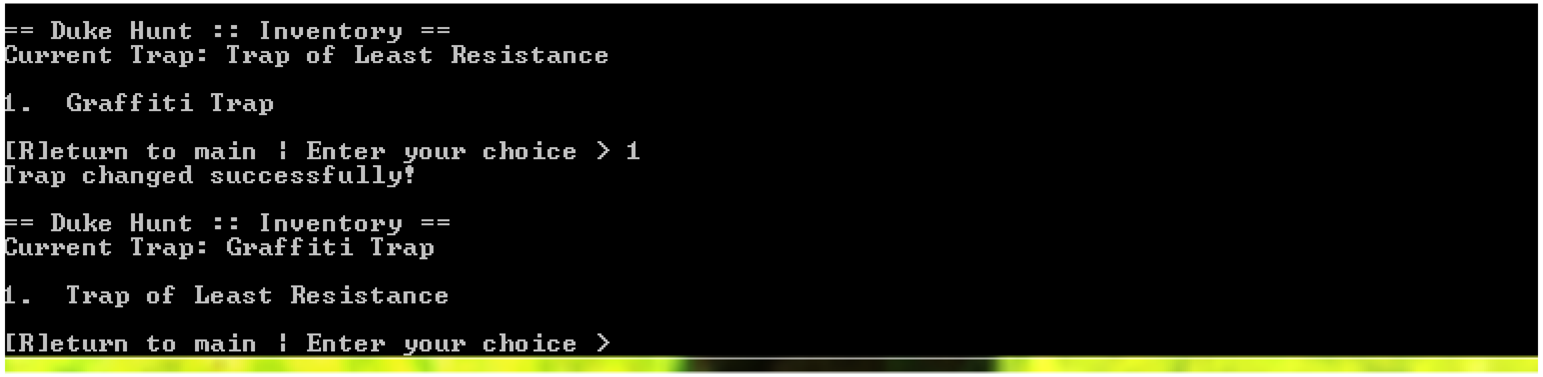
```
[R]eturn to main | [S]ound horn > S

== Duke Hunt :: Hunting Ground ==
1. 06-04-2014 03:03:03 @ SIS
A Greenhorn Duke has escaped from my Trap of Least Resistance.



== Duke Hunt :: Sound my horn ==
06-04-2014 03:03:03 @ SIS
A Greenhorn Duke has escaped from my Trap of Least Resistance.

[R]eturn to main | [S]ound horn > s

== Duke Hunt :: Hunting Ground ==
1. 06-04-2014 03:03:10 @ SIS
A Whiny Duke has escaped from my Trap of Least Resistance.

2. 06-04-2014 03:03:03 @ SIS
A Greenhorn Duke has escaped from my Trap of Least Resistance.



== Duke Hunt :: Sound my horn ==
06-04-2014 03:03:10 @ SIS
A Whiny Duke has escaped from my Trap of Least Resistance.

[R]eturn to main | [S]ound horn >
```

If the user does not have any other inventory other than equipped trap, application does not allow user to change trap.



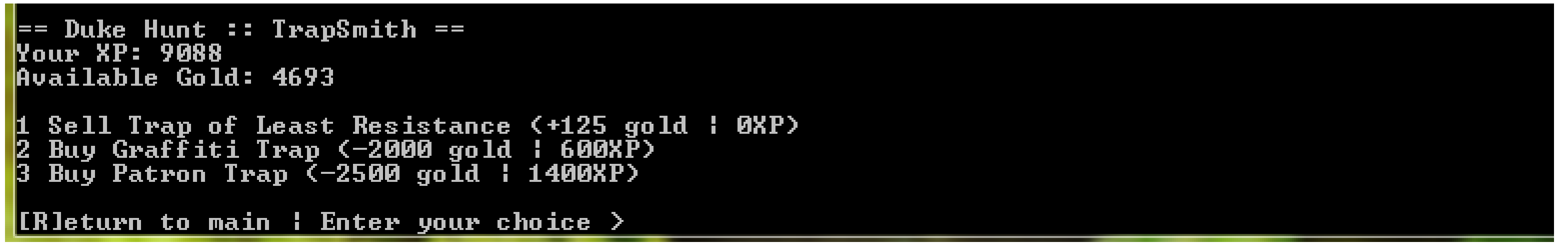
```
== Duke Hunt :: Main Menu ==
Welcome, Sophomore kevin!
Location: SIS
XP:    1,877
Gold:      1,363

1. Hunting Ground
2. My Inventory
3. The Trap Smith
4. Travel
5. Logout

Enter your choice > 2

== Duke Hunt :: Inventory ==
Current Trap: Trap of Least Resistance

You do not have other traps

[R]eturn to main >
```

After the user changes his trap to another, the unequipped trap is stored back into the user's inventory.

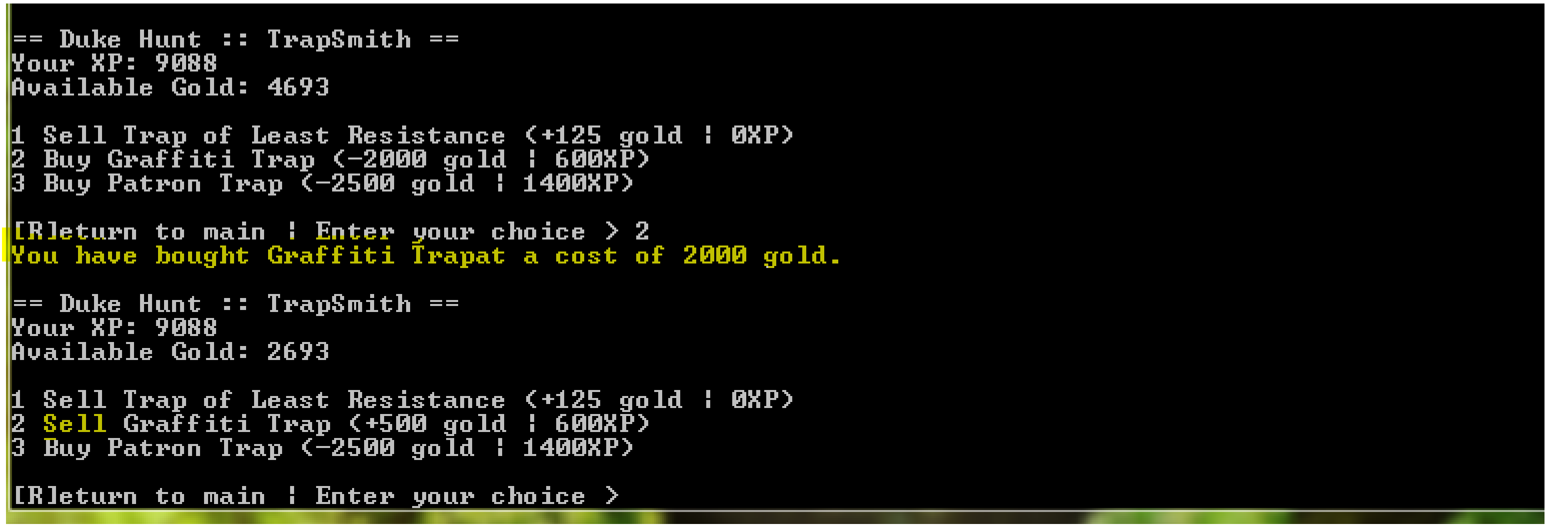


```
== Duke Hunt :: Inventory ==
Current Trap: Trap of Least Resistance

1.  Graffiti Trap

[R]eturn to main | Enter your choice > 1
Trap changed successfully!

== Duke Hunt :: Inventory ==
Current Trap: Graffiti Trap

1.  Trap of Least Resistance

[R]eturn to main | Enter your choice >
```

When the user buys and sells his trap. Status of the trap (if you can buy or sell) is updated.

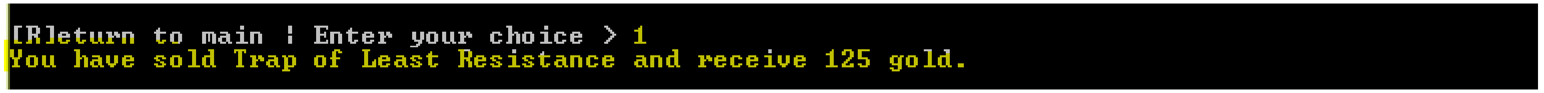


```
== Duke Hunt :: TrapSmith ==
Your XP: 9088
Available Gold: 4693

1 Sell Trap of Least Resistance (+125 gold | 0XP)
2 Buy Graffiti Trap (-2000 gold | 600XP)
3 Buy Patron Trap (-2500 gold | 1400XP)

[R]eturn to main | Enter your choice >
```

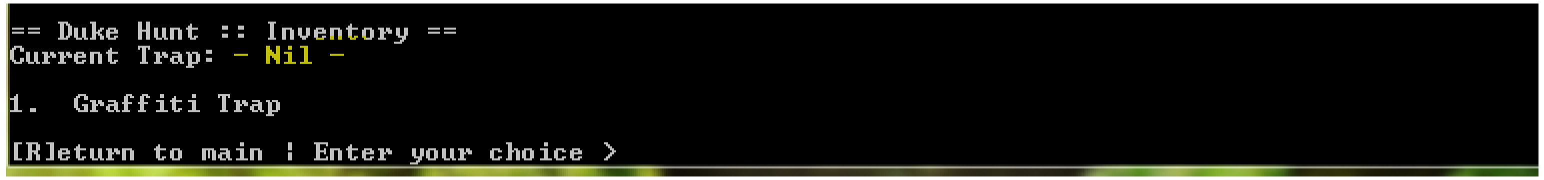


```
== Duke Hunt :: TrapSmith ==
Your XP: 9088
Available Gold: 4693

1 Sell Trap of Least Resistance (+125 gold | 0XP)
2 Buy Graffiti Trap (-2000 gold | 600XP)
3 Buy Patron Trap (-2500 gold | 1400XP)

[R]eturn to main | Enter your choice > 2
You have bought Graffiti Trapat a cost of 2000 gold.

== Duke Hunt :: TrapSmith ==
Your XP: 9088
Available Gold: 2693

1 Sell Trap of Least Resistance (+125 gold | 0XP)
2 Sell Graffiti Trap (+500 gold | 600XP)
3 Buy Patron Trap (-2500 gold | 1400XP)

[R]eturn to main | Enter your choice >
```

If user sells away current equip trap, "-Nil-"is shown.

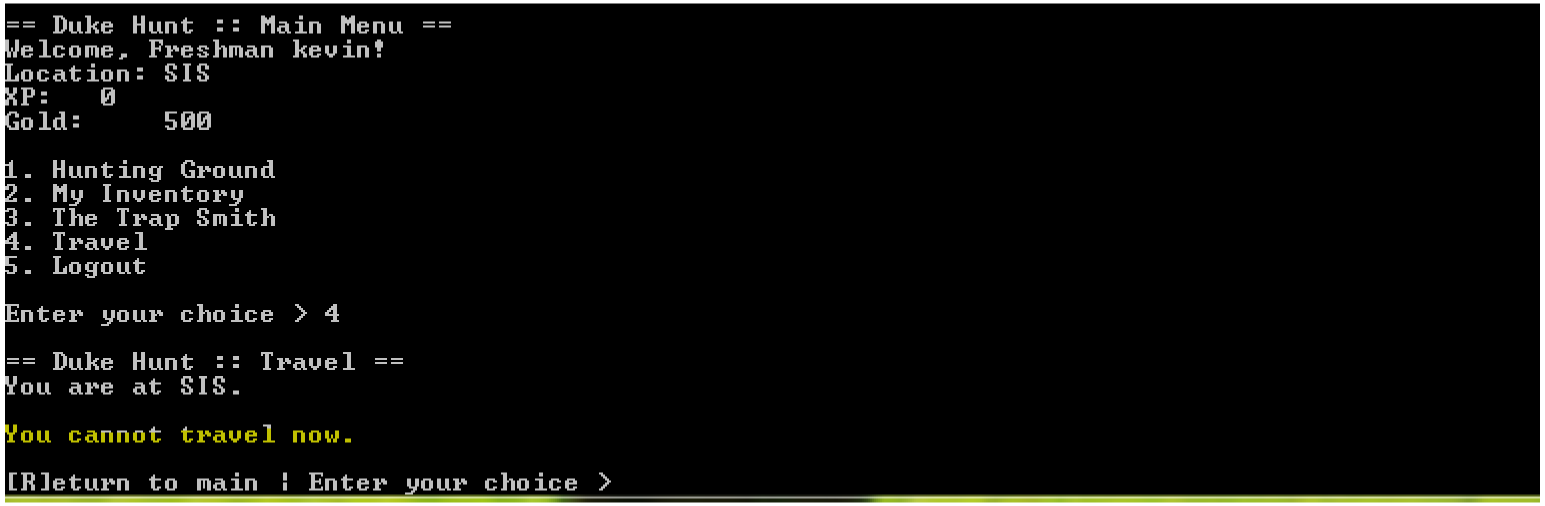


```
[R]eturn to main | Enter your choice > 1
You have sold Trap of Least Resistance and receive 125 gold.
```



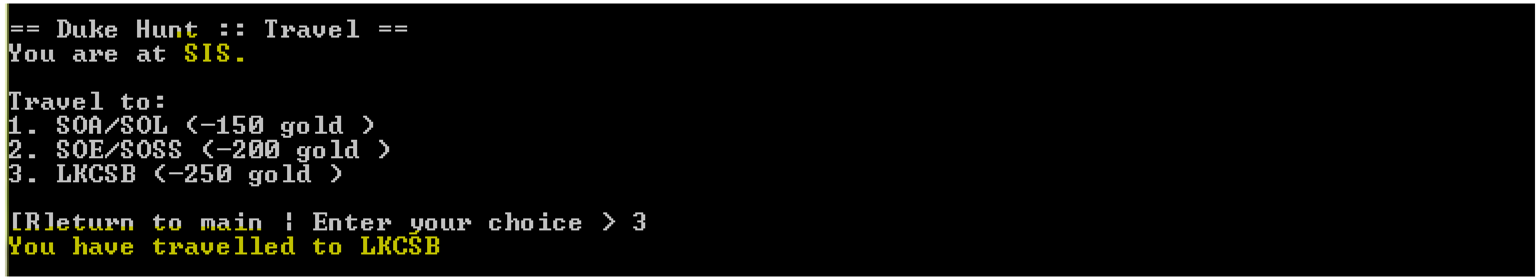
```
== Duke Hunt :: Inventory ==
Current Trap: - Nil -

1.  Graffiti Trap

[R]eturn to main | Enter your choice >
```

When the user does not have any XP, the application do not allow the user to travel.

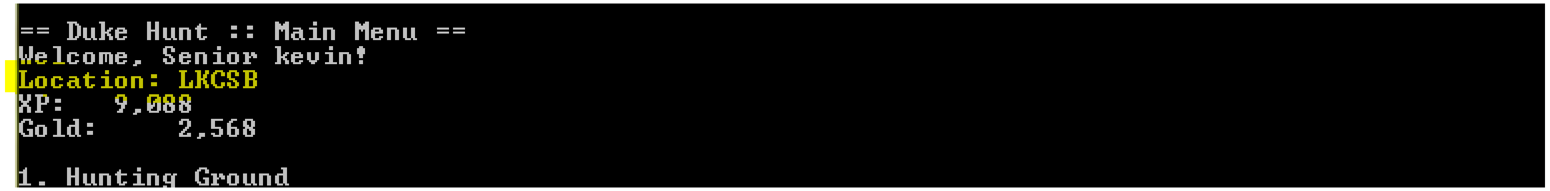


```
== Duke Hunt :: Main Menu ==
Welcome, Freshman kevin!
Location: SIS
XP:    0
Gold:      500

1. Hunting Ground
2. My Inventory
3. The Trap Smith
4. Travel
5. Logout

Enter your choice > 4

== Duke Hunt :: Travel ==
You are at SIS.

You cannot travel now.

[R]eturn to main | Enter your choice >
```

When user travels to the region, his current location is updated.



```
== Duke Hunt :: Travel ==
You are at SIS.

Travel to:
1. SOA/SOL (-150 gold )
2. SOE/SOSS (-200 gold )
3. LKCSB (-250 gold )

[R]eturn to main | Enter your choice > 3
You have travelled to LKCSB
```



```
== Duke Hunt :: Main Menu ==
Welcome, Senior kevin!
Location: LKCSB
XP:    9,088
Gold:      2,568

1. Hunting Ground
```

9. Miscellaneous

Yazhi

"In this OOAD project, firstly, for academic reflection, I learn a lot about how to do a complete project, from the designing to coding, from reading the file to building the user case application. At the same time, because different people may have different ideas about the project, I also learn about reducing dependency on others' work. We need to set realistic goals for our own work. For team work reflection, I learn about how to do a project as a team, from separating the work to helping each other to fix the bugs inside codes. I also learn about perseverance, especially for debugging."

Xin Ning

"I have learnt a lot from this project such as teamwork and time management. The project has taught me how to design a good application and several important designing concepts that is useful as I progress into Software Engineering. Through this project, I have learnt how to work with different people and to adapt to different working styles that each of us has. It was difficult at first since do not know each other well at that time. Overall, I have enjoyed the project as well as my team mates. "

Charlie

"It is a fruitful OOAD journey; I've managed to improve on my critical thinking skills throughout the application creation process while our group have to work together to come up with the optimal solution for every process. I also appreciate the hands-on experience, to enforce object orientation (OO) throughout the application creation process. OO plays a huge role in making the application much more readable, flexible and even reduce the potential workload required to change the project in the future while vastly increases the scalability of the project. Through this project, I've also improved on my interpersonal skills as I have to communicate with my team to make sure everyone is on the same note to ensure consistency throughout application creation while maintaining a balance workload."