Deliverable #3

iSpaceship
Group 4, T02

Pedram Yazdinia, yazdinip
Will Conry, conrywm
Torja Istiaque, istiaqum
Akila Kavisinghe, kavisina
Xiangxin Kong, kongx9

April 8, 2020

1 Introduction

1.1 Purpose

The purpose of this document is to define the flow of control between modules and classes, while also highlighting the implementation details for each class. This has been done through the use of various UML diagrams, including State Chart Diagrams, Sequence Diagrams, and Class Diagrams. Each diagram provides high level information about different pieces of the application. The intended audience for this document includes direct and indirect stakeholders, primarily Dr. Khedri, Andrew Le Clair, and the testing team.

1.2 System Description

"iSpaceship" is a Rogue-like, turn-based, spaceship battle simulator, video game. The software product will provide the user with an engaging gaming experience where they can build their own spaceship and battle other spaceships. The software product will be used for the enjoyment of the user and provide them with a sense of self-accomplishment. The objective is for users to have fun and feel rewarded when they put in the time and dedication to progress in the game. The application should hold interest of users over a long period of time.

1.3 Overview

This document is divided into four sections, including the above Introduction section. In section 2, the state charts are given. The state diagrams show the program at different states and the properties of the attached transitions which include the event, guard (the condition), and the action. In section 3, the sequence diagrams are described. The diagrams in this section describe the communication between different classes, each diagram is derived from a use case. Finally in section 4, a detailed class diagram of the entire project is provided. This section outlines the implementation details for each class which includes the attributes, public and private methods, as well as the relation between the classes.

2 State Charts for Controller Classes

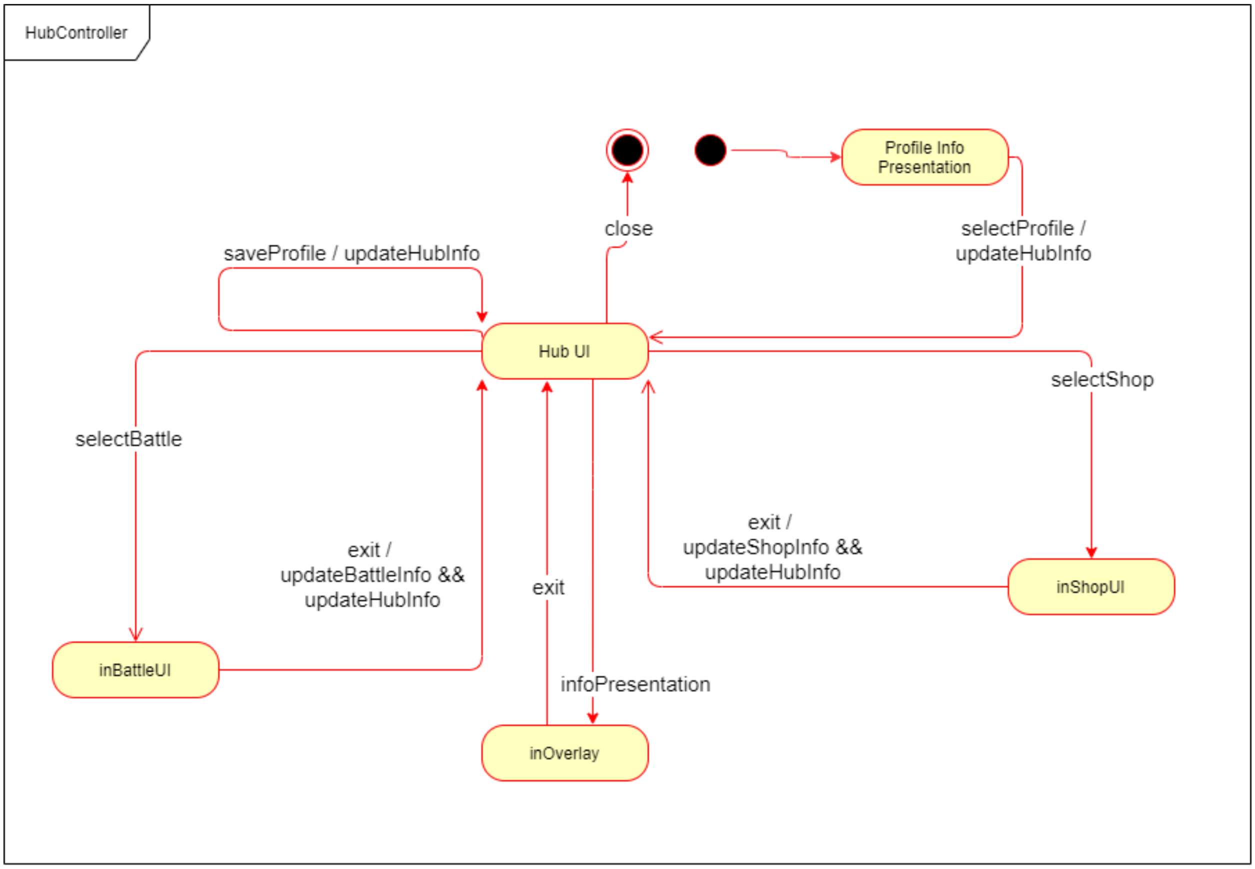
HubController
Profile Info Presentation
close
saveProfile / updateHubInfo
selectProfile / updateHubInfo
Hub UI
selectShop
selectBattle
exit / updateShopInfo && updateHubInfo
exit / updateBattleInfo && updateHubInfo
exit
inShopUI
inBattleUI
infoPresentation
inOverlay

Figure 1: State Diagram for Hub Controller Class.

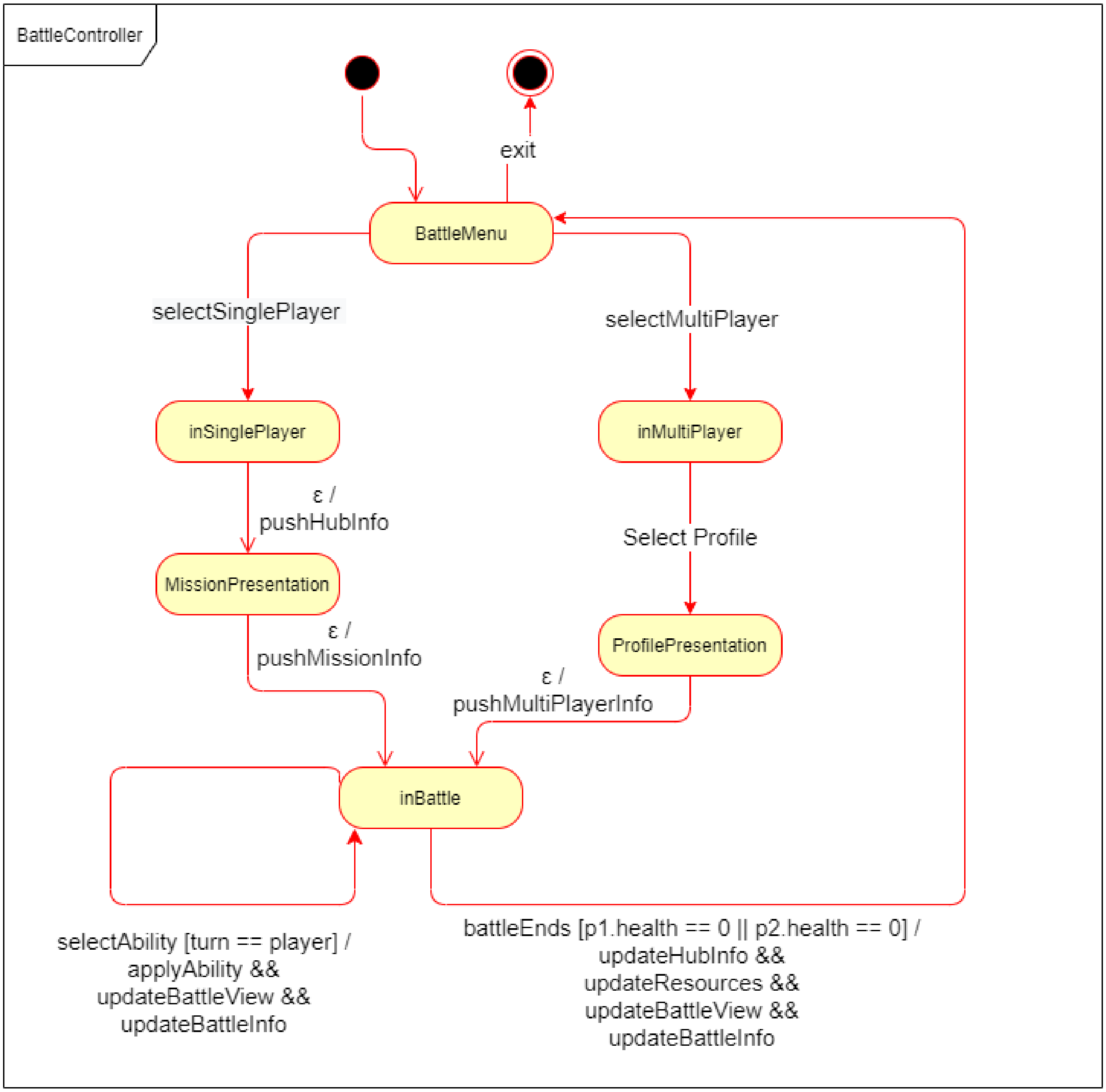

Figure 2: State Diagram for Battle Controller Class.

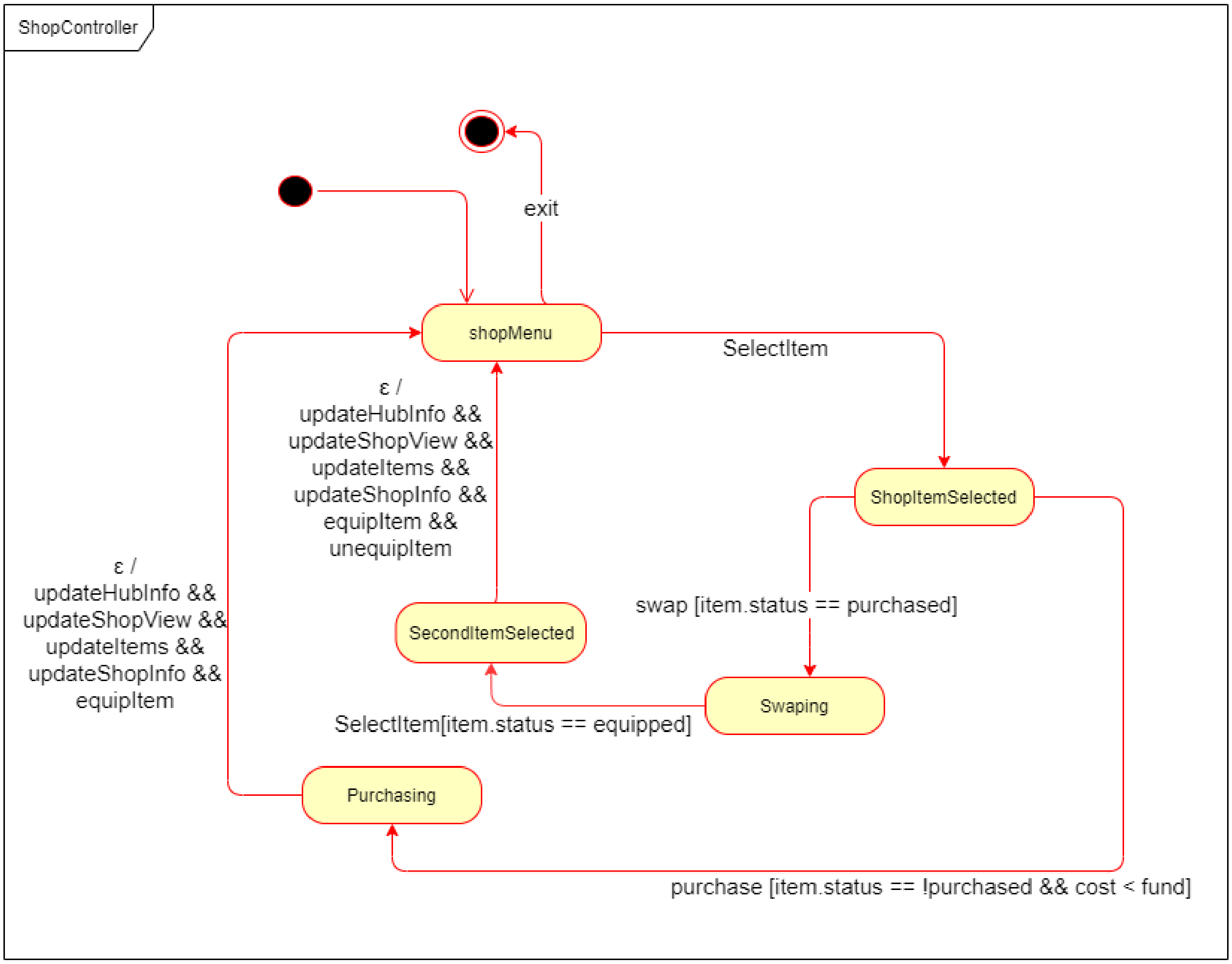
ShopController
exit
shopMenu
SelectItem
ε / updateHubInfo && updateShopView && updateItems && updateShopInfo && equipItem && unequipItem
ShopItemSelected
ε / updateHubInfo && updateShopView && updateItems && updateShopInfo && equipItem
swap [item.status == purchased]
SecondItemSelected
Swaping
SelectItem[item.status == equipped]
Purchasing
purchase [item.status == !purchased && cost < fund]

Figure 3: State Diagram for Shop Controller Class.

3 Sequence Diagrams

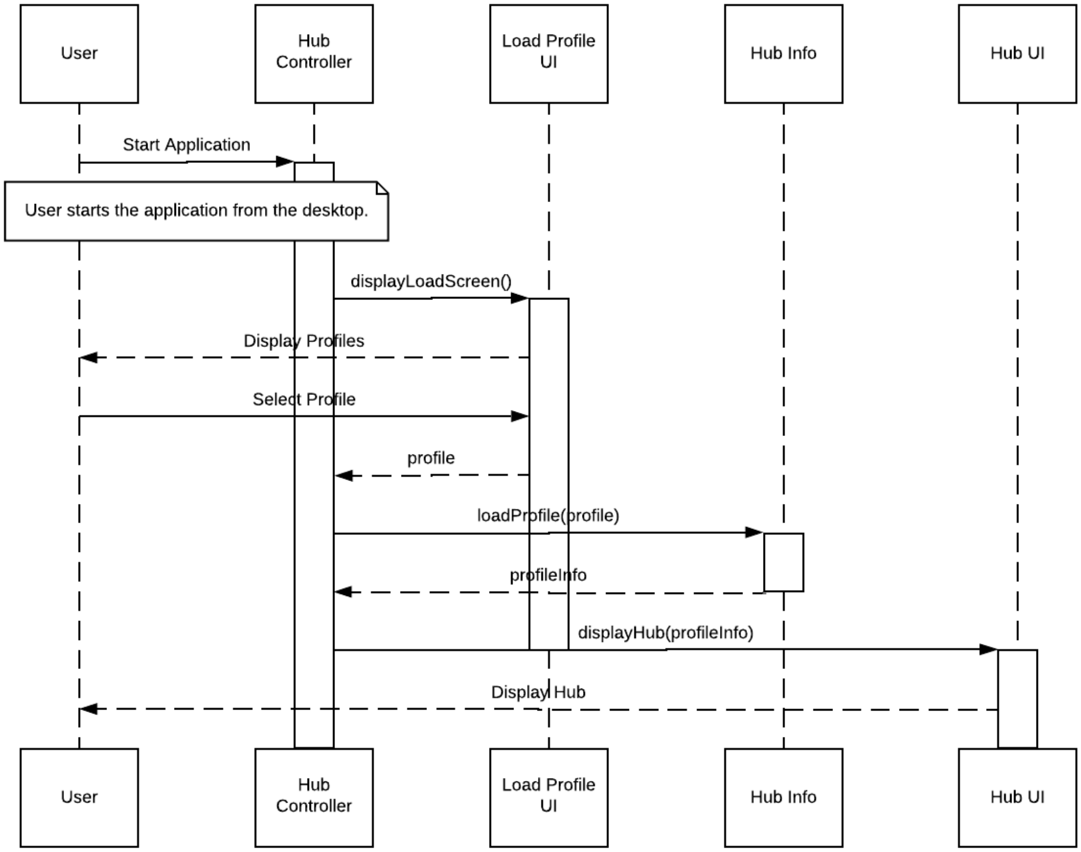

Figure 4: Sequence Diagram for the Load Profile use case.

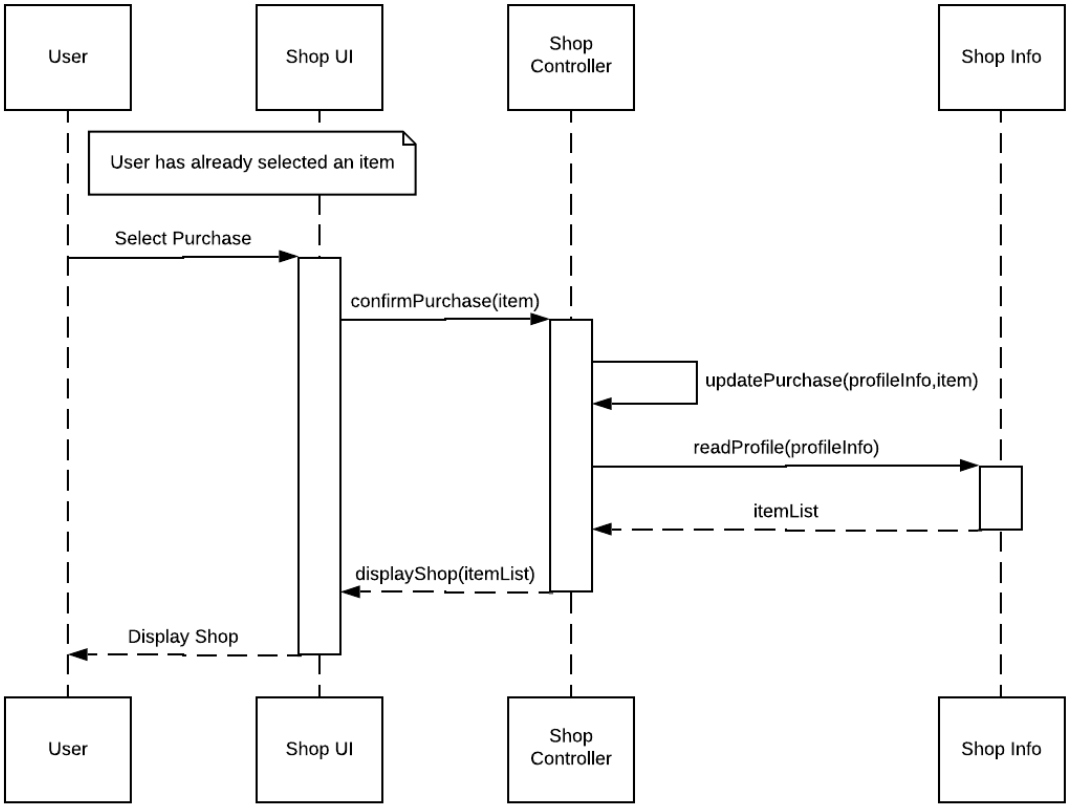

Figure 5: Sequence Diagram for the Purchase Item use case.

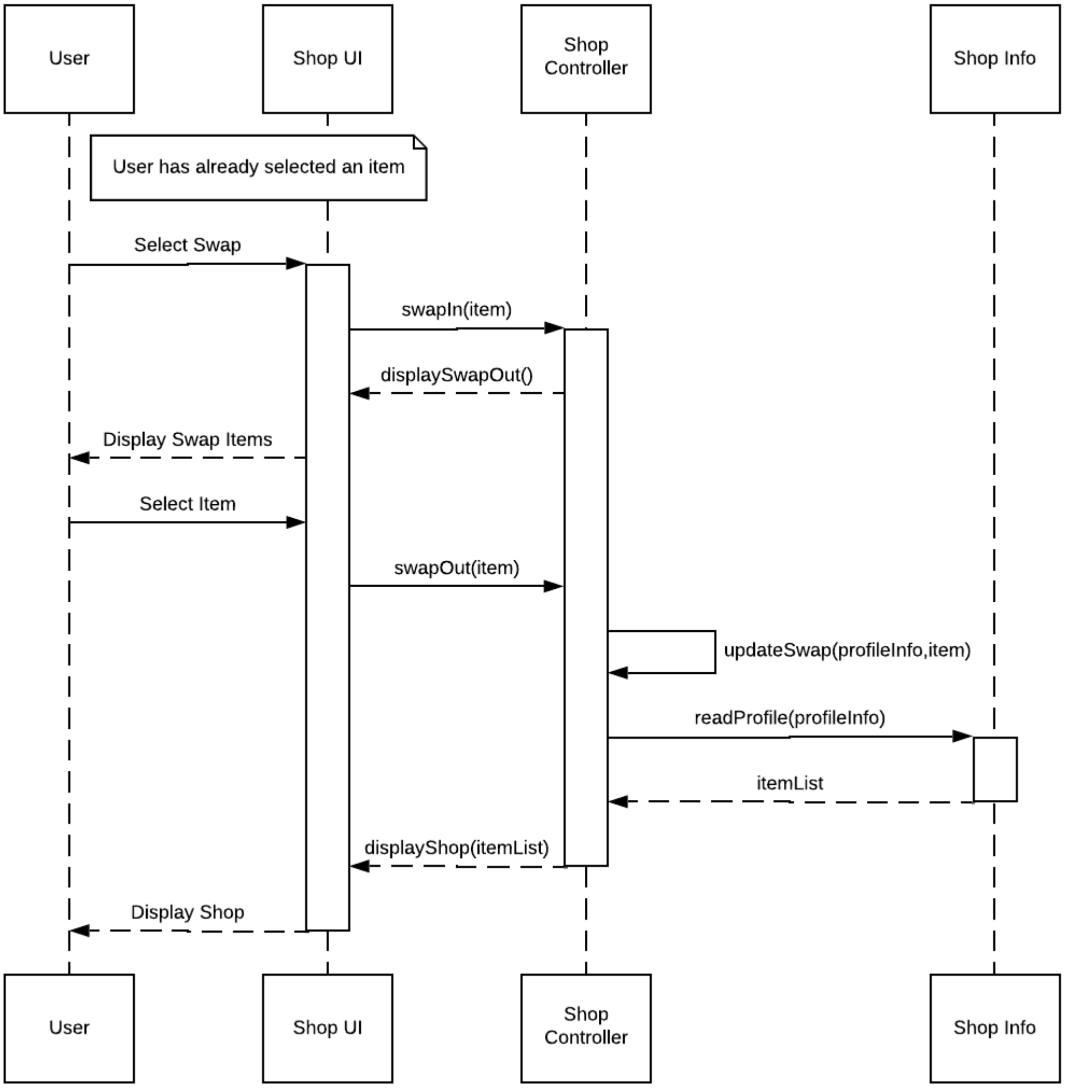

Figure 6: Sequence Diagram for the Swap Item use case.

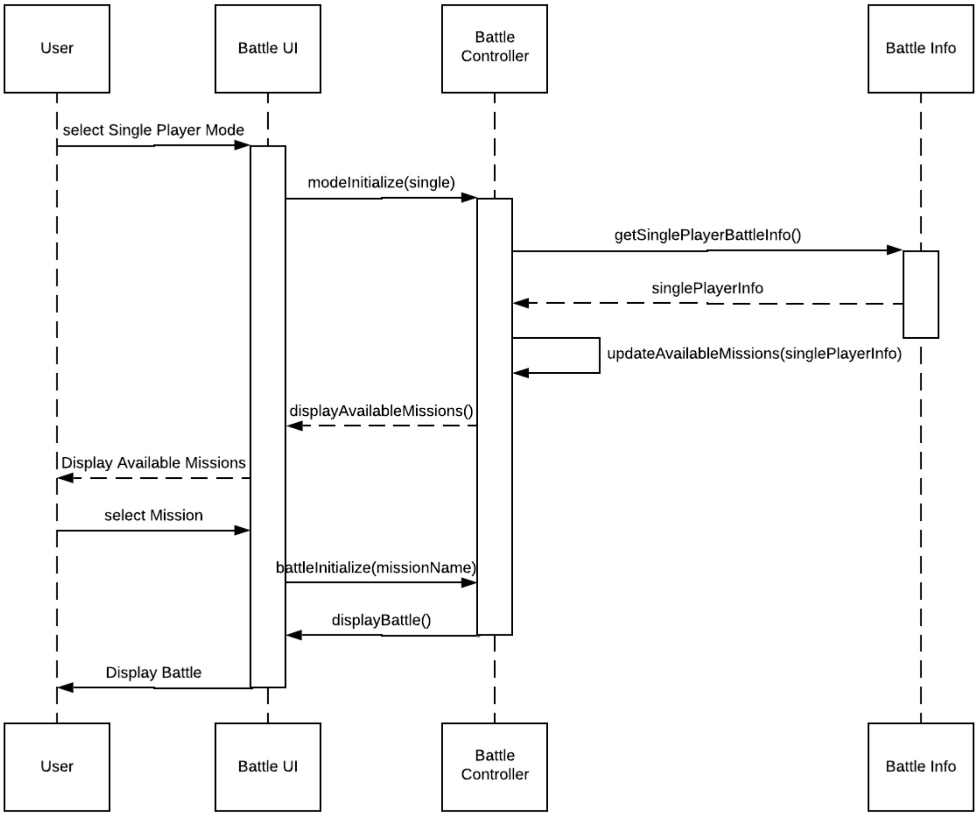
User
Battle UI
Battle Controller
Battle Info
select Single Player Mode
modeInitialize(single)
getSinglePlayerBattleInfo()
singlePlayerInfo
updateAvailableMissions(singlePlayerInfo)
displayAvailableMissions()
Display Available Missions
select Mission
battleInitialize(missionName)
displayBattle()
Display Battle
User
Battle UI
Battle Controller
Battle Info

Figure 7: Sequence Diagram for the Enter Single Player Battle use case.

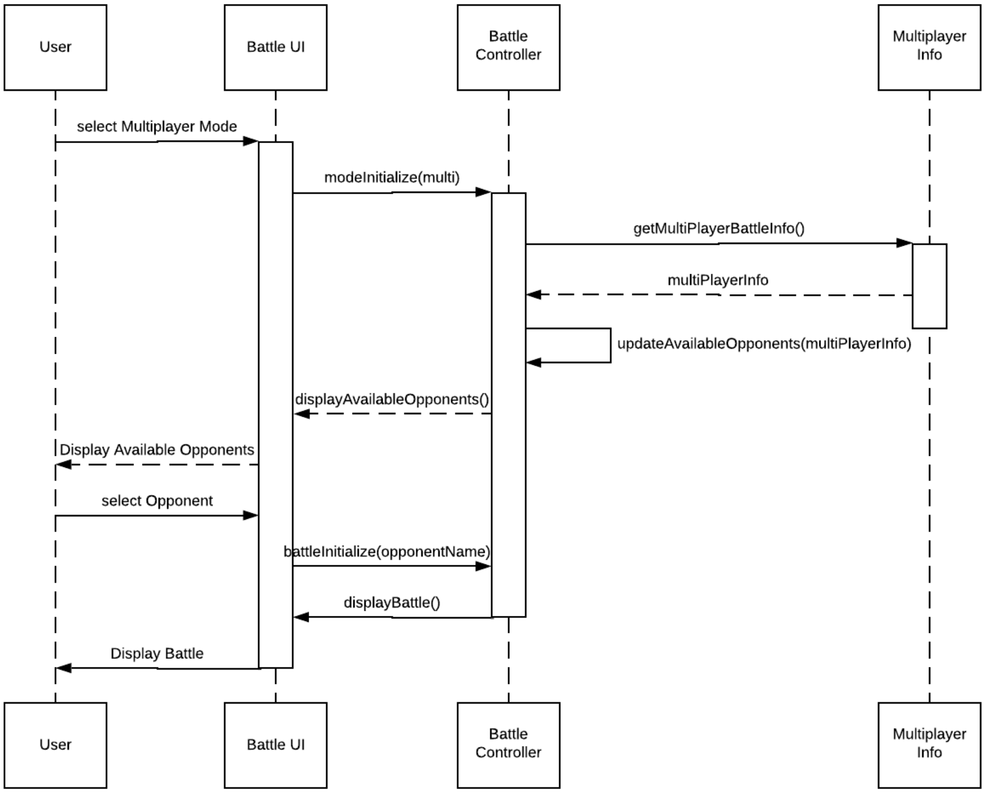

Figure 8: Sequence Diagram for the Enter Multiplayer Battle use case.

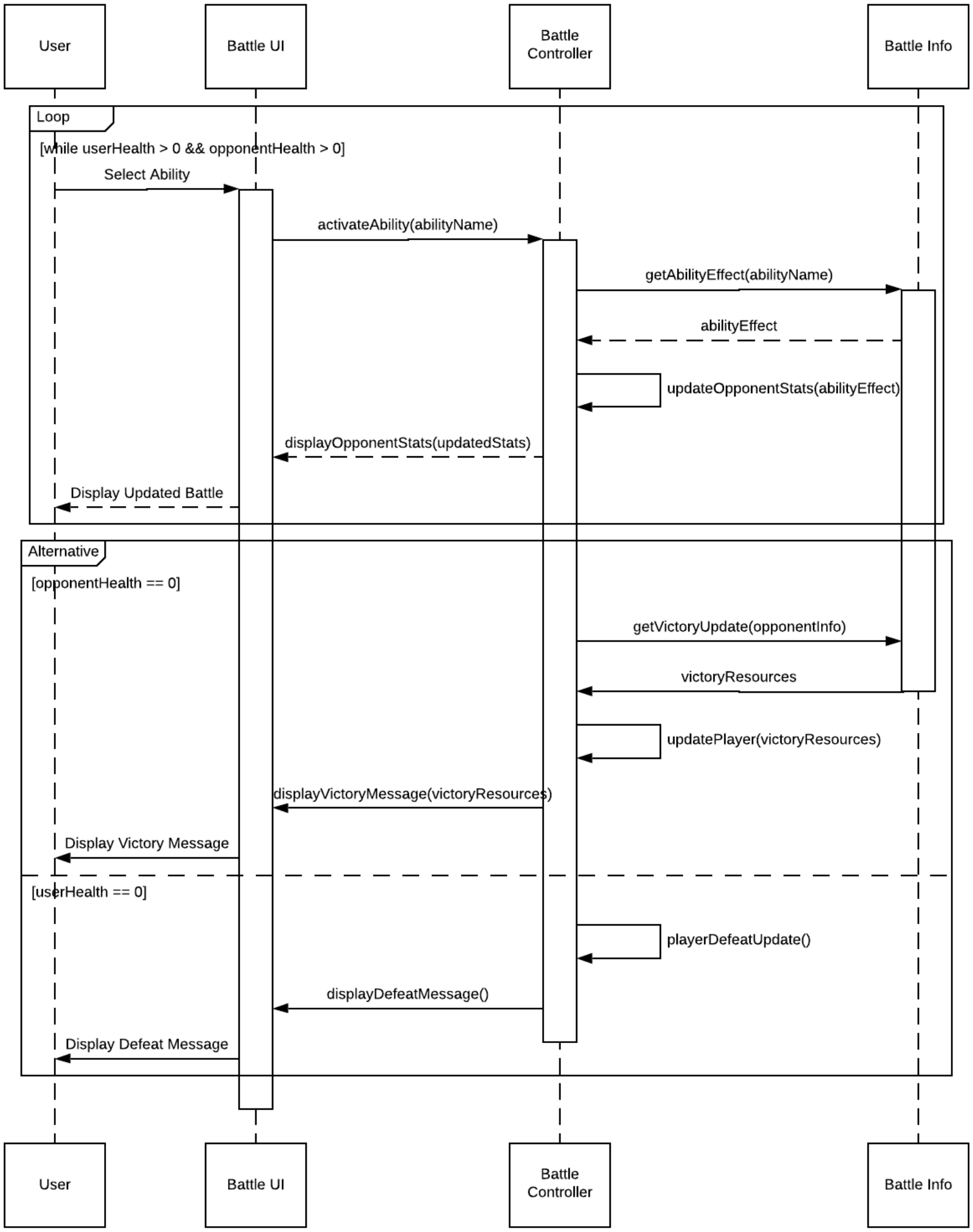

Figure 9: Sequence Diagram for the Select Ability use case.

4 Detailed Class Diagram

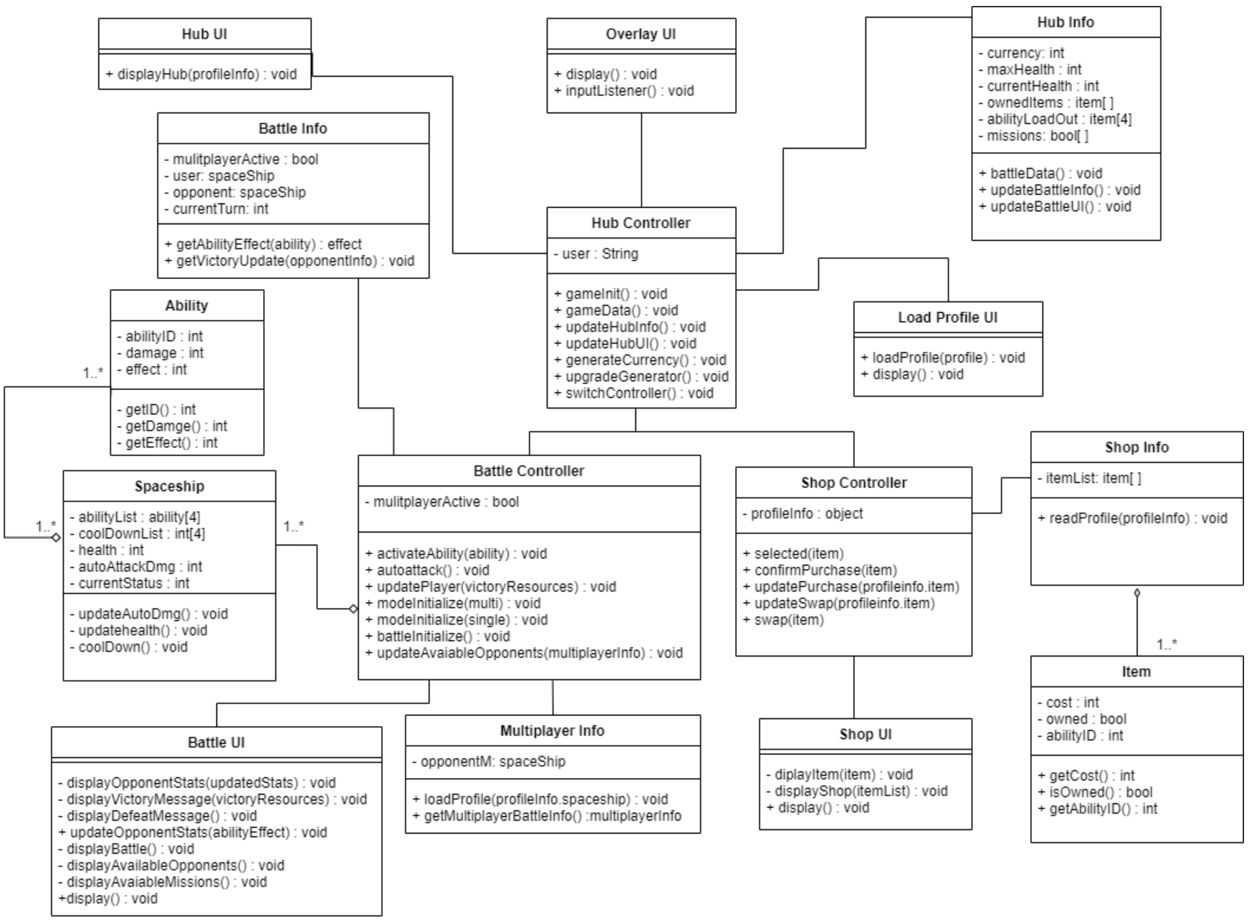
Hub UI
+ displayHub(profileInfo) : void
Overlay UI
+ display() : void
+ inputListener() : void
Hub Info
- currency: int
- maxHealth : int
- currentHealth : int
- ownedItems : item[ ]
- abilityLoadOut : item[4]
- missions: bool[ ]
+ battleData() : void
+ updateBattleInfo() : void
+ updateBattleUI() : void
Battle Info
- mulitplayerActive : bool
- user: spaceShip
- opponent: spaceShip
- currentTurn: int
+ getAbilityEffect(ability) : effect
+ getVictoryUpdate(opponentInfo) : void
Hub Controller
- user : String
+ gameInit() : void
+ gameData() : void
+ updateHubInfo() : void
+ updateHubUI() : void
+ generateCurrency() : void
+ upgradeGenerator() : void
+ switchController() : void
Load Profile UI
+ loadProfile(profile) : void
+ display() : void
Ability
- abilityID : int
- damage : int
- effect : int
- getID() : int
- getDamge() : int
- getEffect() : int
1..*
Spaceship
- abilityList : ability[4]
- coolDownList : int[4]
- health : int
- autoAttackDmg : int
- currentStatus : int
- updateAutoDmg() : void
- updatehealth() : void
- coolDown() : void
1..*
1..*
Battle Controller
- mulitplayerActive : bool
+ activateAbility(ability) : void
+ autoattack() : void
+ updatePlayer(victoryResources) : void
+ modeInitialize(multi) : void
+ modeInitialize(single) : void
+ battleInitialize() : void
+ updateAvaiableOpponents(multiplayerInfo) : void
Shop Controller
- profileInfo : object
+ selected(item)
+ confirmPurchase(item)
+ updatePurchase(profileinfo.item)
+ updateSwap(profileinfo.item)
+ swap(item)
Shop Info
- itemList: item[ ]
+ readProfile(profileInfo) : void
1..*
Item
- cost : int
- owned : bool
- abilityID : int
+ getCost() : int
+ isOwned() : bool
+ getAbilityID() : int
Battle UI
- displayOpponentStats(updatedStats) : void
- displayVictoryMessage(victoryResources) : void
- displayDefeatMessage() : void
+ updateOpponentStats(abilityEffect) : void
- displayBattle() : void
- displayAvailableOpponents() : void
- displayAvaiableMissions() : void
+display() : void
Multiplayer Info
- opponentM: spaceShip
+ loadProfile(profileInfo.spaceship) : void
+ getMultiplayerBattleInfo() :multiplayerInfo
Shop UI
- diplayItem(item) : void
- displayShop(itemList) : void
+ display() : void

Figure 10: Class Diagram for the iSpaceship project.

A Division of Labour

Include a Division of Labour sheet which indicates the contributions of each team member. This sheet must be signed by all team members.

Approved: ______________________________
Group Member 1

Approved: ______________________________
Group Member 2

Approved: ______________________________
Group Member 3

Approved: ______________________________
Group Member 4

Approved: ______________________________
Group Member 5

IMPORTANT NOTES

- You do NOT need to provide a text explanation of each diagram; the diagram should speak for itself
- Please document any non-standard notations that you may have used
  - *Rule of Thumb*: if you feel there is any doubt surrounding the meaning of your notations, document them
- Some diagrams may be difficult to fit into one page
  - It is OK if the text is small but please ensure that it is readable when printed
  - If you need to break a diagram onto multiple pages, please adopt a system of doing so and throughly explain how it can be reconnected from one page to the next; if you are unsure about this, please ask me
- Please submit the latest version of Deliverable 1 and Deliverable 2 with Deliverable 3
  - They do not have to be a freshly printed versions; the latest marked versions are OK
- If you do NOT have a Division of Labour sheet, your deliverable will NOT be marked