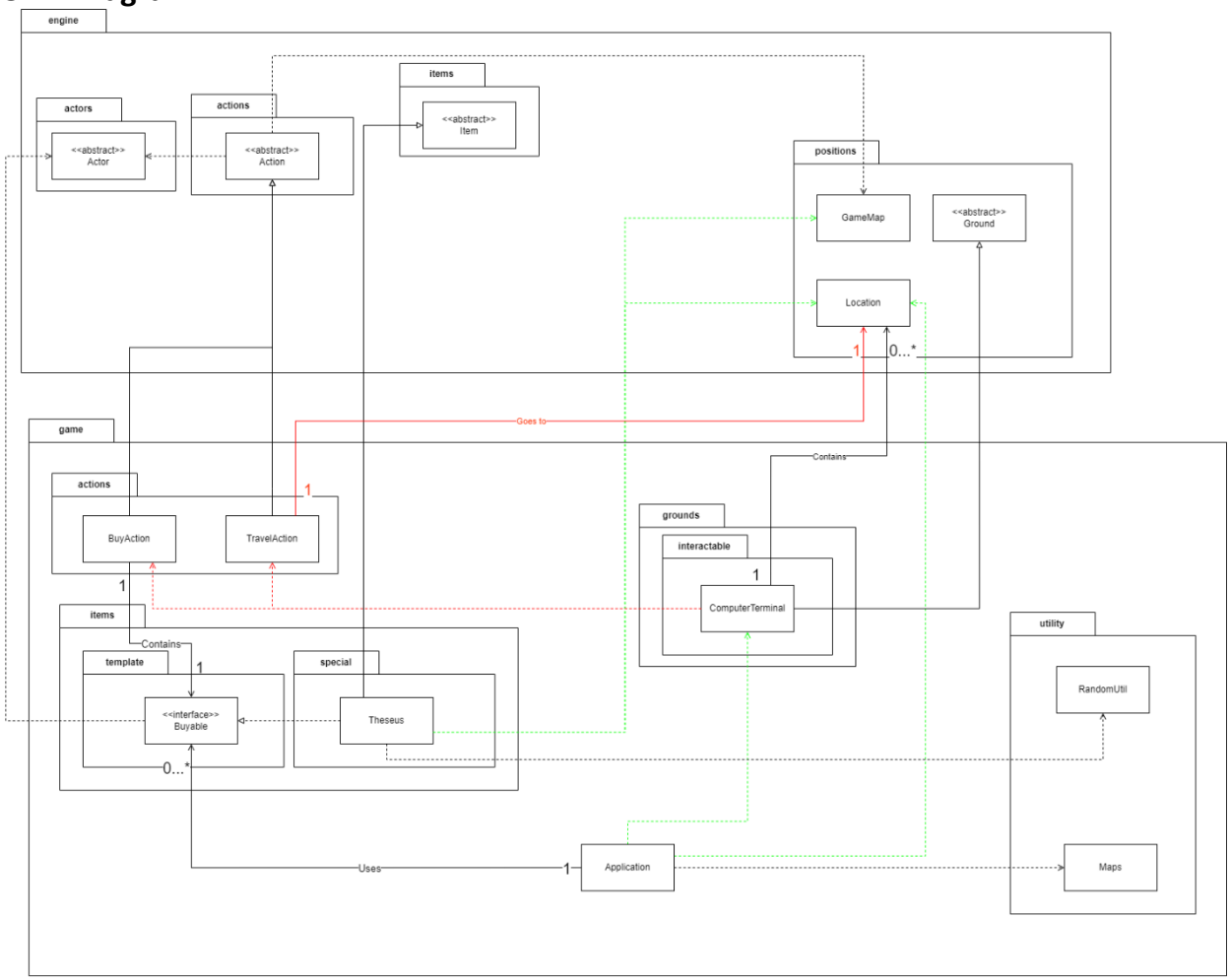
Assignment 3 Design Rationale

Requirement 1: The Ship of Theseus

UML Diagram:



Design Goal:

The design goal for this assignment is to is to implement new features for the game. Particularly a means for the Player or Actor to teleport and travel across maps and within the map. This method of travel will be incorporated into the ComputerTerminal class to allow ease of travel to other moons, planets or factories. In addition to that we will need to create a new item called Theseus, that when placed and stepped directly on top of, will allow the Player to travel anywhere within the map but randomly. Instructions however did specify that if the coordinates already contains an Actor, the travelling action will fail and return an ERROR message.

Other than the new implementation, we should also create a new class to store all the maps instead of letting Application handle it as to keep things more organised. We

would also like to refactor the Buyable items to be added at the application class instead of the Computer Terminal to allow it be more flexible and allow new instances of Computer Terminal to be able to have their own set of Buyable items.

Details of Theseus

Character Symbol	- '^'
Name	- "THESEUS"
Costs	- 100 credits
Special effect when purchased	- None
Purpose	- Give the option to teleport randomly within the map when stepped on

Computer Terminal 2.0

New features	- Will now be able to teleport to maps
Teleportable locations	- Connascence, Static Factory, Polymorphia

Design Decision:
In implementing [specific requirement or aspect of the assignment], the decision was made to [describe the chosen approach or solution]. [Explain the rationale behind the decision, including considerations of efficiency, scalability, maintainability, or other relevant factors].

In implementing Theseus and the new version of Computer Terminal, a new class called TravelAction class was created in order to promote code reusability and the single responsibility principle. Given that Theseus and Computer Terminal have different implementations for making the Player Travel, having the TravelAction class be solely focused on travelling would minimise the risk of being Connascence of Values. This is because if we were to let the classes to handle the movement of the actor themselves, if we wanted a different or more efficient implementation of travel we would need to refactor every single class instead of refactoring the TravelAction class itself.

This will of course lead to a Connascence of Type and/or Position instead, as now the input arguments should be in the correct type and position for the TravelAction.
As for how the Computer Terminal will store all of these destinations, we've implemented a Map to store Location and String. We recognize that this will lead to a Connascence of Values however this is acceptable as without the Location, the Computer Terminal will have no way to teleport the player to their appropriate position and this will adhere more closely to the SOLID principles and make it more flexible. Because an alternative approach would have definitely been to directly add the locations into the default constructor, but that would have violated the SOLID principles of Open Closed Principle.

The Open Closed Principle would've been violated in the sense that if you want a new instance of the Computer Terminal with different sets of coordinates or locations to teleport you will need to modify the new instance which violates it completely. Instead by having the Locations be stored inside an ArrayList outside of the ComputerTerminal and having the non-default constructor accept that ArrayList, will not only allow it to be extendable but flexible. While extendibility is definitely one of the pros for this implementation, we must also recognise the cons that this would mean every instance of the Computer Terminal would now require an ArrayList of coordinates to be instantiated and added into the Computer Terminal. This may violate the Don't Repeat Yourself principle. So the current implementation is acceptable as now it is very flexible for handling the teleportation.

One more alternative approach can be to create a new class that handles the cooridnates or Locations or actions along with a String to describe the action. That approach would have appealed to the Dependency Inversion Principle as now the Computer Terminal can take in all manner of actions based on the abstraction, in which this case we can call it the "StoreAction" class. However this was not done because it would have been over abstraction as the Computer Terminal should not contain an over abundance of Actions and make the system complicated. This also plays into the part of how real life players will not enjoy a game that has too many complicated systems in the game. This is an example of the code smell of message chaining, in the sense that the Computer Terminal now has to request the StoreAction object to request further methods to just to reach to the TravelAction to allow the player to teleport. This, in addition to how little code there will be inside the Computer Terminal, will make it harder for new junior developers to understand the and read the code. Therefore this approach was eventually scrapped as the efficiency of maintenance would be hard if more and more different data structures are used to store each of the actions and their

unique capabilities, therefore it will be more efficient to store it inside the Computer Terminal.

As for Theseus, it extends from the Item class and will be mainly calculating the random coordinates within the map and implements our Buyable interface. The approach to allow the player to teleport when stepped on top of, is to reuse the allowableActions(Location location), which only adds the ActionList to the player if they're directly above the item. As for alternative design, there is not much to say as it should not extend from any other abstract classes from the game engine. An argument could be made that the TravelAction should be the one doing the calculations for random travel within the game map but that would lead to it having unnecessary or unused methods for other Objects that may require TravelAction such as Computer Terminal. So it would be best to let the Theseus handle the calculations for the coordinates while TravelAction will be focused on the transportation.

As for current implementation of Theseus, we've done our best to limit the Connascence of Meaning that will arise from the allowableActions(Location location) method by having the TravelAction to take in the private static final string variable called MAP_NAME, instead of having the string directly be placed inside it's input arguments. This should reduce it to Connascence of Type and make it a bit more clear. While although Connascencec of Execution is relevant as the adding of TravelAction to the ActionList must not be called before getting the GameMap, the Rule of Degree is still within acceptable range as we only have 2 paremeters that the TravelAction takes compared to having 3 parameters if we were to implement the alternative method and allow the TravelAction to accept a third parameter such as a Boolean to check whether it wants to travel randomly. And even then, if we want to implement a random travelling that is restricted to a certain distance away from Theseus, that would cost even MORE refactoring. So with our current implementation of Theseus, it is extendable and flexible with it's usage while also focused solely on it's single responsibility.

As for preventing an incident where an Actor teleports/travels to a location that already has an Actor, the TravelAction's execute method will check for whether an actor already exists by reusing the containsAnActor() method and returning a String to show that the action failed if there is indeed an actor there.