Report
Group: W11-7

Changes Made:

- Create an abstract class Carrier that will be subclassed by Robot and Team
    - This allows Polymorphism, especially with regard to the .step() function which is called on all “Carriers” (as we have called them), also provides high cohesion, low coupling and protected variation
    - Automail now aggregates Carriers instead of Robots
- The responsibility to decide on whether to use a Team to deliver an item and the responsibility of creating that Team is done inside the MailPool class by Information Expert and Creator patterns
- Functionality of Team
    - Changes state and destination floor of Robots by Information Expert (as it aggregates Robots and contains the delivery item)
    - New Robot state - TEAMING to control for when Robot is in a team
        - This allows a Team to take control of its Robots separate from the Robot class, resulting in higher cohesion of the Team class.
    - Make the team break up after delivery
        - Teams aggregate the group of Robots they manage, thus, the Team is given the responsibility of breaking itself up by Information Expert
    - Separated max weight constants between Robot and Team class, in order to increase cohesion
    - A Robot’s responsibility in our design is cohesive since it simply moves towards a floor and potentially delivers, whereas to make the Robots operate as a team without a controller class Team, the Robots would need to perform “Team” operations and this would decrease cohesion of the Robot class and

introduce coupling between robots (of which in our design/implementation none exists)

- Mailpool now needs a reference to Automail in order to modify the list of carriers (adding and removing teams), as .step() is called on every “Carrier” in this list
    - Increases coupling, although Automail already had a reference to MailPool
    - loadRobot() name changed to loadCarrier() as it can “load” either a Robot with a delivery item, or in the case of a heavier item, “load” a team -> increase readability
- removeFromAutomail method included in MailPool class to allow a Team to remove itself from the list of Carriers within Automail. This isn’t ideal but the alternative was to provide team with a reference to Automail so it could remove itself. However this increases coupling to the Automail class. MailPool already has a reference to Automail and thus should be assigned the responsibility of removing and adding to the list of Carriers.
- MailPool contains only the Robots that are in the WAITING state (as is also in the original implementation), this maintains a low representational gap between domain and design/implementation. This is because, in reality, a MailPool is a physical room, it does not make sense for it to track robots not in that room. Additionally a Team shouldn’t be stored in the MailPool list since a Team is never in a WAITING state.
- There is unavoidable coupling between MailPool, Carrier and Automail, due to the nature of Simulation, which requires access to all carriers in one way or another, in order to call .step(). Specifically, MailPool needs access to Automail to add the created teams to the list of carriers (since this is where simulation calls .step() for all the carriers). Automail needs access to the Carriers so it can provide Simulation a way to call .step() for all of them. Carriers need access to the MailPool so they can register themselves as waiting and to (in the case of teams) remove themselves from Automail.

Alternative Solutions Considered:

- An alternative “Team” implementation was to let the mailpool allocate the same delivery item to multiple robots and set their state to TEAMING which would indicate to only step() on every third call. This was rejected because it doesn’t involve linking the robots in a meaningful way and relies on the robots independently performing identically. Allowing a controller class (team) to exist ensures that these robots are moving in unison. Also allows for extension of team functionality (protected variation).
- Another alternative of “Team” that was considered is to have everything as a Team, and allow teams of one Robot, where the logic of speed of movement would be controlled depending on the number of Robots in the Team. This solution, however, increases the representational gap as a class Robot, an important part of the domain space, does not appear in the design/implementation
- We considered adding the control of all “Carriers” (Teams and Robots) to MailPool, but decided on current implementation as it results in a lower representational gap.

Assumptions:

Domain Model

- In domain model show that robot can carry 2 items
    - In design model, distinguish between hand items and tube items as attributes
- Team is included in the domain model only because Robots are going to perform as Teams, and while Team is “abstract” in that it’s purely conceptual, it is a part of the domain, since Robots can act in Teams

Design Model:

- Automail included in Design Class diagram, because it now takes a list of carriers, not just robots i.e. it was changed, therefore ought to be included
- Capitalized attributes => final i.e. Constants

Sequence diagram:

- Assume next item in mailpool is 2600 in weight (so a team of 2 robots is required)
- Step counter not included in diagram, because very specific to the fact Teams move slower. I.e. is not a part of creating, delivering and disbanding teams
- In the case where .step() is being called on a Team, it's assumed that step is called until the delivery is made i.e. currentFloor = destinationFloor. This is because .step() is called in Simulation