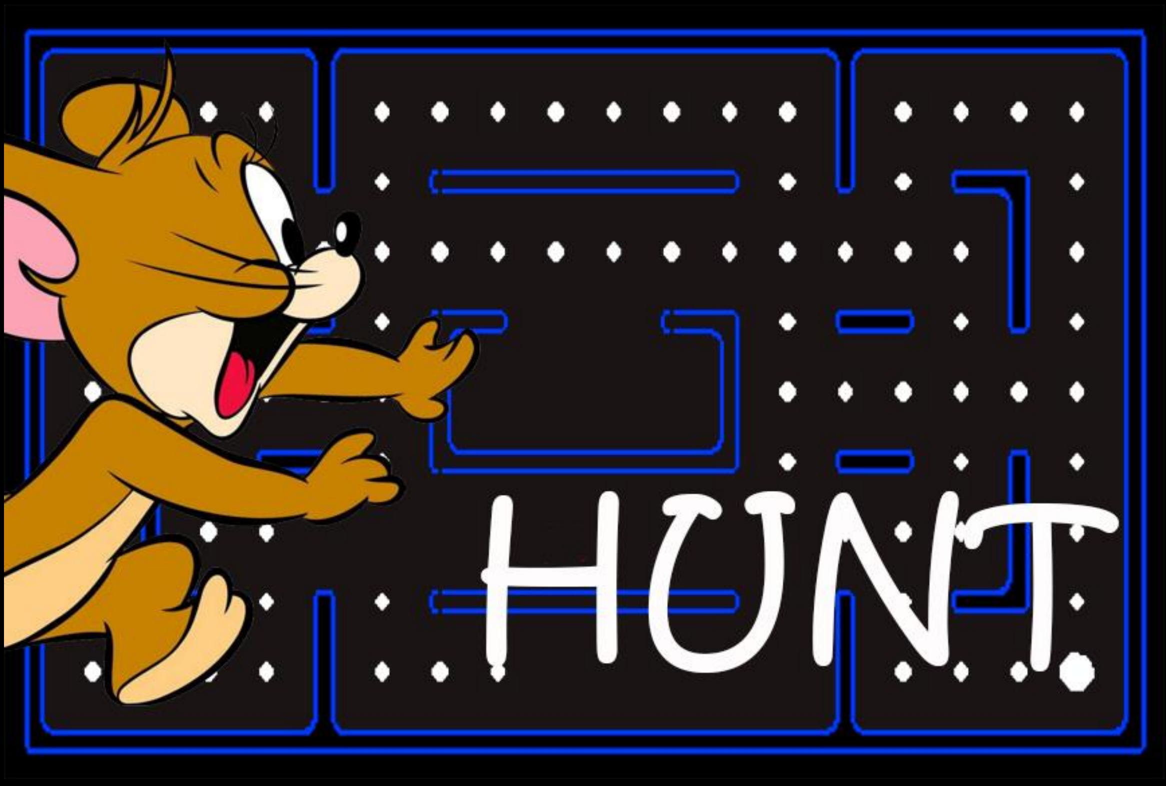
PROJECT DUKEHUNT

G2-T05

IS201 – OBJECT ORIENTED APPLICATION DESIGN

(2013-2014, Term 2)



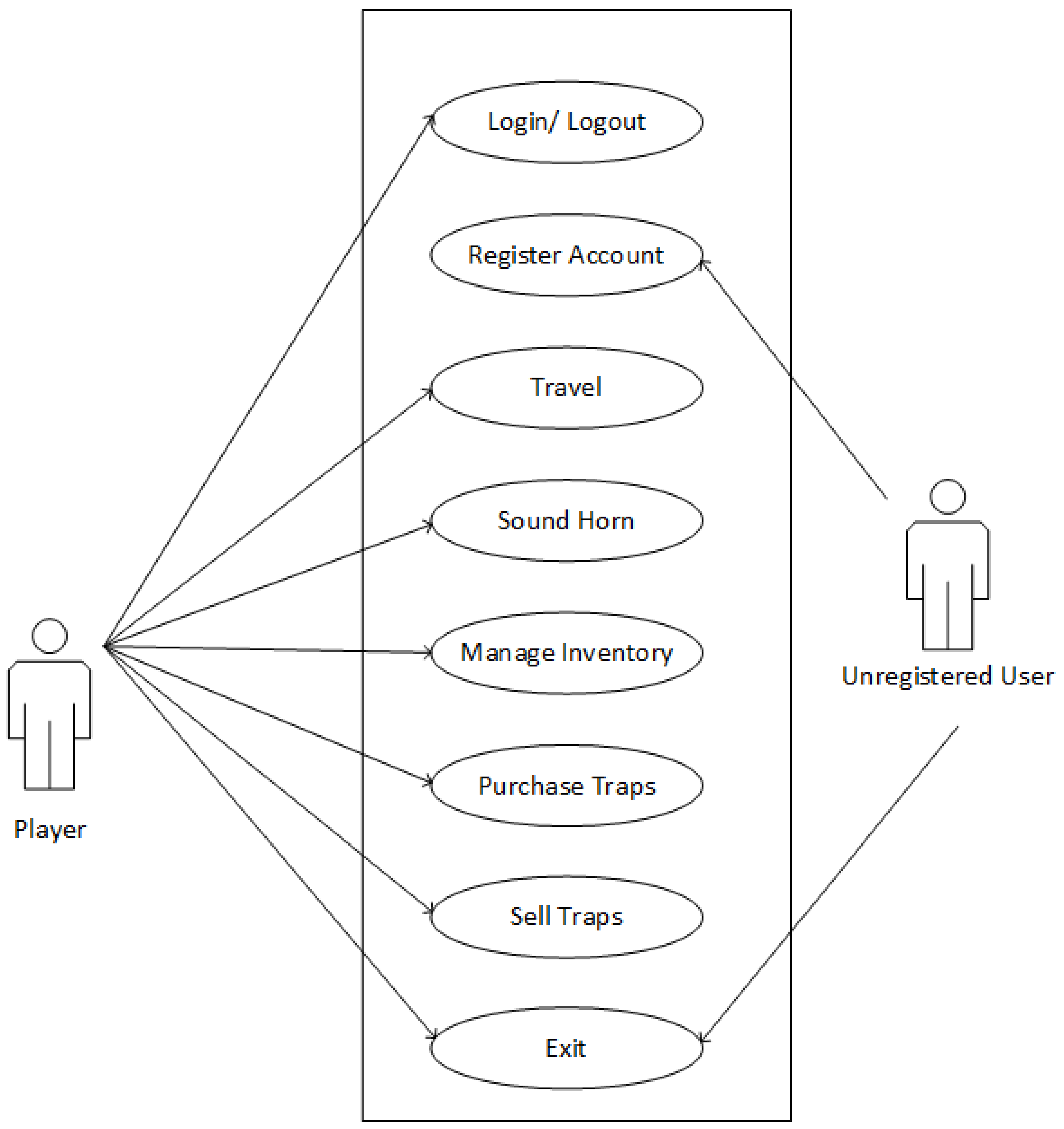
Edison LIM Jun Hao | Cassandra THAI Jia Ying | CHIANG Ling Yi

IS 201 2013/2014 TERM 2 G2 - PROF DAVID LO & MS. JOELLE

Contents

1. Use Case Diagram

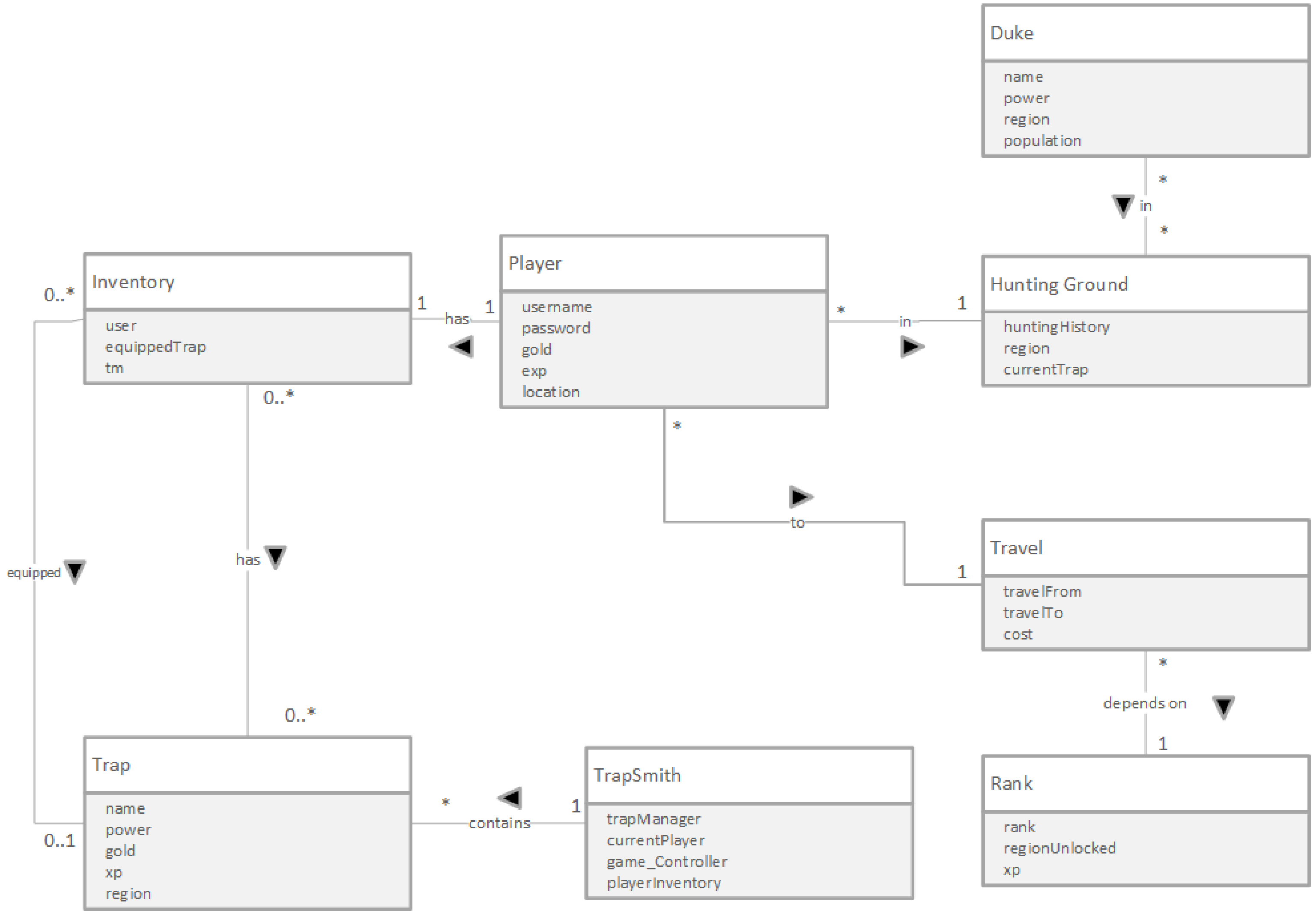
1.1. <u>Simple Use Case Diagram</u>



1.2. Brief Description of the Use-Case Diagram

No.	Function	Description
1	**Login/ Logout**	Authentication of account. This function will allow the player to keep track of his/ her game progress and keep it safe with a password so others cannot access it
2	**Register Account**	Creation of a new player object. To play the DukeHunt Game, users need to have an account to play.
3	**Travel**	This function allows the player to move from one region to another region. This function also tracks the cost to travel.
4	**Sound Horn**	This function is to hunt for duke. When horn is sounded, system will run the game and tell the user of its outcomes
5	**Manage Traps**	This function allow user to change the currently equipped trap or view the traps that he has.
6	**Purchase Traps**	This function allows user to buy traps from the trapsmith.
7	**Exit**	This function allows the user to exit the programme.

2. Domain Diagram



3. Use Case Specifications

3.1. Log In

Actor

Player

Pre-Condition

The player must be a registered user and have not logged in.

Main Flow of events

1. This user case begins when the player selects the log in function on the Login Menu.
2. The system displays the request for the player to key in his/her username and password.
3. The user keys in his/her username and password
4. The system verifies that the username exists and that the password keyed in is correct. Upon successful authentication, the system will display the Game Menu.

Alternate flow of events

4a. Username is not valid

1) System will prompt for user to re-enter username
2) Player enter new username
3) System will check if username is valid. Step 1 to 2 will be repeated till the player gets a correct username and password.
4) This will return to the main flow of events

4a. Password is wrong

1) System will indicate that either the username or password is wrong and prompt for user to re-enter his/her username and password.
2) Player enters new username and password
3) System will check if username and corresponding password is correct and. Step 1 to 2 will be repeated till the player gets a correct username and password.
4) This will return to the main flow of events

3.2. Log Out

Actor

Player

Pre-Condition

The player must be a registered user and have logged in.

Main Flow of events

1. This user case begins when the player selects the log out function on the Game Menu after logging in.
2. The system displays the main menu.

3.3. <u>Register Account</u>

<u>Actor</u>

Player

<u>Pre-Condition</u>

Player has already selected register user function on the login menu.

<u>Main Flow of events</u>

1. The use case begins with the user selecting the register user function on the login menu
2. System display the screen prompting for username, password and confirm password
3. Player will input the similar fields of their choice
4. System will check if username has been registered and if password and confirm password is the same. If not, the new user account and new inventory with the standard trap for the user will be created and the main menu is displayed. System stores the record on the csv file.
5. This use case ends

<u>Alternate flow of events</u>

4a. Username has been registered

5) System will prompt for user to re-enter username
6) Player enter new username
7) System will check if username is not used. Step 1 to 2 will be repeated till the player gets a username unused.
8) This will return to the main flow of events

4b. Password and confirm password is different

1) System will prompt for user to re input password and confirm password
2) Player will re-enter password and confirm password
3) System will check if the password and confirm password is similar
4) This will return to the main flow of events

3.4. Travel

Actor

Player

Pre-Condition

The player must be a registered user and have logged in.

Main Flow of events

1. This user case begins when the player selects the Travel function on the Game Menu after logging in.
2. The system displays the current location the player is at, followed by the list of regions the player is able to travel to.
3. Player may choose to change his/her current location by selecting the option displayed the list or go back to main menu
4. System replaces the current location to the new location to the selected choice of location, deducts the necessary gold. The system will then run to step 2 again.
5. Player can choose to change his/her location again and steps 3 to 4 will be repeated.
6. System brings the user to the main menu and the case ends.

Alternate flow of events

4a. Player does not have enough gold

9) System will indicate that the player do not have enough gold and is unable to travel and runs step 2.

3.5. Sound Horn

Actor

Player

Pre-Condition

Player has already selected the Hunting Ground function on Game menu

Main Flow of events

1. The use case begins with the user selecting the hunting ground function on the game menu
2. System loads the Hunting Ground menu, along with 10 history of the user
3. Player will type S to sound horn
4. System will run the function and return him with the message of success. Message: I caught a <duke's Name> using <current equipped trap name> and gained <amount of gold> and <amount of exp>.
5. System will prompt user for next action. If player presses S, it will go back to step 4. If player presses 'R', it will return to game menu.

Alternate flow of events

2a. User is a new user and no hunting history will be shown

3a. If player enters an input that is R, it will return to game menu.

3b. If player enters an input that is not 'R' or 'S', system will display an invalid input and return to step 3.

4a. Hunting is not successful and a failure message will be returned. Message : A <Duke's name> escaped from my <Trap's name>.

3.6. Manage Inventory

Actor

Player

Pre-Condition

The player must be a registered user and have logged in.

Main Flow of events

1. This user case begins when the player selects the Inventory function on the Game Menu after logging in.
2. The system displays the current trap, followed by the list of traps that the user have in his inventory
3. Player choose to change his trap by selecting the option displayed the list or go back to main menu
4. System replaces the trap to the new trap if the number option is input by user and puts the equipped trap back into the list and runs step 2 again.
5. Player can choose to change its trap again and steps 3 to 4 will be repeated.

Alternate Flow of Events

3a. If player chooses an invalid option, system will return an error message and return to step 3 to prompt user for input.

3.7. Purchase Trap

Actor

Player

Pre-Condition

Player has already selected the Trapsmith function on Game Menu

Main Flow of events

1. The use case begins with the user selecting the Trapsmith function on the game menu.
2. System loads the current equipped trap of the user before displaying the list of traps available for purchase by the region that the user is in.
3. Player will need to input the choice of trap that he/she wants to purchase.
4. System will process the trap and add to the player's inventory.
5. 3 and 4 loops until user press E to exit

Alternate flow of events

4a. If the player do not have enough gold or experience points, the player is unable to purchase the trap.

1) The system will display a message to indicate that the player is unable to purchase the trap and returns to step 2.

3.8. <u>Sell Trap</u>

<u>Actor</u>

Player

<u>Pre-Condition</u>

Player has already selected the Trapsmith function on Game Menu

<u>Main Flow of events</u>

1. The use case begins with the user selecting the Trapsmith function on the game menu.
2. System loads the current equipped trap of the user before displaying the list of traps available for sale by the region that the user is in.
3. Player will need to input the choice of trap that he/she wants to sell.
4. System will process the trap, remove it from the player's inventory and add gold to the player.
5. Steps 3 and 4 loops until user press E to exit.

<u>Alternate flow of events</u>

3a. If the use do not have any traps that could be sold, there won't be any sell option available.

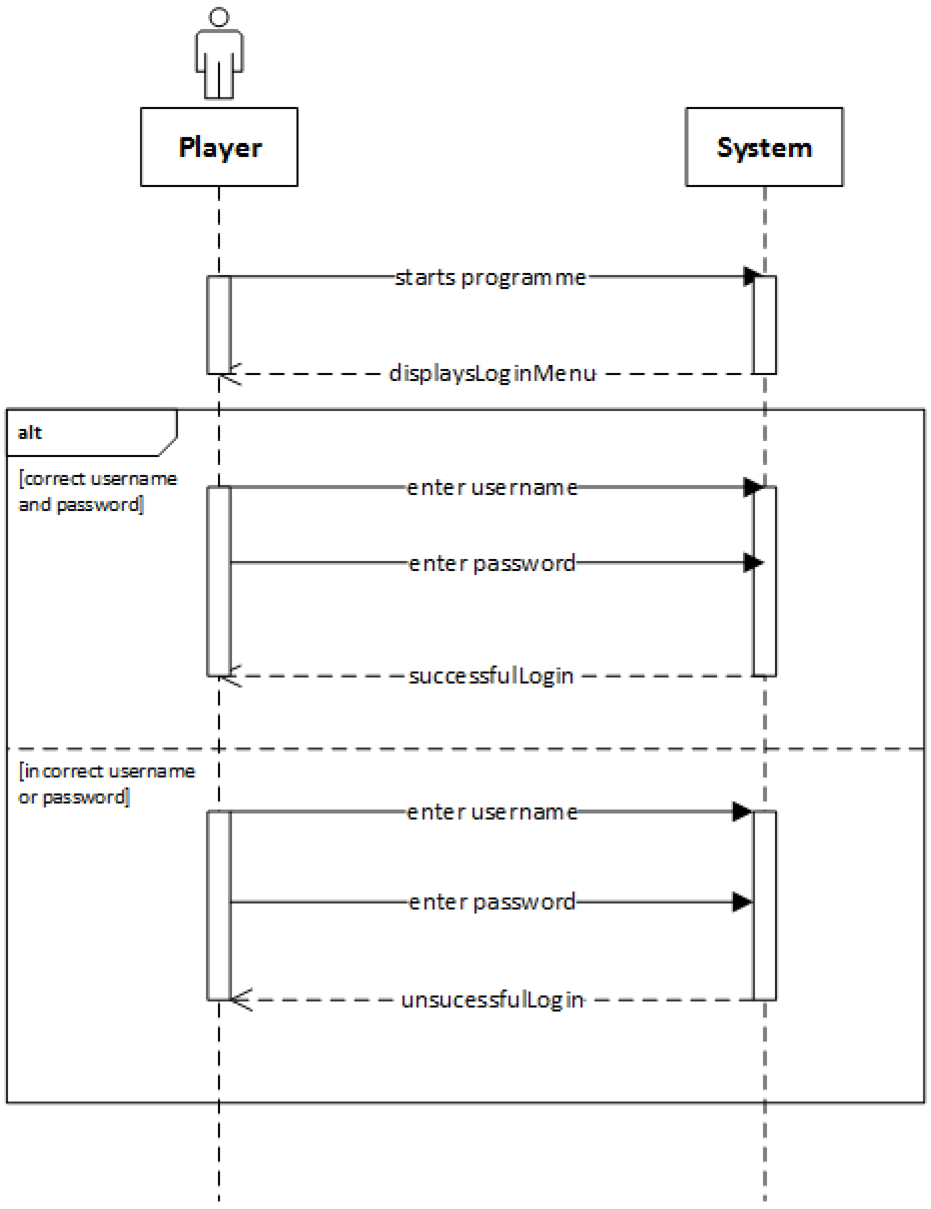
4a. If there is no equipped trap and the inventory only contains one trap, the user is unable to sell his/her last trap.

2) The system will display a message to indicate that the player is unable to sell the last trap and returns to step 2.

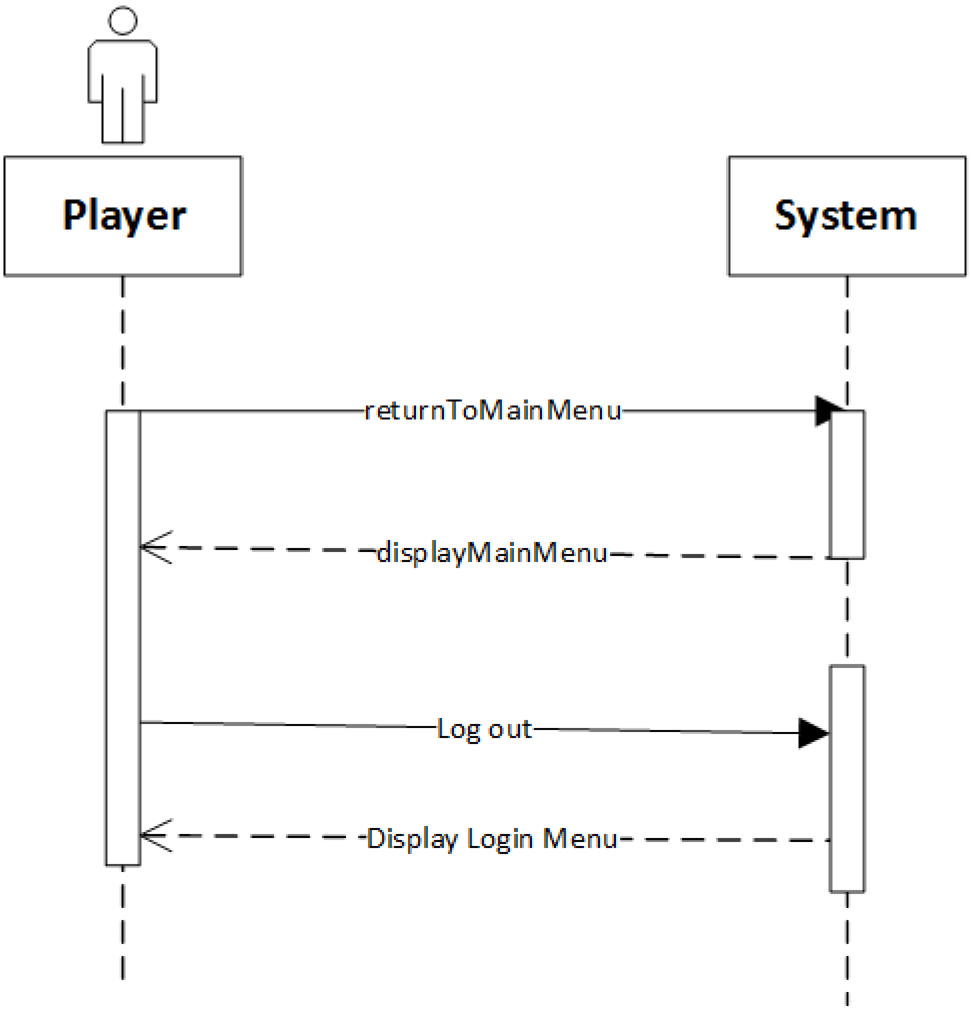
4. System Sequence Diagrams

Only Post-Conditions are shown for Operation Contracts.

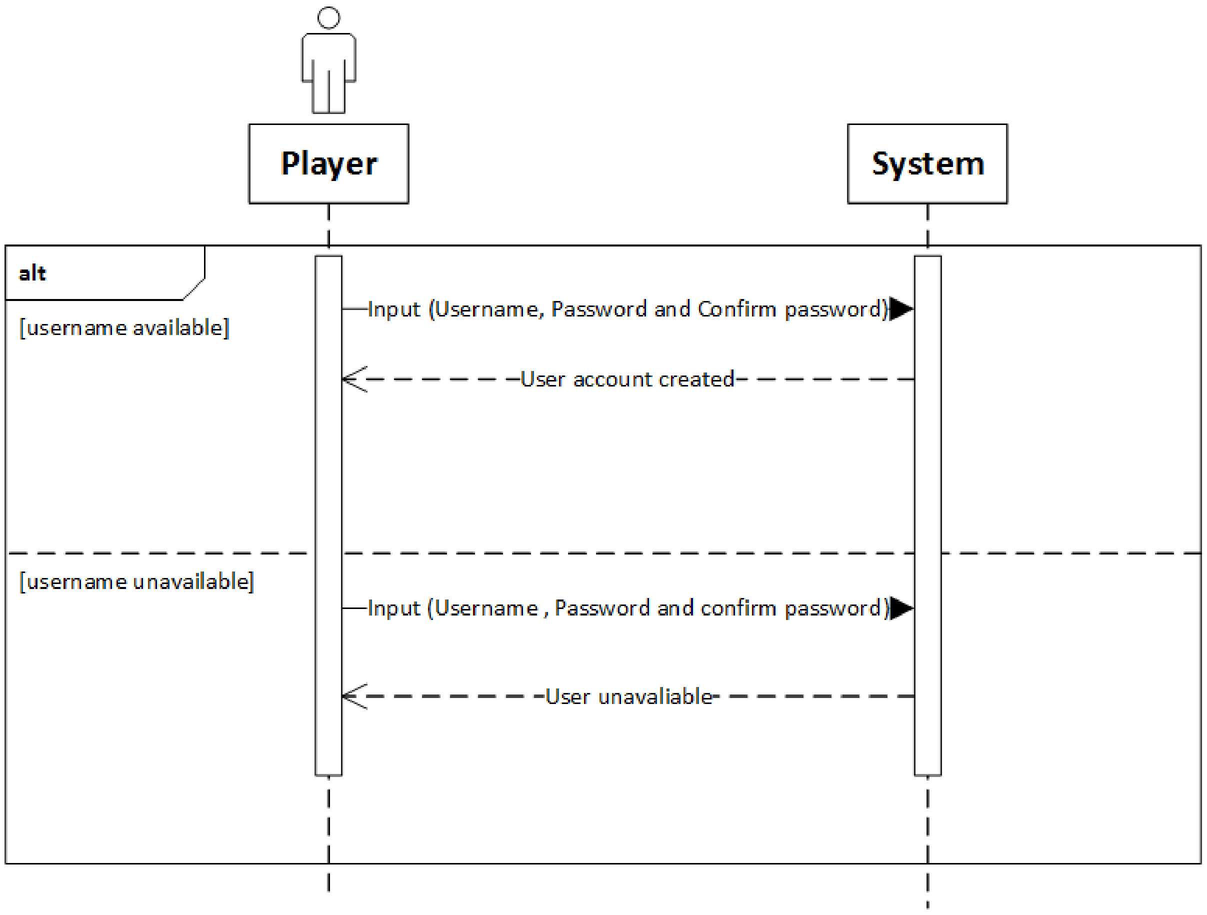
4.1. Login



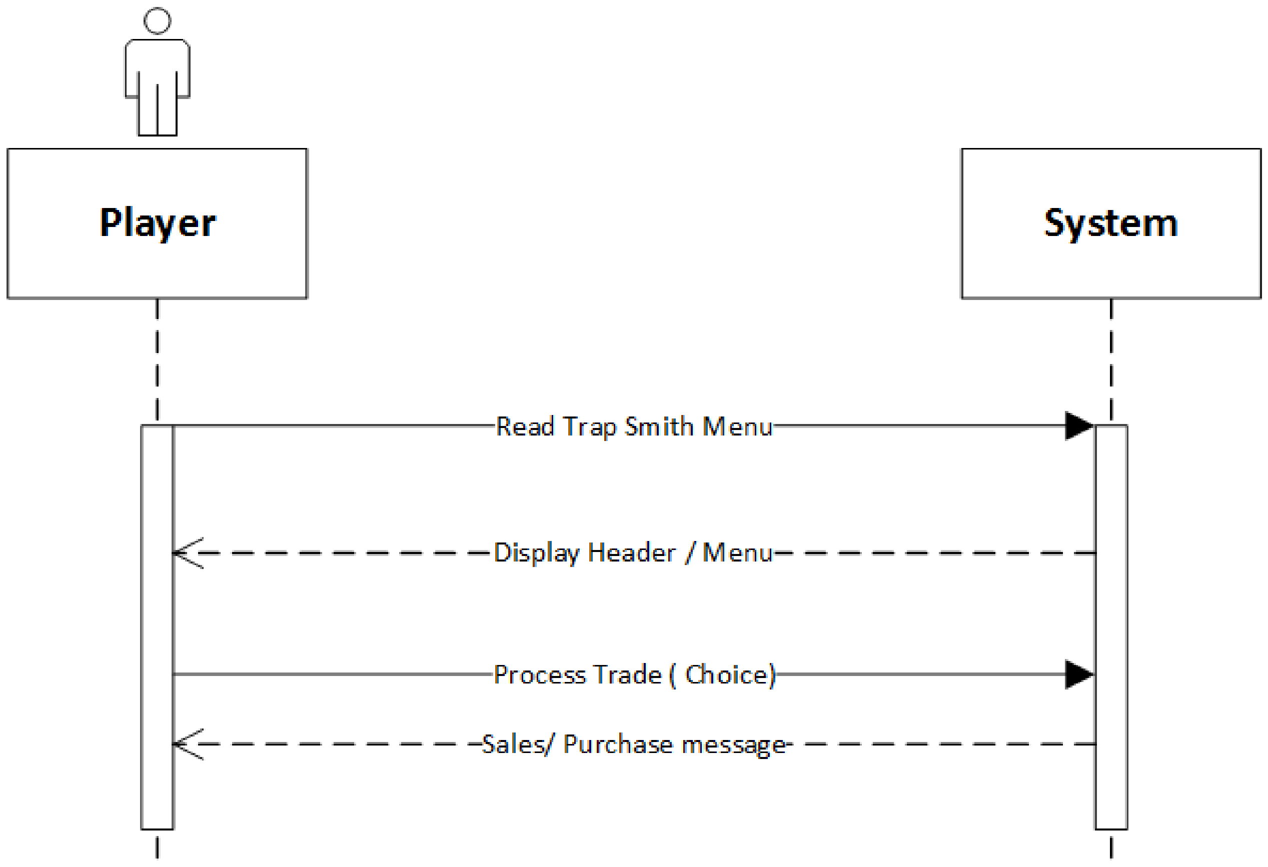
4.2. Log Out


Player
System
returnToMainMenu
displayMainMenu
Log out
Display Login Menu

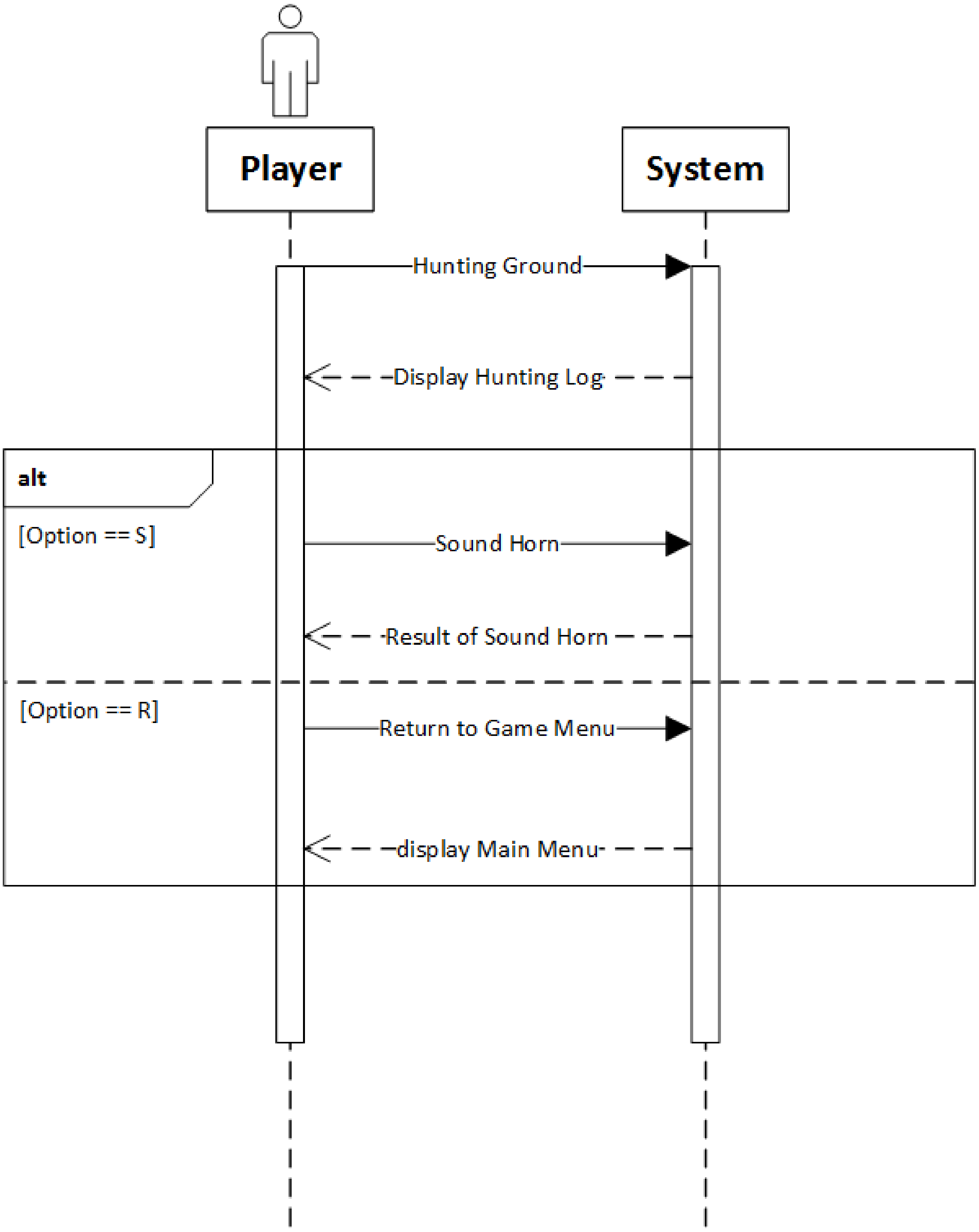
4.3. Register Account


Player
System
alt
[username available]
Input (Username, Password and Confirm password)
User account created
[username unavailable]
Input (Username , Password and confirm password)
User unavaliable

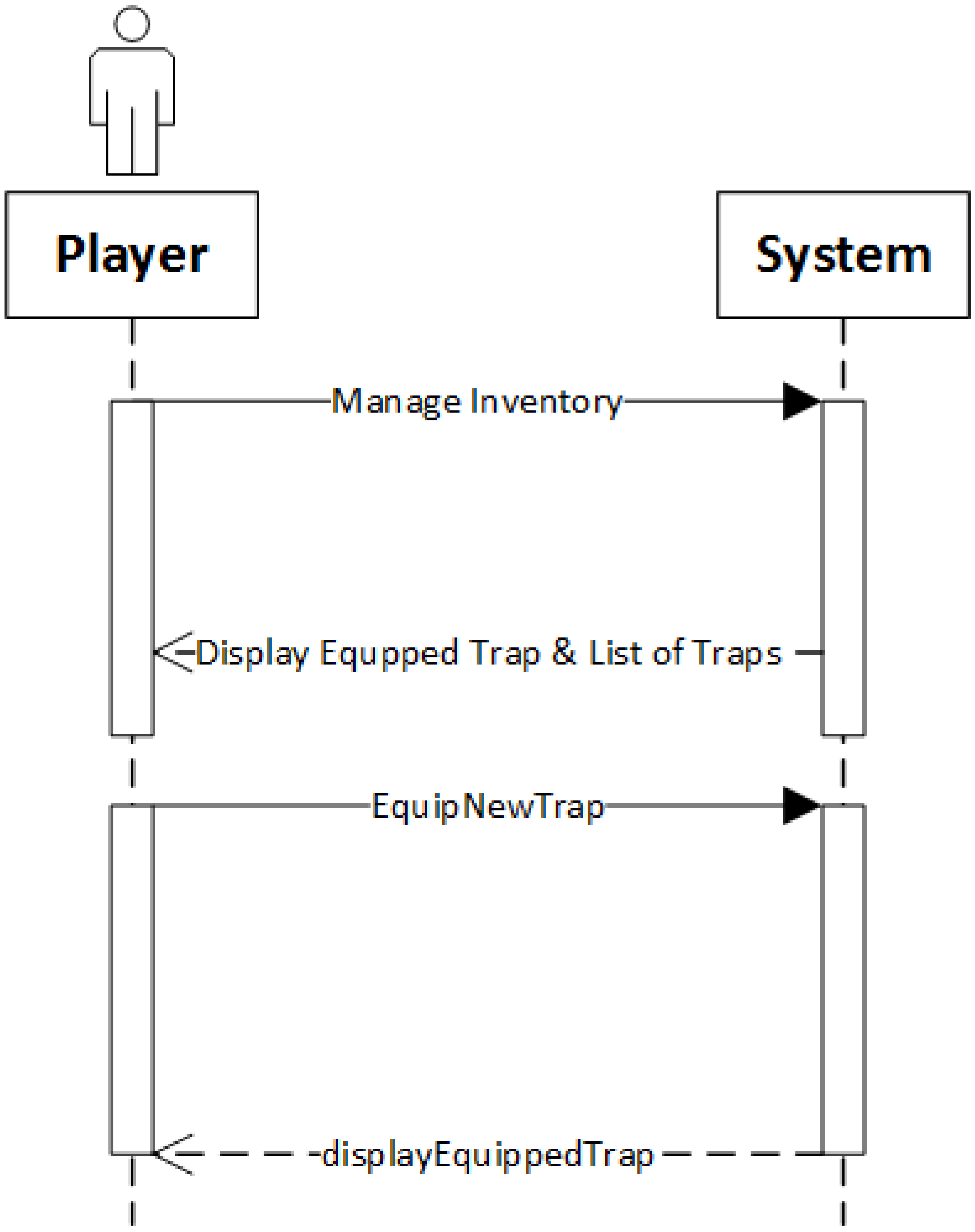
4.4. Travel


Player
System
Read Trap Smith Menu
Display Header / Menu
Process Trade (Choice)
Sales/ Purchase message

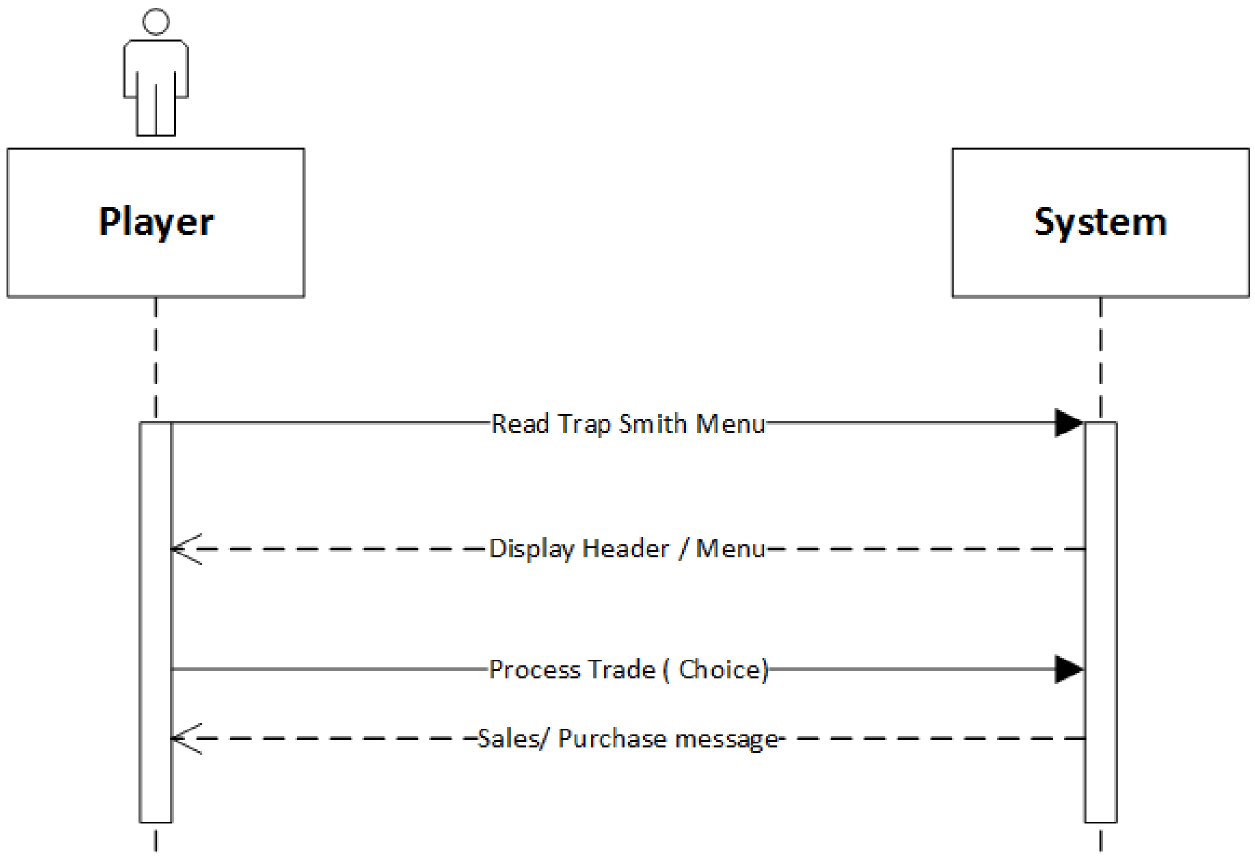
4.5. Sound Horn



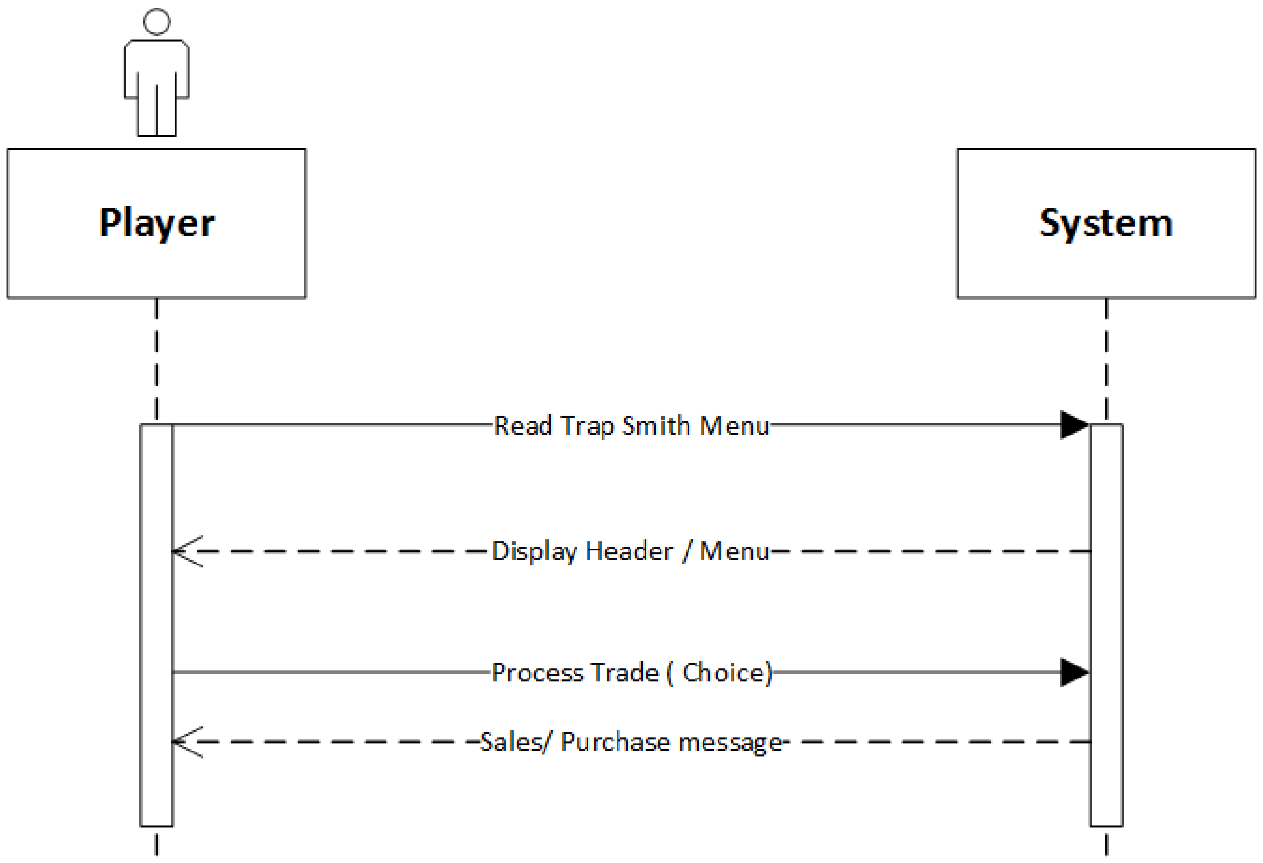
4.6. Manage Inventory



4.7. Purchase Trap

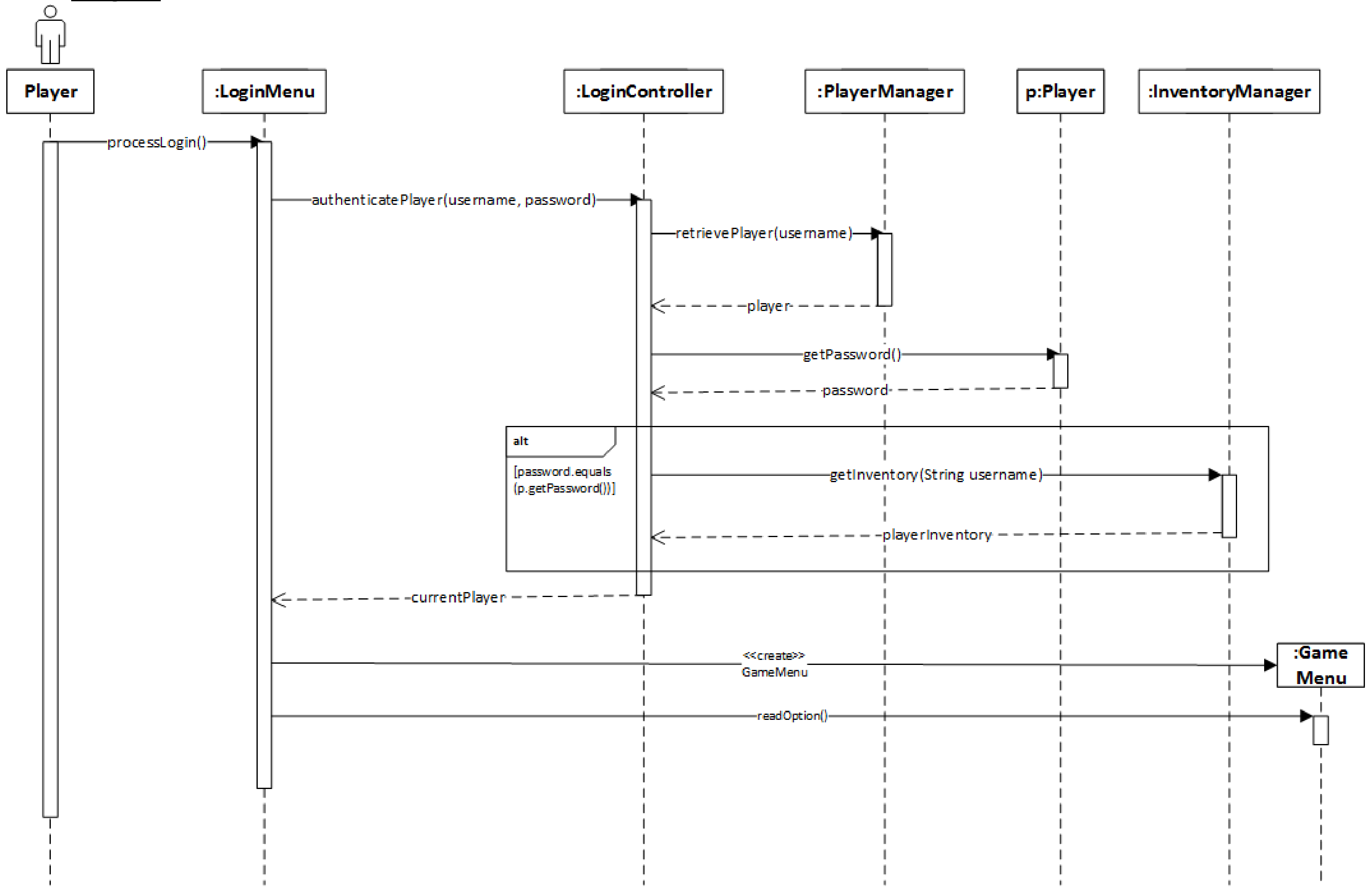


4.8. Sell Trap

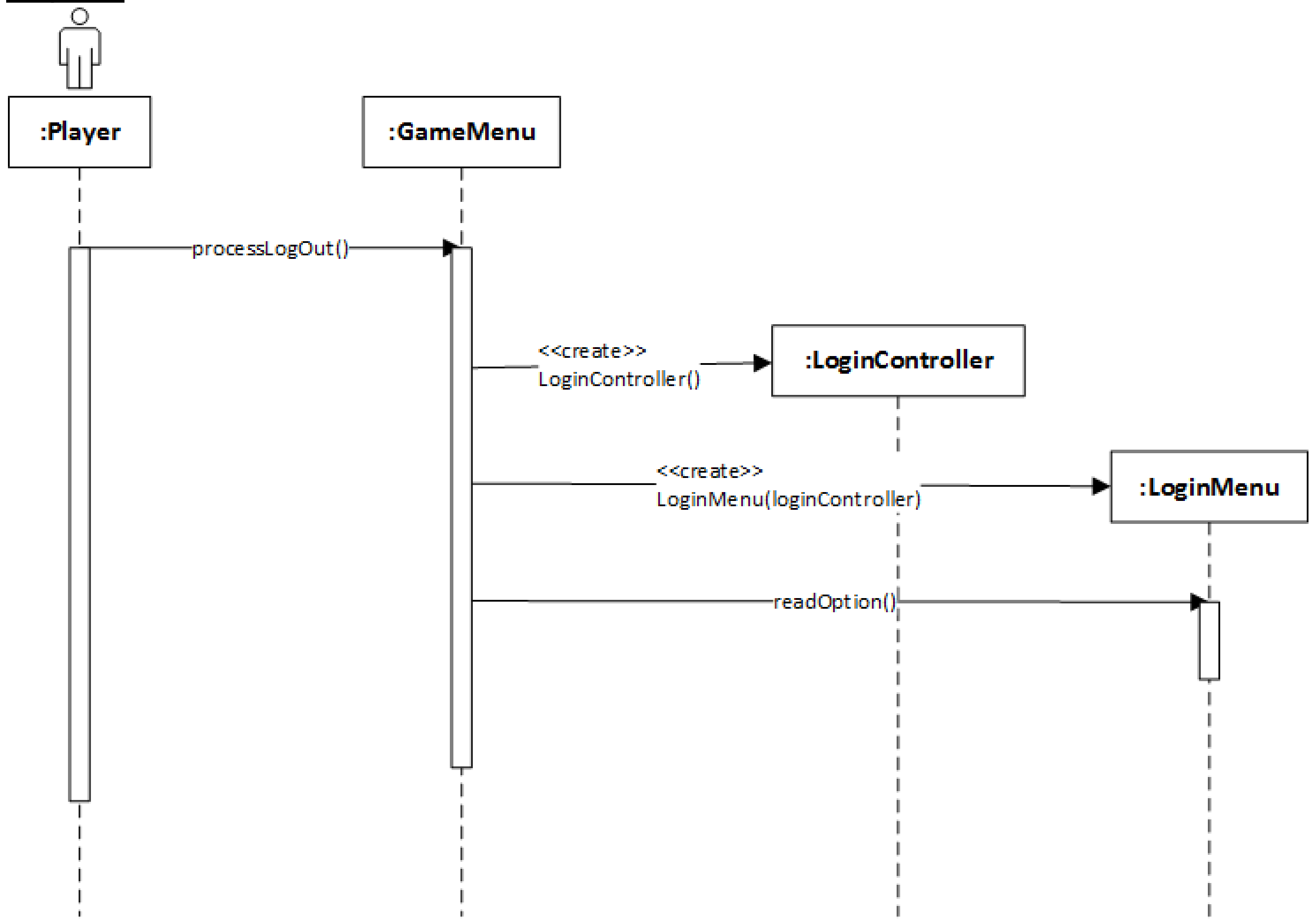

Player
System
Read Trap Smith Menu
Display Header / Menu
Process Trade (Choice)
Sales/ Purchase message

5. Sequence Diagrams

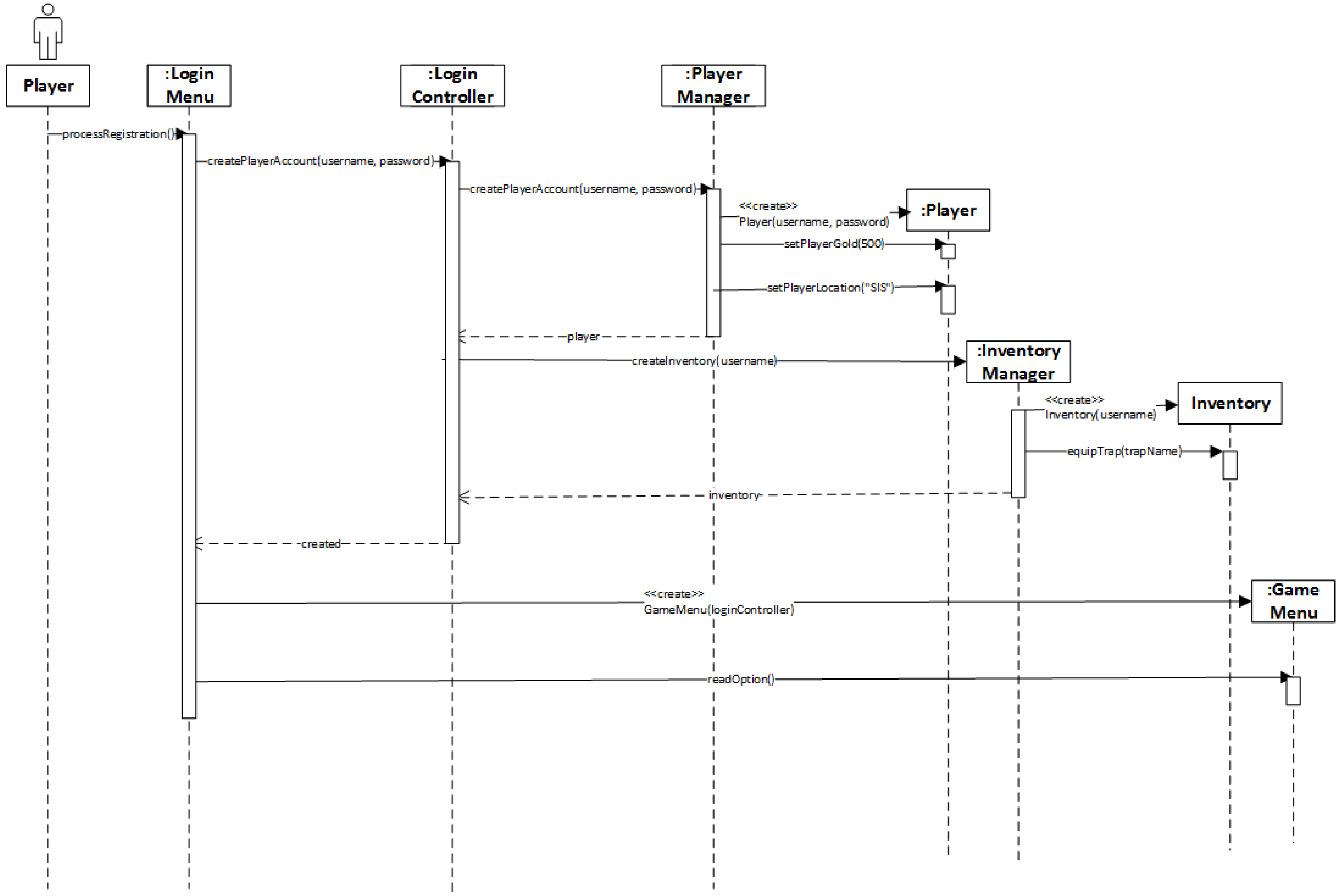
5.1. Log In



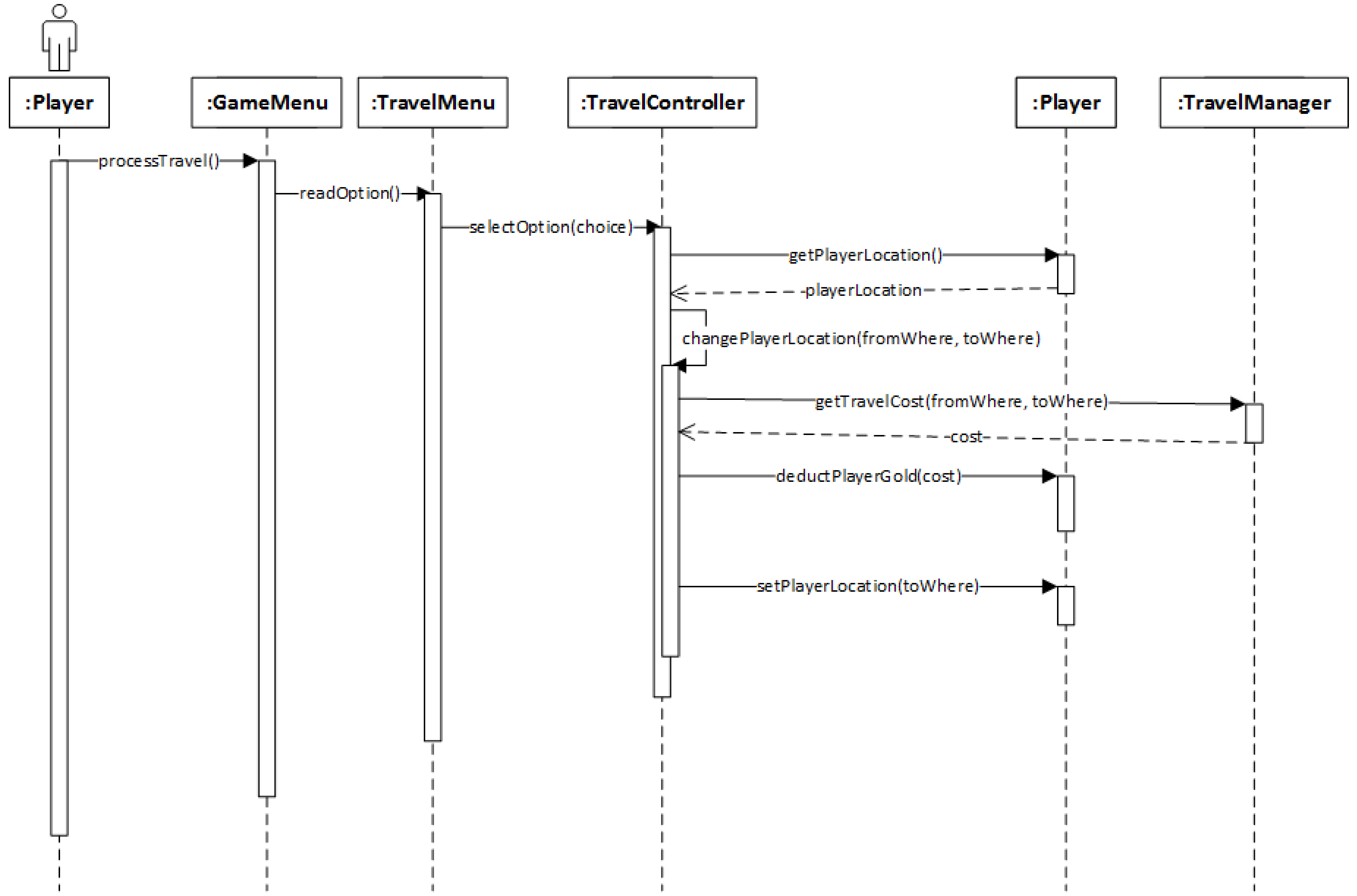
5.2. Log Out


:Player
:GameMenu
processLogOut()
<<create>>
LoginController()
:LoginController
<<create>>
LoginMenu(loginController)
:LoginMenu
readOption()

5.3. Register Account

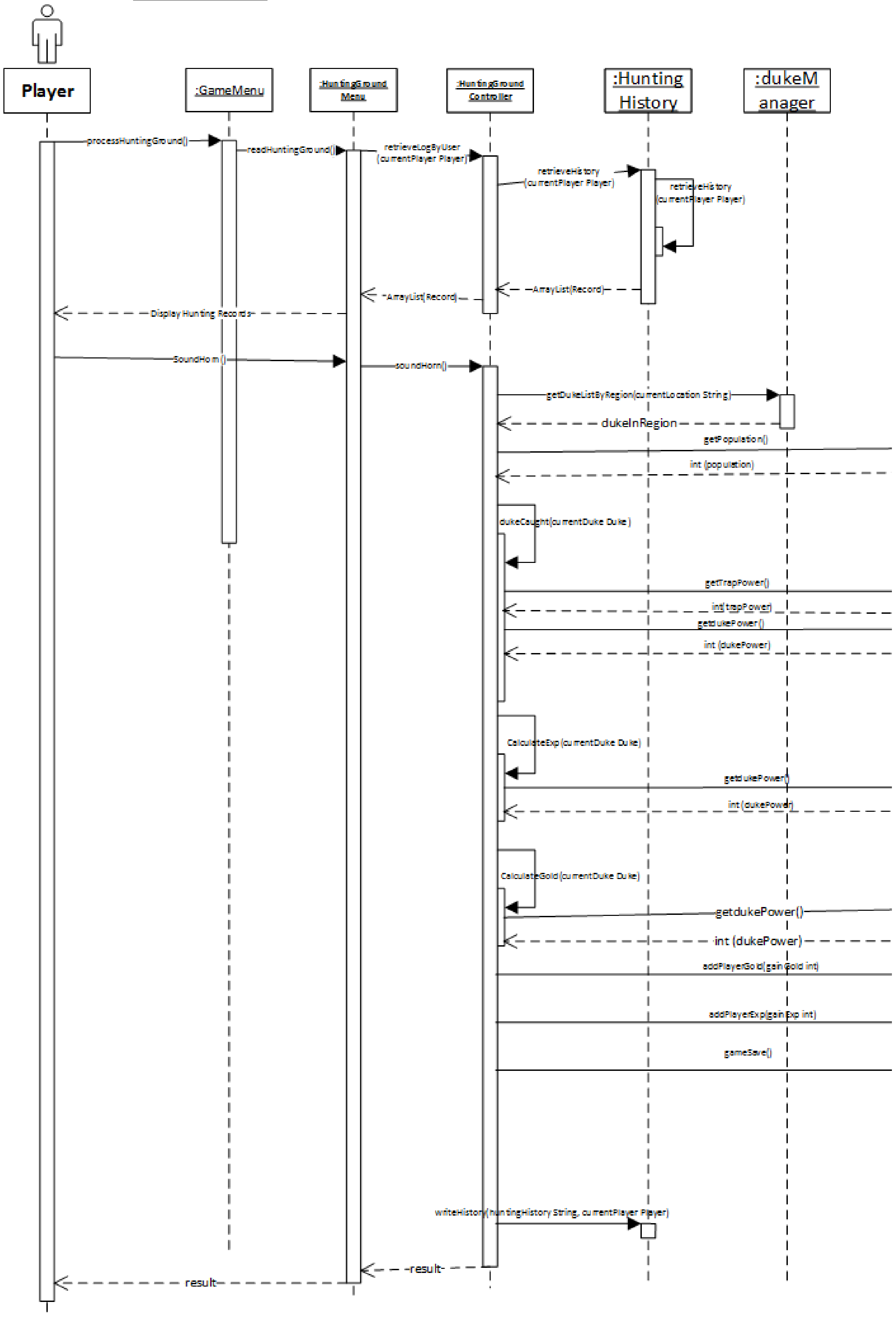


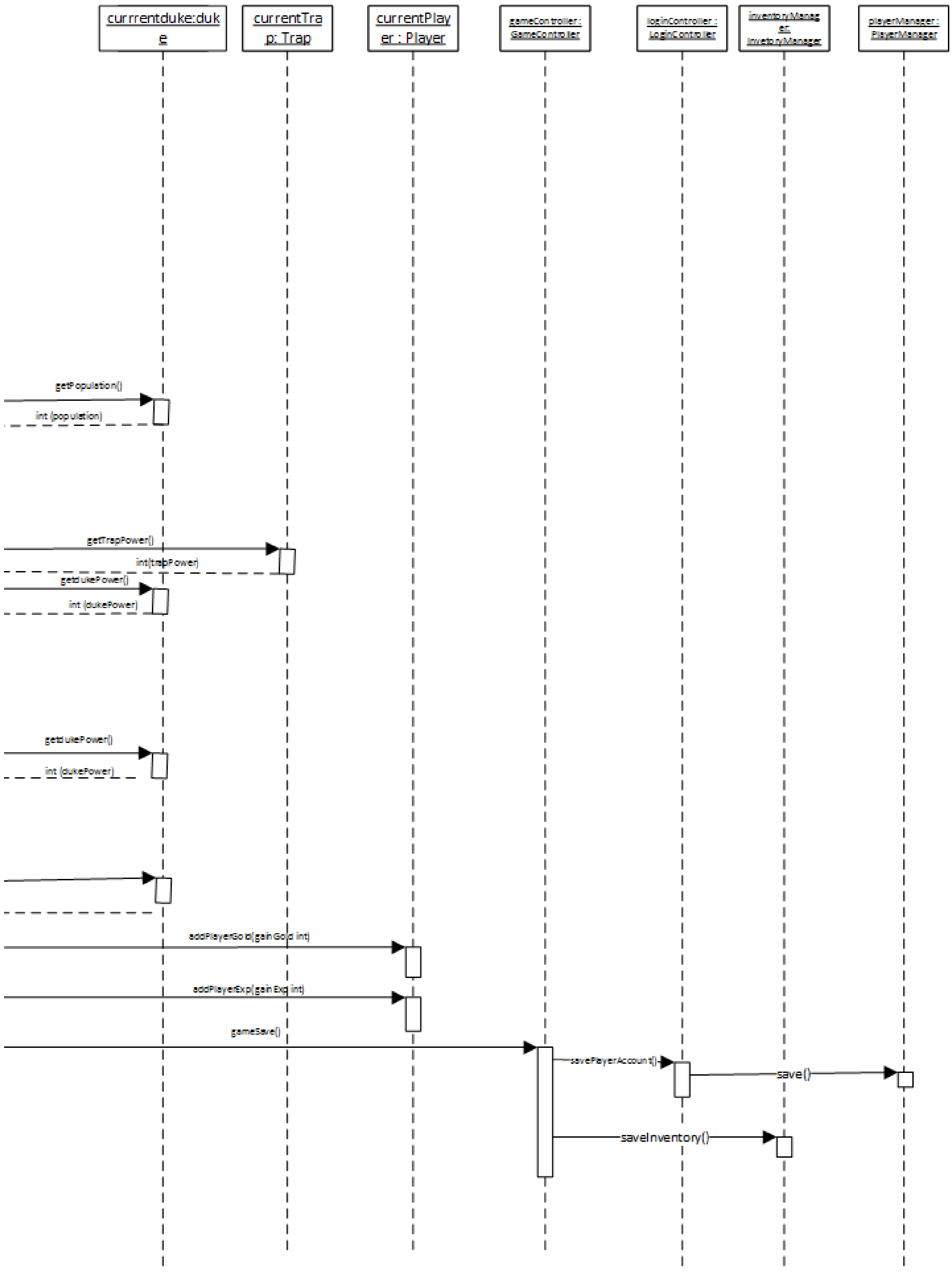
5.4. Travel



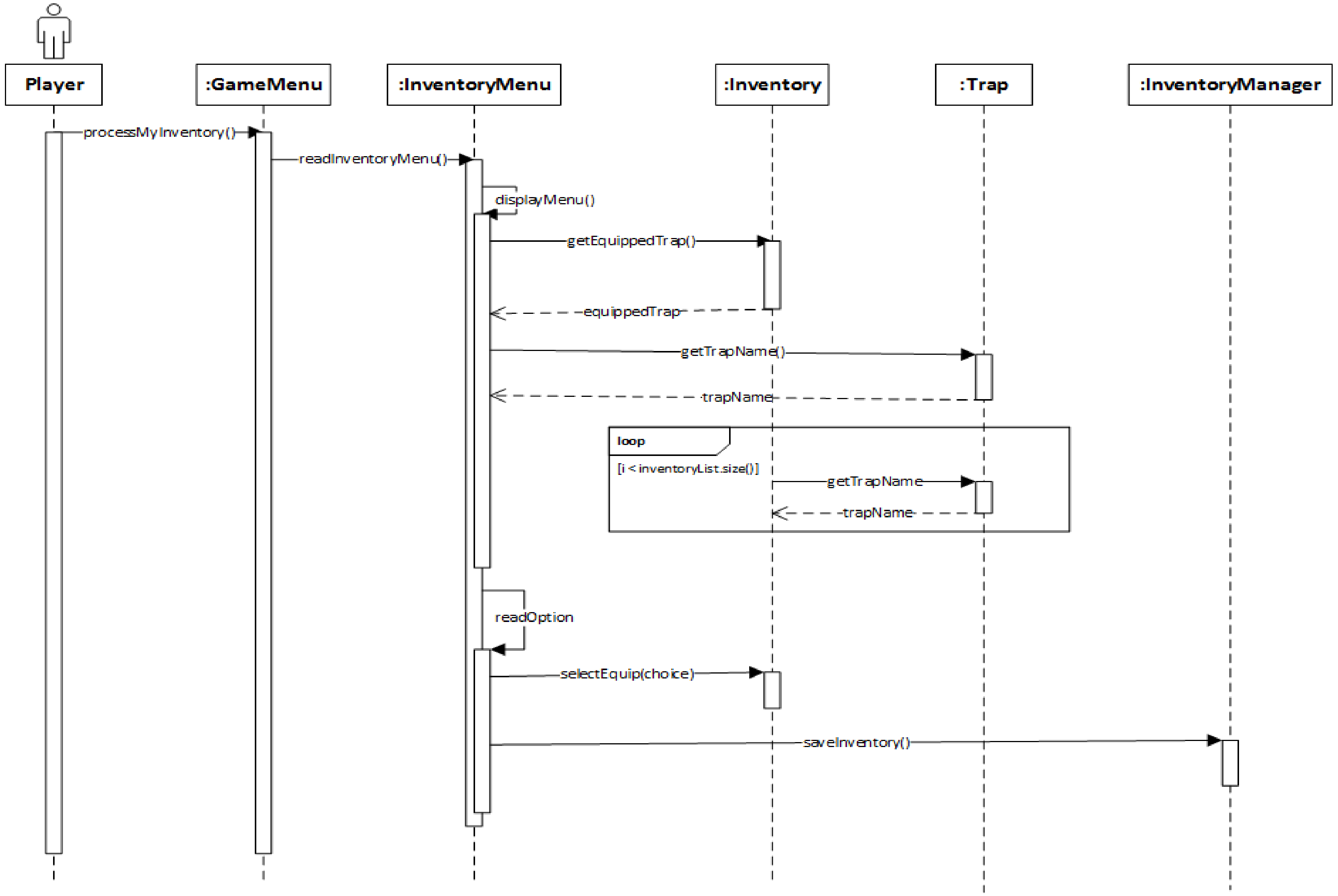
:Player
:GameMenu
:TravelMenu
:TravelController
:Player
:TravelManager
processTravel()
readOption()
selectOption(choice)
getPlayerLocation()
playerLocation
changePlayerLocation(fromWhere, toWhere)
getTravelCost(fromWhere, toWhere)
cost
deductPlayerGold(cost)
setPlayerLocation(toWhere)

5.5. Sound Horn

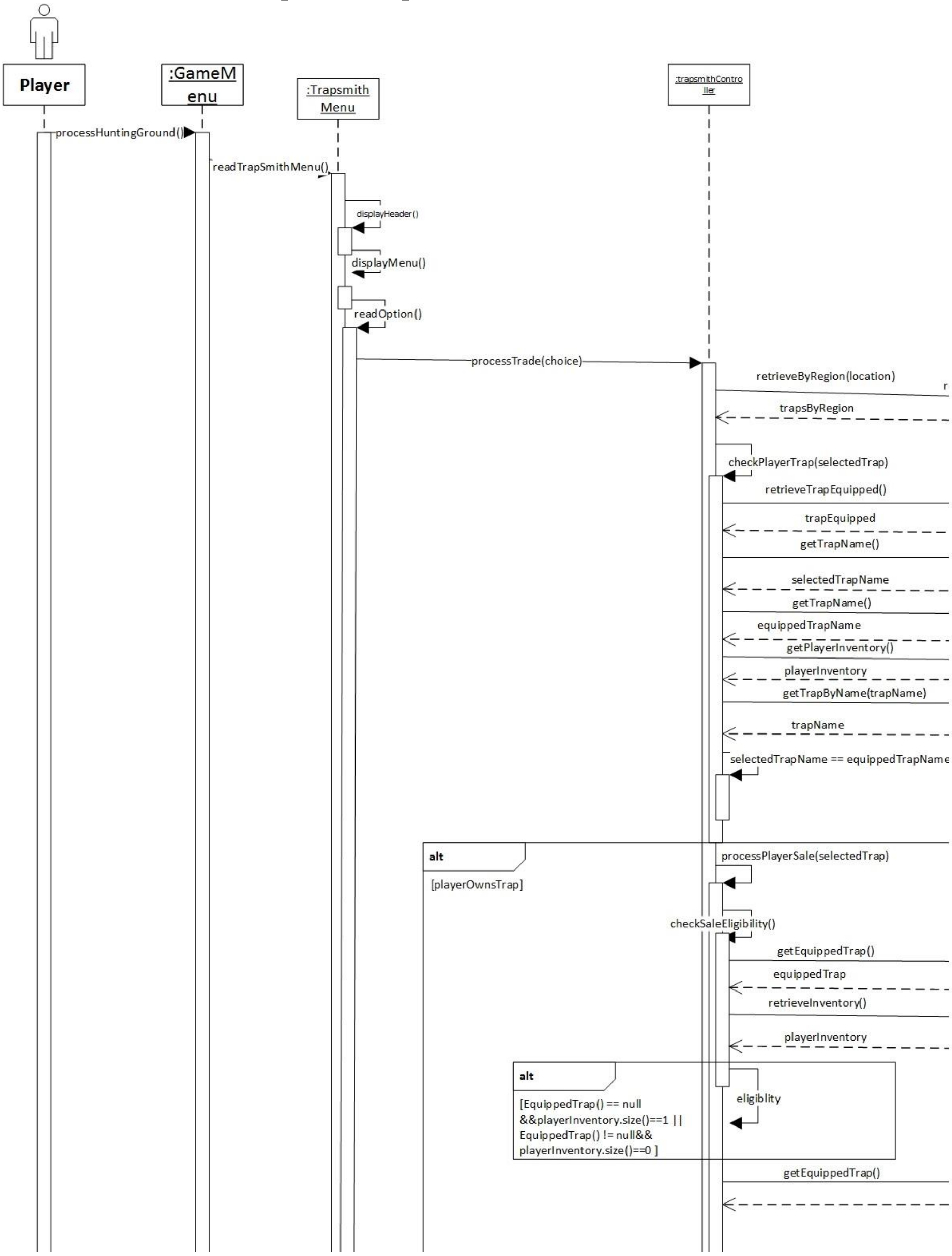

Player
:GameMenu
:HuntingGround Menu
:HuntingGround Controller
:Hunting History
:dukeManager
processHuntingGround()
readHuntingGround()
retrieveLogByUser (currentPlayer Player)
retrieveHistory (currentPlayer Player)
retrieveHistory (currentPlayer Player)
ArrayList(Record)
ArrayList(Record)
Display Hunting Records
SoundHorn()
soundHorn()
getDukeListByRegion(currentLocation String)
dukeInRegion
getPopulation()
int (population)
dukeCaught(currentDuke Duke)
getTrapPower()
int (trapPower)
getdukePower()
int (dukePower)
CalculateExp(currentDuke Duke)
getdukePower()
int (dukePower)
CalculateGold(currentDuke Duke)
getdukePower()
int (dukePower)
addPlayerGold(gainGold int)
addPlayerExp(gainExp int)
gameSave()
writeHistory(huntingHistory String, currentPlayer Player)
result
result

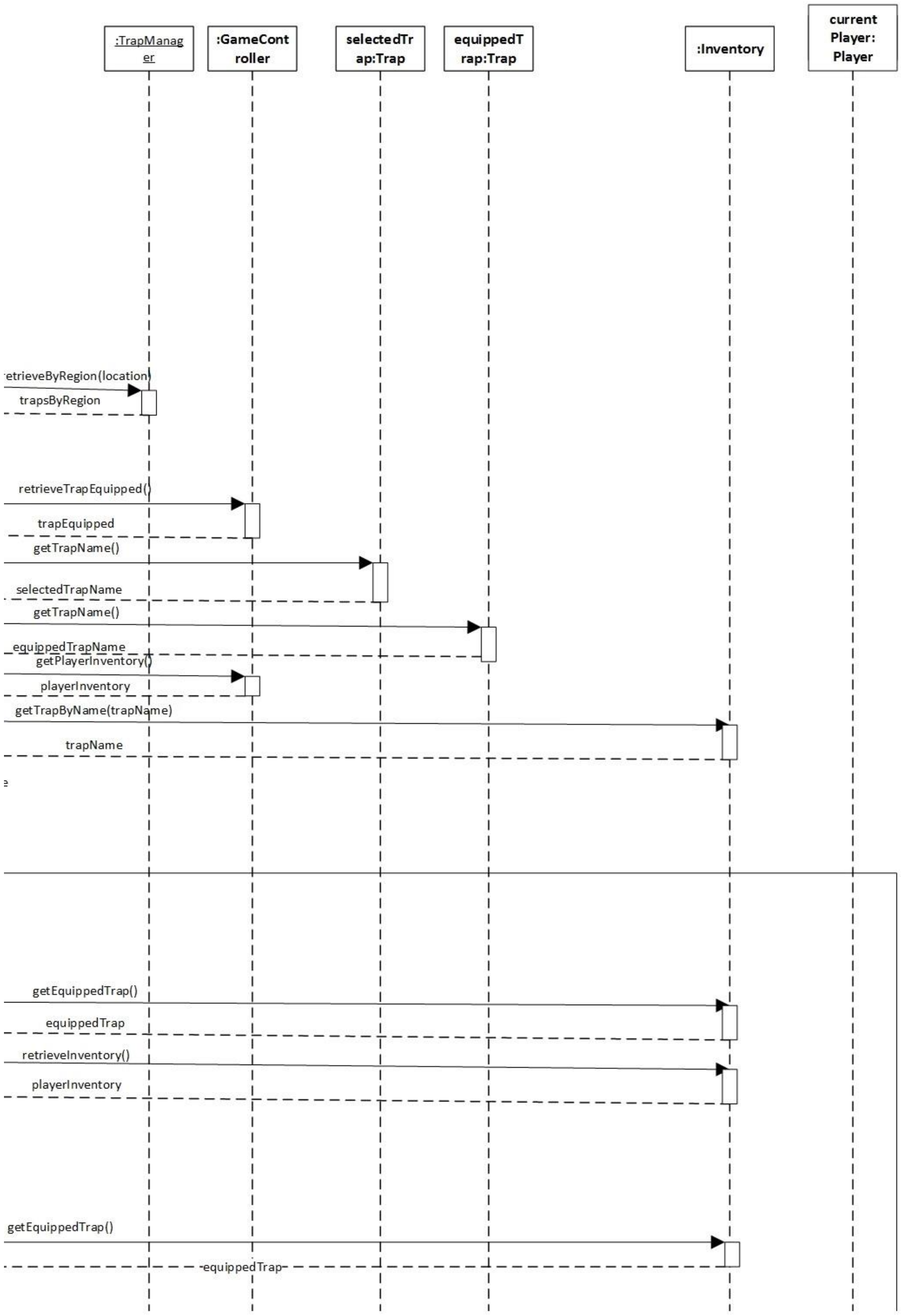

currrentduke:duke
currentTrap: Trap
currentPlayer : Player
gameController : GameController
loginController : LoginController
inventoryManager: InvetoryManager
playerManager : PlayerManager
getPopulation()
int (population)
getTrapPower()
int(trapPower)
getdukePower()
int (dukePower)
getdukePower()
int (dukePower)
addPlayerGold(gainGold int)
addPlayerExp(gainExp int)
gameSave()
savePlayerAccount()
save()
saveInventory()

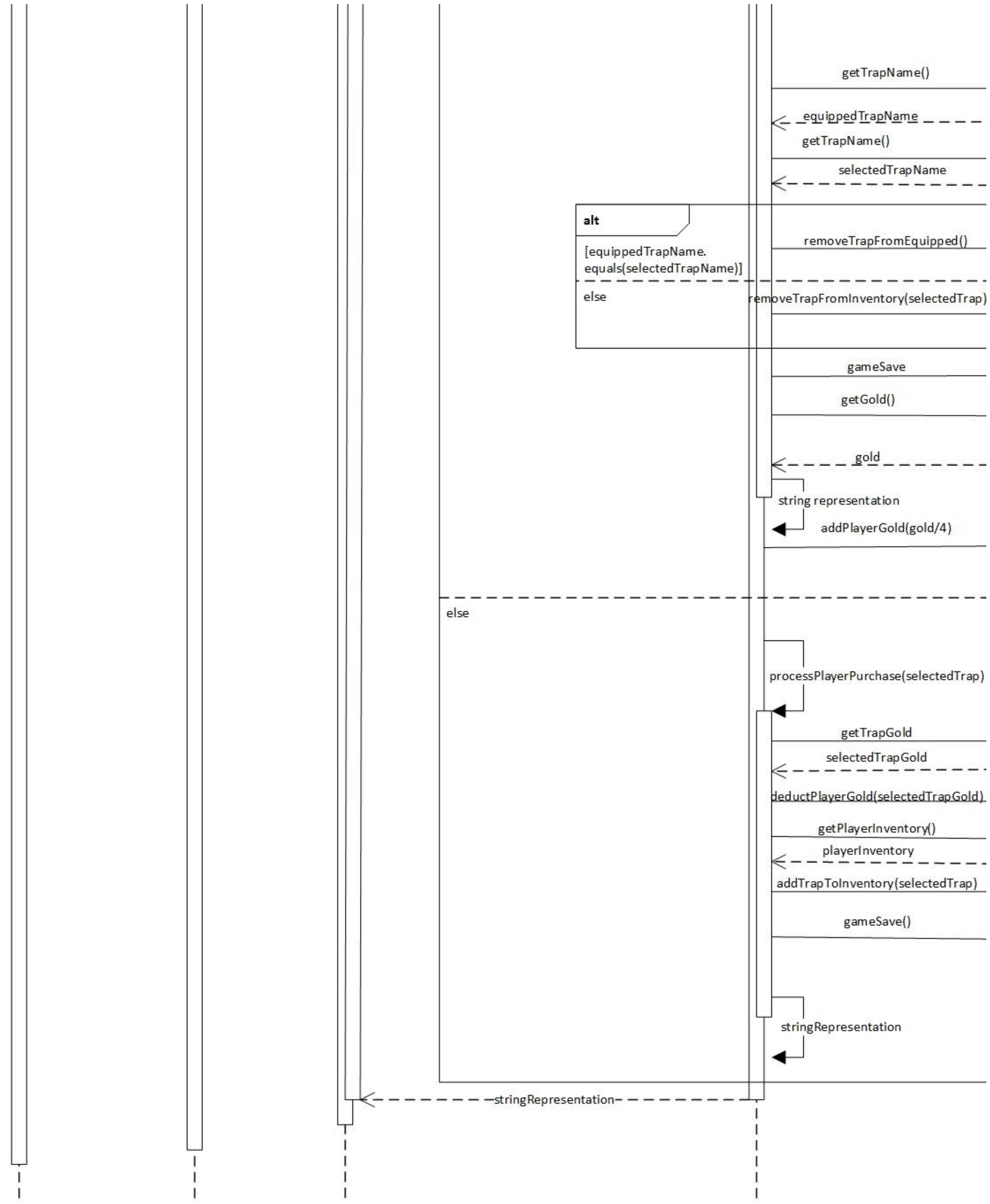
5.6. Manage Inventory

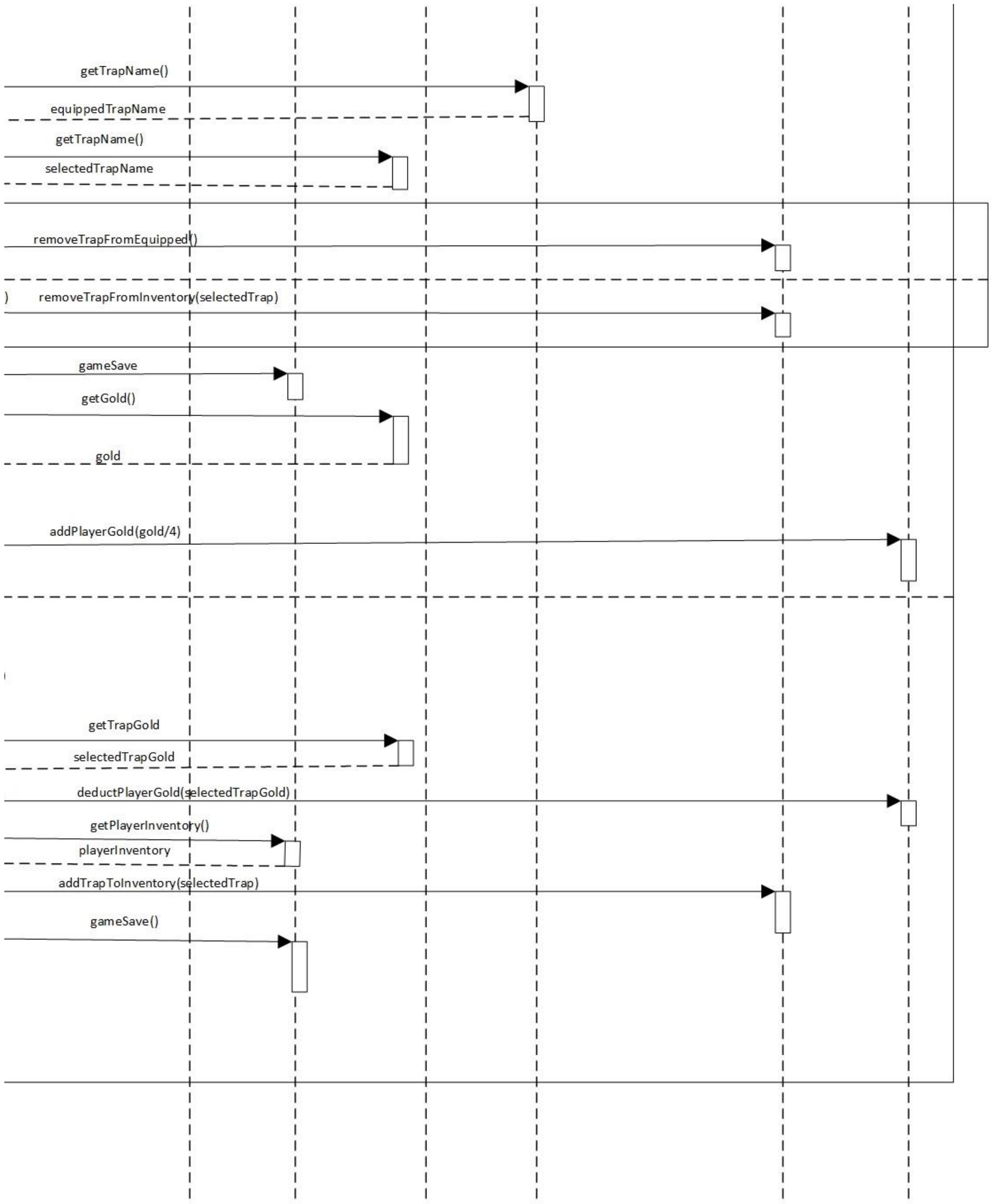


5.7. Purchase Trap / Sell Trap



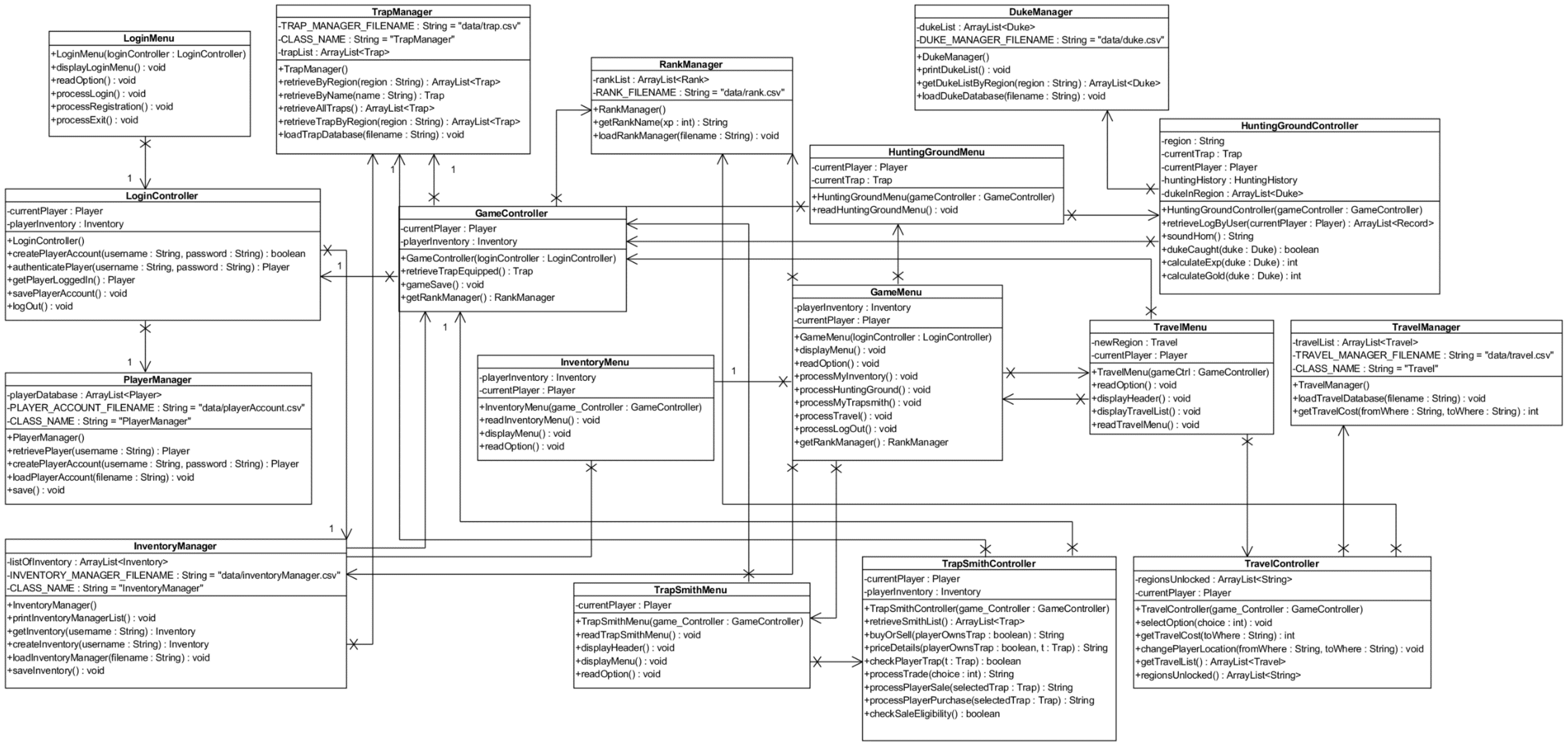

:TrapManager
:GameController
selectedTrap:Trap
equippedTrap:Trap
:Inventory
current Player: Player
etrieveByRegion(location)
trapsByRegion
retrieveTrapEquipped()
trapEquipped
getTrapName()
selectedTrapName
getTrapName()
equippedTrapName
getPlayerInventory()
playerInventory
getTrapByName(trapName)
trapName
e
getEquippedTrap()
equippedTrap
retrieveInventory()
playerInventory
getEquippedTrap()
equippedTrap


getTrapName()
equippedTrapName
getTrapName()
selectedTrapName
alt
[equippedTrapName.
equals(selectedTrapName)]
removeTrapFromEquipped()
else
removeTrapFromInventory(selectedTrap)
gameSave
getGold()
gold
string representation
addPlayerGold(gold/4)
else
processPlayerPurchase(selectedTrap)
getTrapGold
selectedTrapGold
deductPlayerGold(selectedTrapGold)
getPlayerInventory()
playerInventory
addTrapToInventory(selectedTrap)
gameSave()
stringRepresentation
stringRepresentation

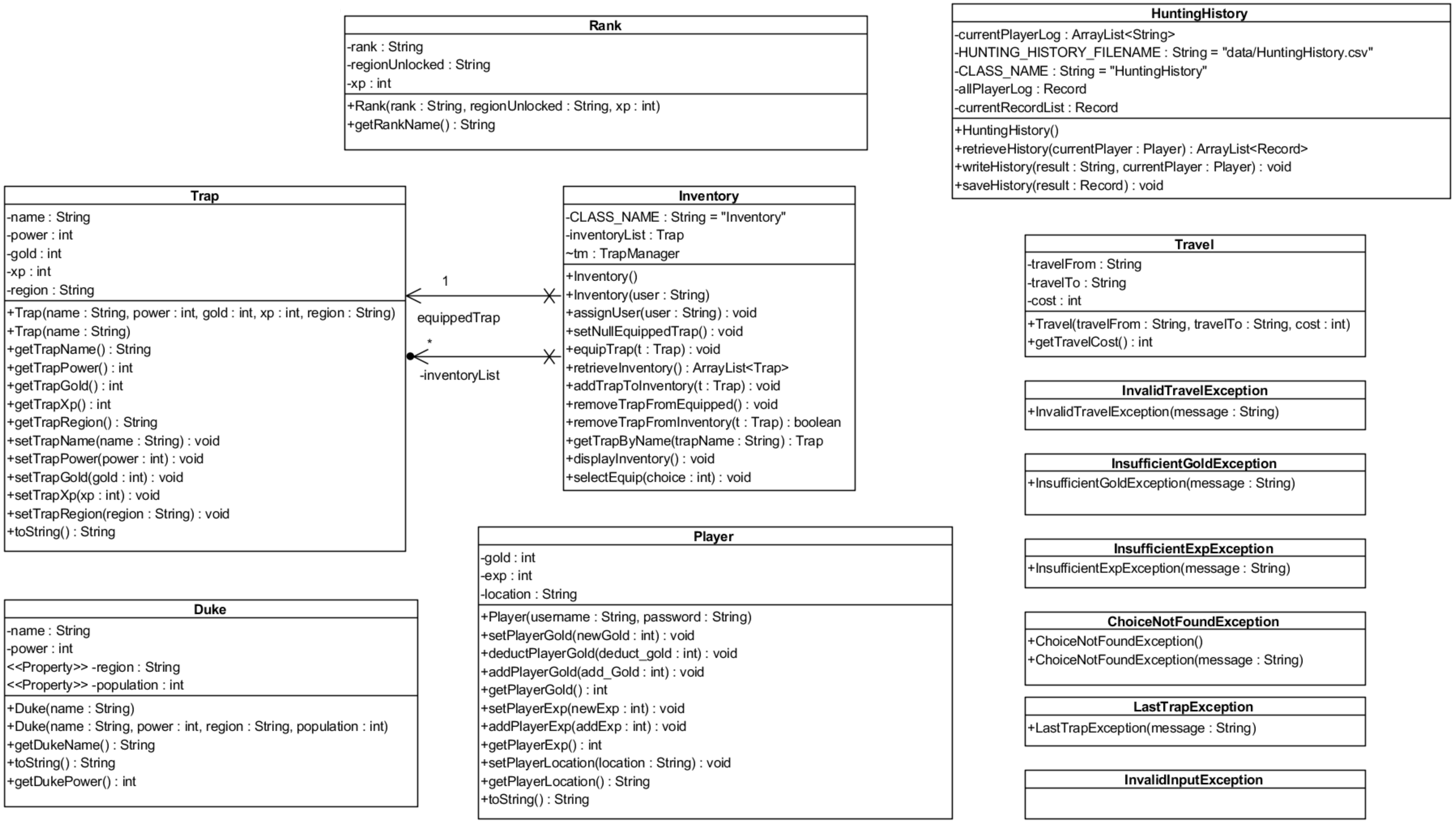

getTrapName()
equippedTrapName
getTrapName()
selectedTrapName
removeTrapFromEquipped()
)
removeTrapFromInventory(selectedTrap)
gameSave
getGold()
gold
addPlayerGold(gold/4)
getTrapGold
selectedTrapGold
deductPlayerGold(selectedTrapGold)
getPlayerInventory()
playerInventory
addTrapToInventory(selectedTrap)
gameSave()

6. Class Diagram

6.1. Boundary, Controller and Data Manager


LoginMenu
+LoginMenu(loginController : LoginController)
+displayLoginMenu() : void
+readOption() : void
+processLogin() : void
+processRegistration() : void
+processExit() : void
TrapManager
-TRAP_MANAGER_FILENAME : String = "data/trap.csv"
-CLASS_NAME : String = "TrapManager"
-trapList : ArrayList<Trap>
+TrapManager()
+retrieveByRegion(region : String) : ArrayList<Trap>
+retrieveByName(name : String) : Trap
+retrieveAllTraps() : ArrayList<Trap>
+retrieveTrapByRegion(region : String) : ArrayList<Trap>
+loadTrapDatabase(filename : String) : void
RankManager
-rankList : ArrayList<Rank>
-RANK_FILENAME : String = "data/rank.csv"
+RankManager()
+getRankName(xp : int) : String
+loadRankManager(filename : String) : void
DukeManager
-dukeList : ArrayList<Duke>
-DUKE_MANAGER_FILENAME : String = "data/duke.csv"
+DukeManager()
+printDukeList() : void
+getDukeListByRegion(region : String) : ArrayList<Duke>
+loadDukeDatabase(filename : String) : void
HuntingGroundController
-region : String
-currentTrap : Trap
-currentPlayer : Player
-huntingHistory : HuntingHistory
-dukeInRegion : ArrayList<Duke>
+HuntingGroundController(gameController : GameController)
+retrieveLogByUser(currentPlayer : Player) : ArrayList<Record>
+soundHorn() : String
+dukeCaught(duke : Duke) : boolean
+calculateExp(duke : Duke) : int
+calculateGold(duke : Duke) : int
HuntingGroundMenu
-currentPlayer : Player
-currentTrap : Trap
+HuntingGroundMenu(gameController : GameController)
+readHuntingGroundMenu() : void
LoginController
-currentPlayer : Player
-playerInventory : Inventory
+LoginController()
+createPlayerAccount(username : String, password : String) : boolean
+authenticatePlayer(username : String, password : String) : Player
+getPlayerLoggedIn() : Player
+savePlayerAccount() : void
+logOut() : void
GameController
-currentPlayer : Player
-playerInventory : Inventory
+GameController(loginController : LoginController)
+retrieveTrapEquipped() : Trap
+gameSave() : void
+getRankManager() : RankManager
GameMenu
-playerInventory : Inventory
-currentPlayer : Player
+GameMenu(loginController : LoginController)
+displayMenu() : void
+readOption() : void
+processMyInventory() : void
+processHuntingGround() : void
+processMyTrapsmith() : void
+processTravel() : void
+processLogOut() : void
+getRankManager() : RankManager
TravelMenu
-newRegion : Travel
-currentPlayer : Player
+TravelMenu(gameCtrl : GameController)
+readOption() : void
+displayHeader() : void
+displayTravelList() : void
+readTravelMenu() : void
TravelManager
-travelList : ArrayList<Travel>
-TRAVEL_MANAGER_FILENAME : String = "data/travel.csv"
-CLASS_NAME : String = "Travel"
+TravelManager()
+loadTravelDatabase(filename : String) : void
+getTravelCost(fromWhere : String, toWhere : String) : int
PlayerManager
-playerDatabase : ArrayList<Player>
-PLAYER_ACCOUNT_FILENAME : String = "data/playerAccount.csv"
-CLASS_NAME : String = "PlayerManager"
+PlayerManager()
+retrievePlayer(username : String) : Player
+createPlayerAccount(username : String, password : String) : Player
+loadPlayerAccount(filename : String) : void
+save() : void
InventoryMenu
-playerInventory : Inventory
-currentPlayer : Player
+InventoryMenu(game_Controller : GameController)
+readInventoryMenu() : void
+displayMenu() : void
+readOption() : void
InventoryManager
-listOfInventory : ArrayList<Inventory>
-INVENTORY_MANAGER_FILENAME : String = "data/inventoryManager.csv"
-CLASS_NAME : String = "InventoryManager"
+InventoryManager()
+printInventoryManagerList() : void
+getInventory(username : String) : Inventory
+createInventory(username : String) : Inventory
+loadInventoryManager(filename : String) : void
+saveInventory() : void
TrapSmithMenu
-currentPlayer : Player
+TrapSmithMenu(game_Controller : GameController)
+readTrapSmithMenu() : void
+displayHeader() : void
+displayMenu() : void
+readOption() : void
TrapSmithController
-currentPlayer : Player
-playerInventory : Inventory
+TrapSmithController(game_Controller : GameController)
+retrieveSmithList() : ArrayList<Trap>
+buyOrSell(playerOwnsTrap : boolean) : String
+priceDetails(playerOwnsTrap : boolean, t : Trap) : String
+checkPlayerTrap(t : Trap) : boolean
+processTrade(choice : int) : String
+processPlayerSale(selectedTrap : Trap) : String
+processPlayerPurchase(selectedTrap : Trap) : String
+checkSaleEligibility() : boolean
TravelController
-regionsUnlocked : ArrayList<String>
-currentPlayer : Player
+TravelController(game_Controller : GameController)
+selectOption(choice : int) : void
+getTravelCost(toWhere : String) : int
+changePlayerLocation(fromWhere : String, toWhere : String) : void
+getTravelList() : ArrayList<Travel>
+regionsUnlocked() : ArrayList<String>
1
1
1
1
1
1
1
1

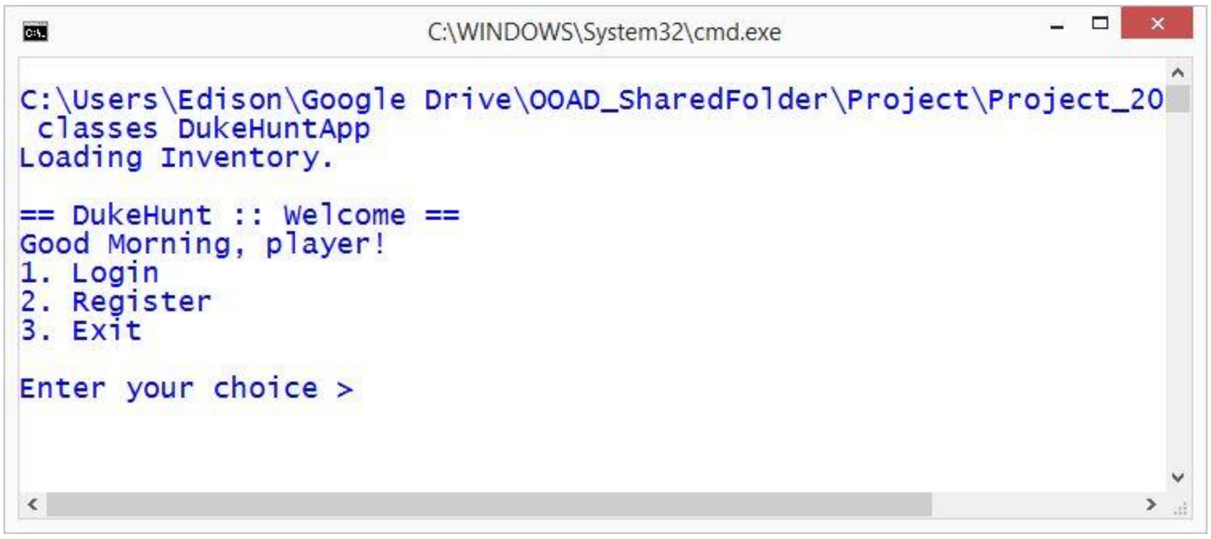
6.2. Entity


Rank
-rank : String
-regionUnlocked : String
-xp : int
+Rank(rank : String, regionUnlocked : String, xp : int)
+getRankName() : String
HuntingHistory
-currentPlayerLog : ArrayList<String>
-HUNTING_HISTORY_FILENAME : String = "data/HuntingHistory.csv"
-CLASS_NAME : String = "HuntingHistory"
-allPlayerLog : Record
-currentRecordList : Record
+HuntingHistory()
+retrieveHistory(currentPlayer : Player) : ArrayList<Record>
+writeHistory(result : String, currentPlayer : Player) : void
+saveHistory(result : Record) : void
Trap
-name : String
-power : int
-gold : int
-xp : int
-region : String
+Trap(name : String, power : int, gold : int, xp : int, region : String)
+Trap(name : String)
+getTrapName() : String
+getTrapPower() : int
+getTrapGold() : int
+getTrapXp() : int
+getTrapRegion() : String
+setTrapName(name : String) : void
+setTrapPower(power : int) : void
+setTrapGold(gold : int) : void
+setTrapXp(xp : int) : void
+setTrapRegion(region : String) : void
+toString() : String
1
equippedTrap
*
-inventoryList
Inventory
-CLASS_NAME : String = "Inventory"
-inventoryList : Trap
~tm : TrapManager
+Inventory()
+Inventory(user : String)
+assignUser(user : String) : void
+setNullEquippedTrap() : void
+equipTrap(t : Trap) : void
+retrieveInventory() : ArrayList<Trap>
+addTrapToInventory(t : Trap) : void
+removeTrapFromEquipped() : void
+removeTrapFromInventory(t : Trap) : boolean
+getTrapByName(trapName : String) : Trap
+displayInventory() : void
+selectEquip(choice : int) : void
Travel
-travelFrom : String
-travelTo : String
-cost : int
+Travel(travelFrom : String, travelTo : String, cost : int)
+getTravelCost() : int
InvalidTravelException
+InvalidTravelException(message : String)
InsufficientGoldException
+InsufficientGoldException(message : String)
InsufficientExpException
+InsufficientExpException(message : String)
ChoiceNotFoundException
+ChoiceNotFoundException()
+ChoiceNotFoundException(message : String)
LastTrapException
+LastTrapException(message : String)
InvalidInputException
Duke
-name : String
-power : int
<<Property>> -region : String
<<Property>> -population : int
+Duke(name : String)
+Duke(name : String, power : int, region : String, population : int)
+getDukeName() : String
+toString() : String
+getDukePower() : int
Player
-gold : int
-exp : int
-location : String
+Player(username : String, password : String)
+setPlayerGold(newGold : int) : void
+deductPlayerGold(deduct_gold : int) : void
+addPlayerGold(add_Gold : int) : void
+getPlayerGold() : int
+setPlayerExp(newExp : int) : void
+addPlayerExp(addExp : int) : void
+getPlayerExp() : int
+setPlayerLocation(location : String) : void
+getPlayerLocation() : String
+toString() : String

7. Screenshot of Sample Runs

7.1. Log In

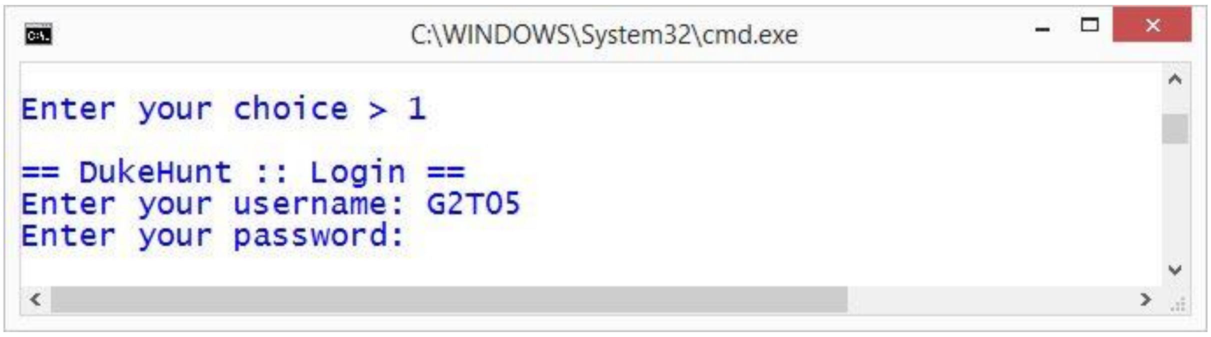
Once the program is launched, the following console will be displayed and user will be required to enter choice number 1 (Login), 2 (Register Account) or 3 (Exit).



```
C:\Users\Edison\Google Drive\OOAD_SharedFolder\Project\Project_20
 classes DukeHuntApp
Loading Inventory.

== DukeHunt :: Welcome ==
Good Morning, player!
1. Login
2. Register
3. Exit

Enter your choice >
```

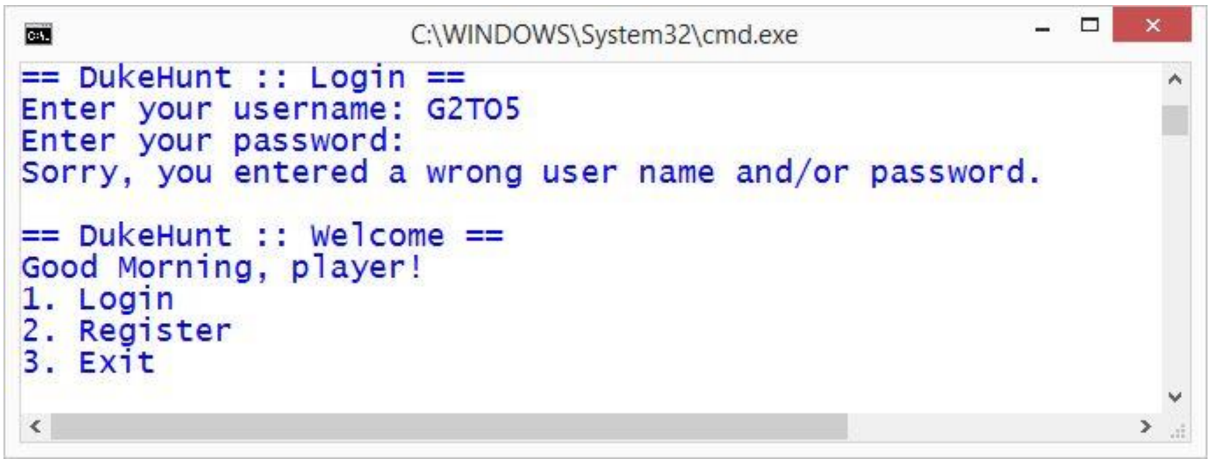
If option one is selected, user will be brought to this page where username and password are required.

As we are concerned about your privacy, the password is masked.



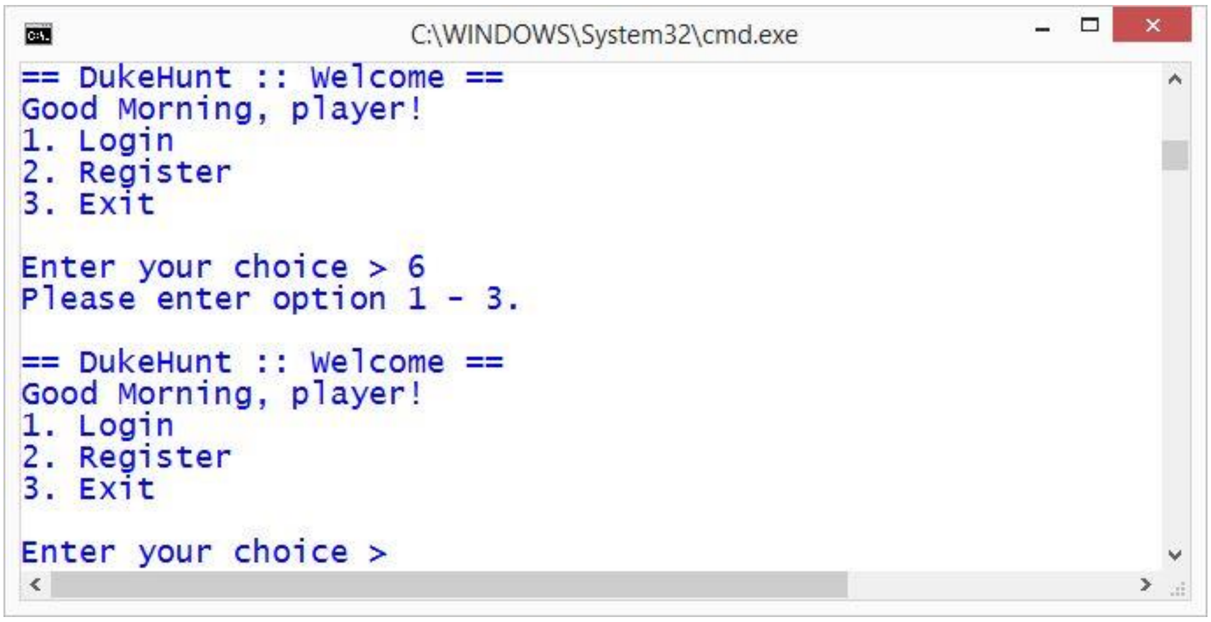
```
Enter your choice > 1

== DukeHunt :: Login ==
Enter your username: G2T05
Enter your password:
```

If user keys in the wrong password, the main console will be displayed. User can then choose to try and log in, register account, or exit the program.



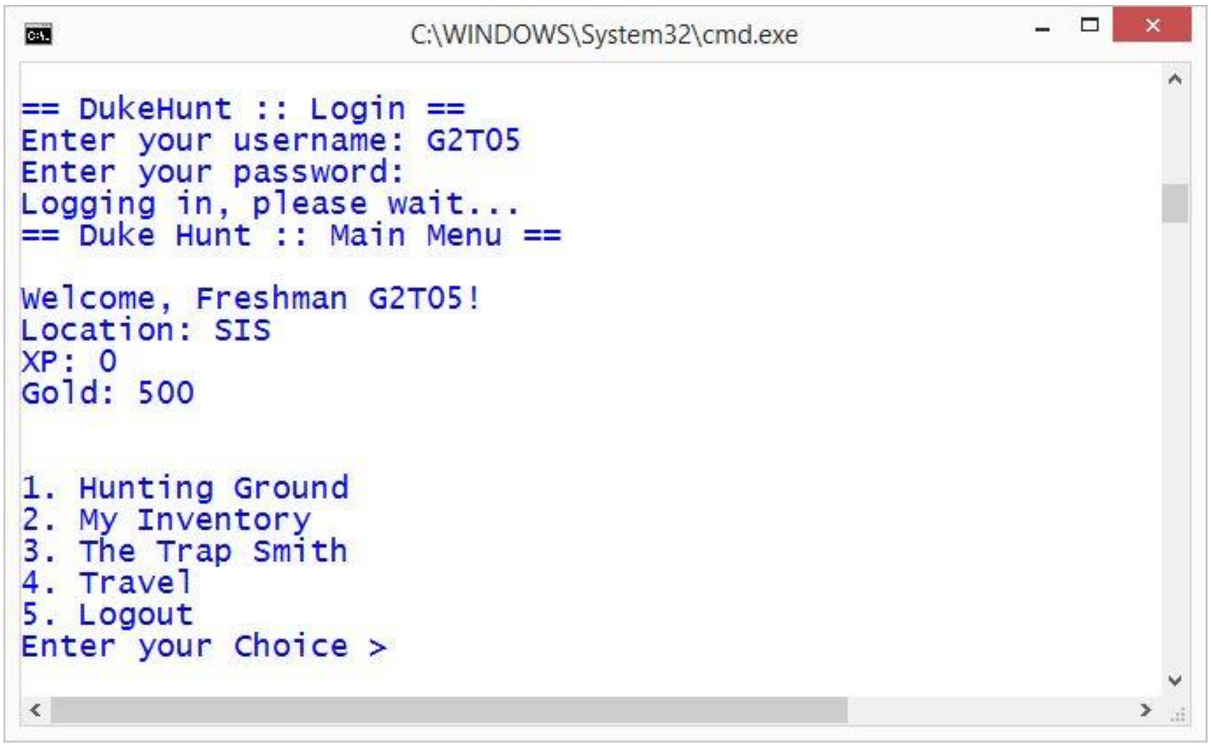
```
== DukeHunt :: Login ==
Enter your username: G2T05
Enter your password:
Sorry, you entered a wrong user name and/or password.

== DukeHunt :: Welcome ==
Good Morning, player!
1. Login
2. Register
3. Exit
```

If user selects choices that are not within the list, the following message will be displayed.



```
== DukeHunt :: Welcome ==
Good Morning, player!
1. Login
2. Register
3. Exit

Enter your choice > 6
Please enter option 1 - 3.

== DukeHunt :: Welcome ==
Good Morning, player!
1. Login
2. Register
3. Exit

Enter your choice >
```

Upon successful authentication, the user will be brought to the Game Menu and he/she can start playing the game.

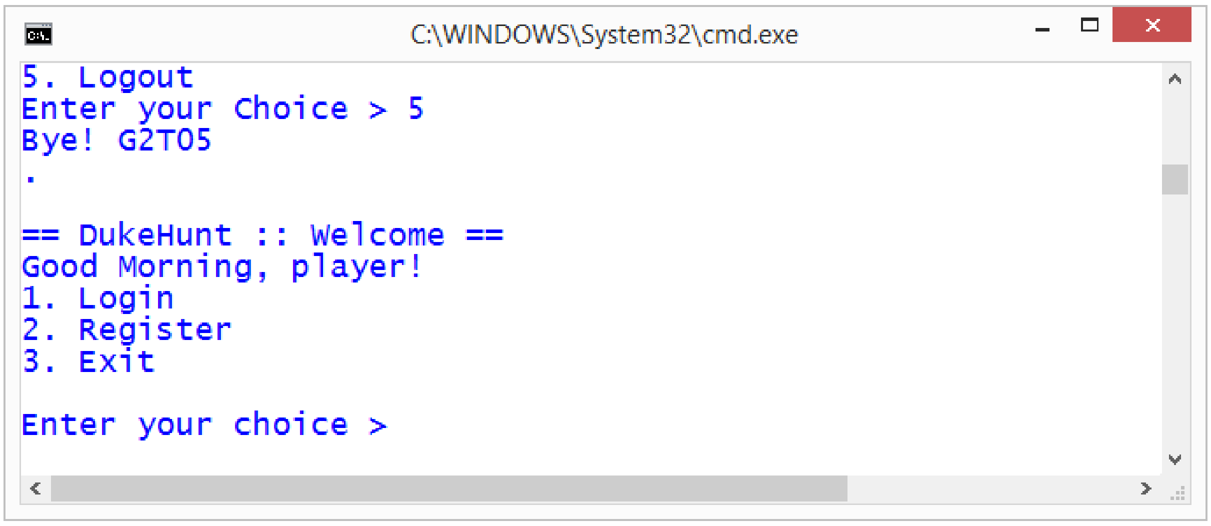


```

== DukeHunt :: Login ==
Enter your username: G2T05
Enter your password:
Logging in, please wait...
== Duke Hunt :: Main Menu ==

Welcome, Freshman G2T05!
Location: SIS
XP: 0
Gold: 500


1. Hunting Ground
2. My Inventory
3. The Trap Smith
4. Travel
5. Logout
Enter your Choice >
```

7.2. Log Out

Should the user need to log out at any point of time, press option 5 in the Game Menu.

Since the game is auto-saved, the user's progress is logged automatically within the system and he/she can resume the game at another time.

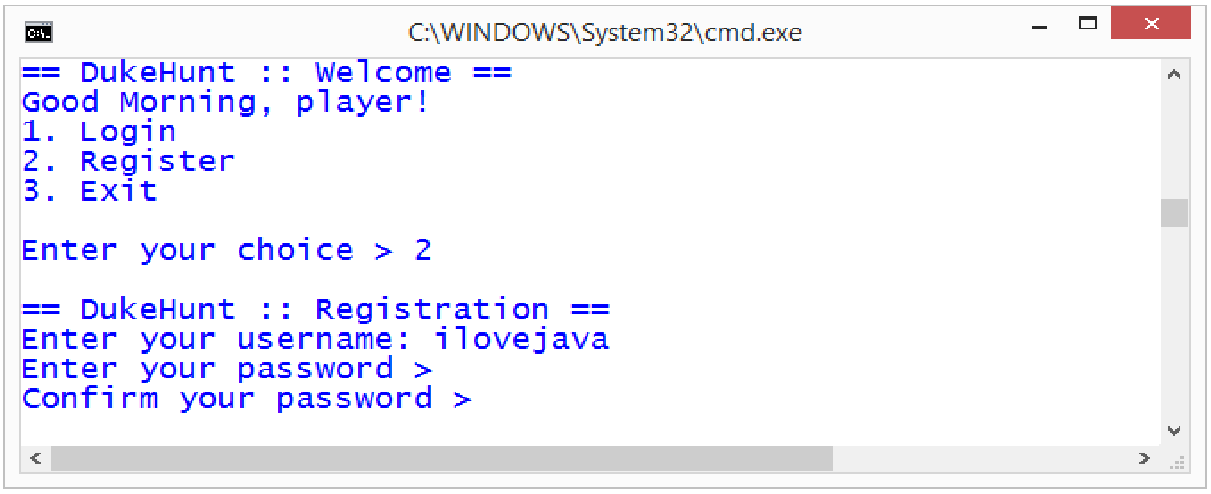


```
5. Logout
Enter your Choice > 5
Bye! G2T05
.

== DukeHunt :: Welcome ==
Good Morning, player!
1. Login
2. Register
3. Exit

Enter your choice >
```

Once logged out, the user can choose to log in, register or exit the program.

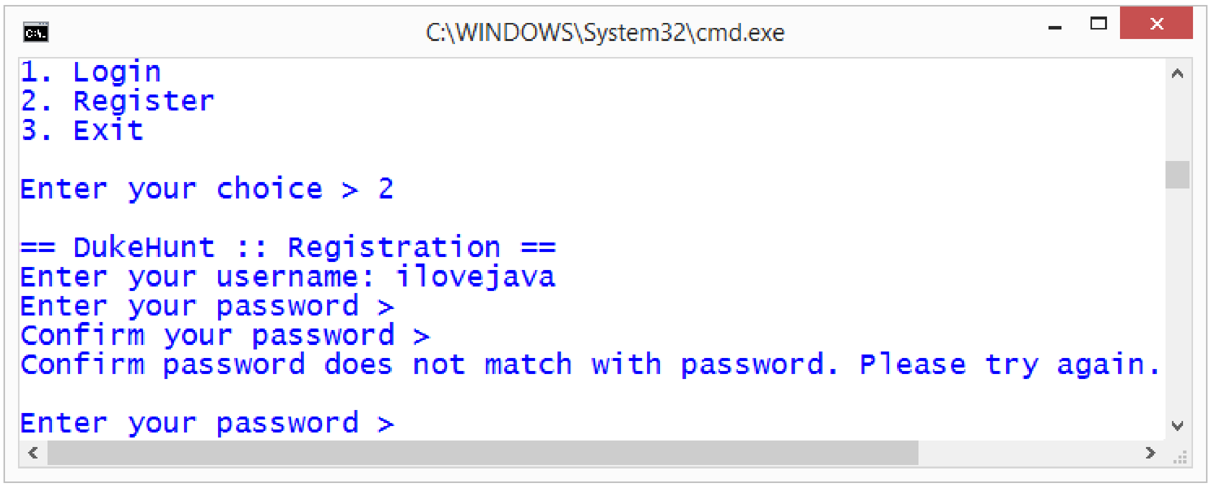
7.3. Register Account

If user chooses option 2, the following page will be shown.



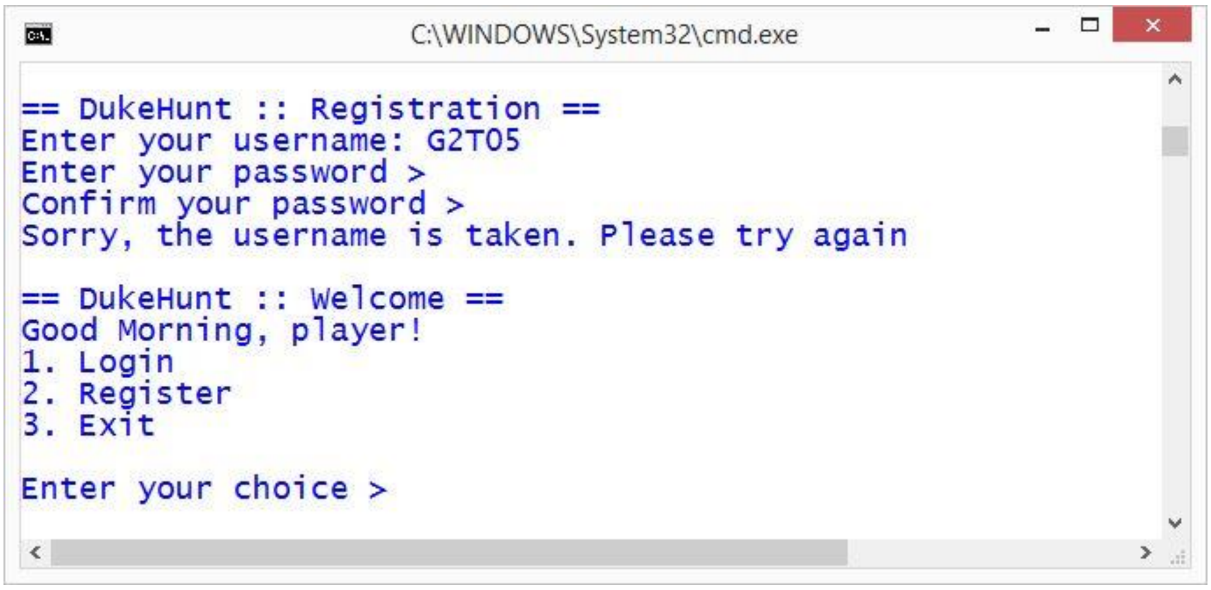
```
== DukeHunt :: Welcome ==
Good Morning, player!
1. Login
2. Register
3. Exit

Enter your choice > 2

== DukeHunt :: Registration ==
Enter your username: ilovejava
Enter your password >
Confirm your password >
```

If the passwords do not match, this error message will be displayed.



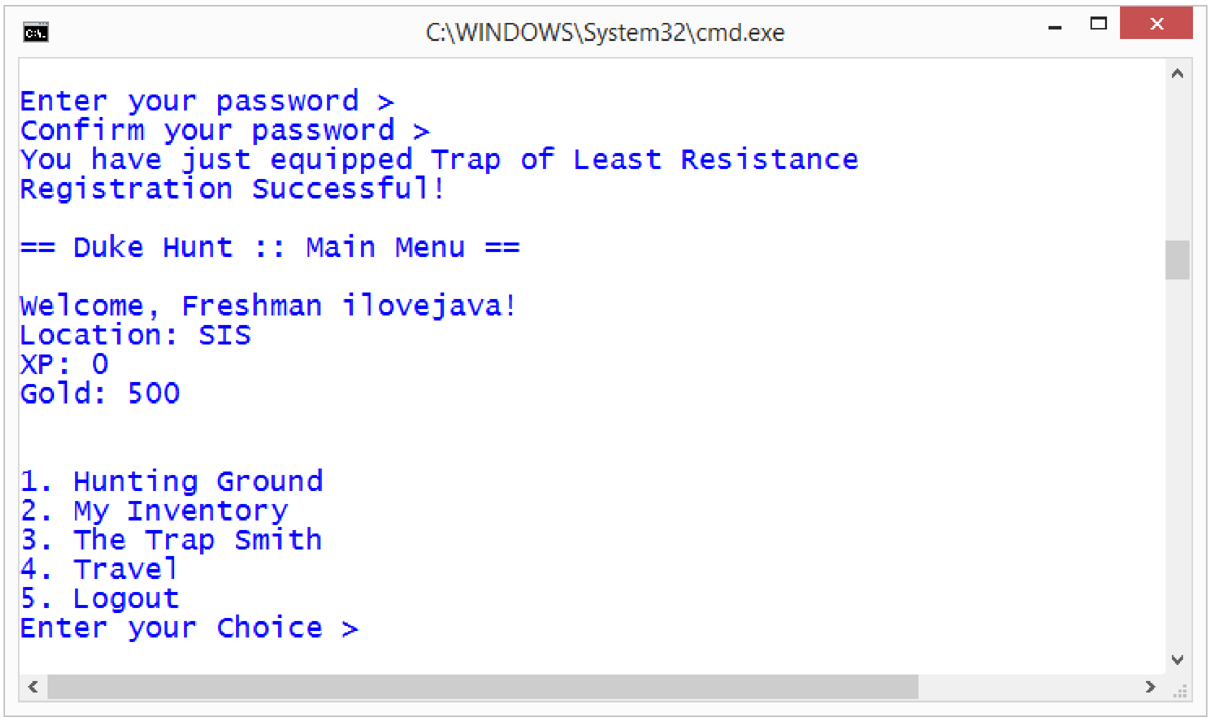
```
1. Login
2. Register
3. Exit

Enter your choice > 2

== DukeHunt :: Registration ==
Enter your username: ilovejava
Enter your password >
Confirm your password >
Confirm password does not match with password. Please try again.

Enter your password >
```

If the username is taken, the console will display this error message.



```

== DukeHunt :: Registration ==
Enter your username: G2T05
Enter your password >
Confirm your password >
Sorry, the username is taken. Please try again

== DukeHunt :: Welcome ==
Good Morning, player!
1. Login
2. Register
3. Exit

Enter your choice >
```

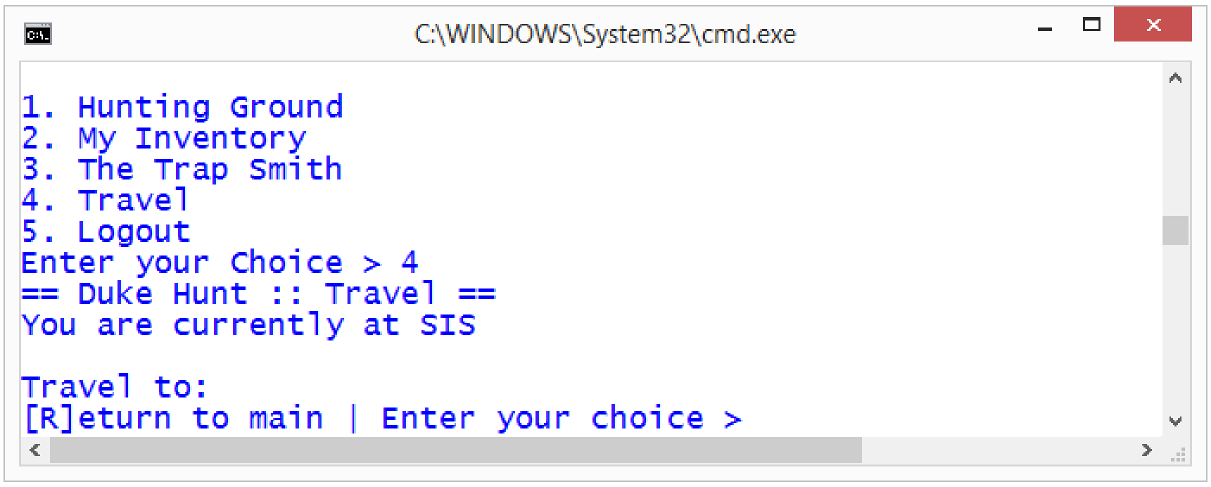
Once successfully registered, the user account will be created and equipped with "Trap of Least Resistance" by default. User will also be given 500 gold as a starter pack so as to begin on the journey of duke hunt.



```
Enter your password >
Confirm your password >
You have just equipped Trap of Least Resistance
Registration Successful!

== Duke Hunt :: Main Menu ==

Welcome, Freshman ilovejava!
Location: SIS
XP: 0
Gold: 500


1. Hunting Ground
2. My Inventory
3. The Trap Smith
4. Travel
5. Logout
Enter your Choice >
```

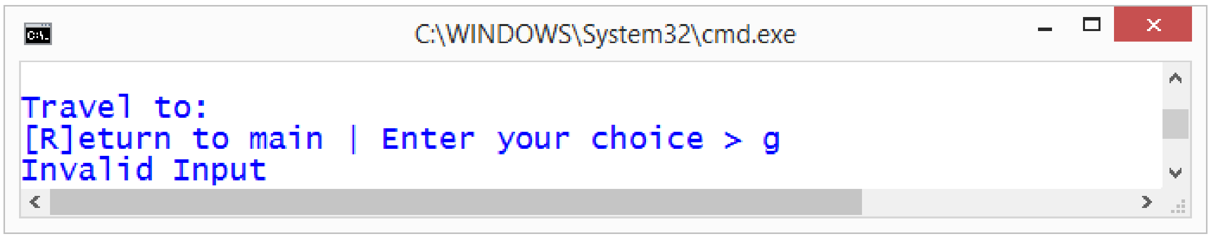
7.4. Travel

Users can travel to different location by hitting the travel option. Traveling locations are determined by experience and users have to unlock new regions by gaining experience. For freshman, the list of places he or she can travel is displayed as blank.



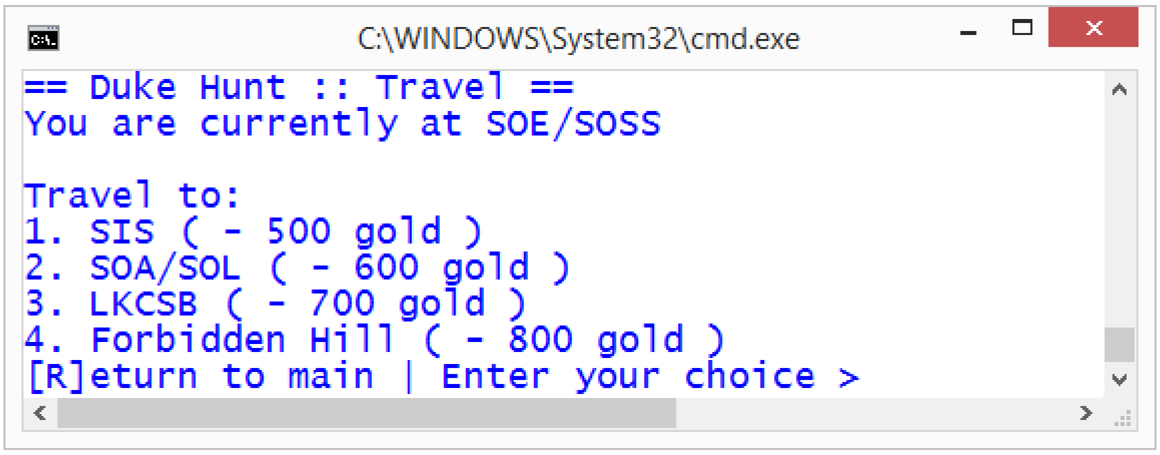
```
1. Hunting Ground
2. My Inventory
3. The Trap Smith
4. Travel
5. Logout
Enter your Choice > 4
== Duke Hunt :: Travel ==
You are currently at SIS

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

If the user keys in a wrong input, an error message will be displayed.



```
Travel to:
[R]eturn to main | Enter your choice > g
Invalid Input
```

New regions will be unlocked as the player progresses through the game by fighting dukes and gaining experience. Ultimately, player should be able to travel to every location as shown below.

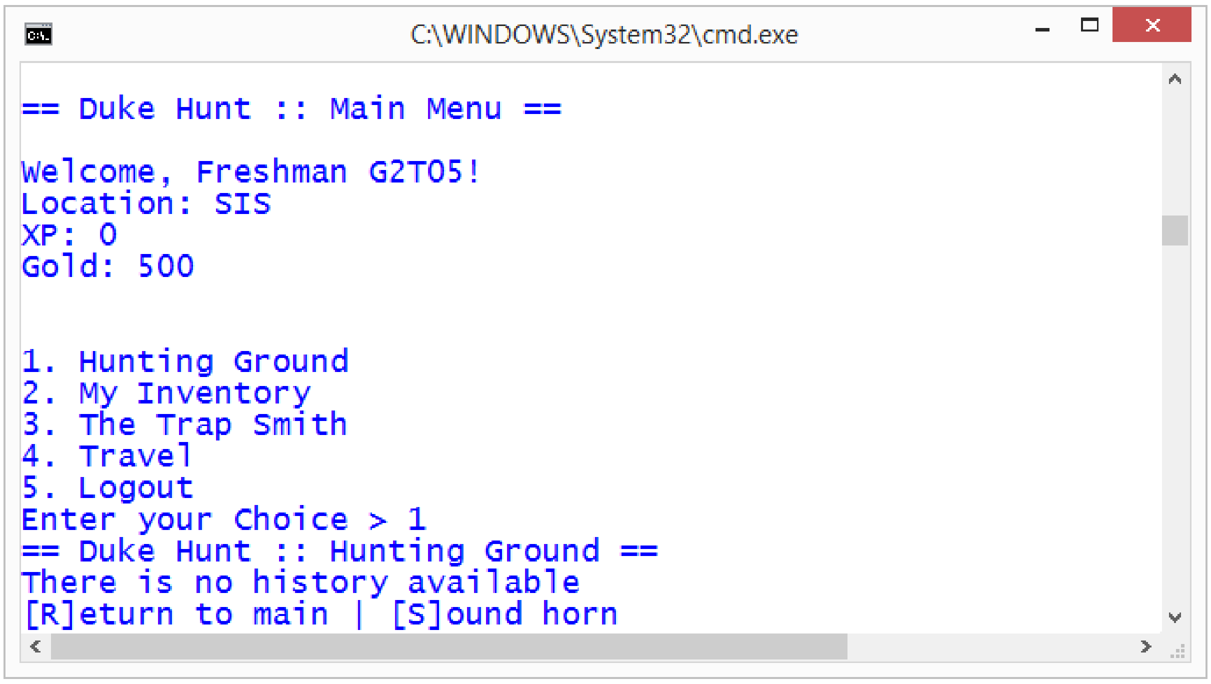


```
== Duke Hunt :: Travel ==
You are currently at SOE/SOSS

Travel to:
1. SIS ( - 500 gold )
2. SOA/SOL ( - 600 gold )
3. LKCSB ( - 700 gold )
4. Forbidden Hill ( - 800 gold )
[R]eturn to main | Enter your choice >
```

7.5. Sound Horn

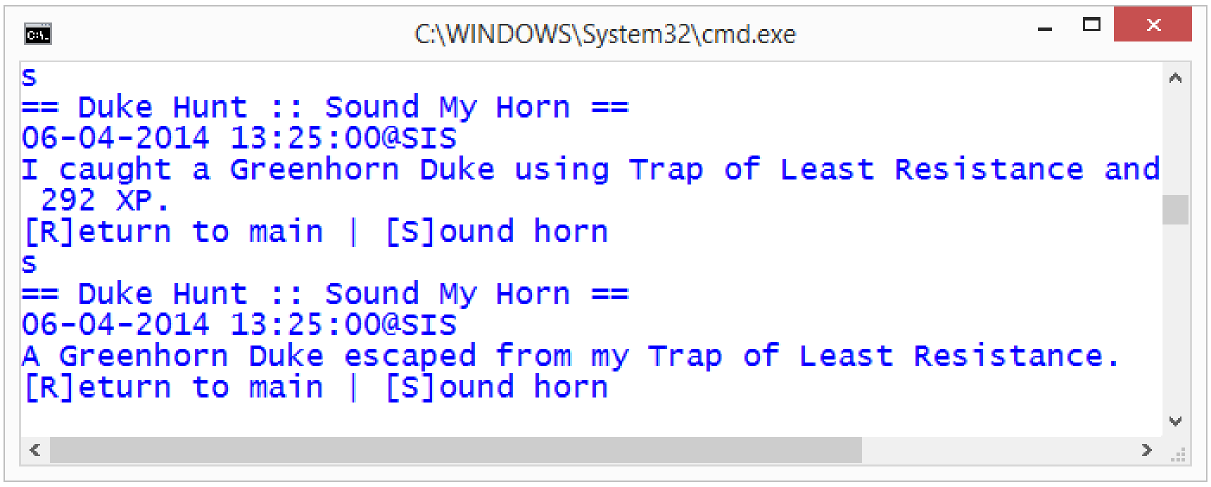
Freshman can earn experience and gold by hunting dukes. By hitting the Hunting Ground option in the main page, the user will hence be able to hunt duke.

The last ten hunting records will be displayed. For newly registered users, no hunting records will be displayed.



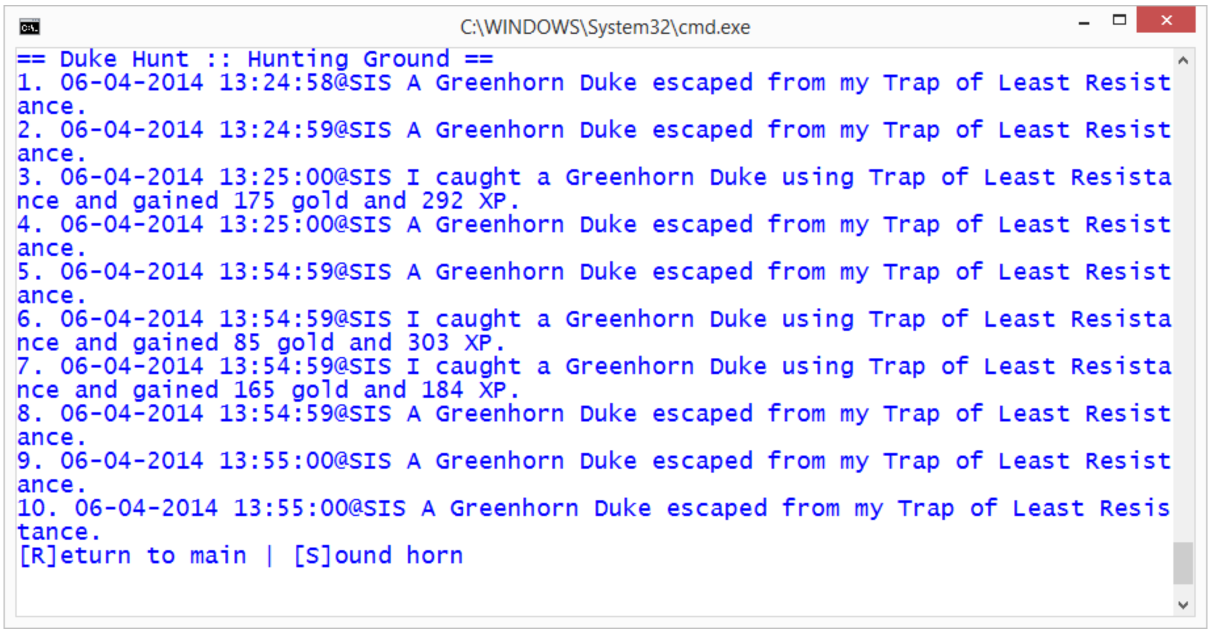
```
== Duke Hunt :: Main Menu ==

Welcome, Freshman G2T05!
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 ==
There is no history available
[R]eturn to main | [S]ound horn
```

User can then hunt for duke by hitting 'S' key. The game will then display if the duke is caught or not. If a duke is caught, gold and experience earned will be displayed and added into the user account. If duke is not caught, the game will display the status of catch and user can then choose to sound horn again or to return to main menu.



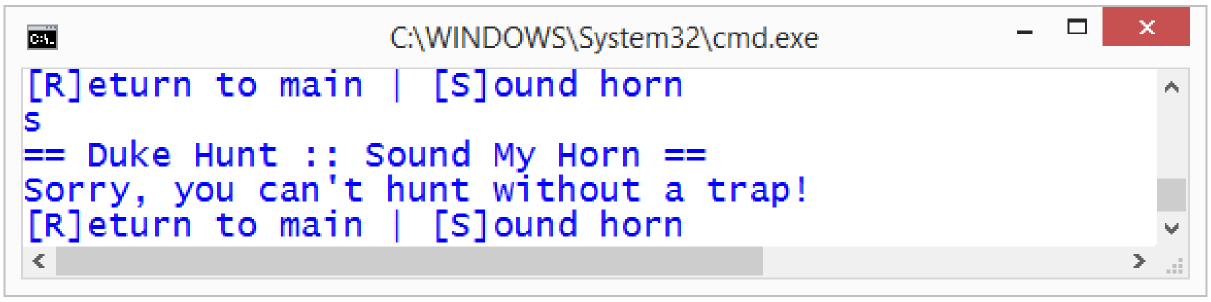
```
s
== Duke Hunt :: Sound My Horn ==
06-04-2014 13:25:00@SIS
I caught a Greenhorn Duke using Trap of Least Resistance and
 292 XP.
[R]eturn to main | [S]ound horn
s
== Duke Hunt :: Sound My Horn ==
06-04-2014 13:25:00@SIS
A Greenhorn Duke escaped from my Trap of Least Resistance.
[R]eturn to main | [S]ound horn
```

Up to 10 hunting records will be displayed for the game.



```
C:\WINDOWS\System32\cmd.exe
== Duke Hunt :: Hunting Ground ==
1. 06-04-2014 13:24:58@SIS A Greenhorn Duke escaped from my Trap of Least Resist
ance.
2. 06-04-2014 13:24:59@SIS A Greenhorn Duke escaped from my Trap of Least Resist
ance.
3. 06-04-2014 13:25:00@SIS I caught a Greenhorn Duke using Trap of Least Resista
nce and gained 175 gold and 292 XP.
4. 06-04-2014 13:25:00@SIS A Greenhorn Duke escaped from my Trap of Least Resist
ance.
5. 06-04-2014 13:54:59@SIS A Greenhorn Duke escaped from my Trap of Least Resist
ance.
6. 06-04-2014 13:54:59@SIS I caught a Greenhorn Duke using Trap of Least Resista
nce and gained 85 gold and 303 XP.
7. 06-04-2014 13:54:59@SIS I caught a Greenhorn Duke using Trap of Least Resista
nce and gained 165 gold and 184 XP.
8. 06-04-2014 13:54:59@SIS A Greenhorn Duke escaped from my Trap of Least Resist
ance.
9. 06-04-2014 13:55:00@SIS A Greenhorn Duke escaped from my Trap of Least Resist
ance.
10. 06-04-2014 13:55:00@SIS A Greenhorn Duke escaped from my Trap of Least Resis
tance.
[R]eturn to main | [S]ound horn
```

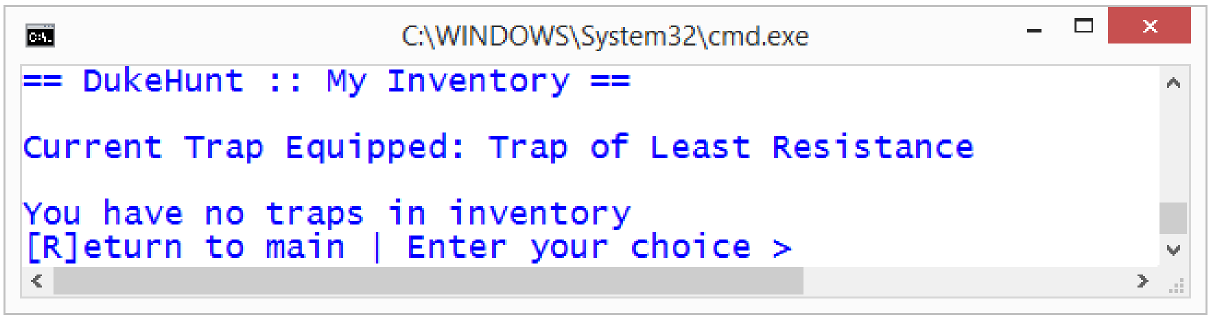
If the player do not have any Trap equipped, player will not be able to hunt in the hunting ground. The following message will be displayed and player has to go to inventory to equip a trap in order to hunt.



```
C:\WINDOWS\System32\cmd.exe
[R]eturn to main | [S]ound horn
s
== Duke Hunt :: Sound My Horn ==
Sorry, you can't hunt without a trap!
[R]eturn to main | [S]ound horn
```

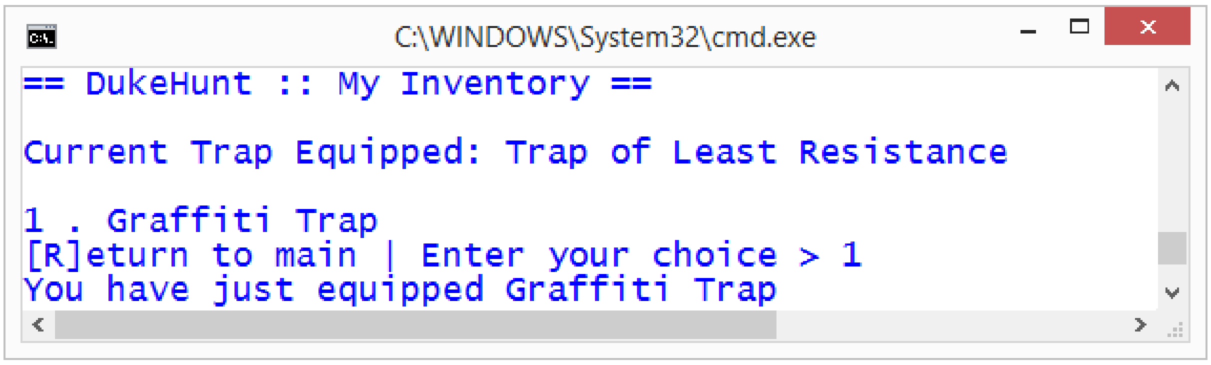
7.6. Manage Inventory

Player can access his/her inventory by choosing the inventory function under game menu. For starters, the Trap of Least Resistance will be given and equipped by default.



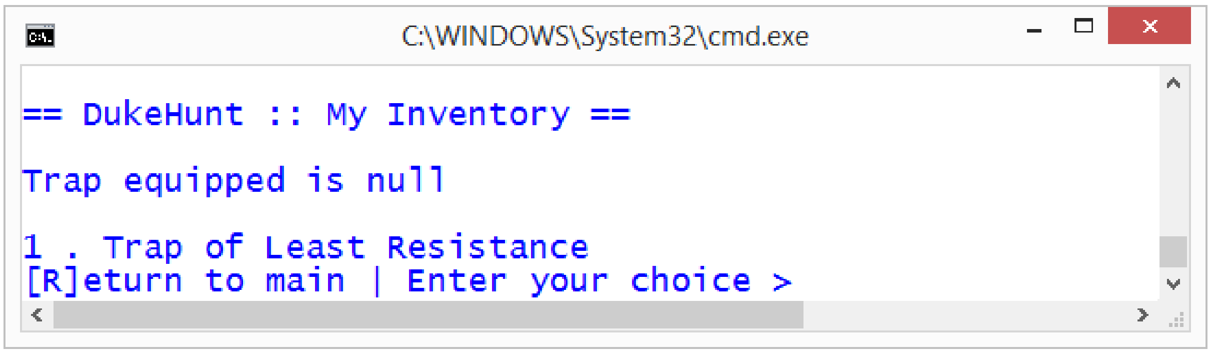
```
== DukeHunt :: My Inventory ==

Current Trap Equipped: Trap of Least Resistance

You have no traps in inventory
[R]eturn to main | Enter your choice >
```

Player can change the equipped trap by selecting it from the list.



```
== DukeHunt :: My Inventory ==

Current Trap Equipped: Trap of Least Resistance

1 . Graffiti Trap
[R]eturn to main | Enter your choice > 1
You have just equipped Graffiti Trap
```

If player sells his equipped trap, the equipped trap will be null. In this case, the player cannot hunt until he equips a trap from his inventory. (See under *Sound Horn)*

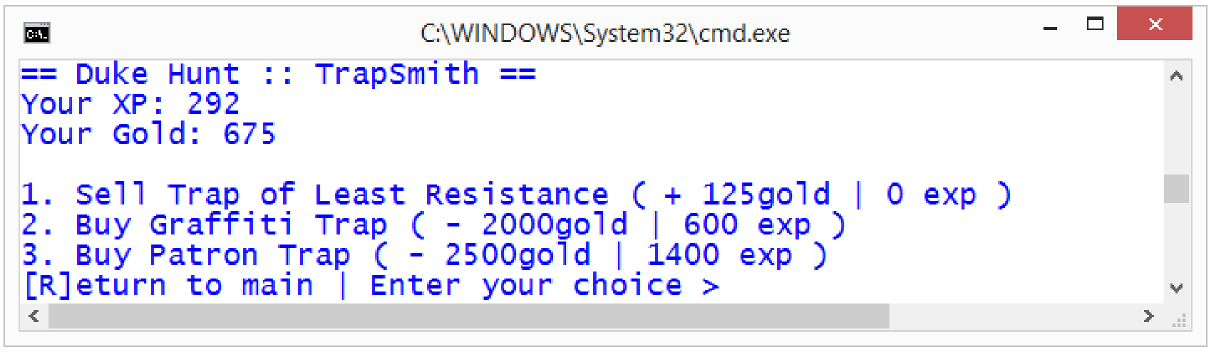


```
== DukeHunt :: My Inventory ==

Trap equipped is null

1 . Trap of Least Resistance
[R]eturn to main | Enter your choice >
```

7.7. Purchase Trap

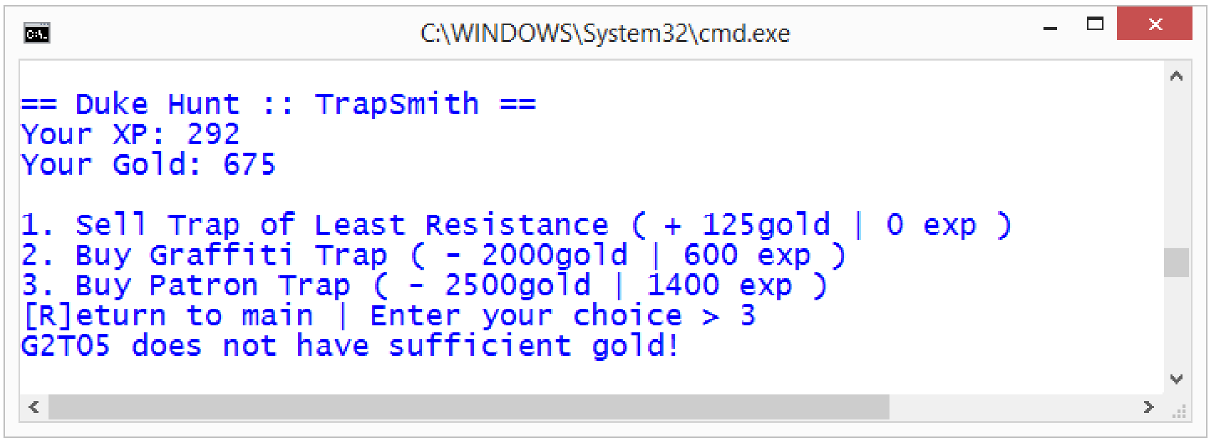
Player can choose to enhance game experience by buying more powerful traps so as to catch dukes with a higher probability. Player can do so by hitting the trapsmith key under the main menu.

The following image displays what it will look like in the trapsmith menu.



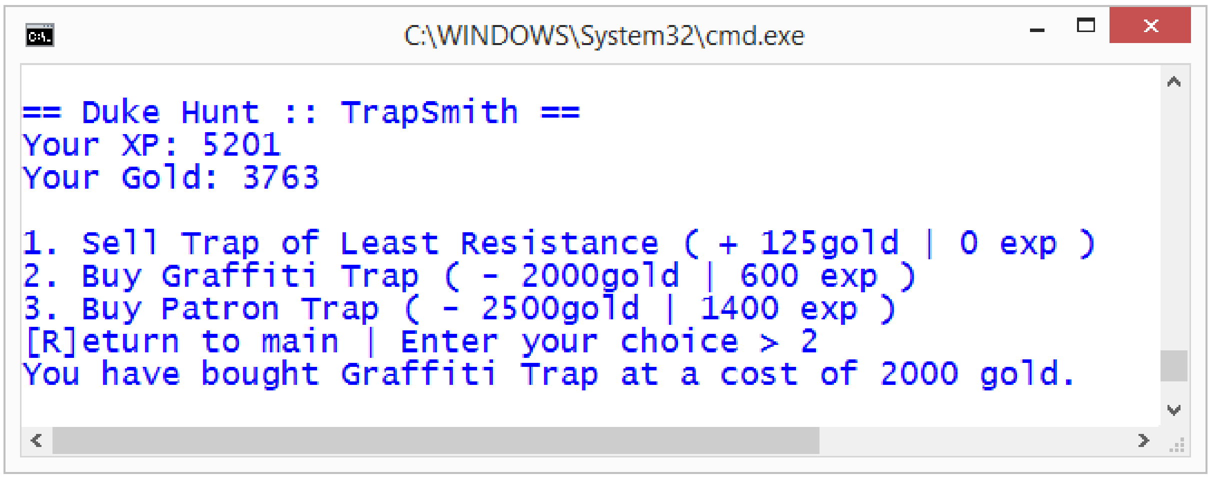
```
== Duke Hunt :: TrapSmith ==
Your XP: 292
Your Gold: 675

1. Sell Trap of Least Resistance ( + 125gold | 0 exp )
2. Buy Graffiti Trap ( - 2000gold | 600 exp )
3. Buy Patron Trap ( - 2500gold | 1400 exp )
[R]eturn to main | Enter your choice >
```

If player do not have sufficient experience or gold, the system will not process the sale and the following message will be displayed. If player do not have sufficient experience, a corresponding message will be displayed accordingly.



```
== Duke Hunt :: TrapSmith ==
Your XP: 292
Your Gold: 675

1. Sell Trap of Least Resistance ( + 125gold | 0 exp )
2. Buy Graffiti Trap ( - 2000gold | 600 exp )
3. Buy Patron Trap ( - 2500gold | 1400 exp )
[R]eturn to main | Enter your choice > 3
G2T05 does not have sufficient gold!
```

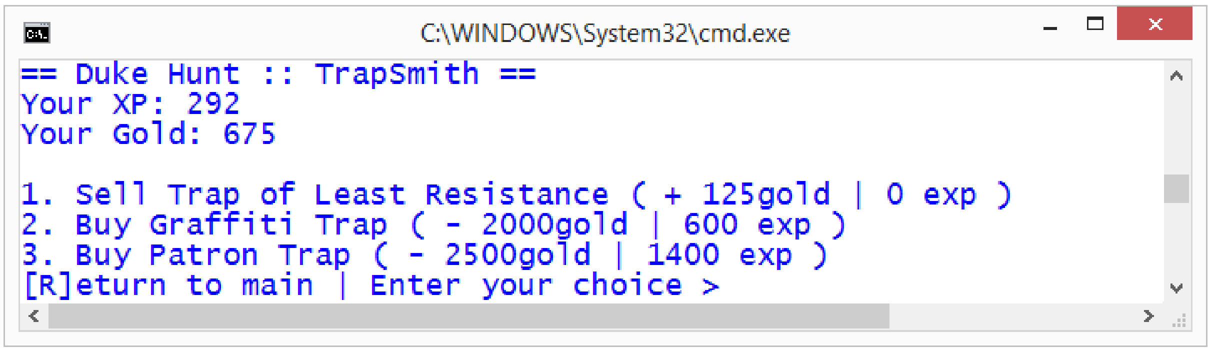
If the purchase is successful, the gold will be deducted.



```
== Duke Hunt :: TrapSmith ==
Your XP: 5201
Your Gold: 3763

1. Sell Trap of Least Resistance ( + 125gold | 0 exp )
2. Buy Graffiti Trap ( - 2000gold | 600 exp )
3. Buy Patron Trap ( - 2500gold | 1400 exp )
[R]eturn to main | Enter your choice > 2
You have bought Graffiti Trap at a cost of 2000 gold.
```

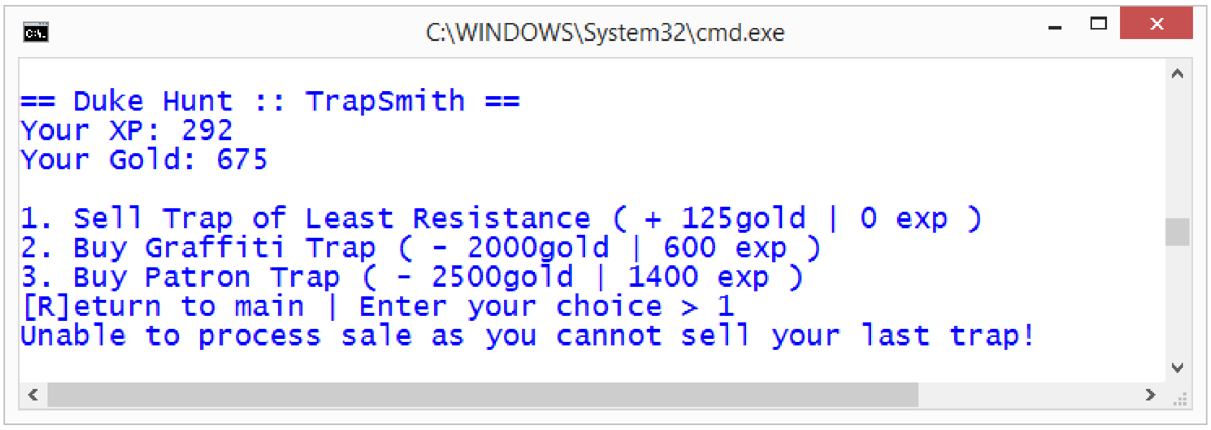
7.8. Sell Trap

Player can choose to sell trap by going to the trap smith. If a sell option is available, player will be able to sell trap.



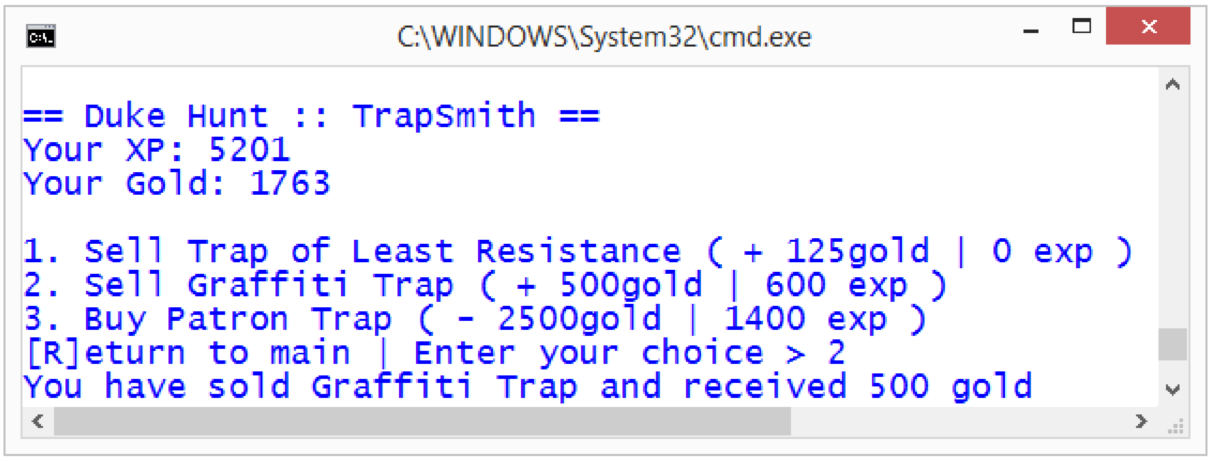
```
== Duke Hunt :: TrapSmith ==
Your XP: 292
Your Gold: 675

1. Sell Trap of Least Resistance ( + 125gold | 0 exp )
2. Buy Graffiti Trap ( - 2000gold | 600 exp )
3. Buy Patron Trap ( - 2500gold | 1400 exp )
[R]eturn to main | Enter your choice >
```

However, Player cannot sell the last trap. At least one trap must be left in the inventory or under equipped for trap smith to process the sale.



```
== Duke Hunt :: TrapSmith ==
Your XP: 292
Your Gold: 675

1. Sell Trap of Least Resistance ( + 125gold | 0 exp )
2. Buy Graffiti Trap ( - 2000gold | 600 exp )
3. Buy Patron Trap ( - 2500gold | 1400 exp )
[R]eturn to main | Enter your choice > 1
Unable to process sale as you cannot sell your last trap!
```

If player has at least one trap left after trap smith sale, the sale will be process and player will have the gold debited into his account.



```
== Duke Hunt :: TrapSmith ==
Your XP: 5201
Your Gold: 1763

1. Sell Trap of Least Resistance ( + 125gold | 0 exp )
2. Sell Graffiti Trap ( + 500gold | 600 exp )
3. Buy Patron Trap ( - 2500gold | 1400 exp )
[R]eturn to main | Enter your choice > 2
You have sold Graffiti Trap and received 500 gold
```

Player can sell the trap he is equipped with. In that case, the equipped trap will be changed to null.

8. Object-Oriented Design Considerations

In the designing of the DukeHunt project, we utilize the *Single Responsibility Principle* for all our classes. Essentially, we try to limit to only one role per class. For instance, under TrapManager, we designed it in such a way that the manager object add, modify and retrieve Traps from the trap.csv.

Also, we utilize multiple controllers to handle the various functionalities that is required for the project. For instance, TrapSmithController handles only the functions that are required for trading of traps. For hunting ground, there will be a *HuntingGroundController* to handle the functionalities that are required by Hunting Ground Menu. In this way, by adhering to the Single Responsibility Principles (SRP), it makes the codes easier to maintain and comprehend. Also, by splitting the controllers into various functionalities, it makes it easier to debug and allocate the workload.

Also, we have tried to abstain from needless complexities in our codes. As Leonardo Da Vinci once said, "Simplicity is the ultimate sophistication". This is true for our DukeHunt project as well. We keep the design principles as minimal as possible and cut away from needless complexities. For instance, when designing the travel class, we were conflicted if we should make a new class for Region (i.e Region.class). However, since region will only carry a String, we decided to do away with this complexity and decided to represent location as a String instead. In this way, we cut down on needless complexities and strive to keep our codes elegant.

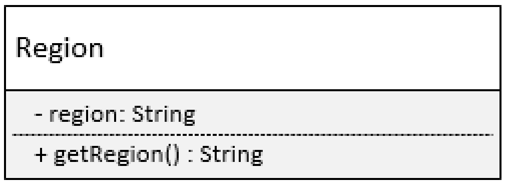


Figure 1:Earlier Design of Region.class results in Needless Complexity

9. Miscellaneous

9.1. Functionalities Dropped

No functionalities are dropped. All requirements made in the wiki page are met.

9.2. Challenges

One of the major challenges we had was to comply with the Object Orientated (OO) concepts. Previously, in IS200, SL 275, we only had to code according to what was required. However, in OOAD, the 4 concepts of OO was a major factor we had to consider when we had to design everything from the start. Therefore, we applied each of the following in their respective ways:

1. Modularity –We grouped methods of the same or similar purpose together into one class and adhered to the Single Responsibility Principle. Each class has a specific purpose and the methods are broken down to serve only one purpose. We passed the relevant objects to the relevant classes that require it.
2. Abstraction – We removed unnecessary classes and attributes. For example, we took away region object as there was not much of a need since it was just a value itself and used it as a String. By doing so, we only kept the relevant information needed for the player to determine his/her location.
3. Encapsulation – We separated the variables like location and used a method to retrieve or append the location each time instead of making location accessible to all. When a class is updated, with encapsulation, other classes will also likewise be updated and avoid inconsistent behaviour.
4. Hierarchy – We extended the Exception class and came up with our own exceptions to better suit the needs of our functions and to allow the user to more specifically understand what happen when a problem occur.

Another major challenge faced was the lack of knowledge when we first started the project. With little knowledge we had to apply whatever we knew and change along the way, making modularity and important factor as we could only add in parts we learn along the way.

9.3. Takeaway Lessons

From this project, we have learnt a lot to make ourselves a better programmer.

Firstly, we drilled ourselves much on java through the designing and redesigning process. The iterative process was useful for us to break components up to work on, to review if the previous part could work with the current one.

Following, we worked together better as a team, spending much time together helping each other when someone else need help. As the project is demanding with lots of deliverable, we learnt to leverage on each other's strength and allocate workload appropriately. Through this, we learnt about what it means to work in a team.