Solar-Powered Drone Test Cases Documentation

Solar-Powered Drone key requirements:

1. Track Distance between stores and customers
2. Implement a simulated clock to update system time
3. Fuel is consumed when delivering orders
4. If Drone does not have enough fuel to deliver an order, it must stay in the same location and wait for certain amount of time until it is fully recharged with solar power (wait time is calculated by the drone max fuel capacity (C) divided by refuel rate (C/min))
5. Implement day/night like clock cycle such that refuel rate is faster during the day
    - The refuel rate will be 2 times faster during the day compared to at night time
6. System user must be able to adjust the following parameters:
    - Refuel rate for drones (C/min)
    - Fuel rate needed to travel a certain distance (C/m)
    - Drone maximum fuel capacity (C)
    - Speed of drone (m/min)
7. Track the order delivery time and implement penalties for order that was not delivered in a timely manner

Test Cases / Scenarios

[Test Scenario 66] Track Distance between stores and customers

Location parameters (i.e. coordinate x and y) are added to **make_store** and **make_customer** commands.

The **display_distance** command will display the distance between the store and customer

```
> // create multiple stores
> make_store,kroger,33000,100,100
OK:change_completed
> make_store,publix,33000,500,500
OK:change_completed

> // create multiple customers
> make_customer,aapple2,Alana,Apple,222-222-2222,4,100,200,200
OK:change_completed
> make_customer,ccherry4,Carlos,Cherry,444-444-
4444,5,300,400,400
OK:change_completed

> // display distance between store and customers
> display_distance,kroger,aapple2
Distance:142
OK:display_completed
> display_distance,kroger,ccherry4
Distance:425
OK:display_completed
> display_distance,publix,ccherry4
Distance:142
OK:display_completed
```

[Test Scenario 67] Implement a simulated clock to track system time

For the simulated clock, we will assume the following commands will consume 1 minute of time.

Commands	Time Required
make_store	1
sell_item	1
make_pilot	1
make_drone	1
fly_drone	1
make_customer	1
start_order	1
request_item	1
cancel_order	1
transfer_order	1

For the **purchase_order** command, the time consumed will be determined based on the distance between store and customer (and the speed of drone).

For all the **display** commands, we will assume they will not affect the system time.

The **display_time** command will output the current system time.

```
> // display time before running any commands
> display_time
Time: 0 min (Day 0, 00:00AM)
OK:display_completed
> // create multiple stores
> make_store,kroger,33000,100,100
OK:change_completed
> // display time after running make_store command
> display_time
Time: 1 min (Day 0, 00:01AM)
OK:display_completed
> make_store,publix,33000,500,500
OK:change_completed
> display_time
Time: 2 min (Day 0, 00:02AM)
OK:display_completed
> // create multiple items to be sold by stores
> sell_item,kroger,pot_roast,5
OK:change_completed
> sell_item,kroger,cheesecake,4
OK:change_completed
> sell_item,publix,cheesecake,8
OK:change_completed
> // display time
> display_time
Time: 5 min (Day 0, 00:05AM)
OK:display_completed
```

```
> // create multiple pilots to control the drones
> make_pilot,ffig8,Finneas,Fig,888-888-8888,890-12-
3456,panam_10,33
OK:change_completed
> display_time
Time: 6 min (Day 0, 00:06AM)
OK:display_completed
> make_pilot,ggrape17,Gillian,Grape,999-999-9999,234-56
7890,twa_21,31
OK:change_completed
> display_time
Time: 7 min (Day 0, 00:07AM)
OK:display_completed
> // create multiple drones to deliver the orders
> make_drone,kroger,1,40,1000,10,10,1
OK:change_completed
> display_time
Time: 8 min (Day 0, 00:08AM)
OK:display_completed
> make_drone,publix,1,40,5000,10,10,1
OK:change_completed
> display_time
Time: 9 min (Day 0, 00:09AM)
OK:display_completed
> make_drone,kroger,2,20,5000,10,10,1
OK:change_completed
> display_time
Time: 10 min (Day 0, 00:10AM)
OK:display_completed
> fly_drone,kroger,1,ffig8
OK:change_completed
> display_time
Time: 11 min (Day 0, 00:11AM)
OK:display_completed
> fly_drone,publix,1,ggrape17
OK:change_completed
> display_time
Time: 12 min (Day 0, 00:12AM)
OK:display_completed
> // create multiple customers
> make_customer,aapple2,Alana,Apple,222-222-2222,4,100,200,200
OK:change_completed
> make_customer,ccherry4,Carlos,Cherry,444-444-
4444,5,300,400,400
OK:change_completed
> display_time
Time: 14 min (Day 0, 00:14AM)
OK:display_completed
> // create multiple orders as requested by customers
> start_order,kroger,purchaseA,1,aapple2,1000
OK:change_completed
```

```
> start_order,kroger,purchaseB,1,aapple2,1000
OK:change_completed
> start_order,kroger,purchaseD,2,ccherry4,1000
OK:change_completed
> start_order,publix,purchaseA,1,ccherry4,1000
OK:change_completed
> display_time
Time: 18 min (Day 0, 00:18AM)
OK:display_completed
> // add multiple items to the orders
> request_item,kroger,purchaseA,pot_roast,3,10
OK:change_completed
> request_item,kroger,purchaseB,pot_roast,4,5
OK:change_completed
> request_item,publix,purchaseA,cheesecake,3,10
OK:change_completed
> request_item,kroger,purchaseD,cheesecake,1,10
OK:change_completed
> display_time
Time: 22 min (Day 0, 00:22AM)
OK:display_completed
```

For testing purposes, we created a **set_time** command to allow system user to adjust the system time according to their need:

```
> // set system time to 0
> set_time,0
OK:change_completed
> display_time
Time: 0 min (Day 0, 00:00AM)
OK:display_completed
> stop
stop acknowledged
simulation terminated
```

[Test Scenario 68] Fuel is consumed when delivering orders

All of the following commands have already been implemented before. They are repeated here to help setup the system states such that we can demonstrate the changes in system behavior when a drone has delivered an order.

```
> // create multiple stores
> make_store,kroger,33000,100,100
OK:change_completed
> make_store,publix,33000,500,500
OK:change_completed
> // create multiple items to be sold by stores
> sell_item,kroger,pot_roast,5
OK:change_completed
> sell_item,kroger,cheesecake,4
OK:change_completed
> sell_item,publix,cheesecake,8
OK:change_completed
> // create multiple pilots to control the drones
> make_pilot,ffig8,Finneas,Fig,888-888-8888,890-12-
3456,panam_10,33
OK:change_completed
> make_pilot,ggrape17,Gillian,Grape,999-999-9999,234-56
7890,twa_21,31
OK:change_completed
```

Drone's Fuel capacity (C), Refuel rate (C/min), Fuel consumption rate (C/m), Speed (m/min) are added to the **make_drone** command.

```
> // create multiple drones to deliver the orders
> make_drone,kroger,1,40,1000,10,10,1
OK:change_completed
> make_drone,publix,1,40,5000,10,10,1
OK:change_completed
> make_drone,kroger,2,20,5000,10,10,1
OK:change_completed
```

In the following sequence, we will assign the drones to pilots, create new customers, create new orders, and add items to orders:

```
> fly_drone,kroger,1,ffig8
OK:change_completed
> fly_drone,publix,1,ggrape17
OK:change_completed
> // create multiple customers
> make_customer,aapple2,Alana,Apple,222-222-2222,4,100,200,200
OK:change_completed
> make_customer,ccherry4,Carlos,Cherry,444-444-
```

```
4444,5,300,400,400
OK:change_completed
```

When a user create an order and assign to a drone, the system will throw an error if the drone maximum flight range does not cover the required distance. The maximum flight range is determined by the drone fuel capacity and fuel consumption rate. (i.e flight range = fuel cap [C] / fuel consumption rate [C/min])

```
> // create multiple orders as requested by customers
> start_order,kroger,purchaseA,1,aapple2,1000
ERROR:order_distance_exceeded_drone_max_flight_range
> start_order,kroger,purchaseB,1,aapple2,1000
ERROR:order_distance_exceeded_drone_max_flight_range
> start_order,kroger,purchaseD,2,ccherry4,1000
OK:change_completed
> start_order,publix,purchaseA,1,ccherry4,1000
OK:change_completed
> // add multiple items to the orders
> request_item,kroger,purchaseA,pot_roast,3,10
ERROR:order_identifier_does_not_exist
> request_item,kroger,purchaseB,pot_roast,4,5
ERROR:order_identifier_does_not_exist
> request_item,publix,purchaseA,cheesecake,3,10
OK:change_completed
> request_item,kroger,purchaseD,cheesecake,1,10
OK:change_completed
```

For simplicity, in this example all drones were set at a speed of 1 m/min. Hence, the time consumed (in minutes) to purchase an order will be equal to the distance between store and customer (time consumed = distance travelled [m] / drone speed [1m/min]).

The drone's remaining fuel will be consumed after an order has been delivered. The fuel consumed will be calculated by Fuel rate (C/m) * Distance (m),

```
> // display drones for in store publix
> display_drones,publix
droneID:1,total_cap:40,num_orders:1,remaining_cap:16,fuel_cap:
5000,remaining_fuel:5000,refuel_rate:10,fuel_consumption_rate:
10,speed:1,flown_by:Gillian_Grape
OK:display_completed
```

In the following sequence, we will demonstrate the changes in system states when an order is being delivered to a customers.

```
> // display time before purchasing an order
> display_time
Time: 22 min (Day 0, 00:22AM)
```

```
OK:display_completed
> // display distance between store and customers
> display_distance,publix,ccherry4
Distance: 142
OK:display_completed
> // deliver an order and display the updated state
> purchase_order,publix,purchaseA
Delivery Time: 142 minutes
OK:change_completed
> // display time after purchasing an order
> display_time
Time: 164 min (Day 0, 02:44AM)
OK:display_completed
> display_drones,publix
droneID:1,total_cap:40,num_orders:0,remaining_cap:40,fuel_cap:
5000,remaining_fuel:3580,refuel_rate:10,fuel_consumption_rate:
10,speed:1,flown_by:Gillian_Grape
OK:display_completed
> stop
stop acknowledged
simulation terminated
```

[Test Scenario 69] If Drone does not have enough fuel, it must stay in the same location and wait for certain amount of time until it is fully recharged with solar power (recharge time [min] = (drone max fuel capacity - remaining fuel) [C] / refuel rate [C/min]).

In the following case, the fuel required to deliver the order is 4250 (C) but the drone remaining fuel only has 3580 (C). The drone has to wait for additional 142 min (i.e. (max fuel cap [5000 C] - remaining fuel [3580 C]) / refuel rate [10 C/min]) to recharge by solar power. Hence, the delivery time is 567 minutes (recharge time [142 min] + time to travel the distance required [425 min])

```
> // create a new order
> start_order,publix,purchaseA,1,aapple2,1000
OK:change_completed
> display_time
Time: 165 min (Day 0, 02:45AM)
OK:display_completed
> display_distance,publix,aapple2
Distance: 425
OK:display_completed
> display_drones,publix
droneID:1,total_cap:40,num_orders:1,remaining_cap:40,fuel_cap:
5000,remaining_fuel:3580,refuel_rate:10,fuel_consumption_rate:
10,speed:1,flown_by:Gillian_Grape
OK:display_completed
> purchase_order,publix,purchaseA
Delivery Time: 567 minutes
OK:change_completed
> // display time after purchasing an order
> display_time
Time: 732 min (Day 0, 12:12PM)
OK:display_completed
```

After the drone has delivered the order, the drone remaining fuel will be 750 C (max_fuel_cap [5000 C] – fuel required for delivery [4250 C])

```
> display_drones,publix
droneID:1,total_cap:40,num_orders:0,remaining_cap:40,fuel_cap:
5000,remaining_fuel:750,refuel_rate:10,fuel_consumption_rate:1
0,speed:1,flown_by:Gillian_Grape
OK:display_completed
```

[Test Scenario 70] Implement day/night like clock cycle such that refuel rate is faster during the day

We will assume drone refuel rate during day time to be 2 times faster than at night.
Using the above example, the fuel required to deliver the order is 4250 (C) but the drone remaining fuel only has 3580 (C). The drone has to wait for additional 71 min ONLY (i.e. (max fuel cap [5000 C] - remaining fuel [3580 C]) / refuel rate [10 C/min] / 2) to recharge by solar power. Hence, the delivery time is 496 minutes (recharge time [71 min] + time to travel the distance required [425 min])
For ease of testing, we can set the system time to noon using the **set_time** command as demonstrated below.

```
> // create a new order
> start_order,publix,purchaseA,1,aapple2,1000
OK:change_completed
> // Set system time to daytime
> set_time,720
OK:change_completed
> display_time
Time: 720 min (Day 0, 12:00PM)
OK:display_completed
> display_distance,publix,aapple2
Distance: 425
OK:display_completed
> display_drones,publix
droneID:1,total_cap:40,num_orders:1,remaining_cap:40,fuel_cap:
5000,remaining_fuel:3580,refuel_rate:10,fuel_consumption_rate:
10,speed:1,flown_by:Gillian_Grape
OK:display_completed
> purchase_order,publix,purchaseA
Delivery Time: 496 minutes
OK:change_completed
> // display time after purchasing an order
> display_time
Time: 1216 min (Day 0, 08:16PM)
OK:display_completed
```

After the drone has delivered the order, the drone remaining fuel will be 750 C
(max_fuel_cap [5000 C] – fuel required for delivery [4250 C])

```
> display_drones,publix
droneID:1,total_cap:40,num_orders:0,remaining_cap:40,fuel_cap:
5000,remaining_fuel:750,refuel_rate:10,fuel_consumption_rate:1
0,speed:1,flown_by:Gillian_Grape
OK:display_completed
```

[Test Scenario 71] Track the order delivery time and implement penalties for order that was not delivered in a timely manner.

An expected delivery time parameter is added to the **start_order** command to allow user to define the expected delivery time. If the actual delivery time is greater than the expected delivery time, a late penalty will be applied to the store.

In the following sequence, we have created an order where the expected delivery time is set to 300 minutes but the actual delivery time took 496 minutes. Hence, a late penalty will be applied to the store.

```
> // display efficiency before applying late penalty
> display_efficiency
name:kroger,purchases:0,overloads:0,transfers:0,penalties:0
name:publix,purchases:1,overloads:0,transfers:0,penalties:0
OK:display_completed
> // create a new order with expected delivery time of 300
minutes
> start_order,publix,purchaseB,1,aapple2,300
OK:change_completed
> purchase_order,publix,purchaseB
Expected_Delivery Time: 300 minutes
Actual_Delivery Time: 496 minutes
OK:late_delivery_penalty_applied
> // display efficiency after applying late penalty
> display_efficiency
name:kroger,purchases:0,overloads:0,transfers:0,penalties:0
name:publix,purchases:2,overloads:0,transfers:0,penalties:1
OK:display_completed
```