Maersk case Study

Simulating a container terminal using SimPy

Overview

Create a working demo of a basic container terminal simulation using SimPy.

- You only need to use the standard python library + SimPy.
- The goal of the exercise is to see that the applicant can translate described logic into code and to asses code quality. So make sure that the code is written using best practices as you know it. An object oriented approach is suggested.
- The SimPy doc page provides information about the API and examples that you can use to get started:
    - https://simpy.readthedocs.io/en/latest
- You will only need to use the components of the SimPy API that is used in this example:
    - https://simpy.readthedocs.io/en/latest/examples/bank_renege.html
    - Note that the Resource request can be made without using the with statement in order to give more control over when the resource should be released.

Task

1. Simulate vessels arriving to the container terminal. The time between vessel arrivals follows an exponential distribution with an average of 5 hours (https://docs.python.org/3/library/random.html#random.expovariate). This is the input that drives the simulation.
    - Each vessel carries 150 containers that needs to be discharged (unloaded).
2. The vessels will need to berth at the terminal and there are only 2 available slots (berth_1 & berth_2). This means that if a vessel arrives and both berths are already occupied, the new vessel will be blocked and has to wait.
3. Once the vessel berths, a quay crane (https://en.wikipedia.org/wiki/Container_crane) will start lifting the containers from the vessel to terminal trucks.
    - There are 2 quay cranes and any quay crane can operate on any berth
    - The vessel will use 1 crane only
    - It takes the crane 3 minutes to move one container.
    - The crane must put the container on a truck. If no truck is available, the crane will have to wait until a truck is free before it can start its next move.
    - The crane can use any of the free trucks, it does not have to wait for the same truck to come back again. This means that if only one vessel is in berth, it will never be blocked by waiting for

a truck. But if two vessels are in beth, the cranes will sometimes be blocked while waiting for a free truck.

4. The terminal has 3 trucks, transporting containers from the quay cranes to the yard blocks (https://en.wikipedia.org/wiki/Terminal_tractor).

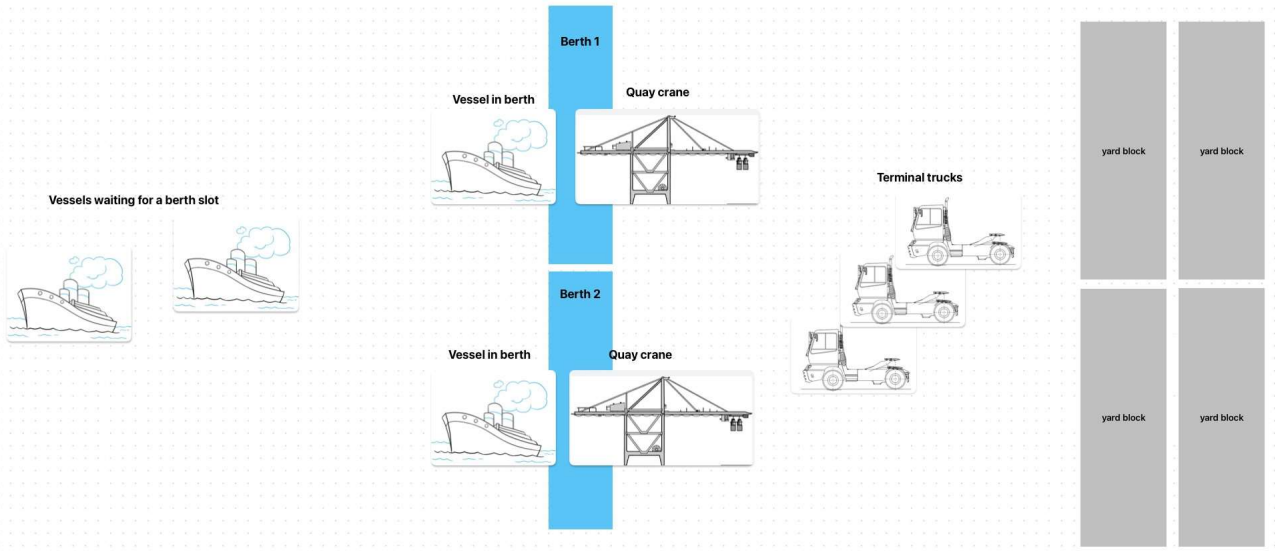
    - It takes the truck 6 minutes to drop off the container at the yard block and come back again

5. Create a simple log (print statements does the job) for each event, e.g. vessel arriving, vessel berthing, quay crane moves a container, etc. The log should include the current time using SimPy Environment.now (https://simpy.readthedocs.io/en/latest/api_reference/simpy.core.html#simpy.core.Environment.now).

6. When all containers have been lifted, the vessel leaves and the berth slot is free to be used by the next vessel.

7. Run the simulation for some time using env.run(until=SIMULATION_TIME) where SIMULATION_TIME is a parameter the user can set.

8. The simulation time in SimPy is unitless but in this exercise we can assume 1 tick = 1 minute



Overview

1. **Imports and Constants**

```
import simpy
import random

# Constants
AVG_ARRIVAL_TIME = 5 * 60  # 5 hours in minutes
NUM_CONTAINERS = 150
CRANE_TIME = 3  # minutes per container
TRUCK_TIME = 6  # minutes round trip
SIMULATION_TIME = 24 * 60  # 24 hours
```

- simpy: Used for discrete-event simulation.
- random: Generates random numbers for vessel arrival times.
- Constants: Define the average arrival time, number of containers per vessel, time taken by cranes and trucks, and total simulation time.

2. **Vessel Process**

```
def vessel(env, name):
    print(f'{env.now}: Vessel {name} arriving')
    with berths.request() as berth_request:
        yield berth_request
        print(f'{env.now}: Vessel {name} berthing')
        yield env.process(unload_vessel(env, name))
```

- vessel: Represents a vessel arriving at the terminal.
- berths.request(): Requests a berth for the vessel. If both are occupied, it waits.
- env.now: Current simulation time.

3. **Unloading Process**

```
def unload_vessel(env, name):
    with cranes.request() as crane_request:
        yield crane_request
        for i in range(NUM_CONTAINERS):
            yield env.timeout(CRANE_TIME)
            print(f'{env.now}: Crane unloading container {i+1} from Vessel {name}')
            with trucks.request() as truck_request:
                yield truck_request
                yield env.timeout(TRUCK_TIME)

    print(f'{env.now}: Vessel {name} finished unloading')
```

- cranes.request(): Requests a crane to unload containers.
- env.timeout(CRANE_TIME): Simulates the time taken to unload each container.
- trucks.request(): Requests a truck to transport the container.

4. **Vessel Generator**

```
def vessel_generator(env):
    vessel_number = 0
    while True:
        yield env.timeout(random.expovariate(1/AVG_ARRIVAL_TIME))
        vessel_number += 1
        env.process(vessel(env, f'Vessel {vessel_number}'))
```

- vessel_generator: Continuously generates vessels arriving at the terminal.
- random.expovariate: Models vessel inter-arrival times using an exponential distribution.

5. **Simulation Setup and Execution**

```
# Environment setup
env = simpy.Environment()
berths = simpy.Resource(env, capacity=2)
cranes = simpy.Resource(env, capacity=2)
trucks = simpy.Resource(env, capacity=3)
```

```
# Run the simulation
env.process(vessel_generator(env))
env.run(until=SIMULATION_TIME)
```

- simpy.Environment(): Sets up the simulation environment.
- Resources: Define available berths, cranes, and trucks.
- env.run: Executes the simulation for the specified duration.

Documentation

Overview: This simulation models a container terminal, focusing on vessel arrivals, berthing, and container handling using SimPy.

Assumptions:

- Vessels arrive with an exponential distribution averaging 5 hours.
- Each vessel carries 150 containers.
- Two berths, two cranes, and three trucks are available.

Results:

- Logs capture vessel activities, showcasing efficient operations.

Analysis

Efficiency:

- Resources are effectively utilized to minimize wait times.
- The simulation demonstrates optimal operation under the given constraints.

Potential Bottlenecks:

- Occur when both berths are occupied.
- Could be alleviated by adding more trucks for higher traffic.