2022

WSC Case Competition Tech Document

Centre for Next Generation Logistics
Industrial Systems Engineering and Management
College of Design and Engineering

Centre of Excellence in Modelling and Simulation for Next Generation Ports
Industrial Systems Engineering and Management
College of Design and Engineering

SUMMARY

In line with the spirit of “Reimagine Tomorrow”, the theme for this year’s Winter Simulation Conference (WSC), we are thrilled to introduce our Case Study Competition which will examine the role of simulation in next-generation industrial systems as well as plant the seeds of collaboration between academia and industry.

Titled “Smart Simulation for Intelligence Incubation”, the competition aims at demonstrating and promoting simulation’s ability to cooperate with optimization rules and data learning in order to improve the overall performance of intelligent systems. It will provide participants with an opportunity to explore and exploit the use of simulation tools in supporting real-time decision analysis under different scenarios.

Using the automated grid mover system as the case study for this competition, we have built a basic simulation system model with discrete-event modelling methods and implemented it in O2DESpy, which is a framework for object-oriented discrete event simulation based on standard Python 3.x. Participants are expected to embed their own algorithms in certain parts of the model, using their skills in model training and large-scale search, to improve the grid mover system performance. The teams that generate the best overall performance stand the best chance to win.

The purpose of this document is to provide participants with detailed descriptions on the model structure, and to guide them on the process of file downloading and submission procedures.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Overview

1.1 Problem Description

Welcome to the challenge. In this journey you are expected to make use of your simulation skills, in cooperation with optimization and data learning techniques to improve the performance of a fully automated grid mover system in a warehouse.

Figure 1 - Traditional Warehouse

A warehouse, as shown in Figure 1, stores products for stocking, packing, and shipping preparations. It is a central location that manages both inbound and outbound SKUs. Traditional operations of a warehouse depend heavily on human labor, such as picker, packer and forklift driver. However, with the force of globalization and the growing demand in e-commerce, warehouses must scale up their operations to facilitate the surging logistics demand in order to make the operations more intelligent, accurate and efficient. Therefore, warehouse digitalization would be the major trend in the next decades. Some of the promising warehouse digitalization technologies include real-time data gathering and interconnectivity, Autonomous Guided Vehicles (AGV), smart analytics and machine learning.

Our warehouse, “Avatar”, recently opened its doors to such disruptive technologies, which involves AGVs as the main transport to facilitate the movement of SKUs in between shelves and different work stations, to meet its ever-rising throughput demand and free up human labors. The AGVs are deployed automatically with real-time data and are capable of self-navigation. However, great challenges arise in ensuring traffic efficiencies due to the large volume of SKUs and limited AGV and warehouse space resources, as well as the potential for traffic congestion.

To solve this problem, the research team has decided to build a simulation model to test on different AGV intelligent deployment, route planning and collision management algorithms. However, the AGV traffic networks in real warehouses are complex, containing intersecting lanes, road obstacles, pick-up and drop off points, and their designs could vary depending on the warehouse layout and equipment resources (shown in Figure 2).

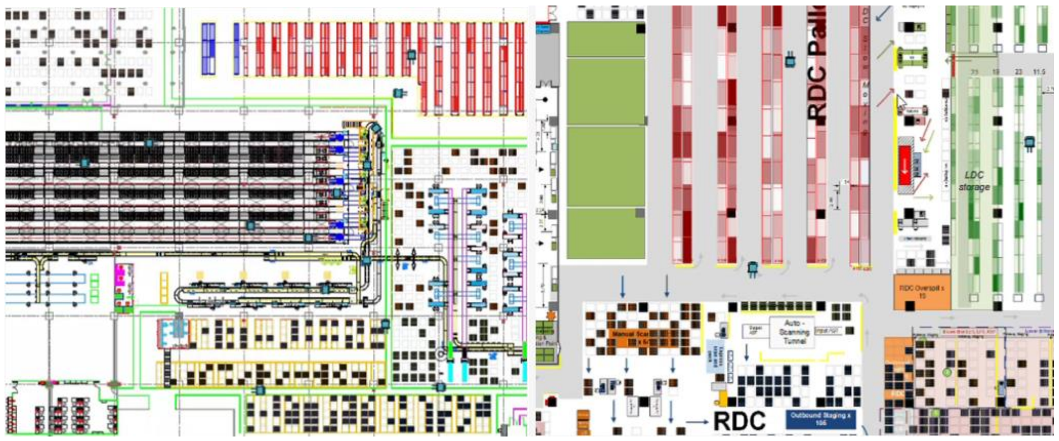


Figure 2 - Various Warehouse Layout and Traffic Designs

To build a model that can represent a generic and robust estimation of AGV traffic network, a grid-based environment is adopted. As shown in Figure 3, grids are used to represent the vehicle traffic network with traversable square units. This mosaic structure allows different traffic layouts to be modeled by blocking certain square units. Building upon this traffic environment, other operations such as job generation, job deployment, vehicle route planning and movement, loading and unloading processes are incorporated into the model, forming the "Grid Mover System". The simulation model is implemented in python using discrete-event methodologies.

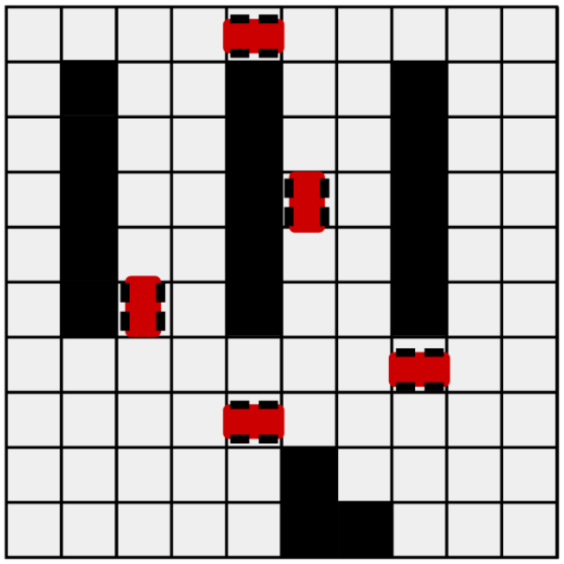


Figure 3 - Grid-based Traffic Environment

First of all, your task in this challenge is to help "Avatar" devise a new vehicle deployment algorithm, route planning algorithm and collision avoidance algorithm (route reservation algorithm). This is then followed by rewriting and replacing the existing decision rules in the given model. Last but not least, you are to use your knowledge in model training and large-scale search to maximize the performance of the Grid Mover System under different simulation scenarios.

Good luck helping Avatar break its new through-put record:)

1.2 Competition Rules

As shown in Figure 4, the given model contains data and source code according to the following three aspects of information:

(A) Input: Example simulation scenarios with specific parameters.

(B) Interface: The three decision modules where users can modify the code for their own decision algorithms for vehicle deployment, route planning and collision management through partial route reservation request. The respective names for the modules are "Deployment", "Routing" and "Request". The default algorithms will be provided too.

(C) Output: Performance indicators to measure the efficiency and quality of the Grid Mover System

Part (A) (B) (C) will be further elaborated in the "Data" section 3.5, 3.6 and 3.7 respectively. Apart from these three aspects, the discrete-event simulation model will also be provided for user's reference.

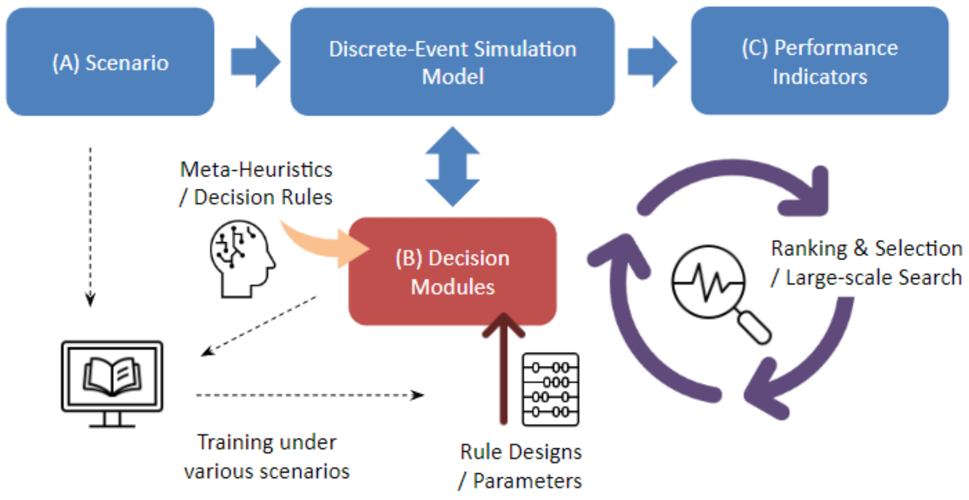

(A) Scenario
Discrete-Event Simulation Model
(C) Performance Indicators
Meta-Heuristics / Decision Rules
(B) Decision Modules
Ranking & Selection / Large-scale Search
Training under various scenarios
Rule Designs / Parameters

Figure 4 - Competition Format

You are expected to rewrite and replace the existing decision modules (B) with Python language, to maximize the performance (C) of the Grid Mover System under different scenarios (A) through model training and large-scale search.

You can generate the logic rules in Part (B) in various ways, including but not limited to:

a. Writing rule-based scripts or heuristic algorithms embedded in decision events
b. Embedding simulation model into external optimization search algorithm
c. Using the machine learning model to identify and conclude the best rule parameters and embed them in decision events

You only need to provide part (B) of the program code and required data, and there is no need to submit optimization and training program.

Note: all other source code besides those for Part (B) will not be evaluated or runed.

1.3 General Evaluation Guide

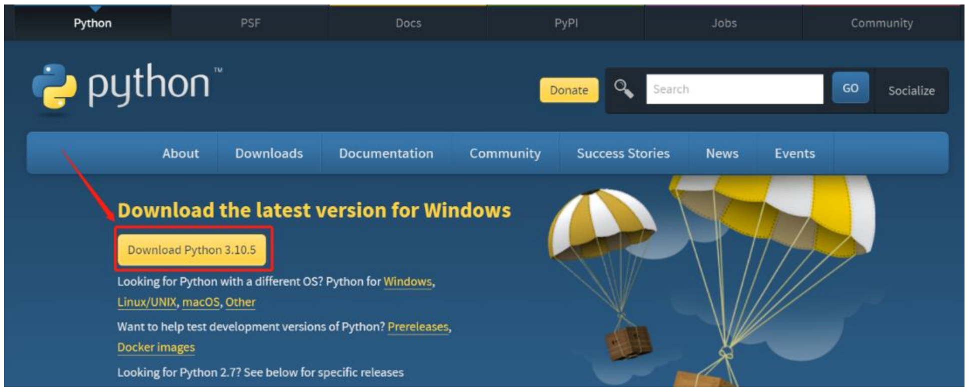
Your data and program code will be embedded in the discrete-event simulation model provided in advance, overwrite the corresponding original code, and compile and generate an executable simulation program. Your program will run under a variety of scenarios and random seeds. The winner will be the one whose model generates top average performance index for each case.

For further instructions regarding file downloads, submissions please see Chapter 2 and Chapter 4. For elaborations on model structures and available data, please see Chapter 3. For detailed evaluation criteria, please see Chapter 4.

User Instruction

2.1 Install Python

1) Go to website https://www.python.org/downloads/ and click **Download** button to download the python executable installer

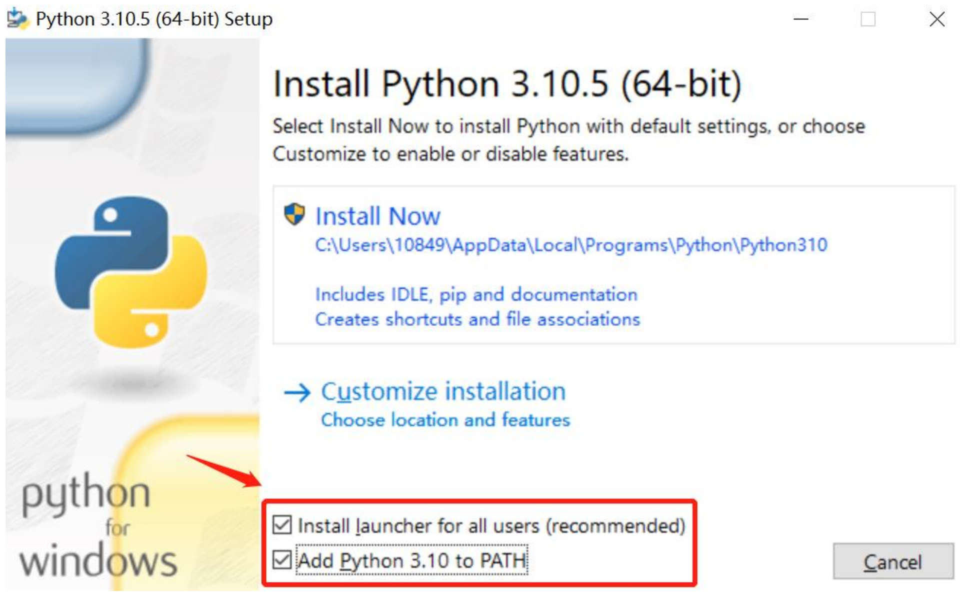


Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python? Prereleases, Docker images

Looking for Python 2.7? See below for specific releases

2) Double click the downloaded .exe file and start the installation process
3) Tick both the **Install launcher for all users** and **Add Python 3.10 to PATH** checkboxes



Install Python 3.10.5 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

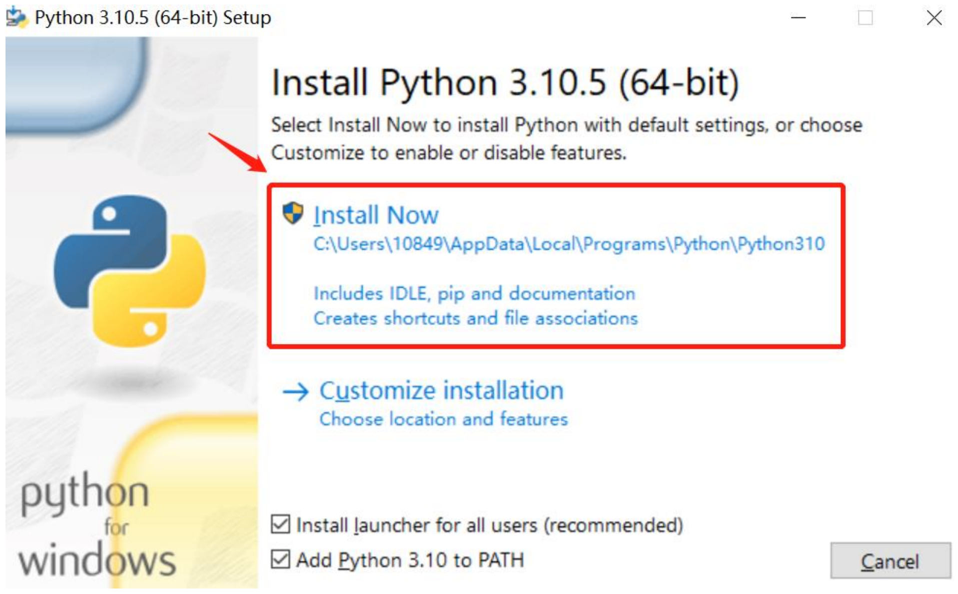
Install Now
C:\Users\10849\AppData\Local\Programs\Python\Python310

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.10 to PATH

4) Click **Install Now**


Python 3.10.5 (64-bit) Setup

Install Python 3.10.5 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

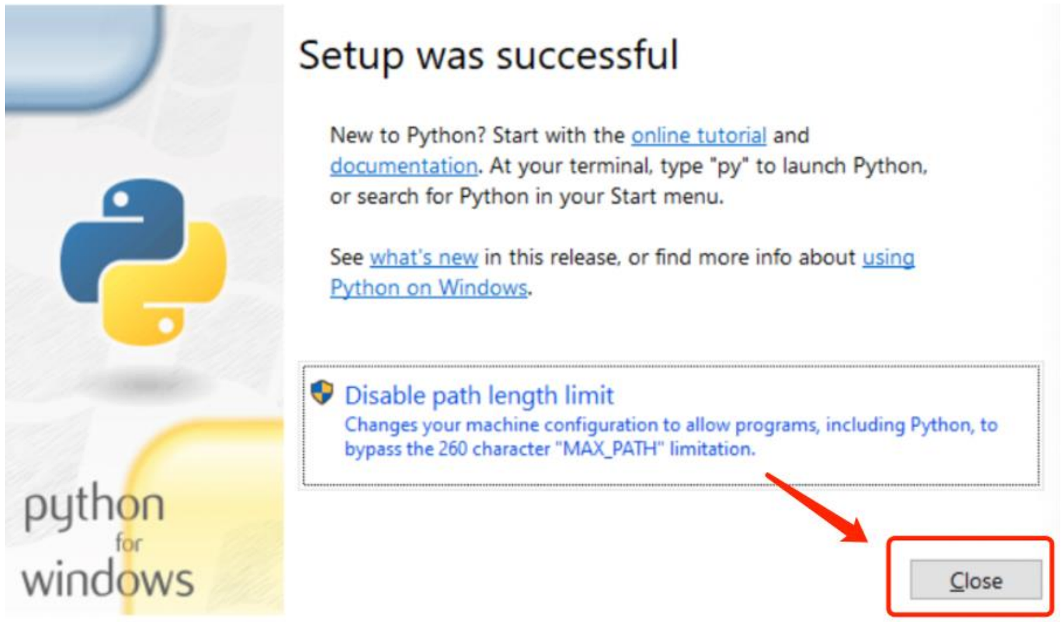
Install Now
C:\Users\10849\AppData\Local\Programs\Python\Python310

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.10 to PATH

5) Wait for a while to complete the installation process and after successful setup, close the dialogue box


Setup was successful

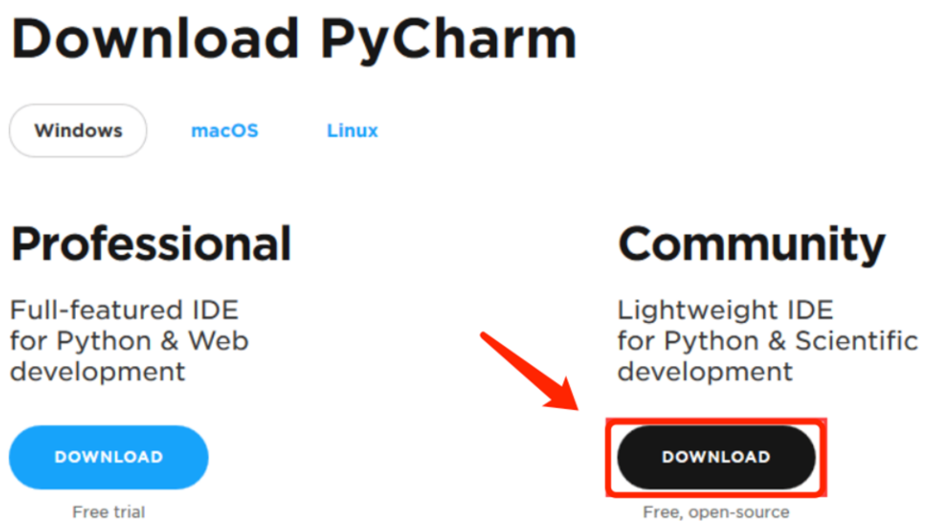
New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

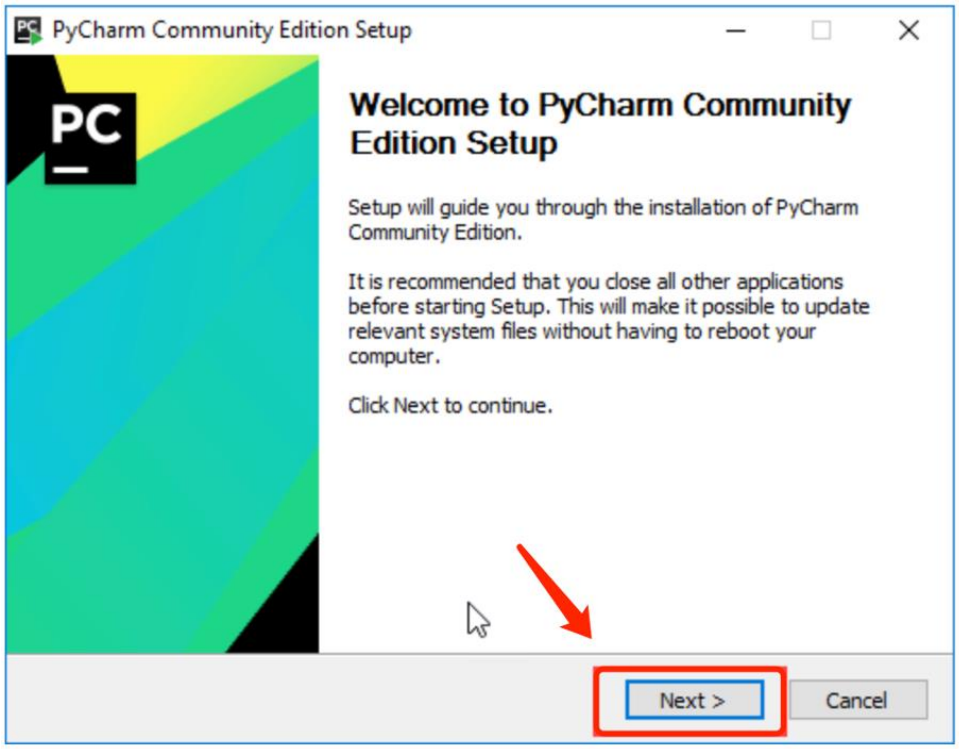
Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

2.2 Install PyCharm

1) Go to the website https://www.jetbrains.com/pycharm/download/ and click the **DOWNLOAD** button under the Community Section.


Download PyCharm
Windows
macOS
Linux
Professional
Full-featured IDE for Python & Web development
DOWNLOAD
Free trial
Community
Lightweight IDE for Python & Scientific development
DOWNLOAD
Free, open-source

2) Double click the downloaded .exe file to start the installation process and click **Next**



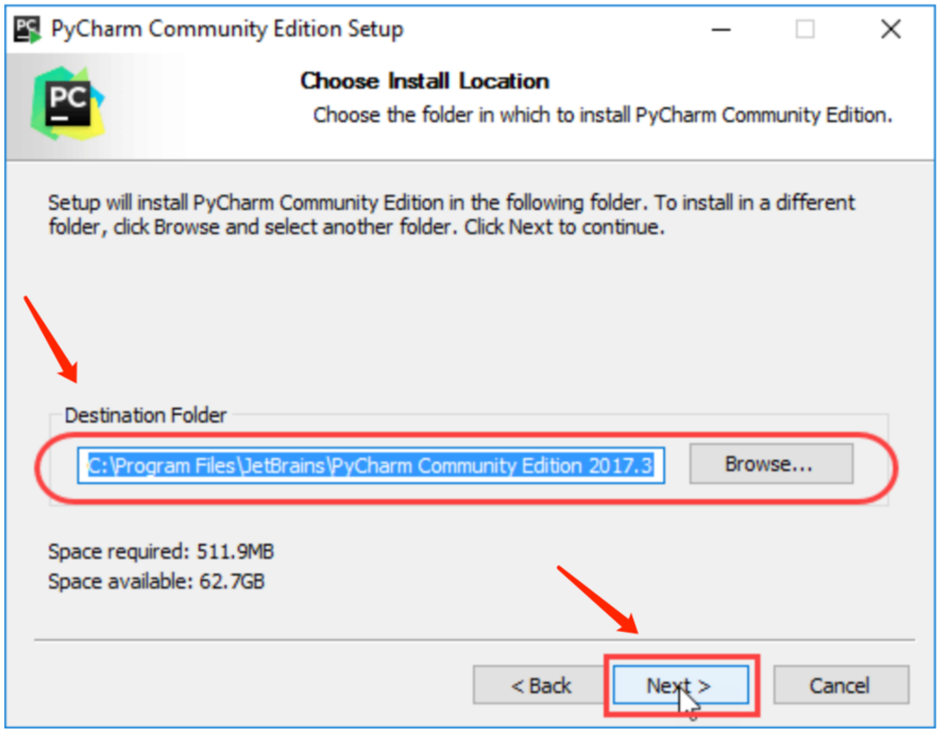
Welcome to PyCharm Community Edition Setup

Setup will guide you through the installation of PyCharm Community Edition.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

3) Modify the installation location if needed and click **Next**


PyCharm Community Edition Setup

Choose Install Location

Choose the folder in which to install PyCharm Community Edition.

Setup will install PyCharm Community Edition in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

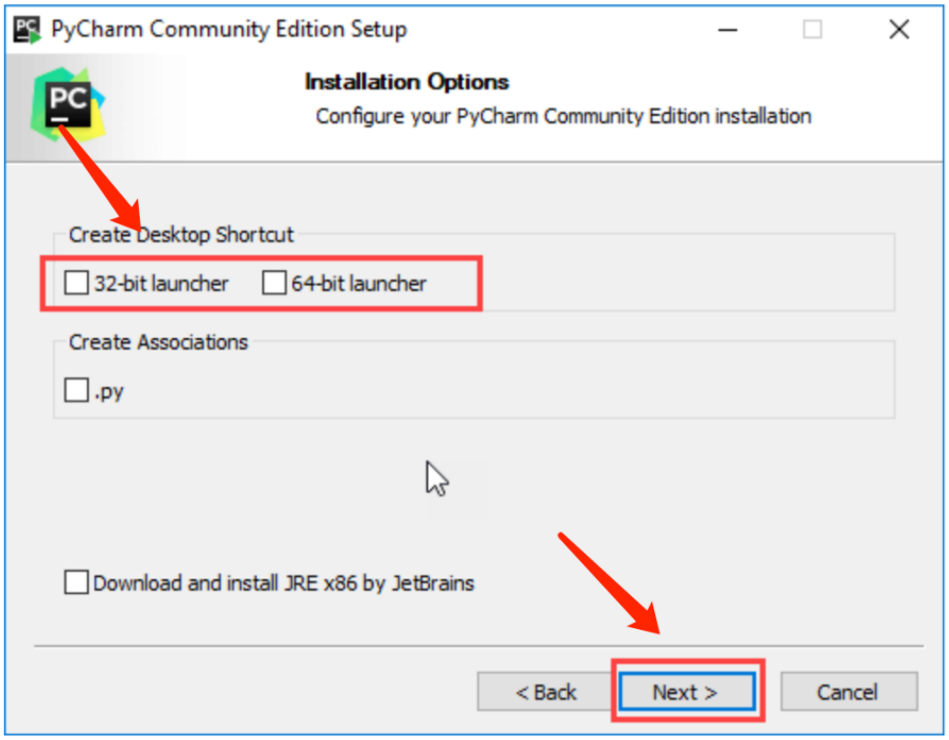
Destination Folder

C:\Program Files\JetBrains\PyCharm Community Edition 2017.3

Space required: 511.9MB

Space available: 62.7GB

4) Choose 64-bit launcher and click **Next**


PyCharm Community Edition Setup

Installation Options

Configure your PyCharm Community Edition installation

Create Desktop Shortcut

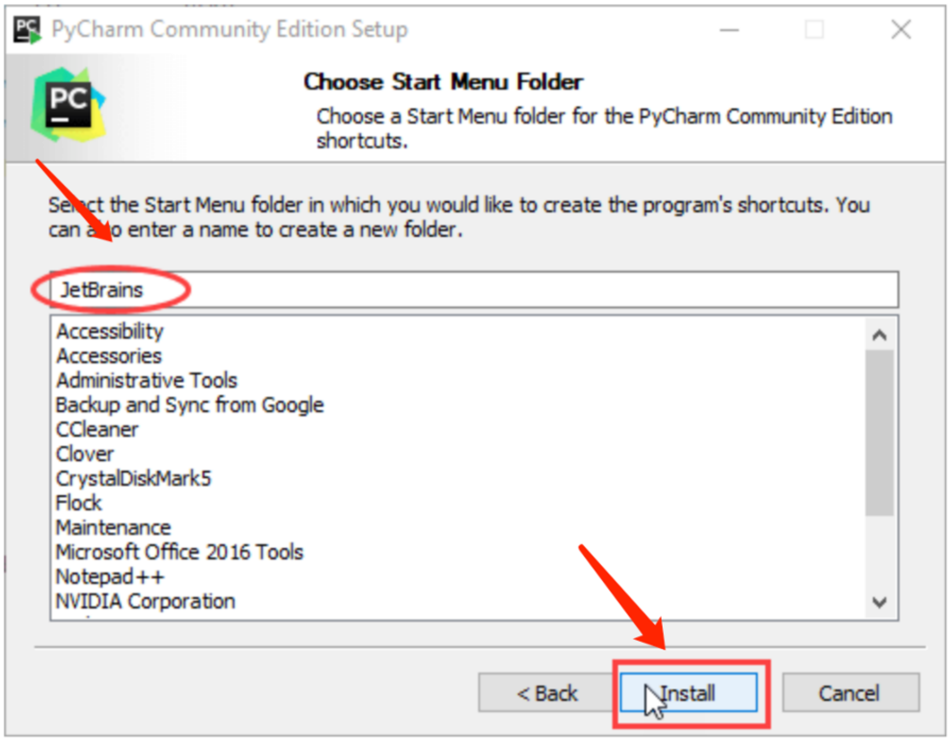
☐ 32-bit launcher ☐ 64-bit launcher

Create Associations

☐ .py

☐ Download and install JRE x86 by JetBrains

5) Modify the default start menu folder if needed and click **Install**


PyCharm Community Edition Setup

Choose Start Menu Folder

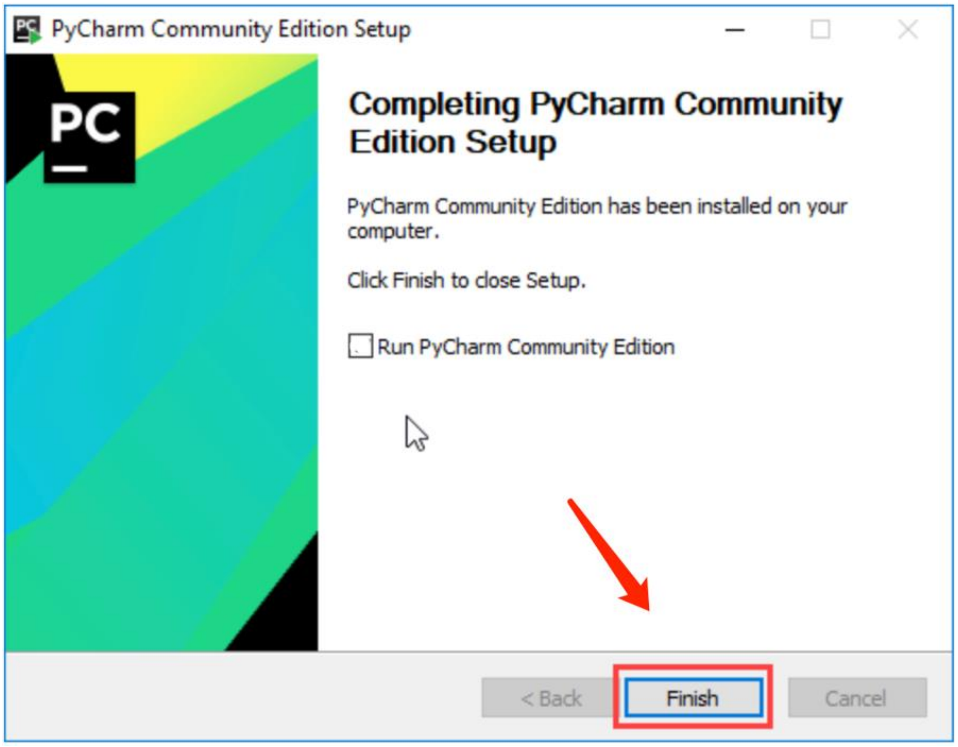
Choose a Start Menu folder for the PyCharm Community Edition shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

JetBrains

Accessibility
Accessories
Administrative Tools
Backup and Sync from Google
CCleaner
Clover
CrystalDiskMark5
Flock
Maintenance
Microsoft Office 2016 Tools
Notepad++
NVIDIA Corporation

6) Wait for the installation process to finish and click **Finish**


PyCharm Community Edition Setup

Completing PyCharm Community Edition Setup

PyCharm Community Edition has been installed on your computer.

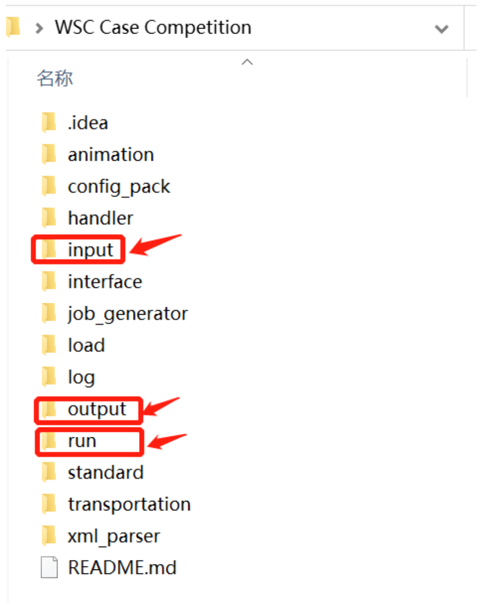
Click Finish to close Setup.

☐ Run PyCharm Community Edition

2.3 Download Source Code

1) After registration and joining in a team, please go to the website https://competition.huaweicloud.com/information/1000041743/introduction and download zip package of source code: "**WSC Case Competition.zip**"
2) Decompose zip package to folder "**WSC Case Competition**"

3）Source code structure of '**WSC Case Competition**'

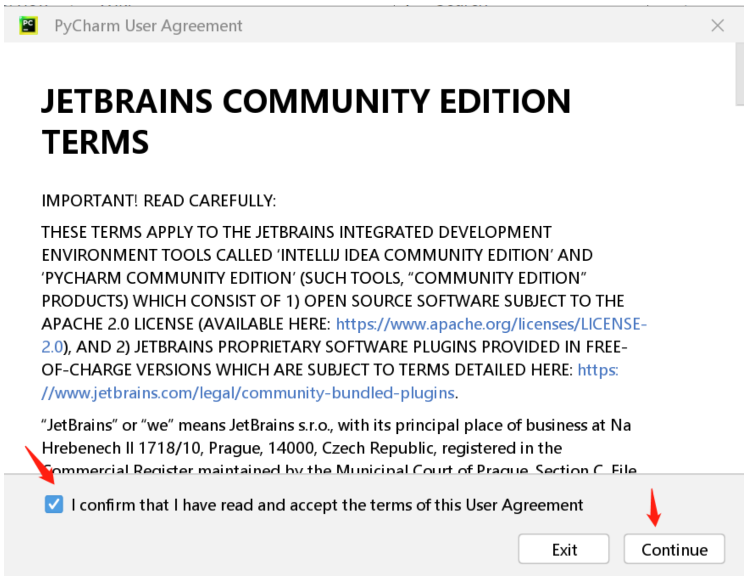


```
WSC Case Competition
名称
.idea
animation
config_pack
handler
input
interface
job_generator
load
log
output
run
standard
transportation
xml_parser
README.md
```

Input folder: includes scenario files (XML files)
Output folder: includes output file (JSON file)
Run folder: includes run files (Python files)

2.4 Add Python Interpreter

1) Open PyCharm, confirm user agreement and click **Continue**



JETBRAINS COMMUNITY EDITION TERMS

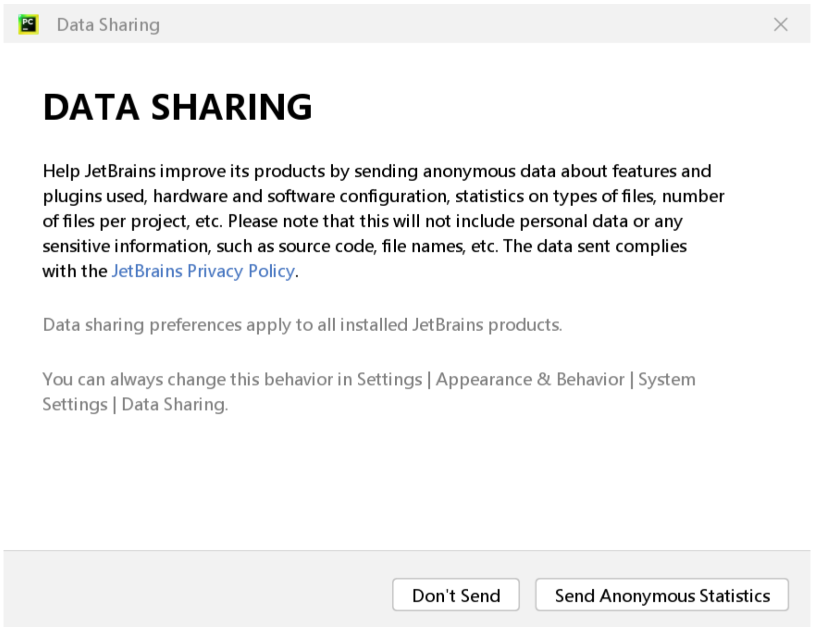
IMPORTANT! READ CAREFULLY:

THESE TERMS APPLY TO THE JETBRAINS INTEGRATED DEVELOPMENT ENVIRONMENT TOOLS CALLED ‘INTELLIJ IDEA COMMUNITY EDITION’ AND ‘PYCHARM COMMUNITY EDITION’ (SUCH TOOLS, “COMMUNITY EDITION” PRODUCTS) WHICH CONSIST OF 1) OPEN SOURCE SOFTWARE SUBJECT TO THE APACHE 2.0 LICENSE (AVAILABLE HERE: https://www.apache.org/licenses/LICENSE-2.0), AND 2) JETBRAINS PROPRIETARY SOFTWARE PLUGINS PROVIDED IN FREE-OF-CHARGE VERSIONS WHICH ARE SUBJECT TO TERMS DETAILED HERE: https://www.jetbrains.com/legal/community-bundled-plugins.

“JetBrains” or “we” means JetBrains s.r.o., with its principal place of business at Na Hrebenech II 1718/10, Prague, 14000, Czech Republic, registered in the Commercial Register maintained by the Municipal Court of Prague, Section C, File

[✓] I confirm that I have read and accept the terms of this User Agreement

2) Click either option of your choice



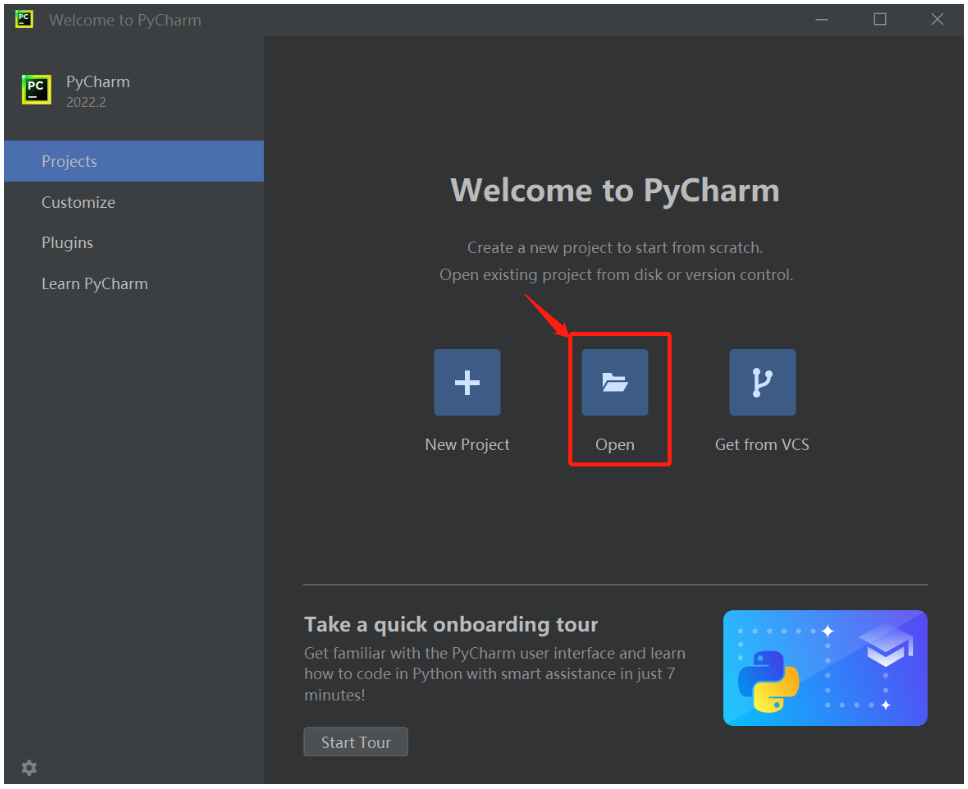
DATA SHARING

Help JetBrains improve its products by sending anonymous data about features and plugins used, hardware and software configuration, statistics on types of files, number of files per project, etc. Please note that this will not include personal data or any sensitive information, such as source code, file names, etc. The data sent complies with the JetBrains Privacy Policy.

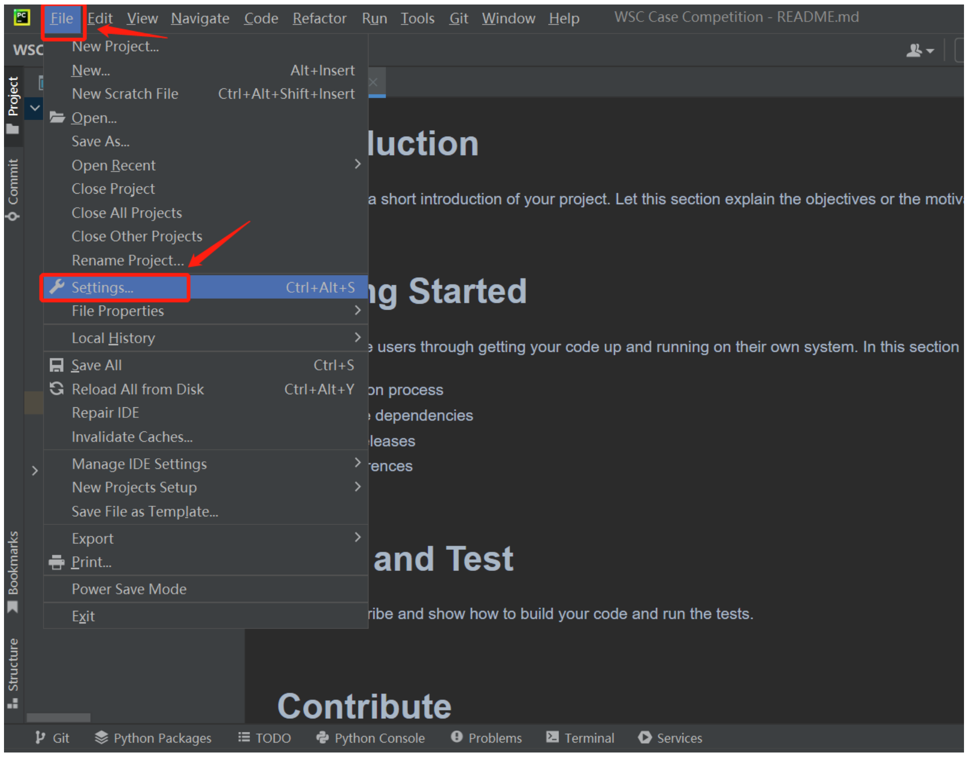
Data sharing preferences apply to all installed JetBrains products.

You can always change this behavior in Settings | Appearance & Behavior | System Settings | Data Sharing.

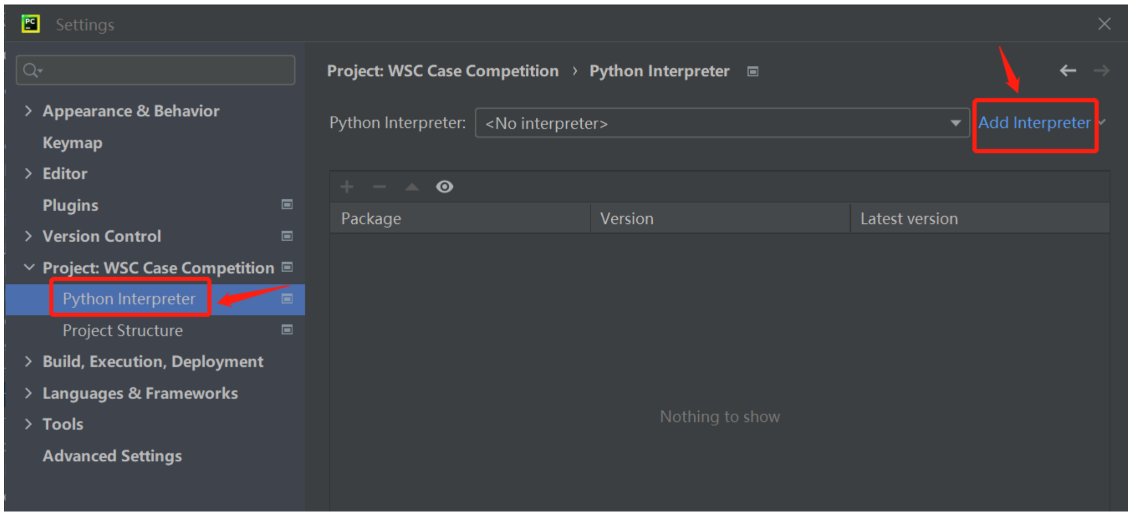
3) Click **Open**, and open “**WSC Case Competition**” folder directly


Welcome to PyCharm
PyCharm
2022.2
Projects
Customize
Plugins
Learn PyCharm
Welcome to PyCharm
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
Take a quick onboarding tour
Get familiar with the PyCharm user interface and learn how to code in Python with smart assistance in just 7 minutes!
Start Tour

4) Click **File** and click **Settings**

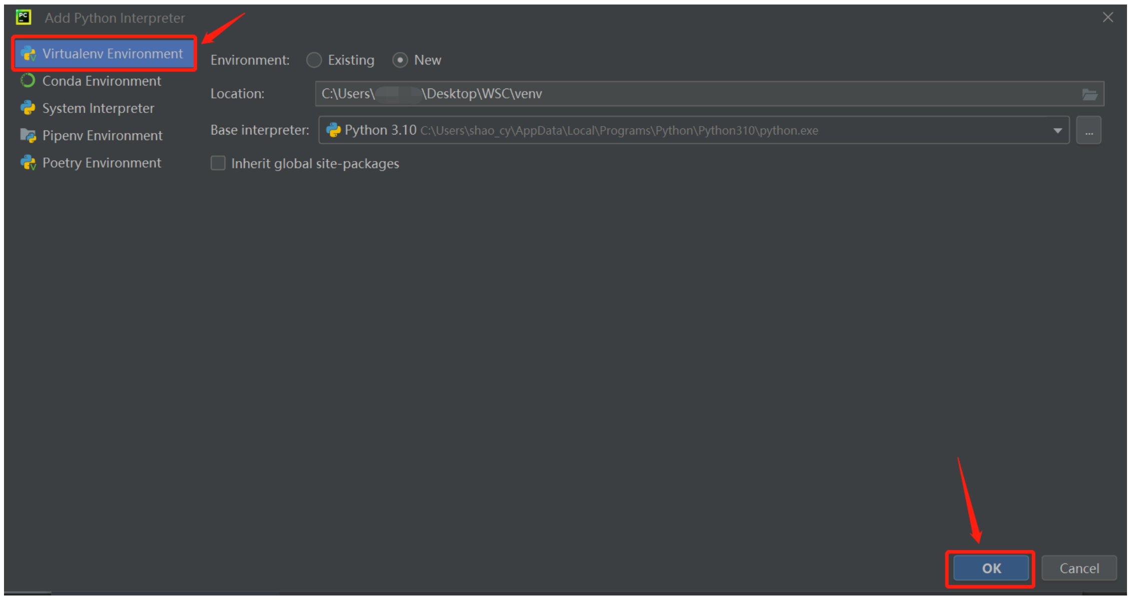


5) Open **Project** tab and click **Python Interpreter**, and click **Add Interpreter**



Python Interpreter: <No interpreter>

6) Choose **Virtualenv Environment**, click **OK**



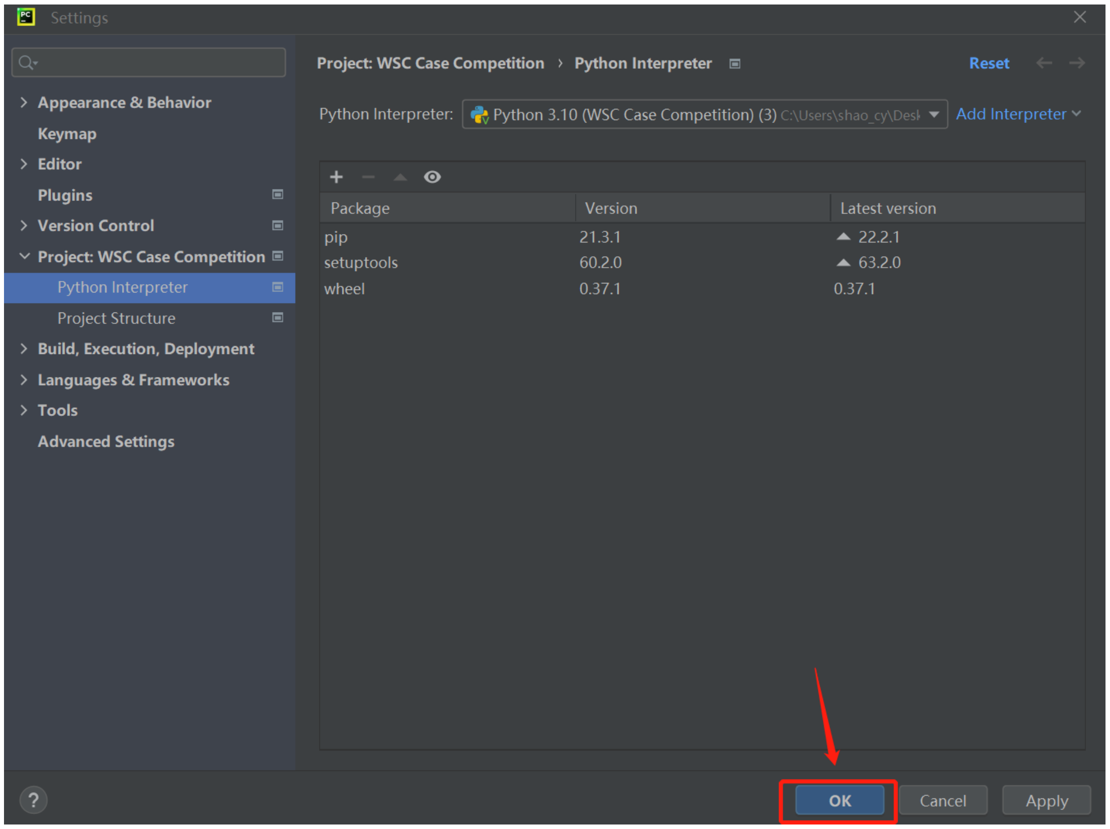
Environment: Existing [✓ New]

Location: C:\Users\[redacted]\Desktop\WSC\venv

Base interpreter: Python 3.10 C:\Users\shao_cy\AppData\Local\Programs\Python\Python310\python.exe

☐ Inherit global site-packages

7) Click **OK**

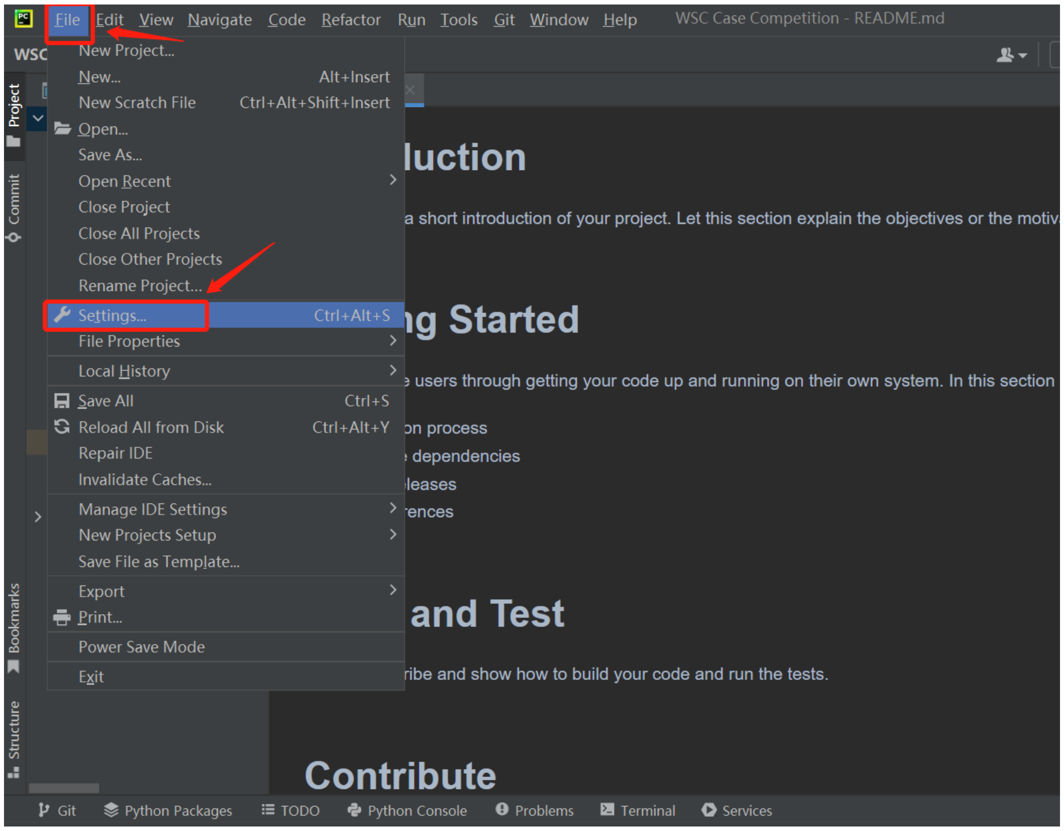


Python Interpreter: Python 3.10 (WSC Case Competition) (3) C:\Users\shao_cy\Desk

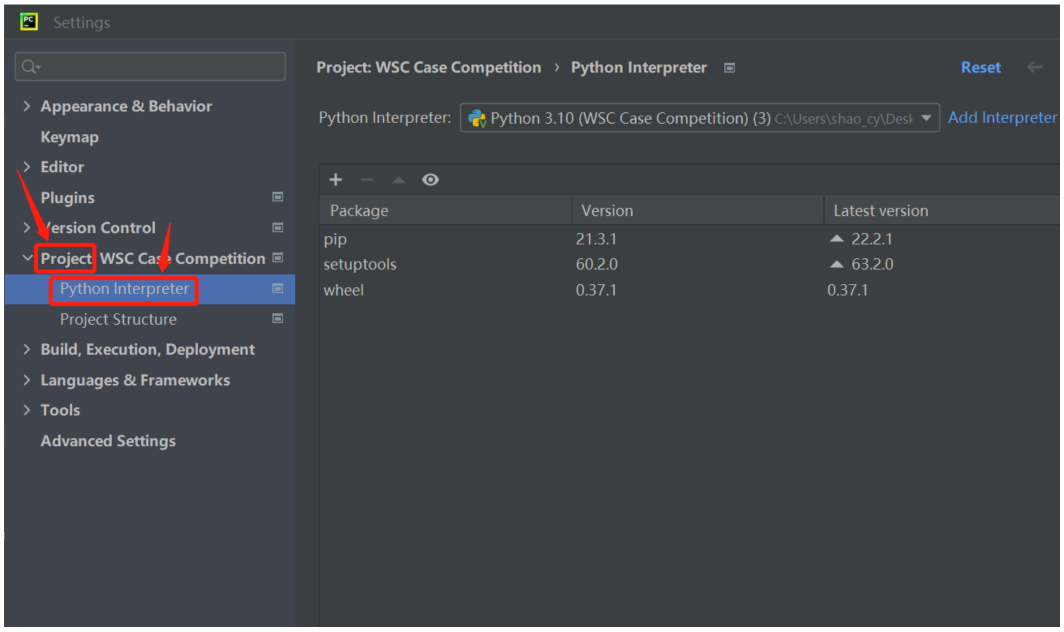
Package	Version	Latest version
pip	21.3.1	▲ 22.2.1
setuptools	60.2.0	▲ 63.2.0
wheel	0.37.1	0.37.1

2.5 Add Python Packages Installation

1) Click **File** and click **Settings**



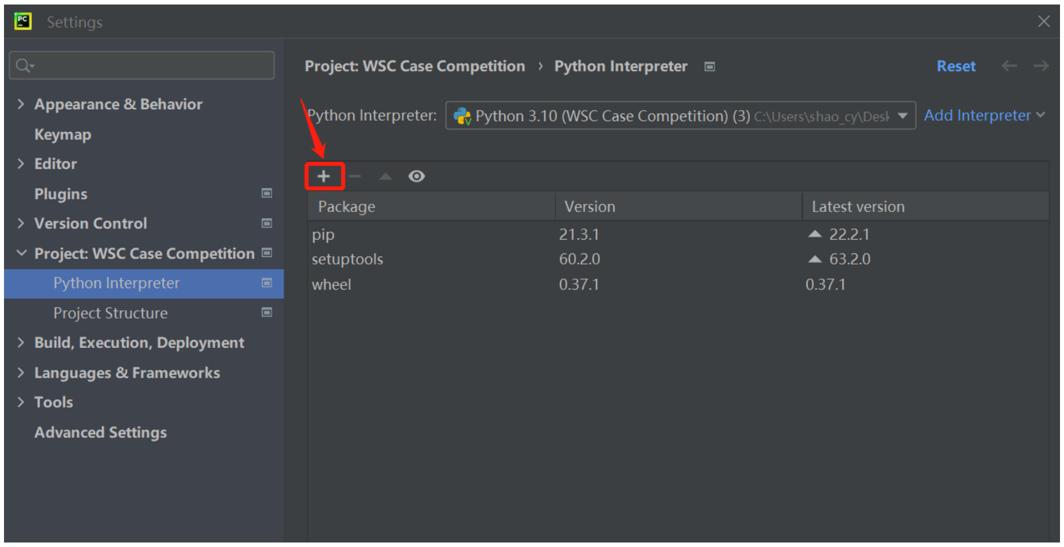
2) Open **Project** tab and click **Python Interpreter**



Python Interpreter: Python 3.10 (WSC Case Competition) (3) C:\Users\shao_cy\Desk

Package	Version	Latest version
pip	21.3.1	▲ 22.2.1
setuptools	60.2.0	▲ 63.2.0
wheel	0.37.1	0.37.1

3) Click + button


Settings

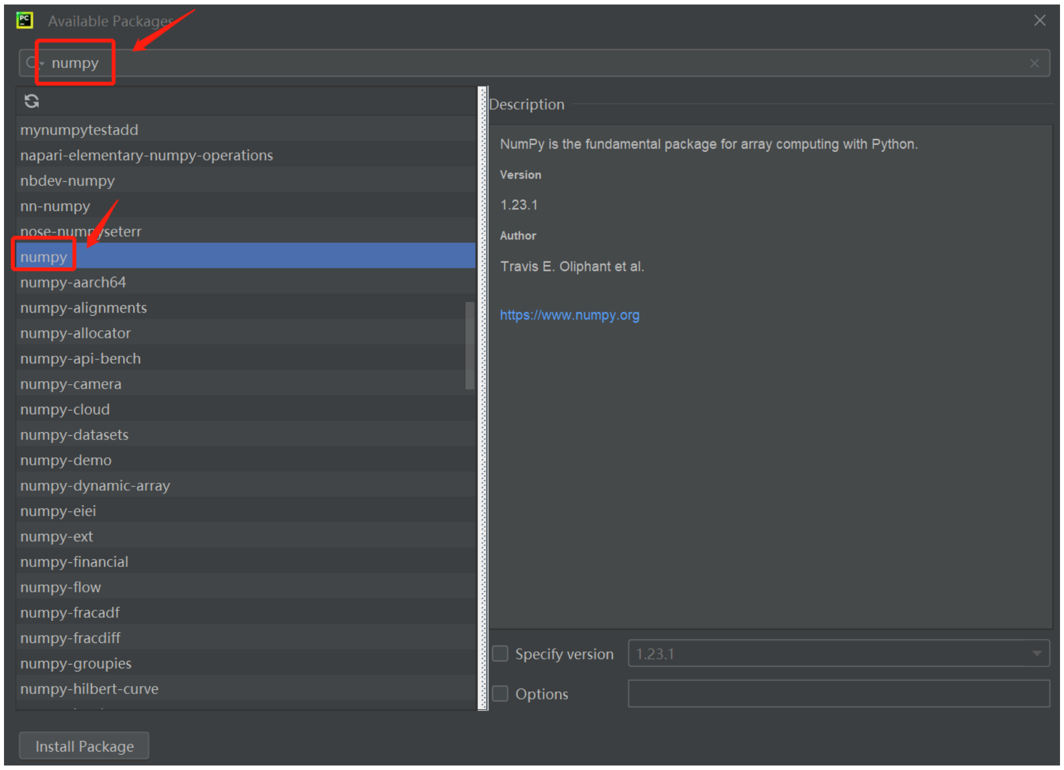
- Appearance & Behavior
- Keymap
- Editor
- Plugins
- Version Control
- Project: WSC Case Competition
  - Python Interpreter
  - Project Structure
- Build, Execution, Deployment
- Languages & Frameworks
- Tools
- Advanced Settings

Project: WSC Case Competition › Python Interpreter

Python Interpreter: Python 3.10 (WSC Case Competition) (3) C:\Users\shao_cy\Desk

Package	Version	Latest version
pip	21.3.1	▲ 22.2.1
setuptools	60.2.0	▲ 63.2.0
wheel	0.37.1	0.37.1

4) Search for the packages to be installed


Available Packages

numpy

- mynumpytestadd
- napari-elementary-numpy-operations
- nbdev-numpy
- nn-numpy
- nose-numpyseterr
- numpy
- numpy-aarch64
- numpy-alignments
- numpy-allocator
- numpy-api-bench
- numpy-camera
- numpy-cloud
- numpy-datasets
- numpy-demo
- numpy-dynamic-array
- numpy-eiei
- numpy-ext
- numpy-financial
- numpy-flow
- numpy-fracadf
- numpy-fracdiff
- numpy-groupies
- numpy-hilbert-curve

Description

NumPy is the fundamental package for array computing with Python.

Version

1.23.1

Author

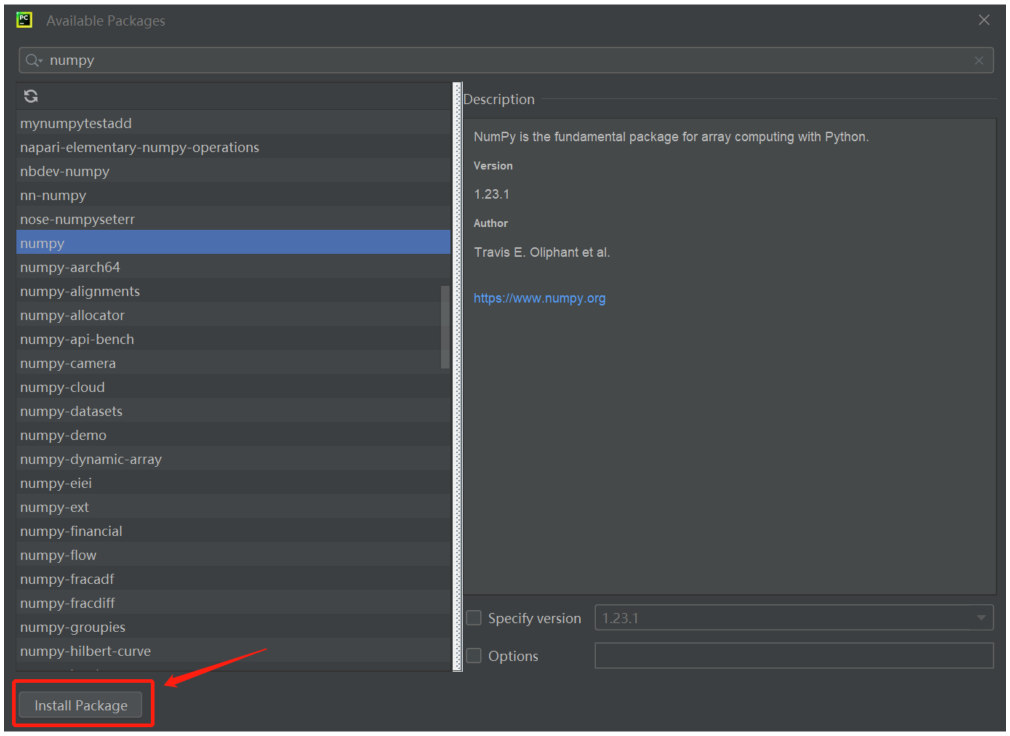
Travis E. Oliphant et al.

https://www.numpy.org

Specify version: 1.23.1

Options

5) Click Install **Package** button to start the installation process



NumPy is the fundamental package for array computing with Python.

Version

1.23.1

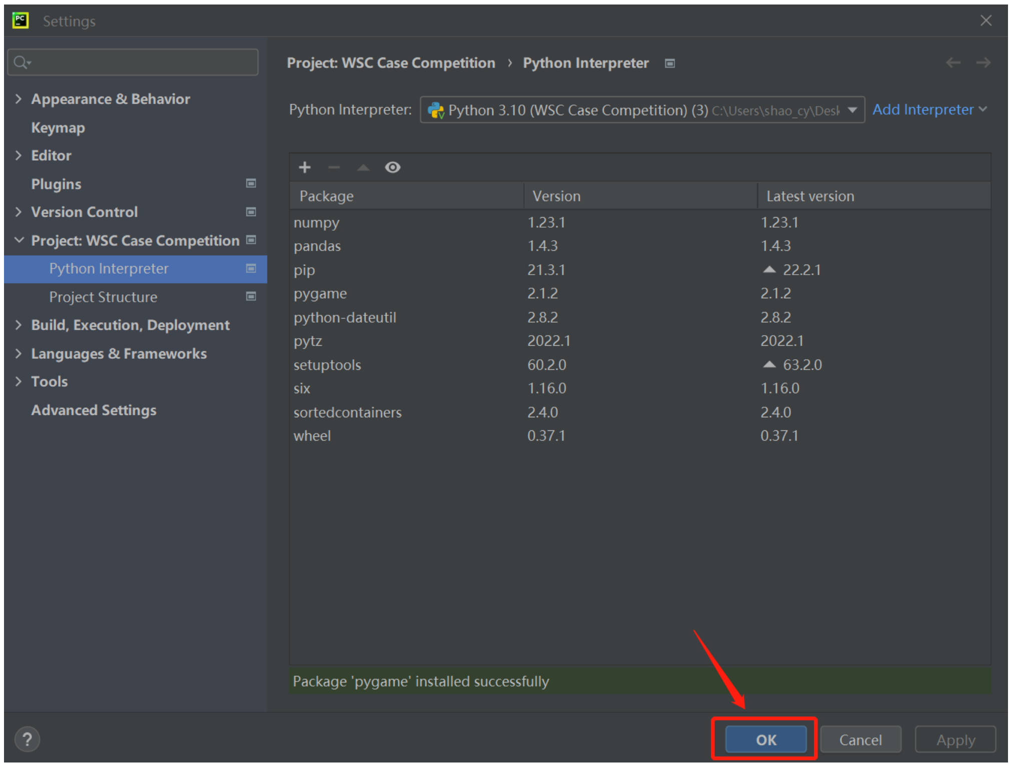
Author

Travis E. Oliphant et al.

https://www.numpy.org

6) For this competition, repeat step 4 - step 5 to install '**numpy**', '**pandas**', '**sortedcontainers**' and '**pygame**' correspondingly

7) Click **OK** to finish all packages installation



Python Interpreter: Python 3.10 (WSC Case Competition) (3) C:\Users\shao_cy\Desk

Package	Version	Latest version
numpy	1.23.1	1.23.1
pandas	1.4.3	1.4.3
pip	21.3.1	▲ 22.2.1
pygame	2.1.2	2.1.2
python-dateutil	2.8.2	2.8.2
pytz	2022.1	2022.1
setuptools	60.2.0	▲ 63.2.0
six	1.16.0	1.16.0
sortedcontainers	2.4.0	2.4.0
wheel	0.37.1	0.37.1

Package 'pygame' installed successfully

Chapter 3

Data in the Discrete-Event Simulation Model

3.1 Entities

There are five entities involved in this model, Square Unit, Transportation Network, Job, Vehicle and Grid Mover System. Each is represented by a class and has its own set of attributes. As shown in Figure 1, the grid map, which is formed by a network of lines, is the Transportation Network. The individual square in the grid is the Square Unit, which represents specific positions in the Transportation Network. A Job is a task to move a Job item in the warehouse from the picking position to the delivery position. A Job item is generated at its picking position and eventually moved to delivery position by a Vehicle. Vehicles move along the adjacent Square Units vertically or horizontally to pick or deliver Job items or park themselves at the park positions if no Job is assigned to them. The system which manages all the above entities is called Grid Mover System. An Entity Relationship Diagram (ERD) is shown in Figure 2 to give an overview of the relationships among the entities and the attributes associated with them.

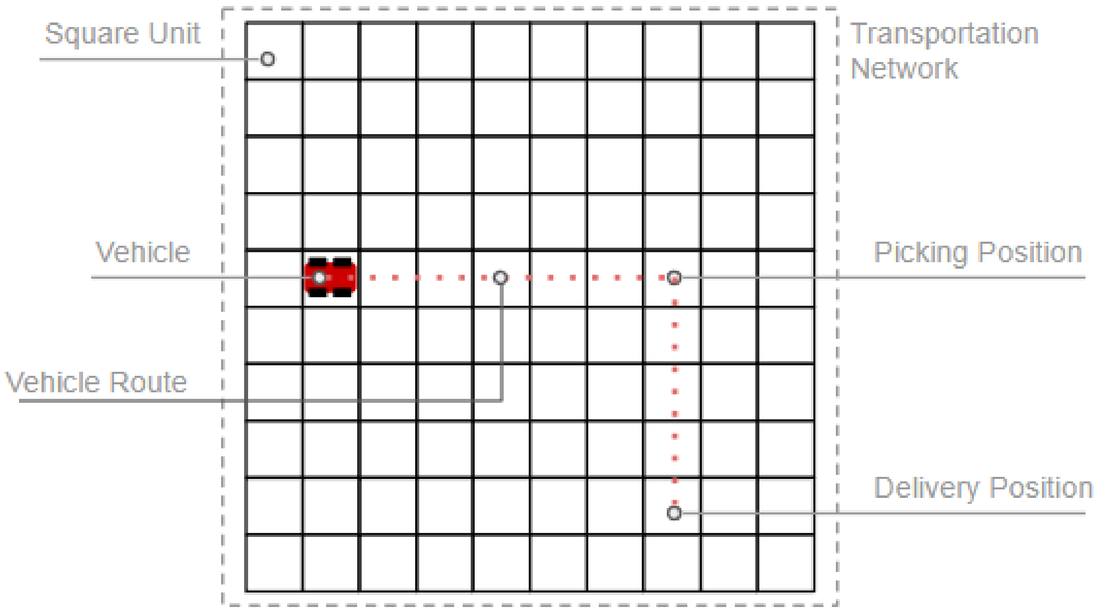
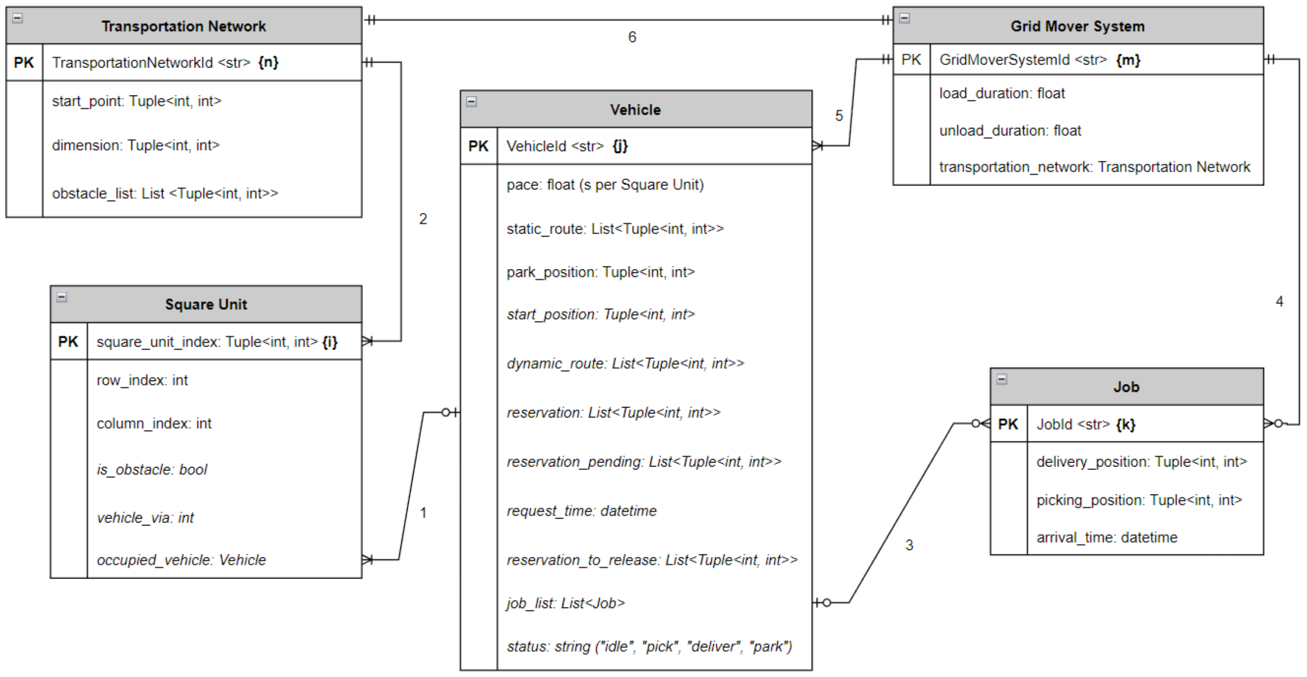


Figure 5 - An Illustration of Entities



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
1
2
3
4
5
6

PK: short term for primary key, which uniquely identifies the entity object. Inside the curly bracket is the alphabetic index of the primary key.

Static attribute: entity attributes that do not change overtime. Static attributes are owned by the class itself.

Dynamic attribute (italic): entity attributes that change overtime. Dynamic attributes are stored in DataFrames. For DataFrame structure, see section 3.2

Entity Relations:
1. A Square Unit can be occupied by at most one and at least zero Vehicle.
   A Vehicle can reserve multiple Square Units, and occupies at least one Square Unit.
2. A Square Unit belongs to one Transportation Network.
   A Transportation network contains at least one Square Unit.
3. A Vehicle can be assigned with no jobs or multiple job.
   A Job can be assigned to at most one Vehicle .
4. A Job generated belongs to at most one Grid Mover System.
   A Grid Mover System contains zero or multiple jobs.
5. A Vehicle belongs to one Grid Mover System.
   A Grid Mover System contains at least one Vehicle.
6. A Grid Mover System contains one Transportation Network
   A Transportation Network belongs one Grid Mover System

Figure 6 - Entity Relationship Diagram (ERD)

The details about the entity attributes and their definitions are explained below.

Table 1 lists the attributes of Square Units. One thing to note is that some of the Square Units can be obstacles and do not allow Vehicles to pass through. Such obstacles will be specified in the model input (see section 3.5 for model input).

Table 1 - Attributes of Square Unit

Attribute Name	**Type**	**Description**
`row_index`	int	The row number of the Square Unit in the Transportation Network, starting from 0.
`column_index`	int	The column number of the Square Unit in the Transportation Network, starting from 0
`square_unit_index`	Tuple<int, int>	The primary key of a Square Unit. The row and column number of the Square Unit in the Transportation Network. The first value is `row_index`, and the second value is `column_index`, starting from (0,0).
`is_obstacle`	bool	Indicating if the Square Unit is an obstacle. If true, the Square Unit cannot be requested or reserved, i.e. no Vehicle can pass through this Square Unit.
`vehicle_via`	int	The total number of Vehicles that have passed through this Square Unit in the current simulation run.
`occupied_vehicle`	Vehicle	The Vehicle that successfully requests or parks at this Square Unit.

Table 2 summarizes the attributes of Transportation Network which are all static attributes and will be specified in the model input too.

Table 2 - Attributes of Transportation Network

Attribute Name	**Type**	**Description**
`id`	str	The primary key of Transportation Network in the form of "TransportationNetwork#" + index
`start_point`	Tuple<int, int>	The start point of the Transportation Network. The first value is `row_index`, the second value is `column_index`.
`dimension`	Tuple<int, int>	The dimension of the Transportation Network. The first value is the number of rows, the second value is the number of columns.
`obstacle_list`	List<Tuple<int, int>>	The list of `square_unit_index` whose respective Square Unit does not allow Vehicles to pass through.

Table 3 gives information on the attributes of Jobs, which are all static attributes. Jobs arrive at the system following an exponential distribution, and they specify the picking positions and delivery positions of the Job items.

Table 3 - Attributes of Job

Attribute Name	Type	Description
`id`	str	The primary key of a Job instance in the form of "Job#" + index
`delivery_position`	Tuple<int, int>	The `square_unit_index` of the Square Unit where a Vehicle delivers the Job item.
`picking_position`	Tuple<int, int>	The `square_unit_index` of the Square Unit where a Vehicle picks the Job item.
`arrival_time`	datetime	The time when the Job arrives at the Grid Mover System.

Table 4 summarizes the attributes of Vehicles.

Table 4 - Attributes of Vehicle

Attribute Name	Type	Description
`id`	str	The primary key of a Vehicle instance in the form of "Vehicle#" + index
`pace`	float	The time in seconds that the Vehicle requires to pass one Square Unit distance. The Unit is [seconds per Square Unit]
`static_route`	List<Tuple<int, int>>	The list of `square_unit_index` of the Square Units in the full route of the Vehicle from one point to another.
`park_position`	Tuple<int, int>	The `square_unit_index` of the Square Unit where the Vehicle parks.
`start_position`	Tuple<int, int>	The `square_unit_index` of the Square Unit that the Vehicle occupies before traveling through the partial route.
`dynamic_route`	List<Tuple<int, int>>	The list of `square_unit_index` of the Square Units in the `static_route` that the Vehicle has yet to reserve and travel.
`reservation`	List<Tuple<int, int>>	List of `square_unit_index` of the Square Units that the Vehicle requests successfully and has not started to pass through.
`reservation_pending`	List<Tuple<int, int>>	List of `square_unit_index` of the Square Units that the Vehicle cannot reserve successfully because they are reserved or occupied by other Vehicles
`request_time`	datetime	The time at which the Vehicle requests its partial route but did not succeed.

`Partial_route_complete_time`	datetime	The time at which the Vehicle completes its current partial route.
`reservation_to_release`	List<Tuple<int, int>>	List of `square_unit_index` of the Square Units that the Vehicle reserves successfully and already starts to pass through.
`job_list`	List<Job>	The list of Job objects assigned to the Vehicle.
`status`	str	A Vehicle has four statuses: idle, pick, deliver, park pick: When the Vehicle is traveling to the Job's picking position deliver: When the Vehicle is traveling to the Job's delivery position park: When the Vehicle is traveling to the park position. idle: When the Vehicle parks at the park position or just finish a job.

3.2 DataFrame

An entity's dynamic attributes are stored in DataFrames. There are three DataFrames in the model: `vehicle_df`, `grid_df` and `available_job_df`.

3.2.1 vehicle_df

`vehicle_df` contains dynamic attributes of each Vehicle instance, matched by its `VehicleId`. Table 5 illustrates an example of the data entries in `vehicle_df` in a scenario with 6 Vehicles. The columns are `VehicleId` and the names of the dynamic attributes, and each row value is associated with one Vehicle instance. To see definitions on each column, please see section 3.1 on entity attribute definitions.

Table 5 - Illustration of vehicle_df

Vehicle	VehicleId	Start Position	DynamicRoute	Reservation	Reservation Pending	Request Time	PartialRoute CompleteTime	ReservationTo Release	JobList	Status
Vehicle	Vehicle1	(1,1)	[(1,2), (1,3), (1,4) ...]	[(1,2), (1,3), (1,4)]	NaN	NaN	NaN	NaN	[Job1, Job2]	Pick
Vehicle	Vehicle2	(3,1)	[(3,2), (3,3), (3,4), (2,4), (2,5), (1,5) ...]	NaN	[(2,4), (2,5), (1,5)]	2022-06-06 10:47:21	2022-06-06 10:44:2	[(3,2), (3,3), (3,4)]	[Job4]	Deliver
Vehicle	Vehicle3	(2,2)	[(2,3), (2,4), (2,5)]	NaN	NaN	NaN	2022-06-06 10:57:2	[(2,3), (2,4), (2,5)]	NaN	Park
Vehicle	Vehicle4	(1,1)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Idle
Vehicle	Vehicle5	(6,9)	NaN	NaN	NaN	NaN	NaN	NaN	[Job5]	Idle
Vehicle	Vehicle6	(7,9)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Idle

3.2.2 grid_df

Similarly, `grid_df` contains dynamic attributes of each Square Unit instance, matched by its \`SquareUnitIndex\`. Table 6 illustrates an example of the data entries in `grid_df`. The columns are \`SquareUnitIndex\` and the names of the dynamic attributes, and each row value is associated with one Square Unit instance.

Table 6 - Illustration of grid_df

SquareUnitIndex	IsObstacle	VehicleVia	OccupiedVehicle
(0,0)	FALSE	3	Vehicle4
(0,1)	TRUE	0	NaN
(0,2)	FALSE	1	NaN
…	…	…	…

3.2.3 available_job_df

However, `available_job_df` is not a DataFrame for dynamic properties. Rather, it is a real time collection on Job instances together with their relevant attributes, including \`JobId\` and \`ArrivalTime\`, for the purpose of job-vehicle deployment. These Jobs not only include those that have arrived at the system and never been assigned to Vehicles, but also those that have already been assigned to Vehicles, but the Vehicles have not started their ways to pick the Jobs. This is to allow the latter type of Jobs to be reassigned to other Vehicles which may be of better choices at a different point in time. The Deployment decision interface module would be explained later in section 3.6.1. Table 7 illustrates an example of the data entries in `available_job_df`. The columns are \`JobId\` and \`ArrivalTime\`, and each row value is associated with one Job instance.

Table 7 - Illustration of available_job_df

Job	JobId	ArrivalTime
Job	Job#1	23/05/2022 10:47:21
Job	Job#2	23/05/2022 10:57:22

3.3 Activity-Based Description

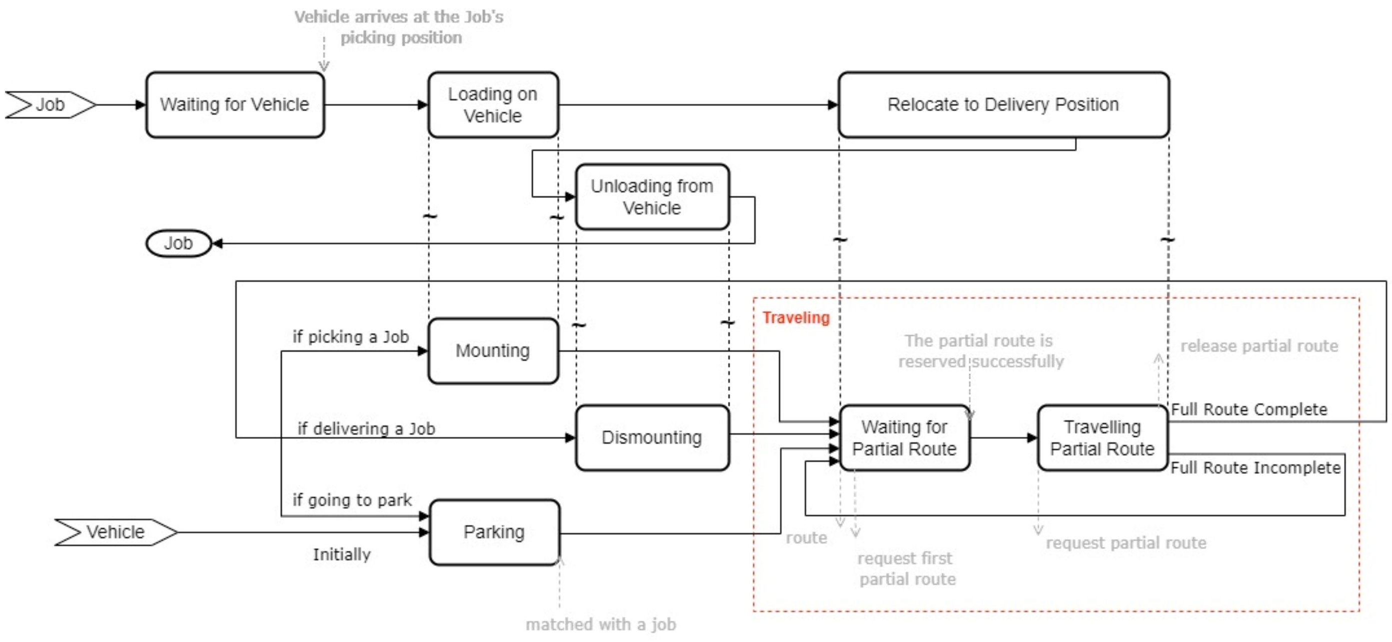


Figure 7 - Entity Flow Diagram (Activity-Based)

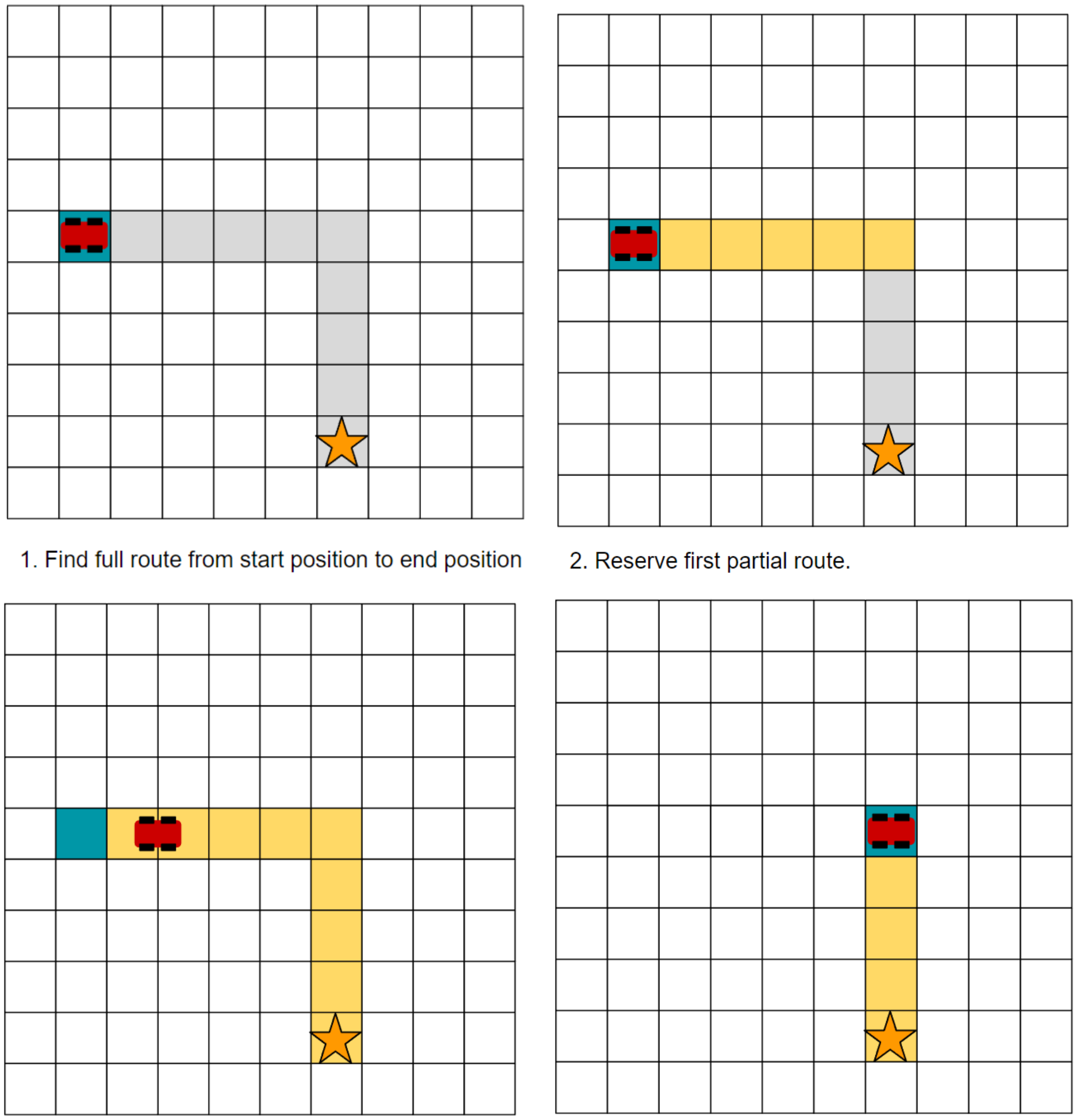
This section aims to provide an overview of the system storyline by describing the activities of different entities. Each box is an activity associated with an entity, and each has an active or passive time delay. Nonetheless, the simulation model is event-based, and Figure 7 only serves as illustration support. Please see section 3.4 for the full event graph.

As illustrated in Figure 7, in the simulation, a Job arrives at the system at its picking position and first waits for a Vehicle. After the assigned Vehicle reaches the Job's picking position, the Job item will start loading on the Vehicle and subsequently is transported to its delivery position where the Job item will be unloaded from the Vehicle. From here, the Job is finished.

A typical Vehicle initially stays at its park position when the simulation starts until it is matched with a Job, then it travels to the Job's picking position to pick the Job item. After successfully loading, it travels to the Job's delivery position to deliver the Job item. After successfully unloading, if the Vehicle has another Job to do, it will travel to pick up the next Job item; If the Vehicle has no more Jobs to do, the Vehicle will travel back to the park position waiting for new Job tasks.

The traveling process is similar for a Vehicle to pick or deliver Job items or to park itself, as shown in Figure 8. It first obtains its full route from its current position (start position) to the destination position (end position), such as from its park position to the Job's picking position during a picking task. Next, it requests its first partial route, i.e., a section of the full route adjacent to the start position. If the Square Units in the partial route are all unoccupied, the Vehicle will reserve the partial route successfully. However, if the Vehicle fails to reserve the partial route, i.e., the Square Units in the partial route may be occupied by other Vehicles at the time of requesting, the Vehicle will wait for the partial route to be released at its current position. After the Vehicle reserves the partial route successfully, it starts to travel through the current partial route. Meanwhile, at any moment during its travel, the Vehicle can request its next partial route, i.e., a section of the full route that is adjacent to the current partial route. Eventually, the current partial route is completed and released when the Vehicle reaches its end Square Unit. This process of requesting, waiting for and traveling through the partial route will repeat until the full route is completed, then it starts to load, unload or park based on different conditions.

To be noted, once deadlock happens, vehilces will stop travelling, leading to failure in finishing jobs.

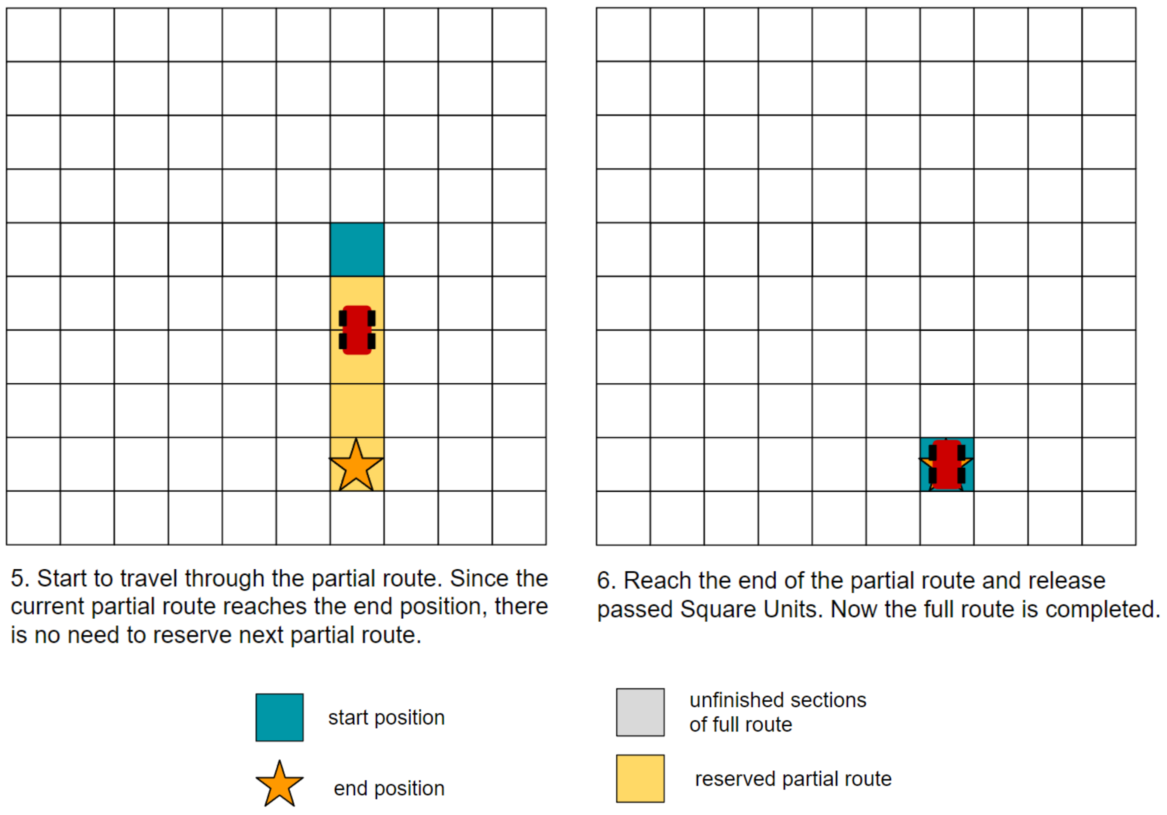


1. Find full route from start position to end position

2. Reserve first partial route.

3. Start to travel through the partial route and reserve the next partial route.

4. Reach the end of the first partial route and release passed Square Units.

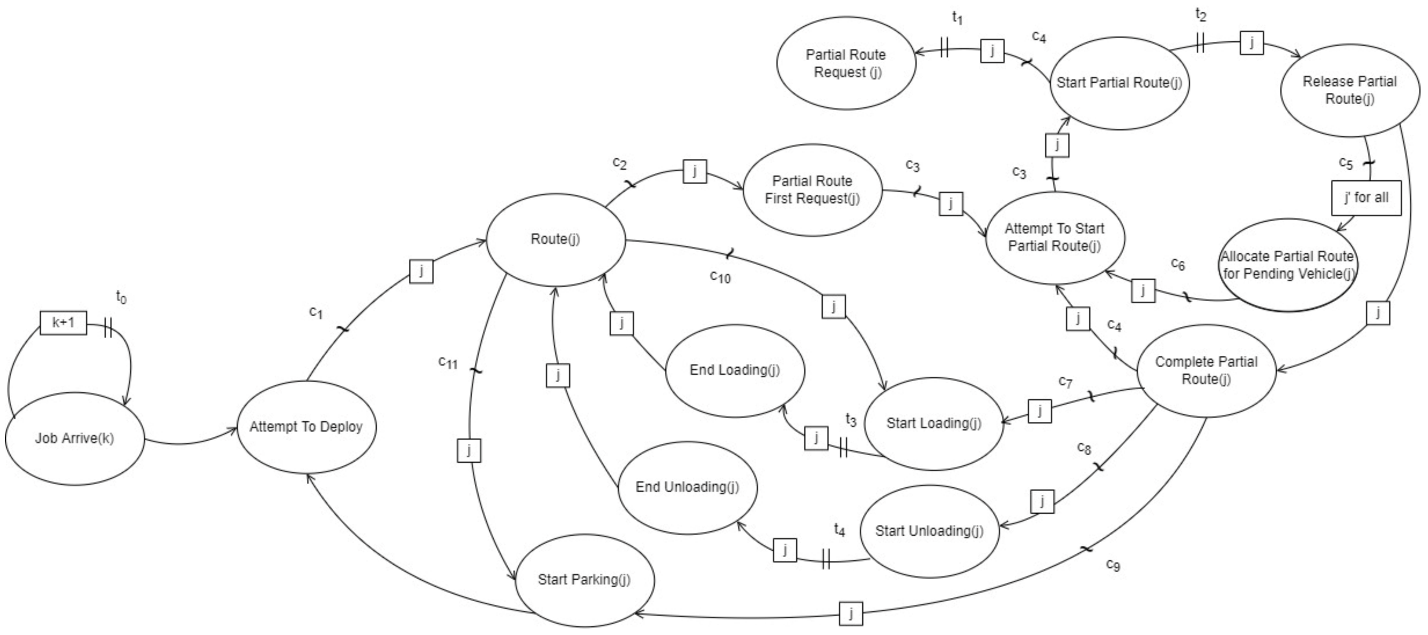

5. Start to travel through the partial route. Since the current partial route reaches the end position, there is no need to reserve next partial route.

6. Reach the end of the partial route and release passed Square Units. Now the full route is completed.

Figure 8 - An Illustration on Traveling Process

3.4 Event

This simulation uses 15 events to describe the above process. As shown in the event graph (EG) in Figure 9. Each event is scheduled with or without a parameter, whose primary key index is shown in the bracket beside the event name. The primary key index can be found in the entity tables in section 3.1. Take "Job Arrive(k)" as an example, `job_arrive()` event is scheduled with a parameter k, a Job object. Similarly, `route()` event is scheduled with a parameter j, a Vehicle object. The arrow " $\longrightarrow$ " indicates the triggering relationship in between events, i.e. an event is scheduled by another event. " c_x ~ " indicates the condition required to schedule the next event. For example, `attempt_to_deploy()` event schedules `route()` event under condition c_1 and passes in parameter j. What's more, " || " represents a time delay in scheduling the next event. For example, `start_partial_route()` event starts at time t, and it schedules `partial_route_request()` event at time t + t_1. In addition, in each event, certain system states or entity attributes might be changed. The following sections will describe each event in detail, and the decision conditions in the events are drawn in flow charts.



c1: if the current Job of a Vehicle is changed or an Idle Vehicle is assigned Jobs

c2: if current position is different from destination and route is found

c3: if partial route request successfully

c4: if remaining route is not empty

c5: if there is vehicle pending for square units

c6: if pending vehicle next partial route is requested successfully and it has finished current partial route

c7: if no remaining route is left and vehicle's status is not park and its current position is picking position

c8: if no remaining route is left and vehicle's status is not park and its current position is delivery position

c9: if no remaining route is left and vehicle's status is park

c10: if vehicle's status is Idle and its current position is picking position

c11: if vehicle's status is Park and its current position is park position

Figure 9 - Event Graph for the Model

3.4.1 job_arrive()

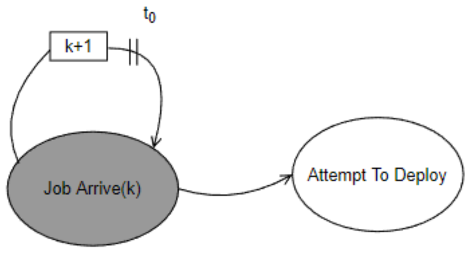


Figure 10 - EG for job_arrive()

In this event, generated Jobs arrive at the Grid Mover System. The Job interarrival time t_0 is specified in the model input, which is further elaborated in section 3.5. Next, the Job object, and its static attributes `id` and `arrival_time` are stored in `available_job_df`. In the end, `attempt_to_deploy()` event is scheduled.

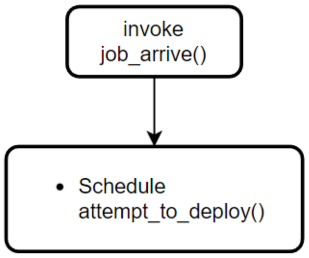
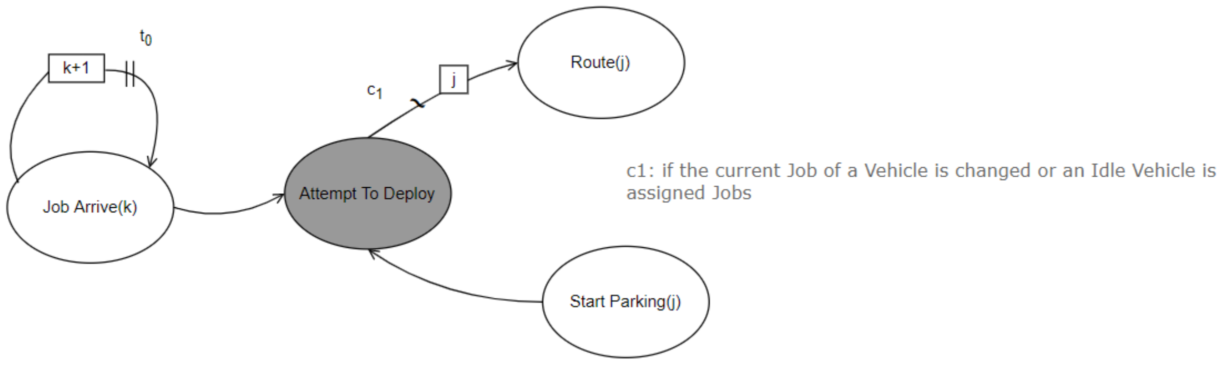


Figure 11 - Flow Chart for job_arrive()

3.4.2 attempt_to_deploy()



t0
k+1
Job Arrive(k)
Attempt To Deploy
c1
j
Route(j)
Start Parking(j)
c1: if the current Job of a Vehicle is changed or an Idle Vehicle is assigned Jobs

Figure 12 - EG for attempt_to_deploy()

In this event, each Job in `available_job_df` is attempted to be assigned to a Vehicle in the Grid Mover System. This event is scheduled by `job_arrive()` event or `start_parking()` event. This is to say, whenever the system receives a new Job or sets free a Vehicle, the system will start to deploy all the available Jobs in the system. If a Job is matched with a Vehicle successfully and the Vehicle is able to pick the Job right away, `route()` event will be scheduled for that Vehicle to plan the route.

The Deployment interface module is called in this event, where users write their own deployment algorithm. The return value from the interface, `all_vehicle_to_jobs_dict`, a dictionary mapping Vehicle object to a list of assigned Job objects, will overwrite the \`JobList\` in `vehicle_df` based on the following algorithm.

- Case 1: When the Vehicle \`Status\` is "Park" in `vehicle_df`:
    - The Vehicle is moving to its park position and has 0 Jobs assigned to it currently (\`JobList\` is NaN), thus the Vehicle can directly take the new job allocation result, i.e. the list of Jobs of this Vehicle in the dictionary will be added to its \`JobList\` in `vehicle_df`;
- Case 2: When the Vehicle \`Status\` is "Idle" in `vehicle_df`:
    - Case 2.1: The Vehicle is parked at its park position with a NaN \`JobList\`, the Vehicle can directly add the new list of Jobs from the dictionary to its \`JobList\`. If the modified \`JobList\` is not NaN, `route()` event will be scheduled;
    - Case 2.2: The Vehicle is waiting to start its first partial route (already requested its first partial route for the current Job but failed). Since the Vehicle has not started moving, the Vehicle will replace its \`JobList\` with the new list of Jobs from the dictionary. After that, if the Vehicle is assigned with a new current Job, the previously requested partial route becomes obsolete, thus the \`DynamicRoute\`, \`ReservationPending\`, \`RequestTime\` will reset to NaN and `route()` event will be scheduled to reroute the Vehicle based on its new current Job.
- Case 3: When the Vehicle \`Status\` is "Pick" or "Deliver" in `vehicle_df`:
    - The Vehicle has an ongoing Job and cannot change its ongoing Job. If the new list of Jobs from the dictionary has the first Job different from the ongoing Job, an error will occur and the simulation will stop. Otherwise, the Vehicle will replace its \`JobList\` with the new list of Jobs from the dictionary.

More information on the Deployment interface can be found in section 3.6.1.

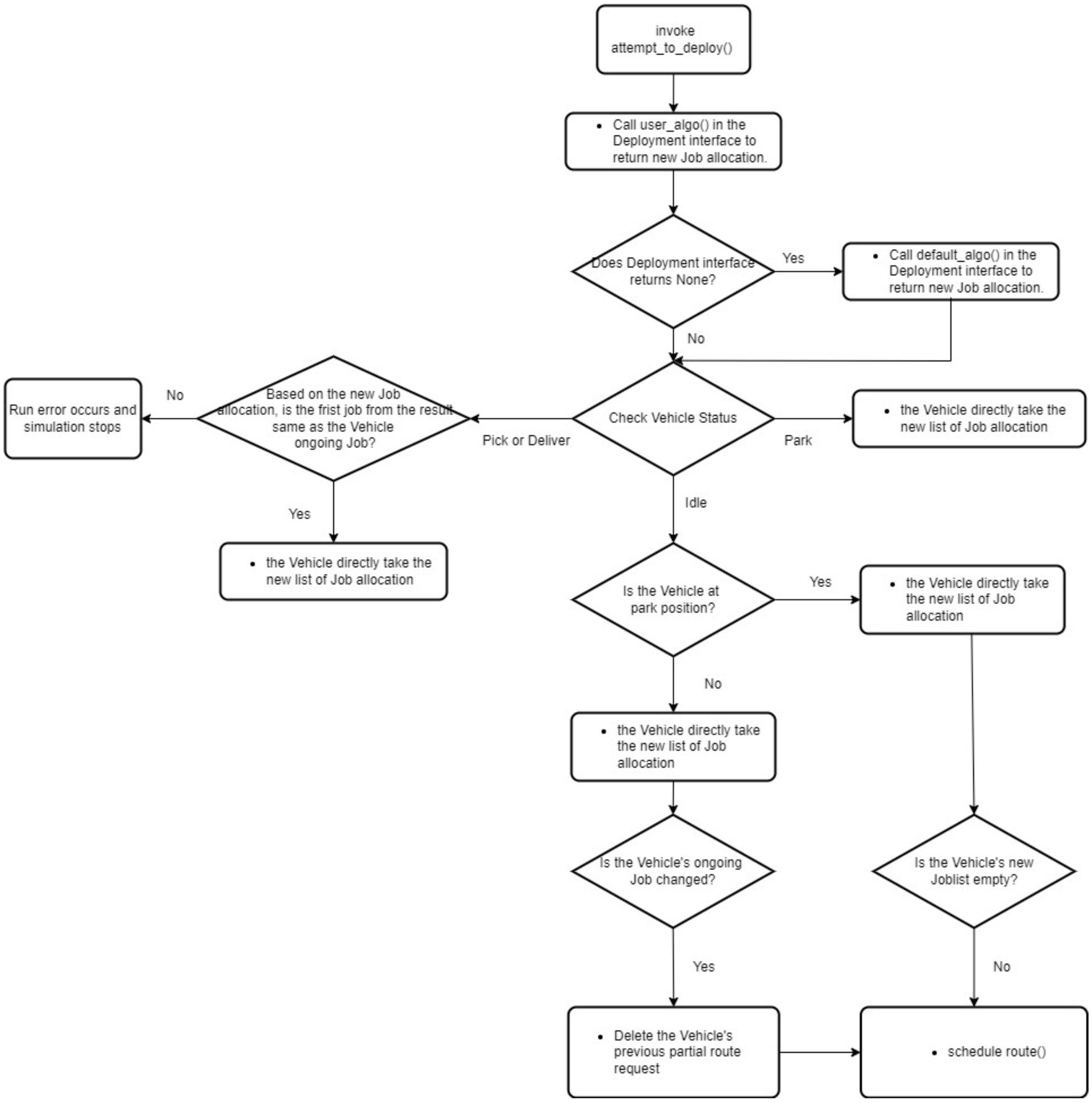
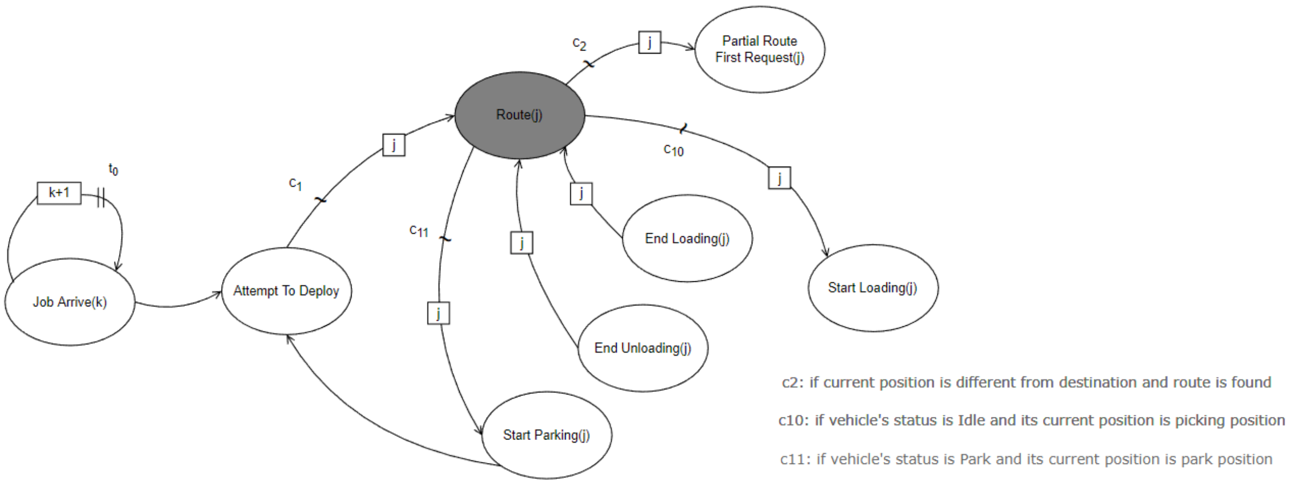

invoke attempt_to_deploy()
Call user_algo() in the Deployment interface to return new Job allocation.
Does Deployment interface returns None?
Yes
Call default_algo() in the Deployment interface to return new Job allocation.
No
Check Vehicle Status
Pick or Deliver
Based on the new Job allocation, is the frist job from the result same as the Vehicle ongoing Job?
No
Run error occurs and simulation stops
Yes
the Vehicle directly take the new list of Job allocation
Park
the Vehicle directly take the new list of Job allocation
Idle
Is the Vehicle at park position?
Yes
the Vehicle directly take the new list of Job allocation
No
the Vehicle directly take the new list of Job allocation
Is the Vehicle's ongoing Job changed?
Yes
Delete the Vehicle's previous partial route request
Is the Vehicle's new Joblist empty?
No
schedule route()

Figure 13 – Flow Chart for attempt_to_deploy()

3.4.3 route()



Partial Route First Request(j)
Route(j)
k+1
t0
c2
c1
c10
c11
j
Job Arrive(k)
Attempt To Deploy
End Loading(j)
Start Loading(j)
End Unloading(j)
Start Parking(j)
c2: if current position is different from destination and route is found
c10: if vehicle's status is Idle and its current position is picking position
c11: if vehicle's status is Park and its current position is park position

Figure 14 - EG for route()

This event allocates a route for the Vehicle from its start position to end position, changes its `Status` accordingly, and schedules `partial_route_first_request()`, `start_loading()`, or `start_parking()` events under different conditions. The event is first scheduled by `attempt_to_deploy()`, `end_unloading()` or `end_loading()` event. After that, the Vehicles passed in as parameters are first categorized by their `Status` to help identify their end positions and schedule the next event based on the following algorithm.

- Case 1: When the Vehicle `Status` is "Idle": The Vehicle could be just ending unloading the previous Job item and heading towards the next assigned Job, or could just be assigned a new Job at its park position, thus it is going to pick the Job.
    - Case 1.1: The Vehicle's current position happens to be its first Job's picking position. It can load the Job item right away, thus `start_loading()` event is scheduled and the Job is deleted from `available_job_df`;
    - Case 1.2: The Vehicle's current position is not its first Job picking position, a route needs to be generated to the picking position, hence Route interface is called and `partial_route_first_request()` event is scheduled.
- Case 2: When the Vehicle `Status` is "Deliver": The Vehicle just ended loading the previous Job, thus is going to deliver the Job:
    - a route needs to be generated to the delivery position, thus Route interface is called and `partial_route_first_request()` event is scheduled.

- Case 3: When the Vehicle `Status` is "Park": The Vehicle just ended unloading a Job and has no Job assigned to it currently, thus is going to be parked:
    - Case 3.1: The Vehicle's current position happens to be its park position, it is parked right away, thus `start_parking()` event is scheduled.
    - Case 3.2: The Vehicle's current position is not its park position, a route needs to be generated to the park position, thus Route interface is called and `partial_route_first_request()` event is scheduled.

The Route interface module is called in this event, where users write their own route algorithm. The return value from the interface, `route_list`, will be the generated route in this event and will be used to update `DynamicRoute` in __`Vehicle_df`. More information on the route interface can be found in section 3.6.2.

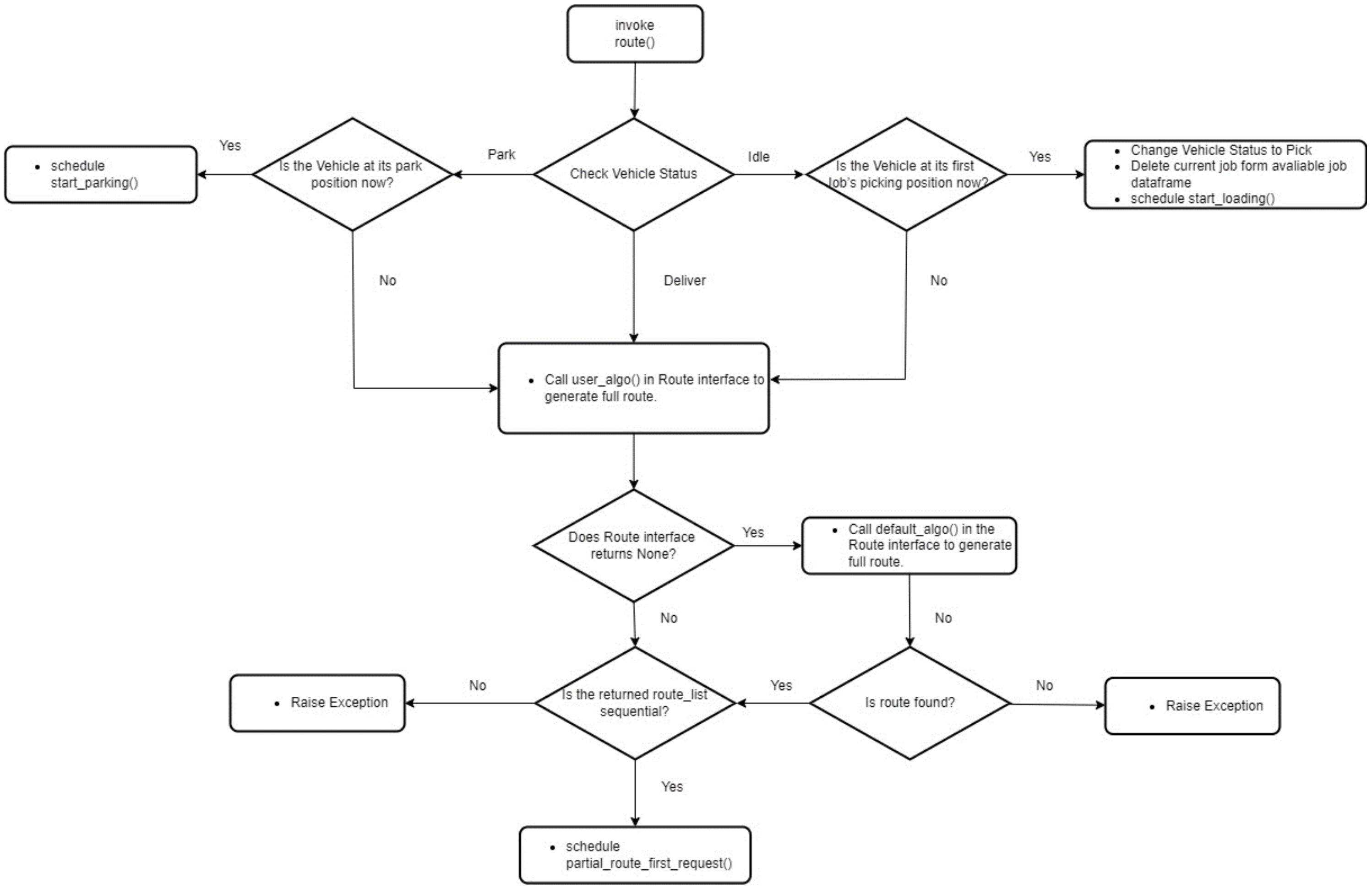

invoke route()
Check Vehicle Status
Park
Is the Vehicle at its park position now?
Yes
schedule start_parking()
Idle
Is the Vehicle at its first Job's picking position now?
Yes
Change Vehicle Status to Pick
Delete current job form avaliable job dataframe
schedule start_loading()
No
Deliver
No
Call user_algo() in Route interface to generate full route.
Does Route interface returns None?
Yes
Call default_algo() in the Route interface to generate full route.
No
No
Is the returned route_list sequential?
No
Raise Exception
Yes
Is route found?
No
Raise Exception
Yes
schedule partial_route_first_request()

Figure 15 - Flow Chart for route()

3.4.4 partial_route_first_request()

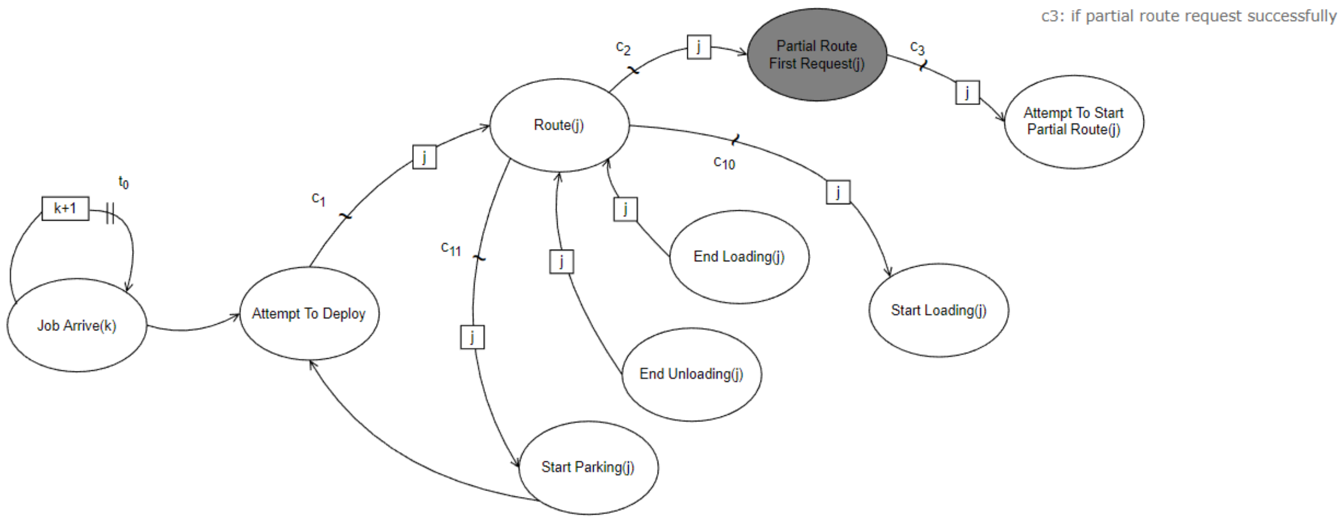


Figure 16 - EG for partial_route_first_request()

This event requests the first partial route of the Vehicle, and is only scheduled by `route()` event. At the start, the Request interface module is called in this event, where users write their own request algorithm. The return value from the interface, `partial_route`, will be the partial route to be requested. The request can be either successful or failed, which will affect the next event scheduled. The conditions are as follows:

- Case 1: Request Successful
  - If all the Square Units in the requested partial route have no vehicle reservation, i.e. the \`OccupiedVehicle\` in `grid_df` is NaN, then the Vehicle will reserve all the Square Units in the partial route, and the partial route is considered successfully requested. Subsequently, the \`OccupiedVehicle\` for these Square Units are updated into this Vehicle in `grid_df`, and `attempt_to_start_partial_route()` event is scheduled.
- Case 2: Request Failed
  - If any of the Square Units in the partial route is already reserved by another Vehicle, the Vehicle will fail to request the partial route, so the partial route is sent to the Vehicle's \`ReservationPending\`, and the request time is recorded in 'RequestTime' column in `vehicle_df`.

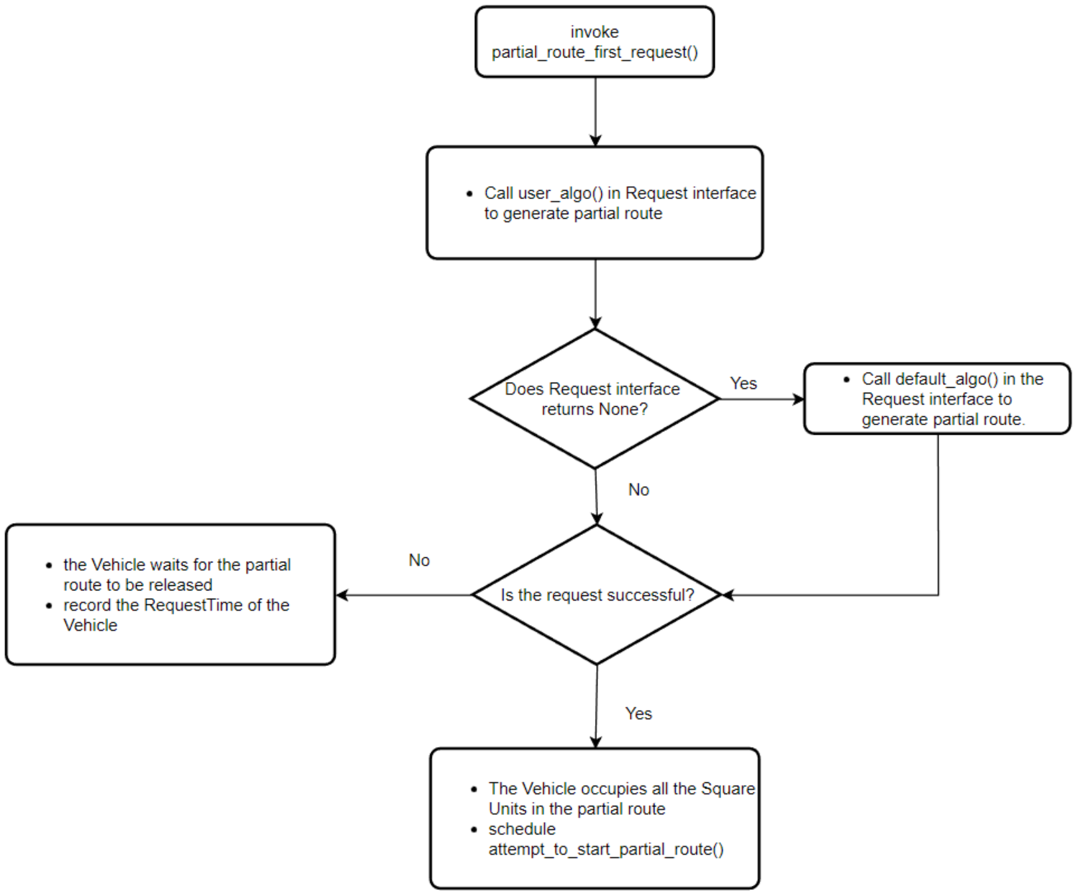

invoke partial_route_first_request()
Call user_algo() in Request interface to generate partial route
Does Request interface returns None?
Yes
Call default_algo() in the Request interface to generate partial route.
No
Is the request successful?
No
the Vehicle waits for the partial route to be released
record the RequestTime of the Vehicle
Yes
The Vehicle occupies all the Square Units in the partial route
schedule attempt_to_start_partial_route()

Figure 17 - Flow Chart for partial_route_first_request()

3.4.5 attempt_to_start_partial_route()

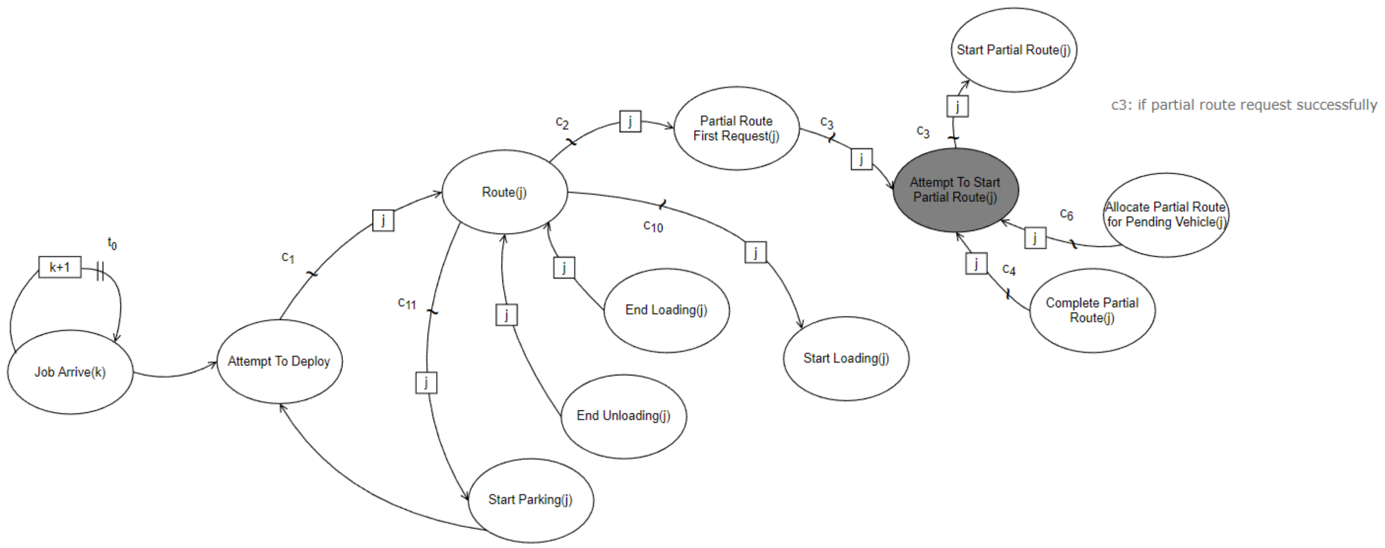

Start Partial Route(j)
c3: if partial route request successfully
Partial Route First Request(j)
Attempt To Start Partial Route(j)
Allocate Partial Route for Pending Vehicle(j)
Complete Partial Route(j)
Route(j)
End Loading(j)
Start Loading(j)
End Unloading(j)
Start Parking(j)
Attempt To Deploy
Job Arrive(k)
k+1
t_0
c_1
c_2
c_3
c_4
c_6
c_{10}
c_{11}
j

Figure 18 - EG for attempt_to_start_partial_route()

This event checks if the Vehicle can start moving along its partial route, i.e. to schedule `start_partial_route()` event. To achieve this, the Vehicle must make sure that it has successfully requested the partial route. Otherwise, the Vehicle cannot start traveling. Events `partial_route_first_request()`, `complete_partial_route()` and `allocate_partial_route_for_pending_vehicle()` can schedule `attempt_to_start_partial_route()` event.

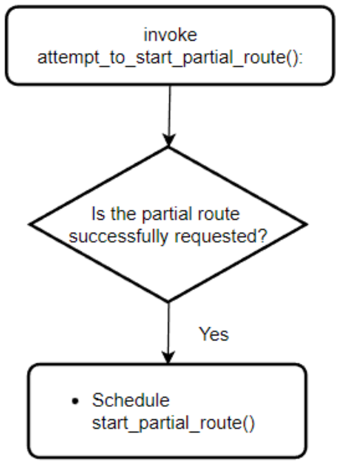

invoke attempt_to_start_partial_route():
Is the partial route successfully requested?
Yes
Schedule start_partial_route()

Figure 19 - Flow Chart for attempt_to_start_partial_route()

3.4.6 start_partial_route()

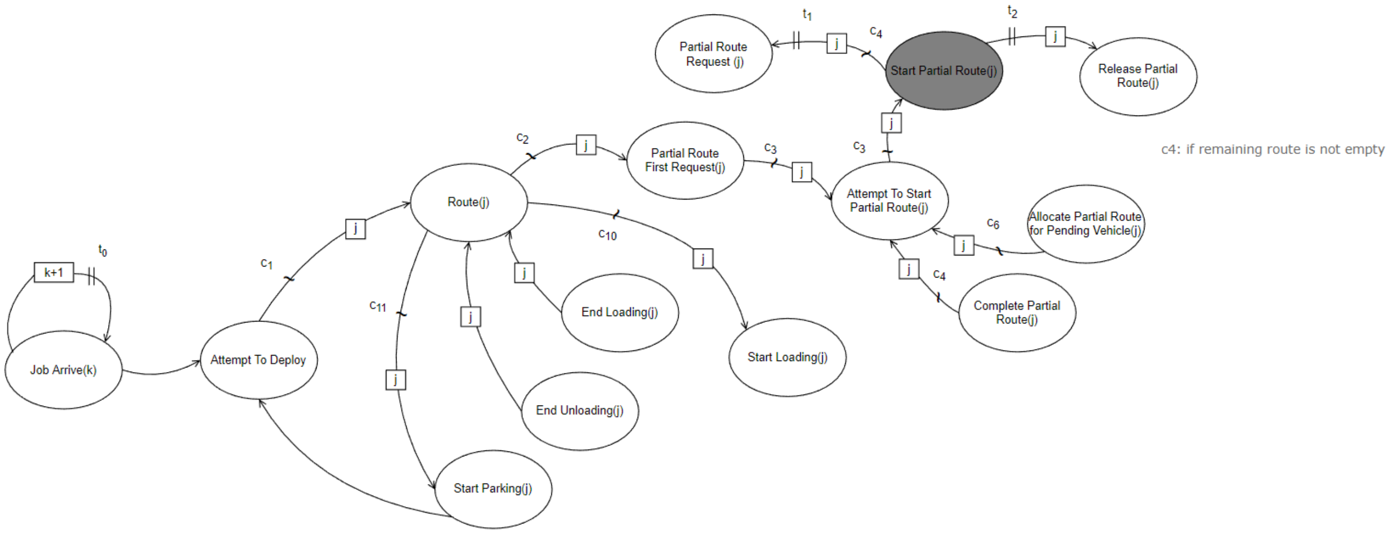

Partial Route Request (j)
Start Partial Route(j)
Release Partial Route(j)
c4: if remaining route is not empty
Partial Route First Request(j)
Route(j)
Attempt To Start Partial Route(j)
Allocate Partial Route for Pending Vehicle(j)
Complete Partial Route(j)
End Loading(j)
Start Loading(j)
End Unloading(j)
Start Parking(j)
Job Arrive(k)
Attempt To Deploy
k+1
t0
t1
t2
c1
c2
c3
c4
c6
c10
c11
j

Figure 20 - EG for start_partial_route()

In this event, Vehicle starts to travel its partial route. It first adds the current partial route to \`ReservationToRelease\` and deletes it from \`Reservation\` in `vehicle_df`. After some random time t_1 before the partial route is finished, the Vehicle requests for its next partial route, i.e. schedule `partial_route_request()` event, where \`Reservation\` in `vehicle_df` will be further updated. After a fixed period t_2, the partial route is finished (since the distance and vehicle pace are fixed), hence `release_partial_route()` event is scheduled.

One thing to take note of is that, at the start of this event, if Vehicle \`Status\` is "Idle", it means the Vehicle just ended unloading the previous Job item and heading towards the next assigned Job, or could just be assigned a new Job at its park position, therefore it is going to pick the Job. Hence, the \`Status\` will be changed to "Pick" and its current Job will be deleted from `available_job_df`.

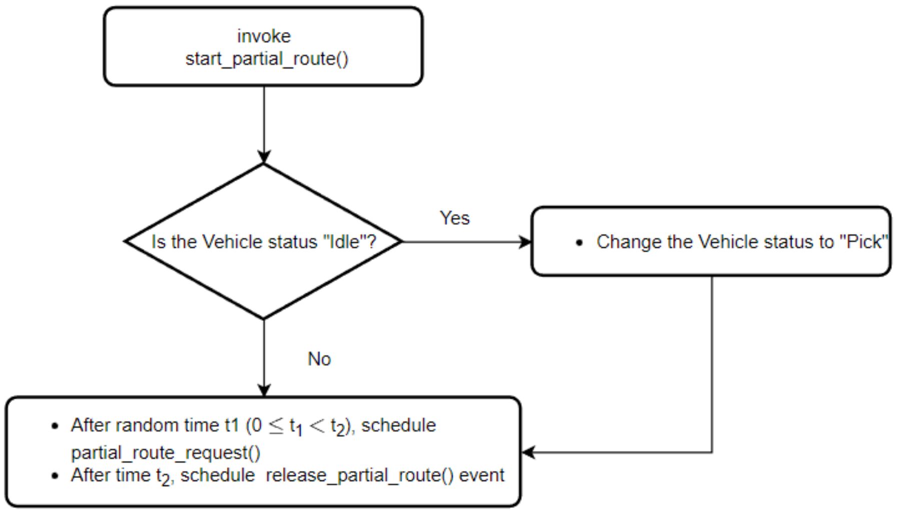

invoke start_partial_route()
Is the Vehicle status "Idle"?
Yes
Change the Vehicle status to "Pick"
No
After random time t1 ($0 \leq t_1 < t_2$), schedule partial_route_request()
After time t_2, schedule release_partial_route() event

Figure 21 - Flow Chart for start_partial_route()

3.4.7 partial_route_request()

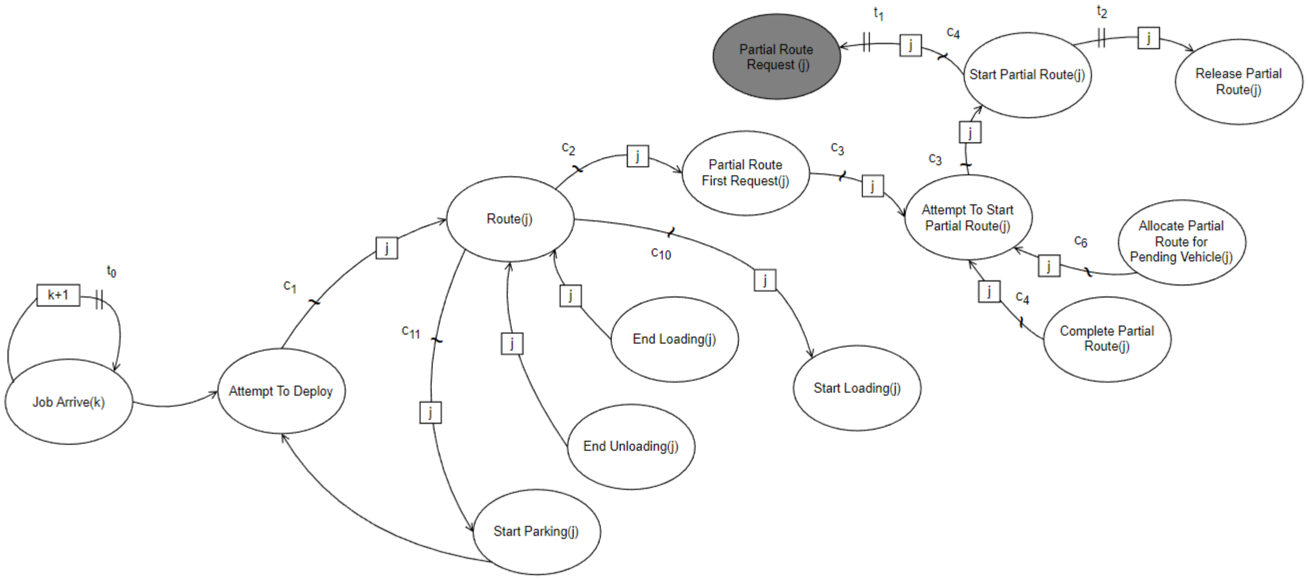


Figure 22 - EG for partial_route_request()

This event requests the partial route of the Vehicle, except for the first partial route and is scheduled by `start_partial_route()` event only. The logic in this event is similar to `partial_route_first_request()`, except that it does not schedule any event. The Request interface module is called in this event, where users write their own request algorithm. The return value from the interface, `partial_route`, will be the partial route to be requested. The request can be either successful or failed, which will affect the next event scheduled. The conditions are as follows:

- Case 1: Request Successful
  - If all the Square Units in the requested partial route have no vehicle reservation, i.e. the \`OccupiedVehicle\` in `grid_df` is NaN, then the Vehicle will reserve all the Square Units in the partial routes, and the partial route is considered successfully requested. Subsequently, the \`OccupiedVehicle\` for these Square Units is updated into this Vehicle in `grid_df`.
- Case 2: Request Failed
  - If any of the Square Units in the partial route is already reserved by another Vehicle, the Vehicle will fail to request the partial route, so the partial route is sent to the Vehicle's \`ReservationPending\` column, and the request time is recorded in 'RequestTime' column in `vehicle_df`.

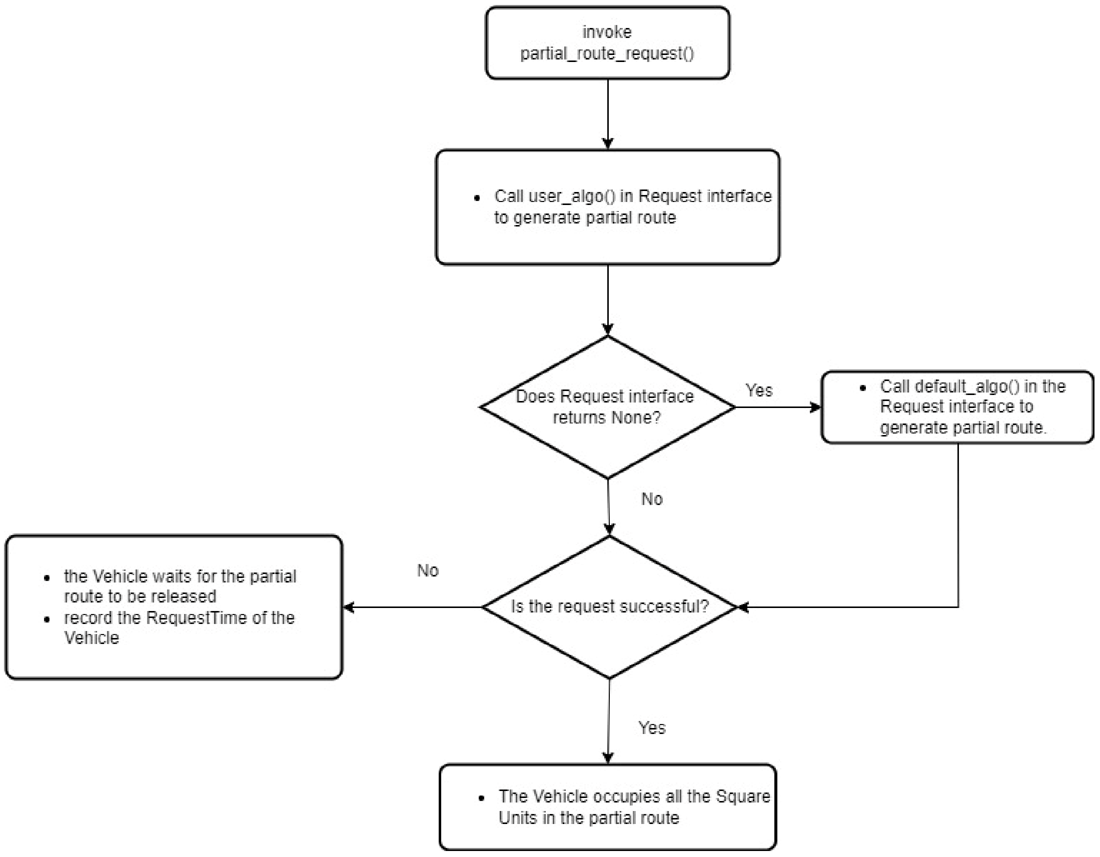

invoke partial_route_request()
Call user_algo() in Request interface to generate partial route
Does Request interface returns None?
Yes
Call default_algo() in the Request interface to generate partial route.
No
Is the request successful?
No
the Vehicle waits for the partial route to be released
record the RequestTime of the Vehicle
Yes
The Vehicle occupies all the Square Units in the partial route

Figure 23 - Flow Chart for partial_route_request()

3.4.8 release_partial_route()

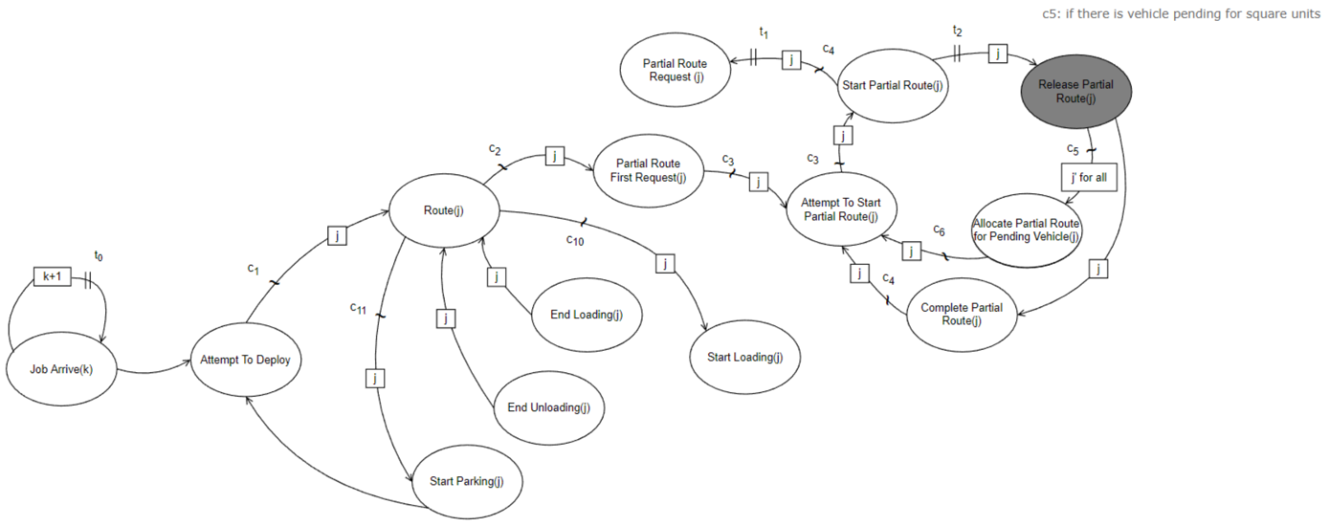

c5: if there is vehicle pending for square units
Partial Route Request (j)
t1
j
c4
Start Partial Route(j)
t2
j
Release Partial Route(j)
c5
j' for all
Allocate Partial Route for Pending Vehicle(j)
c6
j
c3
j
Attempt To Start Partial Route(j)
j
c4
Complete Partial Route(j)
j
c2
j
Partial Route First Request(j)
c3
j
Route(j)
c10
j
Start Loading(j)
j
End Loading(j)
j
End Unloading(j)
c11
j
Start Parking(j)
c1
j
Attempt To Deploy
k+1
t0
Job Arrive(k)

Figure 24 - EG for release_partial_route()

This event indicates the Vehicle has reached the last Square Unit of the partial route. It is scheduled by `start_partial_route()` event and releases the finished partial route. Also, it updates Vehicle `StartPosition`, schedules `complete_partial_route()` event and `allocate_partial_route_for_pending_vehicle()` event under certain conditions.

In detail, it first removes the finished partial route section from the Vehicle's `DynamicRoute` in `vehicle_df`, and sets the Square Unit that the Vehicle is currently at as the new `StartPosition` for the Vehicle's next partial route. Next, it releases the Square Units in the Vehicle's `ReservationToRelease` by setting the `OccupiedVehicle` in `grid_df` to NaN. However, the current `StartPosition` is excluded in releasing as it is still occupied by the Vehicle, but the previous `StartPosition` will be released. In addition, `VehicleVia` in `grid_df` for the released Square Units are increased by one, since one more Vehicle has passed through the Square Units. After that, the system checks if there are any Vehicles in the system pending to reserve a partial route, i.e. failed in the partial route request before. If true, `allocate_partial_route_for_pending_vehicle()` event will be scheduled. In the end, `complete_partial_route()` event is scheduled to indicate the completion of the partial route and data update.

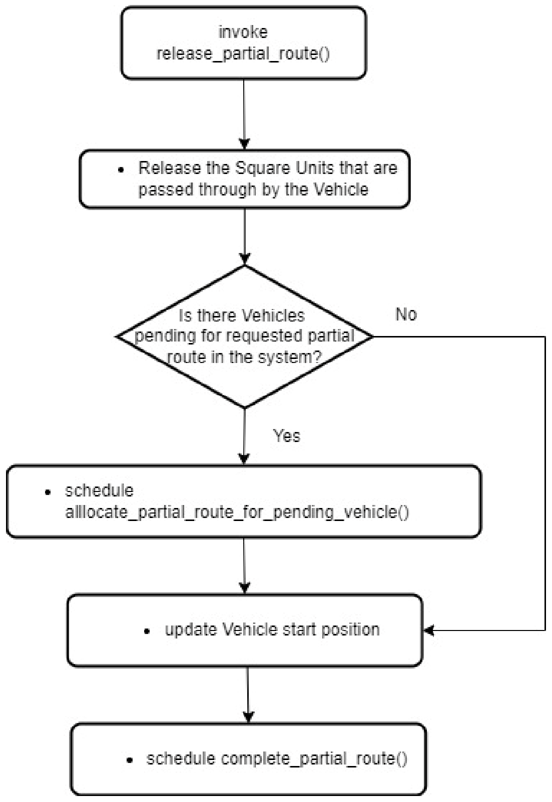

invoke release_partial_route()
Release the Square Units that are passed through by the Vehicle
Is there Vehicles pending for requested partial route in the system?
No
Yes
schedule alllocate_partial_route_for_pending_vehicle()
update Vehicle start position
schedule complete_partial_route()

Figure 25 - Flow Chart for release_partial_route()

3.4.9 allocate_partial_route_for_pending_vehicle()

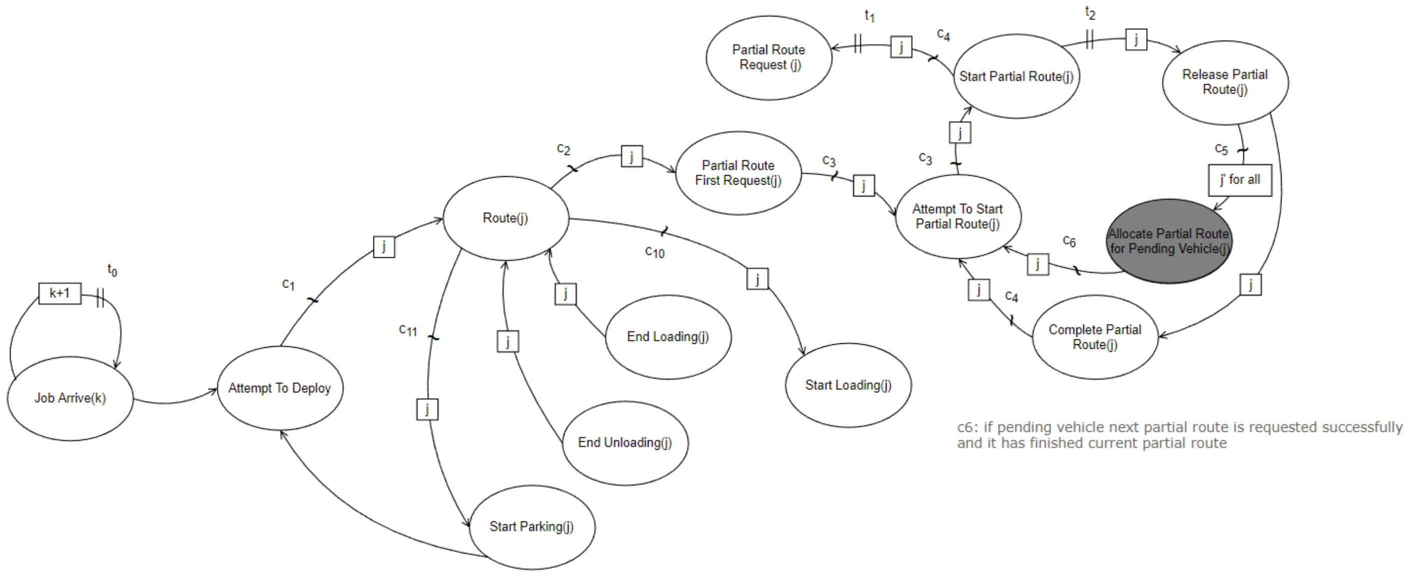

Partial Route Request (j)
Start Partial Route(j)
Release Partial Route(j)
Partial Route First Request(j)
Route(j)
Attempt To Start Partial Route(j)
Allocate Partial Route for Pending Vehicle(j)
Complete Partial Route(j)
Job Arrive(k)
Attempt To Deploy
End Loading(j)
Start Loading(j)
End Unloading(j)
Start Parking(j)
k+1
j
j' for all
t_0
t_1
t_2
c_1
c_2
c_3
c_4
c_5
c_6
c_{10}
c_{11}
c6: if pending vehicle next partial route is requested successfully and it has finished current partial route

Figure 26 - EG for allocate_partial_route_for_pending_vehicle()

This event allocates released Square Units to pending Vehicles, and schedules `attempt_to_start_partial_route()` event if the Vehicle's requested partial route is successfully allocated and the Vehicle has finished the current traveling partial route. The event is scheduled by `release_partial_route()` event, in which some Square Units are released, thus making way for the pending Vehicles. Sorting all the Vehicles that are pending for their partial routes in the system (`ReservationPending` is not NaN) according to `RequestTime` in ascending order, the system allocates partial routes to pending Vehicles from the oldest to the most recent one by one. As long as the Vehicle's requested partial route in `ReservationPending` becomes available now (`OccupiedVehicle` of the Square Units are NaN), the partial route will be allocated to the Vehicle. Hence, the data in `ReservationPending` will be moved to `Reservation` and the Vehicle will be set as the `OccupiedVehicle` for these Square Units. After that, the system proceeds to the next pending Vehicle. For all pending Vehicles that have been successfully allocated with their requested partial routes, and have completed the current traveling partial routes, `attempt_to_start_partial_route()` event will be scheduled for them.

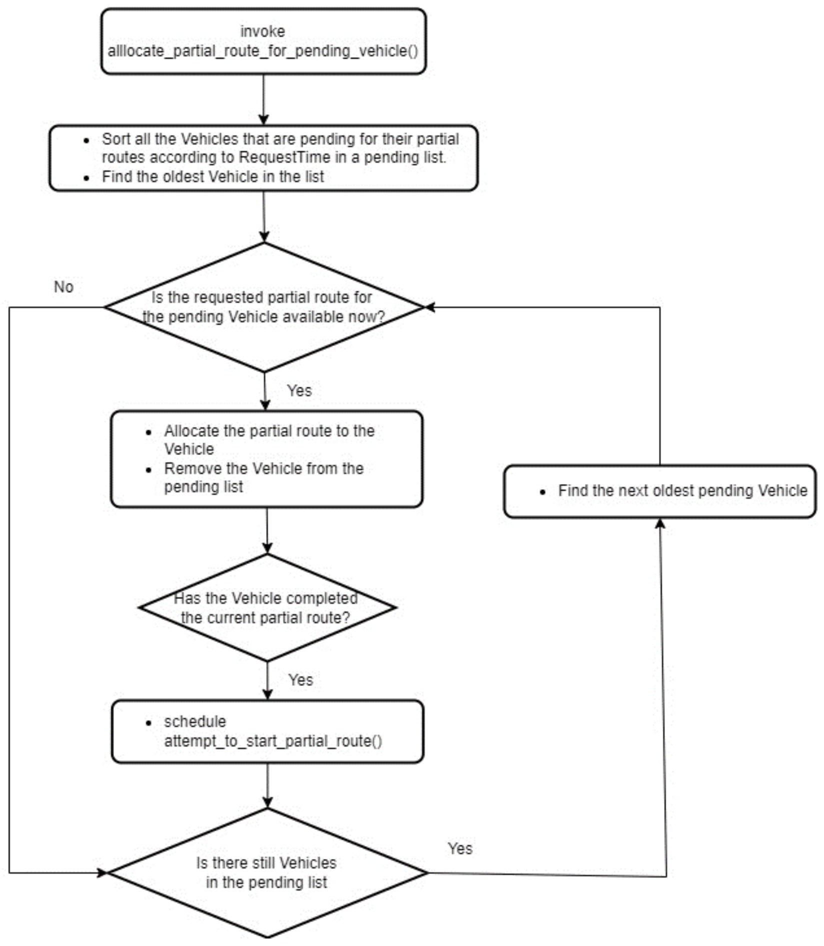
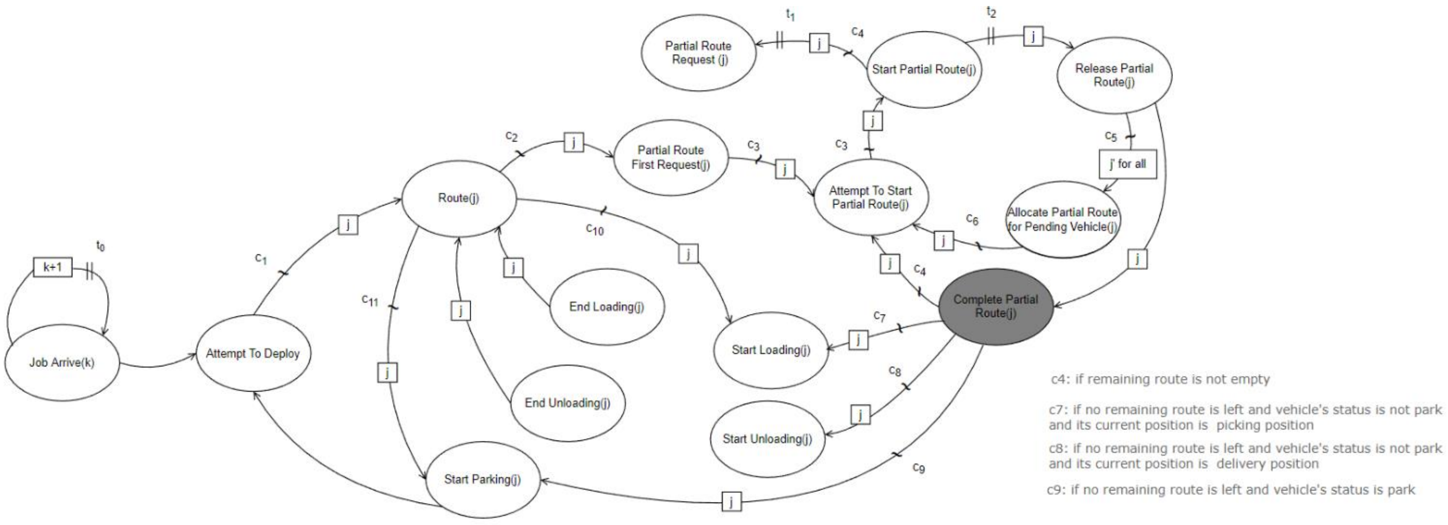

invoke alllocate_partial_route_for_pending_vehicle()
Sort all the Vehicles that are pending for their partial routes according to RequestTime in a pending list.
Find the oldest Vehicle in the list
Is the requested partial route for the pending Vehicle available now?
No
Yes
Allocate the partial route to the Vehicle
Remove the Vehicle from the pending list
Find the next oldest pending Vehicle
Has the Vehicle completed the current partial route?
Yes
schedule attempt_to_start_partial_route()
Is there still Vehicles in the pending list
Yes

Figure 27 - Flow Chart for allocate_partial_route_for_pending_vehicle()

3.4.10 complete_partial_route()



c4: if remaining route is not empty

c7: if no remaining route is left and vehicle's status is not park and its current position is picking position

c8: if no remaining route is left and vehicle's status is not park and its current position is delivery position

c9: if no remaining route is left and vehicle's status is park

Figure 28 - EG for complete_partial_route()

This event indicates the completion of the current partial route and records the completion time to `PartialRouteCompleteTime` in `vehicle_df`. It is scheduled by `release_partial_route()` event only, and it schedules `attempt_to_start_partial_route()`, `start_parking()`, `start_loading()`, `start_unloading()` based on different conditions, as described in the following:

- Case 1: If the Vehicle has not completed its full route, i.e. its `DynamicRoute` is not NaN
    - `attempt_to_start_partial_route()` event will be scheduled to try to start its next partial route.
- Case 2: If it completes its full route, meaning it has already reached its destination position to pick or deliver a Job, or to park itself, the next event will be scheduled based on conditions.
    - Case 2.1: When the Vehicle's `Status` is "Park", the Vehicle is going to park, thus `start_parking()` event will be scheduled.
    - Case 2.2: When the Vehicle's `Status` is "Pick" or "Deliver", the Vehicle is picking or delivering a Job
        - Case 2.1: If the Vehicle is at current Job's picking position, it reaches the destination to pick the Job, thus `start_loading()` event will be scheduled.
        - Case 2.2: If the Vehicle is at current Job's delivery position, it reaches the destination to deliver the Job, thus `start_unloading()` event will be scheduled.

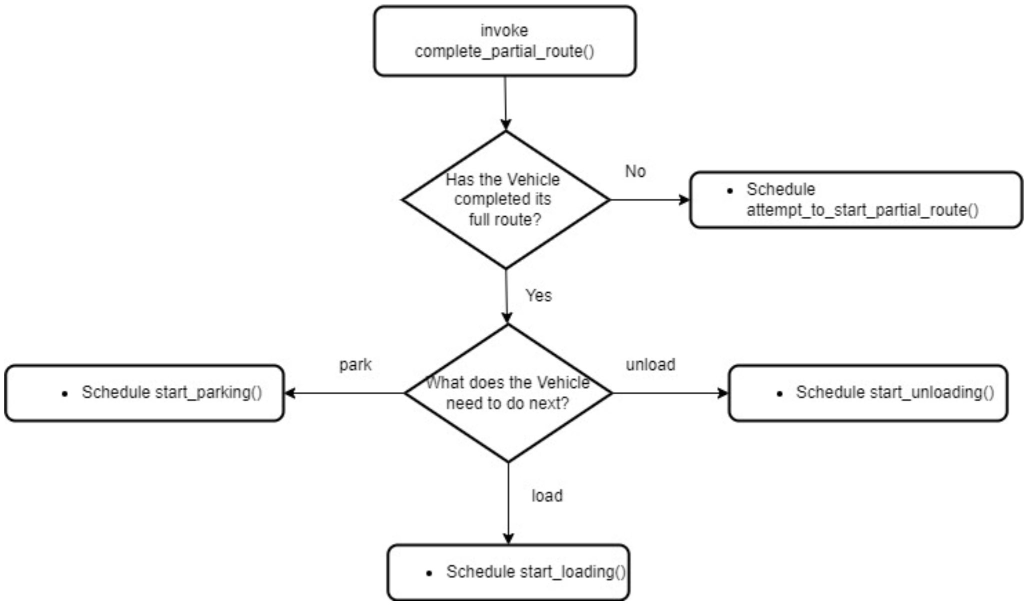

invoke complete_partial_route()
Has the Vehicle completed its full route?
No
Schedule attempt_to_start_partial_route()
Yes
What does the Vehicle need to do next?
park
Schedule start_parking()
unload
Schedule start_unloading()
load
Schedule start_loading()

Figure 29 - Flow Chart for complete_partial_route()

3.4.11 start_loading()

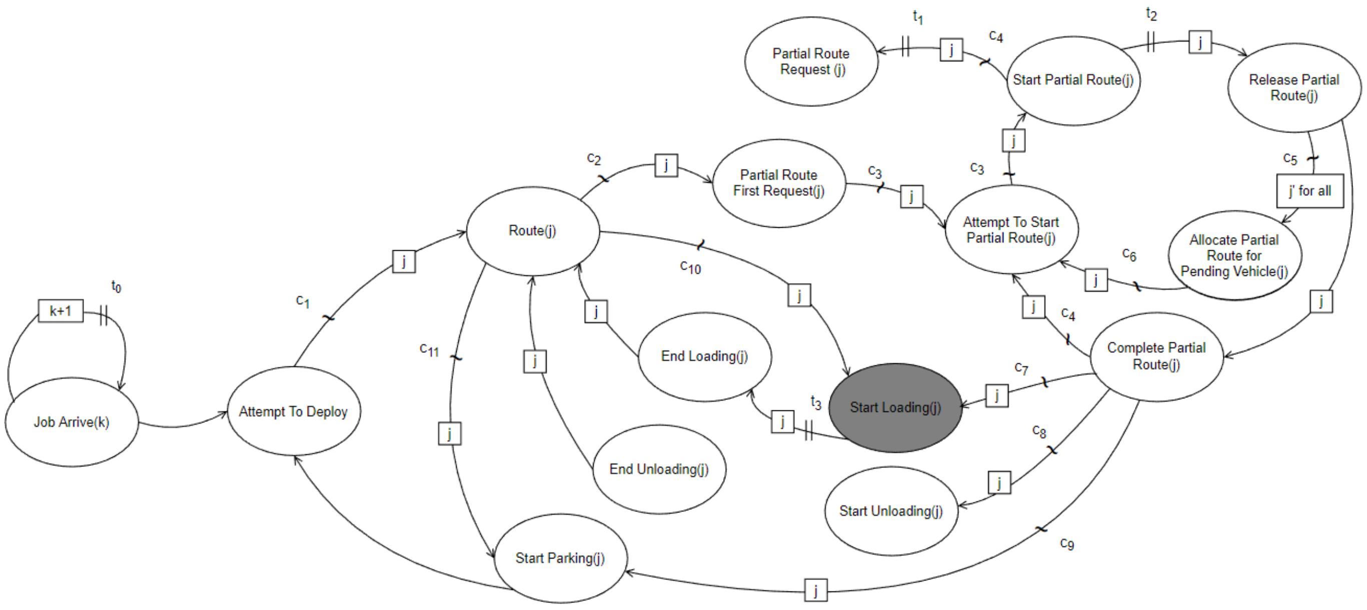


Figure 30 - EG for start_loading()

This event starts loading the Job item onto the Vehicle and clears `PartialRouteCompleteTime`. After a fixed period of loading duration t_3, which is specified in the model input, `end_loading()` event is scheduled.

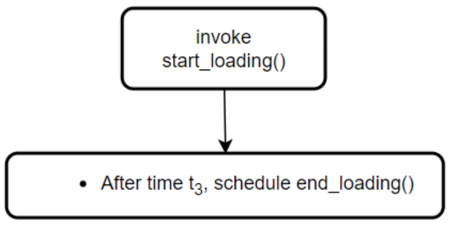


Figure 31 - Flow Chart for start_loading()

3.4.12 end_loading()

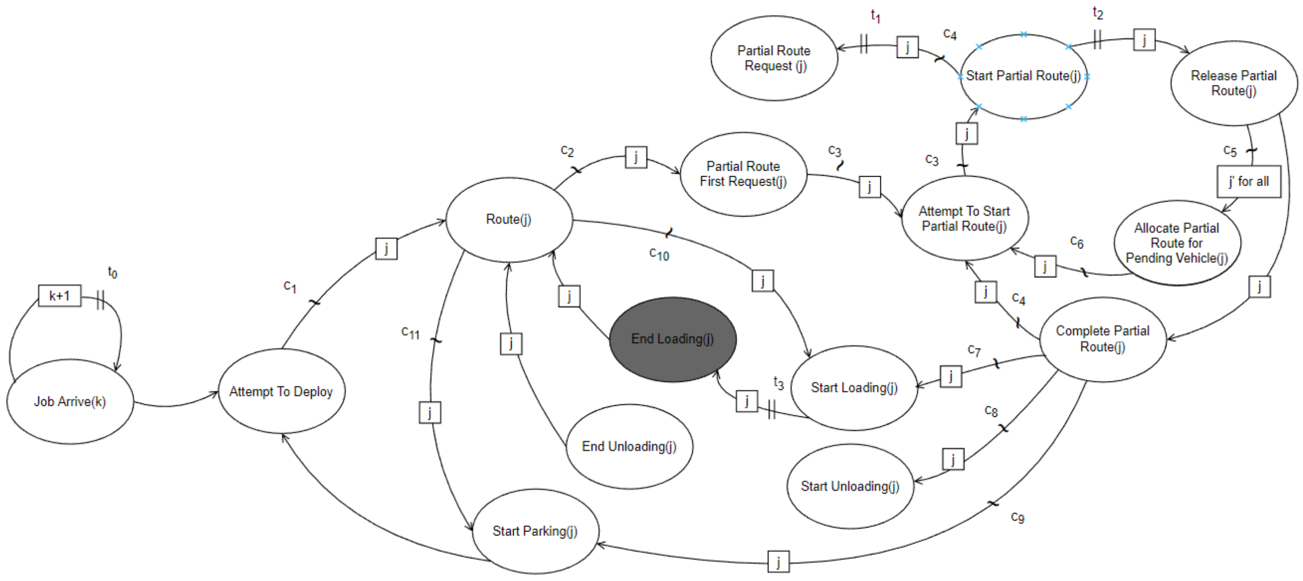


Figure 32 - EG for end_loading()

This event finishes loading the Job item onto the Vehicle. Since the Job item is already picked up, the Vehicle is going to deliver the Job. The Vehicle `Status` is changed to "Deliver" and `route()` event is scheduled to find the route to its delivery position.

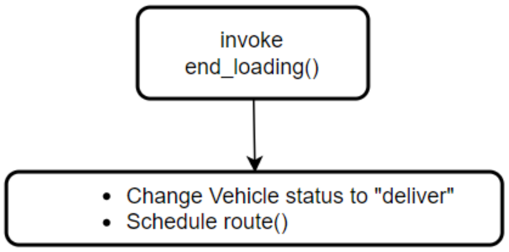


Figure 33 - Flow Chart for end_loading()

3.4.13 start_unloading()

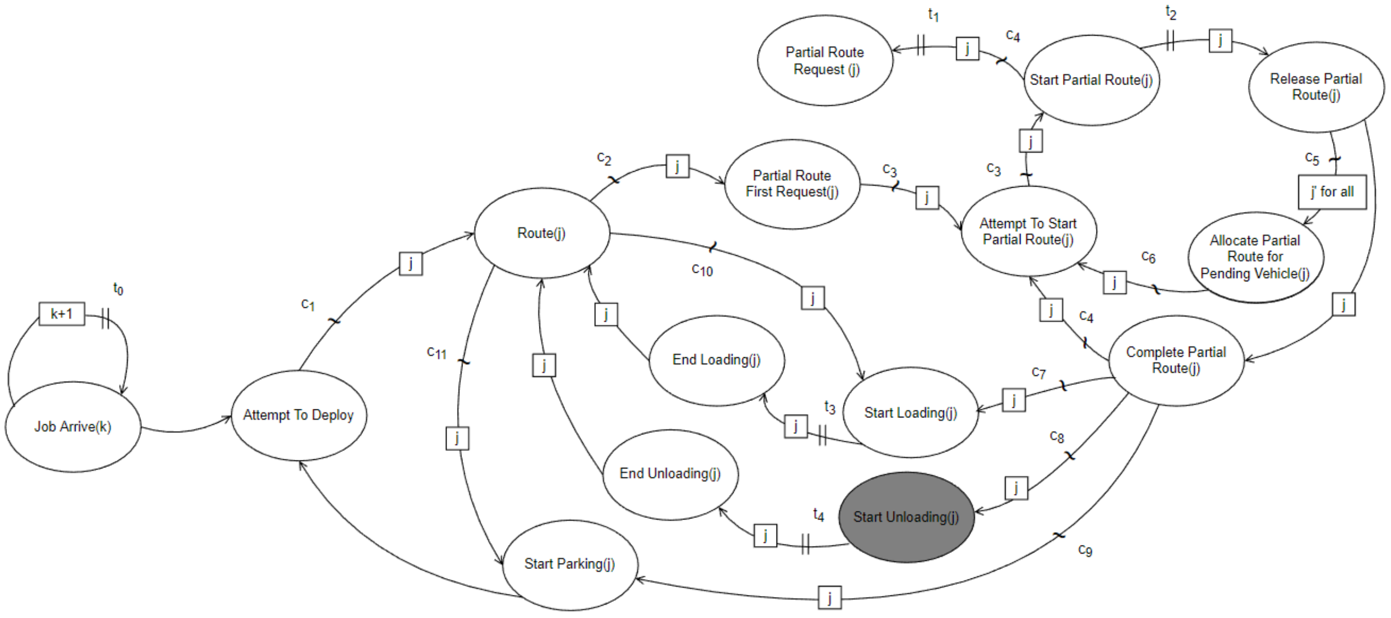


Figure 34 - EG for start_unloading()

This event starts unloading the Job item from the Vehicle and clears `PartialRouteCompleteTime`. After a fixed period of unloading duration t_4, which is specified in the model input, `end_unloading()` event is scheduled.

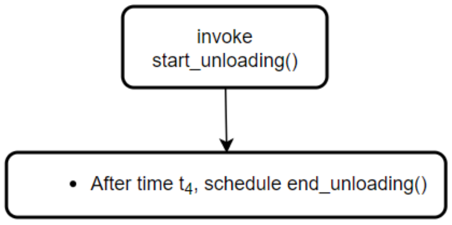


Figure 35 - Flow Chart for start_unloading()

3.4.14 end_unloading()

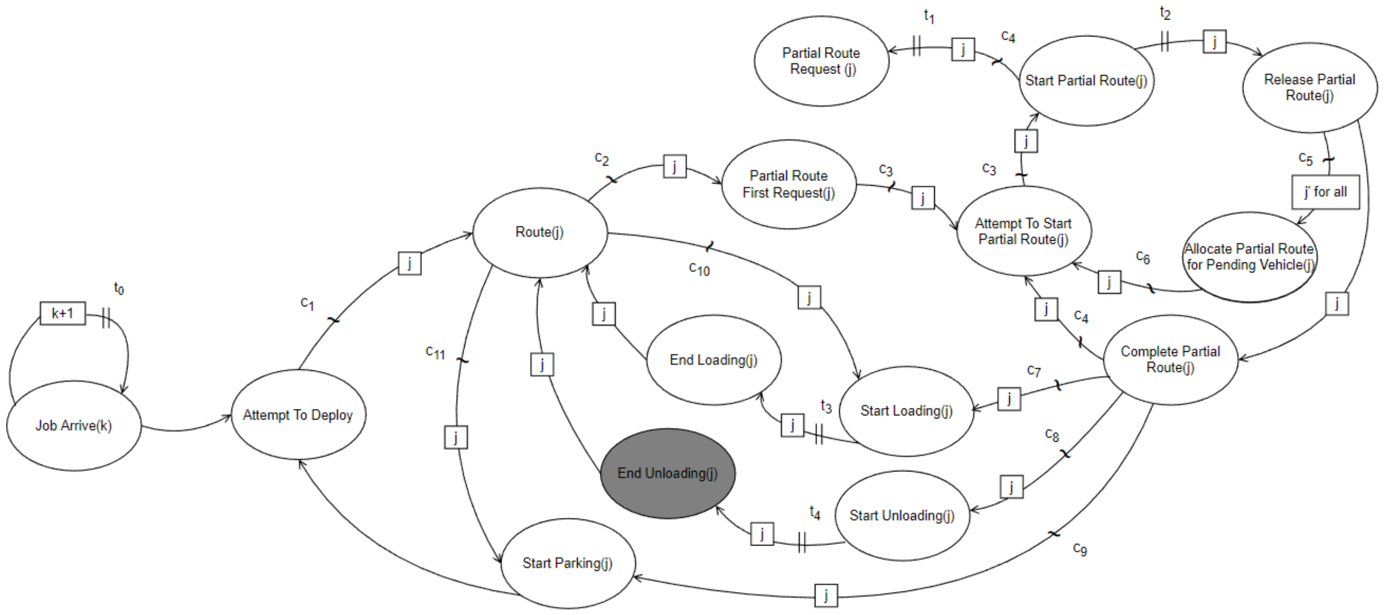


Figure 36 - EG for end_unloading()

This event finishes unloading the Job item from the Vehicle, and the Job is considered finished, thus is removed from the Vehicle's `JobList`. If the Vehicle still has Jobs in its `JobList`, the Vehicle `Status` will be changed to "Idle" and `route()` event will be scheduled to find the route to the picking position of the next Job. The reason for it being idle is that, as long as it has not started traveling on the route, the Vehicle's `JobList` is subject to change, depending on the deployment result. However, if the Vehicle has a NaN `JobList`, meaning it has no Job allocated, its `Status` will be turned into "Park" and `route()` event will be scheduled to find the route to its park position.

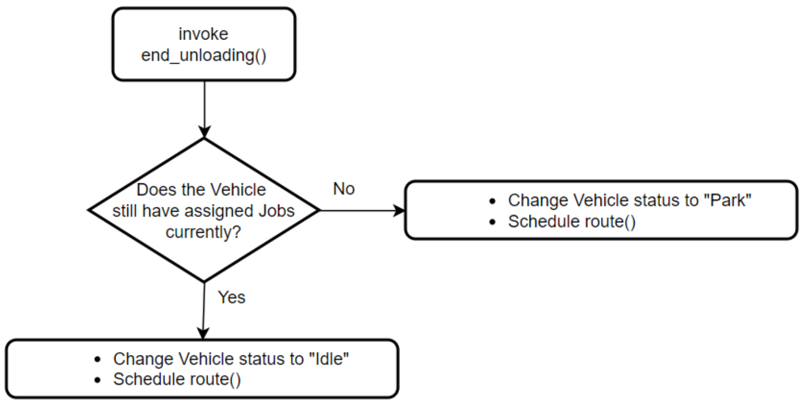

invoke end_unloading()
Does the Vehicle still have assigned Jobs currently?
No
Change Vehicle status to "Park"
Schedule route()
Yes
Change Vehicle status to "Idle"
Schedule route()

Figure 37 - Flow Chart for end_unloading()

3.4.15 start_parking()

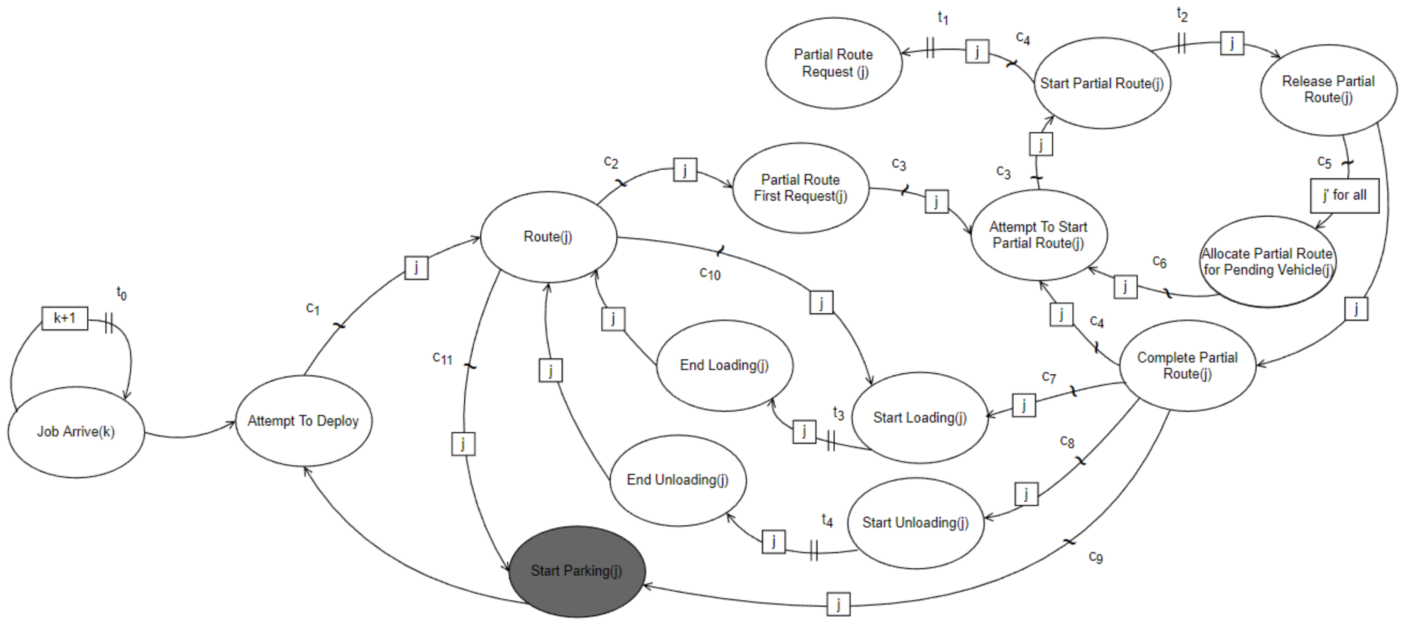
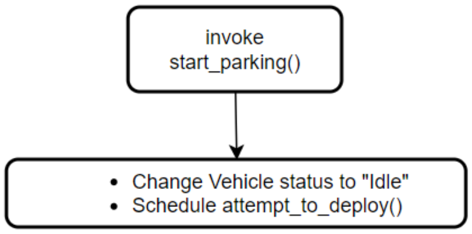


Figure 38 - EG for start_parking()

In this event, the Vehicle is parked at the park position, occupies the Square Unit. Therefore, it becomes the `OccupiedVehicle` of the park position Square Unit and its `Status` is changed from "Park" to "Idle", which means it is ready for new Job allocation or starting to handle new Job. Besides, `PartialRouteCompletTime` will be deleted. In the end, `attempt_to_deploy()` event is scheduled.



invoke start_parking()
• Change Vehicle status to "Idle"
• Schedule attempt_to_deploy()

Figure 39 - Flow Chart for start_parking()

3.5 Input

In the input file, some static attributes for the Grid Mover System are specified, including Transportation Network `id`, `start_point`, `dimension`, `obstacle_list` and available vehicle resources, including the Vehicle's `id`, and `park_position`.

Refer to Figure 40 for the attributes that are specified in the model input. They are highlighted in red in the ERD and the attribute definitions can be found in Table 2 and Table 4.

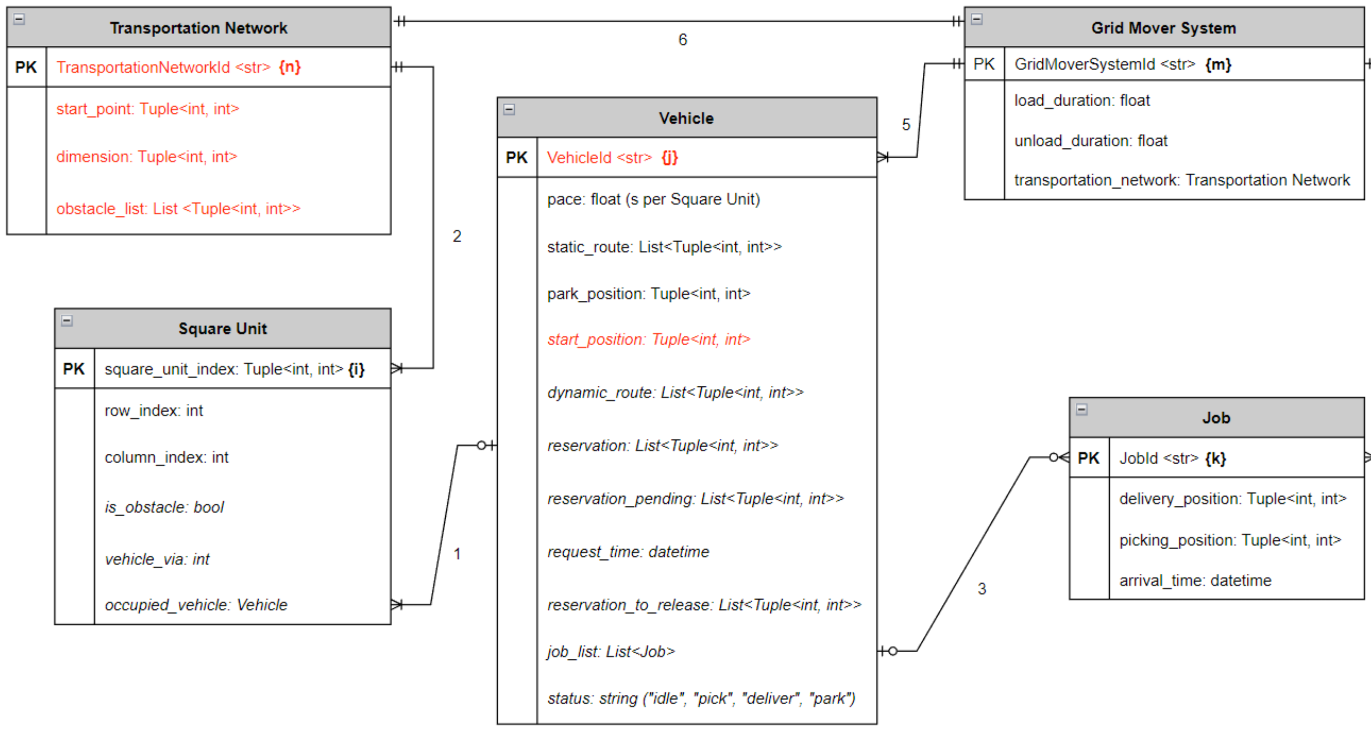

Transportation Network
PK TransportationNetworkId <str> {n}
start_point: Tuple<int, int>
dimension: Tuple<int, int>
obstacle_list: List <Tuple<int, int>>
Square Unit
PK square_unit_index: Tuple<int, int> {i}
row_index: int
column_index: int
is_obstacle: bool
vehicle_via: int
occupied_vehicle: Vehicle
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
1
2
3
4
5
6

Figure 40 - Model Input Data in ERD

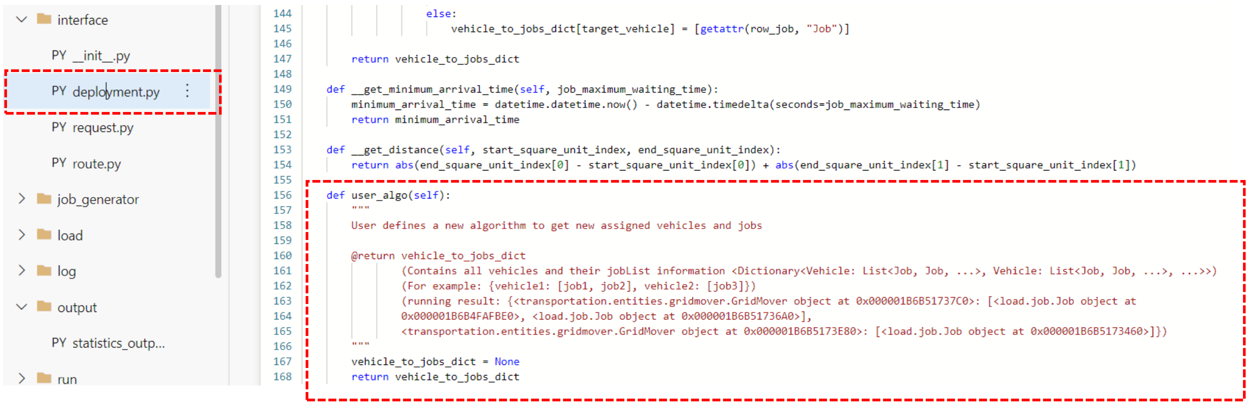
Other model inputs are associated with Job generations. The Jobs arrive at the Grid Mover System with an inter-arrival time of exponential distribution, and the rate parameter λ is specified in the input. Each Square Unit is associated with a Picking Rate and a Delivery Rate. The Picking Rate and Delivery Rate specify the comparative frequency at which a Square Unit becomes a Job's picking position and delivery position, with default value 1. Some Square Units are more popular picking positions while some are more popular delivery positions, or the other way around. Their Square Unit Indexes and the special Picking Rates and Delivery Rates are specified in the input. The input variables associated with Job generations are listed in Table 8. In the end, the assignment of picking and delivery locations to the arrived Jobs is calculated using the input with the Alias method.

Table 8 - Model Input Data not in ERD

Property Name	Type	Description
`Lambda`	int	The rate parameter of the exponential distribution of the Job inter-arrival time.
`DeliveryDefaultRate`	float	The default comparative frequency at which the Square Unit becomes a Job's delivery position.
`PickingDefaultRate`	float	The default comparative frequency at which the Square Unit becomes a Job's picking position.
`SquareUnitIndex`	Tuple<int, int>	The `square_unit_index` of the Square Unit which has a special Picking Rate and/or Delivery Rate.
`DeliveryRate`	float	The special comparative frequency at which the Square Unit becomes a Job's delivery position.
`PickingRate`	float	The special comparative frequency at which the Square Unit becomes a Job's picking position.

3.6 Interface

There are three interface modules connected to the Grid Mover System Handler: Deployment, Route and Request, each contains default algorithms for Job-Vehicle matching deployment, vehicle route planning, and collision avoidance by partial route request. In each module, the users can edit the code and write their coded algorithm inside the `user_algo()` function. Figure 41 illustrates an example of the location of `user_algo()` in Deployment Interface. Similar to Route and Request interfaces.



```
                else:
                    vehicle_to_jobs_dict[target_vehicle] = [getattr(row_job, "Job")]

        return vehicle_to_jobs_dict

    def __get_minimum_arrival_time(self, job_maximum_waiting_time):
        minimum_arrival_time = datetime.datetime.now() - datetime.timedelta(seconds=job_maximum_waiting_time)
        return minimum_arrival_time

    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_square_unit_index[1])

    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: List<Job, Job, ...>, ...>>)
                (For example: {vehicle1: [job1, job2], vehicle2: [job3]})
                (running result: {<transportation.entities.gridmover.GridMover object at 0x000001B6B51737C0>: [<load.job.Job object at
                0x000001B6B4FAFBE0>, <load.job.Job object at 0x000001B6B51736A0>],
                <transportation.entities.gridmover.GridMover object at 0x000001B6B5173E80>: [<load.job.Job object at 0x000001B6B5173460>]})
        """
        vehicle_to_jobs_dict = None
        return vehicle_to_jobs_dict
```

Figure 41 - An Example of the Location of user_algo() in the Deployment Interface

Other than that, the users can use the data that are available in the interface modules in their algorithms, and the algorithms must return data back to the main Grid Mover System Handler in a format that is the same as the default algorithm. If not, the Grid Mover System Handler may return errors. If the user algorithm returns None, it will be skipped, and the default algorithm will be executed. Therefore, the users must be familiar with the input and output data for each interface module, i.e. data available in the interface and data to be returned. They are explained in the following section.

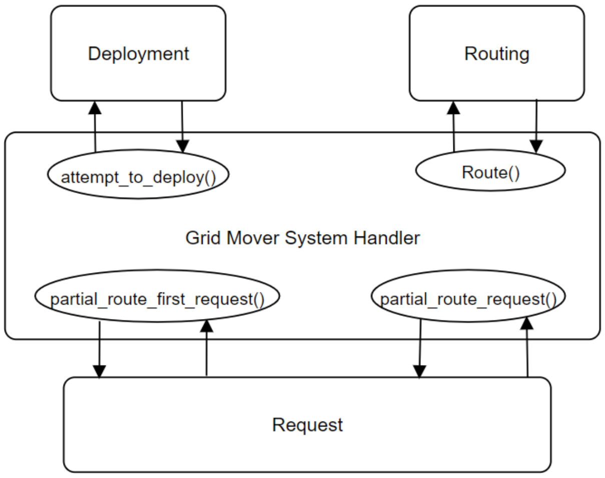


Figure 42 - The Relationship Between the Grid Mover System Handler and the Interface Modules

3.6.1 Deployment

Deployment interface module is called in `attempt_to_deploy()` event, which attempts to match each Job in `available_job_df` with a Vehicle based on written algorithms. The input parameters are two DataFrames, `available_job_df` and `vehicle_df` from the Grid Mover System Handler, and the return value is a dictionary `all_vehicle_to_jobs_dict`, which will overwrite the original `vehicle_df` in the Grid Mover System Handler based on certain rules (see `attempt_to_deploy()` event in section 3.4.2). The description of `available_job_df` and `vehicle_df` can be found in section 3.2, and the description of `all_vehicle_to_jobs_dict` is in Table 9.

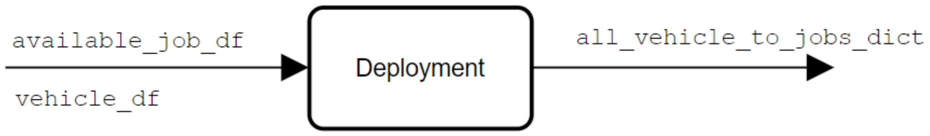


Figure 43 - Input Data and Return data for Deployment Interface Module

Table 9 - Input and Output Variables for Deployment Interface

Type	Variable	Type	Description
Input	`available_job_df`	DataFrame	See section 3.2.3
	`vehicle_df`	DataFrame	See section 3.2.1
Return	`all_vehicle_to_jobs_dict`	Dictionary<Vehicle: List<Job>>	A dictionary where the Vehicle objects are the keys and the lists of Job objects deployed to the respective Vehicles are the values. e.g. `{Vehicle1: [Job1, Job2], Vehicle2: [Job3],...}` Each Job in the `available_job_df` is attempted to be assigned to a Vehicle. All Vehicle objects in the Grid Mover System appear as keys in the dictionary.

The default deployment algorithm is described in Figure 44, which allocates each Job in `available_job_df` with the nearest Vehicles that are within the acceptable range of distance.

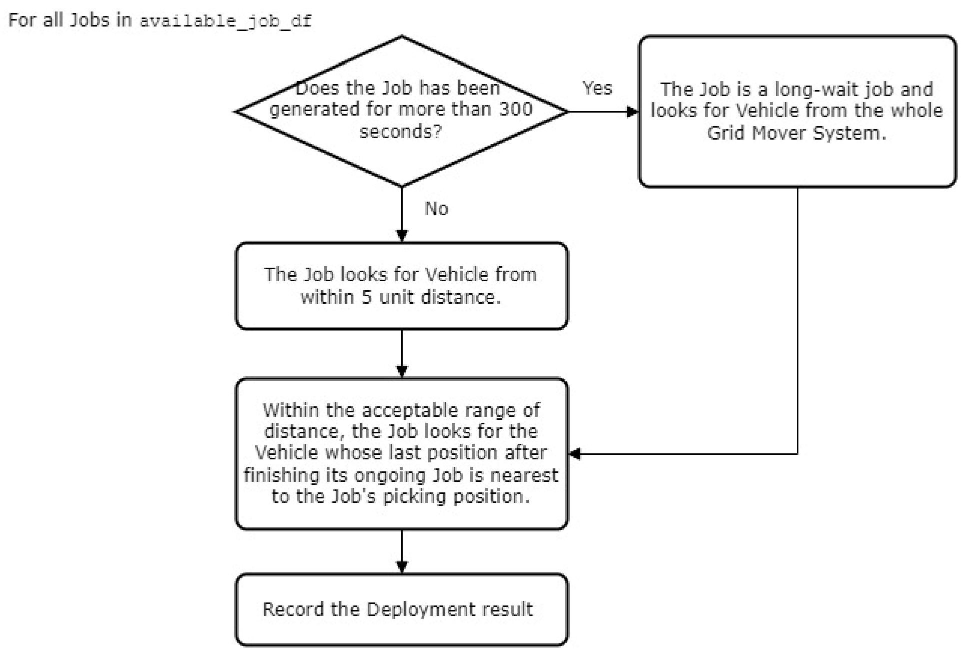

For all Jobs in available_job_df
Does the Job has been generated for more than 300 seconds?
Yes
No
The Job is a long-wait job and looks for Vehicle from the whole Grid Mover System.
The Job looks for Vehicle from within 5 unit distance.
Within the acceptable range of distance, the Job looks for the Vehicle whose last position after finishing its ongoing Job is nearest to the Job's picking position.
Record the Deployment result

Figure 44 - Deployment Default Algorithm

3.6.2 Route

Route interface module is called in `route()` event, which calculates routes for the Vehicles from their origin to destination Square Units based on written algorithms. The input parameters are two objects `transportation_network` and `vehicle` (see section 3.1 for description), and two other variables `start_square_unit` and `end_square_unit`, whose descriptions can be found in Table 10. The return value is a list of Tuple integer pairs `path`, which will overwrite the values of `DynamicRoute' in the original `vehicle_df` in the Grid Mover System Handler based on certain rules (see `route()` event in section 3.4.3). The description of `path` variable can be found in Table 10.

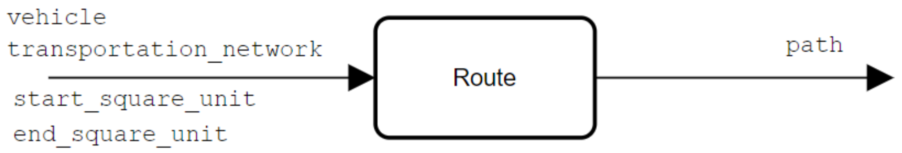


Figure 45 - Input data and Return Data for Route Interface Module

Table 10 - Input and Output Variables for Route Interface

Type	Variable	Type	Description
Input	`vehicle`	Vehicle	See section 3.1
	`transportation_network`	Transportation Network	See section 3.1
	`start_square_unit`	Tuple<int,int>	The `square_unit_index` of the origin Square Unit of the Vehicle.
	`end_square_unit`	Tuple<int,int>	The `square_unit_index` of the destination Square Unit of the Vehicle.
Return	`route`	List<Tuple<int, int>>	The list of `square_unit_index` of the Square Units in the assigned route of the Vehicle, including `start_square_unit` and `end_square_unit`

The default route algorithm uses A* algorithm to generate routes for the Vehicles.

3.6.3 Request

Request interface module is called in `partial_route_request()` and `partial_route_first_request()` events, which reserves the partial route for a traveling Vehicle to avoid traffic collision. The input parameters are one object `vehicle` (see section 3.1 for description), and two DataFrames `vehicle_df` and `grid_df`, whose descriptions can be found in Section 3.2. The return value is a list of Tuple integer pairs `partial_route`, which will overwrite the values of `Reservation` or `ReservationPending` in the original `vehicle_df` in the Grid Mover System Handler depending on if the request being successful or not .The `OccupiedVehicle` in `grid_df` will also be updated if the partial route request is successful (see `partial_route_request()` and `partial_route_first_request()` event in section 3.4.4 and 3.4.7 for the conditional update).The description of return variable `partial_route` can be found in Table 11.

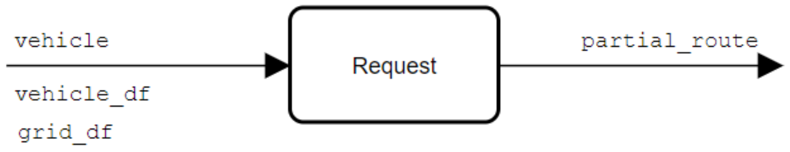


Figure 46 - Input data and Return Data for Request Interface Module

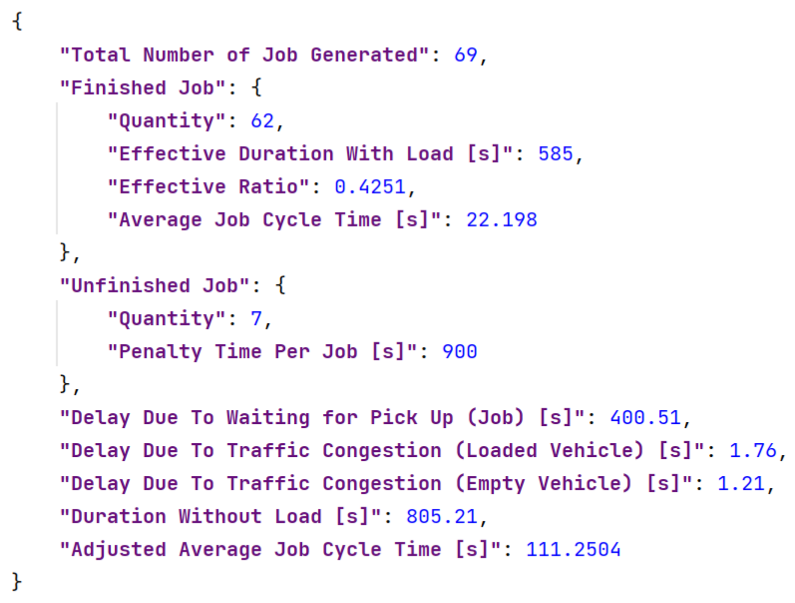
Table 11 - Input and Output Variables for Request Interface

Type	Variable	Type	Description
Input	`vehicle`	Vehicle	See Section 3.1
	`vehicle_df`	DataFrame	See Section 3.2.1
	`grid_df`	DataFrame	See Section 3.2.2
Return	`partial_route`	List<Tuple<int, int>>	The list of `square_unit_index` of the Square Units in the partial route to be reserved for the Vehicle, excluding the `start_position` of the Vehicle

In the default request algorithm, the Vehicle requests 3 Square Units from its remaining route as its next partial route. The remaining route refers to the remaining Square Units in its full route after the Vehicle completes its current partial route.

3.7 Output

The output generated for each simulation run is for users to observe the performance of the Grid Mover System, which is illustrated in Figure 47. The output variables include the Total Number of Jobs Generated, statistics on finished Jobs and unfinished Jobs, and average cycle time. The output variables and their definitions are listed in Table 12.



```
{
    "Total Number of Job Generated": 69,
    "Finished Job": {
        "Quantity": 62,
        "Effective Duration With Load [s]": 585,
        "Effective Ratio": 0.4251,
        "Average Job Cycle Time [s]": 22.198
    },
    "Unfinished Job": {
        "Quantity": 7,
        "Penalty Time Per Job [s]": 900
    },
    "Delay Due To Waiting for Pick Up (Job) [s]": 400.51,
    "Delay Due To Traffic Congestion (Loaded Vehicle) [s]": 1.76,
    "Delay Due To Traffic Congestion (Empty Vehicle) [s]": 1.21,
    "Duration Without Load [s]": 805.21,
    "Adjusted Average Job Cycle Time [s]": 111.2504
}
```

Figure 47 - An Illustration on Model Output

Table 12 - Model Output Data

Output Variable		**Unit**	**Type**	**Description**
`Total Number of Job Generated`		-	int	The total number of jobs generated during the simulation time.
`Finished job`	`Quantity`	-	int	The total number of jobs that are completed successfully, i.e., successfully unloaded at the delivery position.
	`Effective Duration with Load`	seconds	float	The total sum of durations during which the Vehicles travel carrying a load, excluding all the waiting times for Square Units.
	`Effective Ratio`	-	float	The ratio between `Effective Duration with Load` to Total cycle time for finished jobs (from being generated to being unloaded successfully).
	`Average Job Cycle Time`	seconds	float	Total cycle time for finished jobs / `Quantity` for finished job

Unfinished job	Quantity	-	int	The total number of jobs that are not completed.
	Penalty Time Per Job	seconds	float	Penalty time for each unfinished job. It depends on the total number of square units in the transportation network.
Delay Due to Waiting for Pick Up (Job)		seconds	float	The sum of durations of jobs waiting for vehicles to pick up.
Delay Due to Traffic Congestion (Loaded Vehicle)		seconds	float	The pending duration is caused by traffic congestion (i.e. waiting for square units) while the vehicle is loaded.
Delay Due to Traffic Congestion (Empty Vehicle)		seconds	float	The pending duration is caused by traffic congestion while the vehicle is empty.
Duration Without Load		seconds	float	The total sum of durations during which the vehicles travel without carrying a load.
Adjusted Average Job Cycle Time		seconds	float	(Total cycle time for finished job + Quantity for unfinished job * Penalty Time Per Job for unfinished job) / Total Number of Job Generated

Chapter 4

Evaluation

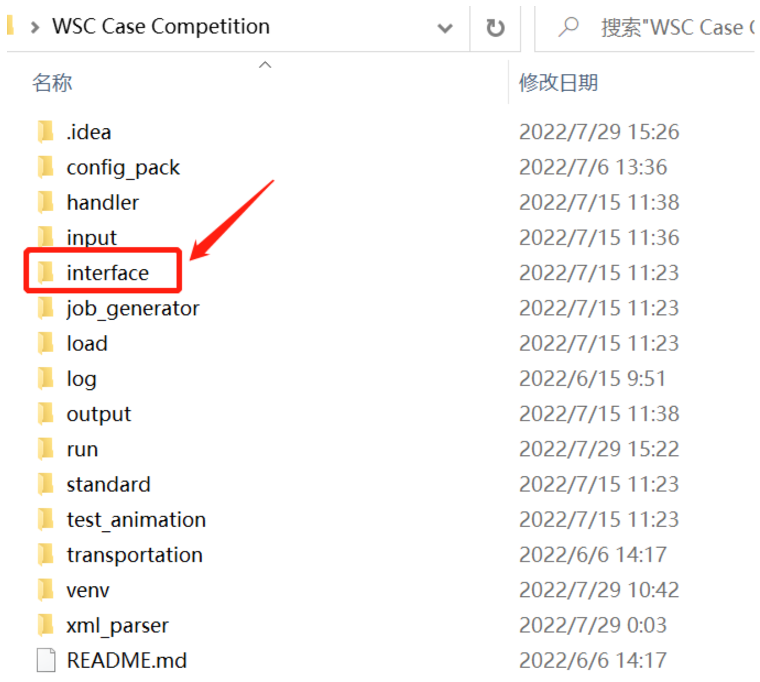
4.1 File Submission Format

1) Zip package of **interface** folder

   Attention:

   a. Any modifaction related to interface should be put under interface folder, including newly created class for interface algorithms purposes. Changes in other folders will not be considered.

   b. Beside interface folder, a text document of additional Python package installed should be included in the zip file.



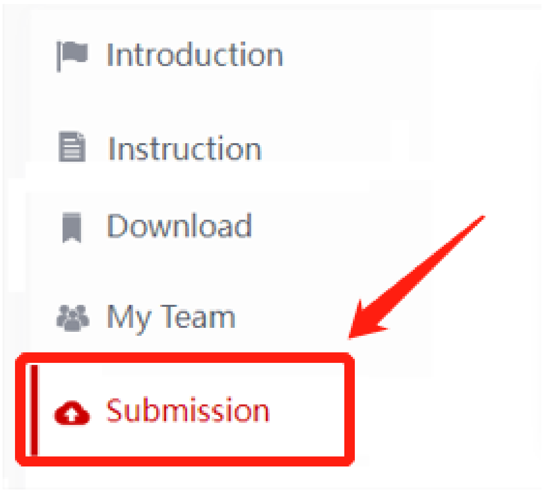
WSC Case Competition | 搜索"WSC Case (

名称	修改日期
.idea	2022/7/29 15:26
config_pack	2022/7/6 13:36
handler	2022/7/15 11:38
input	2022/7/15 11:36
interface	2022/7/15 11:23
job_generator	2022/7/15 11:23
load	2022/7/15 11:23
log	2022/6/15 9:51
output	2022/7/15 11:38
run	2022/7/29 15:22
standard	2022/7/15 11:23
test_animation	2022/7/15 11:23
transportation	2022/6/6 14:17
venv	2022/7/29 10:42
xml_parser	2022/7/29 0:03
README.md	2022/6/6 14:17

2) Naming rule: **Team Name_Round Number** (e.g. **SealTeam_Round1.zip**)

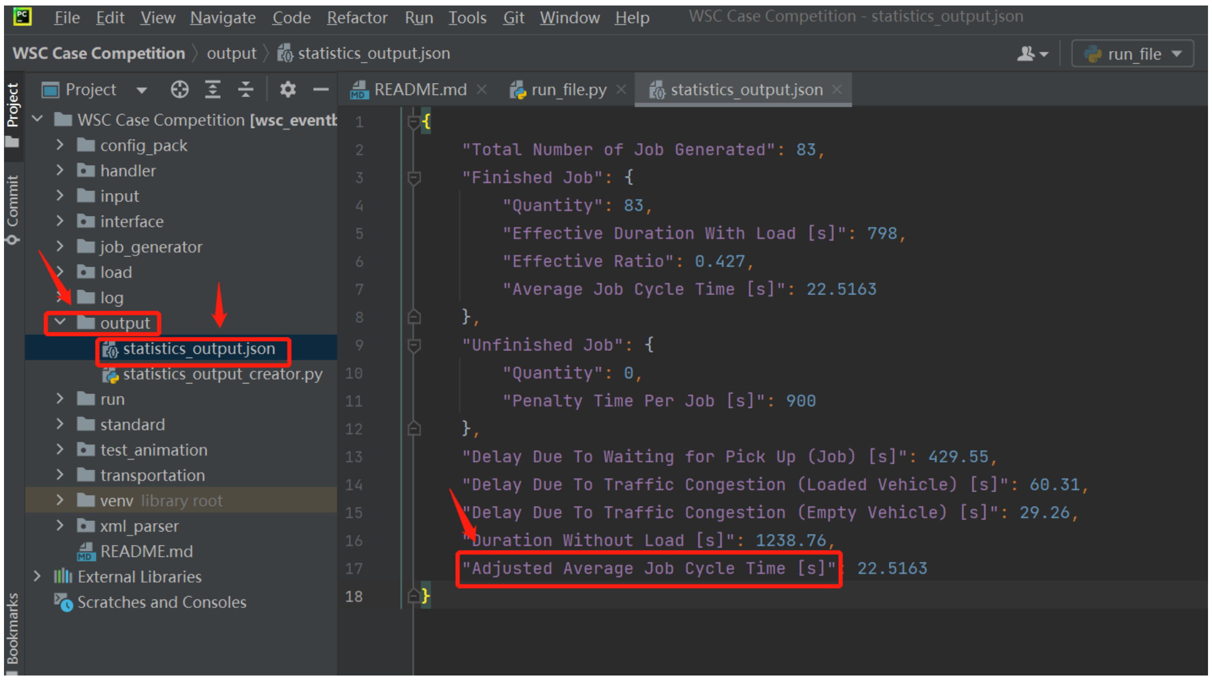
4.2 File Submission Process

After joining a team, there will be a **Submission** section for competitors to submit files.



4.3 Evaluation Criteria

1) Competitors' interface will be adopted to simulate the given scenario as well as the hidden scenario for the last round.
2) "**Adjusted Average Job Cycle Time**" from the output is the only criteria to evaluate system performance.



```
{
    "Total Number of Job Generated": 83,
    "Finished Job": {
        "Quantity": 83,
        "Effective Duration With Load [s]": 798,
        "Effective Ratio": 0.427,
        "Average Job Cycle Time [s]": 22.5163
    },
    "Unfinished Job": {
        "Quantity": 0,
        "Penalty Time Per Job [s]": 900
    },
    "Delay Due To Waiting for Pick Up (Job) [s]": 429.55,
    "Delay Due To Traffic Congestion (Loaded Vehicle) [s]": 60.31,
    "Delay Due To Traffic Congestion (Empty Vehicle) [s]": 29.26,
    "Duration Without Load [s]": 1238.76,
    "Adjusted Average Job Cycle Time [s]": 22.5163
}
```

3) In the evaluation stage, multiple random seeds will be used to calculate the average performance for each round. And only codes that run successfully will get scores.

4) Weightage and score of each round:

Weightage Table

Round Number	**Round 1**	**Round 2**	**Round 3**	**Hidden Round**
Weightage	5%	15%	30%	50%

The full score of each round is 100. The score for the top 10 teams will decrease by 5 from 100 according to team's rank (i.e. the number 1 ranking team will be scored as 100, the second ranking team 95, etc.). The rest of the teams after the top 10 teams will receive a score of 50.