A reusable Python framework for repeatable, replicable, & reproducible experiments using OpenTTD

Michal Charemza

Master of Science

Data Science, Technology, and Innovation

School of Informatics

University of Edinburgh

2024

Abstract

OpenTTD is an open source business simulation game based on the 1994 game Transport Tycoon Deluxe, and in spite of being designed for recreation, OpenTTD has been used in a number of academic studies, notably ones that investigate the behaviour of algorithms programmed into its flexible AI system. However, many of these studies have problems regarding the repeatability, reproducibility, or replicability of their experiments: this is a concern given the ongoing replication crisis in many fields of science. In response, I created OpenTTDLab, a reusable Python framework that allows OpenTTD to run experiments with AIs that avoids many of these problems. Starting with a review of the existing studies to identify and categorise these problems, and then a review of the existing behaviour of OpenTTD, I applied a highly agile process to create OpenTTDLab to augment the behaviour of OpenTTD to address the problems found—highly agile in the sense that I conducted many cycles of development informed by concurrently conducting proof-of-concept experiments using the output of the development. The combination of this design process and these experiments shows three things: firstly, that OpenTTDLab achieves its aims in terms of aiding the repeatability, reproducibility, and replicability of experiments using OpenTTD; secondly, due to the range of the experiments conducted, that OpenTTDLab is reusable; and thirdly, since some of the results appear to have real-world meanings, that OpenTTD has the potential to be used to simulate properties of the real world. Thus, while there are of course further steps that can be taken, OpenTTDLab is already a useful tool for conducting repeatable, reproducible, and replicable research that leverages the rich capabilities of OpenTTD.

Research ethics approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Michal Charemza*)

Acknowledgements

Firstly thank you to Chris Sawyer, the original author of Transport Tycoon Deluxe: without you this project would not have existed. Thank you to Josef Drexler, the original author of TTDPatch, the project that inspired OpenTTD. Then of course thank you to all the contributors to OpenTTD itself over its 20 year history: amongst these thank you to Ludvig Strigeus (A.K.A. ludde), the creator of OpenTTD, and thank you especially to Patric Stout (A.K.A. TrueBrain), who not only gave advice and what I interpreted to be light blessing to this project, but also wrote the OpenTTD save game parser that OpenTTDLab originally forked from.

Thank you also to Ian Earle (A.K.A. BasicBeluga) who found an issue in the documentation of OpenTTDLab and submitted a fix for it. While this was a small change, I took it as evidence that what I was creating stood a chance of being useful, and pushed me to continue.

Thank you to my supervisor, Michael Herrmann; his ongoing advice throughout this project has been invaluable. I thank him especially for making sure that I have some results to show and discuss rather than getting lost in just making OpenTTDLab, and I thank him for guiding me towards what I hope is now a reasonable written dissertation. And I would say he rolled with the changes to the project especially well; I shifted its focus several times between proposal and final output, and I felt supported every step of the way.

And of course thank you to my husband, Matthew Beach, an amazingly loving partner, and whose support throughout this project has been beyond invaluable, and who took my repeated talking at him incredibly well.

Contents

Chapter 1

Introduction

OpenTTD [44] is an open source real time strategy (RTS) business simulation game based on Chris Sawyer's 1994 game Transport Tycoon Deluxe. The aim of the game is to successfully run a business by constructing networks of roads, railways, airports and ports, along with their respective vehicles, trains, planes and ships, in order to transport people and goods in exchange for money. OpenTTD is played on a simulated landscape as can be seen in Figure 1.1.

OpenTTD was created as a game for recreation: it allows a single player to play in a non-competitive world building mode; multiple human players playing cooperatively or competitively; and so-called AI players, which via custom code that controls a company, provide opponents for humans to play against. However, it is remarkably flexible: it has been successfully used as a tool to research algorithms including artificial intelligence (AI), machine learning (ML), and anomaly detection algorithms where OpenTTD AI(s) play without human player involvement [6, 7, 23, 24, 34, 50, 49], as a tool to research scalability and mobile applications [20], and as a teaching aid for concurrency in computer programs [15, 27][1] and for supply chain and logistics management [25].

Simulation games have a long history of being used to inform government policies; see Raghothama & Meije [33] for a summary that focuses on logistics and transportation. In spite of this, no reference to using OpenTTD in this way has been found. A reason in Raghothama & Meije [33] is suggested: while some of the transportation aspects of the networks in OpenTTD are realistic, its economic model is not.

Of all these possible uses for OpenTTD, using OpenTTD as a tool for researching

[1] The work of Hansen & Murphie [15] refers to an *OpenTTD lab*, which is a set of scenarios and exercises for students to follow. It is not directly related to the OpenTTDLab presented here.

Figure 1.1: A small section of an OpenTTD version 13.4 game showing parts of a rail, road, and sea transportation network, as well as several industries. At the top is the control bar through which the player takes actions or see more details on the state of the game, such as finances.

algorithms is the primary focus of the current work, and specifically the focus is the development of a reusable tool, OpenTTDLab, that augments the existing features of OpenTTD to improve its ability to be used in repeatable, reproducible, & replicable research where OpenTTD AI(s) play without human player involvement. Given what is now the well-known replication crisis[2] [19, 3], including in computer science [14, 12], if there are problems with how such research has been conducted using OpenTTD, as I will argue, then the potential usefulness of such a tool for future research is clear.

As a secondary and more speculative aim, it is hoped that the work here allows for OpenTTD to be investigated as a tool to simulate supply chain or transportation networks that could ultimately inform government policies. However, the nature of the correspondence between OpenTTD and what it is simulating must be determined, or in other words it must be *validated* [30].

[2]The replication crisis is also known as the replicability crisis, the reproducibility crisis, and the credibility crisis.

1.1 Repeatability, Reproducibility, & Replicability: The 3Rs

The terms *repeatability*, *reproducibility*, and *replicability*, the so-called *3Rs*, unfortunately do not historically have universally agreed meanings [31]. Even the highly-cited 2016 Nature survey that reported that approximately 50% of scientists believe there is a substantial reproducibility crisis [3] uses the terms reproducible and replicable seemingly interchangeably. And both helpfully and confusingly, the Association for Computing Machinery (ACM) in 2020 swapped their definitions of reproducibility and replicability to align with the broader scientific community [2].

Choosing what I believe to be an authoritative source, I primarily use the post-swap ACM definitions from its *Artifact Review and Badging Version 1.1* [1]; these appear to broadly align with the definitions of *The Turing Way* [48], a collaborative guide to reproducible data science spearheaded by the UK's national institute for data science and artificial intelligence, a similarly authoritative source. The ACM definition is given in terms of artifacts:

> **Artifact** ... we mean a digital object that was either created by the authors to be used as part of the study or generated by the experiment itself. For example, artifacts can be software systems, scripts used to run experiments, input datasets, raw data collected in the experiment, or scripts used to analyze results.

and the definitions of the 3Rs are:

> **Repeatability** (Same team, same experimental setup)
> The measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials. For computational experiments, this means that a researcher can reliably repeat her own computation.
>
> **Reproducibility** (Different team, same experimental setup)
> The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials. For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.
>
> **Replicability** (Different team, different experimental setup)
> The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple

> trials. For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.

These are not independent properties: reproducibility is unlikely to be achieved without repeatability [16], and replicability would be difficult to interpret in the absence of reproducibility [29]. While this highlights how the definitions are intertwined and therefore complex, it also means that a tool that eases repeatability eases reproducibility, a tool that eases reproducibility eases replication, and transitively a tool that eases repeatability indirectly eases replicability.

While ACM's definitions are clear in some respects, there is vagueness, especially in terms of what *same result* means [16]. In general, results can be said to be reproduced even if they are not exactly the same, i.e. not *bitwise* identical. For example, in simulation studies if the original random seeds are not available but all other artifacts are, then results can still be said to be replicated if statistical results are the same [26]. Where appropriate I make a distinction between bitwise and non-bitwise reproducibility.

Note that the definition of artifact is limited to objects created by the authors; if research uses OpenTTD then it appears the research can be deemed replicated without OpenTTD itself having to be developed independently, simply because it was already available; to me this seems an unexpected loophole in the definitions. Similarly to how reproducibility can be bitwise or non-bitwise, if we extend the ACM definition of artifact to include other software used to generate the results, as I would argue we should, there appears to be degrees of replication that depend on how much of the original software used to run the simulation has been developed independently. It is beyond the scope of this project to deal with these issues in general, other than to ensure clarity as to what is or can be developed independently when discussing replication. In spite of the crisis being well known for approximately two decades, even as of January 2024 the replication of simulation studies is a "novel endeavor", and there are "no set criteria to assess the alignment of replicated simulation results with the original results" [26]; so I argue this a reasonable limitation to the scope of this project.

1.2 3Rs issues of existing OpenTTD research

The aim of OpenTTDLab is to help studies that generate results by running OpenTTD without a human player and so only with AIs. From the first 100 results of a search for "OpenTTD" in Google Scholar in July 2024, seven existing studies have been found that

do this: the work of Beuneker [6], Bijlsma [7], Konijnendijk [23], Lakomỳ [24], Rios and Chaimowicz [34], Volna [49], and Wisniewski and Witt [50]. Informally reviewing these in terms of how they have or have not seemingly achieved repeatability (the domain of the original researchers) or help achieve reproducibility or replicability (the domain of independent researchers), I found 7 problems. For brevity not all problems for all studies are listed.

- **Issue 1: Insufficient artifacts supplied to reproduce or analyse results**
  One of the most complex and impressive studies is the work of Bijlsma [7] that used dynamic scripting to evolve the code of an OpenTTD AI using a genetic algorithm. In total there were at least 1250 experiments: 50 experiments each for 25 generations of the genetic algorithm, and after each of these, in-game metrics must have been extracted. There is nothing in OpenTTD that allows this to be done automatically, and it seems infeasible to have run this manually, especially because the AI code would be different each generation. It is mentioned in the text that a modified version of OpenTTD was used, but it does not describe how it was integrated with the genetic algorithm, and I was unable to find it to download.

  Similarly the work of Konijnendijk [23] suggests modifications of OpenTTD were made in order to extend its AI system with the algorithms presented in the work; I could not locate the code of these modifications, nor the code of the AIs that used them.

  No study included the code used to analyse results or produce visualisations.

- **Issue 2: Low or unknown number of repetitions**
  While there is no one-size fits all number to how many repetitions an experiment must have, some of the studies I would argue have repetitions that are so few it makes me doubt the conclusions that are based on them. As examples, the two experiments of Rios and Chaimowicz [34] were run seven times each, the experiments of Wisniewski and Witt [50] were run just three times, and the work of Volna [49] does not list the number of repetitions, nor even suggest there was more than one for each configuration.

- **Issue 3: Non-comparable repetitions**
  Rios and Chaimowicz [34] ran a total of 14 repetitions over 2 experiments, but not for the same amount of in-game time. This calls into question whether these were actually repetitions of the experiment.

- **Issue 4: Manual repetitions of experiments and extraction of results**
  No study explicitly stated that their experiments were run in an automated mechanism, and given the low number and non-comparable repetitions of some of the works, I suspect many were run manually, in that OpenTTD was configured and run through its graphical interface, which I suspect is tedious and error prone.

  The study of Lakomỳ [24] explicitly describes the manual mechanism used to install and configure OpenTTD for its experiments. Having this description is excellent in terms of supporting reproduction of results, but the fact it is manual means it is high cost in terms of time to try to replicate its reported 50 repetitions.

- **Issue 5: Insufficient detail on the version or configuration of OpenTTD**
  The version of OpenTTD used to generate results is frequently omitted. For example, the work of Bijlsma [7] does not mention the version of OpenTTD that was then modified. Similarly the works of Volna [49] and Beuneker [6] do not mention the version of, the presumably unmodified, OpenTTD used.

  OpenTTD has a wide array of configuration options that can affect results, but it varies how well these are described. For example, the work of Wisniewski and Witt [50] describes the configuration exhaustively and in one place, but the work of Beuneker [7] only has limited detail, and what there is is spread throughout the work.

- **Issue 6: No random seeds to bitwise reproduce results**
  Possibly a special case of configuration, but it merits special attention since to gain statistical results typically experiments would be run for a fixed configuration over a range of random seeds. However, only one study reported the random seeds used: the work of Beuneker [6]. Not providing the random seeds impacts reproducibility, and specifically bitwise reproducibility: without the random seeds it is impossible to reproduce the results exactly.

- **Issue 7: Insufficient or unclear detail of the algorithms to replicate results**
  All of the studies included some high level detail of the algorithms implemented, but I found it difficult to judge if it each is sufficient to replicate results within the time constraints of the current work.

  However, even if there is sufficient detail, in all cases bar one they are not presented in a way where replication appeared considered: the detail is spread throughout the work, and not presented as a clear algorithm or set of algorithms

to be replicated. The exception is the work of Lakomỳ [24]: it includes many algorithms each of them explained and discussed, although they are presented as snippets of code rather than pseudo-code. If these snippets of code were used, this would muddy the water between reproducing and replicating their results.

Determining replicability by reading the papers is certainly not as strong evidence as actually attempting to reproduce or replicate the results. However, an independent researcher reading through a draft manuscript has been given as a recommendation to find issues before publication with the strongest of the 3Rs, replicability [26]; so I argue it is reasonable to extend the process to all of the 3Rs, and to base further actions, such as the creation of a framework to help avoid such issues, from such a review. While my review focused on OpenTTD, there are related wider problem in simulations in general: "many published works based on simulation still fail to meet the minimum conditions to ensure reproducibility" [14].

1.3 Reusability: The 4th R

There is a 4th concept given alongside ACM's definitions [2], not so much property of the results that the 3Rs focus on, but a property of the artifacts used to generate the results:

> **[Reusability]** The artifacts associated with the paper are of a quality that significantly exceeds minimal functionality. [...] they are very carefully documented and well-structured to the extent that reuse and repurposing is facilitated. In particular, norms and standards of the research community for artifacts of this type are strictly adhered to.

Repeatability, reproducibility, and replicability are all in the realm of a single piece of research, but *reusability* is concerned with the use of the artifacts to generate results of different-but-related research, possibly by other researchers. It is unfortunately often an afterthought in scientific software as it does not directly concern the current results the software is used to generate; but it has been argued that reusability helps reproducibility, and even increase the impact of the original work [5]. It is clear that a framework that helps with the 3Rs should itself be reusable and encourage reusable artifacts in work that uses it.

1.4 Chapter outline

The following chapters explain how OpenTTDLab helps experiments avoid issues with the 3Rs, especially the 7 issues listed, and in a reusable way. Chapter 2 gives more background into OpenTTD: specifically what properties of OpenTTD make it suitable and interesting enough to research, what limitations it has, and at a high level how a framework could overcome at least some of these limitations. Chapter 3 explains the design process I followed to create such a framework—the process itself provides an argument that it achieves its aims—and gives an overview of how its features help experiments be repeatable, reproducible, and albeit indirectly or through documentation, replicable.

Chapters 4–6 are the experimental chapters: they present results of experiments I ran using OpenTTDLab, and specifically they provide more evidence that there is a problem with existing OpenTTD research, and more direct evidence that OpenTTDLab can help with the 3Rs. Specifically, Chapter 4 explains how I attempted to reproduce existing published results; while I did extract results, they did not fully match the ones originally published. Chapter 4 also shows that it is relatively straightforward to use OpenTTDLab to run experiments and extract results from them, and argues that these should be repeatable and reproducible. Chapter 5 gives the details of a simple OpenTTD AI that I constructed, and easily used with OpenTTDLab to extract results that appear to have real-world meanings, thus giving some evidence that by using OpenTTDLab, OpenTTD could be used to simulate the real world as part of the secondary aim of this work as mentioned earlier. Chapter 5 also contains a clear description of the AI, and so should be replicable in the sense that another author should be able to construct a similar enough OpenTTD AI to replicate the results, possibly using OpenTTDLab. Chapter 6 contains the results of a basic exploration of the performance of OpenTTDLab, showing how some of the features described in Chapter 3 help repeatability.

Chapter 7 contains a discussion of what was found, summarises the limitations of the work, explores possible next steps, and compares OpenTTDLab to its closest equivalent. This dissertation finishes with a few concluding remarks in Chapter 8.

Chapter 2

OpenTTD: Its model and abilities

A game of OpenTTD is made up of a rectangular world with one or more companies in it, typically working in competition against each other. A company is paid for transporting goods and passengers between certain combinations of industries and towns, and in order to do so it needs to invest in road, rail, sea or air infrastructure and vehicles.

When using OpenTTD for recreation there would be one or more human players that each control a company. However, as mentioned at the beginning of Chapter 1, this is not a requirement: OpenTTD allows non-human players, so-called AI players, fully controlled by custom code. If there are no human players, OpenTTD then runs deterministically—using a seed value on startup that controls all pseudo-random behaviour of the game. These abilities are the core abilities that allows OpenTTD to be used in repeatable, reproducible, and hopefully, replicable simulations.

This chapter explores this model and these abilities in more detail, as well other related features that allow OpenTTD to be configured and ultimately used through the OpenTTDLab framework presented in the next chapter, to run simulations. This chapter also shows that the environment of OpenTTD is a rich one; while there are clearly some unrealistic aspects, it is at least a tempting target for running simulations that could be used to extract insights about the real world.

Unless otherwise stated, the details here are from OpenTTD Wiki [45], the OpenTTD source code [43], the OpenGFX source code [38], the OpenTTD AI API documentation [39], the OpenTTD GameScript API documentation [41], or by playing OpenTTD itself. Details are correct as of OpenTTD 13.4.

2.1 Field of play

OpenTTD runs on a two-dimensional rectangular tiled grid representing an area of land and sea, with some limited three-dimensional aspects. The grid size is configurable on startup between 64×64 and 4096×4096 tiles. Each tile is square, albeit presented to the user in a dimetric view, as can be seen in Figure 1.1[1]. Each tile does not necessarily have a single object on it, some objects can co-exist with each other depending on the object type. The positions of vehicles that carry cargo and passengers, which are the main mechanism for making money in the game, are even more granular: they move and appear to be able to take any position on their path between destinations.

There is some concept of height: each section of a grid can have one of a limited number of height levels in the game. This impacts the game in more than just a visual way: when going uphill road and rail vehicles usually slow down, and as discussed later in the current chapter, this can have an impact on money made. Rail and road tunnels can be built by the companies which can avoid this, but they have their downsides: for example they typically cost the company more to build, and can have their own speed restrictions.

2.2 Startup

There are two classes of startup: a pre-defined world or a pseudo-random world. In the pseudo-random case, which is the case of all experiments of Chapters 4–6, the world is deterministically generated from the configuration of the game and an integer seed to the built-in random number generator, referred to as the *random seed*. Specifically, the land, height, trees, coastlines, the sizes and positions of towns, and the positions and types of industries are all deterministically generated. This determinism means that OpenTTD can be used for bitwise repeatable and bitwise replicable experiments.

No transport infrastructure exists on startup other than roads within towns. It is the role of the players, including any AI players, to build these to connect towns and industries in order to earn income. On startup each company is provided a loan with which to begin building this infrastructure.

[1] A *dimetric* view is sometimes inaccurately referred to as an *isometric* view.

2.3 Climates

OpenTTD has four different *climates*: *temperate*, *sub-arctic*, *sub-tropical* and *toyland*. A climate is fixed on game startup and effects a variety of properties of the game such as what transportation types are available, and what industries and supply chains are possible. For brevity all descriptions are limited to the default climate of temperate, as are the experiments of Chapters 4–6.

2.4 Economic and transportation model

The possible supply chains in the temperate climate of OpenTTD can be seen in Figure 2.1. In total there are 13 different types of industries that can produce or receive cargo: iron ore mines, coal mines, forests, farms, steel mills, oil wells, power stations, sawmills, factories, oil refineries, oil rigs, banks and towns; and 11 types of cargo: iron ore, coal, wood, livestock, grain, steel, oil, goods, valuables, mail and passengers. Towns and passengers are maybe not technically industries and cargo, but I include them because in terms of the economic model of OpenTTD they are.

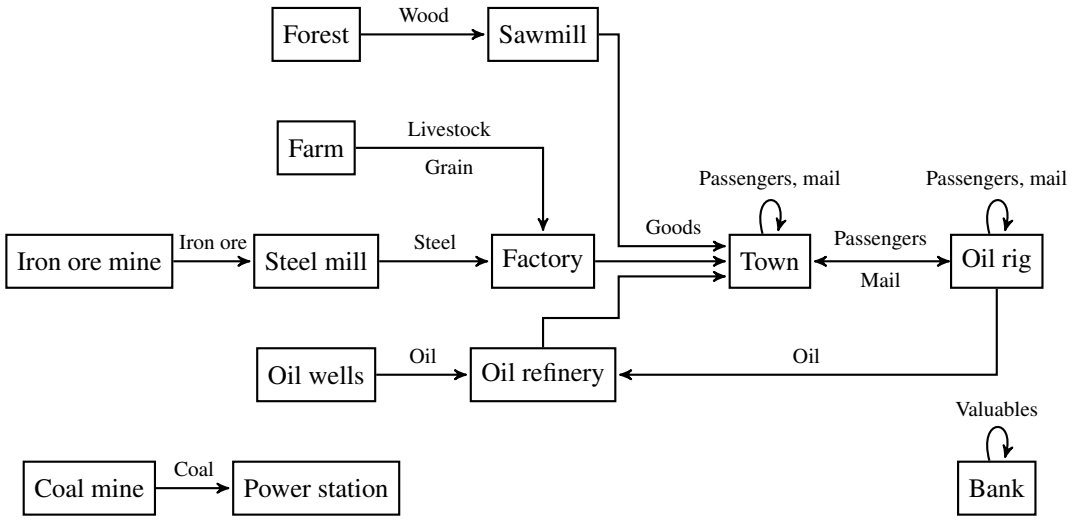


Figure 2.1: The possible supply chains of the temperate climate of OpenTTD 13.4, showing what types of cargo industries and towns accept and produce. Adapted from 'Flow chart for Temperate Cargo' in the *OpenTTD Wiki* [37].

The companies in OpenTTD, controlled by either human or AI players, receive money for transporting the 11 cargo types between the 13 industry types. The amount of money is dependant on a number of factors: the amount of cargo, the cargo type, and

the time taken—a shorter time results in more money—and the distance between source and destination—longer distance results in more money. How much cargo each industry or town outputs also depends on a number of factors: for example, for an industry that receives cargo, its output depends on how much cargo it has recently received. This is similarly the case for towns: they grow in response to nearby activity and then "produce" more passengers ready to travel. The other factors are omitted for brevity.

The 11 cargo types can be transported by 4 classes of transportation: road, rail, sea and air; and each of these have further sub classes. With the exception of oil rigs, none of the industries have stations for transportation to pick up or drop off cargo, and so companies must build them at a cost. Stations can also only be built on flat or almost-flat land, which requires earth works which also costs. Companies must also build the road or rail networks to connect the stations. Road and rail networks cannot be built arbitrarily on sloping land: there are often earth works required to shape it which costs the company; this fact explains some of the results of Chapter 4. With some exceptions, depots must also be built which are then used to build the vehicles that transport the cargo. Once built, the company configures each vehicle with a route: a set of stations to travel between to pick up and drop off cargo; the minimum useful route is 2 stations long. Vehicles can break down and block other vehicles for a period of time; this depends on configuration, and the vehicle's type and age.

There are a number of obvious absurdities to the economic model. The number of goods for example: there are 11 in OpenTTD, but the UK Tariff, the listing of all rules that govern imports into the UK, refers to over 25,000 different commodities [9]. Also, each instance of each industry will always accept any amount its input cargo types that are shown in Figure 2.1, and will pay the company that transported it. It also doesn't matter which specific instance of an industry any particular piece of cargo is taken to. For example when a passenger arrives at a station, that particular passenger has no particular destination in mind—the company will receive money for taking them anywhere, but more money if they are taken further.

It is beyond the scope of this project to more rigorously analyse or validate the economic model of OpenTTD, and so to work out how much of any insights generated by OpenTTD simulations can be applied to the real world. As briefly mentioned in the previous chapter, it has been suggested that some of OpenTTD's network effects could be realistic, which presumably would involve the supply chains of Figure 2.1. I hold out hope that OpenTTD could be used as a valid simulator for investigating such effects, and specifically by using OpenTTDLab.

2.5 Agents, ticks, and threads

OpenTTD is an agent-based system: buses, trains, planes, ships, industries and even buildings in towns run apparently independently and in accelerated real time. However, the real-time aspect is somewhat of an illusion. OpenTTD runs on the concept of *ticks*, where there are 74 ticks per in-game day. During a tick game time progresses in a deterministic way: vehicles move a certain amount, pick up or drop off a certain amount of passengers or goods, industries produce a certain amount of goods, and a certain number of passengers wanting to travel are "produced" by towns and oil rigs.

This determinism makes OpenTTD particularly suitable for bitwise repeatable simulations. If for example, instead of ticks, threads were used for the various agents, since thread scheduling is in general non-deterministic, then bitwise repeatable experiments would not be possible. OpenTTD does use threads for secondary activities, such as showing progress bars or saving games, but with these exceptions the core of OpenTTD can be described as *single-threaded*. This means that OpenTTD cannot speed up run time by leveraging the multiple CPU cores that are commonplace even on low-end computers: a possible issue if using it to run many repetitions of experiments.

2.6 OpenTTD AIs

OpenTTD AIs are scripts written using the Squirrel language that can be plugged into OpenTTD at runtime in order to control a company. OpenTTD offers such AIs a rich API, with 545 functions to control the game or inquire into its state [39] where many of the functions act on specific tile(s) of the field of play. It works within the tick framework, in that every tick of the game only a certain amount of the AI script can progress. OpenTTD AIs can also be parameterised—accepting parameters set before the start of the game. For example, the AI created for the experiments of Chapter 5 accepts a single parameter that controls how many buses to create. For the avoidance of doubt, OpenTTD AIs can be extremely simplistic, without any of the advanced learning behaviour often associated with the term *AI*.

The AI system is the core feature that allows OpenTTD to be used to research algorithms and hopefully simulate properties of the real world. The framework described in Chapter 3 is designed to make it straightforward to source, configure, and run such AIs, and all of the experiments of Chapters 4–6 use them.

2.7 Configurability

OpenTTD has over 400 options set through a combination of its graphical interface or a configuration file, and command line arguments; it is not useful to review them all here; see [45] for the complete list and their defaults. However, there are several that are particularly relevant for repeatable, reproducible, or replicable experiments.

- **Properties of the pseudo-randomly generated world**
  As suggested earlier, OpenTTD allows the configuration of the dimensions of the world: between 64×64 and 4096×4096 tiles. It also allows the specification of other related properties, such as if the generated world is flat or mountainous. Flat and mountainous worlds are compared in the experiments of Chapter 4.

- **Total number of ticks**
  OpenTTD allows the specification of the number of ticks taken before it exits. This allows experiments to run for a fixed amount of in-game time, rather than the default of indefinitely. Each of the experiments of Chapters 4–6 does this.

- **Random seed**
  As described earlier in this chapter, OpenTTD allows the specification of the random seed which controls how the world is generated at the start of play.

  Keeping track of the random seeds allows experiments to be bitwise repeatable, and including the random seeds with published results should help them be bitwise reproducible. Each of Chapters 4–6 includes the random seeds used.

OpenTTD also requires what's known as a *base graphics set*. While this does not affect gameplay, it affects the visuals of the game. OpenGFX [38] is an open source base graphics set that is installable through the OpenTTD interface.

It should be noted that while OpenTTD has a wide range of options, there is nothing in OpenTTD to help run it over a specific range of configurations, for example over two different configurations of worlds, or over a range of random seeds.

2.8 Extracting information

OpenTTD 13.4 and earlier can be configured to automatically save the state of the game in intervals of in-game time in files of a custom binary format that are hierarchical in nature. These *savegames* [sic] are designed to resume games after interruption, but they

are also rich sources of information for analysis, containing for example the complete transportation network each company has constructed, how much each company is worth, or how much money it has. OpenTTD 14.0 no longer has deterministic saving, but there is a way to restore this ready for OpenTTD 15.0.[2]

It is notable that no version of OpenTTD has a built-in ability to extract detailed information from these savegames in a form suitable for analysis, or indeed aggregate such information over multiple savegames in order to see changes over in-game time. Also in spite of their richness and deterministic nature, so far there is no evidence of existing research using savegames to extract information from OpenTTD; I suspect instead information was usually extracted manually through its graphical interface.

2.9 Conclusion

OpenTTD presents an extremely rich landscape for investigation; and because of its tick model, and the fact it accepts a random seed, this landscape is deterministic and so especially suitable for repeatable research. While its economic model is simplified or even absurd in parts, depending on the map size, there are hundreds of millions of possible actions an OpenTTD AI can take on any given tick of the game, and the existing research has only begun to explore this space.

If viewing OpenTTD not as a game for which it was designed, but as a tool for running repeatable, reproducible, and hopefully replicable research, there are three core features that are not present in OpenTTD. Firstly, there is no way to configure OpenTTD so it runs over a range of available configurations, for example over a range of random seeds. Secondly, while it saves information to its savegame files, if offers no way to extract and combine this information into a form ready for analysis. And thirdly, while OpenTTD is largely single-threaded, which contributes to its determinism and bitwise repeatability and reproducibility, this means it has no way to leverage multiple CPU cores. These are the core features that OpenTTDLab, as described in the next chapter, effectively add to OpenTTD.

[2] OpenTTD 14.0 changed how games were saved from intervals of in-game time to intervals of wall-clock time, which makes the savegames non-deterministic, and so makes it virtually impossible to use them to extract information from OpenTTD for bitwise repeatable experiments. However, as part of this project I successfully submitted a change to OpenTTD that allows the scheduling of so-called *console scripts* that can be used to restore the behaviour of OpenTTD 13.4 [10]. This change is scheduled for release in OpenTTD 15.0.

Chapter 3

OpenTTDLab: Its design and features

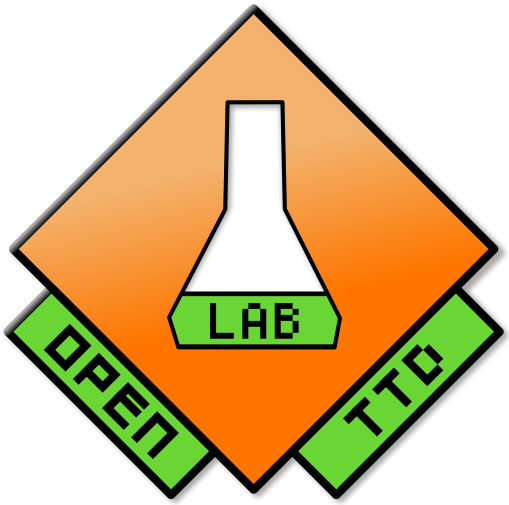


Figure 3.1: The OpenTTDLab logo. I created this based on the existing OpenTTD logo (licensed under GPLv2) [42], and it is displayed alongside the code of OpenTTDLab in its public documentation, as can be seen in Appendix B.

As discussed in Chapter 1, repeatability is the domain of the original researchers, while reproducibility and replicability are the domain of other researchers armed with the results of the original research. Given the different aims and audiences, it's not immediate that a single tool can help with all of these. However, the Python framework I created, OpenTTDLab, attempts to do this by adding the missing features described at the end of the previous chapter to help address the 7 issues detailed in Chapter 1. The current chapter details the design process I followed to create OpenTTDLab, and some of the resulting internal behaviours and user-facing features of the framework.

3.1 Design and creation process

I used a lightweight and highly-agile process to design and create OpenTTDLab; the high level features of this process are shown Figure 3.2. After I identified the problems to address, as discussed in Chapter 1, I conducted many cycles of writing documentation for code, writing code, and using the code, either to conduct the experiments of Chapters 4–6 or in tests; and each of these with tight cycles of evaluating the outputs of these activities and informally judging if the code does—or would if it existed—help produce research that is repeatable, replicable, and reproducible. Each of these activities would inform each other, again in tight cycles. I describe the process as *lightweight*

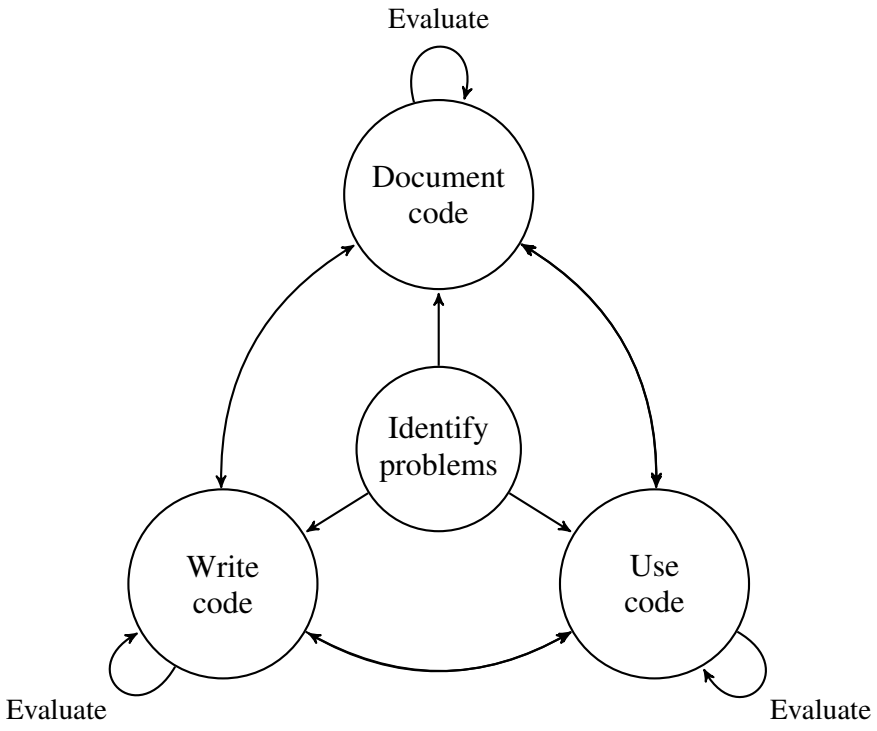


Figure 3.2: The process I followed designing and creating OpenTTDLab. After identifying the problems that I'm trying to address, I conducted cycles of documenting code (that may not have existed), writing the code, and using the code, all coupled with tight cycles of evaluating how it solved, or would solve, the problem of running experiments using OpenTTD that were repeatable, replicable and reproducible.

because were no *events* that are common to *Scrum*-like processes [21][1], and with one exception, no phases planned in advance as is often prescribed, for example for UK government services [35]. And I describe the process as *highly-agile* because in some cases

[1]From my own experience Scrum events are often known as *ceremonies*

iterations would take a low number of seconds, especially when writing documentation.

On the one exception to not having phases planned in advance: I did plan to create an initial version of light usage documentation very early in the project. Taking inspiration from Tom Preston Warner's *Readme driven development* [32] and Amazon's *Working Backwards* method [8], throughout the project, but especially at the beginning, I would write documentation for code without the code actually existing. This allowed me to imagine it being used, evaluate how well it would solve the problems I'm trying to solve, and change its design accordingly by quickly changing its documentation. Of course documentation for code that does not exist has very little value, so I moved quickly to writing and using code, mostly informed by the "Working software is the primary measure of progress" and "Deliver working software frequently" principles of the Agile Manifesto [4], which I judge to be the most useful principles of the Agile Manifesto, especially when working alone—albeit with advice from others. The other principles are less applicable—they focus on communication between different people involved in project.

I did not keep a precise record of the cycles; however, as a guide the GitHub repository that stores the history of code changes can be used [11]. I created 72 releases, which I labelled v0.0.1 through v0.0.72, 215 pull requests (PRs), and over 250 non-merge commits in the repository; and each of these I evaluated against the problems and desired properties of Chapter 1. I then carried on with more of the same activity, or another activity of Figure 3.2. A summary of the releases can be seen in Appendix A.

The first version, v0.0.1, was the result of a fork from Patric Stout's *OpenTTD-savegame-reader* [47], an existing Python project that extracts data from OpenTTD savegame files—as described in Chapter 2, these are files that save the state of play in OpenTTD that allows players to exit but resume at the same point later. This project could be be used to extract data from OpenTTD, and seemed a reasonably place to start. However, as mentioned in the same chapter, even with this code there was still no way to easily run OpenTTD over a range of configurations, or extract information from the savegame files in bulk, and no way to leverage multiple CPU cores.

Then, but still from very early in the process, v0.0.3, I created and maintained a small suite of tests for OpenTTDLab, asserting on its high level behaviour. However, the tests were not just for asserting on the behaviour of the code, they were also a special case of "Use code" in the design process of Figure 3.2; they allowed me to extremely quickly gauge the suitability of the design of OpenTTDLab, because to write these tests I used OpenTTDLab in a way very similar to the ways researchers would

use it. I also configured these test to run automatically on every change: a "very helpful" property for software used in reproducible research according to *The Turing Way* [48]. The tests essentially use OpenTTDLab to run experiments and assert on their results; they must be identical to previous runs of the experiments to pass, and so themselves give evidence that OpenTTDLab can be used to run bitwise repeatable experiments.

This leads to the main downside of this process: while the process allowed a high number of cycles of evaluations, all evaluation was from myself, from either running the code or, even worse, just imagining running the code. Being the creator I knew it extremely well and so was ill-placed to evaluate what amounts to its ease of use for people that don't know it as well: I'm essentially *marking my own homework*, and in some cases before it was even written. This is true when writing any code, but in this case this is especially true as there are different groups of people in the target audience: original researchers, reproducers, and replicators. User research was not included in the process in order to limit the scope of the project and to allow as many iterations as possible within the available time. However, while making OpenTTDLab I conducted the three different experiments of Chapters 4–6 in which I played multiple roles: I conducted original research and used OpenTTDLab to attempt to reproduce existing research, and so partially mitigated this downside.

3.2 Python and portability

I chose Python as the main language for both the internals of OpenTTDLab and its user-facing interface. I knew Python well, as mentioned earlier in this chapter a savegame parser for OpenTTD was available in Python [47], programs written in Python are mostly platform independent, and anecdotally it's a popular language in data science and data analysis—data analysis and visualisation can be performed in Python through libraries such as Pandas [28, 46] and Plotly [18]. To make sure OpenTTDLab did not depend on behaviour unique to a single Python version or a single platform, which would impede reproducibility of experiments especially [48], I made OpenTTDLab *portable*—hiding platform-specific details, and making sure that the tests mentioned in the previous section run and pass not only on multiple version of Python, but also on Linux, Windows and macOS. While OpenTTD itself is classed as portable, downloading, uncompressing, configuring, and starting OpenTTD are slightly different on each platform, so running tests on each is particularly valuable when considering reproducibility.

3.3 Feature 1: Running over a range of configurations

OpenTTDLab can run experiments on explicit versions of OpenTTD and OpenGFX over an arbitrary range of any of the 400 available options, an arbitrary range of random seeds, any OpenTTD AIs and their parameters, and all with the number of runs made clear. During the experiments a progress bar is shown to the user to show the approximate proportion of experiments complete at any given moment. This is all through a single Python statement, with no steps ahead of time needed other than the installation of Python and OpenTTDLab. This requires that under the hood OpenTTDLab downloads and caches OpenTTD, OpenGFX, and OpenTTD AIs and libraries they depend on, which are copied from GitHub, OpenTTD's TCP-based BaNaNaS [sic] content publishing service [36], or from the local file system. An example use of OpenTTDLab can be seen in Listing 3.1.

```
from openttdlab import run_experiments, bananas_ai

results = run_experiments(
    openttd_version='13.4',
    opengfx_version='7.1',
    experiments=(
        {
          'seed': seed,
          'ais': (
              bananas_ai('54524149', 'trAIns', ai_params=()),
          ),
          'days': 365 * 4 + 1,
        }
        for seed in range(0, 10)
    ),
)
```

Listing 3.1: Example usage of OpenTTDLab that runs the trAIns AI, automatically downloaded from BaNaNaS, for a single year of in-game time for random seeds 0 to 9.

Keeping to a single Python statement with no steps required ahead of time I found extremely convenient for repeatability, and I argue is important for reproducibility because it means only a small amount of Python code needs to be shared as an artifact, with no additional information required other than the version of Python and OpenTTDLab; although as discussed the version of Python should not affect the results.

3.4 Feature 2: Output as lists and dictionaries

As noted one of the missing features of OpenTTD, if viewing it as a simulator rather than a game, was the fact that data output was in a series of binary savegame files of a custom format: not in a format immediately suitable for analysis. I decided to convert all the savegame files from all the runs of OpenTTD into a single partially self-documenting data structure: a Python list of dictionaries.

This means that OpenTTDLab is agnostic to the specific way of how a user would prefer to analyse output; or in the language of Chapter 1 it is reusable in this respect. But as can be seen in Listing 3.2, only a single statement is required to convert a subset of the output into a Pandas *data frame*, a common first step when using Pandas to analyse data.

```
1  import pandas as pd
2
3  df = pd.DataFrame(
4      {
5          'seed': row['experiment']['seed'],
6          'date': row['date'],
7          'money': row['chunks']['PLYR']['0']['money'],
8      }
9      for row in results
10 )
```

Listing 3.2: Example usage of the results of OpenTTDLab, such as those generated in 3.1, converting them to a Pandas data frame.

An alternative would be to output Pandas data frame(s) directly from OpenTTDLab. However, the main downside of this approach is that I would have to design a structure for the data frame(s), but since OpenTTD savegame data is hierarchical there is no unique or one-size-fits-all approach to convert such data into such data frames. Any approach would have pros and cons, and to investigate these I decided was out of scope of the current work. For similar reasons I rejected saving the results to a non-ephemeral file such as SQLite: a tempting target since SQLite is a preferred format for datasets according to the US Library of Congress [13]. How to save data non-ephemerally is left to client code; for example a simple CSV format can be chosen to save a subset of the output, which is the approach I chose to persist the results of Chapters 4–6.

Early on in the design process, in v0.0.2, I considered structuring both the API

and the output to have a required *verification* step to make it easier to confirm that both configuration and results were bitwise identical to previous experiments. While this might have been particularly useful for determining if results have been bitwise reproduced, it made generating the results of Chapters 4 & 5 particularly awkward. Or in other words it seemed like this would negatively affect the reusability of OpenTTDLab, and for this reason I abandoned this approach.

3.5 Feature 3: Leveraging multiple CPU cores

OpenTTDLab leverages multiple CPU cores by implementing simple task-based parallelism: it starts a process per experiment, which runs OpenTTD, extracts data from its savegames, serialises it, and returns it to the controlling process. By default the number of processes is the number of CPU cores on the system. To further reduce time, a mechanism to reduce the amount of data returned by each process is available. These features I added to OpenTTDLab to help repeatability: specifically to run more repetitions of experiments in a given amount of time. A contributing reason was my own frustration while waiting for experiments to complete in the original single core version; I suspected that others using OpenTTDLab would feel similar frustration, and repeat experiments a fewer number of times.

3.6 Documentation

As discussed, writing documentation was part of the design process. However, since documentation is especially important for reusable software used for reproducible research [48], this documentation I repeatedly extended and published alongside OpenTTDLab, as it can be seen in Appendices A & B. In addition to usage instructions, examples, and API documentation, it includes recommendations relating to the repeatability, reproducibilty and replicability of work that uses OpenTTDLab; replicability is addressed by encouraging authors to include details of the algorithms used in OpenTTD AIs created as part of the work.

As part of the documentation process I made a logo for OpenTTDLab based on the OpenTTD logo, as can be seen in Figure 3.1. I did not have a rigorous justification for this, but I suspected that creating an identity for OpenTTDLab, especially one that was related to the identity of OpenTTD itself, would encourage the reuse of OpenTTDLab, and even encourage contributions to it.

3.7 OpenTTDLab and the 3Rs issues

The three features of OpenTTDLab address 6 of the 7 issues in existing OpenTTD research described in Chapter 1. Because of the low amount of Python code needed to run experiments and analyse results, it should be straightforward to include these artifacts in published work, or at least make them available separately and link to them (Issue 1). OpenTTDLab leverages multiple cores and so supports running more repetitions in a given amount of time, and sharing the Python artifacts shares how many repetitions were run (Issue 2). As can be seen from Listing 3.1, it is straightforward to run multiple experiments over the exact same amount of in-game time (Issue 3). There are no manual steps involved to download, configure, or run OpenTTD or extract data (Issue 4). The versions of OpenTTD, OpenGFX, and details of configuration are included in the artifacts (Issue 5), which also includes the random seeds used (Issue 6).

The issue of replicating results (Issue 7) is not directly addressed by features of OpenTTDLab. However, this is still addressed in two ways: as mentioned in the previous section, its documentation encourages authors that use OpenTTDLab to include explicit details of the algorithms created; and by the transitive nature of the 3Rs described in Chapter 1, helping repeatability and reproducibility indirectly aids replicability.

3.8 Conclusion

OpenTTDLab offers three features through a single Python statement: it allows a researcher to easily run OpenTTD over a range of configurations, including running ranges of AIs; it returns results from these experiments in a way that's suitable for analysis; and it leverages multiple CPU cores which allows more experiments to be run in a given amount of time. These are all useful for repeating experiments; if artifacts are shared, then reproducing experiments; and if the algorithm of OpenTTD AIs created are shared as encouraged by the OpenTTDLab documentation, also replicating experiments.

In the following chapters I give more direct evidence of OpenTTDLab's usefulness. In Chapter 4, I play the role of reproducer, using OpenTTDLab to attempt to reproduce existing research. In Chapter 5, I play the role of original researcher, creating a simple OpenTTD AI, provide enough information for the results to be replicated, and suggest that OpenTTD is a compelling landscape for simulation and worthy of further research. In Chapter 6, I provide evidence that the parallelism features added actually work, and argue that these help repeatability.

Chapter 4

Experiments 1: Reproducing results

To attempt to validate OpenTTDLab as a useful framework, I used it to attempt to reproduce some of the results of Rios and Chaimowicz [34]: a set of experiments running two AIs: trAIns [sic] being presented in the aforementioned work, and the pre-existing AdmiralAI; and I compared their company values. I generated results, and while trAIns does appear to perform better than AdmiralAI as the original results suggest, the performance of trAIns does not match that of the original work.

4.1 Experimental setup

The setup was a best-effort attempt to match what was described as the setup of the experiments in the original work: the starting year was set to 1960, the map was configured to be 512×512 with a very low amount of seas, industry density and number of towns set to normal, the economy set to be able to fluctuate, subsidy multiplier of $\times 2$, disasters enabled, tolerance of town councils to changes set as tolerant, both vehicle running costs and construction costs set to medium, vehicle breakdowns disabled, trains set to only be able to turn around at the end of the line as opposed to also at stations, the maximum initial loan was set to £300,000, and the initial interest rate set to 3%.

Note that these settings are not all explicitly mentioned in the original paper: it describes running OpenTTD revision 16724 using the *medium* difficulty level with some overrides. However, due to savegame format differences OpenTTDLab does not work under this version, and the recent versions of OpenTTD that do work under OpenTTDLab no longer have the concept of difficulty level. However, the historical meaning of the medium difficulty level is documented [40], and so the explicit list of aforementioned settings recreated, and run under OpenTTD 13.4 and OpenGFX 7.1

that OpenTTDLab does work under. There was one exception to basing off the medium difficulty level: the medium difficulty level was documented to have a maximum initial loan of £150,000, but using this level in pilot experiments resulted in extremely poor performance of AdmiralAI—it appeared to be virtually unable to function.

None of the versions of the AIs or their dependencies were specified in the original paper. In the current experiments, the version of trAIns is 2.1, the version of AdmiralAI is 25, and the version of AdmiralAI's single dependency, Queue.FibonacciHeap, is 2. All the AIs and their dependencies were retrieved from BaNaNaS.

The results presented here are for exactly 15 years of in-game time, while in the original work the time varied from 15 years and 2 days to 15 years and 11 months. In both the original paper and here, there were two sets of experiments performed: one on the *flat* terrain type, and one on the *mountainous* terrain type. Here each set contains 64 repetitions, and in the original paper each set contains 7 repetitions. Here random seeds 0 to 63 were used for each set of experiments, but in the original paper the random seeds are unknown.

The experiments here were run on mac OS 14.5 running on a 2020 M1 and Python 3.11.0 and OpenTTDLab 0.0.72. The OS and CPU of the original experiments are unknown. The experiments here were run with OpenTTDLab's default number of workers—the number of cores on the machine—which in this case was 8.

The Python code used to run the experiments is in Appendix C, and the code to subsequently analyse the results in Appendix D.

4.2 Results

The results of the experiments can be seen in Figures 4.1 & 4.2. Figure 4.1 shows the final distributions of company value for the two sets of experiments for each of trAIns and AdmiralAI, comparing the results using OpenTTDLab to the results of the original paper. Figure 4.2 show how the distribution of company value changes over time for the two sets of experiments and two AIs, but only for the experiments of OpenTTDLab—the original paper did not include this level of detail.

From Figure 4.1 it is clear that while trAIns performed better than AdmiralAI in both OpenTTDLab and original experiments, the difference between the two AIs in the current experiments is slight, as opposed to the original paper when the difference was remarkable. While AdmiralAI's performance is similar or slightly worse than what was originally reported, especially in the mountainous configuration, trAIns can be

characterised as approximately an order of magnitude worse. It can also be seen that in the current experiments there are more cases that can be classed as catastrophic—in the sense that the company value remains approximately zero—across both trAIns and AdmiralAI.

The results over time of Figure 4.2 show that for both trAIns and AdmiralAI the distribution of company value becomes increasingly wider with time. They also show that while the median value for trAIns is slightly higher than for AdmiralAI, the distribution of company value for trAIns is much wider than that for AdmiralAI, especially on flat terrain.

From Figures 4.1 & 4.2 it can be seen that in both the original and current experiments, both AIs appeared to perform better on flat terrain than on mountainous.

4.3 Discussion

The only aspect of the results that can be easily explained is that the experiments on mountainous terrain resulted in lower company values than those on flat terrain. As discussed in Chapter 2, railways on mountainous terrain require more investment, for example by raising and lowering land or constructing more rail to go around obstacles, and there is nothing in the economic model of OpenTTD to offset this by subsequent high income. In fact it is the opposite: trains would take longer to travel between stations, due to either going slower when going uphill or round corners, which increase the time to transfer goods and so reduces income compared to an equivalent route but over flat terrain.

The other aspects of the results are currently unexplained, but I speculate reasons for some of them. For example, the increasingly wider distributions hint at a feedback loop where early strong performance increases the chance of later even stronger performance, and similarly for weaker performance. The only difference between the games in each set of experiments is the random seed that effects world generation, which suggests that something in the generated world results in these vicious and virtuous cycles. Further analysis of the results would show if there is indeed a correlation between early and late performance; this is not included here for brevity. If such a correlation were found, this would hint at an evidence-based route for improving the AI: beyond simply flat or mountainous, there could be other properties of the generated worlds that lend themselves to better or worse performance; finding out exactly what those are could lead to better AIs. However, while the fine-grained details of the map are in savegame

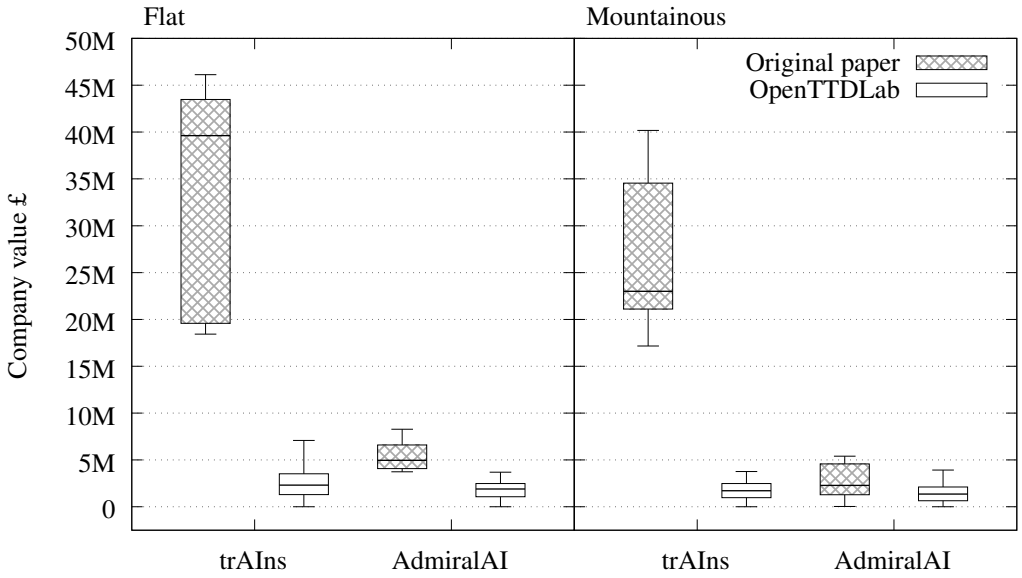


Figure 4.1: Distributions of company values for trAIns and AdmiralAI at the end of experiments on flat and mountainous terrain, comparing the results as reported by the original work of Rios and Chaimowicz [34], and those from experiments run with OpenTTDLab configured to attempt to replicate the original work. The boxes show the median and upper and lower quartiles, and the whiskers show the maxima and minima.

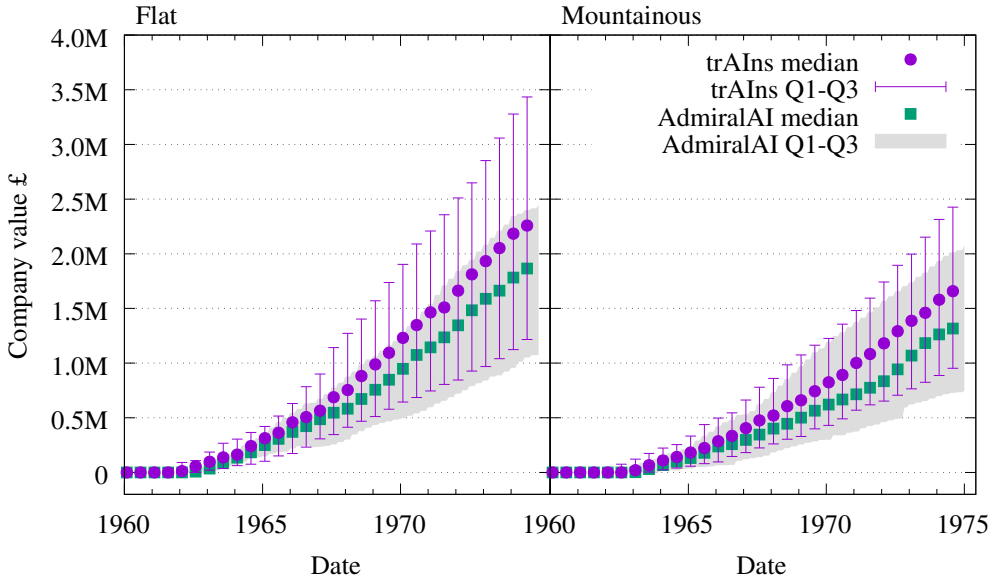


Figure 4.2: How the distributions of company value for trAIns and AdmiralAI change over time for experiments on flat and mountainous terrain using OpenTTLab, configured to attempt to replicate the results of Rios and Chaimowicz [34].

files, extracting these is not currently a feature of OpenTTDLab.

In terms of the seemingly increased prevalence of catastrophic results, it is has been suggested that one of the barriers to reproducibility of simulations is the under reporting of errors [26]; indeed, there are no mention of errors in the original study. It could also be that the number of repetitions in the original study is much lower, 7 per configuration rather than 64, and so just by chance resulted in better results than found here; although this does seem unlikely given the distributions of results between the original and current work in most cases did not overlap. Unfortunately, because the random seeds were not included in the original study, the possibility of *seed hacking*[1] cannot be excluded from the list of possible reason for the different results.

However, the most likely cause of differences are not deliberate, but in fact due to the differences in version of OpenTTD, its configuration, and the AIs. If this is indeed the case, then the worst thing the results here suggest is that that trAIns is less robust than AdmiralAI to whatever the consequences of these differences.

Because of the code provided in Appendices C & D, which due to the design of OpenTTDLab is not long or complex, it should be straightforward for researchers to replicate the results here, and begin to answer some of these unanswered questions by conducting further analyses of the results.

4.4 Conclusion

It is beyond the scope of the current work to go beyond speculation and fully explain the results. However, these experiments have shown three things. Firstly, they give stronger evidence to the claims of Chapter 1 that it is difficult to reproduce existing research that uses OpenTTD. Secondly, they show that OpenTTDLab can be used to extract rich information from OpenTTD, richer than the information available in existing OpenTTD research in at least this case. And thirdly, as can be seen in Appendices C & D, OpenTTDLab allows this to be done with a small amount of code that should be bitwise reproducible, and so would not just confirm the results, but allow them to be furthered through more in-depth analysis without even the full results being shared.

The results here were also for a single metric: company value; however, as mentioned in Chapter 3, there is a wealth of further information available in OpenTTDLab. The exception is fine-grained detail of the map: these are not extracted from the save-game files; adding this feature is left to further work.

[1] Seed hacking in this context would be an example of p-hacking or data dredging.

Chapter 5

Experiments 2: Simulating a network

The previous chapter showed that OpenTTDLab makes it straightforward to compare existing AIs, and its richness of output allows informed speculation as to the reasons for their behaviour differences, and all in a bitwise repeatable and reproducible way. However, this doesn't give much evidence that OpenTTD, run using OpenTTDLab, can be used a a simulator to explore real-world effects. In this chapter I present results of running a simple parameterised AI designed to explore the results of its parameter changing. I find that OpenTTD shows some effects that align with intuitive expectations of the real world, and suggest that further work to validate OpenTTD as a simulator would be valuable. And by using OpenTTDLab the results are repeatable, by providing the code of the AI the results should be replicable, and by providing a high level algorithm for this AI, the results should be replicable.

5.1 Experimental setup

I constructed an OpenTTD AI for this experiment: on startup it chooses the 2 largest towns, finds a route between them using the A* algorithm, builds a road over this route, a depot and station at each end, and builds a configurable number of buses that carry between the two stations—configured for 1, 2, 4, 8, and 16 buses. Each run was for 50 years, and each repeated for random seeds between 0 and 49.

The algorithm can be seen in Algorithm 1, the Squirrel code of the AI in Appendices F & G, and a basic Python regression test that uses OpenTTDLab in Appendix H.

The experiments were run on mac OS 14.5 running on a 2020 M1 and Python 3.11.0, OpenTTDLab 0.0.72, OpenTTD 13.4, and version 4 of the Pathfinder.Road library that finds road routes between tiles using the A* algorithm, configured with a turn cost of

Algorithm 1: Simple parameterised OpenTTD AI

```
Data: Random seed, the number of buses to build in the AI N
OpenTTD generates the map deterministically based on the random seed ;
Choose the 2 biggest towns ;
Set the cost of turns in the route finding algorithm to 5000 ;
while not found route do
    Continue to find route using the A* algorithm ;
end
foreach step in route do
    Build step;
end
Build a depot at each end of the route;
Build a station at each end of the route;
foreach bus in buses do
    Build the fastest available bus;
    Give the bus orders to move between the 2 towns;
end
```

5000. The experiments were run with OpenTTDLab's default number of workers—the number of cores on the machine—which in this case were 8. The Python code to run the experiments is in Appendix I and to analyse the results in Appendix J.

5.2 Results

As can be seen in Figure 5.1, there appears to be a clear trend in terms of mean money in the bank at the end of the 50 years: the more buses, the more money in the bank. However, before approximately 1975 the trend was the other way around: from starting with £100k, this decreased for all configurations, but the more buses the less money in the bank. Interestingly at approximately 1975 there is what seems to be a rough convergence point: all configurations have approximately the same amount of money. There also appears to be a small decrease just before the year 2000.

How the distributions of money in the bank change over time for the experiments with one bus compared with 16 buses can be seen in Figure 5.2. For the majority of the time range the distribution for 16 buses is strictly wider than that for 1 buses; although towards the end of the time range the lower ends converge.

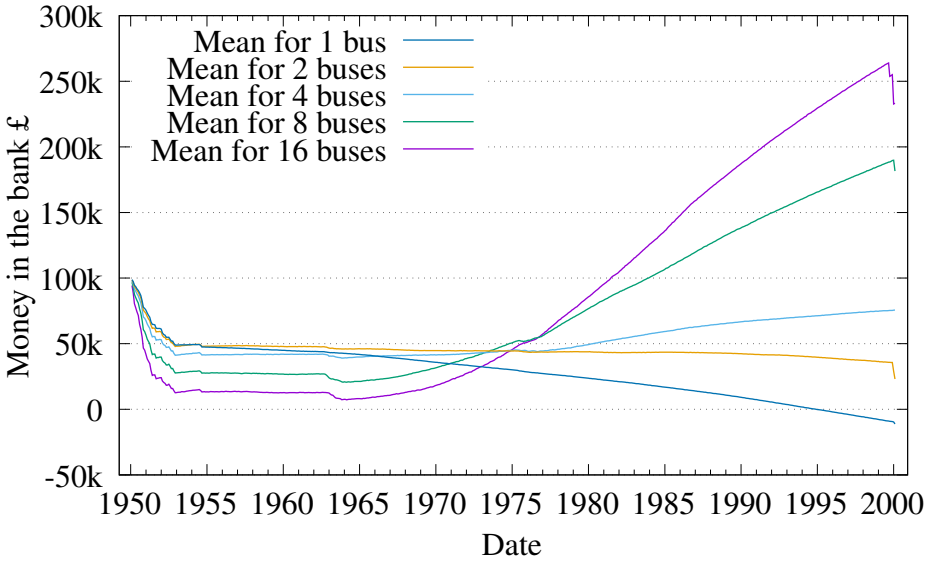


Figure 5.1: How the mean money in the bank changes over time for the AI of Algorithm 1 when run for 1, 2, 4, 8, and 16 buses.

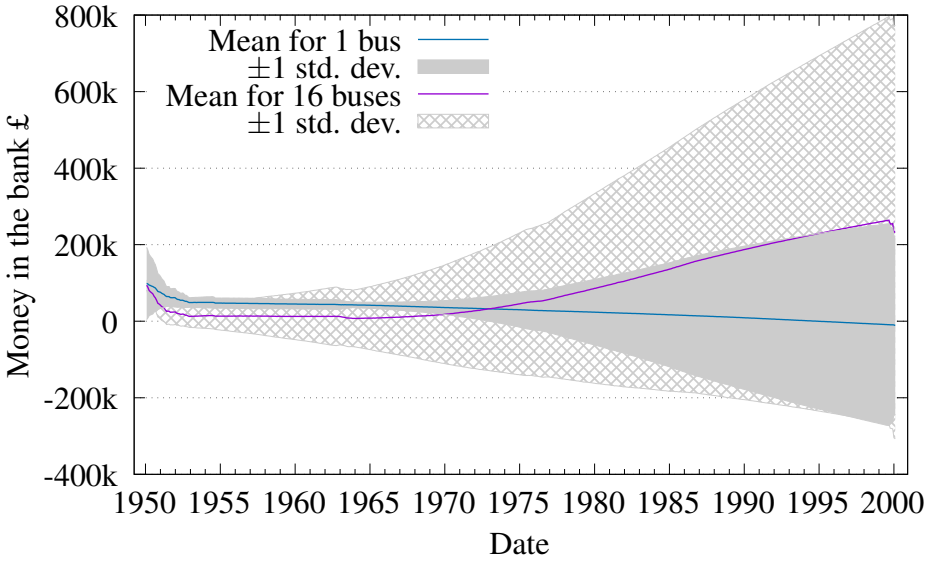


Figure 5.2: How the distribution of money in the bank changes over time for the AI of Algorithm 1 when run for 1 and 16 buses.

5.3 Discussion

The results appear to show two things. Firstly, the flip of trends between 1975 and after 1975 suggests that in this situation, investing more money in the short term can lead to more money in the bank longer term. Secondly, the overlapping distributions suggest that the situation is more complex: while the mean for 16 buses is eventually higher, for most of the experiment the lower side of the distribution is lower than for 1 buses, which can be interpreted as there being a greater risk of making less money with 16 buses than on 1. These results mean that OpenTTD, via OpenTTDLab, can be used to investigate the longer term consequences of shorter term investments, and can be used to investigate risk-benefit trade offs.

Unfortunately the convergence point around 1975 is so far unexplained. Similarly the dip in money at around the year 2000 is also unexplained. Town growth and bus break downs I speculate could be contributing factors to these—further analyses with the data provided by OpenTTDLab could answer if this is the case.

These are of course limited results, with no reference or comparison to any formal economic models. The AI constructed is extremely simple, and the infrastructure it creates and uses borderline does not meet the definition of a network. But in spite of this simplicity, the results show some interesting effects that appear to have real-world meanings, and I would argue interesting enough for further research. And research where, with quite a small amount of code as can be seen in Appendices I & J, OpenTTDLab should be able to be used.

5.4 Conclusion

Through simulating basic transportation networks using OpenTTDLab, OpenTTD can produce some compelling results that appear to have real-world meanings. As described in Chapter 2, OpenTTD's economic and transportation models are limited, so it remains to be seen if such results are valid in terms of simulating the real world. However, even if this is not the case, OpenTTDLab can be used to research algorithms: in this case the effects of changing a single parameter of an algorithm has interesting effects in the world of OpenTTD. Also the Python regression test, having run and passed several times over the course of this work, itself provides evidence that such research is repeatable. And by providing this specific algorithm as pseudocode, it should also be possible for the results to be replicated.

Chapter 6

Experiments 3: Scaling

As mentioned in Chapter 3, in order to make it quicker to repeat more experiments, I added two features to OpenTTDLab: simple task-based parallelism and a mechanism to reduce the amount of data copied between the worker and the main process. To validate these features, and to start to investigate how OpenTTDLab could be further improved on these fronts, I ran a set of basic experiments: measuring how the wall-clock runtime of OpenTTDLab changes with the number of worker processes, and how it changes with the amount of data passed between each worker and the controlling process.

6.1 Experimental setup

All the simulations were conducted on a 2020 Apple M1 with 8 CPU cores, 16GB of RAM, OpenTTDLab 0.0.72, OpenTTD 13.4, Python 3.11.0, trAIns 2.1 retrieved from BaNaNaS, and running for seeds 0 to 49, each for a total of 1465 in-game days—just over 4 in-game years. The Python code to run the experiments can be seen in Appendix K, and the code to subsequently process and visualise the results in Appendix L.

The number of worker processes was varied between 1 and 8, and the amount of data was either all the data extracted from the OpenTTD savegame files, and what I class is a minimal set of data to be useful: the date, the company value at that date, and if an error occurred.

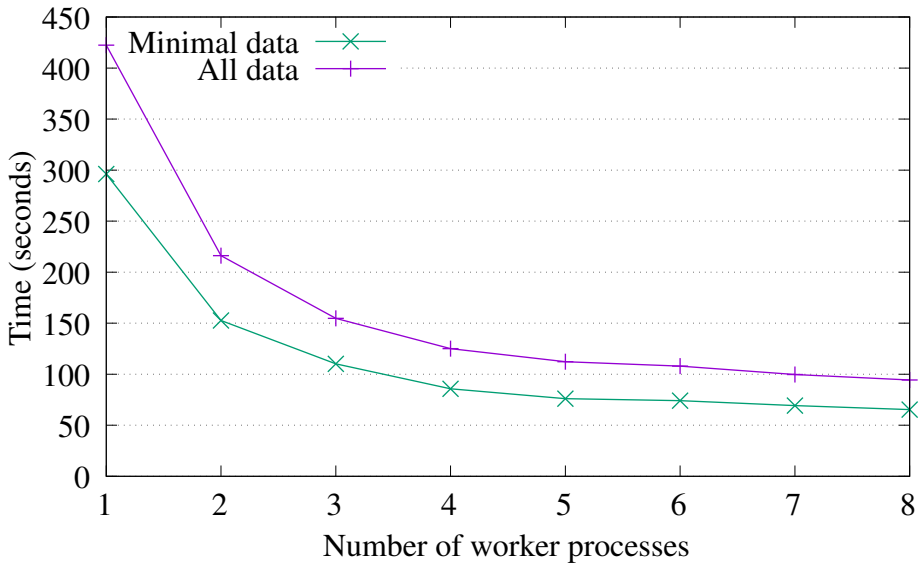


Figure 6.1: Total wall-clock time of when running 50 experiments using OpenTTDLab between 1 and 8 worker processes, comparing returning all data to the controlling process with returning a minimal amount of data.

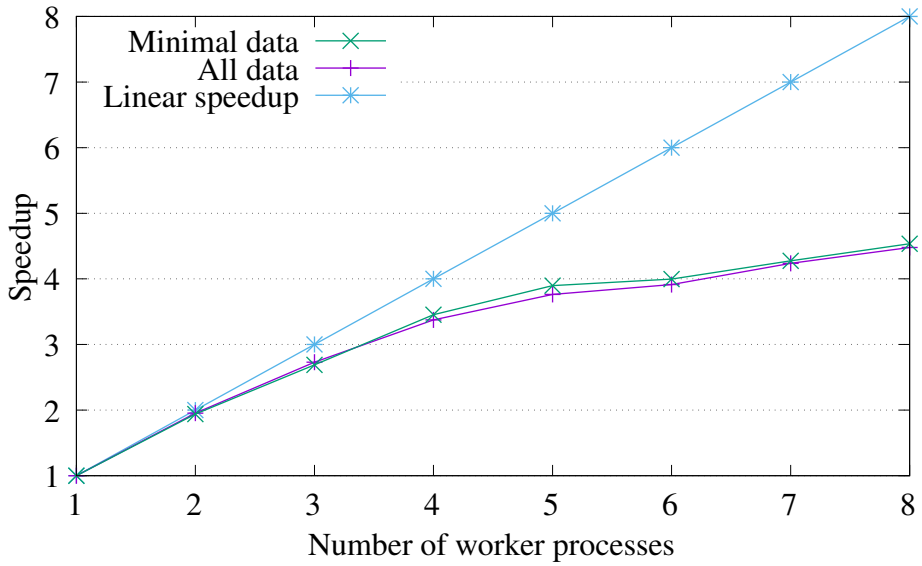


Figure 6.2: Speedup when running 50 experiments using OpenTTDLab between 1 and 8 worker processes, comparing returning all data to the controlling process with returning a minimal amount of data, and comparing with linear speedup.

6.2 Results

As can be seen in Figure 6.1, wall-clock time decreases as the number of processors increases, and so there is some useful parallelisation: running with 8 worker processes results in an approximate 80% reduction of runtime compared to running on 1 process. Also, returning a minimal set of data from the worker resulted in approximately an additional 30% reduction in time. This suggests that it's worth running experiments on multiple workers, and to reduce the amount of data returned from the workers, especially if running a high number of experiments.

However, Figure 6.1 also suggests that there is an aspect of diminishing returns: for example the difference in runtime between 1 and 4 processes is much greater than the difference between 5 and 8. To better show this effect, Figure 6.2 shows *speedup*: how many times faster the program runs for a given number of worker processes compared to the time it takes running on a single worker process. The speedup for the experiments does increase with each addition of a process. However, the greater the number of processes, the greater the deviation from linear speedup. It gets especially worse from 5 processes onwards, ending with a speedup of approximately 4.5 with 8 processes. Figure 6.1 also shows that the speedup profiles of returning all data and minimal data from the worker processes are almost identical.

6.3 Discussion

A reason for the increased deviation from linear speedup when the number of processors is greater than 4 could come from the fact the cores are not identical. According to Apple [17] 4 cores are high performance, but the other 4 are low performance (but have higher efficiency in terms of power usage). The speedup profiles appear roughly consistent with the high/low performance split, and suggests that the high performance cores are being used in preference to the low performance cores.

Another reason for the deviation from linear speedup is Amdhal's Law: it describes how speedup is related to proportion of the program that runs in serial. This law is usually formulated as

$$S_p = \frac{1}{(1-p) + \frac{p}{s}}$$

where S_p is the speedup of the entire program, p is the proportion of the program that has been parallelised, and s is speedup of the parallelised part of the program. The most relevant consequence of this is that $\lim_{S\to\infty} S_p = 1/(1-p)$. This means that,

for example, even if 95% of a program is parallelised, then the maximum speedup is $1/(1-0.95) = 20$, even if thousands of processes are used.

This suggests that a way of increasing this limit is to reduce the proportion of runtime in serial code. However, in light of Amdhal's law the identical speedup profiles in Figure 6.2 suggest that changing the amount of data returned from the worker processes does not change the overall proportion of runtime in serial code. Given wall-clock time reduces with less data transferred, it's reasonable to suspect that reducing the data transferred reduces parallel and serial runtimes in similar proportions, possibly from the serialisation and deserialisation involved. Thus removing all data transfer from worker processes is not likely to improve scaling. One way to remove all data transfer is to use threads rather than processes, but since threads in Python are subject to the Global Interpreter Lock (GIL), and so in many cases would increase the proportion of serial code compared to using processes, there is a strong reason to suspect that this would make scaling worse rather than better.

A reasonable next step would be to investigate what else in the program could be running serially. This is potentially not a trivial project—for example there might be serial code in what appears to be parallel sections because it accesses a shared resource under contention from the other processes, perhaps even indirectly. As examples, there some types of memory caches that are shared between cores, or reading or writing to disk may also effectively be done serially, or at least not fully parallel up to the number of cores. To understand and improve upon the current behaviour this would need a more in-depth understanding of the computer architecture used, and profiling of the Python code, or possibly the code of OpenTTD itself, to understand where time is being spent. This is beyond the scope of the project and so is left to further work.

6.4 Conclusion

The conclusions that can be drawn here are that OpenTTDLab can utilise multiple cores to reduce wall-clock runtime of experiments that have multiple repetitions, albeit with imperfect scaling; and even though reducing the amount of data returned from the workers does not improve scaling, it can also be an effective way to reduce wall-clock runtime. Both of these features should help repeatability by allowing more repetitions of an experiment to run in a given amount of time.

Chapter 7

Discussion

As discussed in Chapter 1, OpenTTD has been used to run experiments to research a variety of algorithms. However, from my own review not a single such study has been found that does not appear to have problems in at least one of the 3Rs of repeatability, reproducibility, or replicability; I classed these problems into 7 issues. Given the wider replication crisis in many fields of science, this is a trend that should not continue. The framework I created, OpenTTDLab, seeks to address these 7 issues by making it straightforward to repeat experiments using OpenTTD; if artifacts are shared, straightforward for others to reproduce them; and if details of the algorithms of AIs are shared, straightforward to replicate them. While OpenTTDLab is useful in terms of the 3Rs, authors must take separate steps when running experiments and publishing results, and the documentation of OpenTTDLab makes suggestions on these fronts.

OpenTTD itself provides a rich and interesting setting for studies of algorithms. Its Squirrel-based AI framework, coupled with its tick-based and so deterministic simulator as described in Chapter 2, provides an excellent foundation for bitwise repeatable and reproducible studies, and if presented with sufficient detail, replicable studies as well. While the supply chain of OpenTTD is clearly a simplification of reality, and its economic model is unrealistic, it has been suggested there are realistic aspects to OpenTTD in terms of network effects [33]; I did not attempt to validate OpenTTD as a simulator, but I have found nothing to contradict this suggestion. If OpenTTDLab indeed provides a good framework for the 3Rs as I claim it does, it puts further research that uses OpenTTD as a simulator of real-world networks in good stead.

From the first set of experiments, those in Chapter 4, I have given stronger evidence of the difficulty of reproducing OpenTTD results: I successfully generated results, but they did not fully match the ones originally reported. However, by providing the

artifacts for running experiments and analysing their results, I have shown that by using OpenTTDLab it is straightforward to repeat experiments, and argue that others should be able to bitwise reproduce the results I have.

Through the second set of experiments, in Chapter 5, I have again shown again that OpenTTDLab can be used to repeat experiments. But going further, the results of these experiments suggest that OpenTTD could be used to simulate real-world properties relating to transportation, business, or logistics: by parameterising the AI written for the experiments, my results appear to show risk-benefit trade offs and the suggestion that more money initially invested can result in higher returns later. This is through an extremely simple AI that takes an extremely narrow range of the possible actions an OpenTTD AI can take: connecting just two towns and transporting once cargo type with one transport type. Given the much more complex chains possible described in Chapter 2, I strongly suspect there is a wealth of insights waiting to be discovered by more complex AIs that take a wider range of actions.

The third set of experiments in Chapter 6 show that OpenTTDLab can run experiments in parallel by leveraging the multiple CPU cores available on a single computer, albeit with poor scaling. This allows for experiments to be repeated more times within the same time constraints, and so allows research with results that have more evidential weight behind them than they otherwise would.

These three sets of experiments are different: one attempts to replicate an existing study that compares two AIs, one parameterises a newly written AI, and one investigates the scaling properties of OpenTTDLab itself. These differences themselves provide evidence that OpenTTDLab is reusable, and by being reusable increases the reproducibility of studies that use it [5]. For all three of the experiments, the artifacts to run the experiments and to analyse their results are included in appendices, and because they delegate to OpenTTDLab, are relatively short in terms of number of Python statements. This suggests that it should be feasible to include such artifacts in published studies, or at least provide permanent links to them published separately, again showing that OpenTTDLab should be able to be used in future reproducible work.

I developed OpenTTDLab using the highly agile mechanism described in Chapter 3, iteratively writing and using the framework to create the results of Chapters 4–6 while attempting to avoid the 7 issues described in Chapter 1; this process itself provides a good reason to believe it is fit for purpose and achieves the aims of helping to achieve the 3Rs. However, according to ACM's definitions, the 3Rs are the domain of multiple sets of authors: those originally conducting the research and those then attempting

to reproduce or replicate it. To fully validate that OpenTTDLab can be a useful tool in these endeavours, others will have to use it to conduct research, and its results reproduced and replicated by other authors, much like the systematic attempts reported by Luijken et al. [26]. I have provided three sets of experiments in this work, and argued that they should be reproducible or replicable, but the true test of these properties are real attempts by other authors. However, as mentioned in Chapter 1, on the replication front there are "no set criteria to assess the alignment of replicated simulation results with the original results" [26], so exactly what this would look like for replication is not known. However, since repeatabilty and reproducibility are de-facto precursors to replication, OpenTTDLab is valuable for replication in spite of this uncertainty.

OpenTTDLab also appears novel: I was unable to find any project takes an existing game created for recreation, and augments it into a reusable simulator that allows ranges of algorithms to run without human involvement and results extracted. The closest I found was Axelrod [22], a framework for investigating the Iterated Prisoner's Dilemma, created in response to similar reproducibility issues identified in its respective literature to the issues described in Chapter 1. I discovered Axelrod towards the end of the current work, but there are a number of similarities to OpenTTDLab: both use Python, both includes tests that run on every change, both leverage multiple CPU cores, and both focus not only reproducibility but also on reusability by allowing future users to use it with new code. A difference is that Axelrod includes functions to visualise results, but OpenTTDLab supports them only by returning the data to be visualised separately. A similar set of functions to generate visualisations would be a worthwhile addition to OpenTTDLab in order to ease analysis. However, given visualisations can be already be produced from the output of OpenTTDLab, this is by no means vital.

There are many other ways that OpenTTDLab can be improved. For example, OpenTTDLab does not extract the map in tile-by-tile detail; if it did, further ranges of analyses would be possible, especially those to study network effects. And while the results returned from OpenTTDLab I describe as partially self-documenting, and there are examples presented in this work and in the OpenTTDLab documentation, in general the meanings of the values returned are not completely clear, and no standalone exhaustive documentation of what is extracted from OpenTTD using OpenTTDLab currently exists. In addition, the results from OpenTTDLab are ephemeral: they exist only in memory and designing a format to persisting them to disk I deemed outside the scope of the project because how to do so is not trivial. Axelrod has done this, but the data structures involved in the Iterated Prisoner's Dilemma are much simpler.

Chapter 8

Conclusion

While there are many improvements that can be made in OpenTTDLab, especially around output and visualisation, and further investigations performed in terms of whether it does in fact achieve its aims, through this work I have provided evidence that OpenTTDLab provides the foundations of a reusable framework for conducting repeatable, reproducible, and replicable research. And through just simple experiments, I provided evidence that OpenTTD can produce interesting results that appear to have real-world meanings.

I am especially pleased with the fact that OpenTTDLab offers rich behaviour through just a single Python statement. It removes the need for manual and error prone steps such as creating ranges of configuration files or command line options. It handles what I think is the fairly technically complex aspect of downloading from BaNaNaS—there is no straightforward API to do this, and no equivalent client outside of OpenTTD itself; indeed a number of the 72 releases of OpenTTDLab focused on this problem. It saves what I think would be a lot of time to potential users, so they should be able to get on with the actual research they care about on algorithms or networks, rather than worrying about downloading AIs, clicking through a graphical interface, or gathering and parsing binary files. And this is all a way that addresses the replication crisis rather than continuing to contribute to it.

OpenTTDLab is already a useful tool, and I am excited about what it can become in future if it's used and contributed to further, which I sincerely hope it will be.

https://github.com/michalc/OpenTTDLab

Bibliography

[1] Association for Computing Machinery. *Artifact Review and Badging Version 1.1*. 24th August 2020. URL: https://www.acm.org/publications/policies/artifact-review-and-badging-current (visited on 15th October 2023).

[2] Association for Computing Machinery. *New Changes to Badging Terminology*. 24th August 2020. URL: https://www.acm.org/publications/badging-terms (visited on 15th October 2023).

[3] Monya Baker. '1,500 Scientists Lift the Lid on Reproducibility'. In: *Nature* 533.7604 (25th May 2016). DOI: 10.1038/533452a.

[4] Kent Beck et al. *Manifesto for Agile Software Development*. 2001. URL: https://agilemanifesto.org/ (visited on 3rd August 2024).

[5] Fabien C. Y. Benureau and Nicolas P. Rougier. 'Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions'. In: *Frontiers in Neuroinformatics* 11 (4th January 2018), p. 69. DOI: 10.3389/fninf.2017.00069.

[6] Geert Beuneker. 'Autonomous Anomaly Detection in Games'. Master's thesis. The Netherlands: Utrecht University, July 2019. HDL: 20.500.12932/32972.

[7] Frank Bijlsma. 'Evolving Dynamic AI Opponents for OpenTTD Using Dynamic Scripting and Grammatical Evolution'. Master's thesis. The Netherlands: Utrecht University, June 2014. HDL: 20.500.12932/17598.

[8] Colin Bryar and Bill Carr. *Working Backwards: Insights, Stories, and Secrets From Inside Amazon*. Pan Macmillan, 2021.

[9] Department for Business and Trade. *Tariffs to trade with the UK from 1 January 2021*. Version v4.0.2046. 2024. URL: https://www.data.gov.uk/dataset/3bee9a8a-e69c-400e-add5-3345a87a8e25 (visited on 11th August 2024).

[10] Michal Charemza. *OpenTTD PR - 'Add: [Console] schedule command to execute a script file later'*. 1st July 2024. URL: https://github.com/OpenTTD/OpenTTD/pull/12761 (visited on 11th August 2024).

[11] Michal Charemza. *OpenTTDLab: A Python framework for running reproducible experiments using OpenTTD*. Version 0.0.72. 16th June 2024. URL: https://github.com/michalc/OpenTTDLab.

[12] Christian Collberg and Todd Proebsting. 'Repeatability in Computer Systems Research'. In: *Communications of the ACM* 59 (25th February 2016), pp. 62–69. DOI: 10.1145/2812803.

[13] United States Library of Congress. *Sustainability of Digital Formats: Planning for Library of Congress Collections (SQLite, Version 3)*. 9th May 2024. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000461.shtml (visited on 25th August 2024).

[14] Olivier Dalle. 'On Reproducibility and Traceability of Simulations'. In: *Proceedings of the 2012 Winter Simulation Conference (WSC)*. IEEE. 9th December 2012, pp. 1–12. DOI: 10.1109/WSC.2012.6465284.

[15] Mattias Hansen and Bobby Murphie. 'Teaching Concurrency in a Modern Manner, Flipped Classroom or Game-Based Learning'. Bachelor's thesis. Sweden: Malmö University, May 2018. URN: urn:nbn:se:mau:diva-20377.

[16] David R. C. Hill. 'Reproducibility of Simulations and High Performance Computing'. In: *ESM 2022, European Simulation and Modelling Conference*. October 2022, pp. 5–9. HAL: hal-04045230v2.

[17] Apple Inc. *M1 Overview*. July 2021. URL: https://www.apple.com/ua/business/mac/pdf/Apple-at-Work-M1-Overview.pdf (visited on 25th June 2024).

[18] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly (visited on 15th August 2024).

[19] John P. A. Ioannidis. 'Why Most Published Research Findings Are False'. In: *PLOS Medicine* 2.8 (30th August 2005), e124. DOI: 10.1371/journal.pmed.0020124.

[20] M. H. Jiang et al. 'A Mirroring Architecture for Sophisticated Mobile Games Using Computation-Offloading'. In: *Concurrency and Computation: Practice and Experience* 30.17 (6th April 2018), e4494. DOI: 10.1002/cpe.4494.

[21] Jeff Sutherland Ken Schwaber. *The Scrum Guide*. November 2020. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf (visited on 24th May 2024).

[22] Vincent Knight et al. 'An open framework for the reproducible study of the Iterated Prisoner's Dilemma'. In: *Journal of Open Research Software* 4.1 (2016), e35. DOI: 10.5334/jors.125.

[23] Geert Konijnendijk. 'MCTS in OpenTTD'. Bachelor's thesis. The Netherlands: Maastricht University, June 2015. URL: https://project.dke.maastrichtuniversity.nl/games/files/bsc/Konijnendijk_BSc-paper.pdf.

[24] Ondřej Lakomý. 'Railroad Network Planning in Open Transport Tycoon Deluxe'. Master's thesis. Czechia: Charles University, January 2020. HDL: 20.500.11956/116805.

[25] Chiung-Lin Liu. 'Using a Video Game to Teach Supply Chain and Logistics Management'. In: *Interactive Learning Environments* 25.8 (21st October 2016), pp. 1009–1024. DOI: 10.1080/10494820.2016.1242503.

[26] K. Luijken et al. 'Replicability of Simulation Studies for the Investigation of Statistical Methods: The RepliSims Project'. In: *Royal Society Open Science* 11.1 (17th January 2024), p. 231003. DOI: 10.1098/rsos.231003.

[27] Robert Marmorstein. 'Teaching Semaphores Using... Semaphores'. In: *Journal of Computing Sciences in Colleges* 30.3 (1st January 2015), pp. 117–125. URL: https://dl.acm.org/doi/10.5555/2675327.2675346.

[28] Wes McKinney. 'Data Structures for Statistical Computing in Python'. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[29] Michèle B. Nuijten et al. 'Verify Original Results Through Reanalysis Before Replicating'. In: *Behavioral and Brain Sciences* 41 (27th July 2018), e143. DOI: 10.1017/S0140525X18000791.

[30] Vincent Peters, Geert Vissers and Gerton Heijne. 'The Validity of Games'. In: *Simulation & Gaming* 29.1 (March 1998), pp. 20–30. DOI: 10.1177/1046878198291003.

[31] Hans E. Plesser. 'Reproducibility vs. Replicability: A Brief History of a Confused Terminology'. In: *Frontiers in Neuroinformatics* 11 (18th January 2018). DOI: 10.3389/fninf.2017.00076.

[32] Tom Preston-Werner. *Readme Driven Development*. URL: https://tom.preston-werner.com/2010/08/23/readme-driven-development (visited on 24th May 2024).

[33] Jayanth Raghothama and Sebastiaan A. Meijer. 'A Review of Gaming Simulation in Transportation'. In: *International Simulation and Gaming Association Conference*. Springer. 2013, pp. 237–244. DOI: 10.1007/978-3-319-04954-0_28.

[34] Luis Henrique Oliveira Rios and Luiz Chaimowicz. 'trAIns: An Artificial Inteligence for OpenTTD'. In: *2009 VIII Brazilian Symposium on Games and Digital Entertainment*. IEEE. October 2009, pp. 52–63. DOI: 10.1109/SBGAMES.2009.15.

[35] Government Digital Service. *GOV.UK Service Manual: Agile Delivery*. URL: https://www.gov.uk/service-manual/agile-delivery (visited on 24th May 2024).

[36] OpenTTD Team. *BaNaNaS: Base graphics/sound And Newgrfs And Noais And Scenarios*. 2024. URL: https://bananas.openttd.org/ (visited on 15th August 2024).

[37] OpenTTD Team. *Flow Chart for Temperate Cargo*. 2016. URL: https://wiki.openttd.org/File/en/Manual/Temperate%20-%20Flow.png (visited on 26th June 2024).

[38] OpenTTD Team. *OpenGFX Source Code*. 2024. URL: https://github.com/OpenTTD/OpenGFX (visited on 27th May 2024).

[39] OpenTTD Team. *OpenTTD AI API Documentation*. 2024. URL: https://docs.openttd.org/ai-api/ (visited on 27th May 2024).

[40] OpenTTD Team. *OpenTTD Difficulty Levels*. 2024. URL: https://wiki.openttd.org/en/Archive/Manual/Settings/Difficulty (visited on 4th August 2024).

[41] OpenTTD Team. *OpenTTD GameScript API Documentation*. 2024. URL: https://docs.openttd.org/gs-api/ (visited on 27th May 2024).

[42] OpenTTD Team. *OpenTTD Logo*. 2nd February 2008. URL: https://en.m.wikipedia.org/wiki/File:Openttdlogo.svg (visited on 11th August 2024).

[43] OpenTTD Team. *OpenTTD Source Code*. 2024. URL: https://github.com/OpenTTD/OpenTTD (visited on 27th May 2024).

[44] OpenTTD Team. *OpenTTD: An Open Source Simulator Based on the Classic Game Transport Tycoon Deluxe*. Version 13.4. 30th July 2023. URL: https://www.openttd.org/.

[45] OpenTTD Team. *OpenTTD's Wiki*. 2024. URL: https://wiki.openttd.org/en/ (visited on 27th May 2024).

[46] The Pandas Development Team. *pandas-dev/pandas: Pandas*. February 2020. DOI: 10.5281/zenodo.3509134.

[47] Patric Stout (A. K. A. TrueBrain). *OpenTTD Savegame Reader*. Version b1a02ca0. 2024. URL: https://github.com/TrueBrain/OpenTTD-savegame-reader.

[48] The Turing Way Community. *The Turing Way: A handbook for reproducible, ethical and collaborative data science (1.0.2)*. Zenodo, 22nd July 2022. DOI: 10.5281/zenodo.3233853.

[49] Eva Volna. 'Fuzzy-Based Decision Strategy in Real-Time Strategic Games'. In: *Proceedings of the International Conference of Computational Methods in Sciences and Engineering 2017 (ICCMSE-2017)*. AIP Publishing. 28th November 2017. DOI: 10.1063/1.5012346.

[50] Maciej Wisniewski. 'Artificial Intelligence for the OpenTTD Game'. Master's thesis. Denmark: Technical University of Denmark, 2011. URL: https://www2.imm.dtu.dk/pubdb/pubs/6091-full.html.

Appendix A

OpenTTDLab: Releases

Following is a summary of the 72 releases of OpenTTDLab [44] I made during the course of this project. The release notes are as written at the time, and not corrected for clarity or typographical errors.

All code and documentation changes were made by myself, with the exceptions of the first release, v0.0.1, which was almost an exact copy of OpenTTD-savegame-reader by Patric Stout (A. K. A. TrueBrain), and v0.0.43, which contained a single-line documentation change by Ian Earle (A. K. A. BasicBeluga). Patric Stout also advised on the protocol used by BaNaNaS, the OpenTTD content service, and how OpenTTDLab should behave in regards to caching content retried from it. The ability to download AIs from BaNaNaS I added to OpenTTDLab in v0.0.30, and I made improvements in later releases.

Release	Commit	Release notes
v0.0.1	f145fd82	Initial release after the fork from `https://github.com/TrueBrain/OpenTTD-savegame-reader` and the rename to OpenTTDLab

Release	Commit	Release notes
v0.0.2	93dcb2ec	**What's Changed** • docs: cite upstream fork by @michalc in #1 • docs: mention that this is a WIP and the README is a design spec by @michalc in #2 • ci: remove unused deploy to GH pages workflow by @michalc in #3 • docs: basic install instructions and remove upstream info that isn't needed by @michalc in #4 • feat: remove webapp by @michalc in #5 • docs: use based on rather than forked by @michalc in #6 • docs: remove cli image that is only applicable for upstream by @michalc in #7 • docs: very initial API design by @michalc in #8 • docs: more details on usage/API and what an experiment is by @michalc in #9 • docs: how to reproduce an experiment by @michalc in #10 • docs: info on API design by @michalc in #11 • docs: allow platform differences by @michalc in #12 • docs: much more concise and accurate definition of experiment by @michalc in #13 • docs: put concept definition right up top by @michalc in #14 • feat: initial empty openttdlab module by @michalc in #15
v0.0.3	3260e19d	**What's Changed** • docs: use Note alert to highlight WIP by @michalc in #16 • docs: include suggestion on how to save and load the file by @michalc in #17 • docs: rename variables so they're shorter by @michalc in #18 • docs: mention that this project is not affiliated with OpenTTD by @michalc in #19 • docs: mention that we're OOP-free mostly for now by @michalc in #20 • feat: support Python from 3.6.7 by @michalc in #22 • feat: initial empty functions by @michalc in #21 • chore: remove requirements files by @michalc in #23 • ci: test with code coverage by @michalc in #24 • docs: add badges for PyPI, GitHub Actions and CodeCov by @michalc in #25
v0.0.4	914378b2	**What's Changed** • feat!: download OpenTTD to a cache directory by @michalc in #26

Release	Commit	Release notes
v0.0.5	b3c74d11	**What's Changed** • docs: use python rather than pip by @michalc in #27 • feat: extract the downloaded file by @michalc in #28 • feat: initial run_experiment and get_config functions by @michalc in #29
v0.0.6	952fd6e3	**What's Changed** • docs: use get_config consistently by @michalc in #30
v0.0.7	d4bf4505	**What's Changed** • feat: run OpenTTD by @michalc in #31
v0.0.8	52b3a471	**What's Changed** • docs: first attempt at logo by @michalc in #32 • docs: remove accidentally duplicated heading by @michalc in #33 • docs: next version of logo by @michalc in #34 • docs: make the height and width of the logo exactly equal to the viewBox by @michalc in #35 • docs: tweak height of logo by @michalc in #36 • docs: tweak accessibility attributes by @michalc in #37 • docs: add image alt by @michalc in #38
v0.0.9	686a6109	**What's Changed** • docs: make it clear we want to extract results from the experiments by @michalc in #39 • docs: use raw url for logo rather than relative link by @michalc in #40
v0.0.10	38042533	**What's Changed** • docs: remove API design considerations by @michalc in #41 • docs: set height of badges to try to avoid jumping as they load by @michalc in #42 • docs: remove granularity of output from experiment description by @michalc in #43 • docs: clearer installation instructions by @michalc in #44 • docs: mention OpenTTD does not need to be installed separately by @michalc in #45 • fix: location of baseset by @michalc in #46
v0.0.11	8a87b4b2	**What's Changed** • docs: add comments as to what the command line arguments do by @michalc in #47 • feat: extract (very unusable) results by @michalc in #48

Release	Commit	Release notes
v0.0.12	82649631	**What's Changed** • feat: include version in source file by @michalc in #49 • fix: deploying to PyPI by @michalc in #50
v0.0.13	a5df2116	**What's Changed** • docs: simpler 1-stage API by @michalc in #51 • feat: implement some of the simpler one stage API by @michalc in #52
v0.0.14	a6a0fc0f	**What's Changed** • feat: use Python libraries for tar and zips to avoid the binary dependencies by @michalc in #53
v0.0.15	a79d44b9	**What's Changed** • docs: fix a few typos/improve clarity by @michalc in #54
v0.0.16	2dcb2c5d	**What's Changed** • refactor: remove unused file by @michalc in #55 • devex: more recent Python .gitignore by @michalc in #56 • docs: add licenses and attributions section by @michalc in #57 • refactor: remove unused files by @michalc in #58 • refactor: combine savegame reader and linkgraph by @michalc in #59 • docs: fix typo/copy-paste error in notice by @michalc in #60
v0.0.17	b7cd9e6f	**What's Changed** • style: comments on same line as code by @michalc in #61 • refactor: tidy imports by @michalc in #62 • docs: put copyright/license notice in SVG logo by @michalc in #63 • docs: use the real name of savegame-reader's author by @michalc in #64 • docs: much more concise introduction by @michalc in #65 • docs: remove accidentally committed double full stop by @michalc in #66 • docs: remove non-affiliation-statement by @michalc in #67 • docs: make the original author of the logo clear by @michalc in #68 • feat: run with AI players by @michalc in #69 • feat: parse just a single save game, asserting can extract the AI name by @michalc in #70 • docs: hint that some things work by @michalc in #71 • feat: allow days to run to be configurable and return results for each save file by @michalc in #72 • fix: more deterministic saving by @michalc in #73

Release	Commit	Release notes
v0.0.18	4a7c3952	**What's Changed** • ci: test on macOS (and fewer Python versions) by @michalc in #74
v0.0.19	be2402f2	**What's Changed** • feat: run experiment into a temporary directory by @michalc in #75 • feat: run for a range of seeds by @michalc in #76 • fix: actually pass the random seed to OpenTTD by @michalc in #77 • feat: extract a much smaller subset of data by @michalc in #78 • feat: use a function to get AI files by @michalc in #79 • docs: example for plotting by @michalc in #80 • docs: a few things work now - say it by @michalc in #81
v0.0.20	1abe35a6	**What's Changed** • refactor: remove double import by @michalc in #83 • refactor: delegate copying of ai file to function by @michalc in #84 • feat: basic AI downloading by @michalc in #85 • docs: make it clear avoiding manual steps by @michalc in #86
v0.0.21	22d02daf	**What's Changed** • docs: make the licensing section clearer - it's all GPL v2 by @michalc in #87 • docs: remove detail about every file being GPL v2 by @michalc in #88 • docs: use the absolute URL for the chart in README by @michalc in #89
v0.0.22	0f0594bc	**What's Changed** • feat: support Windows by @michalc in #90
v0.0.23	b1d2a443	**What's Changed** • perf: less downloading of AIs by @michalc in #91 • docs: supported Python and OS versions in compatibility section by @michalc in #92 • refactor: fewer layers in reader by @michalc in #93 • refactor: towards using iterables and less mutation by @michalc in #94 • refactor: less mutation by @michalc in #95
v0.0.24	290d356e	**What's Changed** • refactor: make binary reader return length read by @michalc in #96 • refactor: reduce binary reading duplication by @michalc in #97 • refactor: remove unused function by @michalc in #98

Release	Commit	Release notes
v0.0.25	d67e005e	**What's Changed** • refactor: less mutation when reading items by @michalc in #99
v0.0.26	2a1f64b8	**What's Changed** • refactor: clearer item parsing by @michalc in #100 • refactor: remove unreachable code by @michalc in #101 • refactor: simpler gamma function by @michalc in #102 • fix: parsing when structs have the same key by @michalc in #103
v0.0.27	1e6d7c47	**What's Changed** • refactor: reduce mutation by @michalc in #104 • feat: JSON-encodable parsed save games by @michalc in #105
v0.0.28	60f9fc0b	**What's Changed** • feat: return a (so far null) metadata for cpu and memory info by @michalc in #106
v0.0.29	612e0463	**What's Changed** • docs: add table of contents by @michalc in #107 • docs: shorter WIP warning by @michalc in #108 • feat: a rough way of settings any config value by @michalc in #109 • refactor: prefer ternary operator to if by @michalc in #110 • refactor: reduce a small amount of nesting by @michalc in #111 • docs: more detail on why no supporting OTTD by @michalc in #112 • refactor: fewer cases of if for flow control by @michalc in #113 • refactor: neater mutable state in non-sparse tables by @michalc in #114 • refactor: use variable passed to function not from scope by @michalc in #115
v0.0.30	60f848a3	**What's Changed** • feat: ability to fetch AIs published on content server by @michalc in #116 • docs: remove incorrect argument to bananas_file by @michalc in #117 • feat: cache bananas AI files by @michalc in #118
v0.0.31	3ddcaa57	**What's Changed** • feat: a more straightforward API by @michalc in #119
v0.0.32	db02801d	**What's Changed** • feat: run experiments in parallel by @michalc in #120

Release	Commit	Release notes
v0.0.33	8760575a	**What's Changed** • docs: add list of features by @michalc in #121 • docs: fix grammar by @michalc in #122
v0.0.34	f6d27c13	**What's Changed** • feat: expose much more data in results object by @michalc in #123
v0.0.35	ad22fa24	**What's Changed** The main change is the addition of two optional parameters to the run_experiment function: openttd_version and opengfx_version. They take a string of the version of OpenTTD and OpenGFX to download and run the experiments on. Bother can be left as their default None to use the latest version available. • feat: allow the specification of OpenTTD and OpenGFX version by @michalc in #124
v0.0.36	87214860	**What's Changed** This release adds two keys in each results object: openttd_version and opengfx_version containing the version of OpenTTD and OpenGFX respectively. • feat: include OpenTTD and OpenGFX version in results by @michalc in #125
v0.0.37	0ddc5f6e	**What's Changed** The main non-documentation change of this release is to retry HTTP requests, and allow the http client to be configurable through the get_http_client client parameter. • docs: improve clarity of main example by @michalc in #126 • docs: slightly more detail on each AI definition function in its own section by @michalc in #127 • docs: change reference from OpenTTD to OpenTTDLab by @michalc in #128 • feat: retry HTTP requests, and make the client configurable by @michalc in #129

Release	Commit	Release notes
v0.0.38	c144428a	**What's Changed** This is a doc-only release • docs: add an example notebook by @michalc in #130 • docs: remove printing of results from experiment by @michalc in #131 • doc: link to examples folder so it's a touch more discoverable by @michalc in #132 • docs: add scaling example by @michalc in #133 • docs: be up front about poor scaling by @michalc in #134 • docs: add more detail to API docs by @michalc in #135 • docs: don't make it sound like there are loads of examples by @michalc in #136
v0.0.39	f9e46c65	**What's Changed** • feat: support an AI in a local folder by @michalc in #137
v0.0.40	28a27ca5	**What's Changed** The main change is how AIs are started - they now use the start_ai command in a startup script under the hood. This does unfortunately slightly change output because it changes exactly when AIs are started. But this pattern is hopefully more reproducible, because AIs are started consistently right at the beginning, and it allows passing parameters into AIs (although this will be a later change). • test: remove unnecessary call to os.path.join by @michalc in #139 • feat: start AIs immediately using a startup script by @michalc in #138
v0.0.41	e7cfbe9c	**What's Changed** The main chain in this release is the breaking change of each member of the ais iterable passed to run_experiment. It was a pair, but now it's a triple. The extra parameter (which is the second parameter) must be an iterable of (key,value) parameters passed to the AI when it starts. • feat: allow passing of parameters to AIs by @michalc in #140 • docs: fix the OpenTTD version in the example notebooks by @michalc in #141
v0.0.42	9759a60c	**What's Changed** This release contains a breaking change - how AIs are passed to the run_experiment function has changed (hopefully for the better). • feat: slightly less clunky API for AIs by @michalc in #142
v0.0.43	0cc75dab	**What's Changed** • docs: Updating example code in README by @BasicBeluga in #143

Release	Commit	Release notes
v0.0.44	c28d5469	**What's Changed** • feat: rename bananas_file to bananas_ai by @michalc in #144
v0.0.45	6f90a53c	**What's Changed** • feat: support download of AI libraries from bananas by @michalc in #145
v0.0.46	07a0a364	**What's Changed** • feat: option to take final screenshot by @michalc in #146
v0.0.47	8fb73c89	**What's Changed** • tests: assert that screenshots do get made by @michalc in #147 • feat: avoid windows popping up during screenshots if xvfb-run is available by @michalc in #148
v0.0.48	62b051e4	**What's Changed** • tests: disable test that uses the latest version of OpenTTD by @michalc in #150 • feat: show basic progress bar when running by @michalc in #149
v0.0.49	a4c19fca	**What's Changed** • feat!: rename run_experiment to run_experiments by @michalc in #151
v0.0.50	6b005b22	**What's Changed** • refactor: swap 'experiment' and 'run' in variable names by @michalc in #152 • feat!: more flexible batching of experiments by @michalc in #153
v0.0.51	a54c78db	**What's Changed** • feat: expose experiment dictionary in results by @michalc in #154
v0.0.52	f29985a5	**What's Changed** • docs: document experiments parameter by @michalc in #155
v0.0.53	5a4191fc	**What's Changed** • docs: update examples to work with latest OpenTTDLabAPI by @michalc in #156 • feat: allow OpenTTD config to be changed per experiment by @michalc in #157
v0.0.54	16f49a08	**What's Changed** • feat: add result processor by @michalc in #158
v0.0.55	7858609a	**What's Changed** • feat: make the result processor return iterable by @michalc in #159

Release	Commit	Release notes
v0.0.56	ede4a1e2	**What's Changed** The main change in this release is the ability to (automatically) download dependencies, and transitive dependencies, for AIs (and AI libraries) fetched from bananas • refactor: read all bytes from socket when querying BaNaNaS by @michalc in #161 • refactor: step towards automatically downloading dependencies by @michalc in #162 • refactor: store files using info from downloading by @michalc in #163 • refactor: clearer variable names by @michalc in #164 • refactor: more code that support multiple files per AI by @michalc in #165 • refactor: towards downloading AI libraries from AI download by @michalc in #166 • refactor: move fetching content ID to function by @michalc in #167 • feat: automatically download dependencies from BaNaNaS by @michalc in #168
v0.0.57	6f6664d4	**What's Changed** • feat: output combined stdout and stderr, and indicate if we think it errored by @michalc in #160 • docs: document that AI libraries are automatically downloaded by @michalc in #169
v0.0.58	b34e941e	**What's Changed** • docs: improve clarity on ai_libraries by @michalc in #170 • docs: fix style around version warning by @michalc in #171 • docs: put downloading and caching of AI libraries up front by @michalc in #172
v0.0.59	8324e4b4	**What's Changed** • refactor: remove unused linkgraph code by @michalc in #173 • feat: uncompress binaries into the run directory rather than the cache directory by @michalc in #175 • docs: explain what versions of OpenTTD are supported by @michalc in #176
v0.0.60	39af96ba	**What's Changed** • feat: move to multiprocess from threads by @michalc in #177 • ci: use token for CodeCov upload to avoid rate limiting errors by @michalc in #178

Release	Commit	Release notes
v0.0.61	0d29074b	**What's Changed** • feat: use built-in Python multiprocessing by @michalc in #180
v0.0.62	79368255	**What's Changed** • docs: update scaling example by @michalc in #181 • fix: don't copy the OpenGFX binary to the current directory by @michalc in #182
v0.0.63	1cfae3e8	**What's Changed** • fix: caching behaviour of libraries when there are multiple by @michalc in #183
v0.0.64	02d72ab0	**What's Changed** • fix: local folder behaviour by @michalc in #184
v0.0.65	2406be3a	**What's Changed** • docs: documentation on parse_savegame by @michalc in #185 • docs: mention that OpenTTDLab can be used as a test harness by @michalc in #186 • docs: links to API sections by @michalc in #187 • docs: rename fetching to configuring by @michalc in #188 • docs: fix link to running experiments by @michalc in #189 • docs: rename second 'Running experiments' section to 'Core function' by @michalc in #190 • refactor: use partial to reduce indentation and step towards public BaNaNaS client by @michalc in #191 • refactor: remove almost unused argument in (private) _bananas_download by @michalc in #192 • refactor: fewer arguments in private bananas function by @michalc in #193 • refactor: fewer arguments private bananas download function by @michalc in #194 • refactor: split querying TCP server and download by @michalc in #195 • perf: make only one connection per content to tcp server by @michalc in #196

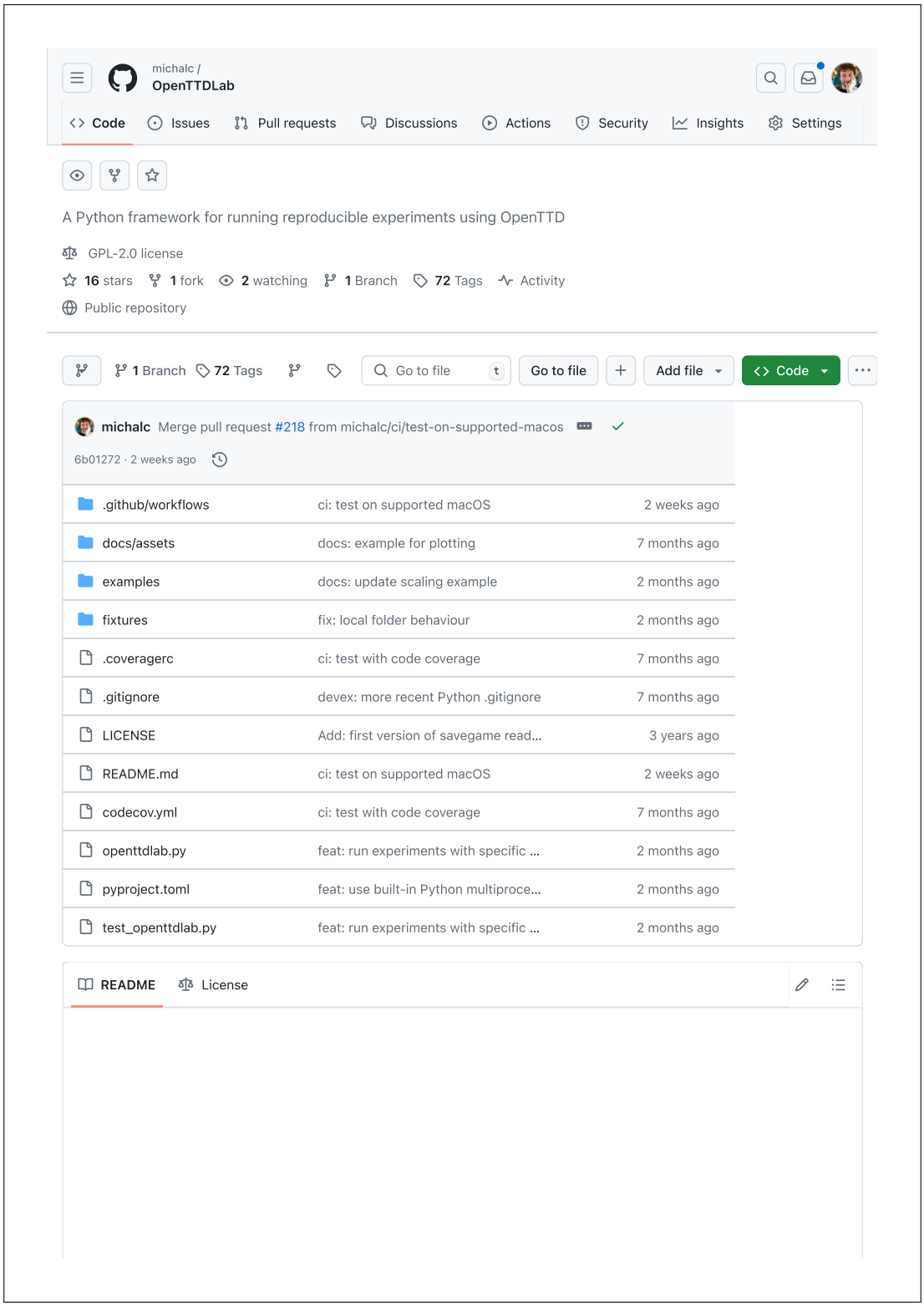
Release	Commit	Release notes
v0.0.66	bf7ec9b5	**What's Changed** • docs: fix titles to match TOC by @michalc in #198 • refactor: AI and AI library funcs return iterables of data by @michalc in #197 • tests: restore overwritten test by @michalc in #199 • feat: use context in bananas download function for http client by @michalc in #200 • feat: bust cache between versions of OpenTTDLab by @michalc in #201
v0.0.67	61b89dc7	**What's Changed** • feat: bananas download has customisable cache dir by @michalc in #202 • feat: bananas http client has default by @michalc in #203 • refactor: more code supports more types from bananas by @michalc in #204 • feat: tweak return types of bananas function to return user-facing content-id by @michalc in #205 • fix: scenario and basemap locations by @michalc in #206
v0.0.68	6aeb9d74	**What's Changed** • feat: return and cache md5sum by @michalc in #207 • feat: public BaNaNaS client by @michalc in #208
v0.0.69	194ac21e	**What's Changed** • docs: tweak description of bananas client by @michalc in #209 • docs: mention to not download everything from BaNaNaS by @michalc in #210 • docs: slightly more practical BaNaNaS download example by @michalc in #211 • docs: put more features up front by @michalc in #212 • fix: multiple AIs from local folders by @michalc in #213
v0.0.70	19561a4d	**What's Changed** • feat: expose the license when downloading from BaNaNaS by @michalc in #214
v0.0.71	32c27928	**What's Changed** • feat: allow download of exact version from BaNaNaS by @michalc in #215

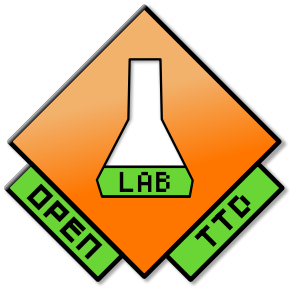
Release	Commit	Release notes
v0.0.72	50c807c3	**What's Changed** • feat: run experiments with specific version of BaNaNaS content by @michalc in #216

Appendix B

OpenTTDLab: Documentation

Following is a copy of the documentation for OpenTTDLab created as part of the current work, published in its GitHub repository [11]. The copy is taken from commit 6b012727.


michalc /
OpenTTDLab
Code
Issues
Pull requests
Discussions
Actions
Security
Insights
Settings
A Python framework for running reproducible experiments using OpenTTD
GPL-2.0 license
16 stars
1 fork
2 watching
1 Branch
72 Tags
Activity
Public repository
1 Branch
72 Tags
Go to file
t
Go to file
Add file
Code
michalc Merge pull request #218 from michalc/ci/test-on-supported-macos
6b01272 · 2 weeks ago
.github/workflows ci: test on supported macOS 2 weeks ago
docs/assets docs: example for plotting 7 months ago
examples docs: update scaling example 2 months ago
fixtures fix: local folder behaviour 2 months ago
.coveragerc ci: test with code coverage 7 months ago
.gitignore devex: more recent Python .gitignore 7 months ago
LICENSE Add: first version of savegame read... 3 years ago
README.md ci: test on supported macOS 2 weeks ago
codecov.yml ci: test with code coverage 7 months ago
openttdlab.py feat: run experiments with specific ... 2 months ago
pyproject.toml feat: use built-in Python multiproce... 2 months ago
test_openttdlab.py feat: run experiments with specific ... 2 months ago
README
License



OpenTTDLab - *Run reproducible experiments using OpenTTD*

PyPI package v0.0.72 | Test suite passing | Code coverage 92%

OpenTTDLab is a Python framework for using OpenTTD to run reproducible experiments and extracting results from them, with as few manual steps as possible. OpenTTDLab can also be used to help run regression tests of OpenTTD AIs, parse OpenTTD savegame files, and download content from BaNaNaS.

OpenTTDLab is based on Patric Stout's OpenTTD Savegame Reader.

> Caution
>
> OpenTTDLab currently does not work with OpenTTD 14.0 or later. The latest version of OpenTTD known to work is 13.4.

Contents

Features

OpenTTDLab essentially turns OpenTTD into a simulator - and through AIs and AI libraries it allows you to experiment with different techniques of building supply chains and study their effects. In more detail OpenTTDLab:

- Allows you to easily run OpenTTD in a headless mode (i.e. without a graphical interface) over a variety of configurations.
- And allows you to do this from Python code - for example from a Jupyter Notebook.
- As is typical from Python code, it is cross platform - allowing to share code snippets between macOS, Windows, and Linux, even though details like how to install and start OpenTTD are different on each platform.
- Downloads (and caches) OpenTTD, OpenGFX, AIs, and AI libraries - no need to download these separately or through OpenTTD's built-in content browser.
- Transparently parallelises runs of OpenTTD, by default up to the number of CPUs. (Although with fairly poor scaling properties.)
- Results are extracted from OpenTTD savegames as plain Python dictionaries and lists - reasonably convenient for importing into tools such as pandas for analysis or visualisation.

Installation

OpenTTDLab is distributed via PyPI, and so can usually be installed using pip.

```
python -m pip install OpenTTDLab
```

When run on macOS, OpenTTDLab has a dependency that pip does not install: 7-zip. To install 7-zip, first install Homebrew, and then use Homebrew to install the p7zip package that contains 7-zip.

```
brew install p7zip
```

You do not need to separately download or install OpenTTD (or OpenGFX) in order to use OpenTTDLab. OpenTTDLab itself handles downloading them.

Running experiments

The core function of OpenTTD is the `run_experiments` function.

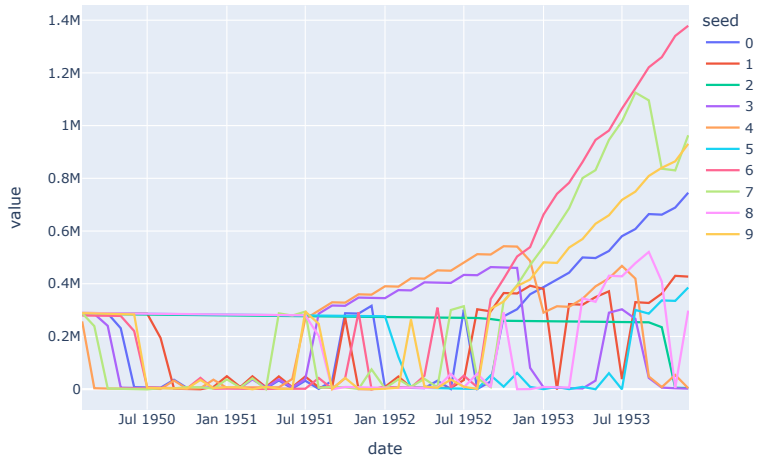
```
from openttdlab import run_experiments, bananas_ai

# Run experiments...
results = run_experiments(
    openttd_version='13.4',  # ... for a specific versions of OpenTTD
    opengfx_version='7.1',   # ... and a specific versions of OpenGFX
    experiments=(
       {
         # ... for random seeds
         'seed': seed,
         # ... running specific AIs. In this case a single AI, with no
         # parameters, fetching it from https://bananas.openttd.org/package/ai
         'ais': (
             bananas_ai('54524149', 'trAIns', ai_params=()),
         ),
         # ... each for a number of (in game) days
         'days': 365 * 4 + 1,
       }
       for seed in range(0, 10)
    ),
)
```

Plotting results

OpenTTD does not require any particular library for plotting results. However, pandas and Plotly Express are common options for plotting from Python. For example if you have a `results` object from `run_experiments` as in the above example, the following code

```
import pandas as pd
import plotly.express as px

df = pd.DataFrame(
    {
        'seed': row['experiment']['seed'],
        'date': row['date'],
        'money': row['chunks']['PLYR']['0']['money'],
    }
    for row in results
)
df = df.pivot(index='date', columns='seed', values='money')
fig = px.line(df)
fig.show()
```

should output a plot much like this one.



Examples

A few examples are available:

- A Jupyter notebook of the above example, briefly exploring the performance of trAIns
- A Jupyter notebook showing how the performance of OpenTTDLab scales with the number of workers
- A pytest .py file using OpenTTDLab as a test harness for an OpenTTD AI, run automatically using a GitHub action workflow

API

Core function

`run_experiments(...)`

The core function of OpenTTDLab is the `run_experiments` function, used to run an experiment and return results extracted from the savegame files that OpenTTD produces. It has the following parameters and defaults.

- `experiments=()`

  An iterable of the experiments to run. Each experiment should be a dictionary with the (string) keys:

  - `'ais'`

    The list of AIs to run in this experiment. See the Fetching AIs section for details on this parameter.

  - `'seed'`

    The integer seed of the random number generator for this experiment.

  - `'days'`

    The integer number of in-game days that this experiment will run for.

  - `openttd_config=''`

    OpenTTD config to run each experiment under. This must be in the openttd.cfg format. This is added to by OpenTTDLab before being passed to OpenTTD.

- `ais_libraries=()`

  The list of AI libraries to have available to AI code. See the Fetching AI libraries section for details on this parameter.

- `result_processor=lambda r: (r,)`

  A function that takes a single result row, which is a parsed save game file from an experiment, alongside other metadata describing the experiment, and returns it processed in some way. The function should return an iterable of zero or more rows that will appear in the the return value of `run_experiments`.

  This is typically used to reduce memory usage with high numbers of experiments where only a small amount of data is needed for analysis.

- `final_screenshot_directory=None`

  The directory to save a PNG screenshot of the entire map at the end of each run. Each is named in the format `<seed>.png`, where `<seed>` is the experiment's seed of the random number generator. If `None`, then no screenshots are saved.

  For technical reasons, a window will briefly appear while each screenshot is being saved. This can be avoided when running on Linux if `xvfb-run` is installed and available in the path.

- `max_workers=None`

  The maximum number of workers to use to run OpenTTD in parallel. If `None`, then `os.cpu_count()` defined how many workers run.

- `openttd_version=None`

The version of OpenTTD to use. If `None`, the latest version available at `openttd_base_url` is used.

> **Caution** OpenTTDLab currently does not work with OpenTTD 14.0 or later. The latest version of OpenTTD known to work is 13.4.

- `opengfx_version=None`

  The version of OpenGFX to use. If `None`, the latest version available at `opengfx_base_url` is used.

- `openttd_base_url='https://cdn.openttd.org/openttd-releases/`

  The base URL used to fetch the list of OpenTTD versions, and OpenTTD binaries.

- `opengfx_base_url='https://cdn.openttd.org/opengfx-releases/`

  The URL used to fetch the list of OpenGFX versions, and OpenGFX binaries.

- `get_http_client=lambda: httpx.Client(transport=httpx.HTTPTransport(retries=3)`

  The HTTP client used to make HTTP requests when fetching OpenTTD, OpenGFX, or AIs. Note that the `bananas_ai` function uses a raw TCP connection in addition to HTTP requests, and so not all outgoing connections use the client specified by this.

Configuring AIs

The value of the `ais` key of each dictionary in the `experiments` parameter configures which AIs will run, how their code will be located, their names, and what parameters will be passed to each of them when they start. In more detail, the `ais` parameter must be an iterable of the return value of any of the the following 4 functions.

> Important
>
> The `ai_name` argument passed to each of the following functions must exactly match the name of the corresponding AI as published. If it does not match, the AI will not be started.

> Important
>
> The return value of each of the following is opaque: it should not be used in client code, other than by passing into `run_experiments` as part of the `ais` parameter.

`bananas_ai(unique_id, ai_name, ai_params=(), md5=None)`

Defines an AI by the `unique_id` and `ai_name` of an AI published through OpenTTD's content service at https://bananas.openttd.org/package/ai. This allows you to quickly run OpenTTDLab with a published AI. The `ai_params` parameter is an optional parameter of an iterable of `(key, value)` parameters passed to the AI on startup.

The `unique_id` is sometimes surfaced as the "Content Id", but it should not include its `ai/` prefix.

If you pass the full MD5 hex string of a specific version of AI as `md5`, for example previously returned from the `download_from_bananas` function, the corresponding version will be used. Otherwise, the latest version will be used.

`local_folder(folder_path, ai_name, ai_params=()))`

Defines an AI by the `folder_path` to a local folder that contains the AI code of an AI with name `ai_name`. The `ai_params` parameter is an optional parameter of an iterable of `(key, value)` parameters passed to the AI on startup.

`local_file(path, ai_name, ai_params=())`

Defines an AI by the local path to a .tar AI file that contains the AI code. The `ai_params` parameter is an optional parameter of an iterable of `(key, value)` parameters passed to the AI on startup.

`remote_file(url, ai_name, ai_params=())`

Fetches the AI by the URL of a tar.gz file that contains the AI code. For example, a specific GitHub tag of a repository that contains its code. The `ai_params` parameter is an optional parameter of an iterable of `(key, value)` parameters passed to the AI on startup.

Configuring AI libraries

The `ai_libraries` parameter of `run_experiments` ensures that AI libraries are available to the AIs running. In more detail, the `ais_libraries` parameter must be an iterable, where each item the the return value of the `bananas_ai_library` function described below.

Note that for AIs specified by `bananas_ai` OpenTTDLab automatically downloads all of their AI library dependencies without them having to be specified through the `ai_libraries` parameter. This includes all transitive AI library dependencies - AI libraries needed by AI libraries needed by AIs, and so on.

Similarly for AI libraries specified by `bananas_ai_library` - OpenTTDLab automatically downloads of their AI library dependencies.

`bananas_ai_library(unique_id, ai_library_name, md5=None)`

Fetches the AI library defined by `unique_id` and `ai_name` of a library published through OpenTTD's content service at https://bananas.openttd.org/package/ai-library.

The `unique_id` is sometimes surfaced as the "Content Id", but it should not include its `ai-library/` prefix.

If you pass the full MD5 string of a specific version of AI library as `md5`, for example previously returned from the `download_from_bananas` function, this will fetch this corresponding version from BaNaNaS. Otherwise, the latest version is fetched.

Parsing savegame files

`parse_savegame(chunks: Iterable[bytes])`

Under the hood the `run_experiments` handles the generation and parsing of savegame files, but if you have your own savegame files generated separately, the `parse_savegame` function is exposed that can extract data from them.

It takes an iterable of `bytes` instances of a savegame file, and returns a nested dictionary of parsed data.

```
from openttdlab import parse_savegame

with open('my.sav') as f:
   parsed_savegame = parse_savegame(iter(lambda: f.read(65536), b''))
```

Downloading from BaNaNaS

> Important
>
> Please do not use this to try to download all content from BaNaNaS. See this discussion about writing a client for BaNaNaS for more details.

> Important
>
> Please note the license of each piece of content you download, and adhere to its rules. As examples, licenses may require you to attribute the author, they can restrict you from using the content for commercial purposes, or they can require you to make the source available if you distribute a compiled version.

`download_from_bananas(content_id: str, md5: Optional[str]=None)`

This function is a Python BaNaNaS client for downloading the content from BaNaNaS. Given a content id, it returns an iterable of that content and all of its direct and transitive dependencies.

```
from openttdlab import download_from_bananas

with download_from_bananas('ai/41444d4c') as files:
    for content_id, filename, license, partial_or_full_md5, get_data in files:
        with get_data() as chunks:
            with open(filename, 'wb') as f:
                for chunk in chunks:
                    f.write(chunk)
```

Each `chunks` iterable are the binary chunks of the non-compressed `.tar` file of the content. Also, under the hood `download_from_bananas` transparently caches content where possible. This is the main reason for using context managers as in the above example - they allow for robust cleanup of resources and caching of data once the data has been iterated over.

If you don't pass `md5`, `download_from_bananas` will return details for the *latest* version of the content. And if the content has a known and acceptable license, `partial_or_full_md5` will contain the full MD5 for the content, which can then be subsequently passed back into `download_from_bananas` to download the same version later. If the file does not have a known license, `download_from_bananas` will contain the only the first 8 characters of the MD5. This means that `download_from_bananas` is deterministic only for content that has an acceptable license, and if you pass an MD5 previously retrieved from a call to `download_from_bananas`.

Note that the function `run_experiments` that uses `bananas_ai` or `bananas_ai_library` will handle automatically downloading from BaNaNaS, so this function is usually only useful if you would like to run experiments without using the `bananas_*` functions, or report on the filename (which includes the version of each piece of content) or the MD5 sum of the file.

Tips for repeatability, reproducibility, and replicability

OpenTTDLab should be able to help with all the 3Rs of repeatability, reproducibility and (although to a lesser degree) replicability of simulations. Definitions of the 3Rs are taken from https://www.acm.org/publications/policies/artifact-review-and-badging-current.

Repeatability

> "For computational experiments, this means that a researcher can reliably repeat her own computation."

The run_experiments function is the primary way OpenTTLab supports repeatability - with a single function call you can run a range of experiments. But to make sure you get the same results on each invocation:

- Pin to a specific OpenTTD and OpenGFX version.
- If fetching AIs and AI libraries from BaNaNaS, use the MD5 to ensure the same versions are used (at the moment the only way of discovering these is to initially separately call the download_from_bananas_function).
- If fetching AIs and AI libraries from another remote source, make sure that source is immutable.

- For all your own code store in version control, e.g. git, and make sure to note version/commit IDs used to generate results.
- Use a virtual environment to pin Python version and all dependencies (most importantly of OpenTTDLab).
- Use fixed random seeds.
- (Out of paranoia mostly, note the OS and its version, and CPU type.)

Reproducibility

> "For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts."

To help others reproduce your results:

- Take all the repeatability steps above, and make sure to communicate all the noted details, e.g. what versions where pinned, along with any results.
- Make any artifacts that you have created available alongide the results / linked from the results. This can include the source of AIs you have used or created, the Python code to run OpenTTDLab, and the Python code to analyse the results.

In general, bitwise reproducibility is not expected, but with fixed random seeds it "should" be the case with OpenTTD/OpenTTDLab (depending on the analysis performed).

Replicability

> "For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently."

This is arguably the most difficult of the 3Rs to plan for, and for other researchers to achieve. To help others replicate your results:

- Take all the repeatability and reproducibility steps above.
- Note and communicate the high level algorithms involved, so others can create their own implementations of them. For example, if you have written an AI, then precisely communicate the algorithm that AI takes so others can implement it, and then run it with OpenTTDLab.

  For *full* replicability, it could be argued that OpenTTD itself, or something equivalent, would also have to be implemented. While this sounds unfeasbile, depending on what the AIs involve and what you argue the results are, it might be feasible. However, for this level of replicability OpenTTDLab is unlikely to be helpful.

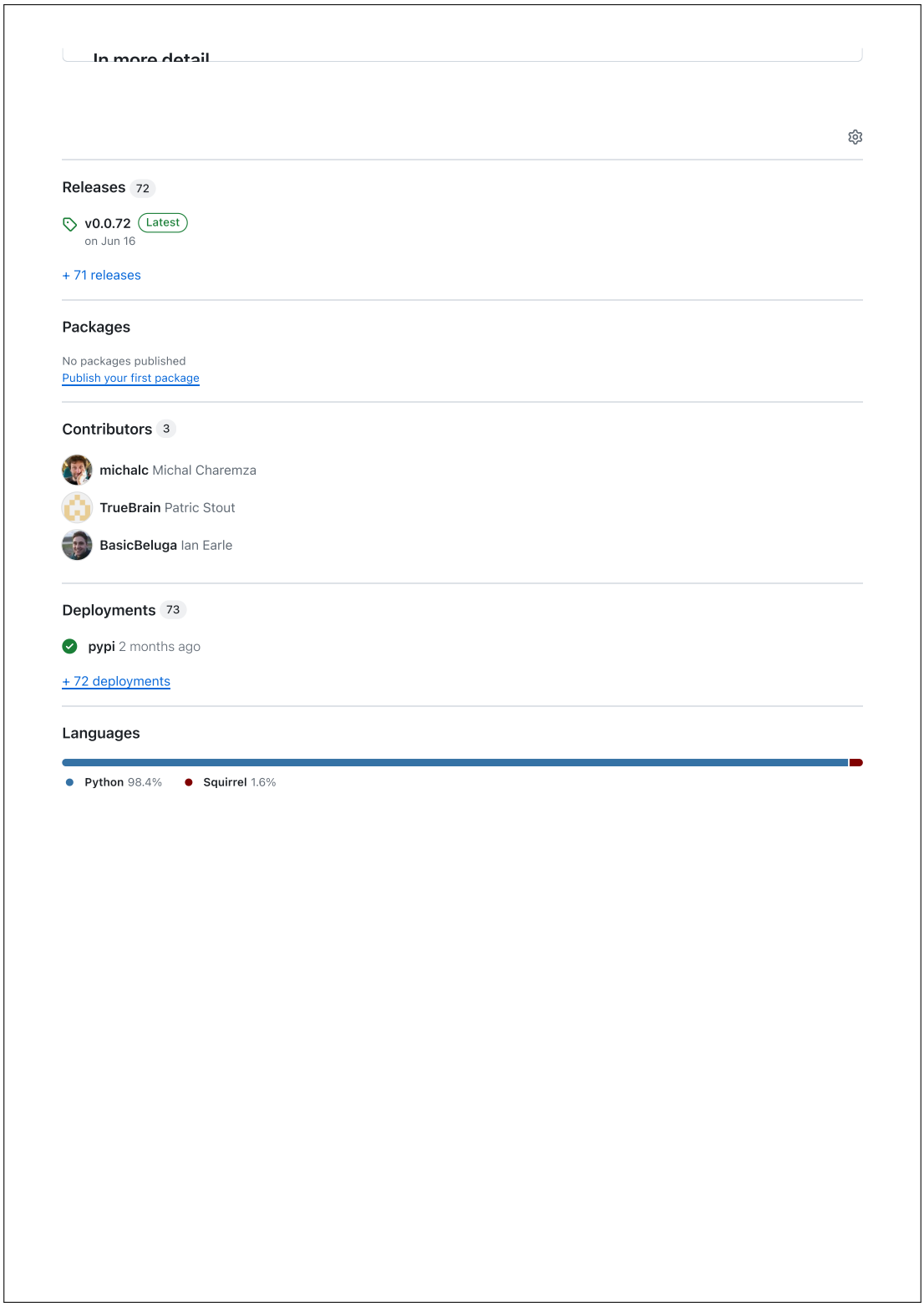
Compatibility

- OpenTTD versions between 12.0 and 13.4 (OpenTTD >= 14.0 is not currently supported. See this discussion on the changes in OpenTTD 14.0.)
- Linux (tested on Ubuntu 20.04), Windows (tested on Windows Server 2019), or macOS (tested on macOS 12)
- Python >= 3.8.0 (tested on 3.8.0 and 3.12.0)

Licenses and attributions

TL;DR

OpenTTDLab is licensed under the GNU General Public License version 2.0.


In more detail
Releases 72
v0.0.72 Latest
on Jun 16
+ 71 releases
Packages
No packages published
Publish your first package
Contributors 3
michalc Michal Charemza
TrueBrain Patric Stout
BasicBeluga Ian Earle
Deployments 73
pypi 2 months ago
+ 72 deployments
Languages
Python 98.4%
Squirrel 1.6%

Appendix C

Reproducing results: Python code to run experiments

Following is a copy of Python notebook used to run the reproducing experiments of Chapter 4. It is available in source code form at https://github.com/michalc/OpenTTDLab-MSc-Dissertation/blob/3f870c6d2f0ab77689572efdbfdf634b45edc4a7/01_trains_ai_vs_admiral_ai_01_run_experiments.ipynb.

01_trains_ai_vs_admiral_ai_01_run_experiments

August 21, 2024

1 trAIns AI vs Admiral AI - run experiments¶

Runs a series of experiments of OpenTTDLab, attempting to replicate the results from "trAIns: An Artificial Inteligence for OpenTTD" DOI 10.1109/SBGAMES.2009.15

Results are saved to 01_trains_ai_vs_admiral_ai_01_raw.csv.

[ ]:
```
!python -m pip install OpenTTDLab==0.0.72 pandas==2.2.0
```

[ ]:
```
from openttdlab import run_experiments, bananas_ai, bananas_ai_library

def process_result(result):
    def get_company_value(player):
        try:
            return player['old_economy'][0]['company_value']
        except (KeyError, IndexError):
            return 0

    return (
        {
            'date': result['date'],
            'seed': result['experiment']['seed'],
            'terrain_type': result['chunks']['PATS']['0']['difficulty.
 ↪terrain_type'],
            'name': \
                'Admiral AI' if player['name'].startswith('AdmiralAI') else \
                'trAIns AI' if player['name'].startswith('trAIns AI') else \
                'Unknown',
            'company_value': get_company_value(player),
            'money': player['money'],
        }
        for player in result['chunks']['PLYR'].values()
    )

results = run_experiments(
    openttd_version='13.4',
    opengfx_version='7.1',
    experiments=(
```

```
        {
            'seed': seed,
            'ais': (
                # To get a specific version of a library from BaNaNaS, we use␣
↪the full MD5 rather than
                # the actual version number.
                # trAIns 2.1
                bananas_ai('54524149', 'trAIns',␣
↪md5='c4c069dc797674e545411b59867ad0c2'),
                # AdmiralAI 25
                bananas_ai('41444d4c', 'AdmiralAI',␣
↪md5='4ccd92fb8f8f01045145be99a28e14a6', ai_params=(
                    ('use_trains', '1'),
                    ('use_busses', '0'),
                    ('use_trucks', '0'),
                    ('use_planes', '0'),
                )),
            ),
            'days': 366 * 15 + 1,
            'openttd_config': f'''
                [difficulty]
                terrain_type={terrain_type}
                number_towns=2
                industry_density=2
                max_loan=300000
                initial_interest=3
                vehicle_costs=1
                subsidy_multiplier=1
                construction_cost=1
                economy=true
                quantity_sea_lakes=0
                vehicle_breakdowns=0
                town_council_tolerance=1
                disasters=true
                line_reverse_mode=true
                [game_creation]
                starting_year=1960
                max_x=512
                max_y=512
            ''',
        }
        for seed in range(0, 64)
        for terrain_type in [1, 3]
    ),
    result_processor=process_result,
)
```

[ ]:
```
import pandas as pd

df = pd.DataFrame(results)
df.to_csv('01_trains_ai_vs_admiral_ai_results_01_raw.csv', index=False)
```

Appendix D

Reproducing results: Python code to analyse results

Following is a copy of the Python notebook used to analyse the results of the reproducing experiments of Chapter 4. It is available in source code form at https://github.com/michalc/OpenTTDLab-MSc-Dissertation/blob/8d432038278685edb0988457c9d2a971a668ac64/notebooks/01_trains_ai_vs_admiral_ai_01_analyse_results.ipynb.

02_trains_ai_vs_admiral_ai_02_analyse_results

June 30, 2024

1 trAIns AI vs Admiral AI - analyse results

Extracts and visualises results from 02_trains_ai_vs_admiral_ai_raw.csv.

[1]:
```
from datetime import datetime
import pandas as pd
import plotly.express as px
import numpy as np
```

[2]:
```
df_openttdlab = pd.read_csv('02_trains_ai_vs_admiral_ai_results_01_raw.csv',␣
↪parse_dates=[0])
df_openttdlab['terrain_type'] = df_openttdlab['terrain_type'].replace(1, 'Flat')
df_openttdlab['terrain_type'] = df_openttdlab['terrain_type'].replace(3,␣
↪'Mountainous')
df_openttdlab['source'] = 'OpenTTDLab'
```

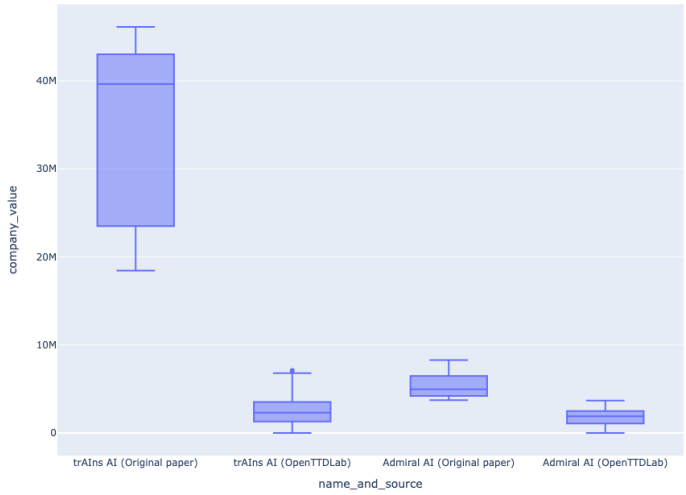
[3]:
```
# Hand copied from trAIns: An Artificial Inteligence for OpenTTD
# DOI 10.1109/SBGAMES.2009.15
# The seed here is _not_ the random seed, which is unknown. However, it allows␣
↪the same
# processing as for results from OpenTTDLab, in which the seeds are known
df_original = pd.DataFrame([
    {'date': datetime(1975, 6, 8), 'name': 'Admiral AI', 'company_value':␣
↪6090193, 'terrain_type': '1', 'seed': 1},
    {'date': datetime(1975, 6, 8), 'name': 'trAIns AI', 'company_value':␣
↪46124685, 'terrain_type': '1', 'seed': 1},
    {'date': datetime(1975, 2, 1), 'name': 'Admiral AI', 'company_value':␣
↪4634504, 'terrain_type': '1', 'seed': 2},
    {'date': datetime(1975, 2, 1), 'name': 'trAIns AI', 'company_value':␣
↪18437790, 'terrain_type': '1', 'seed': 2},
    {'date': datetime(1975, 1, 11), 'name': 'Admiral AI', 'company_value':␣
↪8272364, 'terrain_type': '1', 'seed': 3},
    {'date': datetime(1975, 1, 11), 'name': 'trAIns AI', 'company_value':␣
↪43477220, 'terrain_type': '1', 'seed': 3},
    {'date': datetime(1975, 1, 4), 'name': 'Admiral AI', 'company_value':␣
↪3729935, 'terrain_type': '1', 'seed': 4},
```

```
    {'date': datetime(1975, 1, 4), 'name': 'trAIns AI', 'company_value':␣
↪41622238, 'terrain_type': '1', 'seed': 4},
    {'date': datetime(1975, 1, 21), 'name': 'Admiral AI', 'company_value':␣
↪4073063, 'terrain_type': '1', 'seed': 5},
    {'date': datetime(1975, 1, 21), 'name': 'trAIns AI', 'company_value':␣
↪35198641, 'terrain_type': '1', 'seed': 5},
    {'date': datetime(1975, 11, 1), 'name': 'Admiral AI', 'company_value':␣
↪4955126, 'terrain_type': '1', 'seed': 6},
    {'date': datetime(1975, 11, 1), 'name': 'trAIns AI', 'company_value':␣
↪39619536, 'terrain_type': '1', 'seed': 6},
    {'date': datetime(1975, 1, 2), 'name': 'Admiral AI', 'company_value':␣
↪6591956, 'terrain_type': '1', 'seed': 7},
    {'date': datetime(1975, 1, 2), 'name': 'trAIns AI', 'company_value':␣
↪19586151, 'terrain_type': '1', 'seed': 7},
    {'date': datetime(1975, 1, 1), 'name': 'Admiral AI', 'company_value':␣
↪4581543, 'terrain_type': '3', 'seed': 1},
    {'date': datetime(1975, 1, 1), 'name': 'trAIns AI', 'company_value':␣
↪23004009, 'terrain_type': '3', 'seed': 1},
    {'date': datetime(1975, 9, 23), 'name': 'Admiral AI', 'company_value':␣
↪29700, 'terrain_type': '3', 'seed': 2},
    {'date': datetime(1975, 9, 23), 'name': 'trAIns AI', 'company_value':␣
↪22927440, 'terrain_type': '3', 'seed': 2},
    {'date': datetime(1975, 1, 26), 'name': 'Admiral AI', 'company_value':␣
↪5408988, 'terrain_type': '3', 'seed': 3},
    {'date': datetime(1975, 1, 26), 'name': 'trAIns AI', 'company_value':␣
↪21110332, 'terrain_type': '3', 'seed': 3},
    {'date': datetime(1975, 8, 5), 'name': 'Admiral AI', 'company_value':␣
↪2280652, 'terrain_type': '3', 'seed': 4},
    {'date': datetime(1975, 8, 5), 'name': 'trAIns AI', 'company_value':␣
↪40170005, 'terrain_type': '3', 'seed': 4},
    {'date': datetime(1975, 5, 7), 'name': 'Admiral AI', 'company_value':␣
↪3030691, 'terrain_type': '3', 'seed': 5},
    {'date': datetime(1975, 5, 7), 'name': 'trAIns AI', 'company_value':␣
↪34544942, 'terrain_type': '3', 'seed': 5},
    {'date': datetime(1975, 6, 26), 'name': 'Admiral AI', 'company_value':␣
↪1283846, 'terrain_type': '3', 'seed': 6},
    {'date': datetime(1975, 6, 26), 'name': 'trAIns AI', 'company_value':␣
↪33816215, 'terrain_type': '3', 'seed': 6},
    {'date': datetime(1975, 1, 5), 'name': 'Admiral AI', 'company_value':␣
↪1462148, 'terrain_type': '3', 'seed': 7},
    {'date': datetime(1975, 1, 5), 'name': 'trAIns AI', 'company_value':␣
↪17162685, 'terrain_type': '3', 'seed': 7},
])
df_original['terrain_type'] = df_original['terrain_type'].replace('1', 'Flat')
```

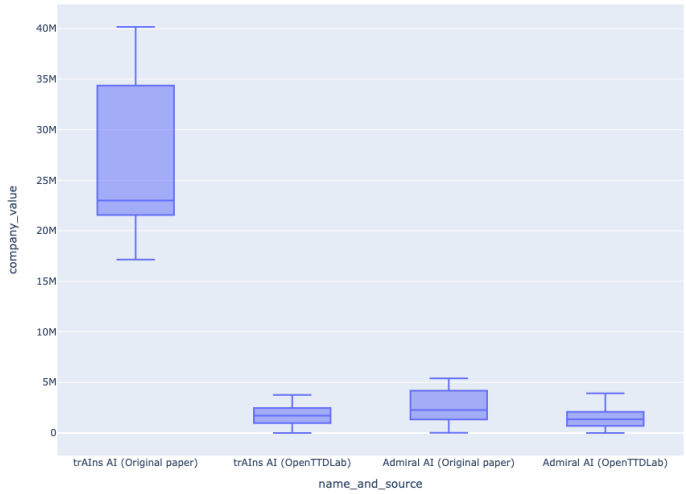
```
df_original['terrain_type'] = df_original['terrain_type'].replace('3',␣
↪'Mountainous')
df_original['source'] = 'Original paper'
```

[4]:
```
df_combined_final = pd.concat([df_original, df_openttdlab[df_openttdlab['date']␣
↪== '1975-01-01']])
df_combined_final['name_and_source'] = df_combined_final['name'] + ' (' +␣
↪df_combined_final['source'] + ')'
df_combined_final.to_csv('02_trains_ai_vs_admiral_ai_results_02_combined_final.
↪csv', index=False)
```

[5]:
```
fig = px.box(df_combined_final[df_combined_final["terrain_type"] == 'Flat'],␣
↪x='name_and_source', y="company_value", height=700)
fig.update_xaxes(categoryorder='category descending')
fig.show()
```

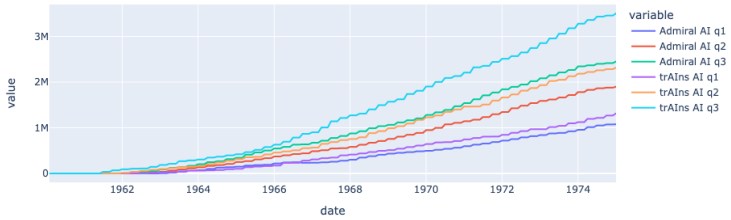


[6]:
```
fig = px.box(df_combined_final[df_combined_final["terrain_type"] ==␣
↪'Mountainous'], x="name_and_source", y="company_value", height=700)
fig.update_xaxes(categoryorder='category descending')
fig.show()
```

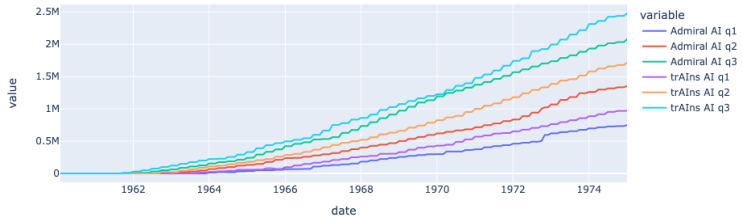


[7]:
```
df_to_group = df_openttdlab[['company_value','terrain_type','name','date']] \
    .groupby(['terrain_type', 'name', 'date'])
df_company_value_quartiles = pd.concat([
    df_to_group.agg('quantile', 0.25)['company_value'].rename('q1'),
    df_to_group.agg('quantile', 0.5)['company_value'].rename('q2'),
    df_to_group.agg('quantile', 0.75)['company_value'].rename('q3'),
], axis=1).stack().unstack(level=0).unstack(level=0).unstack(level=1) \
  .rename_axis(['terrain_type', 'name', 'quartile'], axis='columns')
df_company_value_quartiles.
 ↪to_csv('02_trains_ai_vs_admiral_ai_results_03_openttdlab_company_value_quartiles.
 ↪csv')
```

[8]:
```
df_company_value_quartiles_flat = df_company_value_quartiles.xs('Flat',␣
 ↪level='terrain_type', axis=1)
df_company_value_quartiles_flat.columns = [' '.join(col).strip() for col in␣
 ↪df_company_value_quartiles_flat.columns.values]
px.line(df_company_value_quartiles_flat)
```



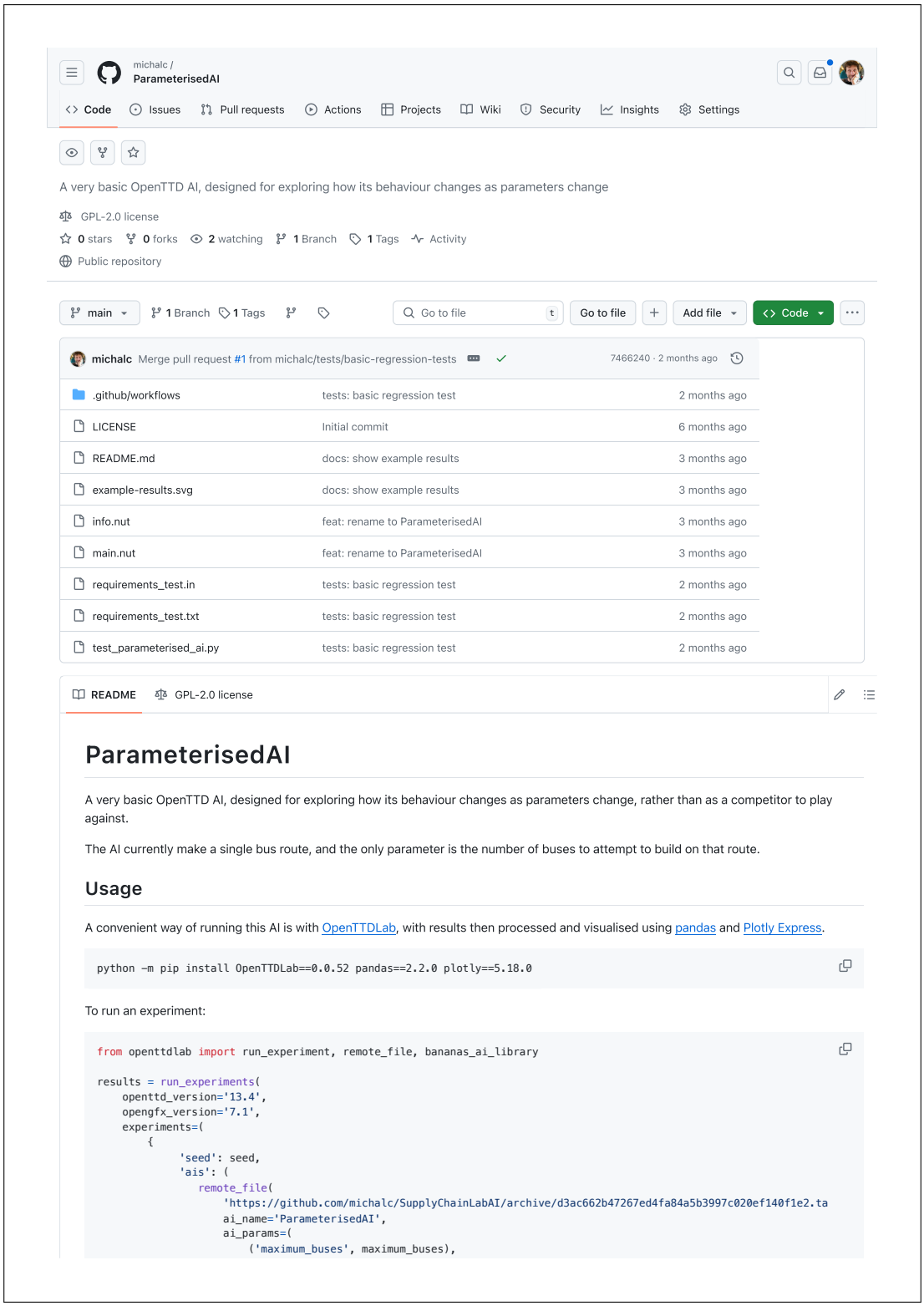
[9]:
```
df_company_value_quartiles_mountainous = df_company_value_quartiles.
 ↪xs('Mountainous', level='terrain_type', axis=1)
df_company_value_quartiles_mountainous.columns = [' '.join(col).strip() for col␣
 ↪in df_company_value_quartiles_mountainous.columns.values]
px.line(df_company_value_quartiles_mountainous)
```



Appendix E

Simulating a network: ParameterisedAI documentation

Following is the documentation of ParameterisedAI, the AI created for the experiments of Chapter 5. The copy is taken from commit 74662403.



michalc / ParameterisedAI

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

A very basic OpenTTD AI, designed for exploring how its behaviour changes as parameters change

GPL-2.0 license

0 stars 0 forks 2 watching 1 Branch 1 Tags Activity

Public repository

main 1 Branch 1 Tags Go to file t Go to file Add file Code

michalc Merge pull request #1 from michalc/tests/basic-regression-tests 7466240 · 2 months ago

.github/workflows	tests: basic regression test	2 months ago
LICENSE	Initial commit	6 months ago
README.md	docs: show example results	3 months ago
example-results.svg	docs: show example results	3 months ago
info.nut	feat: rename to ParameterisedAI	3 months ago
main.nut	feat: rename to ParameterisedAI	3 months ago
requirements_test.in	tests: basic regression test	2 months ago
requirements_test.txt	tests: basic regression test	2 months ago
test_parameterised_ai.py	tests: basic regression test	2 months ago

README GPL-2.0 license

ParameterisedAI

A very basic OpenTTD AI, designed for exploring how its behaviour changes as parameters change, rather than as a competitor to play against.

The AI currently make a single bus route, and the only parameter is the number of buses to attempt to build on that route.

Usage

A convenient way of running this AI is with OpenTTDLab, with results then processed and visualised using pandas and Plotly Express.

```
python -m pip install OpenTTDLab==0.0.52 pandas==2.2.0 plotly==5.18.0
```

To run an experiment:

```
from openttdlab import run_experiment, remote_file, bananas_ai_library

results = run_experiments(
    openttd_version='13.4',
    opengfx_version='7.1',
    experiments=(
        {
             'seed': seed,
             'ais': (
                remote_file(
                    'https://github.com/michalc/SupplyChainLabAI/archive/d3ac662b47267ed4fa84a5b3997c020ef140f1e2.ta
                    ai_name='ParameterisedAI',
                    ai_params=(
                        ('maximum_buses', maximum_buses),
```

```
                    ),
                ),
            ),
            # Increase to run for longer
            'days': 365 * 4 + 1,
        }
        for maximum_buses in [1, 2, 4, 8, 16]
        for seed in range(0, 10)
    ),
    ai_libraries=(
        bananas_ai_library('5046524f', 'Pathfinder.Road'),
        bananas_ai_library('4752412a', 'Graph.AyStar'),
        bananas_ai_library('51554248', 'Queue.BinaryHeap'),
    ),
)
```

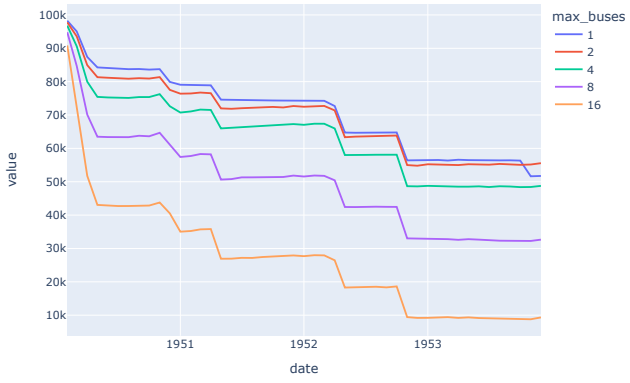
To then extract results:

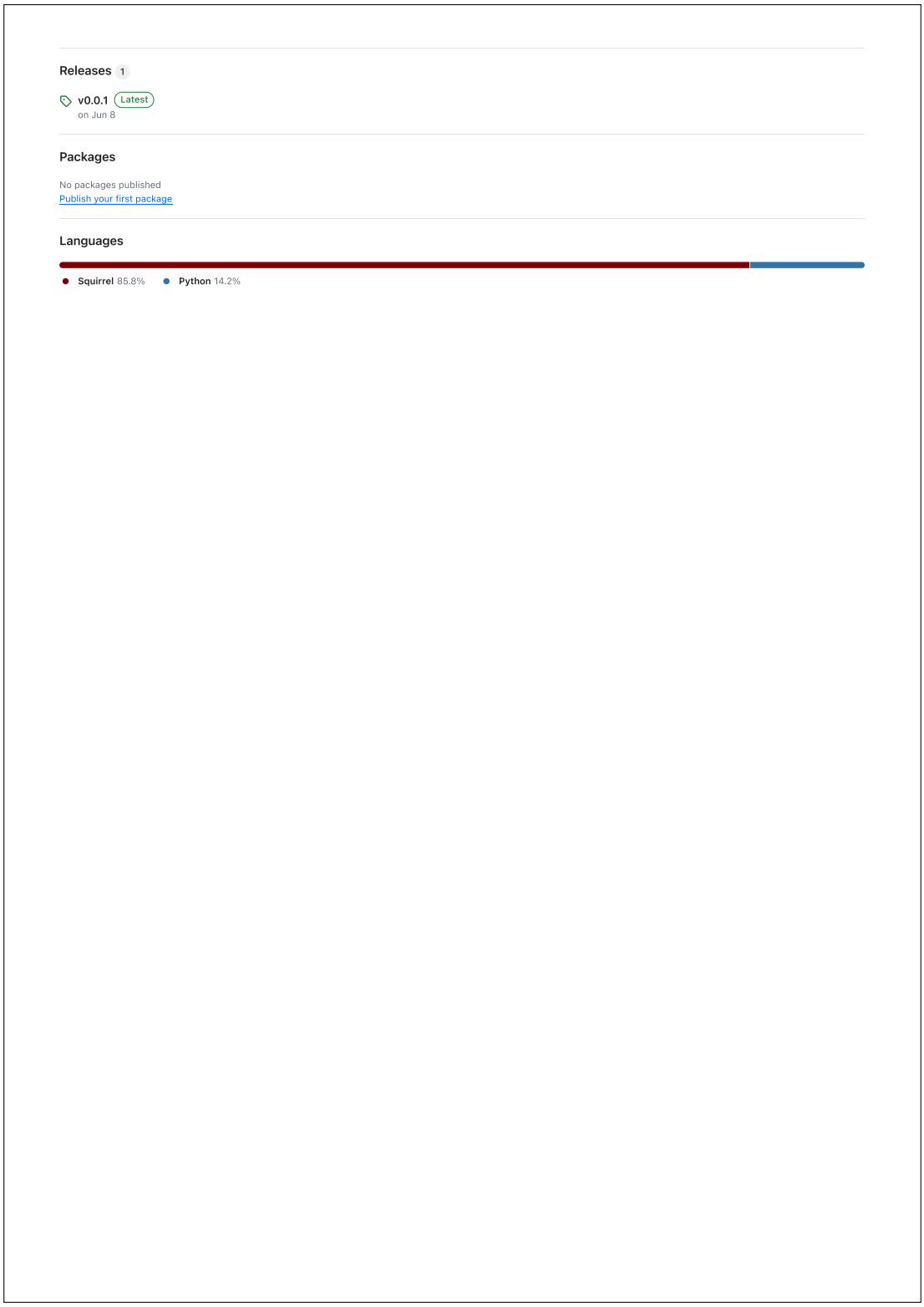
```
import pandas as pd

df = pd.DataFrame(
    {
        # It's slightly awkward right now to get at the original AI params
        'max_buses': row['experiment']['ais'][0][1][0][1],
        'seed': row['experiment']['seed'],
        'date': row['date'],
        'money': row['chunks']['PLYR']['0']['money'],
    }
    for row in results
)
df = df.pivot(index='date', columns=('seed', 'max_buses'), values='money')
df = df.T.groupby(level=1).mean().T
```

And then to plot them:

```
import plotly.express as px

fig = px.line(df)
fig.show()
```

To show:




Releases 1
v0.0.1 Latest
on Jun 8
Packages
No packages published
Publish your first package
Languages
Squirrel 85.8%
Python 14.2%

Appendix F

Simulating a network: ParameterisedAI Squirrel 'info.nut'

Following is the Squirrel code of one of the two files that comprised the AI written to generate the results of Chapter 5, the 'info.nut' file that contains the metadata of the AI. It is also available at https://github.com/michalc/ParameterisedAI/blob/74662403e0764329112dc78e5b279d7f1b5fd510/info.nut.

```
class ParameterisedAIInfo extends AIInfo {
  function GetAuthor()      { return "Michal Charemza"; }
  function GetName()        { return "ParameterisedAI"; }
  function GetDescription() { return "An AI to investigate supply
   chains"; }
  function GetVersion()     { return 1; }
  function GetDate()        { return "2024-02-11"; }
  function CreateInstance() { return "ParameterisedAI"; }
  function GetShortName()   { return "SCLB"; }
  function GetAPIVersion()  { return "13"; }

  function GetSettings() {
    AddSetting({
      name = "maximum_buses",
      description = "Maximum number of buses",
      min_value = 0, max_value = 2147483647,
      easy_value = 1, medium_value = 1, hard_value = 1,
      custom_value = 1,
      flags = 0
    });
  }
```

```
21 }
22 RegisterAI(ParameterisedAIInfo());
```

Appendix G

Simulating a network: ParameterisedAI Squirrel ‘main.nut’

Following is the Squirrel code of one of the two files that comprised the AI written to generate the results of Chapter 5, the ‘main.nut‘ file that contains the core algorithm of the AI. It is also available at https://github.com/michalc/ParameterisedAI/blob/74662403e0764329112dc78e5b279d7f1b5fd510/main.nut.

```
1  import("pathfinder.road", "RoadPathFinder", 4);
2
3  class ParameterisedAI extends AIController
4  {
5  }
6
7  function ParameterisedAI::Start()
8  {
9    local setName = function() {
10       local i = 1
11       local name = "ParameterisedAI"
12       while (!AICompany.SetName(name)) {
13         name = "ParameterisedAI #" + ++i
14       }
15
16       return name;
17   }
18
19   local findTownsToConnect = function()
20   {
21     // Connect the two towns with the highest population
```

```
22    local townlist = AITownList();
23    townlist.Valuate(AITown.GetPopulation);
24    townlist.Sort(AIList.SORT_BY_VALUE, false);
25    local townid_a = townlist.Begin();
26    local townid_b = townlist.Next();
27
28    return [townid_a, townid_b];
29  }
30
31  local findPathToConnect = function(start, end)
32  {
33    /* Tell OpenTTD we want to build normal road (no tram tracks).
     */
34    AIRoad.SetCurrentRoadType(AIRoad.ROADTYPE_ROAD);
35
36    /* Create an instance of the pathfinder. */
37    local pathfinder = RoadPathFinder();
38
39    /* Set the cost for making a turn extreme high. */
40    pathfinder.cost.turn = 5000;
41    pathfinder.InitializePath([start], [end]);
42
43    local path = false;
44    while (path == false) {
45      AILog.Info("Finding path... ")
46      path = pathfinder.FindPath(100);
47      this.Sleep(1);
48    }
49
50    return path
51  }
52
53  local getTiles = function(path)
54  {
55    while (path != null) {
56      yield path.GetTile();
57      path = path.GetParent();
58    }
59  }
60
61  local getTilePairs = function(path)
62  {
```

```
63      while (path != null) {
64        local par = path.GetParent();
65        if (par != null) {
66          yield [path.GetTile(), par.GetTile()];
67        }
68        path = par;
69      }
70    }
71
72    local buildRoad = function(tilePairs) {
73      local tileList = AITileList();
74      foreach (tilePair in tilePairs) {
75        local current = tilePair[0];
76        local next = tilePair[1];
77        tileList.AddTile(current);
78        if (AIMap.DistanceManhattan(current, next) == 1 ) {
79          if (!AIRoad.BuildRoad(current, next)) {
80            /* An error occured while building a piece of road. TODO:
      handle it.
81             * Note that is can also be the case that the road was
      already build. */
82          }
83        } else {
84          /* Build a bridge or tunnel. */
85          if (!AIBridge.IsBridgeTile(current) && !AITunnel.
      IsTunnelTile(current)) {
86            /* If it was a road tile, demolish it first. Do this to
      work around expended roadbits. */
87            if (AIRoad.IsRoadTile(current)) AITile.DemolishTile(
      current);
88            if (AITunnel.GetOtherTunnelEnd(current) == next) {
89              if (!AITunnel.BuildTunnel(AIVehicle.VT_ROAD, current)) {
90                /* An error occured while building a tunnel. TODO:
      handle it. */
91              }
92            } else {
93              local bridge_list = AIBridgeList_Length(AIMap.
      DistanceManhattan(current, next) + 1);
94              bridge_list.Valuate(AIBridge.GetMaxSpeed);
95              bridge_list.Sort(AIList.SORT_BY_VALUE, false);
96              if (!AIBridge.BuildBridge(AIVehicle.VT_ROAD, bridge_list
      .Begin(), current, next)) {
```

```
97              /* An error occured while building a bridge. TODO:
    handle it. */
98            }
99          }
100       }
101     }
102   }
103 }
104
105 local buildAlong = function(tiles, buildFunc) {
106   foreach (pathTileIndex in tiles) {
107     local adjacentTiles = AITileList();
108     adjacentTiles.AddTile(pathTileIndex - AIMap.GetTileIndex(1,0))
    ;
109     adjacentTiles.AddTile(pathTileIndex - AIMap.GetTileIndex(0,1))
    ;
110     adjacentTiles.AddTile(pathTileIndex - AIMap.GetTileIndex(-1,0)
    );
111     adjacentTiles.AddTile(pathTileIndex - AIMap.GetTileIndex(0,-1)
    );
112
113     foreach (index, value in adjacentTiles) {
114       local built = buildFunc(index, pathTileIndex);
115       if (built) {
116         if (!AIRoad.BuildRoad(index, pathTileIndex)) {
117         }
118         return index;
119       }
120     }
121   }
122 }
123
124 local reverse = function(gen) {
125   local reversed = [];
126   foreach (item in gen) {
127     reversed.append(item);
128   }
129   reversed.reverse();
130   return reversed;
131 }
132
133 local getPassengerCargo = function() {
```

```
134     foreach (cargo, dummy in AICargoList()) {
135       if (AICargo.GetTownEffect(cargo) == AICargo.TE_PASSENGERS) {
136         return cargo;
137       }
138     }
139   }
140
141   local chooseBusEngine = function(passengerCargo) {
142     local roadEngines = AIEngineList(AIVehicle.VT_ROAD);
143     roadEngines.Valuate(AIEngine.CanRefitCargo, passengerCargo);
144     roadEngines.KeepValue(1);
145     roadEngines.Valuate(AIEngine.IsBuildable);
146     roadEngines.KeepValue(1);
147     roadEngines.Valuate(AIEngine.GetMaxSpeed);
148     roadEngines.Sort(AIList.SORT_BY_VALUE, false);
149     return roadEngines.Begin();
150   }
151
152   local name = setName();
153   AILog.Info("Chosen company name: " + name);
154
155   local townsToConnect = findTownsToConnect();
156   AILog.Info("Going to connect " + AITown.GetName(townsToConnect[0])
        + " to " + AITown.GetName(townsToConnect[1]));
157
158   local path = findPathToConnect(AITown.GetLocation(townsToConnect
       [0]), AITown.GetLocation(townsToConnect[1]))
159
160   if (path == null) {
161     AILog.Error("No path found");
162   }
163
164   /* If a path was found, build a road over it. */
165   buildRoad(getTilePairs(path));
166   AILog.Info("Done");
167
168   // Build station as close as possible to the start of the path
169   local buildRoadStation = function(tile, front) {
170     return AIRoad.BuildRoadStation(tile, front, AIRoad.
       ROADVEHTYPE_BUS, AIStation.STATION_NEW)
171   }
172   local startStationTile = buildAlong(getTiles(path),
```

```
    buildRoadStation);
173
174   // Build station as close as possible to the end of the path
175   local reversedTiles = reverse(getTiles(path));
176   local endStationTile = buildAlong(reversedTiles, buildRoadStation)
       ;
177
178   // Build depot as close as possible to the start of the path
179   local buildRoadDepot = function(tile, front) {
180     return AIRoad.BuildRoadDepot(tile, front);
181   }
182   local depotTile = buildAlong(getTiles(path), buildRoadDepot)
183
184   // Find bus type to build
185   local busEngine = chooseBusEngine(getPassengerCargo());
186   AILog.Info("Have chosen bus " + AIEngine.GetName(busEngine));
187
188   // Set the bus going between the stations
189   local maximum_buses = AIController.GetSetting("maximum_buses");
190   for (local i = 0 ; i < maximum_buses; i++) {
191     local busVehicle = AIVehicle.BuildVehicle(depotTile, busEngine);
192     AIOrder.AppendOrder(busVehicle, startStationTile, AIOrder.
      OF_NONE);
193     AIOrder.AppendOrder(busVehicle, endStationTile, AIOrder.OF_NONE)
      ;
194     AIVehicle.StartStopVehicle(busVehicle);
195     AILog.Info("Have started bus");
196   }
197
198   // Inifinite loop so the AI doesn't register as exited
199   while (true) {
200     AILog.Info("Sleeping");
201     this.Sleep(50)
202   }
203 }
204
205 function ParameterisedAI::Save()
206 {
207  return {};
208 }
209
210 function ParameterisedAI::Load(version, data)
```

```
211 {
212 }
```

Appendix H

Simulating a network: ParameterisedAI Python regression test

Following is the Python regression test of the AI written to generate the results of Chapter 5. It is also available at https://github.com/michalc/ParameterisedAI/blob/74662403e0764329112dc78e5b279d7f1b5fd510/test_parameterised_ai.py.

```
from datetime import date
from openttdlab import bananas_ai_library, run_experiments,
    local_folder, remote_file


def test_regression():
    results = run_experiments(
        experiments=(
            {
                'seed': seed,
                'ais': (
                    local_folder('.', 'ParameterisedAI'),
                ),
                'days': 366 * 1 + 1,
            }
            for seed in range(0, 10)
        ),
        ai_libraries=(
```

```
            bananas_ai_library('5046524f', 'Pathfinder.Road'),
        ),
        openttd_version='13.4',
        opengfx_version='7.1',
        result_processor=lambda result_row: [{
            'seed': result_row['experiment']['seed'],
            'date': result_row['date'],
            'openttd_version': result_row['openttd_version'],
            'opengfx_version': result_row['opengfx_version'],
            'name': result_row['chunks']['PLYR']['0']['name'],
            'money': result_row['chunks']['PLYR']['0']['money'],
        }],
    )

    assert results[-1] == {
        'seed': 9,
        'date': date(1951, 1, 1),
        'openttd_version': '13.4',
        'opengfx_version': '7.1',
        'name': 'ParameterisedAI',
        'money': 73886,
    }
```

Appendix I

Simulating a network: Python code to run experiments

Following is a copy of Python notebook used to run the network simulation experiments of Chapter 5. It is available in source code form at https://github.com/michalc/OpenTTDLab-MSc-Dissertation/blob/8d432038278685edb0988457c9d2a971a668ac64/notebooks/02_own_parameterised_ai_01_run_experiments.ipynb.

02_own_parameterised_ai_01_run_experiments

August 3, 2024

1 Own parameterised AI - run experiments¶

Runs a series of experiments of OpenTTDLab, running https://github.com/michalc/ParameterisedAI for a varying number of buses.

Results are saved to 02_own_parameterised_ai_01_raw.csv.

[ ]:
```
!python -m pip install OpenTTDLab==0.0.72 pandas==2.2.0
```

[4]:
```
from openttdlab import run_experiments, remote_file, bananas_ai_library

def process_result(result):
    return [{
        # It's slightly awkward right now to get at the original AI params
        'max_buses': result['experiment']['ais'][0][1][0][1],
        'seed': result['experiment']['seed'],
        'date': result['date'],
        'money': result['chunks']['PLYR']['0']['money'],
    }]

results = run_experiments(
    openttd_version='13.4',
    opengfx_version='7.1',
    experiments=(
        {
             'seed': seed,
             'ais': (
                remote_file(
                    'https://github.com/michalc/ParameterisedAI/archive/
↪d3ac662b47267ed4fa84a5b3997c020ef140f1e2.tar.gz',
                    ai_name='ParameterisedAI',
                    ai_params=(
                        ('maximum_buses', maximum_buses),
                    ),
                ),
             ),
            'days': 366 * 50 + 1,
        }
```

```
        for maximum_buses in [1, 2, 4, 8, 16]
        for seed in range(0, 50)
    ),
    ai_libraries=(
        bananas_ai_library('5046524f', 'Pathfinder.Road',␣
↪md5='999de61cd3a10680b4ff91547299dc53'),
        bananas_ai_library('4752412a', 'Graph.AyStar',␣
↪md5='f385497c3c922bfd9f61e1bc33b3a4dc'),
        bananas_ai_library('51554248', 'Queue.BinaryHeap',␣
↪md5='8ce55e1397e9c51f8032c3a8a29e9cf5'),
    ),
    result_processor=process_result,
)
```

```
Output()
```

[5]:
```
import pandas as pd

df = pd.DataFrame(results)
df.to_csv('02_own_parameterised_ai_results_01_raw.csv', index=False)
```

Appendix J

Simulating a network: Python code to analyse results

Following is a copy of Python notebook used to analyse results of the network simulation experiments of Chapter 5. It is available in source code form at https://github.com/michalc/OpenTTDLab-MSc-Dissertation/blob/8d432038278685edb0988457c9d2a971a668ac64/notebooks/notebooks/02_own_parameterised_ai_02_analyse_results.ipynb.

02_own_parameterised_ai_02_analyse_results

August 3, 2024

1 Own parameterised AI - analyse results

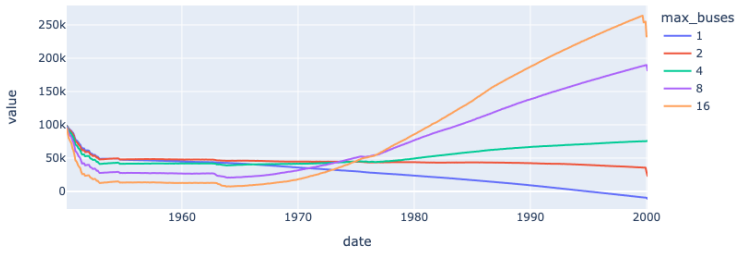
Extracts and visualises results from 02_own_parameterised_ai_results_01_raw.csv.

[1]:
```
from datetime import datetime
import pandas as pd
import plotly.express as px
import numpy as np
```

[2]:
```
df = pd.read_csv('02_own_parameterised_ai_results_01_raw.csv', parse_dates=[2])
```

[3]:
```
df = pd.read_csv('02_own_parameterised_ai_results_01_raw.csv', parse_dates=[2])
df_to_group = df.groupby(['max_buses', 'date'])
df_mean_standard_deviation = pd.concat([
    df_to_group.agg('mean')['money'].rename('mean'),
    df_to_group.agg('std')['money'].rename('standard_deviation'),
], axis=1).stack().unstack(level=2).unstack(level=0)
df_mean_standard_deviation.columns.set_names(['statistic', 'max_buses'],␣
↪inplace=True)
df_mean_standard_deviation.
↪to_csv('02_own_parameterised_ai_results_02_money_mean_standard_deviation.
↪csv')
```

[4]:
```
px.line(df_mean_standard_deviation["mean"])
```



Appendix K

Scaling:
Python code to run experiments

Following is a copy of Python notebook used to run the scaling experiments of Chapter 6. It is available in source code form at https://github.com/michalc/OpenTTDLab-MSc-Dissertation/blob/8d432038278685edb0988457c9d2a971a668ac64/notebooks/03_scaling_01_run_experiment.ipynb.

01_scaling_01_run_experiment

June 25, 2024

1 Scaling - running experiment

Runs a series of experiments of OpenTTDLab, varying the number of workers used, what data is retrieved from the worker processes, recording how long each takes.

Results are saved to 01_scaling_raw.csv.

[ ]:
```
!python -m pip install OpenTTDLab==0.0.72 pandas==2.2.0
```

[ ]:
```
import time
from openttdlab import run_experiments, bananas_ai

def all_results(row):
    return [row]

def minimal_results(row):
    # It is expected early on the game that no company value is recorded
    def get_company_value(player):
        try:
            return player['old_economy'][0]['company_value']
        except (KeyError, IndexError):
            return 0

    return [{
        'date': row['date'],
        'company_value': get_company_value(row['chunks']['PLYR']['0']),
        'error': row['error'],
    }]

def run_experiment_timed(max_workers, result_processor):
    start = time.monotonic()
    results = run_experiments(
        openttd_version='13.4',
        opengfx_version='7.1',
        experiments=(
           {
             'seed': seed,
             'ais': (
```

```
                # This is trAIns 2.1. The md5 enforces a specific version
                # of trAIns and was retrieved by a previous call to
                # openttdlab.dowload_from_bananas
                bananas_ai(
                    '54524149', 'trAIns',
                    md5='c4c069dc797674e545411b59867ad0c2',
                ),
            ),
            'days': 366 * 4 + 1,
        }
        for seed in range(0, 50)
    ),
    result_processor=result_processor,
    max_workers=max_workers,
)
assert all(not row['error'] for row in results)
end = time.monotonic()
return (end - start)

results = [
    {
        'max_workers': max_workers,
        'results': 'all',
        'wallclock_time': run_experiment_timed(max_workers, all_results),
    }
    for max_workers in range(1, 9)
] + [
        {
        'max_workers': max_workers,
        'results': 'minimal',
        'wallclock_time': run_experiment_timed(max_workers, minimal_results),
    }
    for max_workers in range(1, 9)
]
```

[ ]:
```
import pandas as pd

df = pd.DataFrame(results)
df.to_csv('01_scaling_results_01_raw.csv', index=False)
```

Appendix L

Scaling:
Python code to analyse results

Following is a copy of Python notebook used to analyse results of the scaling experiments of Chapter 6. It is available in source code form at https://github.com/michalc/OpenTTDLab-MSc-Dissertation/blob/8d432038278685edb0988457c9d2a971a668ac64/notebooks/03_scaling_02_analyse_results.ipynb

01_scaling_02_analyse_results

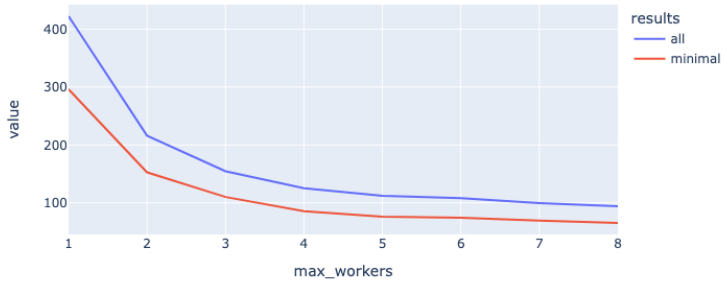
June 25, 2024

1 Scaling - analyse results

Extracts and visualises results 01_scaling_raw.csv.

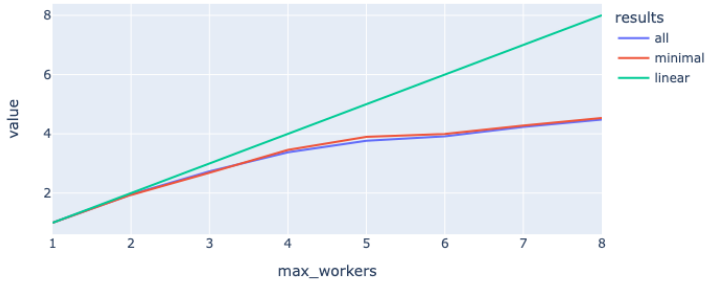
[1]:
```
import pandas as pd
import plotly.express as px
```

[2]:
```
# Plot raw runtime
df = pd.read_csv('01_scaling_results_01_raw.csv')
df_wallclock_times = df.pivot(
    index='max_workers',
    columns='results',
    values='wallclock_time',
)
px.line(df_wallclock_times)
```



[3]:
```
# Plot speedup
df_speedups = df_wallclock_times.loc[1] / df_wallclock_times
df_speedups['linear'] = df_speedups.index
```

```
px.line(df_speedups)
```



[4]:
```
# Save runtime and speedups
df_wallclock_times.to_csv('01_scaling_results_02_wallclock_times.csv')
df_speedups.to_csv('01_scaling_results_03_speedups.csv')
```