Determining the viability of a non-stopping railway system on the UK Railway Network using computer-generated simulations.

UNIVERSITY OF
LINCOLN

Alexandre Pierre Barlaam
19695898

19695898@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of BSc(Hons) Computer Science

Supervisor Dr. Charles Fox

May 2022

Acknowledgements

Firstly, I would like to thank my supervisor, Charles Fox, for his support and knowledge which helped tremendously in the making of this project. I would also like to thank all my friends that have supported me and given me moral support during this projects and its ups and downs.

Abstract

This project aims to investigate the viability of a new prototype train system on the railway of the UK called a non-stop train system. It would do so using computer generated simulations in order to compare conventional train and non-stop trains. This report details the successful engineering of these simulations as well as the findings during their testing and their shortcomings. The conclusion stated that these types of trains could be used on some lines suited to them.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the first quarter of 2019-2020, before the wake of the pandemic, 439 million journeys were made on the railway across the UK. While the coronavirus pandemic made those figures substantially lower, (285 million in the third quarter of 2021-2022, figures courtesy of the ORR (Gower, 2022)), it is predicted that the railway will see strong post-pandemic growth."In Great Britain, passenger rail usage has seen growth across all measures since 2002" (Preston, Pritchard and Waterson, 2017). Pre-pandemic, the railway kept growing at a pace seen only in what many refer to as its golden age (see Fig. 1.1). Taking into account the trends shown and the possible growth in passenger numbers in the next decade, we could be faced with further overcrowding issues as seen before the pandemic. This would cause a huge congestion problem as more trains are put onto the network to mitigate this consequence. One solution would be to provide a railway that is more efficient allowing more trains to run and reducing congestion as a result. Another point of consideration is to build new lines employing new technologies to alleviate congestion. Casson, 2004 underlines that it "provides an opportunity to introduce advanced signalling systems and other technological features"

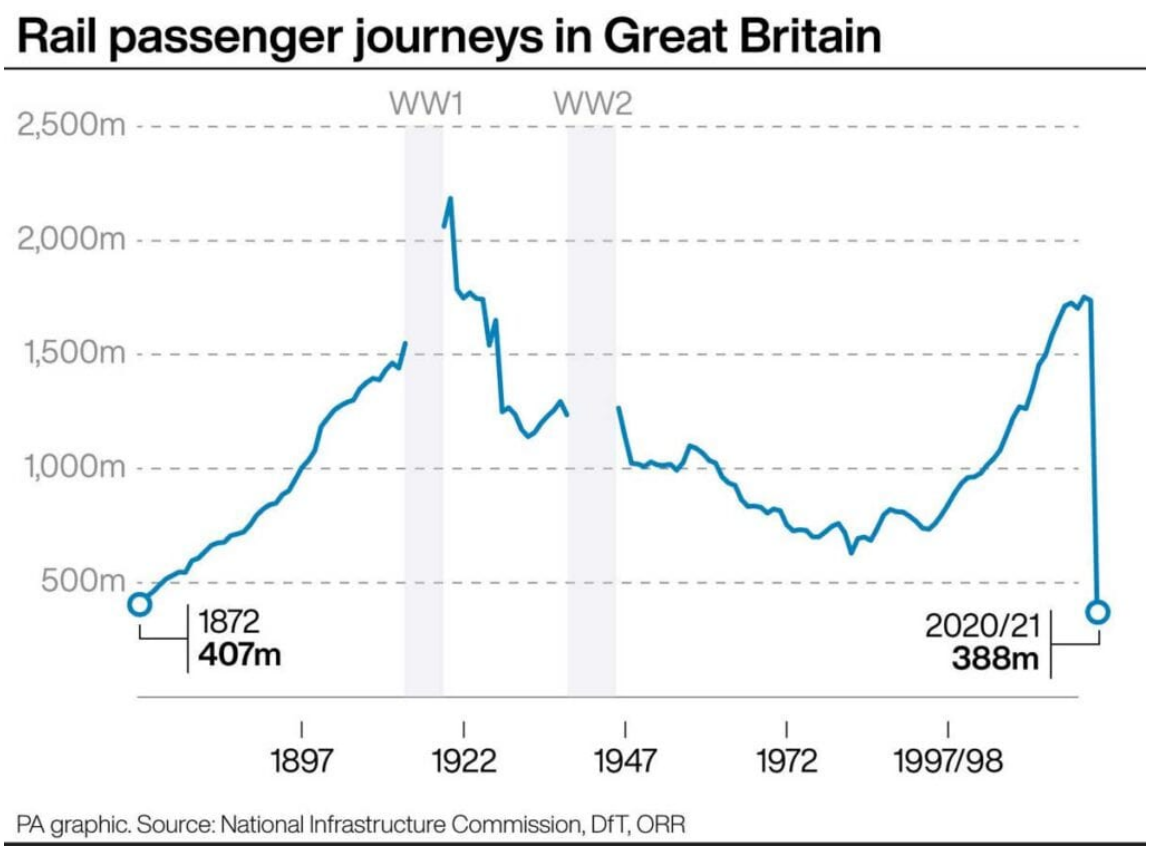

Figure 1.1: Railway Passenger journeys, 1872-2021

Another point of consideration here is the lack of investment seen on some regional lines in the UK reducing the quality of service compared to intercity routes. While there have been some improvements over the last 5 years, there is visible gaps in services. Intercity routes have seen a lot of investment in the last 20 years such as HS1 and more recently HS2, both being multi-million pound projects while some regional lines still run on the Victorian absolute block signalling system. After taking into account the points made, the following research question was considered: Could a non-stopping railway system bring improvements? and how much of an improvement could it bring?

This brings us onto the aim of the project: to build a railway simulation to simulate the running of trains on a network and assess the efficiency of a non-stopping system compared to a conventional stopping system. This aim can be summarised with the following research questions:

- Can a non-stopping system be implemented in the UK?
- How much extra efficiency can it bring compared to a conventional system and how many more trains would that mean?
- Is this a long enough lasting solution to the growth estimated?

To achieve the aim and answer the research questions above, there are some objectives to consider regarding software engineering. These are mentioned below:

- Choose a network to investigate and build its topology
- Build a simulation algorithm
- Insert trains and conventional timetables to run on the simulation
- Repeat previous step with non-stopping timetable
- Collect and analyse data to answer research questions
- If time allows, add more advanced features such as signalling and simulating a full timetable

This thesis also has some key milestones to mention, these were carried out as follows:

1. Investigate the project further through a literature review giving context and rationale.
2. Discuss and justify the project management methodologies, tools used throughout the project.
3. Discuss and justify language and software choices used for development
4. Document the design and development of the artefact as well as its operation
5. Draw conclusion from the artefact and answer the above research questions according to the data
6. Reflect and evaluate the development of the completed artefact.

Chapter 2

Literature Review

This section aims to provide rationale and context behind the project, background for the project to ease understanding of the concepts involved. It will also present, review, and evaluate literature in the project's domain of interest and inform the final outcomes of the project. Firstly, we will discuss context and rationale in the subsection below with the following section reviewing the literature relevant to the project.

2.1 Background

With the railway being as important as it is and with the growth it has seen over the last 20 years, the railway must continue to progress in efficiency. As stated by Armstrong and Preston, 2011, "Rail transport has significant advantages over more energy intensive modes of road and air". With the "dwindling reserves of fossil fuel" being more relevant than ever, the railway has the potential to be a dominant mode of transport in the future. Therefore, it is important to make the railway as competitive as it can be. This can be done by boosting the efficiency of the railway. As previously stated, this project's primary objective is to investigate non-stopping train systems for use on the rail network of the United Kingdom and to assess its efficiency. Providing the results are satisfactory, this could provide a possible boost to the railway's efficiency if it is implemented. This is important as a "driver of change" identified by Armstrong and Preston, 2011 is "Semi-autonomous/autonomous vehicles becoming safer and more efficient" Regional Lines have been chosen for the investigation in this project. There are 2 reasons for this. They are a great medium to test new tech-

nologies within the railway. Most new technologies that have proven to be effective will be deployed on such lines before trunk lines. This has been seen before with electrification for example with the Manchester, Sheffield and Wath Electric railway being the first in Britain. Discussing this however would fall outside the scope of the project. Some regional lines also have seen a lack of investment compared to trunk routes meaning that the quality of service can vary and fall short in respects. One reason for this can be attributed to the infrastructure. "Speed Capability of the new rolling stock is under-utilised due to infrastructure quality" states Tyrrall, 2004. While infrastructure may have improved since 2004, it is clear that new technologies would nevertheless bring improvements. Another issue worth stating is overcrowding as stated by Wellings, 2014 , "increased overcrowding, which clearly represents a deterioration in quality, has been the other accommodating factor" for "an increase in services". While it also states that the problem has been alleviated, we can expect overcrowding to be relevant again as passenger journeys increase further.

2.2 Related Literature

With the project undertaken being specific and niche, relevant literature can be difficult to find especially accounting for it being rooted somewhere between computer science and civil engineering. Despite this, several papers of various relevancy have been chosen to serve as literature:

Two papers have been used to gain background information on the state of the railway over the past 20 years and the gaps that the railway performance may have. This has been discussed extensively in the previous sub-section. These two papers are Tyrrall, 2004 and Wellings, 2014. They both assess the railway model used in the UK that is a privatized railway. Previously the railway was run by the government controlled British Rail (BR). During the 1990s, the John Major directed conservative government privatized the railway, a system which albeit revised is still used today. These papers provide knowledge useful to assess which direction the project should follow to be as successful as possible in filling gaps within the current railway and promote progress within it. It is, however, important to notice that one paper was

written 18 years ago and therefore is not as relevant as it once was. This must be taken into consideration when reading this paper and basing the project direction on it.

Furthermore, this paper Armstrong and Preston, 2011 highlights the possible "future roles for the railway", this is especially relevant in our case. The given scenarios here could help us see how the railway the trends of change in the railway in the coming years. This is especially useful in this project as this will ensure that the investigation made is relevant to the railway of the future. As discussed previously, the paper highlights the need for more efficiency which is one of the main objectives of this project. Indeed, this project final outcome is to offer a more efficient mean for passenger to alight and board from train to platform. It is also worth noting that the scenarios presented are only scenarios and may not be entirely realistic or may not be an accurate depiction of the future of rail transport. This must be kept in mind. The growth in passenger numbers is also highlighted in the paper. The railway will also need to be able to cater for this increase by an increase in capacity. This could be a target, showing the maximum theoretical capacity of a line within a non-stopping system.

Another important point to consider in a railway related project is signalling, this is an aspect that is very well covered by academic literature. The first paper worth mentioning here is Pearson, 1973 This paper examines the performance of multiple signalling systems namely "four and five aspect fixed block signalling, theorical pure moving block signalling and quantised moving block signalling". While this paper is very in depth and quite long, it most notably provides some very useful equations, such as, for the block length. This is important to consider as it defines capacity of a line because only one train can be in one block at any time. The equations is shown below:

$$BM = \frac{VM^2}{(4 * B)} \tag{2.1}$$

The above Equation calculates the block length where VM is the maximum line speed and B is the minimum breaking rate of trains. It is only valid for fixed block

signalling. This is the preferred method of signalling on the railway today. While the paper also discusses moving block signalling, this type of signalling has not been implemented in the UK. Perhaps this could form another target as it could help increase capacity on a line which could improve efficiency. While this paper is very valuable, it has been written 50 years ago making some of the concepts outdated. It would therefore be useful to have another paper that is more up to date on signalling principles. It also mentions 5-aspect signalling, this is irrelevant as this is not used on the network in the UK.

A more up to date paper that can be used is Pachl, 2020. This paper proposes a broader view of signalling principles instead of focusing on specific signalling systems. While the previous paper takes an evaluative approach, this paper is much more descriptive. This is very useful in understanding signalling principles; however, it only describes fixed block principles potentially limiting its usefulness. It is also much more up to date being written in 2021 therefore describing much more up to date principles which is much more valuable.

Lastly, another time of signalling system to consider is absolute block. A Victorian system still used on some lines today. A paper that describes this type of signalling is Ireland, n.d. It is however quite brief and while it covers most principles, it omits some aspects of the absolute block system such as starter, home and distant signals limiting its usefulness. This is unfortunate as it is one of the only academic papers available for this area of signalling. This means that gaps could be present in the required knowledge here however, this could be avoided by not using routes equipped with absolute block in the investigation.

Similar Project have also been undertaken before, one such project is a movable platform concept explored inGąska, Margielewicz and Pypno, 2011. This paper describes the use of moving platforms allowing the train to slow down instead of stopping to pick up passengers. The speed of the platform and train would be identical allowing passengers to board and alight. The paper also describes the energy consumption involved by such a system. It gives a good evaluation of such a system and considers the opinions of the passengers using such platform. This

system is not far off the chosen system for this investigation explained here Roktop and Yu-lun, 2008. While the movable platform system presented shares similarity to our chosen system, it is not as efficient. The train still must slow down substantially to allow passengers to board and alight. Our system allows the train to remain at full speed using a separate boarding "car" that is left behind at each stop and picked up by the next train.

The last paper to be reviewed is a paper specifying an OOP solution for optimal train control Takagi et al., 2006. This paper outlines the “Object-Oriented Multi-Train Simulator” (OOMTS) program developed by the University of Birmingham. This algorithm is of similar structure to the algorithm in development for this investigation. It is however much more refined and complex than that of ours. Nevertheless, it provides a useful reference point for development and shows that such an algorithm can be used for research on the railway. While the paper looks at a different problem, it is still relevant to us as it solves its problem in a similar way by the use of programming and OOP programming techniques.

Chapter 3

Methodology

3.1 Project Management

Our project management methodologies should follow from an identification of the characteristics of our project, as well as the requirements we expect from it. With this being identified, we can select the mean of project management that best suits our project and its demands. This would benefit the project hugely as it would ensure that the project is delivered on time and in a successful manner.

3.1.1 Project Characteristics

When choosing a mean of project management, it is important to consider the characteristics and nature of the project we are working with. In our case, we can consider this project to be quite unusual in many ways:

Most project in the field of computer science tend to be team project, however this is an individual project. This means that only one person would be managing and developing the project. Arguably this means the the project management methods employed have to take this into account and be appropriate for this type of project.

Additionally, there is no direct costumer for this project as it is geared towards investigating a possible future technology. One could argue that an indirect costumer is Network Rail (the management body of the railway in the UK). While this is true, the requirements are already set and won't change throughout the project to satisfy a customer even an indirect one. Moreover, this project has a fixed schedule defined

by the deadline imposed giving us a set amount of time to work with that isn't prone to change.

As stated above, the nature of the project is also an important factor to consider. Simulations require a specific framework for development as they do not follow the trend of other software: They are not trying to provide a solution to a problem in themselves. Rather, their existence is based solely on simulating a real-life event or occurrence. As such, this may have an impact on the type of project management structure that is used. Moreover, they require careful development before any development start. This is to ensure that they have the correct structure and methodology to effectively simulate a particular part of the railway system.

3.1.2 Project Management methods

At the beginning, a schedule was made for the entire project. This chart not only included the development part but also the planning and other assignments related to the project. The development phases are quite vague here too as this was done before the planning of the artefact and as a result, fairly little was known about the implications of building it. The Gantt chart made to this effect can be seen below in Fig 3.1.

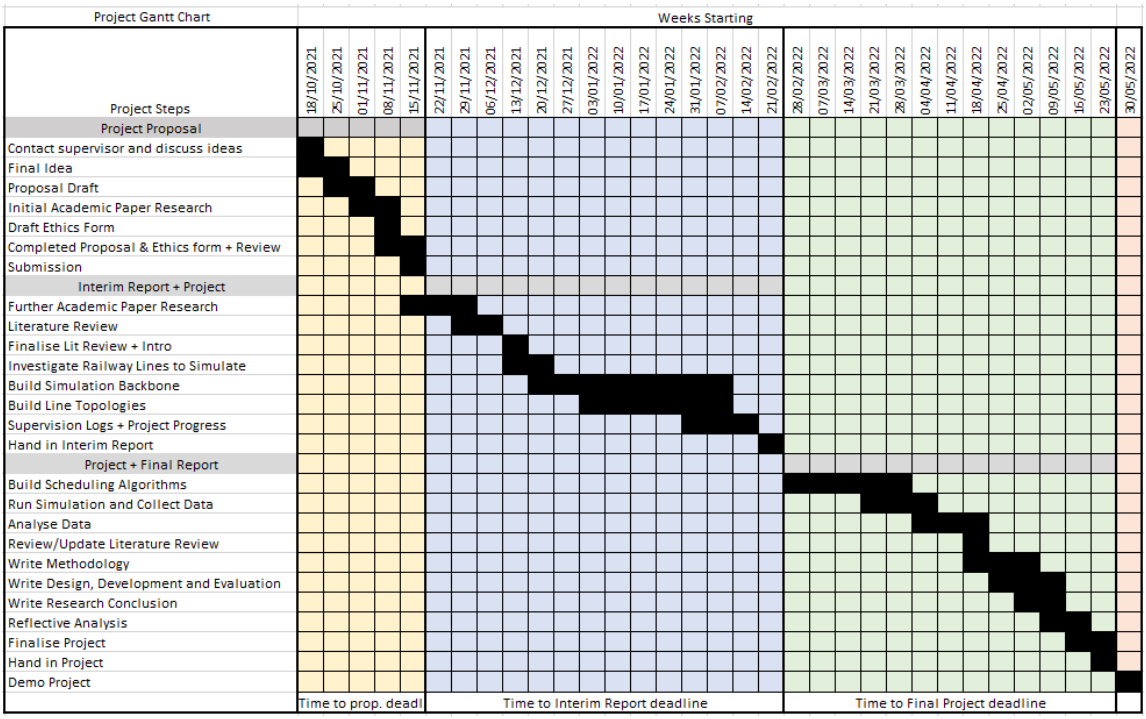

Figure 3.1: This figure shows the Gantt chart created for this project

Once the planning was done however and the artefact was investigated further. Another chart was made to in order to manage tasks and development phases that were needed to achieve the artefact. As time was already discussed in the Gantt chart, it was not felt that another chart discussing time was needed. This chart was chosen to be a PERT chart (Fig 3.2) instead. This chart was based on a sequential framework as we had identified the use of the Waterfall framework and used colours to identify tasks that were either to-do, in-progress or completed. Towards the end of the project, it was decided to not do some tasks as the project had overrun slightly. These tasks have been left in blue to show that they have not been completed.

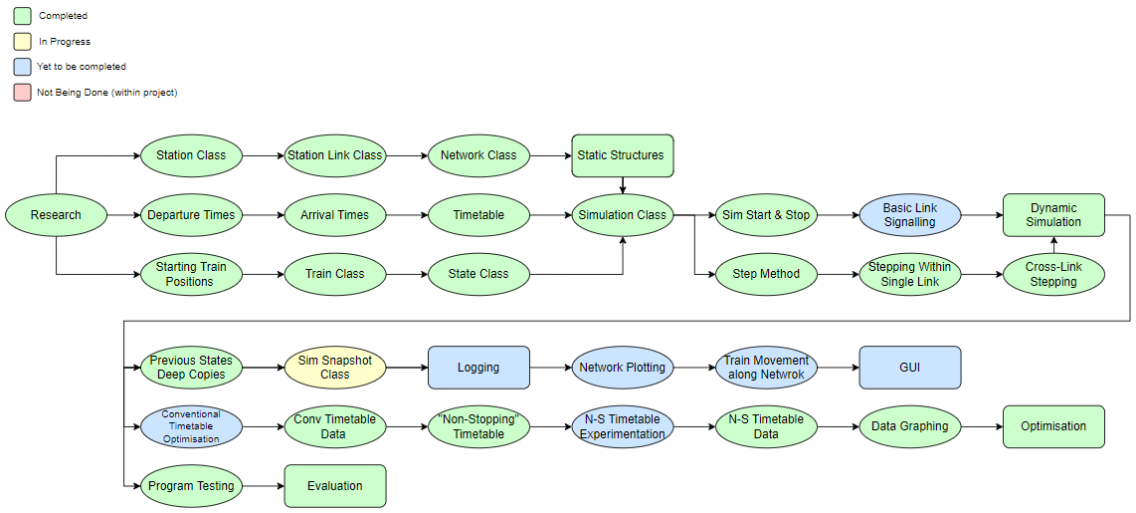

Figure 3.2: This figure shows the PERT chart created for this project

Additionally, there was fortnightly project supervisor meetings. This allowed the progress on the project to be presented and tracked as well as to set targets for the next 2 weeks. the PERT chart was also presented to see which tasks had been completed and which tasks required attention. This helped the project keep on track as much as possible as well as keeping the project well-managed.

3.2 Software Development & Management Approaches

3.2.1 Requirements

In the previous section, we identified characteristics of our project. Despite them being in the context of project management, we can identify requirements that we should follow to adequately choose a method from them. These requirements are listed below:

1. requirement of extensive planning beforehand
2. suitable for an individual project
3. suitable for a small, low-cost project
4. early delivery due to fixed deadline

3.2.2 Model Evaluation

Stemming from the requirements set out, a few project management and software development methodologies were considered and their suitability for the project evaluated:

Firstly, a few agile approaches were considered. The first of those was the Kaban approach. Andrei et al., 2019 states that the aim of this approach is "to maximize efficiency and collaborative teamwork across the whole team" and to "remove any “bottlenecks” from a streamlined process". While most of the interest in this approach came from maximising efficiency to cater for early delivery, one cannot deny that this approach is not suitable for this project in many ways. It is clear that this approach is aimed at a team. However, this project is an individual project meaning there is no collaboration to maximise. Furthermore, it does not satisfy a number of the other requirements that we have set out. Notably, each task is planned and discussed before it is implemented in this approach, leading to a more unit based approach rather than the pre-development planning that we have opted for.

Another approach that was considered is the Scrum approach. it provides "flexibility to control and manage requirements" (Hayat et al., 2019) which in our case is not needed as we are not dealing with a costumer, our goals are unlikely to change but however we could still benefit from this. Scrum also tends to have roles such as scrum master, scrum team and product owner. In this project, those would all be managed by the same person. Indeed, Hayat et al., 2019 mentions that "Scrum concentrates on teamwork" which wouldn’t be taken advantage of in this application. Additionally, scrum does not also fulfill the requirement of having planning beforehand as it utilises a "iterative and incremental base model"(Hayat et al., 2019). Each step is a unit and is planned beforehand rather than planning it all at the beginning. From the two agile approaches that we have considered, it is clear that these are not suited to our requirements as they tend not to have planning beforehand but rather plan as you go. This isn’t the approach that we are looking for.

The final approach that was considered and eventually selected was the Waterfall approach. This approaches favors a design stage where the software is fully planned

before the development stage. A. A. Adenowo and B. A. Adenowo, 2013 defines it as a "static model" that "approaches systems development in a linear and sequential manner". This falls more in-line with the requirements previously set-out. This makes it much less flexible, however, than both of the previous approaches considered, however that is unlikely to have a impact in this project. this is mostly because "customers can change their opinion towards different features"(Andrei et al., 2019) requiring flexibility. This project having no customers makes this risk much less plausible and should not result in possible delays. Andrei et al., 2019 also states that "Waterfall will be a better solution for small projects that have well-defined requirements that will not change". This again satisfies the requirements previously set out as our project can be considered fairly small and low-cost as well as having fixed requirements as discussed previously. Additionally this project has a very fixed deadline. This approach would also allow to plan towards that deadline ahead to make sure that it is met promptly while still retaining a good standard of production.

While waterfall was adopted as a framework, it is worth noting that some of the principles of it have been adapted as they were not fit for the project as they were. For instance, in traditional waterfall, "Once a phase is completed, there is no room to revisit it" (A. A. Adenowo and B. A. Adenowo, 2013). To over some flexibility, this was somewhat overlooked. Some tasks were revisited as the artefact was tested to make sure that it worked as intended and that the simulation was as accurate as possible.

3.3 Toolsets and Machine Environments

To ensure the delivery of this project, there several toolsets and machine environments to consider. With the artefact being programming based, we will of course require a programming language as well as an IDE to develop, debug and test it. Additionally, we will require a back-up tool and relevant libraries to aid functionality.

3.3.1 Programming Language

There was several language that were considered for this project, These were Python, C# and C++. Python is a very readable and very powerful OOP language. As well as that, it has a lot of very useful libraries which could be used. On the other hand C# and C++ are also very powerful strongly typed OOP languages. Notably C++ allows for the creation of pointers which could be very handy for our application. The C languages however do not have the extensive libraries that exist for Python. Python is also dynamically typed unlike the C languages making very quick to program which would be ideal for our application. Overall python was chosen for this project for the extensive libraries such as the copy library as well as graphing libraries to plot data such as matplotlib and for its simplicity making it a language that is quick to program in.

3.3.2 IDE

There was 3 consideration when adopting an IDE, these were Visual Studio Code, IDLE and Pycharm. While Pycharm is a very good IDE, it is considered somewhat slow compared to VSCode. VSCode is also much more expendable with endless extensions available for different functionality making it easier to develop. As well as that, they both have very good debuggers which is a very important factors of consideration when making a simulation. They also both offer line counting which makes it easier to identify parts of the code. IDLE is only a very basic editor and does not have any debugger meaning that it was ruled out very quickly. VSCode was selected as the editor due to the wide variety of extensions available compared to Pycharm as well as the added speed that VSCode has giving it the edge.

3.3.3 Back-up Tool

For the back-up tool is this project, there were three options that were considered: GitHub, Google Drive and a local backup. All three are very easy to use: Github has the Github desktop app which features a very user-friendly UI. Google Drive and local back-ups are also very easy to use as they feature a drag and drop style

interface. Git also has a CLI which is a very powerful tool if used correctly however this takes a long time to master. Even the Github app has many features which can be overwhelming and hard to learn. Nevertheless, Git has very good version control by using commits, pulls and reverts meaning that you can effectively backup the artefact and revert your mistakes should they occur. On the other hand, Google Drive does not have such functionalities as version control or the branch system that github has. There is also a cap on how much can be stored on a google drive account while Github can host unlimited numbers of repositories. While a local back-up is simple and useful, it is by far the least flexible option as it does not offer the cross-machine accessibility that the cloud based approach do. It is also very unsafe as drives can fail and files can be deleted or corrupted.

For this project, a Github repository will be used as it is the option that gives the best service and is also industry standard. We will be able to make effective use of its version control to track progress of our artefact and revert any changes should mistake happen. As well as that, it will allow us to work remotely on different machines. While the CLI is a good option, we will make use of Github desktop as it has lass of a learning curve whihc will benefit us considering the early delivery model.

Chapter 4

Design, Development and Evaluation

4.1 Requirements elicitation, collection and analysis

The idea for this project came about with the discovery of a concept non-stopping system that has been invented in Taiwan. It consists of letting passengers off in a different way to a conventional train, by way of a "boarding car that is capable of attaching itself and detaching itself from the train" (Roktop and Yu-lun, 2008). This means that the train does not have to stop to let passengers off and can continue moving at full speed while doing so. It would leave the boarding car behind when it reaches a station with the passengers wishing to get off on it. The boarding car would remain at the station until the next train reaches the station. The boarding car would attach to that train and would travel to the next station for the same thing to repeat until the terminus is reached.

The artefact built for this project simulates the time gain that would be attained by using such a system. This is achieved by having both types of trains being simulated in the same environment in order to compare them and evaluate the theoretical non-stopping train against a conventional train that is required to stop at stations along the routes. The artefact therefore must be able to effectively answer the research questions by effectively being able to simulate both types of trains.

It was also decided quite early on to target regional routes in this project. There was a few reasons for this choice. Firstly, there has been a notable lack of investment in the UK on regional lines in the last 20 years as discussed in the introduction. They were therefore targeted by this project to improve the services along these lines. The service offered can sometimes leave something to be desired especially when compared to services in mainland Europe. One could therefore argue that something is needed to revitalise some regional lines that have been lacking investment. Furthermore, regional as well as commuter routes would benefit the most from such a design. This is because they often have many stops lengthening journey time significantly when using conventional trains.

A number of railway lines were considered to undertake the study on. These included a lot of the rural lines in Cambridgeshire and some in Lincolnshire such as the Lincoln to Peterborough Line, abbreviated "LP Line", Peterborough to Ely line, abbreviated "PE line", and the Fen Line (Cambridge to King's-Lynn). The LP Line and PE Line were considered to have too few intermediate stations, 4 stations and 3 stations respectively, to give an accurate representation of the benefits of the concept studied. The Fen Line in this respect, was better as it has 6 intermediate stations. Moreover, the Fen line can be classified as both a regional line and a commuter line to Cambridge and London whilst the other lines considered had more of a regional role. Due to this, the Fen line from Cambridge to King's Lynn was selected as the line for this study.

4.2 Design

When building a simulation, it is important to consider the reasons for developing that simulation and the problem that it resolves. This ensures that it provides the accurate level of simulation that is required to effectively answer the research questions set out in the introduction. We can draw the conclusion from them, that this is primarily an efficiency problem as we are attempting to investigate the extra improvements a non-stopping system could bring to the railway system.

From this we can deduce that the primary factor that we must consider is getting an

accurate sense of the movements of trains. Ho et al., 2002 states that train movement is the "calculation of the speed and when a train is traveling from one point to another". For this, it was established that we must keep track of three things: time, position and speed. This was the most important goal of the simulation. Another thing to consider is signalling as this impacts the speed of the trains: they may have to slow down for a warning signal or stop entirely for a danger signal as passing a signal at danger is prohibited. While this is an integral part of the railway, from a purely theoretical standpoint, it is not essential to implement a full signalling system in this case as the project takes an approach more focused on the theoretical comparison of two railway models rather than accurately modelling the functioning of the railway. The decision was taken to implement a basic simplified form of link signalling providing that the core functionality were achieved first in an effort to save time. This was in the form of having entire links as blocks and setting them to be occupied or free based in the presence of a train on that link.

The next step taken in designing was choosing a structure for the development of the simulation. The literature studied has employed OOP structures exclusively. This structure has been successfully applied in multiple artefacts. One of those is the Object-Oriented Multi-Train Simulator (OOMTS) from the University of Birmingham used in Takagi et al., 2006. Other Simulators have also employed a similar structure such as Ho et al., 2002 and Sin and Goodm, 1994. This suggests that this is the best structure to use when building a simulation and was selected to be used to build the artefact. There was little evidence of any other structure that can be used. It would not be practical nor effective to build a simulation using a functional or logical approach. Objects are needed in order to be able to effectively model a rail network and trains while being able to interact with them. This would not be possible using the aforementioned approaches.

Having established a development structure at this point, we were able to create a specification for the classes we would need. Previously, we identified the requirement to keep track of several data pieces. These included time, position and speed. There are two possibilities for storing this data: either as an attribute of a class or as a standalone class. There are different reasons for choosing either. Storing it as a

standalone class allows us to keep the data organised into individual entities. This is very useful as it allows classes to be kept to doing one tasks and do it well. This makes debugging easier and keeps the code tidy. This is relevant for larger pieces of data such as the position of a train. However, the speed and time can be built as an attribute of a class as it would be inefficient to build them as their own class and would over complicate things.The table on the next page illustrates the decisions taken

Table 4.1: This table describes the decisions taken regarding data storage frameworks

Data	**Storage Method**	**Data Type**	**Class**
Train Current Speed	Class Attribute	Integer	State
Train Max Speed	Class Attribute	Integer	Train
Train Position	Standalone Class	N/A	Position
Train Direction	Class Attribute	string	Train
Simulation Timetable	Standalone Class	N/A	Timetable
Network Station	Standalone Class	N/A	Station
Network Links	Standalone Class	N/A	StationLink
Network Topology	Standalone Class	N/A	Network

As can be noted from the table, this project while require a fair amount of data storage, some data was stored inside standalone classes. This was usually large data pieces that required multiple attributes. Train Position especially was required to be dynamic so it could be updated as the positions of the trains changed. Therefore it had to be stored in a standalone class with methods allowing it to be updated accordingly. Other data using this approach were the timetable, stations, links and the network topology. This was done in this manner mainly for practicality and ease of maintenance and building. Other smaller pieces of data were stored as attributes of classes. This is notably the case for the current speed and max speed of the trains in the simulation and the train direction. While we chose to store the data for everything directly inside of the artefact as was done in the OOMTS (Takagi et al., 2006). There are also alternative ways to store data. Sin and Goodm, 1994 takes an innovative approach of storing the dynamic information inside of the program as was done in our artefact however it differs in storing the "static train information such as train length and train class number" in an external database managed by a "rolling stock manager". While this is a sensible approach of there is a lot of static

data to handle, it was deemed unnecessary in our application as there is not enough data to justify this extra step. It is simply easier to store it directly in the source code in the relevant class.

It was deemed necessary at this point to discuss the workings of the simulation. A single class was decided upon to run the simulation. This class would take all of the data that it needs from other classes including data about the trains: their maximum speed and position and data from the network: stations, links and overall network topology. It was chosen to use the metric system to measure distance. This is because it is the standard measure of distance in European investigations and therefore makes the investigation relevant to the rest of Europe as well as the UK. Additionally, It was decided to employ a second based time keeping: Every tick of time would be one second. This would mean that we would be using meters per second (m/s) to measure speed. This is an effective design as it means that we can simply add the speed of the train in meters to the distance already travelled for every tick of time in the simulation. This keeps the algorithm simple and saves us from having many equations to calculate different aspects of it which could lead to many bugs and reduce ease of programming and maintainability. Moreover, it is important to clear one thing up here. Every numerical value in the simulation will be and integer. This is for two reasons. This level of accuracy is not required within the simulation as the difference between the two types of trains will be much greater than decimals. Furthermore, using floats could bring inaccuracies to the simulation and could lead to bugs when values are compared. This could be mitigated by using doubles which are more accurate and rounding them when comparing them but as stated previously, this level of accuracy will not be required in the scope of this investigation.

It was envisaged that the Simulation class would require the following methods in order to run: "startSim" and "Step":

"startSim" as its name suggests would begin the simulation and run it. Additionally it would be responsible for ticking the time after the trains change positions in the previous tick and copying the last state before changing the time. The latter is

important is this context was it allows us to keep a trace of where the trains were at a point in time and being able to do so for any point of time. This will be handled by the "SimulationSnapshot" class and will be discussed later in this section. It is important to stress that the copy should also be a deep copy as opposed to a shallow copy. This is because a shallow copy is merely just a reference to the contents copied. Therefore the changes to the original area of memory (A) will be reflected in the area of memory of the copy (B) as they reference each other. In a deep copy however, that is not the case as A and B are completely independent from each other and changes made in one will not be reflected in the other as the content of A is copied to a completely different memory area in B and they do NOT reference each other as they would in a shallow copy. After the previous step having been taken, this method would call the "Step" method.

The step method, again as its name suggests, handles the "stepping" of the trains, that is, their movement. In each time tick, it would check the state of each train and react accordingly to it: if the train is stopped at the station and the timetable and current time match, it changes the state of the train to move. However, if the train is already moving it would only add the speed in meters to the distance of the train along the link. Lastly if the train distance along the link is bigger than the length of the link then the train stop at the station. The pseudo-code below details the workings of these two methods.

Upon Further investigation, additional methods in the "Simulation" class are required. Notably we identified the need for a while loop in the pseudo-code. We therefore need a way for it to halt. This is a very important fact to consider as there is a real risk when using a while loop that it will not halt and that the program would never terminate. To mitigate this risk, we will require a method to terminate the while loop when it is called by setting the boolean attribute "simRun" to false which will halt the loop. This method will be called when the simulation reaches a time allowing us to run to for a set amount of time. This poses a risk however if the wrong increment value is used. This presents a very low risk however and does not need to be mitigated, it is however something to be aware of.

Moreover, we will also require a separate method to find the next link when the train departs a station. Originally it was envisaged to use a single method to handle both the up and down direction, (The meaning of up and down will be discussed at the end of this section). Upon testing this however, it was found that this was not an adequate mean of doing this. Several bugs were found subsequently leading to some trains being along the wrong link and going back on themselves particularly along the single line sections of the network. It was decided to this part of the simulation being re-engineered midway through development. We opted to separate the method into two. These would be named "NextLinkUp" and "NextLinkDown" handling the up and down direction respectively.

As Discussed earlier, we also require the class "SimulationSnapshot". This class would take the deep copies of the states from "Simulation". It would do with the "getStates" method from that class and store it as an attribute. Another method is required to print the states. It would take in the chosen time of type integer as an argument and print the state corresponding to the given time argument

When all of the class and their relevant function had been defined, a table was made to compile all of this information in a single place this can be seen in fig . Furthermore a entity-relationship chart was also made. This specifies the relationships between all of the different classes in the way they feed data to each other. This can be seen in fig.

Table 4.2: This table describes each of the classes' functions and

Class	**Purpose**	**Data only?**	**Methods**
Network	Network topology	Yes	No
Position	Train Position Data	Yes	Yes
Simulation	Simulation Running	No	Yes
SimulationSnapshot	State Checking	No	Yes
State	Simulation State Data	Yes	No
Station	Network Stations	Yes	Yes
StationLink	Inter Station Links	Yes	No
Timetable	Train timetable Data	Yes	No
Train	Train Data	Yes	No

As shown in the relationship entity chart on fig, there is some degree of data redundancy. The links are defined and also stored as a attribute in the instances of

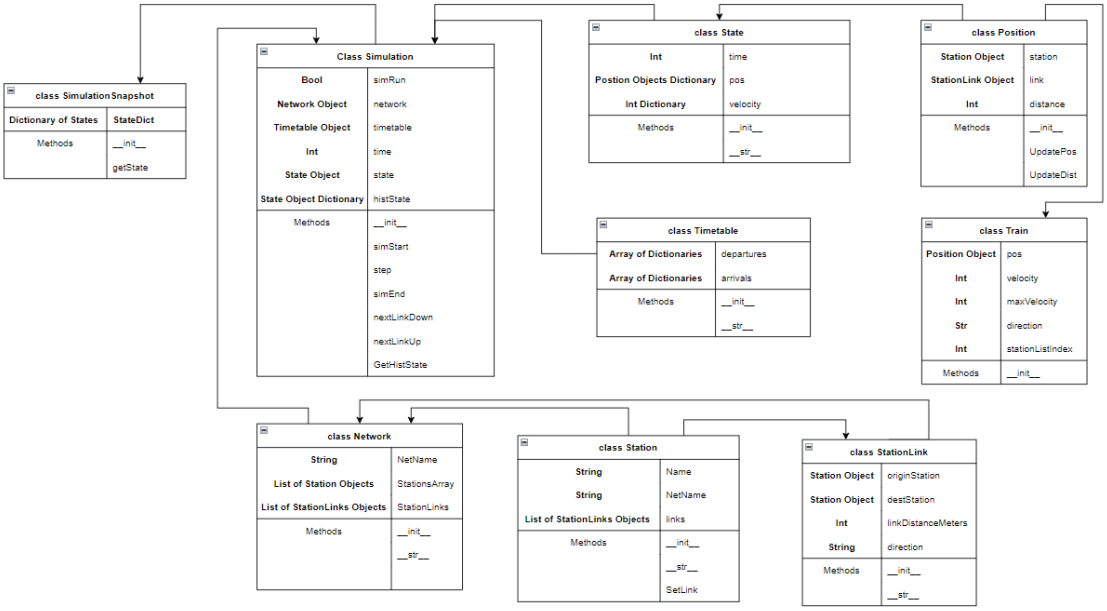

Figure 4.1: Entity Relationship Chart Detailing

the Station class. Normally this would be considered simply as memory bloat and would be considered bad practice. In this application however, the links stored as attributes are the links that are available to the trains at the relevant station. They are required in order for the "nextLink" methods discussed previously hence the data redundancy.

Lastly, when taking about train direction, you will have noted the use of the terms "up" and "down". In railway terminology, Down means away from a major destination whilst Up means towards a major destination. This major destination would, in our case, be London which is at the Cambridge end of the network. Therefore the down direction would be Cambridge to King's Lynn and the up direction would be King's Lynn to Cambridge. The links were designed to all have a direction of up or down depending on their direction (bi-directional link were defined as being so). This was designed so to make it easier to find the relevant for going in the relevant direction. The trains also had an direction attribute to define their direction along the network.

4.3 Development

The Development constitutes the main and longest step of the Waterfall methodology. It comes after the artefact has been designed in the previous section. In standard Waterfall however, each phase of development once finished and tested cannot be edited in any way again. In this case however, this was not adhered to as some changes to finished phases were needed through development in order for the artefact to work properly. This will be further addressed in the reflective analysis.

The first phase of building the algorithm was building the Station and StationLink classes. These must be built first. They are the building blocks of the network as they hold the data for the stations and the links between stations. These class are only data classes and are, as a result quite simple. They consists simply of a constructor and a str dunder method. These methods can be seen in fig 7.2 and fig7.3

Station has the name of the station, the name of the network and a list of instances of the StationLink class to represent links that are available from that station as attributes. StationLink stores instances of the Station class for the origin station and the destination station ,the length of the link in meters and the direction ("up or "down"). Both the list of links in Station and the instances of Stations in StationLink are utilised to find the correct link for a train stopped at an instance of station to route it along a link. A point of note is the SetLink method in the Station Class. This is used to set the links inside the list in the main file. The Stations are instantiated before the StationLinks therefore this is required to be able to do this.

Now that the lines and stations were implemented, it was possible to implement the network class (fig 7.4). This class has the following attributes: the name of the network stored as a string and two lists, a list of stations used to find the relevant station when a train reaches the end of a link and a list of Station Links.

The next class to implement was the Position class (fig 7.5). As stated earlier this keep track of the position of an individual train on a one instance to one train basis.

This class has three attributes: an instance of Station, an instance of StationLink and

a distance value. This class can be instantiated in 2 states which can be interpreted in the following way. If the station atribute is set to an instance of station, the Link attribute is set to None and the distance attribute is 0, this can be interpreted as a train stopped at a station. However, if link is set to an instance of StationLink, station to None and distance is NOT 0, this can be interpreted as the train travelling along a link. This class has two methods: UpdatePos updates the class from one state to the other as discussed. UpdateDist is used when a train is travelling on a link to increase the distance along that link.

After implementing this class, we turned our attention to implementing the train class(fig 7.6). Similarly to the previous data classes, this class only has a constructor and a string dunder class. This class is responsible for storing the information about individual trains. it has the following attributes: an instance of the Position class, the velocity, the maximum velocity, the direction ("up" or "down"), and the Station Index. The maximum velocity is the integer value that is used to set the velocity when the train is travelling while the station index keeps track of the station that the train was at before it started travelling and is incremented or decremented in the simulation class depending on the direction of travel stored in the direction attribute.

The next class to be implemented was the state class (fig 7.7). this is because it requires a dictionary of the position of the trains and a dictionary of the velocities of each train as it store these as attributes. This means that the previous classes had to be made before this one in order to have the required data. the last attribute is the time. These are all stored as reference and therefore are updated in this class as they are updated in instances of position and train. Every time the time is incremented in the simulation class, a deep copy of the previous state is made so that the state at each time tick can be checked. For each dictionary in this class, the keys are the train numbers (i.e. train1, train2 and so on) for ease of access by the simulation class.

The last class before we can implement the simulation is the timetable(fig7.8). This class handles the timetables which state where the trains should be at what time. They are split into two lists of dictionaries stored as attributes. The first list has the

departure times for each train whilst the other has the arrival times. Each dictionary represents a timetable for a train. The keys for the dictionaries are the station name. This was done in this way so that the relevant time from the timetable can be easily accessed with the station name.

The Simulation class was now implemented. This is by far the biggest class and runs the whole simulation. The attributes (fig.7.9) for this class are as follows: a Boolean SimRun attribute set to True by default. The simulation end when this attribute is set to False. An instance of network to have access to the instances of station, an instance of timetable to check where the trains should be at the relevant time, a list of train instances to have access to the relevant data for each train, an integer attribute "time" which keeps the time the simulation is at, an instance of state to access the position of each train as well as their velocity, and lastly an empty dictionary to store the deep copies of state whilst the simulation runs.

Due to the complexity of this class, the building will be explained method by method. Firstly, we have the simStart method (fig7.10). As the name suggests, this starts and also runs the simulation. It consists of a while loop which halts when the simRun parameter is set to false. In every iteration of this, the histState attribute is updated with a deep copy of the state from the previous iteration. The time attribute is then incremented by one to begin the next time tick. Then the Step method can be called.

The next method to build is the step method (fig7.11) called by simStart. This class is where the magic happens. Firstly, an index is set to be able to access the correct timetable for the relevant train. A for loop is then defined to loop through each train's position through the state attribute with "key" being the key for the dictionaries in state (i.e. train1). In each iteration of the loop, the relevant local variables are initialised: the trains' position, velocity and departure timetable. The timetable is accessed with the index set earlier. The current state of the train is then checked by a range of if statements. the first if statement (fig7.11) checks whether the train is stopped at a station and whether the current time matches the departure time in the timetable. If true, the train will change state and begin travelling along a link. To do so, the direction would be checked to call the right NextLink method

(NextLinkUp or NextLinkDown) to find the correct link. The position will then be updated by calling the UpdatePos method of the postition class. the velocity is then set to the trains' maximum velocity. You will notice in (fig) that this if statement is inside a try except statement. This is to handle the KeyError Exception that occurs once a train has reached its destination and the timetable has no key for the station the train is stopped at. Without this the program would terminate.

The next if statement (fig7.11) checks whether the train is moving along a link and if so it calculates a new distance by adding the speed in m/s to the current distance and updates the distances in the instance of Position using the UpdateDist method.

This if statement (fig7.12) checks if the train is on a link, if so it will check is it has reached the end of the link, if it has, it will check the direction and increment or decrement the stationListIndex accordingly to find the correct station the train should be at. Once it has done that it will update the position using the UpdatePos method of the Position class and set the velocity to zero

The last if statement (fig) simply checks the time and if the time has reached the specified value, the SimEnd method is called. After all of the conditional statements have been executed, the timetable index is incremented ready for the next iteration.

The building of the last three methods (Fig7.12) will be discussed together as they are fairly small. Firstly, the SimEnd method sets the simRun attribute to False halting the while loop and stopping the simulation in doing so. This is a very important method as it ensure that the while loop halts. The NextLinkDown and NextLinkUp method were built next and were originally a single method which was split into two as discussed previously. These retrieve the link for the relevant directions by doing a linear search of the links at the station passed as an argument. the GetHistState method was the last to be built and simply returns the histState attribute to build the SimulationSnapshot class

The last class to be built was that Simulation Snapshot class (fig7.13), this class is simple and has just a single attribute. This is the HistState dictionary compiled by the Simulation class and returned by the GetHistState method. Additionally, it

has a single method which prints the relevant state from the dictionary using the timeIndex parameter as a key for the dictionary.

Lastly, the main file was built were all the classes were instantiated as intended as well as the command line interface that prints the status of each train when the simulation concludes. This just takes the relevant information from the relevant instances of position and prints it to the console.

4.4 Testing

As is the case for many simulation projects, it is often difficult to predict the result of a test. More "conventional" projects would benefit from having unit tests to ensure that every part of the source code is working as intended. This is the standard set of testing on software engineering, notably and is known as "Unit Testing" Nidhra and Dondeti, 2012. For this artefact however, this is not a practical way of testing. This is because, with simulating a real life event, one cannot always correctly predict the outcome meaning that we cannot easily nor accurately test our artefact using this type of testing. Furthermore, our algorithm requires every part of the program in order to accurately execute. This makes the program difficult to split into units to test. It is much easier to test it as a whole rather than testing every class individually.

Instead, it was decided to opt for an integration testing approach which "validates that two or more units or other integration work together properly" (Nidhra and Dondeti, 2012). Using this approach, a new part of code would be integrated into the whole build and tested for an outcome. If the outcome matched the expectation the test would pass. Had the test not passed, the cause would be investigated. Test cases consisted of whether the train reacted in the expected way relating to the timetable and the state it should be in at a particular time. For instance, if the time is 1 and the train should depart at 1 from a station and it is still at that station at 2, the test would fail as the expected outcome did not happen.

When a bug was identified, the program was extensively debugged to check the source of the error. This was done until a source for the bug could be identified and

fixed and the expected outcome reached within that point in the algorithm. This allows us to ensure that the predicates of the algorithm were as expected and correct and also allowed us to check that the algorithms was working as expected

As this is an algorithm featuring heavy use of loops, we are required to have a special conditional statement (fig4.2) to catch the correct point of iteration in the loop. When that condition gets thrown, we can have the algorithm hit a breakpoint in order to be able to effectively debug the simulation and gain an insight into the state and predicates of the algorithm at a said iteration.

```
#(Used for testing)
#if self.time == 1062 and key == "train1":
#   print("breakpoint here")
```

Figure 4.2: Step Test Code

As expected upon testing, a number of bugs were found, the most common causes for bugs included unreliable algorithms such as the first NextLink method design, Timetable bugs stemming from badly calculated timetables as well as typing mistakes.

4.5 Operation

The program operates from the Project main.py file. This files instantiates all of the classes with the required information. The classes must instantiated in a specific order. Doing it in other orders would not be possible as some classes require instances of other classes to be active as they depend on that information to be properly instantiated. Once the classes have all been instantiated in the correct order, the startSim method is called which is the class running the simulation for a specified amount of time (3700 seconds in the demo). Every time a train changes state, a prompt will appear in the command line stating the new station or link as well as the train number and the times in seconds. At the end of the simulation, the status of every train will appear. When reading the states, if link is none and station is Cambridge and the distance is 0 for example, this means that the trains is stopped at

Cambridge. However, if the station is none and the link is set to between Cambridge and Cambridge North for example, this means that the train is travelling along that link. The distance should then not be 0 in most cases. All distances are in meters while all measure of time are in seconds.

The project main has a linear complexity of O(n) whilst the Simulation class has a quadratic complexity of O(n^2). This is due to the nested loop that is present in the methods when running the simulation. Although this hasn't been tested, it is estimated that the program would be about to handle many trains and networks. Although in an extreme case, it may run out of memory resulting in a crash. It would also multiply the execute time substantially due to the time complexities mentioned above especially the simulation class time complexity.

Chapter 5

Conclusions

To conclude, we can see that there are some improvements on the time taken to travel over the chosen network. Indeed, we can observe from the timetable that the stopping train took 2026 seconds to travel along the network. In minutes, this equates to 34 minutes. The non-stopping trains took 1671 minutes which equates to 28 minutes. This gives us a difference of 6 minutes. This does not seem like much of a difference. It is important, however, to bring a few points to the foreground. The Simulation did not simulate accurately the acceleration present on the railway in reality. In the simulation, the trains went instantly for 0 to 40m/s which is not realistic in the slightest. This means that the time gap between to two train would be greater when simulated on a real railway. Furthermore, the simulation only had very short stops. The stops programmed where only one minute however a train would stop for at least double the time in a station.

This means that even if the data from the simulation is a bit flawed and only really gives us a picture of an unrealistically bad case. We can conclude that we can still see a visible improvement of Non-Stopping trains over conventional trains. It begs the questions: Would it be worthwhile to implement in the UK? This depends on several factor such as the cost of the new infrastructure, the line it is implemented on , as well as the current state of the route. Indeed, it would be worthwhile if the route was already very overcrowded and busy as well as having a high density of stations to benefit the most from such a system. Now having this small of a gap between the two system would mean that it would only accommodate for one more train if any which considering the implication of this system isn't too impressive. However with

the data not being overly accurate. One cannot have an accurate representation of this. This would be to be investigated as a problem of its own had we have more time. The third question to answer is if it provides a long enough lasting solution to the estimated growth. This is very difficult to answer as the numbers show that real usage is very low at the moment due to the tolls of the pandemic. Some sources suggests that the railway will see pre-pandemic numbers. If we were to see that happening, it would be hard for our system to keep up with the ever-rising demand.

Chapter 6

Reflective Analysis

The overall project can be said to have been a success. A working simulation was built which allowed us to estimate numbers and answer the research questions that were asked in the introduction. However while we consider the project a success in that regard, there are many points of improvements that have to be considered and that could have made the study much more accurate and meaningful .

Firstly, train acceleration would have been nice to implement. It would have made the simulation more accurate. the current system is extremely basic and unrealistic meaning that our study lacks some accuracy. A more accurate result would have meant that the answer given in the conclusion could have been more concrete Furthermore, we could have done more research on the stop times that trains tend to follow to once again, gain more accuracy. This would have meant that the timetable would have been more polished and more like what you would come to expect to see on the real railway. This is probably the main point of improvement that would have benefited the study.

Another point is that there is easier ways of programming the network topology with an array for up and one for down in Network class instead of the current system of having the available links as a attribute of the station class. This would lead to less data redundancy and a more efficient artefact that is easier to maintain

Leading on from the previous point, more research on existing simulations would have also been beneficial. This would have meant that there would have been a greater understanding of the processes involved win making a simulation which would have

lead to a better more efficient artefact which would have undoubtedly given better result as it would have highlighted things that were not necessarily obvious this time around. The GUI, or lack thereof, also leaves something to be desired. Currently the artefact outputs the data in the command line which is the very basic way of doing it. Granted there is actually data. It could be input into a library which could plot a time-distance graph from the data. Not only would it look lovely in the conclusion, it would also illustrate the data much better. It would have also been nice to plot the network topology and animate it. It would show the trains move along it and would give a much more impressive and convincing simulation.

The main reason for the lack of GUI was time management. Whilst there was a good effort to have the relevant project management methods, they were not exactly stuck to the letter. Therefore, a more comprehensive review of project management and a more conscientious decision would have benefited the project. Actually sticking to the project management doctrine would have also made a difference. It would have meant that the project would not have overrun as much giving a better artefact directly leading to a better investigation.

Lastly, It would have been nice to have added a signalling algorithm to this artefact as that would have allowed us to simulate the network to the full extent and to really see how many more trains we could have put on a network. This is especially relevant considering the fact that a lot of the research resolved around signalling in the literature review.

References

Adenowo, Adetokunbo AA and Basirat A Adenowo (2013). 'Software engineering methodologies: a review of the waterfall model and object-oriented approach'. In: *International Journal of Scientific & Engineering Research* 4.7, pp. 427–434 (cit. on p. 15).

Andrei, Bogdan-Alexandru et al. (2019). 'A study on using waterfall and agile methods in software project management'. In: *Journal Of Information Systems & Operations Management*, pp. 125–135 (cit. on pp. 14, 15).

Armstrong, John and John Preston (2011). 'Alternative railway futures: growth and/or specialisation?' In: *Journal of Transport Geography* 19.6. Special section on Alternative Travel futures, pp. 1570–1579. ISSN: 0966-6923. DOI: https://doi.org/10.1016/j.jtrangeo.2011.03.012. URL: https://www.sciencedirect.com/science/article/pii/S0966692311001281 (cit. on pp. 5, 7).

Casson, Mark (2004). 'The future of the UK railway system: Michael Brooke's vision'. In: *International Business Review* 13.2. Special Issue in Honour of Michael Brooke, pp. 181–214. ISSN: 0969-5931. DOI: https://doi.org/10.1016/j.ibusrev.2003.10.002. URL: https://www.sciencedirect.com/science/article/pii/S096959310300132X (cit. on p. 1).

Gąska, D., J. Margielewicz and Cz. Pypno (2011). '"Movable platform" - the idea and energy consumption'. In: *Transport Problems* 6.44, pp. 87–92 (cit. on p. 8).

Gower, T. Leveson (Mar. 2022). *Passenger rail usage, October to December 2021*. URL: https://dataportal.orr.gov.uk/media/2050/passenger-rail-usage-2021-22-q3.pdf (cit. on p. 1).

Hayat, Faisal et al. (2019). 'The Influence of Agile Methodology (Scrum) on Software Project Management'. In: *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 145–149. DOI: 10.1109/SNPD.2019.8935813 (cit. on p. 14).

Ho, T.K. et al. (2002). 'Computer simulation and modeling in railway applications'. In: *Computer Physics Communications* 143.1, pp. 1–10. ISSN: 0010-4655. DOI: https://doi.org/10.1016/S0010-4655(01)00410-6. URL: https://www.sciencedirect.com/science/article/pii/S0010465501004106 (cit. on p. 20).

Ireland, Andrew (n.d.). *The Absolute Block System of Railway Signalling: An Example of a Distributed Concurrent System*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.8718&rep=rep1&type=pdf (cit. on p. 8).

Nidhra, Srinivas and Jagruthi Dondeti (2012). 'Black box and white box testing techniques-a literature review'. In: *International Journal of Embedded Systems and Applications (IJESA)* 2.2, pp. 29–50 (cit. on p. 30).

Pachl, Jörn (2020). 'Railway Signaling Principles'. In: URL: https://publikationsserver.tu-braunschweig.de/servlets/MCRFileNodeServlet/dbbs_derivate_00047453/eBook_RSP.pdf (cit. on p. 8).

Pearson, Leonard V. (Feb. 1973). *Moving block railway signalling*. URL: https://repository.lboro.ac.uk/articles/thesis/Moving_block_railway_signalling/9526697 (cit. on p. 7).

Preston, John, James Pritchard and Ben Waterson (2017). 'Train Overcrowding: Investigation of the Provision of Better Information to Mitigate the Issues'. In: *Transportation Research Record* 2649.1, pp. 1–8. DOI: 10.3141/2649-01. eprint: https://doi.org/10.3141/2649-01. URL: https://doi.org/10.3141/2649-01 (cit. on p. 1).

Roktop, Noa and Peng Yu-lun (2008). *"Non Stopping Train"*. URL: https://thefutureofthings.com/3656-non-stopping-train/ (cit. on pp. 9, 18).

Sin, L.K. and C.J. Goodm (1994). 'An object-oriented power network simulator for multi-train simulations'. In: *Transactions on the Built Environment* 6, pp. 484–490. ISSN: 1743-3509. URL: https://www.witpress.com/Secure/elibrary/papers/CR94/CR94057FU.pdf (cit. on pp. 20, 21).

Takagi, R. et al. (2006). 'Optimal Train Control At A Junction In The Main Line Rail Network Using A New Object-oriented Signalling System Model'. In: *Computer In Railways X* 88, pp. 479–488. DOI: 10.2495/CR060481. URL: https://www.witpress.com/elibrary/wit-transactions-on-the-built-environment/88/16650 (cit. on pp. 9, 20, 21).

Tyrrall, David (2004). 'THE UK RAILWAY PRIVATISATION : FAILING TO SUCCEED?' In: *Economic Affairs* 24.3, pp. 32–38. DOI: https://doi.org/10.1111/j.1468-0270.2004.t01-1-00488.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1468-0270.2004.t01-1-00488.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1468-0270.2004.t01-1-00488.x (cit. on p. 6).

Wellings, Richard (2014). 'The Privatisation of the UK Railway Industry: An Experiment in Railway Structure'. In: *Economic Affairs* 34.2, pp. 255–266. DOI: https://doi.org/10.1111/ecaf.12083. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ecaf.12083. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ecaf.12083 (cit. on p. 6).

Chapter 7
Appendices

7.1 Appendix 1 - Links & Word Count

Word Count - 10337

Video Demonstration - `https://youtu.be/ky5jl4lkrcc`

Source Code Repo - `https://github.com/AlexPBarlaam/Y3-Project_Code`

7.2 Appendix 2 - Railway lines Figures

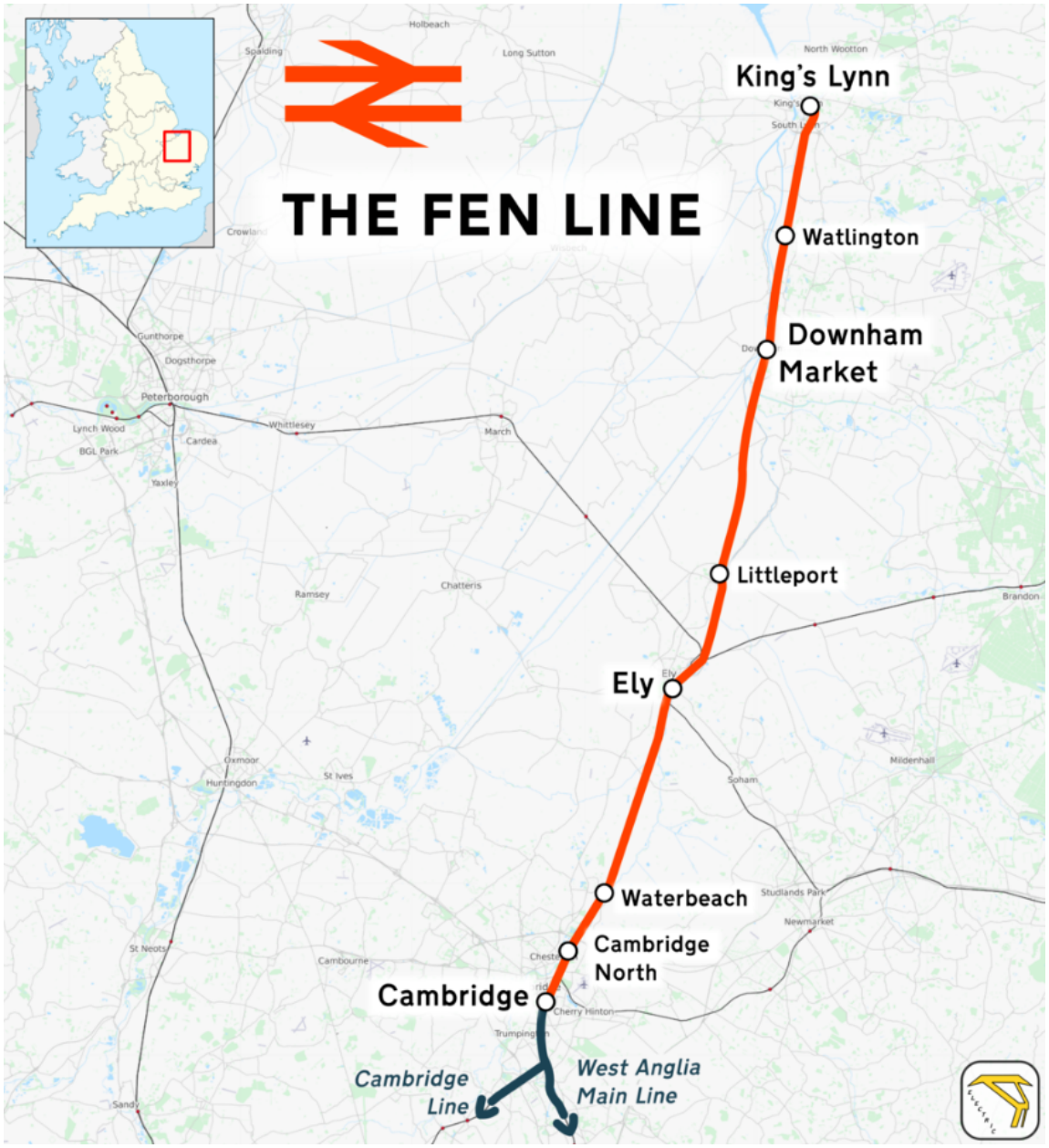

Figure 7.1: Diagram Of the Fen line

7.3 Appendix 3 - Code Figure

```python
class Station:

    def __init__(self, sN) -> None:
        """Station Constructor (Represents the station on the network)

        Args:
            sN (str): Station Name
            I (int): Station Index for simulation
        """
        self.Name = sN
        self.NetName = "Fen Line"
        self.links = []


    def __str__(self) -> str:
        """Station ToString Method

        Returns:
            str: text representation of Station attributes
        """
        string = self.Name + " is a station on the " + self.NetName
        return string

    def SetLink(self, l) -> None:
        """Sets the links after the StationLink objects are instanciated

        Args:
            l (StationLink Object List) : Links available at station
        """
        self.links = l
```

Figure 7.2: Station Class

```python
class StationLink:

    def __init__(self, oS, dS, lDm, D) -> None:
        """StationLink Constructor (Represents rail links between station)

        Args:
            oS (Station Object): Link Origin Station
            dS (Station Object): Link Destination Station
            lDm (Int): Lenght of Link
            D (Str): Direction of link (Has to be "down" or "up" or "Bi-Dir")
        """

        self.originStation = oS
        self.destStation = dS
        self.linkDistanceMeters = lDm
        self.direction = D


    def __str__(self) -> str:
        """StationLink ToString Method

        Returns:
            str: text representation of StationLink attributes
        """
        string = "this is a "+ str(self.linkDistanceMeters) +" m long link between " + self.originStation.Name + " and " + self.destStation.Name
        return string
```

Figure 7.3: Station Link Class

```python
class Network:

    def __init__(self, nN, sA, sL) -> None:
        """Network Constructor (Represents Railway Network)

        Args:
            nN (string): Name of network
            sA (Station objects Array): Stations on the network
            sL (StationLink objects Array): Station links on the network
        """

        self.netName = nN
        self.stationsArray = sA #List of objects of type Station
        self.stationsLinks = sL #List of objects of type StationLinks

    def __str__(self) -> str:
        """Network ToString Method

        Returns:
            string: text representation of Network attributes
        """

        string = "the " + self.netName + " is a railway line between " + self.stationsArray[0].Name + " and " + self.stationsArray[-1].Name
        return string
```

Figure 7.4: Network Class

```python
class Position:

    def __init__(self, s, l, d) -> None:
        """Position Constructor (Stores the position of a single train)
        Args:
            s (Station Object): Station train is at if stopped
            l (StationLink Object): Link train is travelling on
            d (int): distance of train along the link
        """

        self.station = s
        self.link = l
        self.distance = d

    def UpdatePos(self, s, l, d) -> None:
        """Updates the position of the train

        Args:
            s (Station Object): Station train is at if stopped
            l (StationLink Object): Link train is travelling on
            d (int): distance of train along the link
        """

        self.station = s
        self.link = l
        self.distance = d

    def UpdateDist(self, d) -> None:
        """Updates the distance of a train along a link

        Args:
            d (int): distance of train along the link
        """
        self.distance = d
```

Figure 7.5: Position Class

```python
class Train:

    def __init__(self, p, v, Mv, d, sI) -> None:
        """Train Constructor

        Args:
            p (Position Object): Instance of postition class for the train
            v (int): current speed of the train
            Mv (int): max speed of the train
            d (str): direction of train (Has to be "up" or "down")
            SI (int): Index to find stations
        """
        self.pos = p
        self.velocity = v
        self.maxVelocity = Mv
        self.direction = d
        self.stationListIndex = sI
```

Figure 7.6: Train Class

```python
class State:
    #represents the state of the world at a point in time
    # could be dicts of each train
    #pos =  instance of position class


    def __init__(self,t,p,v) -> None:
        """State Constructor (Represents the state of the world at a point in time)

        Args:
            t (Int): Time of the State
            p (Dictionary of Position Objects): Positions of the trains
            v (Dictionary of Ints): Velocity of the trains
        """

        self.time = t
        self.pos = p #dict of the pos of the trains
        self.velocity = v #dict of the velocity of each trains

    def ProtoPrinting(self):
        """Temporary Method to test printing of the class
        """

        print(self.pos)
        print(self.velocity)

    def __str__(self) -> str:
        """State toString Method

        Returns:
            str: text representation of State attributes
        """
        pass
```

Figure 7.7: State Class

```
class Timetable:

    def __init__(self,d ,a):
        """Timetable Constructor (Stores the Timetable for the Simulation)

        Args:
            d (array of dicts); ordered list of departures times
            a (array of dicts); ordered lists of arrival times
        """
        self.departures = d
        self.arrivals = a

    def __str__(self) -> str:
        """Timetable ToString Method

        Returns:
            str: text representation of Timetable attributes
        """
        pass
#ordered list of times with ordered list of stations
#would work as program for train driver
```

Figure 7.8: Timetable Class

```
import copy as C

class Simulation:

    def __init__(self, n, t, SI, tr) -> None:
        """Simulation Constructor (Runs the Simulation)

        Args:
            n (Network Object): Network to host Simulation
            t (Timetable Object): Simulation Timetable
            SI (State Object): Represent Simulation State at a point in time
            tr (list of Train Objects): Trains used by the simulation
        """

        self.simRun = True
        self.network = n
        self.timetable = t
        self.trains = tr
        self.time = 0;
        self.state = SI #this has to be an instance of the state class
        self.histState = {} #dictionary of previous states
```

Figure 7.9: Simulation Class Attributes

```
    def simStart(self) -> None:
        """Starts and Runs the Simulation
        """

        '''loop
            historical state = state.copy HAS TO BE DEEP COPY
            append historical state to historical states array
            time = time + 1
            calls step to update the state of simulation
        starts the step '''

        while self.simRun == True:
            self.histState.update({self.time : C.deepcopy(self.state)})
            self.time += 1
            #print("time = " + str(self.time))
            self.step()
```

Figure 7.10: Sim Start Method

```
def step(self) -> None:
    """updates the positions
    """

    timesIndex = 0 #index for timetable

    for key in self.state.pos:
        pos = self.state.pos[key]
        velocity = self.state.velocity[key]
        times = self.timetable.departures[timesIndex]

        #(Used for testing)
        #if self.time == 1062 and key == "train1":
        #   print("breakpoint here")


        try:
            if velocity == 0 and pos.link == None and times[pos.station.Name] == self.time:
                if self.trains[key].direction == "down":
                    nextLink = self.nextLinkDown(pos.station)
                elif self.trains[key].direction == "up":
                    nextLink = self.nextLinkUp(pos.station)
                print(nextLink)
                print(key + " @ " + str(self.time))
                pos.UpdatePos(None, nextLink, 0)
                self.state.velocity[key] = self.trains[key].maxVelocity
        except KeyError:
            pass

        if velocity != 0 and pos.station == None:
            newDist = pos.distance + velocity
            #print("newdist = " + str(newDist))
            pos.UpdateDist(newDist)
```

Figure 7.11: Step Method Picture 1

```
            if pos.link != None:
                if pos.link.linkDistanceMeters <= pos.distance:
                    #print("should go to next link")

                    if self.trains[key].direction == "down":
                        self.trains[key].stationListIndex += 1
                        station = self.network.stationsArray[self.trains[key].stationListIndex]
                        print(station)
                        print(key + " @ " + str(self.time))
                        pos.UpdatePos(station, None, 0)
                        self.state.velocity[key] = 0

                    elif self.trains[key].direction == "up":
                        self.trains[key].stationListIndex -= 1
                        station = self.network.stationsArray[self.trains[key].stationListIndex]
                        print(station)
                        print(key + " @ " + str(self.time))
                        pos.UpdatePos(station, None, 0)
                        self.state.velocity[key] = 0

            if self.time == 3700:
                self.simEnd()

            timesIndex += 1
            #print("step ended")


    def simEnd(self) -> None:
        """Terminates the Simulation
        """

        self.simRun = False

    def nextLinkDown(self, station):
        for link in station.links:
            if link.originStation.Name == station.Name:
                return link

    def nextLinkUp(self, station):
        for link in station.links:
            if link.originStation.Name == station.Name and link.direction == "Up":
                return link
            elif link.destStation.Name == station.Name and link.direction == "Bi-Dir":
                return link

    def GetHistState(self):
        return self.histState
```

Figure 7.12: Step Method Picture 2

```python
class SimulationSnapshot():

    def __init__(self,SD) -> None:
        """SimulationSnapshot Constructor

        Args:
            SD (Dict of states): States from the simulation after it has concluded
        """

        self.stateDict = SD


    def getState(self,timeIndex):
        """Method to print a state at time x

        Args:
            timeIndex (int): timekey to access a state from the dict
        """

        print(self.stateDict[timeIndex])
```

Figure 7.13: SimSnapshot Class