Chookon: A Data Science-Powered Software Platform for Mass Transit Railway Interim Report

Tam Ho Lun (UID: 3035571357)
Yuen Chun Hin Luigi (UID: 3035566467)
Lau Yi Lok Trevor (UID: 3035566857)
Lai Yu Ching (UID: 3035567722)
Kong Kwai Yee (UID: 3035566948)
Michael Tam (UID: 3035568568)
Feng Guo Ze (UID: 3035641528)
Supervisor: Prof. Cheng Reynold

January 23, 2022
Abstract

This interim report illustrates the details, methodologies, current progress and results of the Data Science for Mass Transit Railway Final Year Project. The Mass Transit Railway (MTR) is responsible for over 3.5 million journeys across Hong Kong per day. Despite being under such a constant heavy pressure, the MTR has served Hong Kong with consistent stability, maintaining its normal operations for 99.9% of the time. However, any delays due to various incidents are very costly to the Mass Transit Railway Corporation (MTRC), as it is penalised very heavily for each delay due to the heavy reliance upon the system. Therefore the aim of this project is to develop a simulation software that simulates train activities within the railway system, with a particular focus on incident response. The software will be made to simulate the real-life operations of the railway systems as best as possible, and will calculate responses towards the incidents. It would then be able to output the best course of action when dealing with an incident, which is heavily focused on minimising the delays caused by such incidents. Throughout this academic year, this group from the University of Hong Kong will be collaborating closely with representatives from the MTRC, in the hope of creating the best simulation software possible. As of now, the simulator is able to visualise train movement within the system, perform basic incident detection and flag any affected components. The next step would be to implement the machine learning models to enhance the simulation accuracy, as well as some fine tuning of the software.
Acknowledgements

We would like to express our deepest gratitude to our supervisor, Prof. Cheng Reynold, for his valued guidance and supervision since the inception stage of this project. We would also like to thank Ms. Wenya Sun, the research assistant of this project, for her support and patience on explaining everything to clear our doubts. Moreover, I would also want to thank Miss Mable Choi for teaching me technical writing. Lastly, we would like to thank the Mass Transit Railway Corporation Limited for granting us such a precious opportunity to work on this large-scale project.
## Contents

Abstract i  
Acknowledgements ii  
List of Figures v  
Abbreviations vi

### 1 Introduction
1.1 Background ........................................... 1  
1.2 Motivation ............................................ 1  
1.3 Objectives ............................................. 1  
1.4 Existing Materials .................................... 2  
1.4.1 The Visualizer ...................................... 2  
1.4.2 Data from the MTRC ................................. 2  
1.5 Scope of Work ........................................ 2  
1.5.1 Railway Line ...................................... 2  
1.5.2 Incident Type ...................................... 3  
1.5.3 Information Displayed .............................. 3  
1.6 Outline ............................................... 3

### 2 Methodology
2.1 Introduction ......................................... 3  
2.2 Data Analysis ........................................ 3  
2.2.1 Plain data analysis .................................... 4  
2.2.2 Aggregated data analysis ......................... 4  
2.3 Simulation Model ..................................... 4  
2.3.1 System Definition .................................... 4  
2.3.2 Model Formulation .................................. 5  
2.3.3 Model Translation .................................. 6  
2.3.3.1 Object-Oriented Programming ................. 6  
2.3.3.2 Good Software Engineering Practises .......... 7  
2.3.4 Building the Simulator ............................ 7  
2.3.4.1 Unity ........................................... 7  
2.3.4.2 Train Movement Logic ........................... 7  
2.3.5 Source Control Software ......................... 8  
2.3.6 Model Evaluation .................................. 9

### 3 Results and Discussion
3.1 Introduction ......................................... 9  
3.2 Data Analysis ........................................ 9  
3.2.1 Track Usage Distribution ......................... 10  
3.2.2 Time Series of Traffic Per Hour ................... 11  
3.2.3 Time Series of Traffic Per Date ................... 11
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Simulator development process flowchart</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>ER diagram of the system</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Flow diagram of simulation</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td>An example of the Unity 2D GUI</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Track usage frequency distribution</td>
<td>10</td>
</tr>
<tr>
<td>3.2</td>
<td>Filtered track usage frequency distribution</td>
<td>10</td>
</tr>
<tr>
<td>3.3</td>
<td>Time series of track traffic per hour</td>
<td>11</td>
</tr>
<tr>
<td>3.4</td>
<td>Time series of track traffic per date</td>
<td>12</td>
</tr>
<tr>
<td>3.5</td>
<td>Track occupied time against hour</td>
<td>12</td>
</tr>
<tr>
<td>3.6</td>
<td>Track occupied time against hour in service period</td>
<td>13</td>
</tr>
<tr>
<td>3.7</td>
<td>Snippets of the incident log</td>
<td>15</td>
</tr>
<tr>
<td>3.8</td>
<td>Screenshot of the simulation, showing affected trains</td>
<td>15</td>
</tr>
<tr>
<td>3.9</td>
<td>Diagram showing the train after being affected by an incident</td>
<td>16</td>
</tr>
<tr>
<td>3.10</td>
<td>Screenshot showing the difference between the simulated arrival time and the historical arrival time</td>
<td>17</td>
</tr>
</tbody>
</table>
Abbreviations

CSV Comma-separated Values. 1, 3, 4

GUI Graphical User Interface. 1, 7, 9

KTL Kwun Tong Line. 1–3

MSE Mean Squared Error. 1, 10

MTR Mass Transit Railway. i, 1–3, 11, 18, 19, 21

MTRC Mass Transit Railway Corporation. i, 1, 2, 9, 14, 18, 19

MTRCL MTR Corporation Limited. 1, 2

OOP Object-Oriented Programming. 1, 6

RS Rolling Stock. 1, 3, 4
1 Introduction

This section will introduce this collaborative project, by first presenting a short history and background of this three-year project, followed by the goal, purposes and some basic information of the corresponding field of work.

1.1 Background

Being the biggest public transport network located in Hong Kong, MTR helps over 4.6 million passengers finish their journey with an on-time rate of 99.9% every weekday. The company’s continuous investment in and maintenance of the railway system are the keys of its high quality service and excellent punctuality. Nevertheless, technical and mechanical errors still happen, and it depends on experienced staff to make decisions in services recovery. Further unfavourable delays may result from human errors and consistency. As a result, this project will analyse the massive data collected from real-life and simulate different situations in a virtual environment in order to provide a quick and economical way for finding the best standard procedures for corresponding emergencies.

1.2 Motivation

When incidents arise, the MTR system is reliant on senior engineers and train operators to command the trains based on their prior experiences. Nonetheless, human errors and inaccuracies may result in unsatisfactory and unwanted railway delays. These inaccuracies are to be expected, given how difficult it is for people to consume and process a vast amount of data in order to effectively identify the optimal solution.

On the other hand, data science and machine learning aid train operators in making decisions. Machines’ ability to process massive data sets enables them to carry out sophisticated computations and conduct systematic data analysis. By incorporating data science and machine learning into the MTR emergency response system, the likelihood of human errors and discrepancies can be lowered, hence increasing the system’s accuracy and efficiency during incidents.

1.3 Objectives

This team focuses on enhancing the simulation model to incorporate the effects of train incidents, building on the visualisation tool established in past years of this project. The objectives are as follows:

1. Simulate the impact of incidents on the train travel when no action is taken to address them.

2. Examine various remedies to the problems.

3. Determine the optimal solution that minimises service delay and the affected area.
The ultimate purpose of this project is to create an accurate and efficient software platform capable of supplanting the current methods used by MTR to address train problems.

1.4 Existing Materials

As this project has been ongoing for a few years, past collaborations with the MTRC have yielded some useful tools that can be well utilised in this project.

1.4.1 The Visualizer

There currently exists a functional visualizer that the team from the previous year have left behind. This tool, built using Unity, is able to successfully visualise a complete schedule that is given by the MTR. Users are able to view the entire system of trains operating concurrently, as the tool takes note of the train start times, arrival times, and stop durations to display the journey of each train with respect to the above information. However, it is only able to function upon viewing the train schedule data that the MTRC have provided, meaning it can only replay the operations of previous days. It lacks any actual ability to simulate or calculate the movements of any of the trains, and serves no predictive purpose.

1.4.2 Data from the MTRC

The MTRC have provided data from the past two years to aid the development of this simulator. These materials come in the form of Excel spreadsheets detailing the information of each track, including track name, track number, adjacent tracks, track coordinates, etc. There are also train schedules containing the start times, stop times, station wait times etc. A map of the KTL railway system was also given, which is what the simulator environment is supposed to be modelled after. This data will help in the construction of the environment, as well as assist in training the agents in the later phases of the project.

1.5 Scope of Work

This project focuses on simulating train activities, especially how the trains handle different incidents to minimise the delays. The scope of simulation is limited to the designated railway line and incident type.

1.5.1 Railway Line

As per the requirements of MTRCL, the simulator covers only Kwun Tong Line (KTL). It should be noted that the interceptions of KTL and other railway lines will not be taken into account in this project.
1.5.2 Incident Type

There are numerous sorts of train events, which can be classified into 19 categories, with 77% of them involving only one frequency. To ensure the simulator’s correctness, low-frequency classes with less than three incident occurrences will be removed from the data set. The most frequent class is Rolling Stock (RS)-related incidents (class “RS”), which account for around 60% of all occurrences. The team will begin by quantifying the fault descriptions of incidents within the RS class in order to design and fit the iterations into the simulator.

1.5.3 Information Displayed

On the simulation platform, the real-time train location and concurrent occurrence of incidents will be displayed. Because passenger flow data has no bearing on train movement, it will be omitted from the displayed statistics. The simulator will determine the best course of action to take in the event of an incident and will also illustrate the anticipated decrease in train delay time.

1.6 Outline

This report describes the background and current progress of the project. Section 2 illustrates the steps of developing the simulator:

1. Perform data analytics.
2. Build a logical module

Section 3 summarises the key findings and gives recommendations. Section 5 narrates the current progress of this project and indicates the next step of approaching the simulator development.

2 Methodology

2.1 Introduction

This section will briefly describe the platforms and technical concepts that will be utilised in the project, including the construction of the simulation environment, the agents, machine learning, and source control software. The testing methods that will be used to measure the simulator’s performance will also be discussed.

2.2 Data Analysis

The original data set provided by MTR is a set of 2 months’ signals generated by trains on the Kwun Tong Line. Through the hard work of the past few years’ students, the signals are converted into 62 Comma-separated Values (CSV) files, each file containing the train activities of each day from 1 Nov 2019 to 31 Dec 2019. There are 15 columns in each file, 9 out of the 15 columns were used in the
data analysis, including the train arrival time, departure time, track number, occupied time, station of the train, stop status, and the train id.

2.2.1 Plain data analysis

Plain data analysis is focused on analysing data without aggregation. After simple data cleaning, the distribution of different columns was plotted. Extreme values were excluded and the graphs were replotted. Relationship and interpretation of the plots were investigated.

2.2.2 Aggregated data analysis

The 62 CSV files are first grouped into one file. The duplicated train records are removed. Then, the dates were categorised into hours or dates. After that, the DataFrame is filtered so only the rows with the condition stop == True remain. Then, multiple group-bys were applied to the DataFrame so that the resulting DataFrame is the train arrival frequency by hour or date. Finally, these DataFrames are grouped by the sum and the frequency. The average train arrival frequency by hour or date is then calculated by dividing these two. Time series of data by hour or date were plotted and the outliers were investigated.

2.3 Simulation Model

This section describes the team’s approach to the whole simulator development process, from defining and designing the simulation model, to implementation and evaluation of the model.

Figure 2.1: Simulator development process flowchart

Figure 2.1 illustrates the flow of the iterative model development process. The process will begin with building a simplistic, naive simulator with very basic functionalities and non-complex logic, and then additional features will be added one by one to improve and refine the entire product.

2.3.1 System Definition

Simplification is first carried out. The complicated real-world railway network is simplified into a 2D not-to-scale map with only straight tracks perfectly aligned, which is provided. Afterwards, different assumptions are made. Factors affecting train punctuality that cannot be observed, such as track and RS maintenance
status and weather conditions, are assumed to be constant all the time. Apparently, the factors the team is focusing on are mainly traffic volumes. Since the information on the length of tracks cannot be obtained, the track layout map is assumed to be on scale.

![ER diagram of the system](image)

Figure 2.2: ER diagram of the system

Lastly, abstraction of the system is carried out. Figure 2.2 illustrates the components to be modelled.

**Trip** is a train object used in a single trip. States are defined as the instant track and speed of all operating trains in the system. Train movements on track, indicated by the actual arrival and departure time of each train at each track, are simulated. It is presented as the aggregated entity “train move on track” in the diagram as shown in Figure 2.2. Occurrence of an incident event may affect train movements in the following ways:

- Trips cancelled: Prevent initialization of trip objects.
- Trains cancelled: Prevent initialization of all trip objects with that specific train.
- Trains withdrawn: Change of destination and early disposal of trip objects.
- Trains changeover: Change of train in a trip.
- Journey delay: Change of arrival time and departure time of movement.

Performance measures to be analysed are how each incident affects train movements in the whole system.

### 2.3.2 Model Formulation

Figure 2.3 conceptualises and generalises the actual behaviour of the system.
Constraints in Figure 2.3 are defined as described below.

- $i$. operational timetable (departing station only) or train departure intervals.
- $s$. estimated running speed.
- $ds$. estimated speed in the degraded mode.

$i$, $s$, and $ds$ are varied depending on the system and environment with factors included but not limited to traffic volume, operational priority rules and temporary speed restrictions. At this stage, they are roughly estimated by simple methods such as taking the average. As the project progresses, tweaking and calibration will be carried out by taking these parameters into account. The green part is a simplification of the stochastic outcome of train incidents. At the current stage, a simple incident detection is implemented. As the project progresses, the stochastic outcome will depend on the incidents’ parameters. A decision flow of actions to be taken in response to the train incident will also be added right afterwards. The blue part will include the constraints that control the train movements. In Figure 2.3, only the implementation of blockage detection is demonstrated. As the project progresses, additional operational rules will be implemented.

2.3.3 Model Translation

2.3.3.1 Object-Oriented Programming

Object-Oriented Programming (OOP) is a form of computer programming that focuses on structuring software design around data, or objects, instead of functions and logic. Objects can be defined as a class of data attributes, states, and can have their own behaviour. OOP primarily focuses on the objects that developers would like to manipulate, instead of the flow of logic that operates on said objects.

This approach is fit for large, complex projects that require constant updates and maintenance, as it allows for easy identification of the parts that need changing. Another benefit of OOP is that it allows for easier collaboration between different developers. Such a method of organisation allows for the project to be divided into many smaller groups that each have their own functions and characteristics. Work delegation can then be arranged based on these groups, and different developers can work on different parts of the project simultaneously.
2.3.3.2 Good Software Engineering Practises  To prevent a large amount of bugs going undetected, and to minimise technical debt, a “Keep everything working constantly” approach towards coding will be taken, meaning that the product must be able to compile and run at each step of the process, from start to finish. At any stage of development, there will be constant testing and experimentation to check whether it runs properly. Whenever a change is made, any affected part should at least be tested briefly in order to ensure that this new change did not cause any malfunctions. This software engineering practice is very effective in preventing bugs in the sense that it can quickly help one narrow down the cause of a problem whenever a bug or inconsistency is discovered, as whatever new malfunction must have something to do with the change that was just made.

Despite the high efficiency in catching compilation errors that this approach offers, it is not intended to catch runtime errors that do not cause crashes, as those problems could have been caused by previous changes and were undiscovered back then. Depending on the level and intensity of testing each time, some relatively hidden and inconspicuous errors during runtime can be identified and resolved through comparing the results from different runs. However, overly extensive testing for every change would be way too time consuming, without any guarantee of catching any bugs. One should aim to strike a balance between, to catch the optimal number of errors without losing too much time to testing and quality assurance.

2.3.4 Building the Simulator

This section explains the tools and software required to build the simulator environment and functionality.

2.3.4.1 Unity  Unity, normally known for making games, is a great fit for the software to be used in this project. It is excellent in visualising 2D and 3D animations, its GUI is simple and intuitive, and it runs on the C# programming language. As the previous group’s visualizer was built using Unity, the simulator functionality and logic will also be implemented in Unity using C#.

2.3.4.2 Train Movement Logic  The implementation of train movement logic would ideally start from a simple, naive model, and slowly transform into a more complex model that can accurately simulate real-life train movement within the railway system. Trains will spawn periodically from the train stops, and begin travelling at constant speed with instant acceleration towards their destination, only stopping for a fixed amount of time in each station. To begin, a constant-speed movement model will be used, with a very basic speed formula: 

\[ v = \frac{d}{t} \]

Where \( v \), \( d \), and \( t \) stand for velocity, distance travelled, and time respectively. This formula allows for the calculation of the speed thresholds to satisfy any time requirements for the train’s arrival at specific destinations. Each train will be operating individually, without taking the locations and speeds of other
trains into account (at the initial stage only). When a crash is detected by two trains sharing the same track space, its crash boolean will be turned true, and the incident will be reported by printing a line to the console. However, no action will be taken, and the trains will resume their original intended journey as if nothing ever happened.

After the first stage implementation has been completed, more detail will be added to improve the simulator accuracy and realisticness in the later stages. For example, changing the constant-speed model to a more complex and realistic model. The trains would not have instant acceleration and deceleration, instead their speed and position would then be calculated using the formulas:

\[ v = u + at \]
\[ v^2 = u^2 + 2as \]

Where \( v, u, a, s, t \) stands for current velocity, initial velocity, acceleration, displacement, and time respectively. This time, trains would take acceleration and deceleration times into account, and cannot reach their top speeds instantly. Also, a simple collision aversion system will be implemented, where the trains will have to maintain a minimum distance from each other. When a train is approaching this minimum distance, the front train will attempt to speed up, and the back train will try to slow down.

Subsequent stages will further refine and improve upon the train movement with reasonable accuracy, and take incident response into account as well, which brings the discussion into the next phase.

2.3.5 Source Control Software

In this project, Git will be used for source control. Throughout the duration of this project, there will be numerous instances where multiple people are working on the same piece of code at the same time, and this would inevitably cause conflict between versions made by different members. To solve this, Git will be used. By monitoring each branch and version of the project, different code from different contributors can be gracefully merged into the same codebase, giving everyone a common ground to work upon. It also lessens the difficulty of any future debugging and helps everyone keep on track with the project.
To achieve this, GitHub will mainly be used for the main code origin/repository. One will be able to clone the repository and begin working on whatever changes that are needed. For better visualisation of the version flow and branch management, SourceTree will also be used to keep track of the whole project.

For any changes or updates made to the Unity project, its own in-built Git system, Unity Collaborate can be used. It is functionally identical to regular Git, just with a different GUI and a requirement for every collaborator to have a Unity account.

2.3.6 Model Evaluation

Validation of the model will be completed based on two components of the simulator. The first is the correctness of position and length of track compared to the track in real life. The second is the train is always on track and has correct real-time physics implemented, for example, when two trains crash, they should decelerate, stop, and report the incident to the simulator. The evaluation of the model accuracy will be carried out by tracking the mean squared error between the original data log provided by MTRC and the data log generated by the train object in the simulator. As the development process of the simulator is iterative, it is expected that the mean squared error will decrease sharply at the beginning and slowly when the simulator is more fine-tuned. One of the methods to improve the accuracy is the tweaking and calibration of values in Section 2.3.2. If the performance of the simulator can no longer be improved by continuing the iterative process, the simulator will be further improved using agent training, which is a whole new phase of the project, and will be explained in later sections.

3 Results and Discussion

3.1 Introduction

This section will discuss the current results of the project.

3.2 Data Analysis

To calibrate and measure the module, data from the MTRC and the simulator will be analysed and interpreted using Python. The spreadsheet containing all necessary track records will be filtered and converted to a Pandas DataFrame before being used with NumPy for further calculation. For further processing or delivery, aggregate statistics such as mean, median, mode, and variance of the indicated track’s occupied time, arrival time, and track number will be collected. The virtual record will be collected using the same application.

Track records collected by the MTRC and results generated by the simulator will be compared based on different factors such as MSE to evaluate the accuracy of
the simulator. After achieving a low MSE, RS incidents will be applied to the simulator randomly to see the difference between different results.

3.2.1 Track Usage Distribution

Figure 3.1: Track usage frequency distribution

Figure 3.1 shows the track usage frequency distribution. It can be seen that there are outliers around 0 to 10000 and they occupied 12% of the dataset. It is believed that these 12% tracks are used for repair or extra side tunnels, thus leading to the low usage and was not the focus of this study. The outliers were then excluded and the distribution was replotted.

Figure 3.2: Filtered track usage frequency distribution

From the replotted distribution as shown in Figure 3.2, it can be seen that the new distribution is similar to a normal distribution with a mean of 22000. The reason behind this could be the tracks with higher usage are the central traffic area or are the paths for trains to head back to the factory every day. On the other hand, tracks with lower usage may be the tracks in the station that have lower usage, such as the Whampoa station, which is skipped sometimes.
3.2.2 Time Series of Traffic Per Hour

Figure 3.3 shows the time series of track traffic per hour. Each line represents the tracks inside the station. It can be seen that there are two peaks and one trough. The two peaks are from 7 am to 9 am and from 7 pm to 8 pm, which is the time when people go to work or off work. The trough is from 1 am to 5 am, which is the time when MTR is closed. However, there is still some traffic activity, this may indicate the activities of trains that are undergoing testing or repair. Lastly, there is an outstanding purple line with lower usage than other lines, it is the track inside the Whampoa station. As mentioned above, sometimes the Whampoa station is skipped; it is estimated one out of two trains will skip Whampoa station based on the proportion of the usage in the graph.

3.2.3 Time Series of Traffic Per Date

Figure 3.4 shows the time series of track traffic per date. It can be seen that a lot of outliers are circled in red. The outliers represent some data that are missing. The two steep troughs are 19 Nov 2019 and 26 Dec 2019, after investigating the data, it was found that data on 19 Nov 2019 from 6 am to 11 pm are missing, and on 26 Dec 2019, only data at 11 am appeared. Other circled data are also believed to be missing as data are not in order, which is the opposite of the other normal data and some data seems to be duplicated as well. Thus, these incomplete data may not be included in the simulation.

3.2.4 Track Occupied Time

Figure 3.5 depicts the average track occupy time in Mong Kok station, which can be used to estimate the average speed of trains. From 1am to 5am, the
Figure 3.4: Time series of track traffic per date

Figure 3.5: Track occupied time against hour

graph’s pattern is irregular. As the time slot is non-service hour, the train operation is mainly used for routine train testing and maintenance. Figure 3.6 is the magnification of Figure 3.5 from 5am to 11pm (service hour).

Figure 3.6 demonstrates that a station has two distinct types of tracks: station tracks for allowing passengers to board and trains to stop, and travelling tracks for trains passing between two stops. As a result, the graphic depicts two distinct sorts of lines, one lighter and one deeper in hue. The former refers to
station tracks that are occupied for a longer period of time than other types of tracks, while the latter refers to moving tracks that are occupied for a shorter period of time. Additionally, two light blue circles depict the two peaks on the graph, from 6 am to 9 am and from 5 pm to 7 pm, showing the high passenger flow associated with going to and leaving work.

3.3 Details Regarding the Unity Simulator

3.3.1 Introduction

Details regarding the Unity simulator are discussed below.

3.3.2 Fixing the Bugs

Despite the previous team’s hard work, there were still several critical bugs that hinder the functionality of the simulator. Before proceeding to adding features and implementing the simulation, the team had decided to fix these bugs first to avoid technical debts. Below is one of the problems noticed and fixed by the team.

3.3.2.1 Unexpected Stopping of Trains

The first problem occurs when the game speed is increased to x20. According to the previous code, a train fetches for its next station by searching through the train log after arriving at a node. To detect whether a train has “arrived” at a node, the simulator calculated the difference in position between the train and its destination node. If the difference is lower than 0.01 units, the train is considered to have arrived.
This implementation works when the game speed is low as a train moves for a shorter distance each frame. However, if the game speed is increased, the number of pixels each train moved were also increased. Therefore, it was possible for a train to move past its target node without falling in the 0.01 unit range. As a result, many trains will suddenly stop moving when they arrive at a node.

To fix this problem, the team has decided to increase the detection range to a suitable value so that the train can consistently fall between in any game speed, while small enough to not cause a train to switch paths too early. After careful inspection and several trials, the team settled on a window of 0.5 units.

### 3.3.3 Incident Detection

To better simulate how the system handles incidents, a realistic set of incidents is needed, and should be inputted into the simulator during runtime. The incident logs provided by the MTRC have great reference value, as they contain real historical data about the incidents that occurred, including the date, time, affected trains, affected tracks, incident type, and incident description, etc. Therefore these logs were selected for incident detection within the simulation.

During simulation runtime, the software iteratively scans through the list of incidents, and takes note of the events that have occurred according to the incident time and the current simulator timestamp. The affected trains and tracks are then added to a stored list, which will be used for incident detection. This process is repeated periodically throughout runtime, and the lists are updated frequently in order to keep the simulation accurate and responsive. The two lists are then broadcasted to all the running trains, which will frequently check if their own train ID number is in the list, or if their current occupied track ID number is in the list. If either of the above is true, the train will be flagged as AFFECTED, and will proceed to the incident response mode.

### 3.3.4 Trains React When Involved in an Incident

When a train is flagged as AFFECTED, it can choose to perform one of the following actions:

1. Continue traveling its intended path as normal.
2. Slow down briefly, and reaccelerate to resume its original schedule.
3. Stop briefly, and reaccelerate to resume its original schedule.
4. Stop completely, and abandon its original schedule entirely.

Different actions taken towards the incident will result in different outcomes in regards to time spent on reaching each checkpoint destination. Comparisons will also be made between the simulated arrival times and the historical arrival times according to the schedule logs, which will be explained in Section 3.3.6.
Figure 3.7: Snippets of the incident log

Figure 3.8: Screenshot of the simulation, showing affected trains
3.3.5 Fetching the Destinations

The existing build completed by the previous team included a function to spawn trains and allowed each train object to fetch its next destination track node from the train log. However, the implementation is limited by Unity’s current game time. Therefore, the team has decided to change the mechanics of spawning trains as well as fetching the next node in order to better suit the simulation model.

Originally, the simulator would loop through the entire train log until the time field of an entry matches the current game time stored in the simulator. Then it will check whether its train name data is present in the simulator. If that specific train name is not found in the simulation, it interprets that as a new train and will spawn a new train. The new train object is created with a name, initial position and a first destination. However, the complete train path is not generated nor stored in any way. To look for its next target track, a train iterates through the train log starting from the current entry to the end of the log until an entry matches its train name. The first found entry will be stored in the train as its next station.

This approach could work when incidents are not present in the simulation because the train is moving according to the schedule. Any train will be fetching its correct next location as long as its movement does not deviate from the train log. However, for the simulator to function properly, trains should be able to identify themselves and perform actions not stated in the train log. Actions, such as slowing down and stopping, will cause the train to fetch for an incorrect next track as the train might be multiple tracks ahead according to the original schedule. This causes the train to skip tracks and travel in abnormal ways.

To account for this problem, the method a train fetches for its next destination has been changed. Instead of having to loop through the train log every time it arrives at a node, the entire train log is iterated only once when the train is generated. All its intermediate nodes and final destination are stored in a list of nodes. Whenever the train arrives at a node, the train can simply pop the first item of the list and keep its next destination as the first item of the list. This way, no matter how late a train arrives at a node, it can always look for the correct subsequent node. Moreover, one can expect this approach to be more
efficient due to the few times needed to loop through the train log.

The old worst case computational complexity is $O(mk^k)$, where $m$ is the number of trains, and $k$ is the number of lines in the log file, because each train loops over the log file every time it needs to find a new destination, and there are at most $k$ destinations for each train; while the new worst case computational complexity is $O(mk)$. This computational complexity could be further improved in the future to $O(k)$ by storing the scanned lines if such a performance boost is needed. The space complexity for both new cases is $O(k)$ as there are only $k$ lines in the log file.

3.3.6 Measuring the Punctuality

As each train’s journey shares the same set of destinations with their corresponding historical scheduled trips, a comparison can be made at each checkpoint destination. As the simulated journeys do not adhere to the arrival times in the logs and operate on a different speed, as well as various courses of action being taken during the simulation, there will be a difference between the simulated arrival times and the ones listed in the historical datasets. Therefore the time saved or lost can then be computed, and a punctuality comparison can be made whenever a train arrives at a station, so that the impact of each choice made can be estimated and known.

![Figure 3.10: Screenshot showing the difference between the simulated arrival time and the historical arrival time](image)

The results are currently based on arbitrary variables, most notably the speed of each train, as well as the stopping time at each station. The accelerative motion of the trains also requires refining, as they currently operate on a simple constant speed model. These factors are important in determining the accuracy of measurement, as the more realistic the variables, the higher the accuracy. As the program is still in its testing and developmental phases, results with high reference value cannot be guaranteed. However, the main purpose and functionality of the simulator is already in place, which is to measure how different conditions and events affect the punctuality of the trains. The next step is to
improve individual steps within the operational flow, so that the software will produce increasingly reliable results.

4 Challenges, Obstacles, and Limitations

4.1 Introduction
This section will discuss the challenges, obstacles, and limitations encountered in the project so far.

4.2 Project Direction Uncertainties
Despite this project being a collaboration between the team and the MTRC, a meeting between the two parties has yet to be held. Due to the nature of the working relationship, and the sparse availability of the MTR team, no clear channel of communication could be established. Without any direction from the MTRC, it was unclear to the team about what the precise requirements were, therefore various parts were done purely by intuition or some rough instructions and guidance given by the team’s supervisor. Certain doubts that the team may have about the project cannot be resolved satisfactorily, most notably the problem of track length, which will be explained in Section 4.3. Multiple minor issues arose during the development, such as whether the system map was proportional, and considerable amounts of time were spent on resolving these issues or finding workarounds. The lack of transparency from the MTRC due to corporate and safety reasons is understandable, yet it is nonetheless a major cause of confusion and uncertainty, which interfered greatly with the whole working process. It would be greatly beneficial if meetings with the MTRC team could be arranged in the future, so that the group may obtain a better understanding and clear direction towards the next step.

4.3 Uncertainties for Unit of Speed
In the data provided by the MTRC, the real-life dimensions of the tracks were not listed, making it extremely difficult for realistic, Newtonian-physics-based calculations to be made. This caused great uncertainty and confusion within the group, as it was previously expected that a kilometres-per-hour unit was to be used as the unit for the trains’ simulated speed, and would allow the trains to traverse the map in a realistic manner. It was originally planned that train arrival times would be obtained by a distance/speed formula, but the lack of track length data does not permit this.

It was suggested that time can be used as a representation of speed, where time would be directly used as a measurement of how long a train needs to complete a journey. However this would compromise the accuracy and realism of the simulator, as the time used is simply a regression from previous data, and no actual simulation would be done.
Another alternative was to use a percentage scaling to indicate the progress a train has made towards the next node. This method would only work if every track had identical lengths, as the same scaling on tracks with different lengths does not mean the same speed, therefore an unrealistic motion would be simulated and the results would be unreliable.

After some discussion with Professor Cheng, it was decided that the original approach with physics-based simulations would be taken, using arbitrarily defined track lengths for the time being, until further communication with the MTR team. Attempts to obtain track length data from the MTRC in future meetings will be made to enhance the current model. If it cannot be obtained, one of the aforementioned alternatives will be used.

4.4 Insufficient Data

Although a total of two months of train logs are given by the MTRC over the past two years, the amount of data is still far from enough. Partial of the data is unreliable due to the missing of special timeslot and all train logs are obtained in 2019. As a result, the evaluation process of the simulator is affected since the structure of the railway and the train schedule are changed. To obtain accurate simulator and reliable data analysis results, additional data in recent times will be requested. Also, the extra data help us understand incident cases and its consequences more so that a more realistic result can be created and evaluated.

4.5 Technical Limitations

As seen from the figures and the methodologies, this simulator operates in 2D, instead of 3D, unlike real-life railways. Due to the immense computational power required to run 3D simulations, and the limited time for development, the project was unable to be developed in 3D, and unable to support fully comprehensive physical calculations. This means that the simulation is not a complete imitation of its real-life counterpart, and cannot achieve perfect accuracy. To develop it in 2D is a reasonable compromise, as train movement along the railways is straightforward and leaves little room for variation. Therefore a 2D bird’s eye view approximation of the real-life scenario is perfectly acceptable, given the circumstances.

5 Future Plans

5.1 Introduction

This section will discuss the future plans of the project, notably detecting train collisions in Section 5.2, and measuring punctuality in Section 5.3.
5.2 Train Collision Detection

Allowing the trains to detect collisions with one another is part of the project’s future plans. When a train’s hitbox collides with the hitbox of another train, a crash event will be created, where the affected trains and tracks will be entered into the incident detection lists. Although this rarely happens in real life, the detection of trains crashing is important for the simulation, as it filters out the blatantly incorrect actions when a train is flagged, so that any future collisions do not happen. The implementation of this function can be done in the very near future, as its operations do not interfere with the existing functionalities, and is conceptually simple to implement. This will be the immediate next step after the writing of this interim report.

5.3 Further Punctuality Measurement

A function will be added to indicate the significance of the arrival time difference described in Section 3.3.6. If the simulated arrival time is one minute or above later than the historical arrival time, that specific arrival will be listed as LATE, and printed into the console log. If a train does not arrive within five minutes of its original scheduled arrival time, it will be flagged as SEVERELY LATE, and its previous actions will be recorded, and labelled as a cause for severe tardiness. In this case, it could be safely assumed that the train has encountered an accident, or passed through a problematic track. This function is very important, and one of the key points of this whole project. It allows the user to know the significance and impact of any previously chosen actions, and most importantly learn which ones to pick and which ones to avoid. As this is a tremendously important feature to have in the simulator, its implementation should be done as soon as possible. Given that it has no conflict with the development of the train collision detection feature, the two functions can be developed simultaneously as part of the immediate next step.
6 Conclusion

The simulator aims to be able to accurately simulate the situation within the railway system, and provide the best response suggestions as soon as possible. The project also hopes to foster a healthy and collaborative working relationship with the MTR company, and accomplish the overall goal of providing a safer and reliable public transport network for the people of Hong Kong.

Hopefully, by using the aforementioned methods and technology, the desired results can be achieved, and a satisfactory and impressive product can be produced at the end of this academic year. In the near future, there will hopefully be much more to see, and many more results to share with everybody.