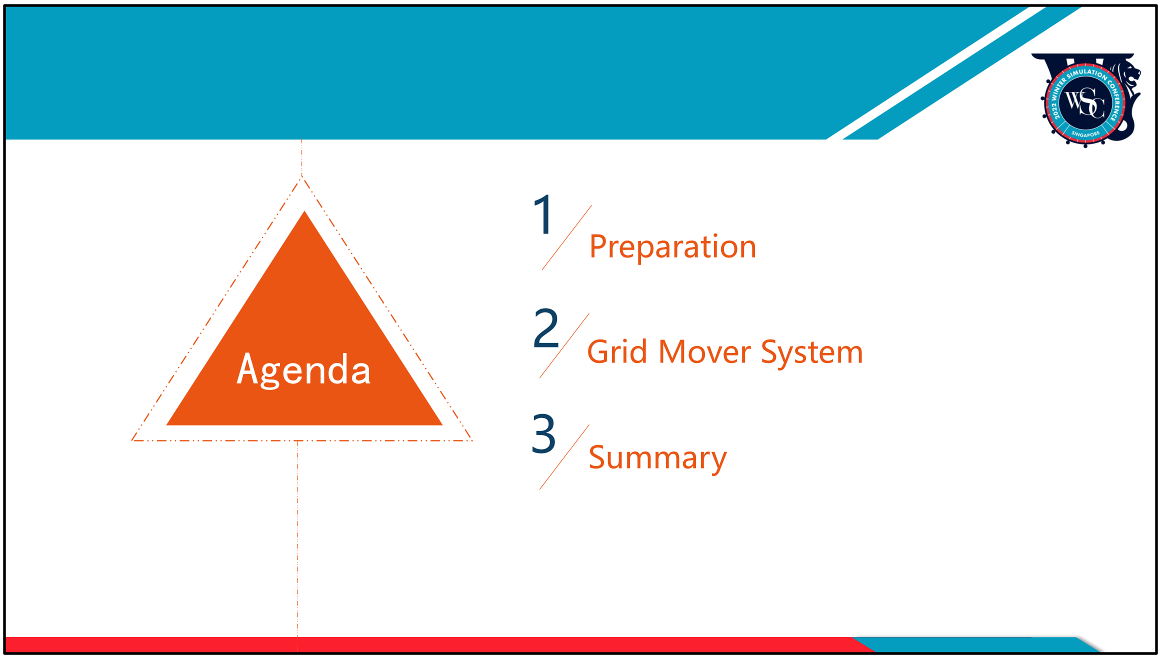
2022 WINTER SIMULATION CONFERENCE
WSC
SINGAPORE

CASE STUDY COMPETITION

Automated Grid Mover System

Welcome to the WSC case study competition. The case study for this competition is based on the automated grid mover system and this set of materials aims to help participants get a better understanding of the system as well as what they are required to do.



Agenda

1 Preparation

2 Grid Mover System

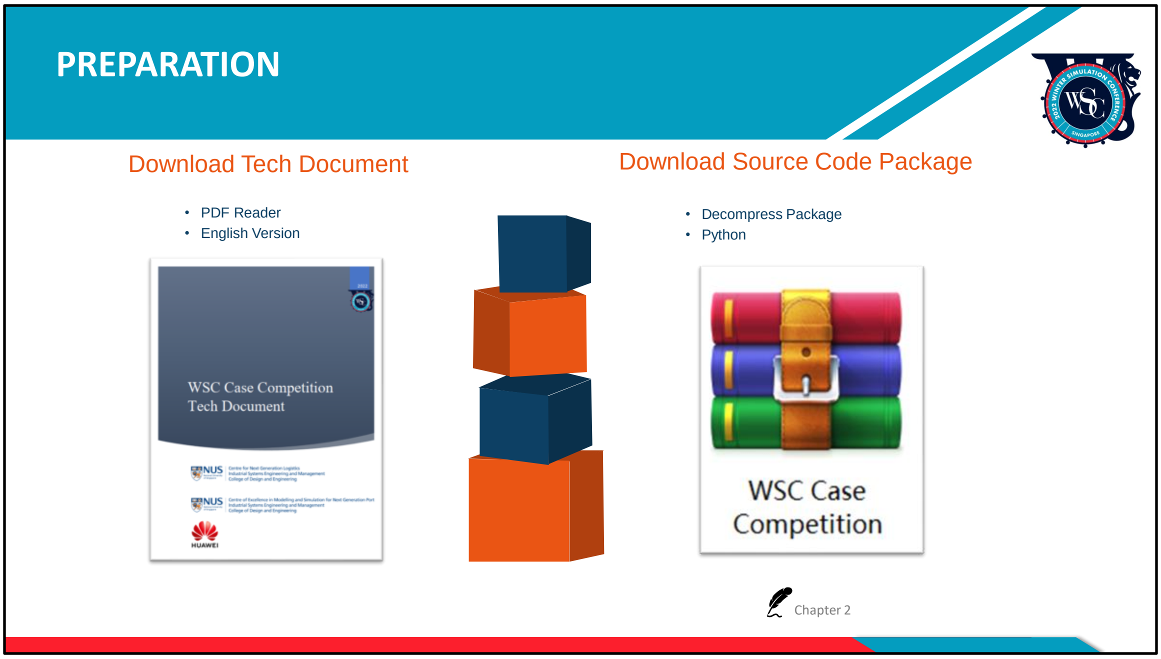
3 Summary

First, we will cover the preparation material offered to participants and how they can make full use of the materials; Second, we will provide a detailed illustration of the grid mover system and what is expected from participants. This will then be followed with a summary.

01 Preparation

- Download Tech Document
- Download Source Code Package

Let’s start with the first part.



PREPARATION

Download Tech Document

- PDF Reader
- English Version

Download Source Code Package

- Decompress Package
- Python

WSC Case Competition
Tech Document
HUAWEI
WSC Case
Competition
Chapter 2

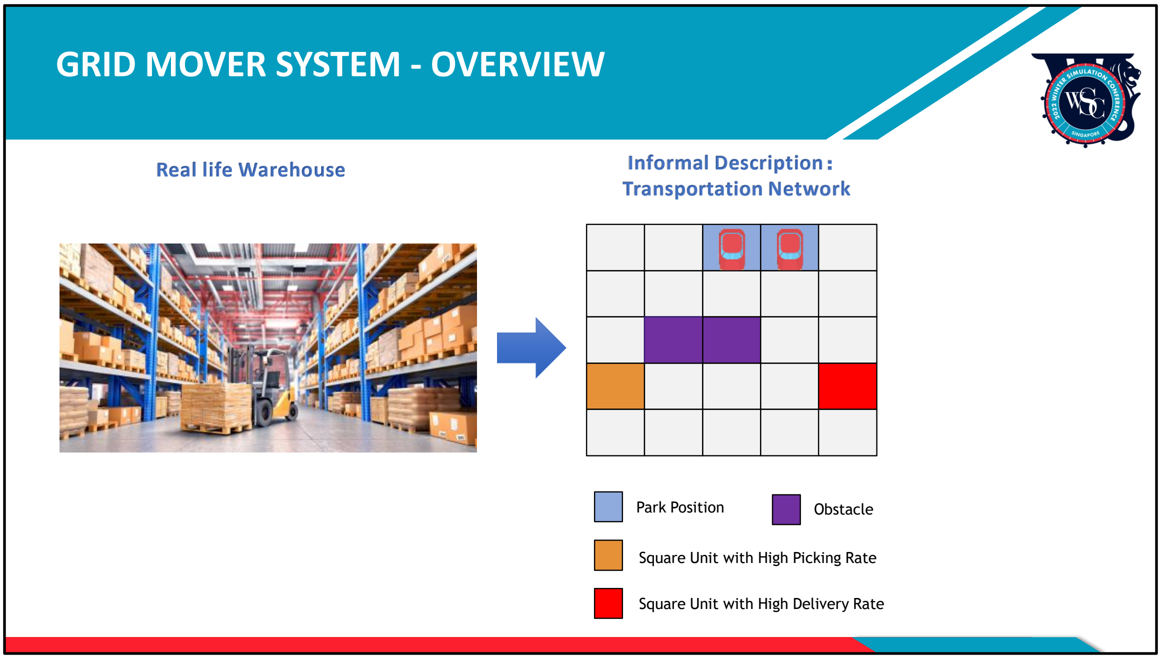
Upon registration and sign-in, a tech document and a source code zip package will be made available to participants. Please refer to the document for further details regarding the descriptions on the case description, model structure, and instructions on file downloading and submission procedures.

After downloading and unzipping source code, you can view and run the whole system through Pycharm.

02 Grid Mover System

- Overview
- Gridmover System Handler
- XML Input
- Element Creator & Job Generator
- Run
- Interfaces

Next, we have a detailed illustration of the grid mover structure.



GRID MOVER SYSTEM - OVERVIEW

Real life Warehouse

Informal Description:
Transportation Network

		🟦	🟦	
	🟪	🟪		
🟧				🟥

🟦 Park Position

🟪 Obstacle

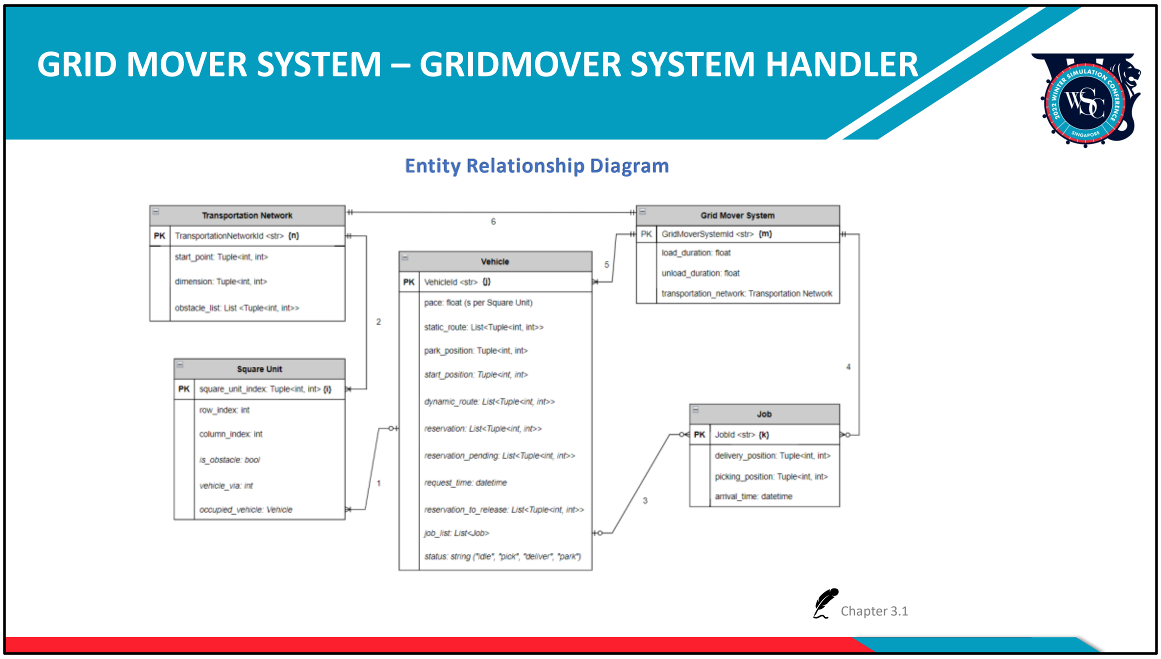
🟧 Square Unit with High Picking Rate

🟥 Square Unit with High Delivery Rate

First of all, this is an overview of the automated grid mover system case study.

The picture on the left is a real-life warehouse. On the right, there is an informal description of this warehouse, which is called the transportation network. As you can see, the warehouse is divided into many square units, with different colours representing different functions. Blue represents vehicle parking positions, Purple represents obstacles, while square units with High Picking Rates are Orange and square units with High Delivery Rates are Red.

Next, let's explore entities in this grid mover system.



GRID MOVER SYSTEM – GRIDMOVER SYSTEM HANDLER

Entity Relationship Diagram

	Transportation Network
PK	TransportationNetworkId <str> {n}
	start_point: Tuple<int, int>
	dimension: Tuple<int, int>
	obstacle_list: List <Tuple<int, int>>

	Square Unit
PK	square_unit_index: Tuple<int, int> {i}
	row_index: int
	column_index: int
	is_obstacle: bool
	vehicle_via: int
	occupied_vehicle: Vehicle

	Vehicle
PK	VehicleId <str> {j}
	pace: float (s per Square Unit)
	static_route: List<Tuple<int, int>>
	park_position: Tuple<int, int>
	start_position: Tuple<int, int>
	dynamic_route: List<Tuple<int, int>>
	reservation: List<Tuple<int, int>>
	reservation_pending: List<Tuple<int, int>>
	request_time: datetime
	reservation_to_release: List<Tuple<int, int>>
	job_list: List<Job>
	status: string ("idle", "pick", "deliver", "park")

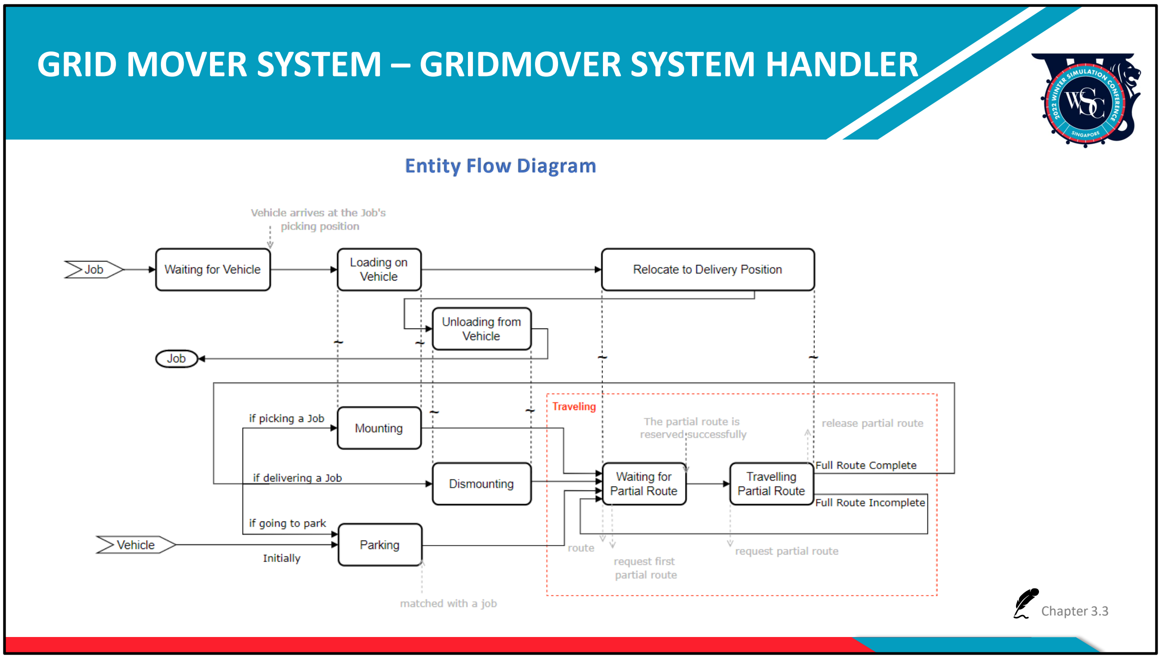
	Grid Mover System
PK	GridMoverSystemId <str> {m}
	load_duration: float
	unload_duration: float
	transportation_network: Transportation Network

	Job
PK	JobId <str> {k}
	delivery_position: Tuple<int, int>
	picking_position: Tuple<int, int>
	arrival_time: datetime

Chapter 3.1

This is a standard entity relationship diagram. We can see the transportation network which holds a one-to-one relationship with the grid mover system. This is because 1 grid mover system contains at most 1 transportation network and one transportation network can only appear in 1 grid mover system. While 1 transportation network is composed of at least 1 square unit and 1 square unit can only appear at 1 transportation network, the relationship between them is one-to-many.

As for vehicle and square unit, their relationship is one-to-many, because 1 vehicle can reserve more than 1 square units, whereas 1 square unit can only be occupied by 1 vehicle at one time. 1 vehicle can handle a lot of jobs and 1 job can only appear in 1 vehicle, so the relationship is one-to-many. Of course, in 1 system there can be many jobs and vehicles, so system to job and to vehicle are all one-to-many. For more details, please refer to chapter 3.1 “Entities" of the tech document.



GRID MOVER SYSTEM – GRIDMOVER SYSTEM HANDLER

Entity Flow Diagram

Vehicle arrives at the Job's picking position

Job

Waiting for Vehicle

Loading on Vehicle

Relocate to Delivery Position

Unloading from Vehicle

Job

Traveling

if picking a Job

Mounting

if delivering a Job

Dismounting

if going to park

Parking

Vehicle

Initially

The partial route is reserved successfully

release partial route

Waiting for Partial Route

Travelling Partial Route

Full Route Complete

Full Route Incomplete

route

request first partial route

request partial route

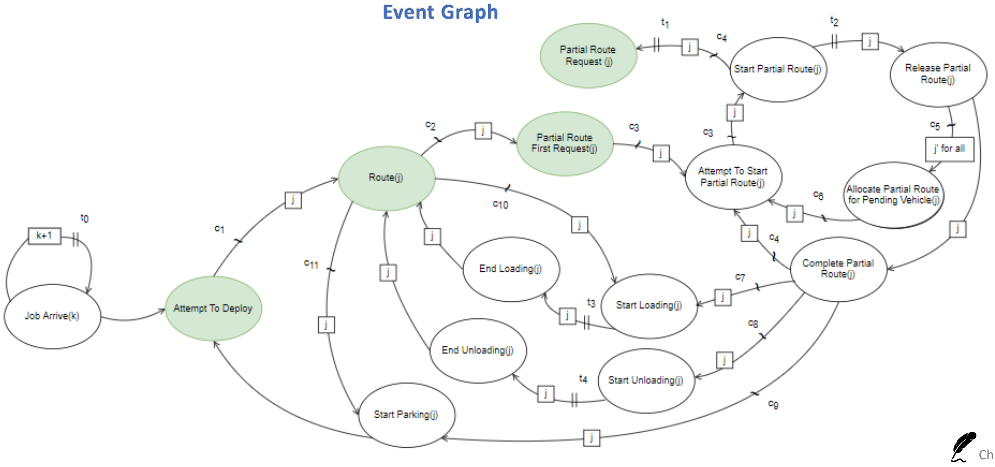
matched with a job

Chapter 3.3

The entity flow chat here describes the main activities of job and vehicle, as well as the relationship between these activities. For more details, please refer to chapter 3.3 "Activity-Based Description" of the tech document.

GRID MOVER SYSTEM – GRIDMOVER SYSTEM HANDLER

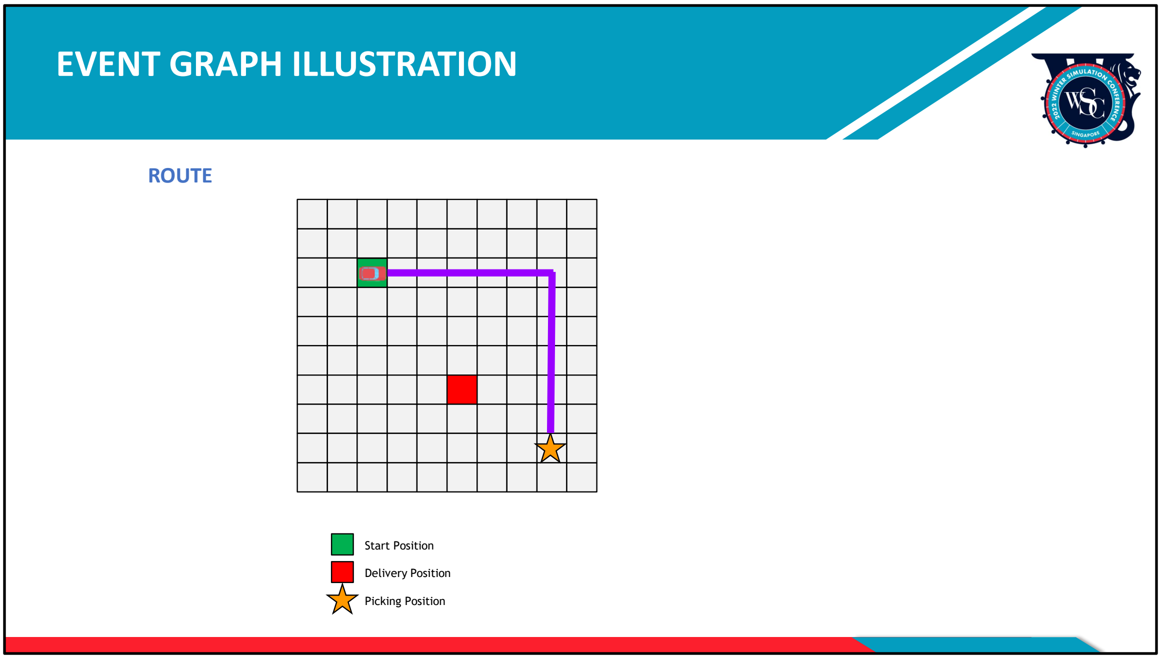
Event Graph



Chapter 3.4

Our grid mover system is a discrete event-based system, and this is a standard event graph. It is drawn based on the events in our grid mover system. We can understand the operation logic of the entire system through the event graph, take note of the events marked in green.

First, when the job enters the system, the job_arrive event is triggered, and then this event triggers the attempt_to_deploy event, which assigns available jobs to suitable vehicles. After the vehicle and job are matched, the route event will be triggered. Next, we will show the process of the vehicle moving on the transportation network in order to provide a better understanding of the events.



EVENT GRAPH ILLUSTRATION

ROUTE

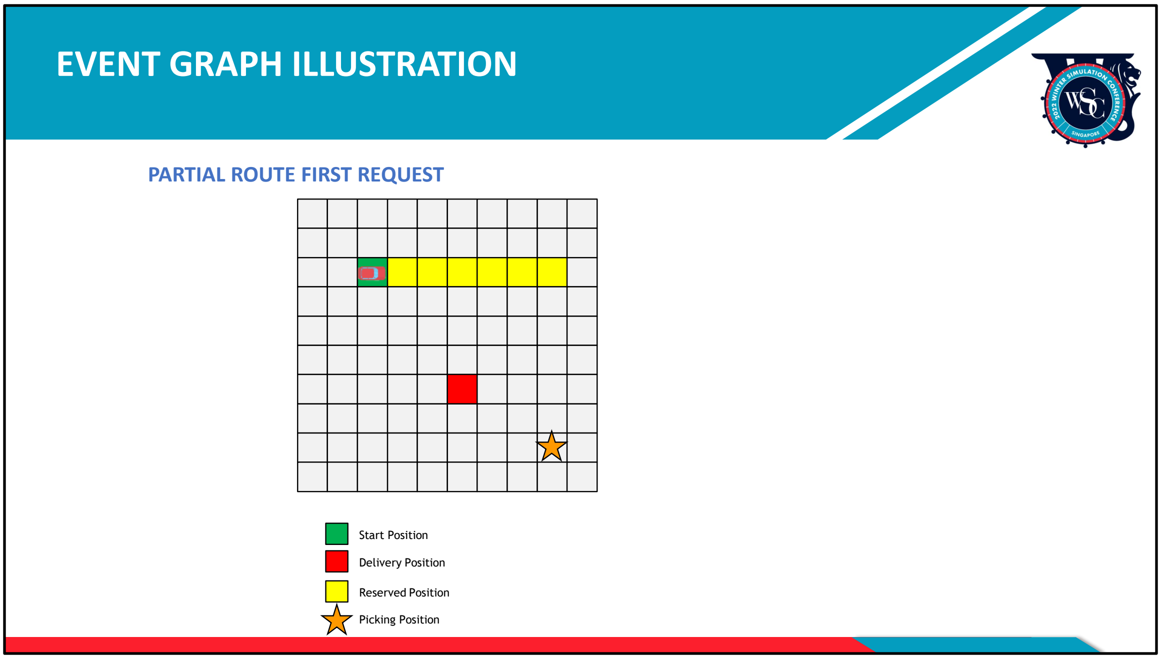
🟩 Start Position

🟥 Delivery Position

★ Picking Position

First, the route event will use an algorithm to calculate the entire path from one place to another. As shown in the picture, if a vehicle needs to go from its parking position to the picking position, it must pass a total of 12 square units, which make up for its travel route.

At this moment the vehicle is parked at the start position.



EVENT GRAPH ILLUSTRATION

PARTIAL ROUTE FIRST REQUEST

🟩 Start Position

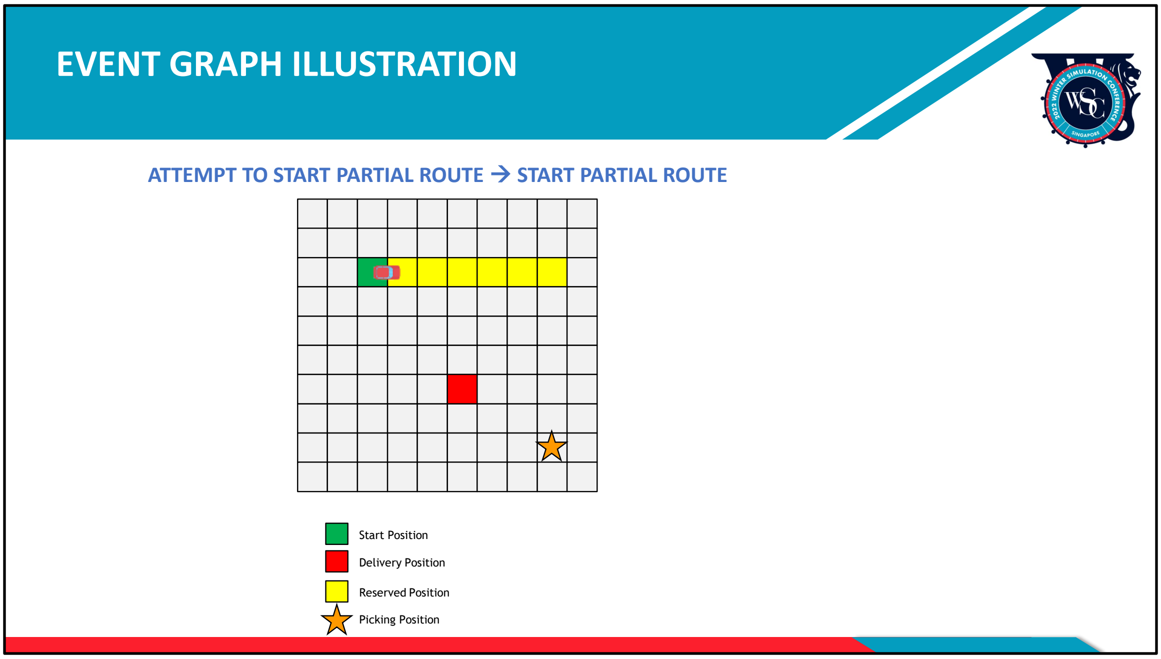
🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

The partial_route_first_request event will be triggered once the vehicle gets its path and it will then try to reserve a partial route.

For example, for a total of 12 square units, the first 6 square units are reserved to avoid collision during travelling. When the vehicle reserves these 6 square units successfully, other vehicles cannot reserve these grids again, and can only wait for the requested square units to be released. The attempt_to_start_partial_route event is only triggered after the vehicle has successfully reserved its partial route.



EVENT GRAPH ILLUSTRATION

ATTEMPT TO START PARTIAL ROUTE → START PARTIAL ROUTE

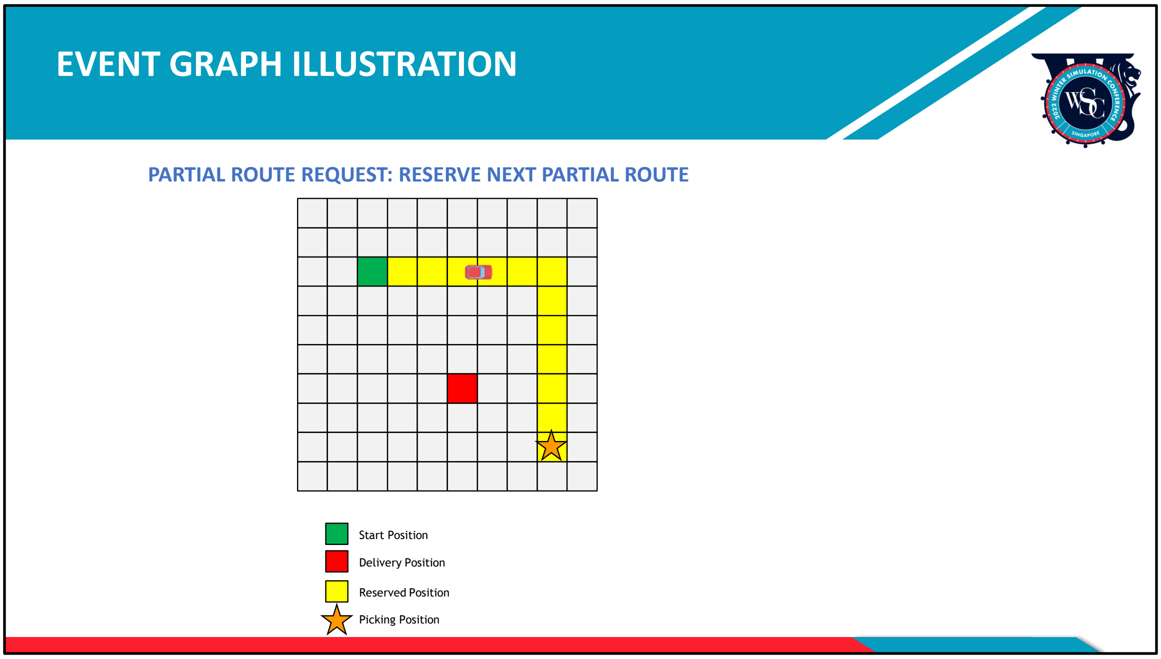
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

The attempt_to_start_partial_route event will verify whether the reservation was successful. If successful, the start_partial_route event will be triggered, which allows the vehicle to start moving.



EVENT GRAPH ILLUSTRATION

PARTIAL ROUTE REQUEST: RESERVE NEXT PARTIAL ROUTE

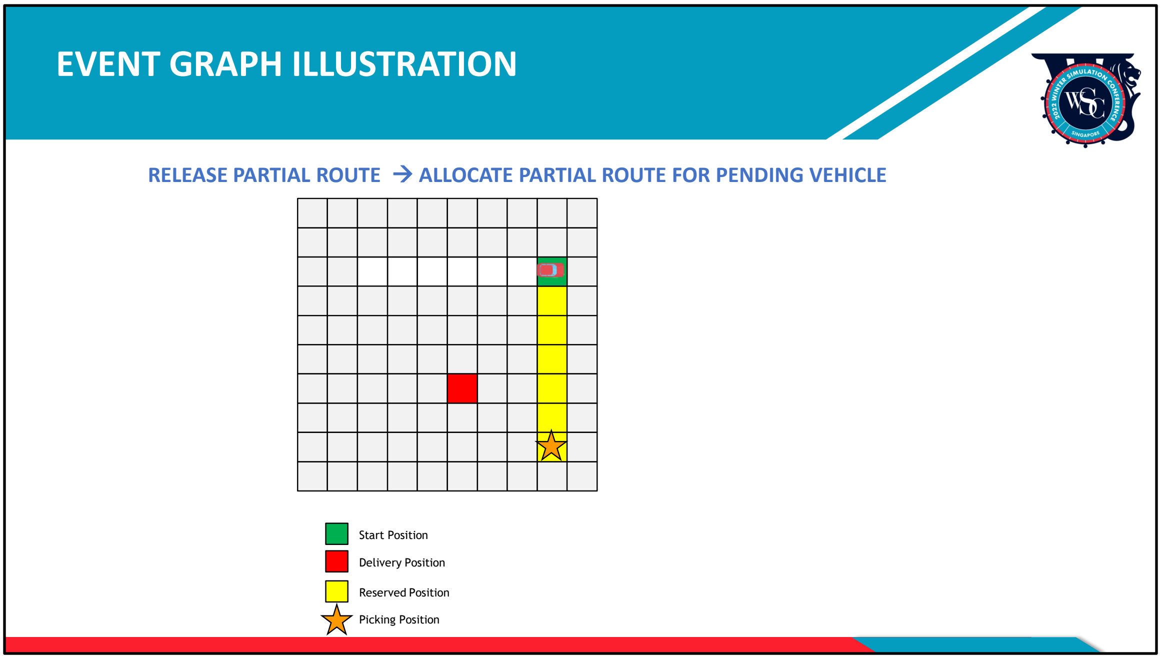
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

During the movement of the vehicle, the partial_route_request event will be triggered to reserve the next 6 square units that the vehicle will need to pass through later.



EVENT GRAPH ILLUSTRATION

RELEASE PARTIAL ROUTE → ALLOCATE PARTIAL ROUTE FOR PENDING VEHICLE

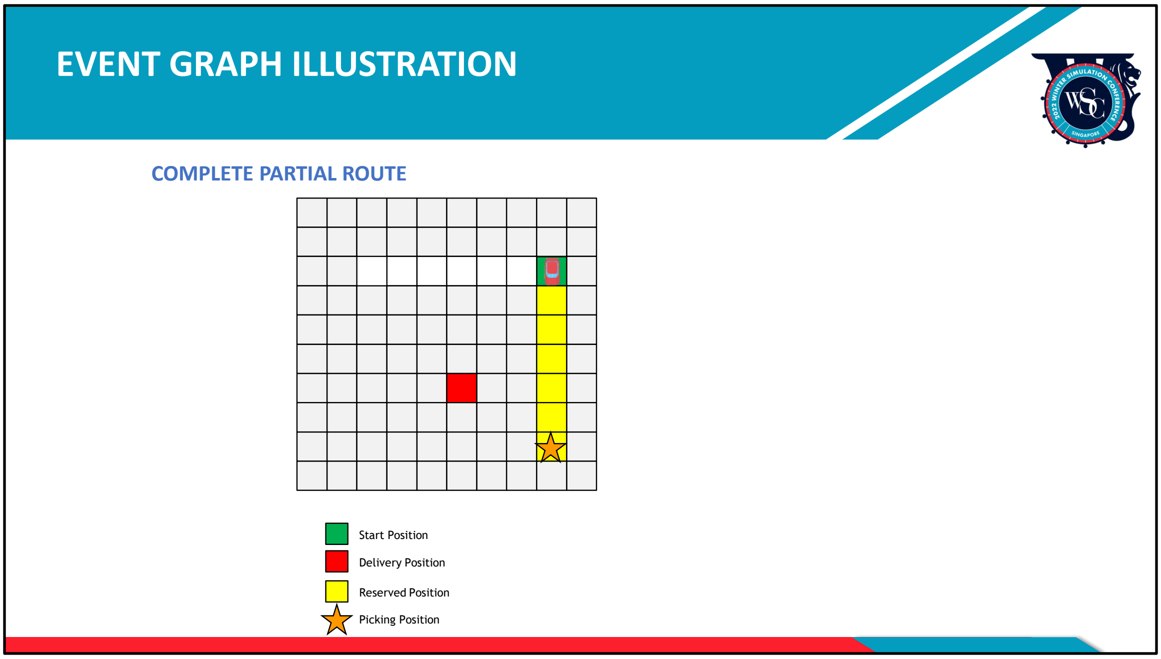
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

When the vehicle reaches the sixth square unit, the release_partial_route event will be triggered and it will release the previous start position as well as the 5 square units that have been passed. The current position of the vehicle at the sixth square unit will be updated as the new start position. After releasing these square units, if there are other vehicles waiting for a square unit, the allocate_partial_route_for_pending_vehicle event will be triggered. If the pending vehicle is able to reserve its partial route from these released square units, then the attempt_to_start_partial_route event will be triggered and the pending vehicle will start moving.

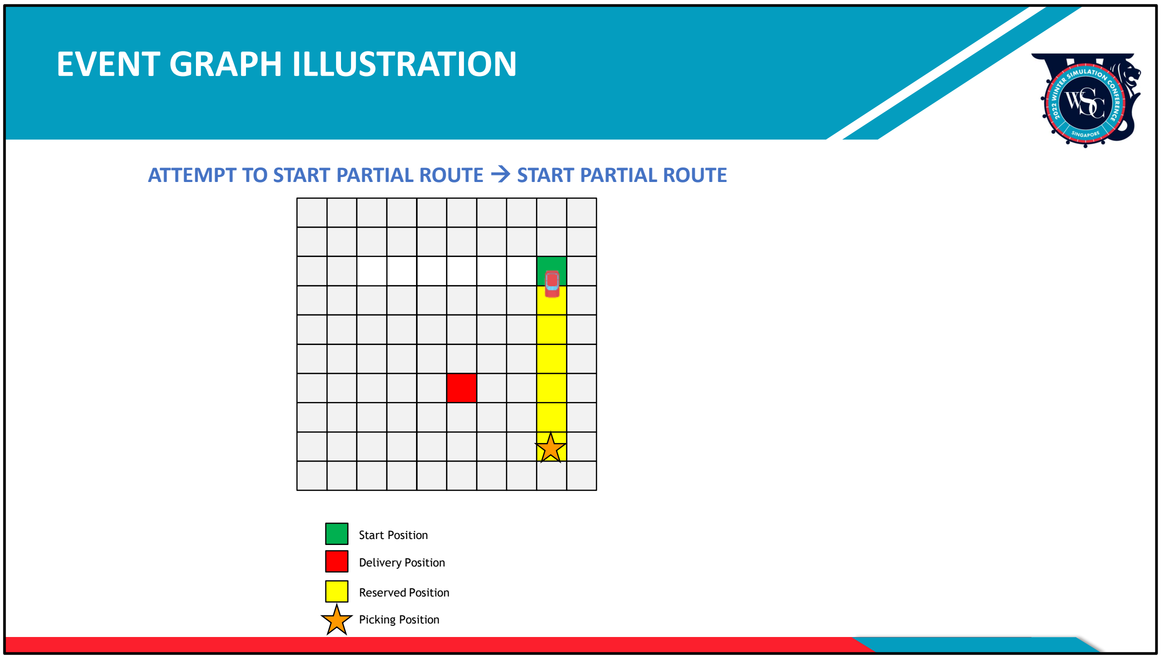


EVENT GRAPH ILLUSTRATION

COMPLETE PARTIAL ROUTE

- 🟩 Start Position
- 🟥 Delivery Position
- 🟨 Reserved Position
- ★ Picking Position

The release_partial_route event will also trigger the complete_partial_route event, which means that the partial route has been successfully completed.



EVENT GRAPH ILLUSTRATION

ATTEMPT TO START PARTIAL ROUTE → START PARTIAL ROUTE

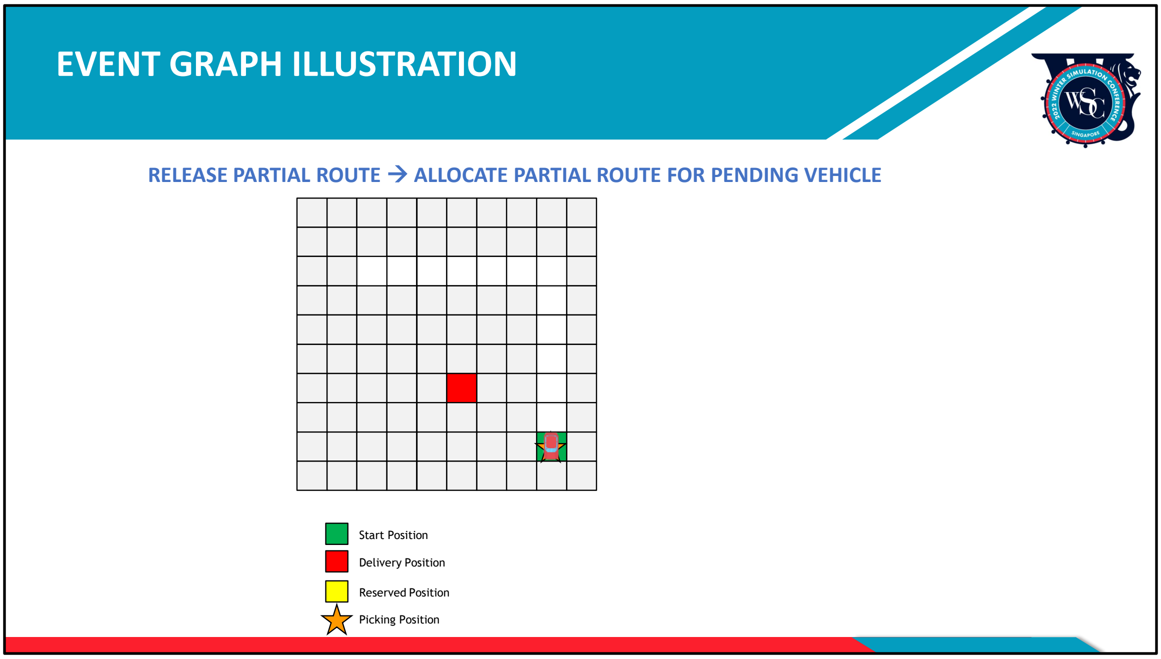
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

After the previous partial route is completed, the attempt_to_start_partial_route event will be triggered, which will further trigger the start_partial_route event, which means the vehicle will start to move again.

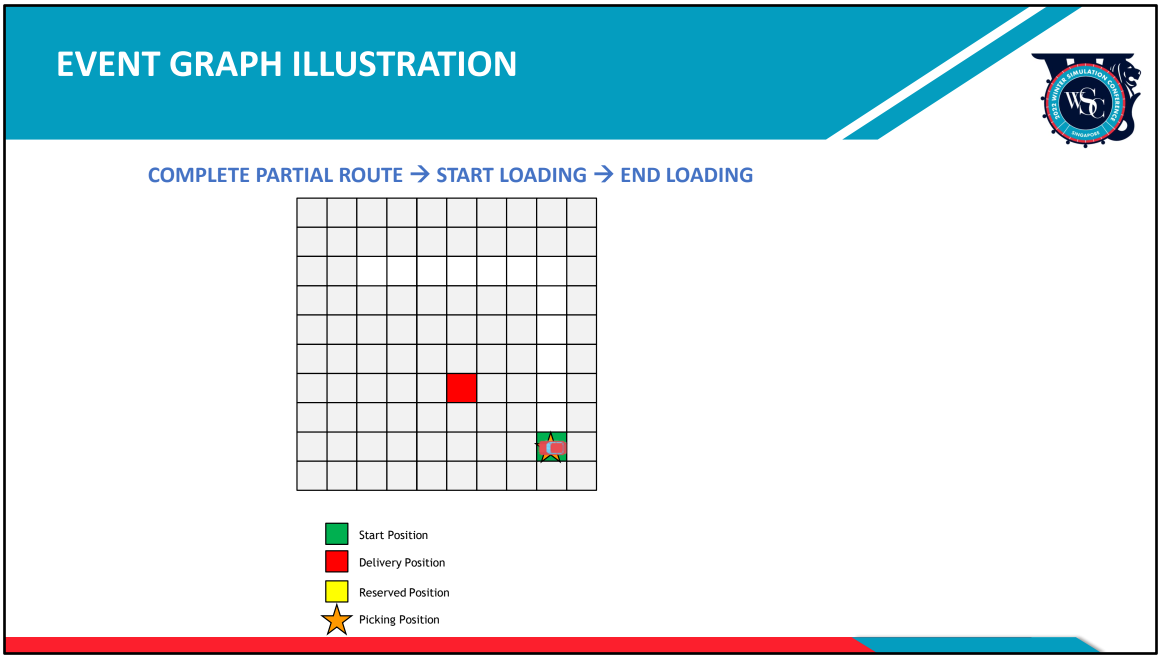


EVENT GRAPH ILLUSTRATION

RELEASE PARTIAL ROUTE → ALLOCATE PARTIAL ROUTE FOR PENDING VEHICLE

- 🟩 Start Position
- 🟥 Delivery Position
- 🟨 Reserved Position
- ★ Picking Position

Once again, when the vehicle moves to the last square unit of the partial route, it will release its partial route and the allocate_partial_route_for_pending_vehicle event will be triggered for any other pending vehicles in the system.



EVENT GRAPH ILLUSTRATION

COMPLETE PARTIAL ROUTE → START LOADING → END LOADING

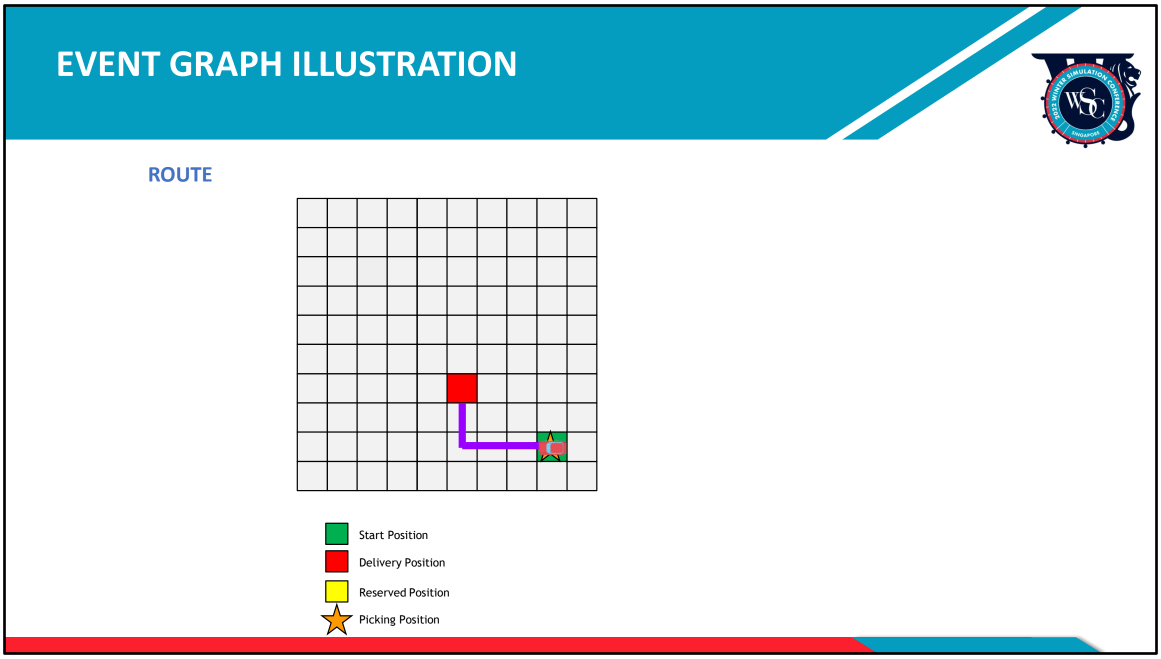
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

After the partial route is completed, if the vehicle's start position is now the picking position, this means the vehicle has completed the full route. The start_loading event will then be triggered. After some time, the end_ loading event will be scheduled, indicating the completion of loading the job item onto the vehicle.



EVENT GRAPH ILLUSTRATION

ROUTE

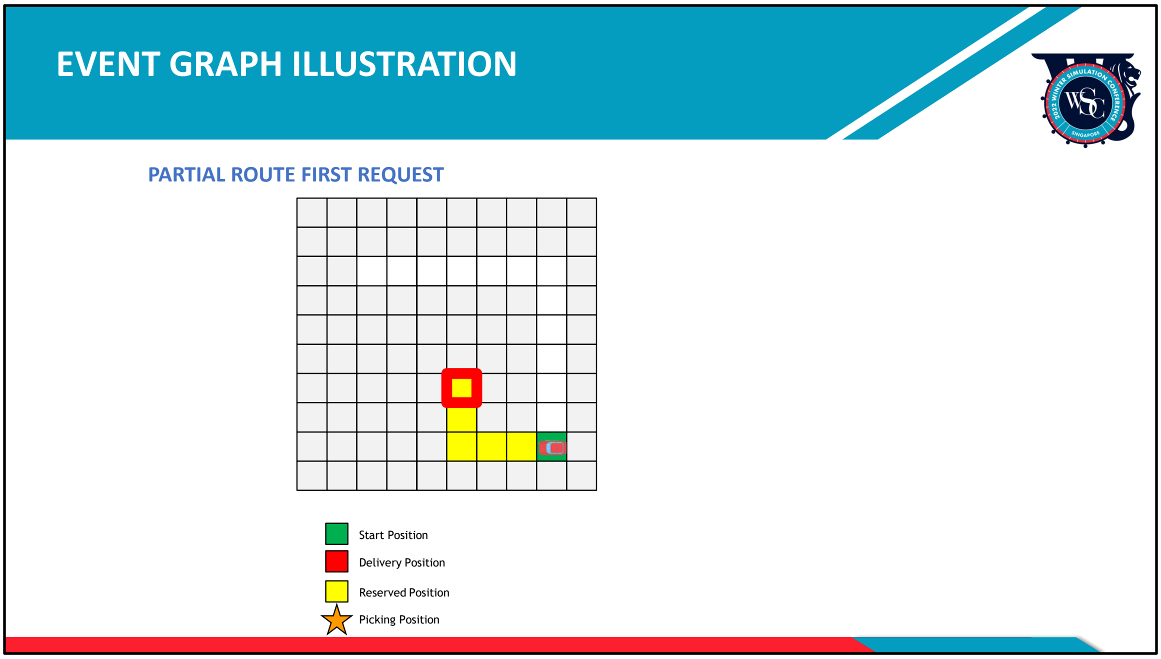
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

After the job item is loaded onto the vehicle, it needs to transport the job item to the delivery position. Therefore, the route event will be triggered to get the next full route necessary for delivery.



EVENT GRAPH ILLUSTRATION

PARTIAL ROUTE FIRST REQUEST

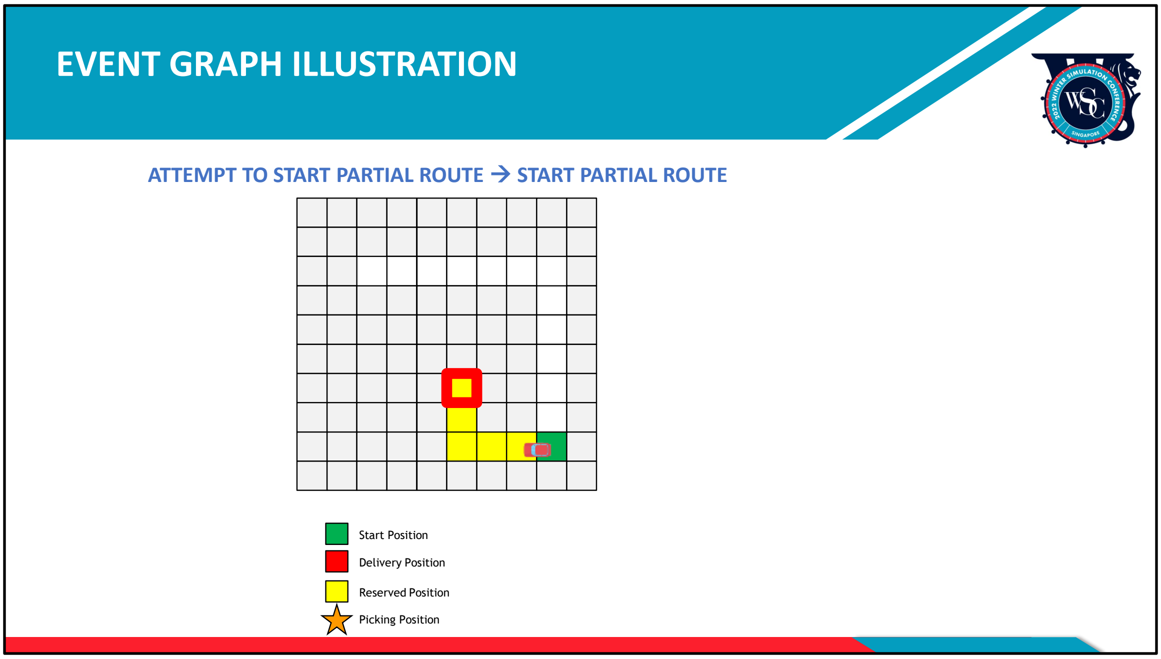
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

After a full route is obtained, the partial_route_first_request event will be triggered to reserve square units accordingly.



EVENT GRAPH ILLUSTRATION

ATTEMPT TO START PARTIAL ROUTE → START PARTIAL ROUTE

🟩 Start Position

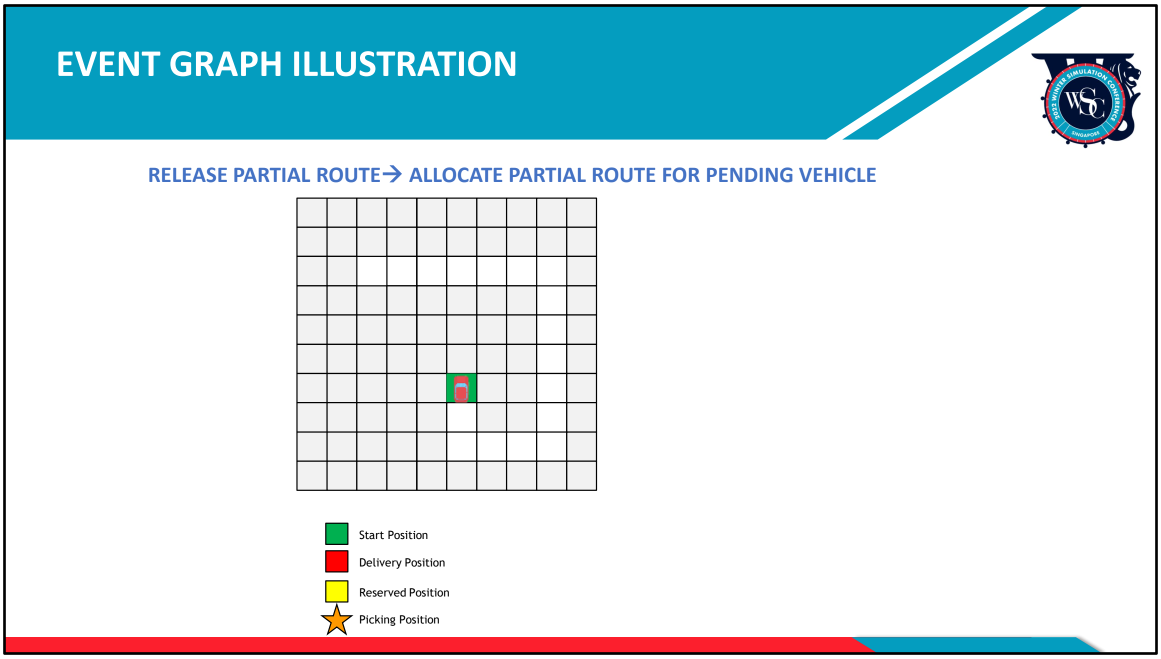
🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

If the partial route is reserved successfully, the attempt_to_start_partial_route event will be triggered and start_partial_route event will be further triggered. After which, the vehicle starts to move.

Since the partial route length covers the full route length, there is no need to reserve the next partial route. The vehicle will move directly to the delivery position.



EVENT GRAPH ILLUSTRATION

RELEASE PARTIAL ROUTE→ ALLOCATE PARTIAL ROUTE FOR PENDING VEHICLE

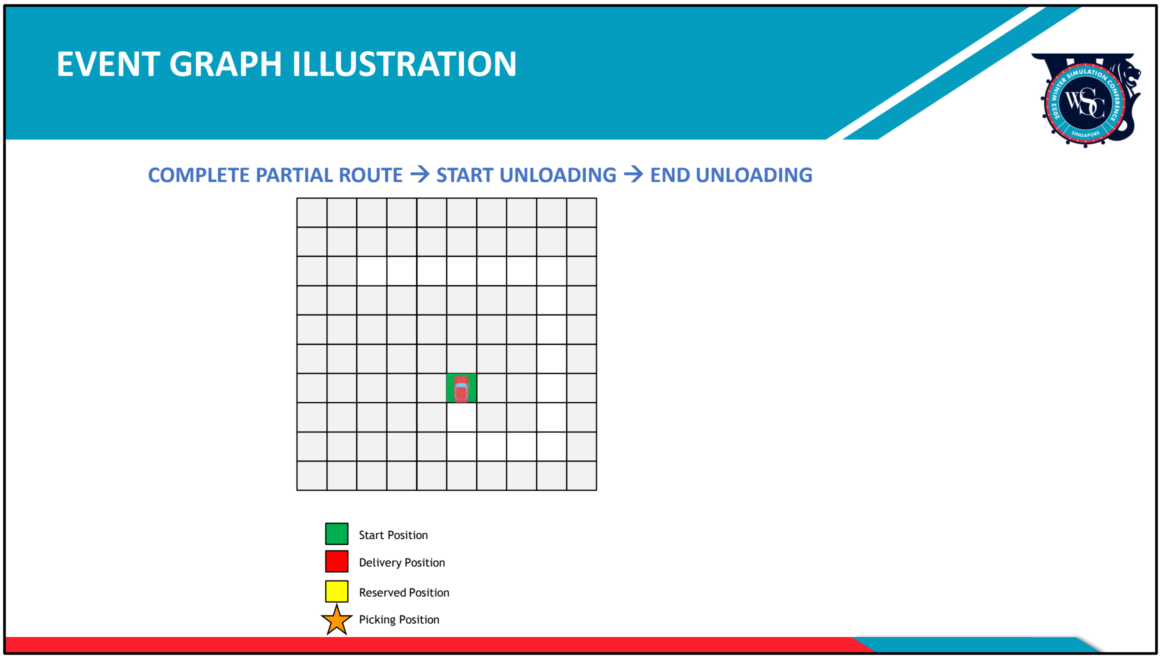
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

When the vehicle has finished the partial route, and the current occupied square unit is the delivery position, the release_partial_route event will be triggered. And the allocate_partial_route_for_pending_vehicle event will be triggered if there are pending vehicles in system like before.



EVENT GRAPH ILLUSTRATION

COMPLETE PARTIAL ROUTE → START UNLOADING → END UNLOADING

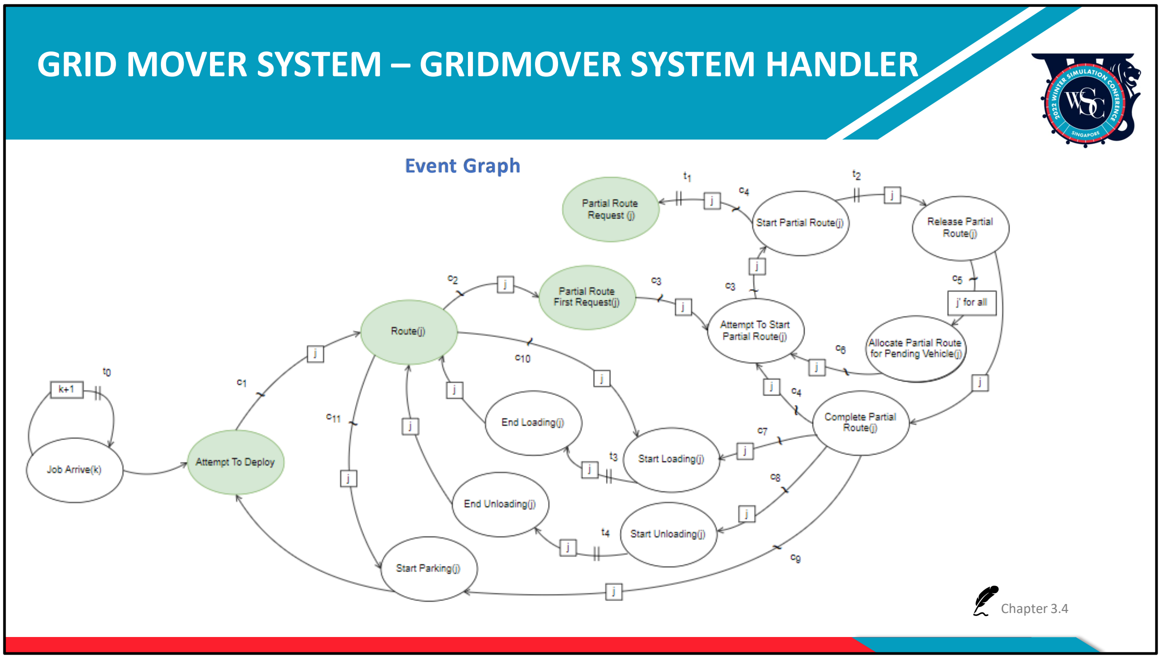
🟩 Start Position

🟥 Delivery Position

🟨 Reserved Position

★ Picking Position

After the partial route is released, the complete_partial_route event will be triggered. Because the full route is finished, the complete_partial_route event will trigger the start_unloading event. After the vehicle has finished unloading, the end_unloading event will be scheduled. The vehicle has now finished the job successfully.



GRID MOVER SYSTEM – GRIDMOVER SYSTEM HANDLER

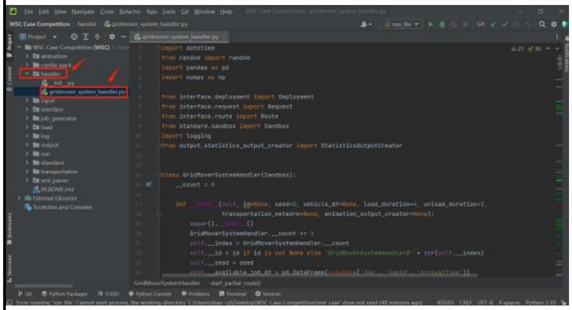
Event Graph

Chapter 3.4

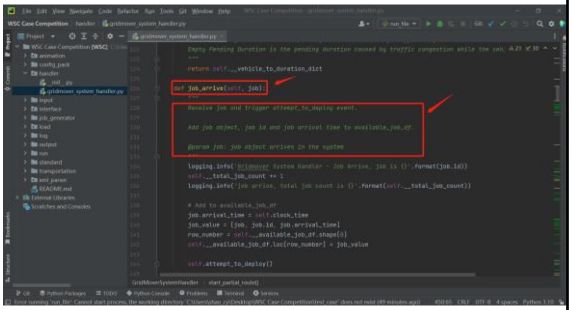
The route event will be triggered again if the vehicle has other jobs to do. The route event will trigger the partial_route_first_request event again to move to the picking position of the new job. But if the current position of vehicle happens to be the picking position of the new job, the route event will directly trigger the start_loading event to load new job items onto the vehicle. However, if the vehicle has no other jobs to process, it will move to its parking position. If the current position is not its parking position, the route event must trigger partial_route _first_request event, and all other previously mentioned events will be triggered sequentially until the start_parking event is triggered. If the start position is exactly in its parking position, the route event can directly trigger the start_parking event. When the vehicle reaches its parking position, the start_parking event will trigger the attempt_to_deploy event to match new jobs coming into the system.

Detailed description of all events displayed on the event graph is provided in Chapter 3.4 of the tech document.

GRID MOVER SYSTEM – GRIDMOVER SYSTEM HANDLER

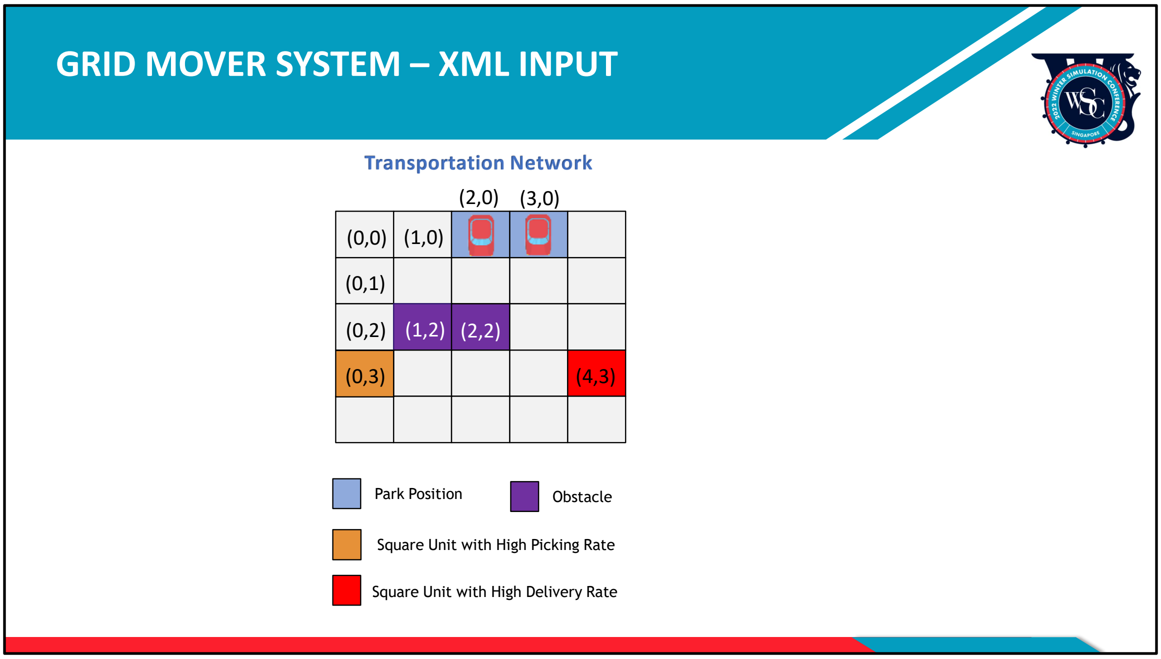


```
import datetime
from random import random
import pandas as pd
import numpy as np

from interface.deployment import Deployment
from interface.request import Request
from interface.route import Route
from standard.sandbox import Sandbox
import logging
from output.statistics_output_creator import StatisticsOutputCreator


class GridMoverSystemHandler(Sandbox):
    __count = 0

    def __init__(self, id=None, seed=0, vehicle_df=None, load_duration=4, unload_duration=3,
                 transportation_network=None, animation_output_creator=None):
        super().__init__()
        GridMoverSystemHandler.__count += 1
        self.__index = GridMoverSystemHandler.__count
        self.__id = id if id is not None else 'GridMoverSystemHandler#' + str(self.__index)
        self.__seed = seed
        self.__available_job_df = pd.DataFrame(columns=['Job', 'JobId', 'ArrivalTime'])
```



```
        Empty Pending Duration is the pending duration caused by traffic congestion while the veh
        """
        return self.__vehicle_to_duration_dict

    def job_arrive(self, job):
        """
        Receive job and trigger attempt_to_deploy event.

        Add job object, job id and job arrival time to available_job_df.

        @param job: job object arrives in the system
        """
        logging.info('Gridmover System Handler - Job Arrive, job is {}'.format(job.id))
        self.__total_job_count += 1
        logging.info('job arrive, total job count is {}'.format(self.__total_job_count))

        # Add to available_job_df
        job_arrival_time = self.clock_time
        job_value = [job, job.id, job.arrival_time]
        row_number = self.__available_job_df.shape[0]
        self.__available_job_df.loc[row_number] = job_value

        self.attempt_to_deploy()
```

All these events are written into gridmover_system_handler.py file. The grid mover system handler manages events by using O2DES.PY structure, and each event has comment.



GRID MOVER SYSTEM – XML INPUT

Transportation Network

		(2,0)	(3,0)	
(0,0)	(1,0)			
(0,1)				
(0,2)	(1,2)	(2,2)		
(0,3)				(4,3)

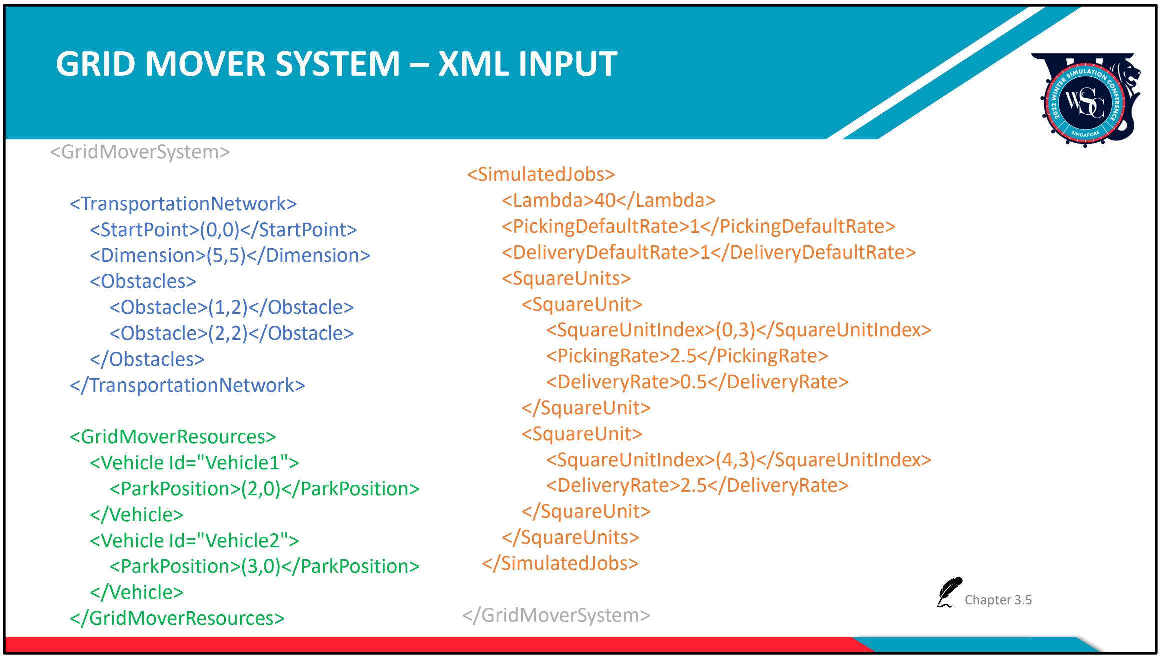
🟦 Park Position 🟪 Obstacle

🟧 Square Unit with High Picking Rate

🟥 Square Unit with High Delivery Rate

Now that we know how the system works, we should consider what input our system needs. As seen before, we will now give each square unit a unique index in the transportation network of our simulation system.

The square unit on the first row and first column is given (0,0), the square unit on the first row and second column is (1,0), and so on. We can also see that the parking positions are at (2,0) and (3,0) respectively, while the two obstacles are at (1,2) and (2,2) respectively. The square unit with the high picking rate is at (0,3), while the square unit with high delivery rate is at (4,3).



GRID MOVER SYSTEM – XML INPUT

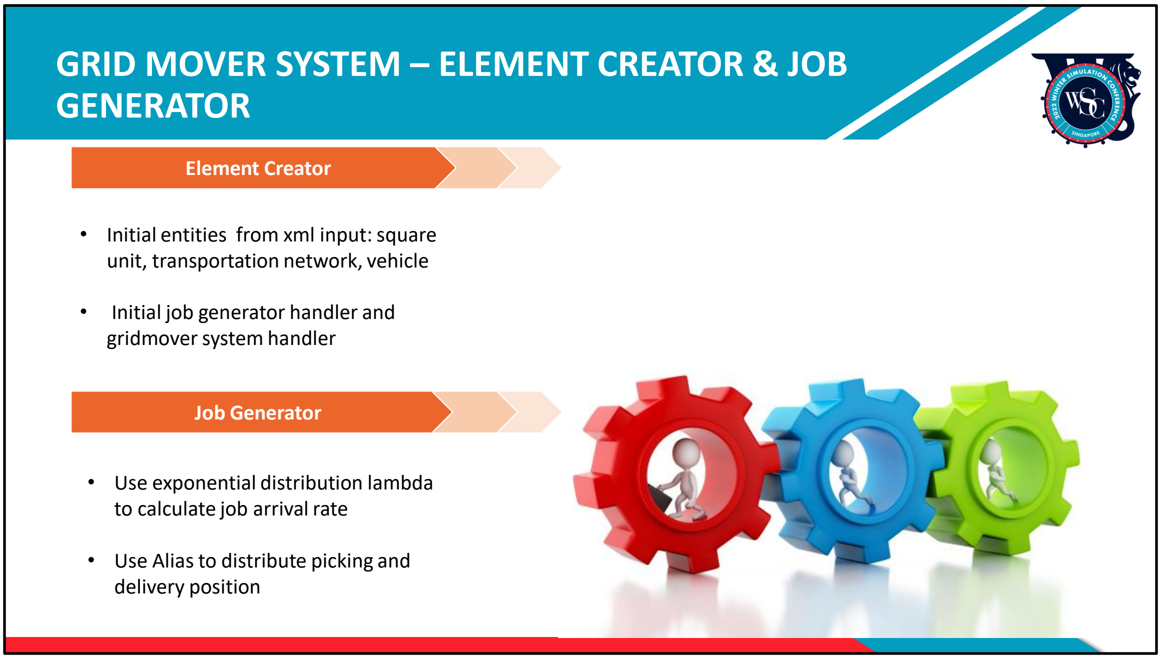
```xml
<GridMoverSystem>

  <TransportationNetwork>
    <StartPoint>(0,0)</StartPoint>
    <Dimension>(5,5)</Dimension>
    <Obstacles>
      <Obstacle>(1,2)</Obstacle>
      <Obstacle>(2,2)</Obstacle>
    </Obstacles>
  </TransportationNetwork>

  <GridMoverResources>
    <Vehicle Id="Vehicle1">
      <ParkPosition>(2,0)</ParkPosition>
    </Vehicle>
    <Vehicle Id="Vehicle2">
      <ParkPosition>(3,0)</ParkPosition>
    </Vehicle>
  </GridMoverResources>

<SimulatedJobs>
    <Lambda>40</Lambda>
    <PickingDefaultRate>1</PickingDefaultRate>
    <DeliveryDefaultRate>1</DeliveryDefaultRate>
    <SquareUnits>
      <SquareUnit>
         <SquareUnitIndex>(0,3)</SquareUnitIndex>
         <PickingRate>2.5</PickingRate>
         <DeliveryRate>0.5</DeliveryRate>
      </SquareUnit>
      <SquareUnit>
         <SquareUnitIndex>(4,3)</SquareUnitIndex>
         <DeliveryRate>2.5</DeliveryRate>
      </SquareUnit>
    </SquareUnits>
  </SimulatedJobs>

</GridMoverSystem>
```

Chapter 3.5

We will now enter all the information in XML, which is our XML input file.

We can see that the transportation start point is (0,0) and that the dimensions is a 5*5 grid. We can also see the square unit index for the obstacles as well as the two vehicles' parking positions. With all the inputs on transportation, Jobs are then automatically generated next. We use exponential distribution to generate jobs and this lambda means approximately how many jobs are generated within 1 hour. Since we know different square units have different picking and delivery rates, we assume all default rates are 1 and other square units with special picking or delivery rates are listed below. Since the picking rate of (0,3) is higher than 1, this means that this square unit has a High Picking Rate.

Further explanations on input is provided in Chapter 3.5 of the tech document.



GRID MOVER SYSTEM – ELEMENT CREATOR & JOB GENERATOR

Element Creator

- Initial entities from xml input: square unit, transportation network, vehicle
- Initial job generator handler and gridmover system handler

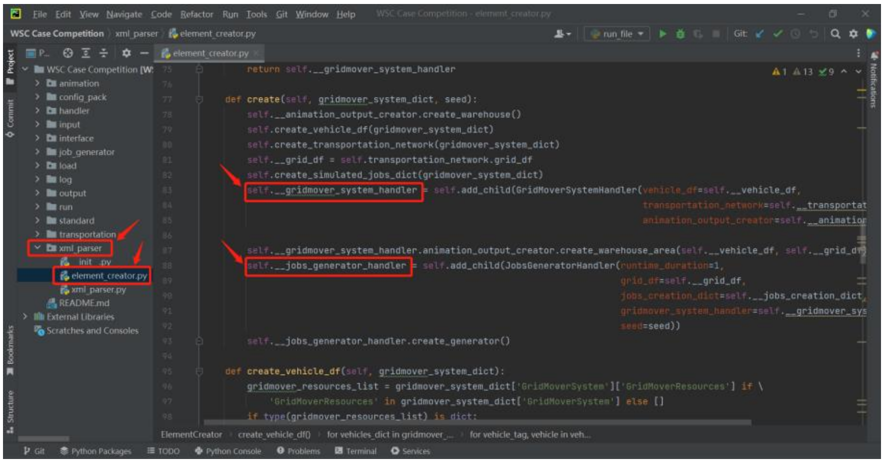
Job Generator

- Use exponential distribution lambda to calculate job arrival rate
- Use Alias to distribute picking and delivery position

In order to read XML input file in the system, we need an element creator file and a job generator file. The element creator file has two main tasks. One, it initializes entities from a XML input file (e.g. square unit, transportation network and vehicle). Second, it initializes handlers (e.g. job generator handler and grid mover system handler).

Now let's take a look at the job generator. Exponential distribution is applied to obtain the job arrival rate. The Lambda for this exponential distribution is included in the XML file. The job generator adopts Alias to randomize the job's picking and delivery position based on possibilities from the XML file.

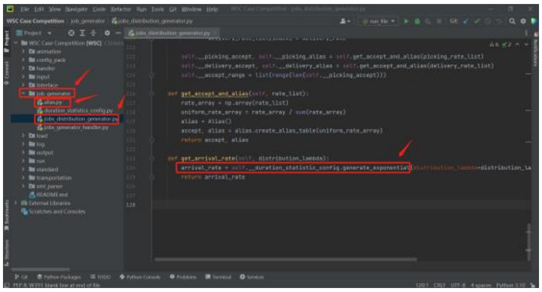
GRID MOVER SYSTEM – ELEMENT CREATOR



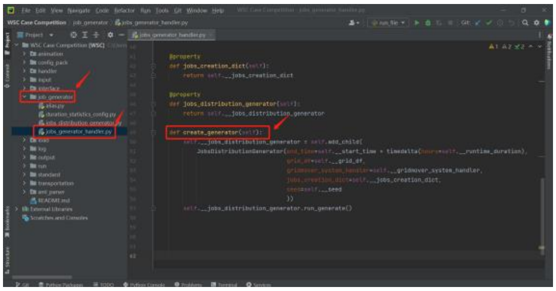
```
        return self.__gridmover_system_handler

    def create(self, gridmover_system_dict, seed):
        self.__animation_output_creator.create_warehouse()
        self.create_vehicle_df(gridmover_system_dict)
        self.create_transportation_network(gridmover_system_dict)
        self.__grid_df = self.transportation_network.grid_df
        self.create_simulated_jobs_dict(gridmover_system_dict)
        self.__gridmover_system_handler = self.add_child(GridMoverSystemHandler(vehicle_df=self.__vehicle_df,
                                                                                transportation_network=self.__transportat
                                                                                animation_output_creator=self.__animatior

        self.__gridmover_system_handler.animation_output_creator.create_warehouse_area(self.__vehicle_df, self.__grid_df
        self.__jobs_generator_handler = self.add_child(JobsGeneratorHandler(runtime_duration=1,
                                                                            grid_df=self.__grid_df,
                                                                            jobs_creation_dict=self.__jobs_creation_dict,
                                                                            gridmover_system_handler=self.__gridmover_sys
                                                                            seed=seed))

        self.__jobs_generator_handler.create_generator()

    def create_vehicle_df(self, gridmover_system_dict):
        gridmover_resources_list = gridmover_system_dict['GridMoverSystem']['GridMoverResources'] if \
            'GridMoverResources' in gridmover_system_dict['GridMoverSystem'] else []
        if type(gridmover_resources_list) is dict:
```

Looking at these two files in source code, the element creator is in the xml_parser folder, and through the create function, we can see that the job generator handler and grid mover system handler are initialized.

GRID MOVER SYSTEM – JOB GENERATOR



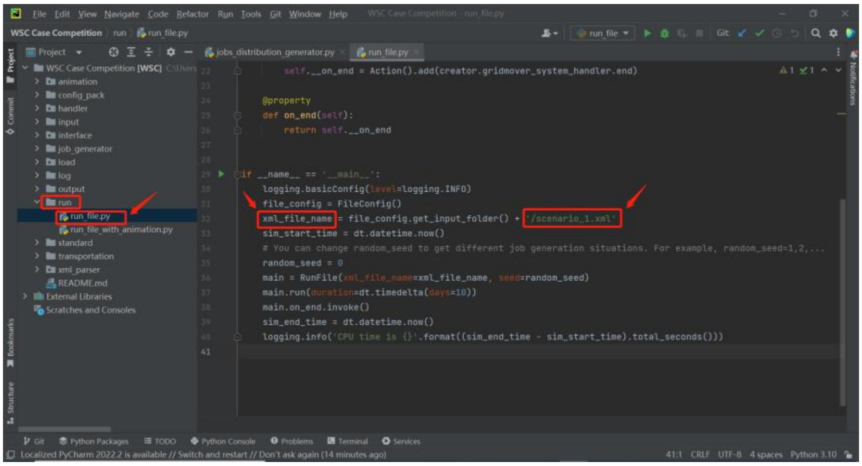
```
        self.__picking_accept, self.__picking_alias = self.get_accept_and_alias(picking_rate_list)
        self.__delivery_accept, self.__delivery_alias = self.get_accept_and_alias(delivery_rate_list)
        self.__accept_range = list(range(len(self.__picking_accept)))

    def get_accept_and_alias(self, rate_list):
        rate_array = np.array(rate_list)
        uniform_rate_array = rate_array / sum(rate_array)
        alias = Alias()
        accept, alias = alias.create_alias_table(uniform_rate_array)
        return accept, alias

    def get_arrival_rate(self, distribution_lambda):
        arrival_rate = self.__duration_statistic_config.generate_exponential(distribution_lambda=distribution_la
        return arrival_rate
```



```
    @property
    def jobs_creation_dict(self):
        return self.__jobs_creation_dict

    @property
    def jobs_distribution_generator(self):
        return self.__jobs_distribution_generator

    def create_generator(self):
        self.__jobs_distribution_generator = self.add_child(
            JobsDistributionGenerator(end_time=self.__start_time + timedelta(hours=self.__runtime_duration),
                                      grid_df=self.__grid_df,
                                      gridmover_system_handler=self.__gridmover_system_handler,
                                      jobs_creation_dict=self.__jobs_creation_dict,
                                      seed=self.__seed
                                      ))
        self.__jobs_distribution_generator.run_generate()
```

The job generator handler is in the job generator folder, which manages the job generator file. We can see that the job generator generates jobs based on exponential distribution and Alias.

GRID MOVER SYSTEM – RUN: RUN_FILE.PY

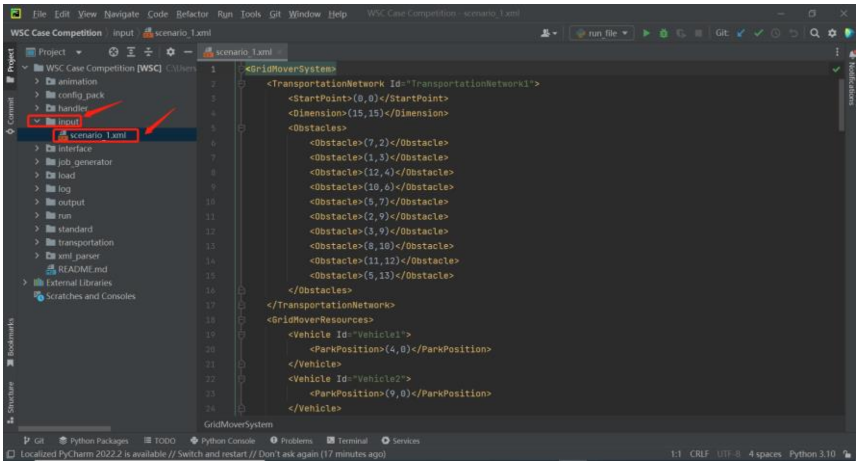

File Edit View Navigate Code Refactor Run Tools Git Window Help
WSC Case Competition - run_file.py
WSC Case Competition 〉 run 〉 run_file.py
run_file
Git:
Project
jobs_distribution_generator.py
run_file.py
WSC Case Competition [WSC] C:\Users
animation
config_pack
handler
input
interface
job_generator
load
log
output
run
run_file.py
run_file_with_animation.py
standard
transportation
xml_parser
README.md
External Libraries
Scratches and Consoles
Project
Commit
Bookmarks
Structure
Notifications
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
Localized PyCharm 2022.2 is available // Switch and restart // Don't ask again (14 minutes ago)
41:1 CRLF UTF-8 4 spaces Python 3.10

```python
        self.__on_end = Action().add(creator.gridmover_system_handler.end)

    @property
    def on_end(self):
        return self.__on_end


if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)
    file_config = FileConfig()
    xml_file_name = file_config.get_input_folder() + '/scenario_1.xml'
    sim_start_time = dt.datetime.now()
    # You can change random_seed to get different job generation situations. For example, random_seed=1,2,...
    random_seed = 0
    main = RunFile(xml_file_name=xml_file_name, seed=random_seed)
    main.run(duration=dt.timedelta(days=10))
    main.on_end.invoke()
    sim_end_time = dt.datetime.now()
    logging.info('CPU time is {}'.format((sim_end_time - sim_start_time).total_seconds()))
```

Now that we know more about the input and how to use it, the next step is to run the system. There are two running files in the run folder: run file and run file with animation.

Let's look at the run file first. Run file has the xml_file_name we wish to run.

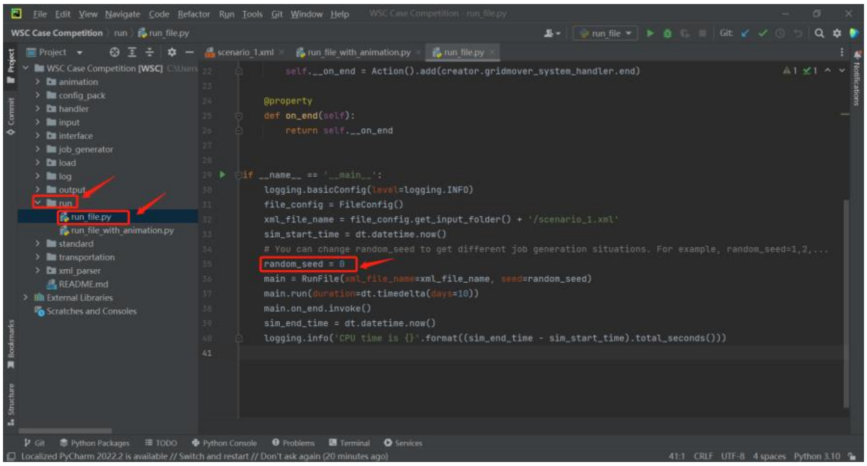
GRID MOVER SYSTEM – RUN: XML INPUT FILE



```
<GridMoverSystem>
    <TransportationNetwork Id="TransportationNetwork1">
        <StartPoint>(0,0)</StartPoint>
        <Dimension>(15,15)</Dimension>
        <Obstacles>
            <Obstacle>(7,2)</Obstacle>
            <Obstacle>(1,3)</Obstacle>
            <Obstacle>(12,4)</Obstacle>
            <Obstacle>(10,6)</Obstacle>
            <Obstacle>(5,7)</Obstacle>
            <Obstacle>(2,9)</Obstacle>
            <Obstacle>(3,9)</Obstacle>
            <Obstacle>(8,10)</Obstacle>
            <Obstacle>(11,12)</Obstacle>
            <Obstacle>(5,13)</Obstacle>
        </Obstacles>
    </TransportationNetwork>
    <GridMoverResources>
        <Vehicle Id="Vehicle1">
            <ParkPosition>(4,0)</ParkPosition>
        </Vehicle>
        <Vehicle Id="Vehicle2">
            <ParkPosition>(9,0)</ParkPosition>
        </Vehicle>
```

For now, the input file is scenario_1. When the competition progresses to Round 2 and 3 where will provide scenario_2 and scenario_3 respectively, you can put all these scenario files in the input folder and change the name accordingly in run_file.py and run_file_with_animation.py which will be shown later. The system will run the file you need.

GRID MOVER SYSTEM – RUN: RUN_FILE.PY

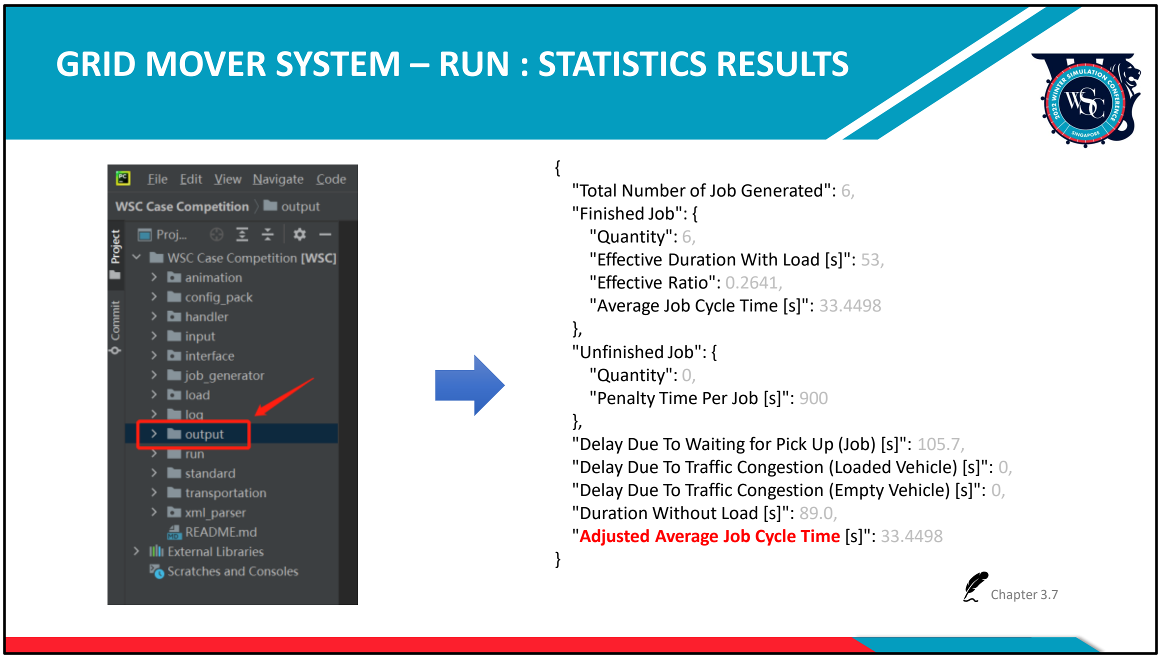


```python
        self.__on_end = Action().add(creator.gridmover_system_handler.end)

    @property
    def on_end(self):
        return self.__on_end


if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)
    file_config = FileConfig()
    xml_file_name = file_config.get_input_folder() + '/scenario_1.xml'
    sim_start_time = dt.datetime.now()
    # You can change random_seed to get different job generation situations. For example, random_seed=1,2,...
    random_seed = 0
    main = RunFile(xml_file_name=xml_file_name, seed=random_seed)
    main.run(duration=dt.timedelta(days=10))
    main.on_end.invoke()
    sim_end_time = dt.datetime.now()
    logging.info('CPU time is {}'.format((sim_end_time - sim_start_time).total_seconds()))
```

Localized PyCharm 2022.2 is available // Switch and restart // Don't ask again (20 minutes ago)

The random seed is at 0. When we test your code, we will use multiple random seeds to achieve the average performance.

When you click the run button, the project will run successfully.



GRID MOVER SYSTEM – RUN : STATISTICS RESULTS

```
{
  "Total Number of Job Generated": 6,
  "Finished Job": {
    "Quantity": 6,
    "Effective Duration With Load [s]": 53,
    "Effective Ratio": 0.2641,
    "Average Job Cycle Time [s]": 33.4498
  },
  "Unfinished Job": {
    "Quantity": 0,
    "Penalty Time Per Job [s]": 900
  },
  "Delay Due To Waiting for Pick Up (Job) [s]": 105.7,
  "Delay Due To Traffic Congestion (Loaded Vehicle) [s]": 0,
  "Delay Due To Traffic Congestion (Empty Vehicle) [s]": 0,
  "Duration Without Load [s]": 89.0,
  "Adjusted Average Job Cycle Time [s]": 33.4498
}
```

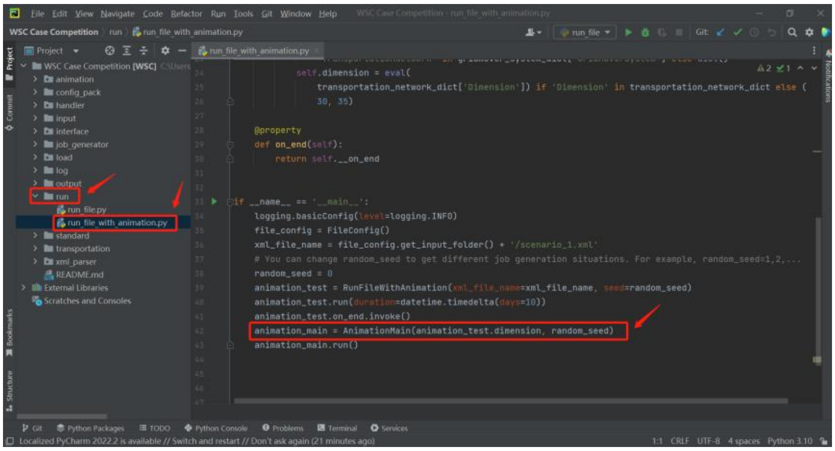
Chapter 3.7

Now, after running successfully, we can see a JSON output file from the output folder. Each time you run the system, the JSON file will be updated. We will grade the system based on the output indexes.

Let's look at the output in detail. It lists the total number of generated, finished and unfinished jobs. Each unfinished job will be allocated a penalty time of 4 times the duration used for 1 vehicle to travel the whole transportation network. Our main criterion of judgment is the adjusted average job cycle time, which shows the average time used by one job from arriving to being delivered in the system. Other outputs such as delay due to waiting for pick up and duration without load are given to participants as a reference to improve and refine their system.

Participants can check each statistic result explanation in Chapter 3.7 of the tech document.

GRID MOVER SYSTEM – RUN: ANIMATION



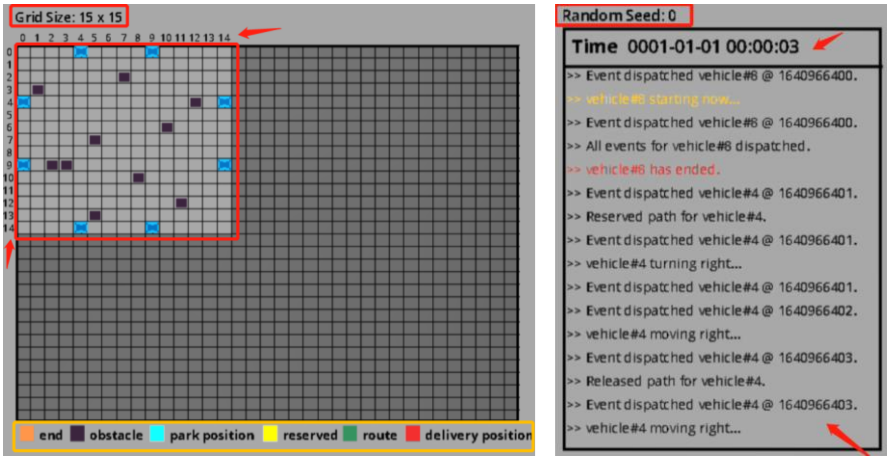
```
            self.dimension = eval(
                transportation_network_dict['Dimension']) if 'Dimension' in transportation_network_dict else (
                30, 35)

    @property
    def on_end(self):
        return self.__on_end


if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)
    file_config = FileConfig()
    xml_file_name = file_config.get_input_folder() + '/scenario_1.xml'
    # You can change random_seed to get different job generation situations. For example, random_seed=1,2,...
    random_seed = 0
    animation_test = RunFileWithAnimation(xml_file_name=xml_file_name, seed=random_seed)
    animation_test.run(duration=datetime.timedelta(days=10))
    animation_test.on_end.invoke()
    animation_main = AnimationMain(animation_test.dimension, random_seed)
    animation_main.run()
```

Localized PyCharm 2022.2 is available // Switch and restart // Don't ask again (21 minutes ago)

In order to show the system running, we have provided the run file with animation. When running the project, a 2D animation will be shown.

GRID MOVER SYSTEM – RUN: ANIMATION

2D Animation



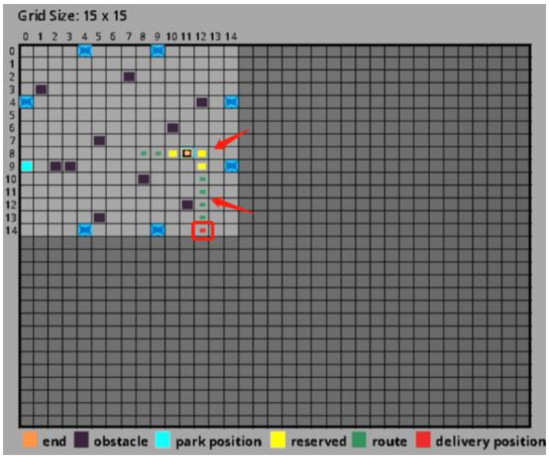
Random Seed: 0

```
Time  0001-01-01 00:00:03
>> Event dispatched vehicle#8 @ 1640966400.
>> vehicle#8 starting now...
>> Event dispatched vehicle#8 @ 1640966400.
>> All events for vehicle#8 dispatched.
>> vehicle#8 has ended.
>> Event dispatched vehicle#4 @ 1640966401.
>> Reserved path for vehicle#4.
>> Event dispatched vehicle#4 @ 1640966401.
>> vehicle#4 turning right...
>> Event dispatched vehicle#4 @ 1640966401.
>> Event dispatched vehicle#4 @ 1640966402.
>> vehicle#4 moving right...
>> Event dispatched vehicle#4 @ 1640966403.
>> Released path for vehicle#4.
>> Event dispatched vehicle#4 @ 1640966403.
>> vehicle#4 moving right...
```

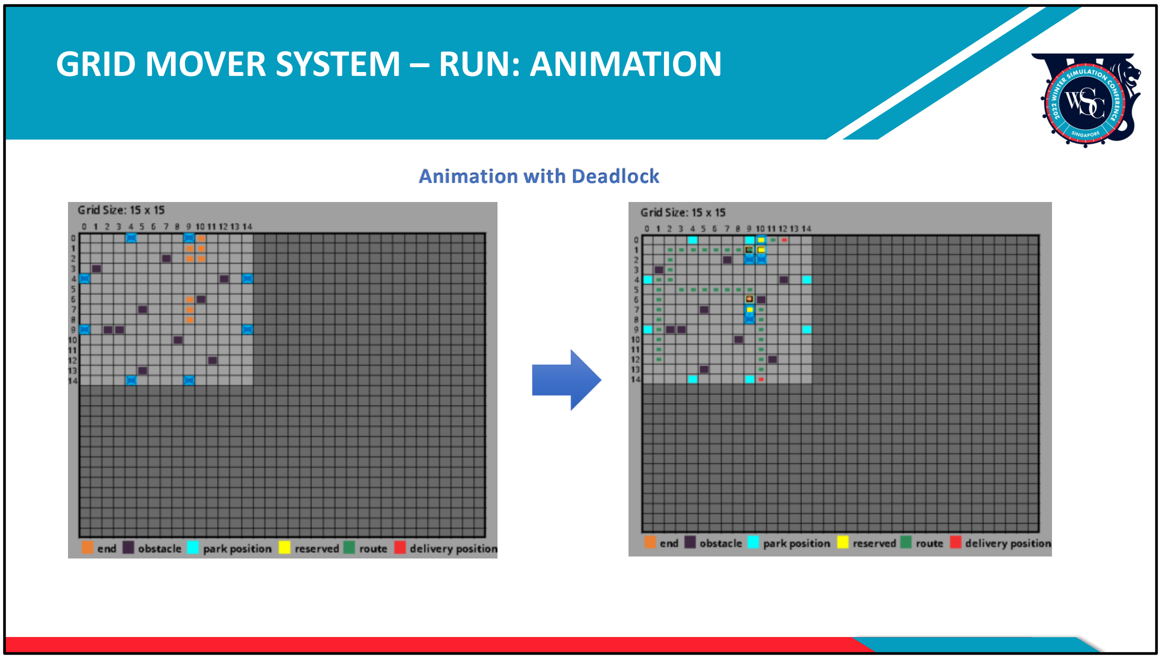
The light gray area is our transportation network area, now set at 15*15. The column number and row number are indicated on the sides. There is a legend provided below the grid indicating what each colour block represents. The right-hand side box indicates simulation seed applied, time and vehicle travel information.

GRID MOVER SYSTEM – RUN: ANIMATION

2D Animation



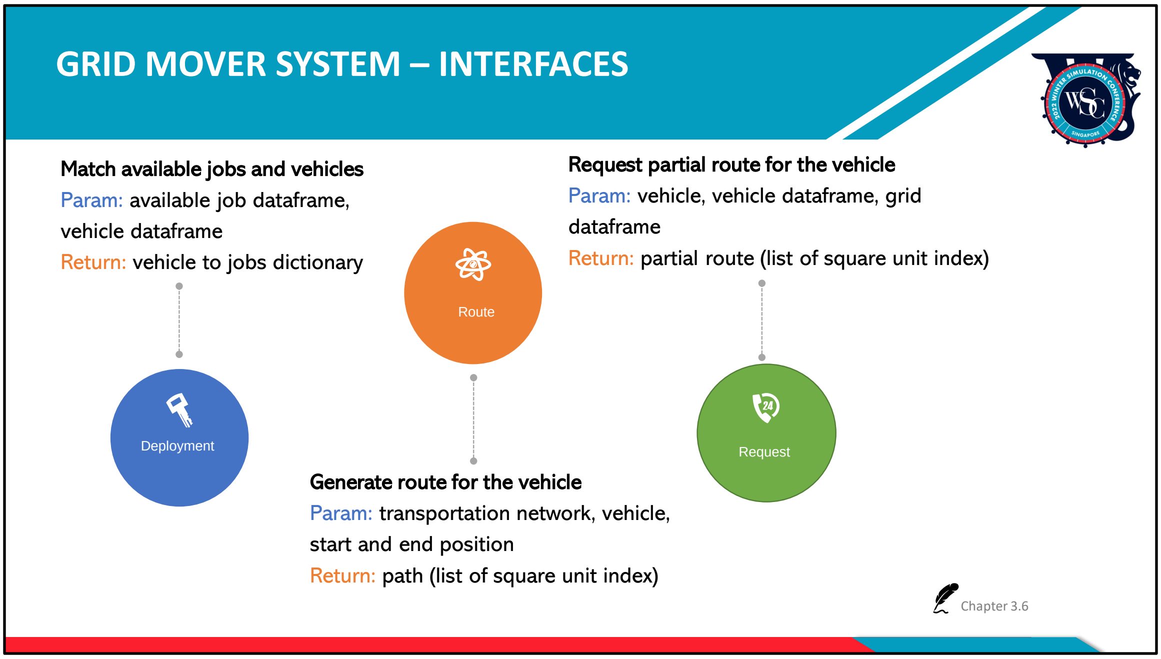
We can see the vehicle move out from its parking position. The green square units are its job route. The vehicle will move to the picking position to load the job item and then travel to the red delivery position to unload. Once the job is completed, if there are no new jobs, the vehicle will travel to its parking position. Otherwise, it will travel to its next location to pick up the next job.



GRID MOVER SYSTEM – RUN: ANIMATION

Animation with Deadlock

On the left image, the orange square units that represent the end positions will show up before vehicles start to move. This means that deadlock has happened, and vehicles cannot continue travelling. When the run is over, it will look like the image on the right, where some jobs are unfinished.



GRID MOVER SYSTEM – INTERFACES

Match available jobs and vehicles
Param: available job dataframe, vehicle dataframe
Return: vehicle to jobs dictionary

Deployment

Route

Generate route for the vehicle
Param: transportation network, vehicle, start and end position
Return: path (list of square unit index)

Request partial route for the vehicle
Param: vehicle, vehicle dataframe, grid dataframe
Return: partial route (list of square unit index)

Request

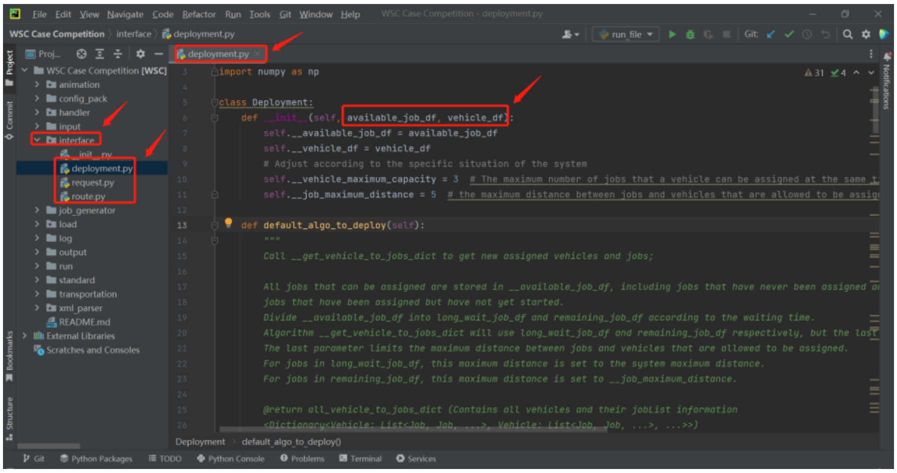
Chapter 3.6

Last but not least, what else can be modified in this system? The answer is, three interfaces can be modified: deployment, route, request. These three interfaces are connected to the events in green mentioned earlier.

The Deployment interface matches available jobs and vehicles. We will give this interface an available job dataframe which contains information on jobs as well as a vehicle dataframe with information on vehicles, and the Deployment interface needs to return a dictionary containing vehicles and jobs. The Route interface generates routes for vehicles. The parameters of this interface are the transportation network, the vehicle that requires the route as well as the start and end position of the route. The Route interface needs to return a full path for the vehicle, which is a list of square unit index. The Request interface requests partial routes for vehicles, and this interface provides the following: vehicle asking for partial route, vehicle dataframe, grid dataframe which contains info of transportation network. The Request interface needs to return a partial route, which is a list of square unit index. For more details, please refer to chapter 3.6 "Interface" of the tech document.

There are also three important dataframes here, available_job_df, vehicle_df and grid_df. Please see chapter 3.2 of the tech document for more details.

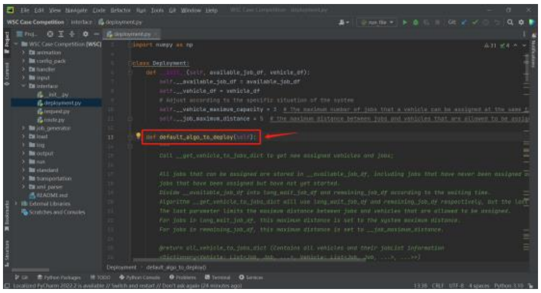
GRID MOVER SYSTEM – INTERFACES



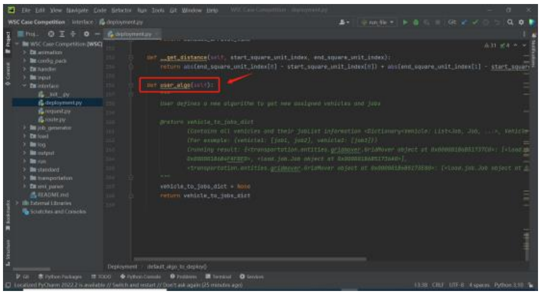
```python
import numpy as np

class Deployment:
    def __init__(self, available_job_df, vehicle_df):
        self.__available_job_df = available_job_df
        self.__vehicle_df = vehicle_df
        # Adjust according to the specific situation of the system
        self.__vehicle_maximum_capacity = 3  # The maximum number of jobs that a vehicle can be assigned at the same t
        self.__job_maximum_distance = 5  # the maximum distance between jobs and vehicles that are allowed to be assig

    def default_algo_to_deploy(self):
        """
        Call __get_vehicle_to_jobs_dict to get new assigned vehicles and jobs;

        All jobs that can be assigned are stored in __available_job_df, including jobs that have never been assigned ar
        jobs that have been assigned but have not yet started.
        Divide __available_job_df into long_wait_job_df and remaining_job_df according to the waiting time.
        Algorithm __get_vehicle_to_jobs_dict will use long_wait_job_df and remaining_job_df respectively, but the last
        The last parameter limits the maximum distance between jobs and vehicles that are allowed to be assigned.
        For jobs in long_wait_job_df, this maximum distance is set to the system maximum distance.
        For jobs in remaining_job_df, this maximum distance is set to __job_maximum_distance.

        @return all_vehicle_to_jobs_dict (Contains all vehicles and their jobList information
        <Dictionary<Vehicle: List<Job, Job, ...>, Vehicle: List<Job, Job, ...>, ...>>)
```

Now, let’s see these three interfaces in source code. All interfaces are in the interface folder. Using Deployment as an example, all required parameters are given to deployment.

GRID MOVER SYSTEM – INTERFACES


File Edit View Navigate Code Refactor Run Tools Git Window Help
WSC Case Competition › interface › deployment.py
run_file
WSC Case Competition [WSC]
animation
config_pack
handler
input
interface
__init__.py
deployment.py
request.py
route.py
job_generator
load
log
output
run
standard
transportation
xml_parser
README.md
External Libraries
Scratches and Consoles
Deployment › default_algo_to_deploy()
Git Python Packages TODO Python Console Problems Terminal Services
Localized PyCharm 2022.2 is available // Switch and restart // Don't ask again (24 minutes ago)
13:38 CRLF UTF-8 4 spaces Python 3.10

```
import numpy as np

class Deployment:
    def __init__(self, available_job_df, vehicle_df):
        self.__available_job_df = available_job_df
        self.__vehicle_df = vehicle_df
        # Adjust according to the specific situation of the system
        self.__vehicle_maximum_capacity = 3  # The maximum number of jobs that a vehicle can be assigned at the same t
        self.__job_maximum_distance = 5  # the maximum distance between jobs and vehicles that are allowed to be assig

    def default_algo_to_deploy(self):
        """
        Call __get_vehicle_to_jobs_dict to get new assigned vehicles and jobs;

        All jobs that can be assigned are stored in __available_job_df, including jobs that have never been assigned a
        jobs that have been assigned but have not yet started.
        Divide __available_job_df into long_wait_job_df and remaining_job_df according to the waiting time.
        Algorithm __get_vehicle_to_jobs_dict will use long_wait_job_df and remaining_job_df respectively, but the last
        The last parameter limits the maximum distance between jobs and vehicles that are allowed to be assigned.
        For jobs in long_wait_job_df, this maximum distance is set to the system maximum distance.
        For jobs in remaining_job_df, this maximum distance is set to __job_maximum_distance.

        @return all_vehicle_to_jobs_dict (Contains all vehicles and their joblist information
        <Dictionary<Vehicle, List<Job, Job, ...>, Vehicle: List<Job, Job, ...>, ...>>)
```


File Edit View Navigate Code Refactor Run Tools Git Window Help
WSC Case Competition › interface › deployment.py
run_file
WSC Case Competition [WSC]
animation
config_pack
handler
input
interface
__init__.py
deployment.py
request.py
route.py
job_generator
load
log
output
run
standard
transportation
xml_parser
README.md
External Libraries
Scratches and Consoles
Deployment › default_algo_to_deploy()
Git Python Packages TODO Python Console Problems Terminal Services
Localized PyCharm 2022.2 is available // Switch and restart // Don't ask again (25 minutes ago)
13:38 CRLF UTF-8 4 spaces Python 3.10

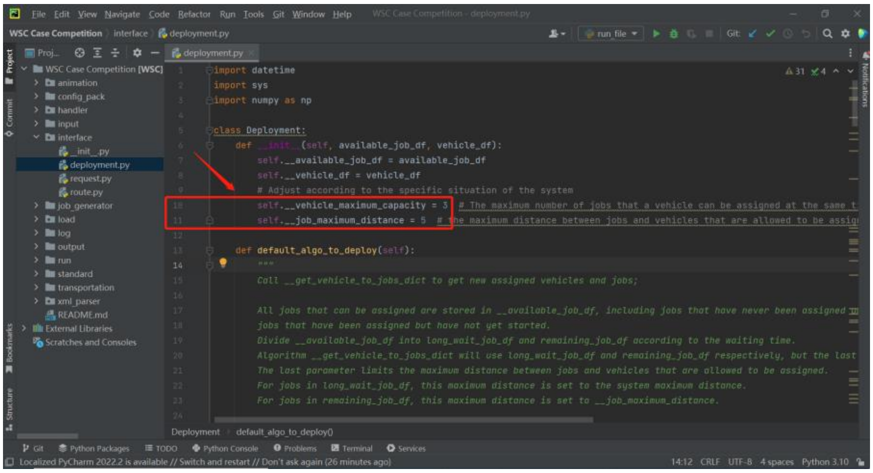
```
    def __get_distance(self, start_square_unit_index, end_square_unit_index):
        return abs(end_square_unit_index[0] - start_square_unit_index[0]) + abs(end_square_unit_index[1] - start_squar

    def user_algo(self):
        """
        User defines a new algorithm to get new assigned vehicles and jobs

        @return vehicle_to_jobs_dict
                (Contains all vehicles and their joblist information <Dictionary<Vehicle: List<Job, Job, ...>, Vehicle
                (for example: {vehicle1: [job1, job2], vehicle2: [job3]})
                (running result: {<transportation.entities.gridmover.GridMover object at 0x000001BAB5173700>: [<load.jo
                0x000001BAB4F4F8E0>, <load.job.Job object at 0x000001BAB5173A40>],
                <transportation.entities.gridmover.GridMover object at 0x000001BAB5173E80>: [<load.job.Job object at 
        """
        vehicle_to_jobs_dict = None
        return vehicle_to_jobs_dict
```

There are two important methods, namely default_algo_to_deploy and user_algo.

default_algo_to_deploy is the algorithm the system uses currently, while user_algo is currently empty and only has comments of description of return.

Participants need to write their own algorithm under user_algo, and the system will run their algorithms automatically.

GRID MOVER SYSTEM – INTERFACES


File Edit View Navigate Code Refactor Run Tools Git Window Help
WSC Case Competition - deployment.py
WSC Case Competition › interface › deployment.py
run file
Git:
Project
WSC Case Competition [WSC]
animation
config_pack
handler
input
interface
__init__.py
deployment.py
request.py
route.py
job_generator
load
log
output
run
standard
transportation
xml_parser
README.md
External Libraries
Scratches and Consoles
Commit
Bookmarks
Structure
Notifications
Deployment › default_algo_to_deploy()
Git
Python Packages
TODO
Python Console
Problems
Terminal
Services
Localized PyCharm 2022.2 is available // Switch and restart // Don't ask again (26 minutes ago)
14:12 CRLF UTF-8 4 spaces Python 3.10

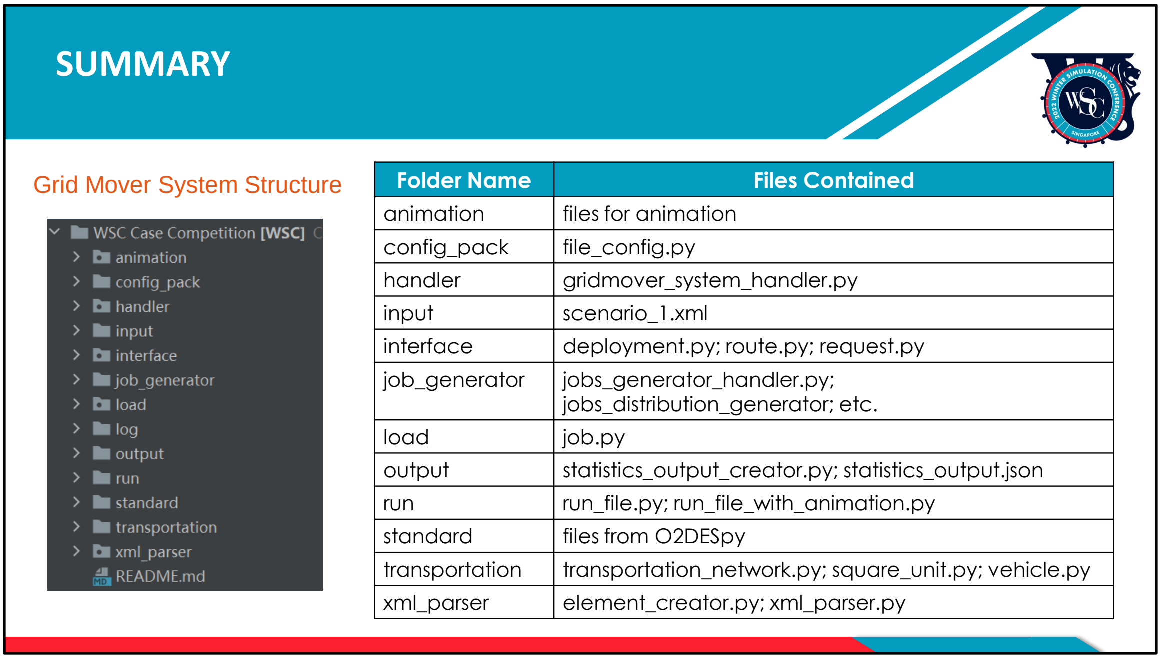
```
1  import datetime
2  import sys
3  import numpy as np
4
5  class Deployment:
6      def __init__(self, available_job_df, vehicle_df):
7          self.__available_job_df = available_job_df
8          self.__vehicle_df = vehicle_df
9          # Adjust according to the specific situation of the system
10         self.__vehicle_maximum_capacity = 3  # The maximum number of jobs that a vehicle can be assigned at the same t
11         self.__job_maximum_distance = 5  # The maximum distance between jobs and vehicles that are allowed to be assig
12
13     def default_algo_to_deploy(self):
14         """
15         Call __get_vehicle_to_jobs_dict to get new assigned vehicles and jobs;
16
17         All jobs that can be assigned are stored in __available_job_df, including jobs that have never been assigned
18         jobs that have been assigned but have not yet started.
19         Divide __available_job_df into long_wait_job_df and remaining_job_df according to the waiting time.
20         Algorithm __get_vehicle_to_jobs_dict will use long_wait_job_df and remaining_job_df respectively, but the last
21         The last parameter limits the maximum distance between jobs and vehicles that are allowed to be assigned.
22         For jobs in long_wait_job_df, this maximum distance is set to the system maximum distance.
23         For jobs in remaining_job_df, this maximum distance is set to __job_maximum_distance.
24
```

This is very easy to do even if you don't have simulation knowledge. You can use the default algorithm but only change the vehicle_maximum_capacity value or job_maximum_distance value in line 10 and 11, this is also accepted since the system will have a different performance. But of course, more improvements will achieve higher scores.

03 Summary

◆ Structure of Simulation System

Finally, let’s summarise the simulation system.



SUMMARY

Grid Mover System Structure

```
WSC Case Competition [WSC]
  > animation
  > config_pack
  > handler
  > input
  > interface
  > job_generator
  > load
  > log
  > output
  > run
  > standard
  > transportation
  > xml_parser
    README.md
```

Folder Name	Files Contained
animation	files for animation
config_pack	file_config.py
handler	gridmover_system_handler.py
input	scenario_1.xml
interface	deployment.py; route.py; request.py
job_generator	jobs_generator_handler.py; jobs_distribution_generator; etc.
load	job.py
output	statistics_output_creator.py; statistics_output.json
run	run_file.py; run_file_with_animation.py
standard	files from O2DESpy
transportation	transportation_network.py; square_unit.py; vehicle.py
xml_parser	element_creator.py; xml_parser.py

The animation folder only contains files for animation, participants are not required to understand them. This also applies to the config_pack.

The handler folder contains the gridmover system handler which manages all grid mover events. The input folder contains XML input files. The interface folder has three interfaces that are the only components that participants can change in the system.

The job generator folder contains files for generating jobs. The load folder contains job class. You may ignore the log folder.

The output folder contains JSON output files, the run folder has two running files: run file and run file with animation, the standard folder contains files from O2DES.PY, the transportation folder has transportation, square unit and vehicle class files.

Last but not least, the final xml parser folder contains the element creator file.

2022 WINTER SIMULATION CONFERENCE
WSC
SINGAPORE

THANK YOU

NUS National University of Singapore | Centre for Next Generation Logistics
Industrial Systems Engineering and Management
College of Design and Engineering

NUS National University of Singapore | Centre of Excellence in Modelling and Simulation for Next Generation Ports
Industrial Systems Engineering and Management
College of Design and Engineering

HUAWEI

Thank you for participating in the 2022 WSC case study competition. Good luck!