4 Orion Expedition

4.1 Additional functionality

As the Orion space shuttle ventures deeper into the cosmos, NASA has developed advanced systems to enhance its exploration capabilities. Each destination (celestial body) now includes a scientific value metric, observation completion percentage, and specific fuel consumption rate.

The navigation system now features a navigation mode with three distinct strategies to interstellar travel. Standard mode takes the direct approach found in the original design. Efficient mode factors fuel consumption into routing decisions by calculating a fuel-to-distance ratio for each destination. Exploration mode chooses destinations with the highest value, sometimes choosing longer paths deliberately to study cosmic phenomena of interest.

4.2 State diagram

The shuttle's operational behavior is now governed by a state model that modifies its functionality. Initially, the shuttle is in the docked state, where it can refuel but cannot move as the engine remains inactive. When the engine starts, it transitions to cruising state, traveling at standard velocity by consuming fuel per move and increasing travel progress by 5%. Upon reaching a destination, the shuttle enters orbiting state, performing observations with reduced fuel consumption at 3%. If fuel levels drop critically low, the shuttle transitions to emergency state, where it stops the observation and attempts to refuel. If refueling succeeds, the shuttle returns to the cruising state; if unsuccessful, the engine stops, and shuttle returns to docked state.

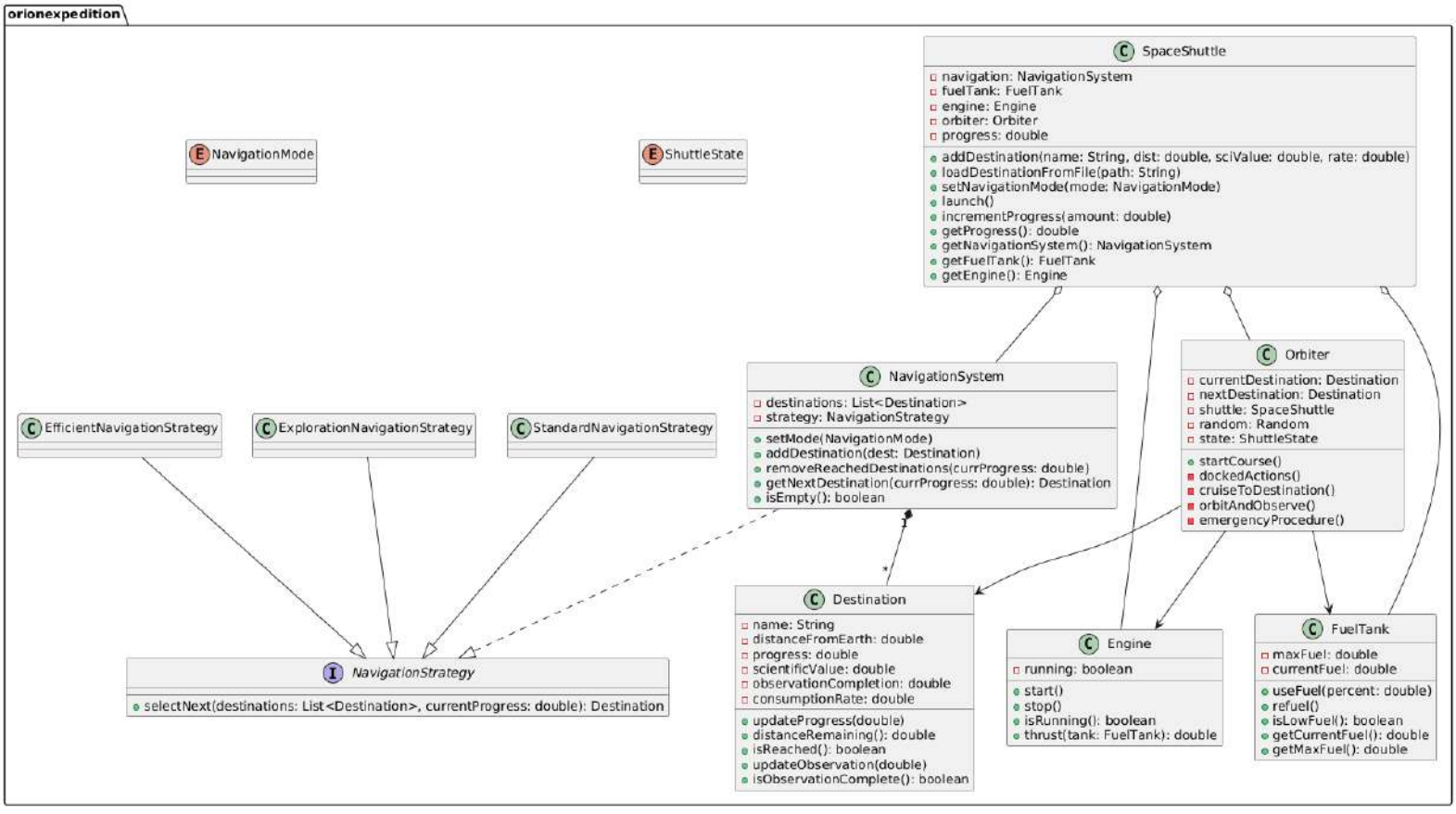
4.3 Scenarios to model with sequence diagrams

4.3.1 Scenario 1: Successful Planetary Visit

When the shuttle goes to a new destination, the navigation course begins with the selected destination. The engine starts, and the shuttle enters cruising state. As the shuttle advances, its movement gradually increases the destination's travel progress until it reaches 100%. Once complete, the current destination is updated, the destination is removed from the navigation system, and the closest destination becomes the next target using the active navigation strategy.

4.3.2 Scenario 2: Fuel Emergency Management

When the fuel drops below 15% of capacity, the shuttle enters Emergency state. The system attempts refueling with a 50% success chance. If successful, the fuel is restored to maximum, and the mission continues. If unsuccessful, the engine stops, halting progress until the situation is resolved. The shuttle can implement an enhanced emergency protocol that enables solar energy collection in deep space, modifying the random refueling chance based on proximity to stars, or calculating a minimum-energy return trajectory to the nearest known refueling point by selecting the closest destination with refueling capabilities.


orionexpedition
NavigationMode
ShuttleState
SpaceShuttle
- navigation: NavigationSystem
- fuelTank: FuelTank
- engine: Engine
- orbiter: Orbiter
- progress: double
+ addDestination(name: String, dist: double, sciValue: double, rate: double)
+ loadDestinationFromFile(path: String)
+ setNavigationMode(mode: NavigationMode)
+ launch()
+ incrementProgress(amount: double)
+ getProgress(): double
+ getNavigationSystem(): NavigationSystem
+ getFuelTank(): FuelTank
+ getEngine(): Engine
NavigationSystem
- destinations: List<Destination>
- strategy: NavigationStrategy
+ setMode(NavigationMode)
+ addDestination(dest: Destination)
+ removeReachedDestinations(currProgress: double)
+ getNextDestination(currProgress: double): Destination
+ isEmpty(): boolean
Orbiter
- currentDestination: Destination
- nextDestination: Destination
- shuttle: SpaceShuttle
- random: Random
- state: ShuttleState
+ startCourse()
- dockedActions()
- cruiseToDestination()
- orbitAndObserve()
- emergencyProcedure()
EfficientNavigationStrategy
ExplorationNavigationStrategy
StandardNavigationStrategy
NavigationStrategy
+ selectNext(destinations: List<Destination>, currentProgress: double): Destination
Destination
- name: String
- distanceFromEarth: double
- progress: double
- scientificValue: double
- observationCompletion: double
- consumptionRate: double
+ updateProgress(double)
+ distanceRemaining(): double
+ isReached(): boolean
+ updateObservation(double)
+ isObservationComplete(): boolean
Engine
- running: boolean
+ start()
+ stop()
+ isRunning(): boolean
+ thrust(tank: FuelTank): double
FuelTank
- maxFuel: double
- currentFuel: double
+ useFuel(percent: double)
+ refuel()
+ isLowFuel(): boolean
+ getCurrentFuel(): double
+ getMaxFuel(): double
1
*

Destination

```
updateProgress(travelledDistance):
    progress = Math.min(travelledDistance,
                        distanceFromEarth)

distanceRemaining():
    return Math.max(distanceFromEarth - progress, 0.0)

isReached():
    return progress >= distanceFromEarth

updateObservation(percent):
    observationCompletion =
            Math.min(observationCompletion +
            percent, 100.0)

isObservationComplete():
    return observationCompletion >= 100.0
```

FuelTank

```
useFuel(percent):
    amount = (percent / 100.0) * maxFuel;
    currentFuel = Math.max(currentFuel - amount, 0.0)

refuel():
    currentFuel = maxFuel

isLowFuel():
    return currentFuel < (0.15 * maxFuel)

getCurrentFuel():
    return currentFuel

getMaxFuel():
    return maxFuel
```

Engine

```
start():
    running = true

stop():
    running = false

isRunning(){
    return running

thrust(FuelTank tank){
    if not running:
        return 0.0;
    tank.useFuel(10.0)
    return 5.0;
```

NavigationSystem

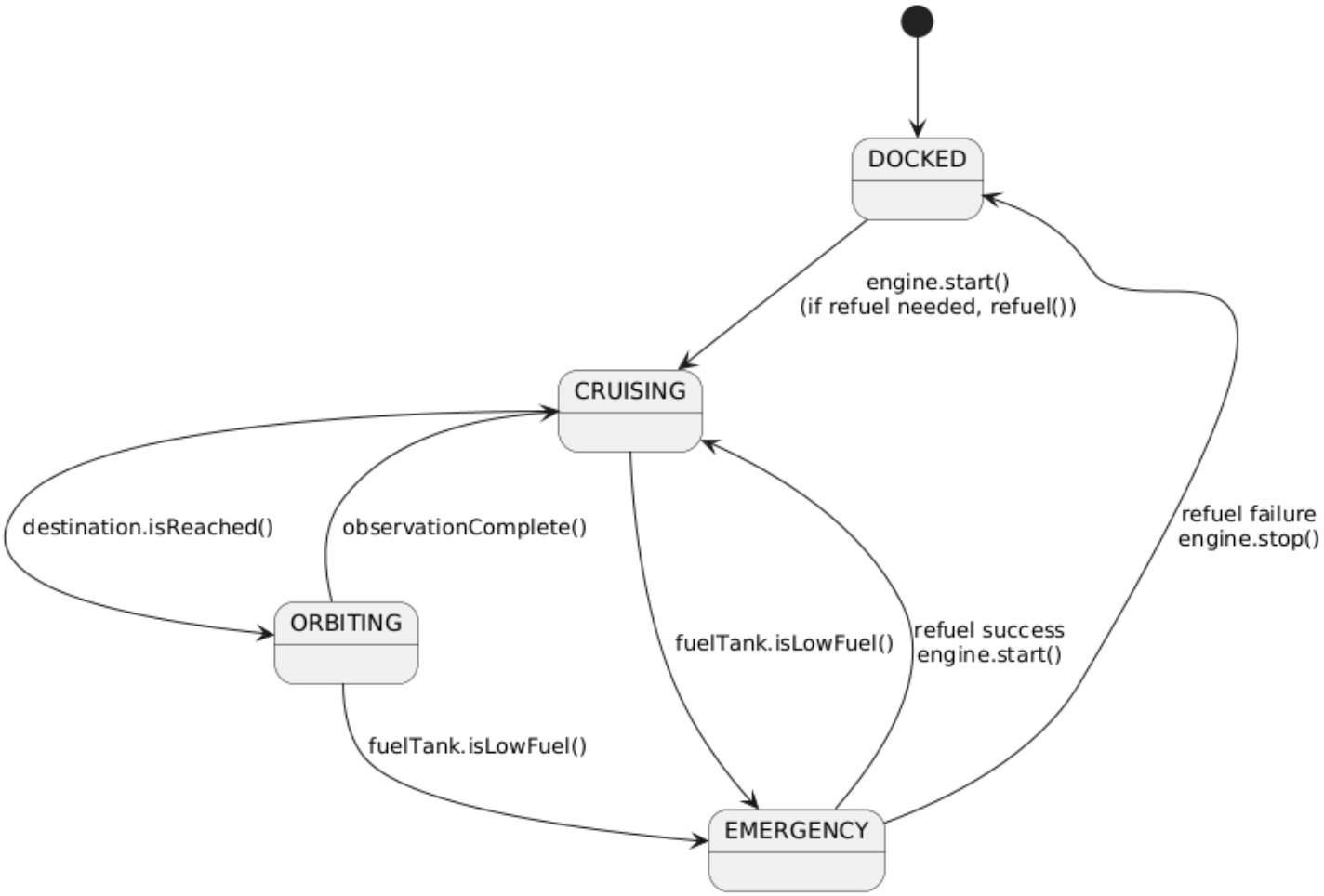
```
setMode(mode):
    switch (mode):
        case STANDARD:
            strategy = new StandardNavigationStrategy();
            break;
        case EFFICIENT:
            strategy = new EfficientNavigationStrategy();
            break;
        case EXPLORATION:
            strategy = new ExplorationNavigationStrategy();
            break;

addDestination(dest):
    destinations.add(dest)

removeReachedDestinations(currProgress):
    for d in destinations:
        d.updateProgress(currProgress)
    destinations.removeIf(isReached)

getNextDestination(currProgress):
    return strategy.selectNext(destinations, currProgress)

isEmpty():
    return destinations.isEmpty()
```

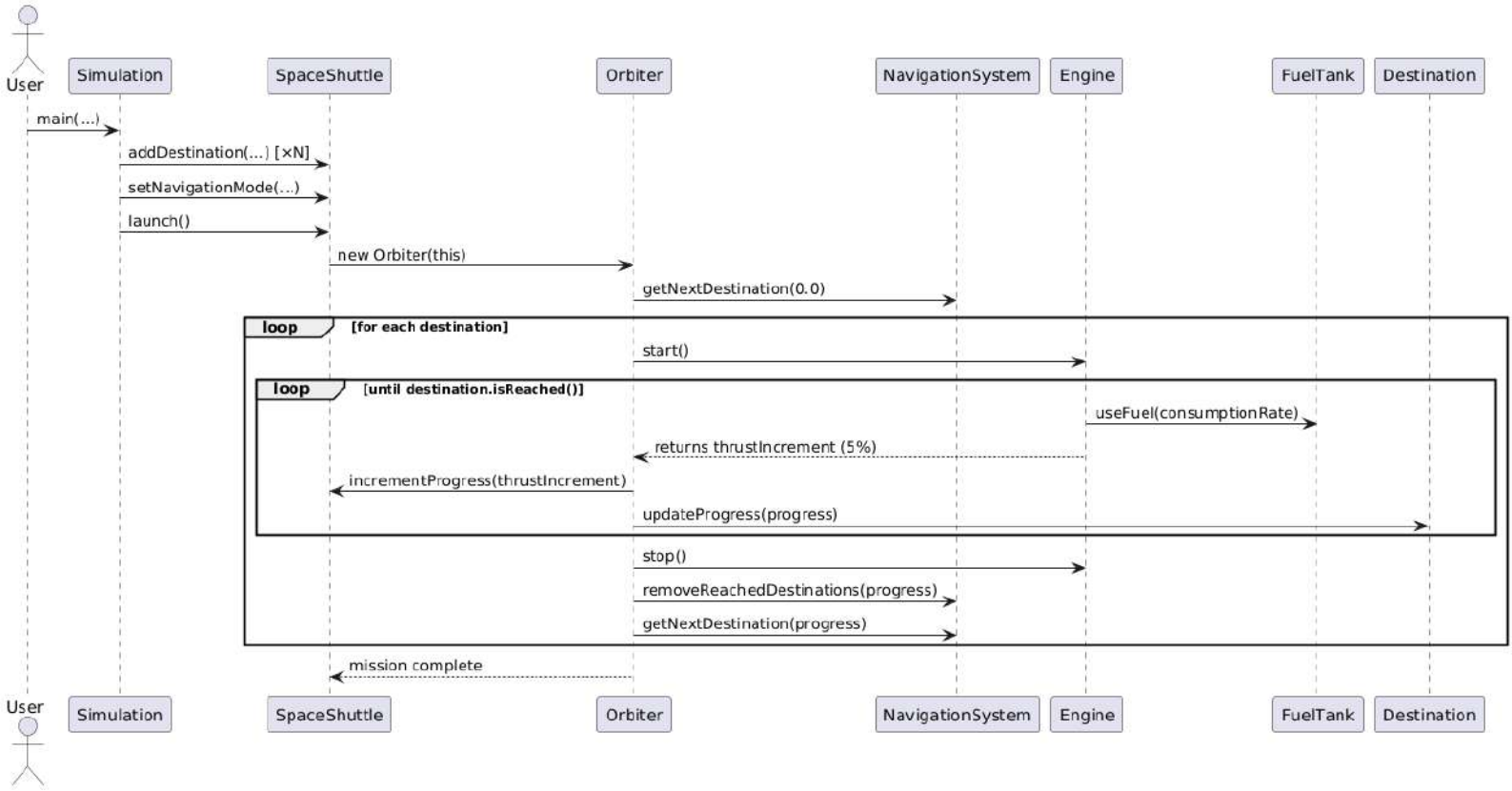
Orbiter

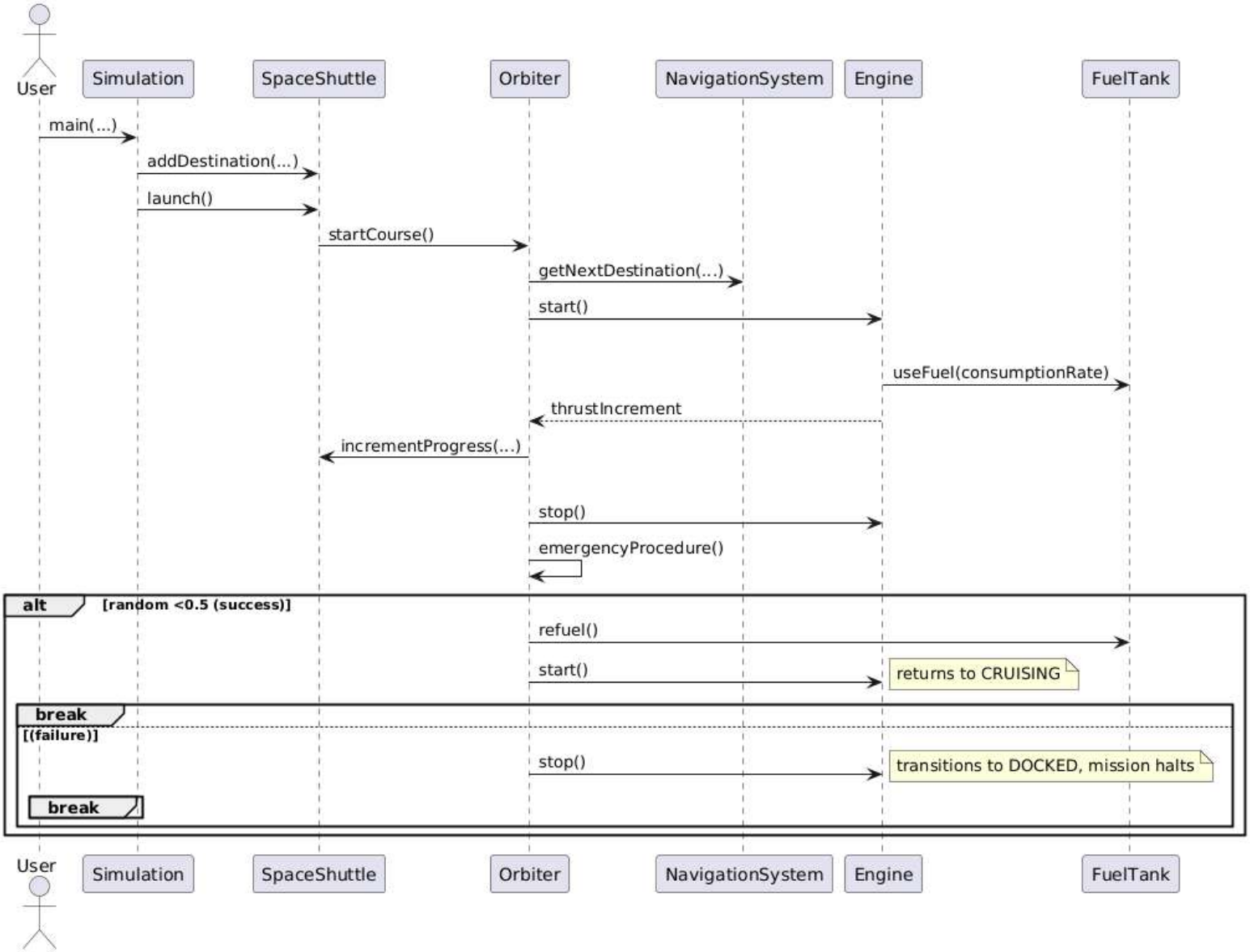
```
startCourse():
        while (nextDestination not equals null):
            dockedActions();
            if (state == ShuttleState.DOCKED):
                return
            cruiseToDestination();
            if (state equals ShuttleState.EMERGENCY and shuttle.getFuelTank().getCurrentFuel() <= 0):
                return
            if (state equals ShuttleState.ORBITING):
                orbitAndObserve();
                shuttle.getNavigationSystem().removeReachedDestinations(shuttle.getProgress())
                currentDestination = nextDestination
                nextDestination = shuttle.getNavigationSystem().getNextDestination(shuttle.getProgress())

    dockedActions():
        if (shuttle.getFuelTank().isLowFuel()):
            shuttle.getFuelTank().refuel()
        shuttle.getEngine().start()
        state = ShuttleState.CRUISING;

    cruiseToDestination():
        state = ShuttleState.CRUISING
        shuttle.getEngine().start()
        while (state equals ShuttleState.CRUISING);
            progInc = shuttle.getEngine().thrust(shuttle.getFuelTank())
            shuttle.incrementProgress(progInc)
            nextDestination.updateProgress(shuttle.getProgress())
            if (nextDestination.isReached()):
                shuttle.getEngine().stop()
                state = ShuttleState.ORBITING
                break
            if (shuttle.getFuelTank().isLowFuel()):
                shuttle.getEngine().stop()
                state = ShuttleState.EMERGENCY
                emergencyProcedure()

    orbitAndObserve():
        state = ShuttleState.ORBITING
        while (not nextDestination.isObservationComplete()):
            shuttle.getFuelTank().useFuel(3.0)
            nextDestination.updateObservation(5.0)
            if (shuttle.getFuelTank().isLowFuel()):
                state = ShuttleState.EMERGENCY
                emergencyProcedure()
                if (state not equals ShuttleState.CRUISING):
                    return

    emergencyProcedure():
        if (random.nextDouble() < 0.5):
            shuttle.getFuelTank().refuel()
            shuttle.getEngine().start()
            state = ShuttleState.CRUISING
        } else {
            state = ShuttleState.DOCKED
            shuttle.getEngine().stop()
```

SpaceShuttle

```
addDestination(name, distance, sciValue, rate):
    navigation.addDestination(new Destination(name, distance, sciValue, rate)

loadDestinationFromFile(filename) throws IOException:
    try (reader = new BufferedReader(new FileReader(filename))):
        line = ""
        while ((line = reader.readLine()) not equals null):
            parts = line.trim().split("\\s+")
            if (parts.length equals 2):
                name := parts[0]
                distance := Double.parseDouble(parts[1])
                defaultSciValue = 0.0
                defaultRate = 10.0
                addDestination(name, distance, defaultSciValue, defaultRate)
            else if (parts.length equals 4):
                name := parts[0]
                distance = Double.parseDouble(parts[1])
                sciValue = Double.parseDouble(parts[2])
                rate = Double.parseDouble(parts[3]);
                addDestination(name, distance, sciValue, rate)

setNavigationMode(mode):
    navigation.setMode(mode)

incrementProgress(amount):
    progress += amount

getProgress():
    return progress

getNavigationSystem():
    return navigation

getFuelTank():
    return fuelTank

getEngine():
    return engine

launch():
    orbiter = new Orbiter()
    orbiter.startCourse()
```


DOCKED
engine.start()
(if refuel needed, refuel())
CRUISING
destination.isReached()
observationComplete()
ORBITING
fuelTank.isLowFuel()
refuel success
engine.start()
refuel failure
engine.stop()
fuelTank.isLowFuel()
EMERGENCY


User
Simulation
SpaceShuttle
Orbiter
NavigationSystem
Engine
FuelTank
Destination
main(...)
addDestination(...) [×N]
setNavigationMode(...)
launch()
new Orbiter(this)
getNextDestination(0.0)
loop
[for each destination]
start()
loop
[until destination.isReached()]
useFuel(consumptionRate)
returns thrustIncrement (5%)
incrementProgress(thrustIncrement)
updateProgress(progress)
stop()
removeReachedDestinations(progress)
getNextDestination(progress)
mission complete
User
Simulation
SpaceShuttle
Orbiter
NavigationSystem
Engine
FuelTank
Destination


User
Simulation
SpaceShuttle
Orbiter
NavigationSystem
Engine
FuelTank
main(...)
addDestination(...)
launch()
startCourse()
getNextDestination(...)
start()
useFuel(consumptionRate)
thrustIncrement
incrementProgress(...)
stop()
emergencyProcedure()
alt
[random <0.5 (success)]
refuel()
start()
returns to CRUISING
break
[(failure)]
stop()
transitions to DOCKED, mission halts
break
User
Simulation
SpaceShuttle
Orbiter
NavigationSystem
Engine
FuelTank